

CMSC 430, Practice Problems 1 (Solutions)

1. Describe the languages denoted by the following regular expressions:

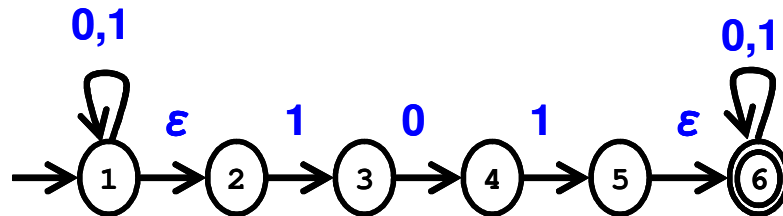
- a. $0(01)^*0$
strings beginning and ending in 0
- b. $((\epsilon | 0)1)^*$
all strings of 0 and 1
- c. $(01)^*0(01)(01)$
strings with 0 as third digit from right

2. Regular expressions and languages

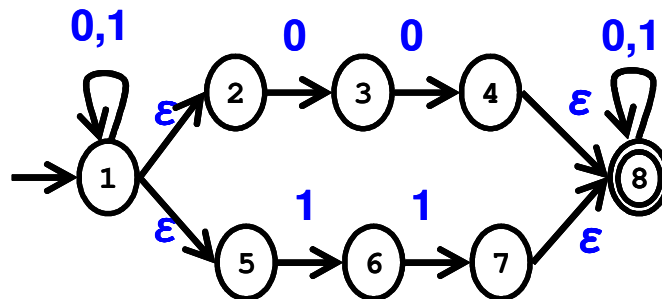
- a. Give a regular expression for all binary numbers including the substring “101”.
 $(01)^*101(01)^*$
- b. Give a regular expression for all binary numbers with an even number of 1’s.
 $0^*(10^*10^*)^*$
- c. Give a regular expression for all binary numbers that don’t include “000”.
 $(01 | 001 | 1)^*(0 | 00 | \epsilon)$

3. Finite automata

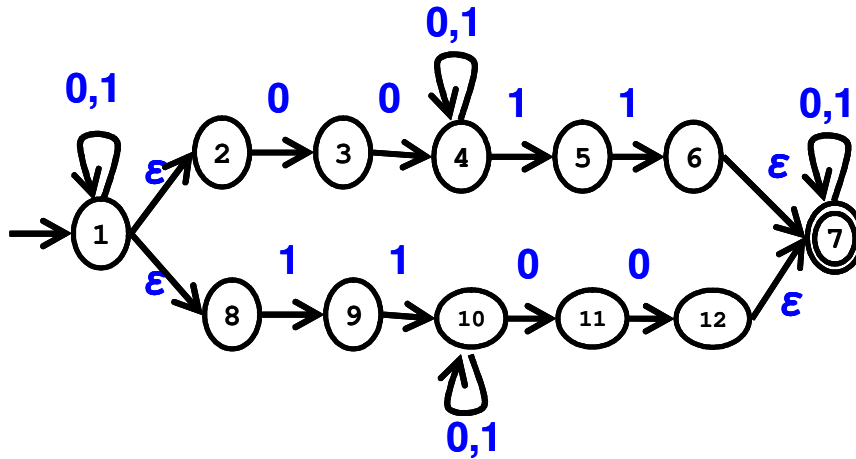
- a. Give a NFA that only accepts binary numbers including the substring “101”.



- b. Give a NFA that only accepts binary numbers that include either “00” or “11”.



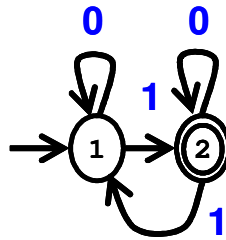
c. Give a NFA that only accepts binary numbers that include both “00” and “11”.



