Name:

Instructions:

1. Write your name at the top of this page.
2. There are 3 problems in this exam and you have 120 minutes to answer them.
3. This is a closed book exam. The algorithms relevant to the exam problems are provided on pages 3 and 4.
4. To receive full credit, your solution must not only be correct but also show all the steps.
5. Discussing the exam contents with anyone who has not taken the exam is a violation of the academic honesty code.

Problem 1. (32 points) Consider the language L of binary strings (ie, strings over the alphabet $\{0, 1\}$) of length at least 1.

a. (2 points) Provide a regular expression for L?

b. (10 points) Use the Thompson's construction algorithm to derive a non-deterministic finite automaton (NFA) that recognizes L. It is enough to draw the final NFA. You must use $0, 1, 2, \ldots$ for state labels.

c. (10 points) Use the subset construction algorithm to derive an equivalent deterministic finite automaton (DFA). You must show the computation of ϵ-closures and draw the DFA.

d. (10 points) Use the partitioning algorithm to derive an equivalent, minimal DFA. You must show the partitioning steps and draw the minimal DFA.

Problem 2. (32 points) Consider the following context-free grammar.

1. S ::= A `a`
2. A ::= `b` B
3. A ::= `c` B
4. A ::= ϵ
5. B ::= `c` A
6. B ::= `d` B
7. B ::= ϵ

a. (8 points) Compute the first sets for S, A, and B. You must show the iterations of the algorithm separately.

b. (8 points) Compute the follow sets for S, A, and B. You must show the iterations of the algorithm separately.

c. (8 points) Construct the LL(1) parsing table for the grammar. It is enough to just show the table.

d. (8 points) Show the steps in parsing the input sentence `c d c b a`.

Problem 3. (36 points) The Action and Goto tables for the grammar

0. S' ::= S
1. S ::= `(` L `)`
2. S ::= `a`
3. L ::= L `,` S
4. L ::= S

are shown below.

	a	,	(	)	#	S	L
0	$s3$		$s2$			1	
1					✓		
2	$s7$		$s6$			5	4
3					$r2$		
4		$s9$		$s8$			
5		$r4$		$r4$			
6	$s7$		$s6$			5	10
7		$r2$		$r2$			
8					$r1$		
9	$s7$		$s6$			11	4
10		$s9$		$s12$			
11		$r3$		$r3$			
12		$r1$		$r1$			

a. (12 points) Show the steps in the LR(1) parse for `( ( a ) )`.

b. (8 points) Compute the itemset $s_0 = \text{closure}(\{[S' \to \cdot S, \#]\})$.

c. (16 points) Compute the following itemsets:

i $s_1 = \text{goto}(s_0, S)$

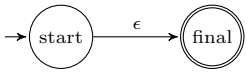
ii $s_2 = \text{goto}(s_0, \texttt{(})$

iii $s_3 = \text{goto}(s_0, \texttt{a})$

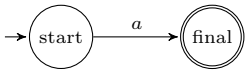
iv $s_6 = \text{goto}(s_2, \texttt{(})$

Thompson's Construction

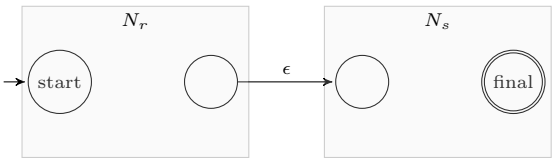
NFA N_r for recognizing $L(r = \epsilon)$



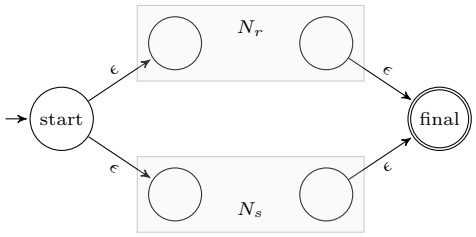
NFA N_r for recognizing $L(r = a)$



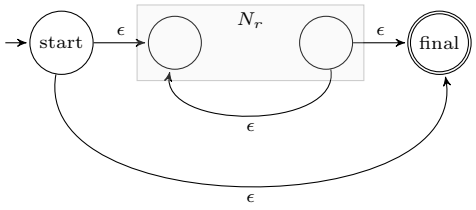
NFA N_{rs} for recognizing $L(rs)$



NFA $N_{r|s}$ for recognizing $L(r|s)$



NFA N_{r*} for recognizing $L(r*)$



NFA N_r for recognizing $L(r)$ also recognizes $L((r))$

ϵ-closure of a Set of States

Input: a set of states S
Output: ϵ-closure(S)

```
P ← Stack(S)
C ← Set(S)
while not P.isEmpty() do
    r ← P.pop()
    for s ∈ m(r, ϵ) do
        if s ∉ C then
            P.push(s)
            C.add(s)
        end if
    end for
end while
return C
```

