$q_1 = \varepsilon + q_1 b$
$q_2 = q_1 a + q_2 a + q_3 a$
$q_3 = q_2 b$

a aa ab ac ...

# 2023 NCKU CSIE Compiler Midterm Exam

$q_1 = b^*$

1. (25%) Write regular expressions for the following languages: a e i o u
   (a) (5%) All strings of lowercase letters that contain the five vowels in order. a b c d
   (b) (5%) All strings of lowercase letters in which the letters are in ascending lexicographic order.
   (c) (5%) Comments, consisting of a string surrounded by /* and */, without an intervening */, unless it is inside double-quotes (").
   (d) (5%) All strings of a's and b's that do not contain the substring abb. $a^*b^*c^*$
   (e) (5%) Defines a C-like, fixed-decimal literal with no superfluous leading or trailing zeros. That is, 0.0, 123.01, and 123005.0 are legal, but 00.0, 001.000, and 002345.1000 are illegal.

$q_2 = b^*a + q_2 a + q_2 ba$
$= b^*a + q_2(a+ba)$
$q_2 = b^*a(a+ba)^*$
$q_3 = q_2 b$

0 $1^*\Sigma^\wedge$

$b^* \mid q_2 \mid q_2 b$
$b^* + q_2(\varepsilon + b)$

2. (20%)
   (a) (10%) Use Thompson's construction to convert the regular expression $(a|b)^*a(a|b|\epsilon)^*$ into an NFA.
   (b) (10%) Convert the NFA of part (a) into a DFA using the subset construction.

3. (15%) Given the grammar

$$
\begin{aligned}
&Exp \rightarrow Exp\ Addop\ Term \mid Term \\
&Addop \rightarrow + \mid - \\
&Term \rightarrow Term\ Mulop\ Factor \mid Factor \\
&Mulop \rightarrow * \\
&Factor \rightarrow (\ Exp\ ) \mid number
\end{aligned}
$$

Exp — Exp Addop Term — Factor — ( Exp ) — Exp A T

write down leftmost derivations, parse trees, and abstract syntax trees for the following expressions:
   (a) (5%) $3 + 4 * 5 - 6$
   (b) (5%) $3 * (4 - 5 + 6)$
   (c) (5%) $3 - (4 + 5 * 6)$

( ) a+axb

4. (10%) Write a grammar for Boolean expressions that includes the constants **true** and **false**, the operators **and**, **or**, and **not**, and parentheses. Be sure to give **or** a lower precedence than **and** and **and** a lower precedence than **not** and to allow repeated **not**'s, as in the Boolean expression **not not true**. Also be sure your grammar is not ambiguous.

not > and > or

S → AS | (A) | A
A → true | false | B
B → or | C
C → and | D
D → not D | λ

5. (10%) Given the grammar

$$
\begin{aligned}
&Statement \rightarrow Assign\text{-}stmt \mid Call\text{-}stmt \mid other \\
&Assign\text{-}stmt \rightarrow identifier\ :=\ Exp \\
&Call\text{-}stmt \rightarrow identifier\ (\ Exp\text{-}list\ )
\end{aligned}
$$

write pseudocode to parse this grammar by recursive-descent.

6. (20%) Consider the following grammar, which is already suitable for LL(1) parsing:

$$
\begin{aligned}
&Start \rightarrow Value\ \$ \\
&Value \rightarrow num \mid lparen\ Expr\ rparen \\
&Expr \rightarrow plus\ Value\ Value \mid prod\ Values \\
&Values \rightarrow Value\ Values \mid \lambda
\end{aligned}
$$

   (a) (5%) Construct First and Follow sets for each nonterminal in the grammar.

A A A A
not not λ

(b) (5%) Construct the Predict sets for the grammar.
(c) (5%) Build an LL(1) parse table based on the grammar.
(d) (5%) Use the Table-driven LL parser for the following expression:
*lparen plus num lparen prod num num num rparen rparen* $

7. (10%) Given the grammar

$$A_1 \to A_1\alpha_1 \mid A_1\alpha_2 \mid A_1\alpha_3 \mid A_2\beta_1 \mid A_3\beta_2$$
$$A_2 \to A_2\alpha_4 \mid A_1\beta_3 \mid A_3\beta_4$$
$$A_3 \to A_3\alpha_5 \mid A_1\beta_5 \mid A_2\beta_6$$

Please convert left recursion to right recursion.