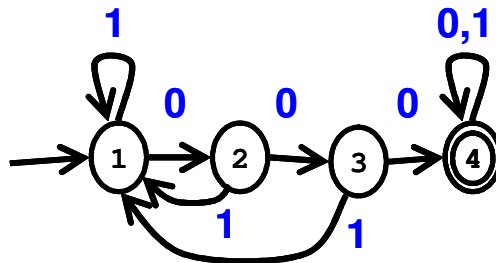
d. Compute the ϵ -closure of the start state for each of the NFA above.

- For NFA in (a) ϵ -closure(1) = {1,2}
- For NFA in (b) ϵ -closure(1) = {1,2,5}
- For NFA in (c) ϵ -closure(1) = {1,2,8}

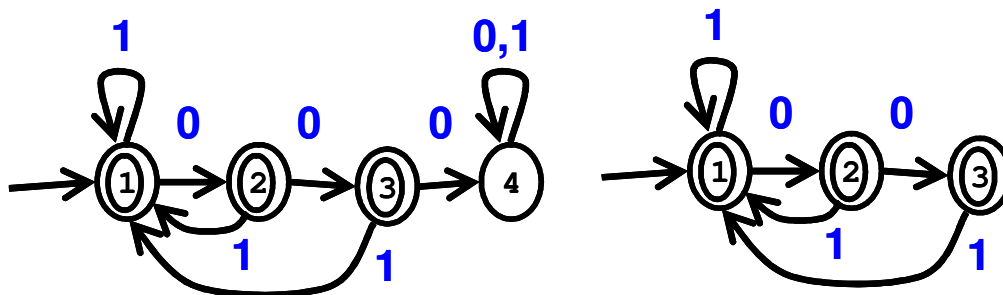
e. Give a DFA that only accepts binary numbers with an odd number of 1's.



f. Give a DFA that only accepts binary numbers that include “000”.



g. Give a DFA that only accepts binary numbers that don't include “000”.

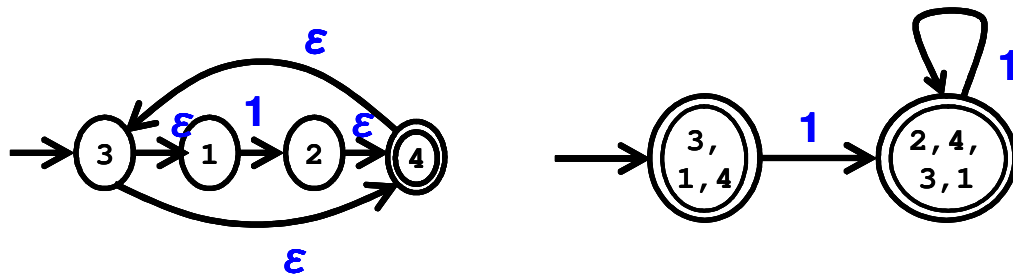


4. Reducing REs to DFAs

For each regular expression: 1^* , $(0101)^*0$

- Reduce the RE to an NFA using the algorithm described in class.
- Reduce the resulting NFA to a DFA using the subset algorithm.
- Show whether the DFA accepts / rejects the strings “1”, “11”, “101”
- Minimize the resulting DFA using Hopcroft reduction

$1^* \rightarrow \text{NFA} \rightarrow \text{DFA}$



Accept / reject

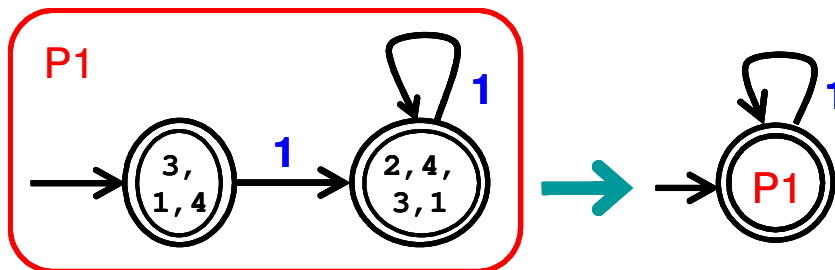
- “1” $\{3,1,4\} \rightarrow \{2,4,3,1\}$ accept
- “11” $\{3,1,4\} \rightarrow \{2,4,3,1\} \rightarrow \{2,4,3,1\}$ accept
- “101” $\{3,1,4\} \rightarrow \{2,4,3,1\} \rightarrow \text{reject}$