ϵ-closure of a Single State

Input: a state s
Output: ϵ-closure(s)

```
return ϵ-closure(Set(s))
```

Subset Construction

Input: an NFA $N = (\Sigma, S, s_0, M, F)$
Output: an equivalent DFA $D = (\Sigma, S_D, s_{D0}, M_D, F_D)$

```
s_D0 ← ϵ-closure(s_0)
S_D ← Set(s_D0)
M_D ← Moves()
stk ← Stack(s_D0)
i ← 1
while not stk.isEmpty() do
    r ← stk.pop()
    for a ∈ Σ do
        s_Di ← ϵ-closure(∪_{r'∈r} m(r', a))
        if s_Di ≠ {} then
            if s_Di ∉ S_D then
                S_D.add(s_Di)
                stk.push(s_Di)
                i ← i + 1
                M_D.add((r, a) → s_Di)
            else
                M_D.add((r, a) → s_j), where s_j ∈ S_D such that s_j = s_Di
            end if
        else
            S_D.add(ϕ); M_D.add((r, a) → ϕ); and M_D.add((ϕ, a) → ϕ)
        end if
    end for
end while
F_D ← Set()
for s_D ∈ S_D do
    for s ∈ s_D do
        if s ∈ F then
            F_D.add(s_D)
        end if
    end for
end for
return D = (Σ, S_D, s_D0, M_D, F_D)
```

Partiotioning

Input: a DFA $D = (\Sigma, S, s_0, M, F)$
Output: a partition of S

```
P ← {S − F, F}
while splitting occurs do
    for Q ∈ P do
        if Q.size() > 1 then
            for a ∈ Σ do
                r ← a state chosen from Q
                T ← the subset in the P containing m(r, a)
                Q_1 ← {s ∈ Q|m(s, a) ∈ T}
                Q_2 ← {s ∈ Q|m(s, a) ∉ T}
                if Q_2 ≠ {} then
                    replace Q in P by Q_1 and Q_2
                    break
                end if
            end for
        end if
    end for
end while
return P
```

First Set of a Single Symbol

Input: a context-free grammar $G = (N, T, S, P)$
Output: first(X) for all symbols $X \in T \cup N$

```
for X ∈ T do
    first(X) ← {X}
end for
for X ∈ N do
    first(X) ← {}
end for
if X ::= ϵ ∈ P then
    Add ϵ to first(X)
end if
repeat
    for Y ::= X_1 X_2 ... X_n ∈ P do
        Add first(X_1 X_2 ... X_n) to first(Y)
    end for
until no new symbols are added to any set
```

First Set of a Sequence of Symbols

Input: a context-free grammar $G = (N, T, S, P)$ and a sequence of symbols $X_1 X_2 \dots X_n$
Output: first$(X_1 X_2 \dots X_n)$

```
1: F ← first(X_1)
2: i ← 2
3: while ε ∈ F and i ≤ n do
4:     F ← F − ε
5:     Add first(X_i) to F
6:     i ← i + 1
7: end while
8: return F
```

Follow Set of a Symbol

Input: a context-free grammar $G = (N, T, S, P)$
Output: follow(X) for all symbols $X \in N$

```
1: follow(S) ← {#}
2: for X ∈ N do
3:     follow(X) ← {}
4: end for
5: repeat
6:     for Y ::= X_1 X_2 ... X_n ∈ P do
7:         for X_i ∈ X_1 X_2 ... X_n do
8:             Add first(X_{i+1} X_{i+2} ... X_n) − {ε} to follow(X_i)
9:             If X_i is the last symbol or ε ∈ first(X_{i+1} ... X_n), add
               follow(Y) to follow(X_i)
10:        end for
11:    end for
12: until no new symbols are added to any set
```

LL(1) Parsing

Input: parse table *table*, productions *rules*, and a sentence w
Output: a left-most derivation for w

```
1: stk ← Stack(#, S)
2: sym ← first symbol in w#
3: while true do
4:     top ← stk.pop()
5:     if top = sym = # then
6:         Halt successfully
7:     else if top is a terminal then
8:         if top = sym then
9:             Advance sym to be the next symbol in w#
10:        else
11:            Halt with an error: sym found where top was expected
12:        end if
13:    else if top is a non-terminal Y then
14:        index ← table[Y, sym]
15:        if index ≠ err then
16:            rule ← rules[index]
17:            If Y ::= X_1 X_2 ... X_n, then stk.push(X_n, ..., X_2, X_1)
18:        else
19:            Halt with an error: no rule to follow
20:        end if
21:    end if
22: end while
```