Minimized DFA

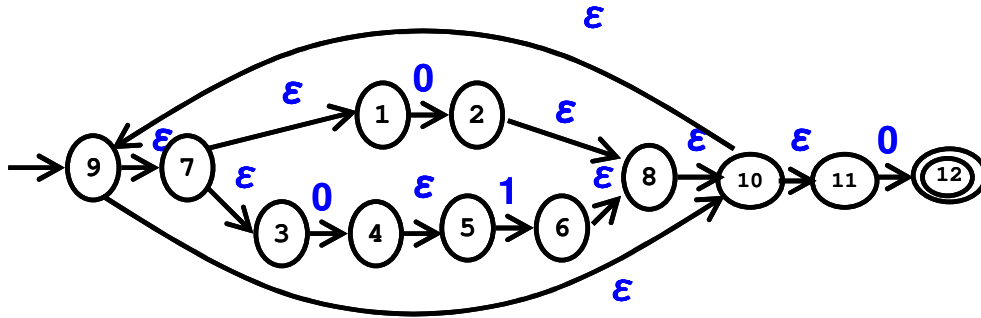
Initial partitions: $\text{accept} = \{ \{3,1,4\}, \{2,4,3,1\} \} = P1$,
 $\text{nonfinal} = \emptyset$

- $\text{move}(\{3,1,4\}, 1) \rightarrow P1$
- $\text{move}(\{2,4,3,1\}, 1) \rightarrow P1$

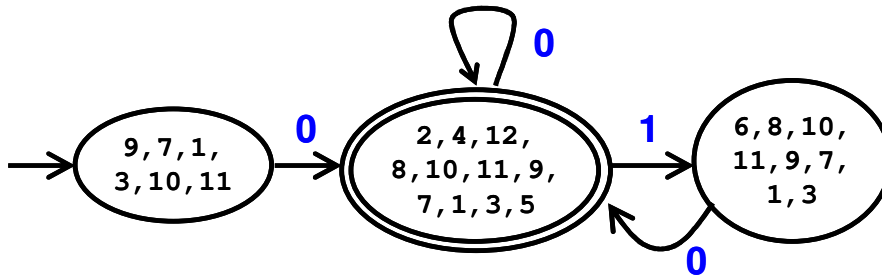
No need to split P1, minimization done. After cleanup, minimal DFA is



$(0101)^*0 \rightarrow \text{NFA}$



$(0101)^*0 \rightarrow \text{NFA} \rightarrow \text{DFA}$



Accept / reject

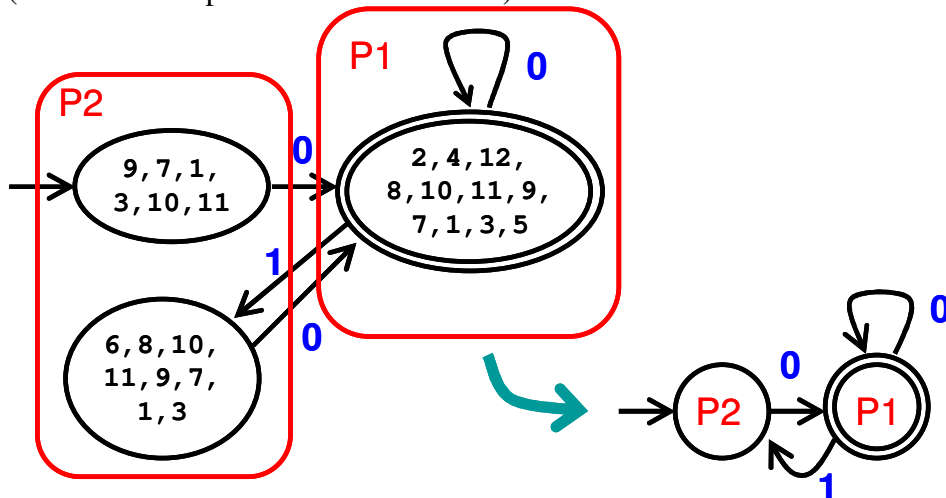
- “1” {9,7,1,3,10,11} → reject
- “11” {9,7,1,3,10,11} → reject
- “101” {9,7,1,3,10,11} → reject

Minimized DFA

Initial partitions: accept = { {2,4...} } = P1,
nonfinal = { {9,7...}, {6,8...} } = P2

- move({9,7...}, 0) → P1
- move({6,8...}, 0) → P1
- move({9,7...}, 1) → reject
- move({6,8...}, 1) → reject

No need to split P2, minimization done. After cleanup, minimal DFA (different from previous minimal DFA) is



5. Describing grammars

- a. Describe the language accepted by the following grammar:
 $S \rightarrow abS \mid a$
 $(ab)^*a$
- b. Describe the language accepted by the following grammar:
 $S \rightarrow aSb \mid \epsilon$
 $a^n b^n, n \geq 0$
- c. Describe the language accepted by the following grammar:
 $S \rightarrow bSb \mid A \quad A \rightarrow aA \mid \epsilon$
 $b^n a^* b^n, n \geq 0$
- d. Describe the language accepted by the following grammar:
 $S \rightarrow AS \mid B \quad A \rightarrow aAc \mid Aa \mid \epsilon \quad B \rightarrow bBb \mid \epsilon$
Strings of a & c with same or fewer c's than a's and no prefix has more c's than a's, followed by an even number of b's
- e. Describe the language accepted by the following grammar:
 $S \rightarrow S \text{ and } S \mid S \text{ or } S \mid (S) \mid \text{true} \mid \text{false}$
Boolean expressions of true & false separated by and & or, with some expressions enclosed in parentheses
- f. Which of the previous grammars are left recursive?
5d, 5e
- g. Which of the previous grammars are right recursive?
5a, 5c, 5d, 5e
- h. Which of the previous grammars are ambiguous? Provide proof.
Examples of multiple left-most derivations for the same string
5d: $S \Rightarrow AS \Rightarrow AaS \Rightarrow aS \Rightarrow aB \Rightarrow a$
 $S \Rightarrow AS \Rightarrow S \Rightarrow AS \Rightarrow AaS \Rightarrow aS \Rightarrow aB \Rightarrow a$
5e: $S \Rightarrow S \text{ and } S \Rightarrow S \text{ and } S \text{ and } S \Rightarrow \text{true and } S \text{ and } S$
 $\Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$
 $S \Rightarrow S \text{ and } S \Rightarrow \text{true and } S \Rightarrow \text{true and } S \text{ and } S$
 $\Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$

6. Creating grammars

- a. Write a grammar for $a^x b^y$, where $x = y$
 $S \rightarrow aSb \mid \epsilon$
- b. Write a grammar for $a^x b^y$, where $x > y$
 $S \rightarrow aL \quad L \rightarrow aL \mid aLb \mid \epsilon$
- c. Write a grammar for $a^x b^y$, where $x = 2y$
 $S \rightarrow aaSb \mid \epsilon$
- d. Write a grammar for all strings of a and b that are palindromes.
 $S \rightarrow aSa \mid bSb \mid L \quad L \rightarrow a \mid b \mid \epsilon$
- e. Write a grammar for all strings of a and b that include the substring baa .
 $S \rightarrow LbaaL \quad L \rightarrow aL \mid bL \mid \epsilon \quad // L = \text{any}$
- f. Write a grammar for all strings of a and b with an odd number of a 's and b 's.
 $S \rightarrow EaEbE \mid EbEaE \quad E \rightarrow EaEaE \mid EbEbE \mid \epsilon \mid SS \quad // E = \text{even \#}$

7. Derivations and parse trees

For the following grammar: $S \rightarrow S \text{ and } S \mid \text{true}$

a. List 4 derivations for the string “true and true and true”.