LL(1) Parse Table

Input: a context-free grammar $G = (N, T, S, P)$
Output: LL(1) parse table for G

```
1: for Y ∈ N do
2:     for Y ::= X_1 X_2 ... X_n ∈ P with index i do
3:         for a ∈ first(X_1 X_2 ... X_n) − {ε} do
4:             table[Y, a] ← i
5:             if ε ∈ first(X_1 X_2 ... X_n) then
6:                 for a ∈ follow(Y) do
7:                     table[Y, a] ← i
8:                 end for
9:             end if
10:        end for
11:    end for
12: end for
```

Closure of an Itemset

Input: itemset s
Output: closure(s)

```
1: C ← Set(s)
2: repeat
3:     If C contains an item of the form

       [Y ::= α · X β, a],

       then add the item

       [X ::= · γ, b]

       to C for every rule X ::= γ in P and for every token b in first(βa)
4: until no new items may be added
5: return C
```

goto(s, X)

Input: a state s, and a symbol $X \in T \cup N$
Output: the state goto(s, X)

```
1: r ← Set()
2: for [Y ::= α · Xβ, a] ∈ s do
3:     r.add([Y ::= αX · β, a])
4: end for
5: return closure(r)
```

LR(1) Parsing

Input: Action and Goto tables and a sentence w
Output: a right-most derivation in reverse

1: Initially, the parser has the configuration,

Stack	Input
s_0	$a_1 a_2 \dots a_n \#$

where $a_1 a_2 \dots a_n$ is the input sentence

2: **repeat**

3: If Action$[s_m, a_k] = ss_i$, the parser executes a shift (the s stands for "shift") and goes into state s_i

Stack	Input
$s_0 X_1 s_1 X_2 s_2 \dots X_m s_m a_k s_i$	$a_{k+1} \dots a_n \#$

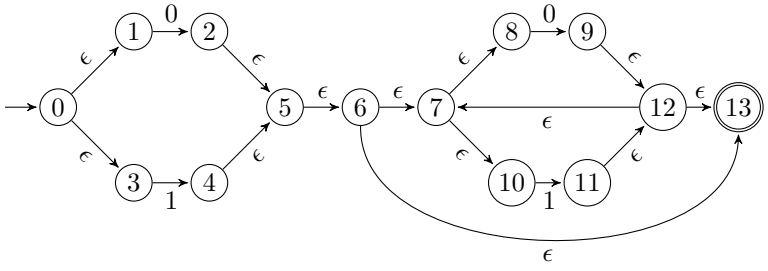
4: Otherwise, if Action$[s_m, a_k] = ri$ (the r stands for "reduce"), where i is the index of the rule $Y ::= X_j X_{j+1} \dots X_m$, the parser replaces the symbols and states $X_j s_j X_{j+1} s_{j+1} \dots X_m s_m$ by $Y s$, where $s =$ Goto$[s_{j-1}, Y]$, and outputs i

Stack	Input
$s_0 X_1 s_1 X_2 s_2 \dots X_{j-1} s_{j-1} Y s$	$a_{k+1} \dots a_n \#$

5: Otherwise, if Action$[s_m, a_k]$ = accept, the parser halts successfully
6: Otherwise, if Action$[s_m, a_k]$ = error, the parser raises an error
7: **until** either the sentence is parsed or an error is raised

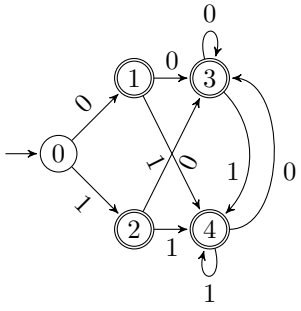
Solution 1. a. $(0|1)(0|1)*$

b.



c.

r	a	$m(r,a)$
$s_0 = \epsilon$-closure$(0) = \{0,1,3\}$	0	ϵ-closure$(2) = \{2,5,6,7,8,10,13\} = s_1$
s_0	1	ϵ-closure$(4) = \{4,5,6,7,8,10,13\} = s_2$
s_1	0	ϵ-closure$(9) = \{7,8,9,10,12,13\} = s_3$
s_1	1	ϵ-closure$(11) = \{7,8,10,11,12,13\} = s_4$
s_2	0	ϵ-closure$(9) = s_3$
s_2	1	ϵ-closure$(11) = s_4$
s_3	0	ϵ-closure$(9) = s_3$
s_3	1	ϵ-closure$(11) = s_4$
s_4	0	ϵ-closure$(9) = s_3$
s_4	1	ϵ-closure$(11) = s_4$

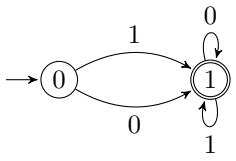


d. The initial partition is $\mathcal{P} = \{\{0\}, \{1,2,3,4\}\}$.