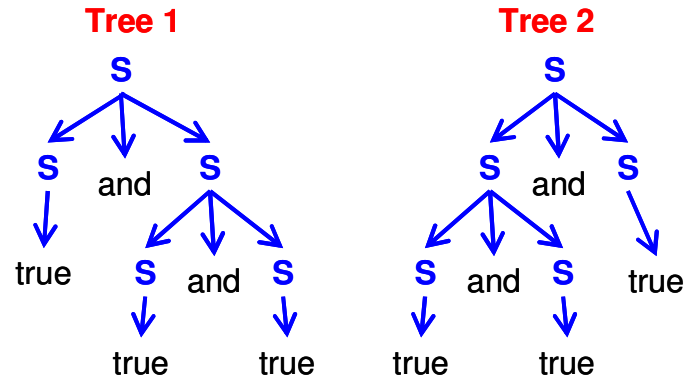
- i. $S \Rightarrow \underline{S} \text{ and } S \Rightarrow \underline{S} \text{ and } S \text{ and } S \Rightarrow \text{true and } \underline{S} \text{ and } S \Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$
- ii. $S \Rightarrow \underline{S} \text{ and } S \Rightarrow \text{true and } S \Rightarrow \text{true and } \underline{S} \text{ and } S \Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$
- iii. $S \Rightarrow S \text{ and } \underline{S} \Rightarrow S \text{ and true} \Rightarrow S \text{ and } \underline{S} \text{ and true} \Rightarrow S \text{ and true and true} \Rightarrow \text{true and true and true}$
- iv. $S \Rightarrow S \text{ and } \underline{S} \Rightarrow S \text{ and } S \text{ and } \underline{S} \Rightarrow S \text{ and } \underline{S} \text{ and true} \Rightarrow S \text{ and true and true} \Rightarrow \text{true and true and true}$
- v. $S \Rightarrow \underline{S} \text{ and } S \Rightarrow \underline{S} \text{ and } S \text{ and } S \Rightarrow \text{true and } S \text{ and } \underline{S} \Rightarrow \text{true and S and true} \Rightarrow \text{true and true and true}$
- vi. $S \Rightarrow \underline{S} \text{ and } S \Rightarrow S \text{ and } \underline{S} \text{ and } S \Rightarrow \underline{S} \text{ and true and } S \Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$
- vii. $S \Rightarrow \underline{S} \text{ and } S \Rightarrow S \text{ and } \underline{S} \text{ and } S \Rightarrow S \text{ and true and } \underline{S} \Rightarrow S \text{ and true and true} \Rightarrow \text{true and true and true}$
- viii. $S \Rightarrow \underline{S} \text{ and } S \Rightarrow S \text{ and } S \text{ and } \underline{S} \Rightarrow \underline{S} \text{ and } S \text{ and true} \Rightarrow \text{true and S and true} \Rightarrow \text{true and true and true}$
- ix. $S \Rightarrow \underline{S} \text{ and } S \Rightarrow S \text{ and } S \text{ and } \underline{S} \Rightarrow S \text{ and } \underline{S} \text{ and true} \Rightarrow S \text{ and true and true} \Rightarrow \text{true and true and true}$
- x. $S \Rightarrow \underline{S} \text{ and } S \Rightarrow \text{true and } S \Rightarrow \text{true and } S \text{ and } \underline{S} \Rightarrow \text{true and } S \text{ and true} \Rightarrow \text{true and true and true}$
- xi. $S \Rightarrow S \text{ and } \underline{S} \Rightarrow S \text{ and true} \Rightarrow \underline{S} \text{ and } S \text{ and true} \Rightarrow \text{true and } S \text{ and true} \Rightarrow \text{true and true and true}$
- xii. $S \Rightarrow S \text{ and } \underline{S} \Rightarrow \underline{S} \text{ and } S \text{ and } S \Rightarrow \text{true and } \underline{S} \text{ and } S \Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$
- xiii. $S \Rightarrow S \text{ and } \underline{S} \Rightarrow \underline{S} \text{ and } S \text{ and } S \Rightarrow \text{true and } S \text{ and } \underline{S} \Rightarrow \text{true and S and true} \Rightarrow \text{true and true and true}$
- xiv. $S \Rightarrow S \text{ and } \underline{S} \Rightarrow S \text{ and } \underline{S} \text{ and } S \Rightarrow \underline{S} \text{ and true and } S \Rightarrow \text{true and true and } S \Rightarrow \text{true and true and true}$
- xv. $S \Rightarrow S \text{ and } \underline{S} \Rightarrow S \text{ and } \underline{S} \text{ and } S \Rightarrow S \text{ and true and } \underline{S} \Rightarrow S \text{ and true and true} \Rightarrow \text{true and true and true}$
- xvi. $S \Rightarrow S \text{ and } \underline{S} \Rightarrow S \text{ and } S \text{ and } \underline{S} \Rightarrow \underline{S} \text{ and } S \text{ and true} \Rightarrow \text{true and } S \text{ and true} \Rightarrow \text{true and true and true}$