The subset $\{1,2,3,4\}\}$ does not split on the symbol 0 because from every state in the subset, the DFA transitions to 3 (ie, an identical subset) on a 0.

The subset $\{1,2,3,4\}\}$ does not split on the symbol 1 because from every state in the subset, the DFA transitions to 4 (ie, an identical subset) on a 1.

So the final partition is $\mathcal{P} = \{\{0\}, \{1,2,3,4\}\}$. Labeling the two subsets of $\mathcal{P}$ as 0 and 1 , we have the following minimal DFA:



Solution 2. a.

X	first(X) (iteration 1)	first(X) (iteration 2)
S	{a}	{a, b, c}
A	{ϵ, b, c}	{ϵ, b, c}
B	{ϵ, c, d}	{ϵ, c, d}

b.

X	follow(X) (iteration 1)
S	{#}
A	{a}
B	{a}

c.

	a	b	c	d	#
S	1	1	1		
A	4	2	3		
B	7		5	6	

d.

Stack	Input	Output
#S	cdcba#	1
#aA	cdcba#	3
#aBc	cdcba#	-
#aB	dcba#	6
#aBd	dcba#	-
#aB	cba#	5
#aAc	cba#	-
#aA	ba#	2
#aBb	ba#	-
#aB	a#	7
#a	a#	-
#	#	✓

Solution 3. a.

Stack	Input	Output
0	`((a))#`	$s2$
0`(`2	`(a))#`	$s6$
0`(`2`(`6	`a))#`	$s7$
0`(`2`(`6`a`7	`))#`	$r2$
0`(`2`(`6S5	`))#`	$r4$
0`(`2`(`6L10	`))#`	$s12$
0`(`2`(`6L10`)`12	`)#`	$r1$
0`(`2S5	`)#`	$r4$
0`(`2L4	`)#`	$s8$
0`(`2L4`)`8	`#`	$r1$
0S1	`#`	✓

b. $s_0 = \epsilon\text{-closure}(\{[S' ::= \cdot S, \texttt{\#}]\}) = \{[S' ::= \cdot S, \texttt{\#}], [S ::= \cdot \texttt{(}L\texttt{)}, \texttt{\#}], [S ::= \cdot \texttt{a}, \texttt{\#}]\}$

c. $s_1 = \text{goto}(s_0, S) = \epsilon\text{-closure}(\{[S' ::= \cdot S, \texttt{\#}]\}) = \{[S' ::= S\cdot, \texttt{\#}]\}$

$s_2 = \text{goto}(s_0, \texttt{(}) = \epsilon\text{-closure}(\{[S ::= \cdot \texttt{(}L\texttt{)}, \texttt{\#}]\}) = \{[S ::= \texttt{(} \cdot L\texttt{)}, \texttt{\#}], [L ::= \cdot L\texttt{,}S, \texttt{)}], [L ::= \cdot S, \texttt{)}][S ::= \cdot \texttt{(}L\texttt{)}, \texttt{)}][S ::= \cdot \texttt{a}, \texttt{)}]\}$

$s_3 = \text{goto}(s_0, \texttt{a}) = \epsilon\text{-closure}(\{[S ::= \cdot \texttt{a}, \texttt{\#}]\}) = \{[S ::= \texttt{a}\cdot, \texttt{\#}]\}$

$s_6 = \text{goto}(s_2, \texttt{(}) = \epsilon\text{-closure}(\{[S ::= \texttt{(} \cdot L\texttt{)}, \texttt{)}]\}) = \{[S ::= \texttt{(} \cdot L\texttt{)}, \texttt{)}], [L ::= \cdot L\texttt{,}S, \texttt{)}], [L ::= \cdot S, \texttt{)}][S ::= \cdot \texttt{(}L\texttt{)}, \texttt{)}][S ::= \cdot \texttt{a}, \texttt{)}]\}$