b. Label each derivation as left-most, right-most, or neither.

i and ii are left-most derivations, iii and iv are right-most derivations, remaining derivations are neither

c. List the parse tree for each derivation

Tree 1 = ii, iii, x, xi, Tree 2 = rest



8. Consider the following grammar $S \rightarrow aSbS \mid bSaS \mid \epsilon$
- Prove grammar is ambiguous by finding 2 left-most derivations for "abab"
 - $S \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab$
 - $S \Rightarrow aSbS \Rightarrow abSaSbS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab$
 - Prove grammar is ambiguous by finding 2 right-most derivations for "abab"
 - $S \Rightarrow aSbS \Rightarrow aSbaSbS \Rightarrow aSbaSb \Rightarrow aSbab \Rightarrow abab$
 - $S \Rightarrow aSbS \Rightarrow aSb \Rightarrow abSaSb \Rightarrow abSab \Rightarrow abab$
9. Consider the following grammar: $S \rightarrow S \text{ and } S \mid S \text{ or } S \mid (S) \mid \text{true} \mid \text{false}$
- Compute First sets for each production and nonterminal
 - $\text{FIRST}(\text{true}) = \{ \text{"true"} \}$
 - $\text{FIRST}(\text{false}) = \{ \text{"false"} \}$
 - $\text{FIRST}((S)) = \{ \text{"("} \}$
 - $\text{FIRST}(S \text{ and } S) = \text{FIRST}(S \text{ or } S) =$
 $\text{FIRST}(S) = \{ \text{"(", "true", "false"} \}$
 - Explain why the grammar cannot be parsed by a LL(1) parser
 First sets of productions intersect, grammar is left recursive

10. Consider the following grammar: $S \rightarrow abS \mid acS \mid c$

- a. Compute First sets for each production and nonterminal

FIRST(abS) = { a }

FIRST(acS) = { a }

FIRST(c) = { c }

FIRST(S) = { a, c }

- b. Show why the grammar cannot be parsed by a LL(1) parser.

First sets of productions overlap

FIRST(abS) \cap FIRST(acS) = { a } \cap { a } = { a } $\neq \emptyset$

- c. Rewrite the grammar so it can be parsed by a LL(1) parser.

$S \rightarrow aL \mid c$ $L \rightarrow bS \mid cS$

- d. Write a recursive descent parser for the rewritten grammar.

```
parse_S() {
    if (lookahead == "a") {
        match("a"); // S  $\rightarrow$  aL
        parse_L();
    }
    else if (lookahead == "c")
        match("c"); // S  $\rightarrow$  c
    }
    else error();
}

parse_L() {
    if (lookahead == "b") {
        match("b"); // L  $\rightarrow$  bS
        parse_S();
    }
    else if (lookahead == "c") {
        match("c"); // L  $\rightarrow$  cS
        parse_S();
    }
    else error();
}
```

11. Consider the following grammar: $S \rightarrow Sa \mid Sc \mid c$

a. Show why the grammar cannot be parsed by a predictive parser.

First sets of productions intersect, grammar is left recursive

b. Rewrite the grammar so it can be parsed by a predictive parser.

$S \rightarrow cL$ $L \rightarrow aL \mid cL \mid \epsilon$

c. Using FIRST and FOLLOW, build the LL(1) table for the new grammar

$\text{FIRST}^+(L \rightarrow \epsilon) = \text{FOLLOW}(L) = \{\$\}$

	a	c	\$
S		$S \rightarrow cL$	
L	$L \rightarrow aL$	$L \rightarrow cL$	$L \rightarrow \epsilon$

d. Using the LL(1) table, show how a table-driven parser parses the string “caca”

Stack	Input	Action
[\$, S]	caca\$	$S \rightarrow cL$, pop S + push L c
[\$, L, c]	caca\$	next input + pop c
[\$, L]	aca\$	$L \rightarrow aL$, pop L + push L a
[\$, L, a]	aca\$	next input + pop a
[\$, L]	ca\$	$L \rightarrow cL$, pop L + push L c
[\$, L, c]	ca\$	next input + pop c
[\$, L]	a\$	$L \rightarrow aL$, pop L + push L a
[\$, L, a]	a\$	next input + pop a
[\$, L]	\$	$L \rightarrow \epsilon$, pop L
[\$]	\$	accept

e. Write a recursive descent parser for your new grammar

```

parse_S() {
    if (lookahead == "c") {
        match("c"); // S → cL
        parse_L();
    }
    else error();
}
parse_L() {
    if (lookahead == "a") {
        match("a"); // L → aL
        parse_L();
    }
    else if (lookahead == "c") {
        match("c"); // L → cL
        parse_L();
    }
    else if (lookahead == "$")
        ; // L → ε                      since FOLLOW(L) = {$}
    else error();
}

```

12. Consider the following grammar

```

Goal      →  Expr
Expr      →  Term + Expr
           |  Term - Expr
           |  Term
Term      →  Factor * Term
           |  Factor / Term
           |  Factor
Factor    →  num
           |  id
    
```

- Apply left factoring to create a LL(1) grammar
- Compute FIRST and FOLLOW for the modified grammar

Left Factored Grammar		FIRST	FOLLOW
		Productions	Nonterminals
Goal	→ Expr	{ num, id }	Goal { num, id } { \$ }
Expr	→ Term Expr'	{ num, id }	Expr { num, id } { \$ }
Expr'	→ + Expr'	{ + }	Expr' { +, -, ε } { \$ }
	- Expr'	{ - }	Term { num, id } { +, -, \$ }
	ε	{ ε }	Term' { *, /, ε } { +, -, \$ }
Term	→ Factor Term'	{ num, id }	Factor { num, id } { *, /, +, -, \$ }
Term'	→ * Term	{ * }	
	/ Term	{ / }	
	ε	{ ε }	
Factor	→ num	{ num }	
	id	{ id }	

- Using FIRST and FOLLOW, build the LL(1) table for the new grammar

	num	id	+	-	*	/	\$
Goal	Goal → Expr	Goal → Expr					
Expr	Expr → Term Expr'	Expr → Term Expr'					
Expr'			Expr' → + Expr'	Expr' → - Expr'			Expr' → ε
Term	Term → Factor Term'	Term → Factor Term'					
Term'			Term' → ε	Term' → ε	Term' → * Term	Term' → / Term	Term' → ε
Factor	Factor → num	Factor → id					