$A_1 \to \alpha_1 A_1 \mid \alpha_2 A_1 \mid \alpha_3 A_1$

$A_1' \to A_2\beta_1' \mid A_3\beta_2' \mid \varepsilon$

$A_2 \to X_4 A_2$

$A_2' \to A_1\beta_3 A_2' \mid A_3\beta_4 A_2' \mid \varepsilon$


$A \to Aa \mid b$

$A \to aA'$ (+3(

$A' \to bA' \mid \varepsilon$

$bA'$

$A' \to bA \mid \varepsilon$

$A \to aA \mid bA'$

$A' \to bA' \mid \varepsilon$

$A \to \beta A'$

$A' \to aA' \mid \varepsilon$

$A \to \beta A$

$A \to \beta A' \mid aA' \mid \varepsilon$

$A \to Aa \mid \beta$

$A \to \beta A'$

$A' \to aA' \mid \varepsilon$

(1)

(1-a)

Letter → [b-d f-h j-n p-t v-z]

String → (Letter|a)* (Letter|e)* (Letter|i)* (Letter|o)* (Letter|u)*

String ->
Letter*a+(Letter|a)*Letter*e+(Letter|e)*Letter*i+(Letter|i)*Letter*o+(Letter|o)*Letter*u+(Letter|u)*

(1-b)

String → a*b*c*d*............z*

(1-c)

Character → [a-zA-Z0-9]

Comment → /* (Character|“*/”|(*|/) Character)* */

(1-d)

b* (a (∈|b))*

(1-e)

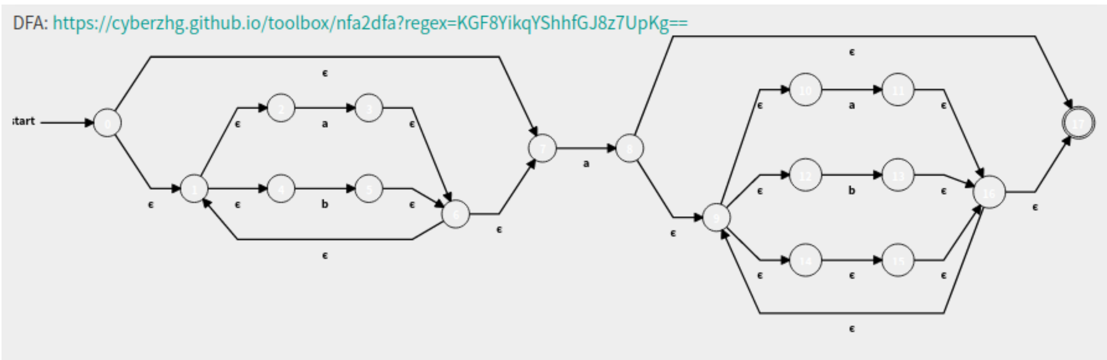
Let DNOTZ be the set of digits from 1 to 9.

Let D be the set of digits from 0 to 9.

Define (0 | (DNOTZ D*) . (0 | (D* DNOTZ)

(2)

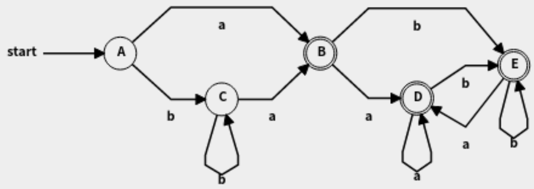
(2-a)




(2-b)

NFA: https://cyberzhg.github.io/toolbox/regex2nfa?regex=KGF8YikqYShhfGJ8z7UpKg==
Min-DFA: https://cyberzhg.github.io/toolbox/min_dfa?regex=KGF8YikqYShhfGJ8z7UpKg==

| NFA STATE | DFA STATE | TYPE | a | b |
|---|---|---|---|---|
| {0,1,2,4,7} | A | | B | C |
| {1,2,3,4,6,7,8,9,10,12,14,15,16,17} | B | accept | D | E |
| {1,2,4,5,6,7} | C | | B | C |
| {1,2,3,4,6,7,8,9,10,11,12,14,15,16,17} | D | accept | D | E |
| {1,2,4,5,6,7,9,10,12,13,14,15,16,17} | E | accept | D | E |




(3)

(3-a)

exp → exp addOp term
→ exp addOp term addOp term
→ term addOp term addOp term
→ factor addOp term addOp term
→ number addOp term addOp term
→ number + term addOp term
→ number + term multop factor addOp term
→ number + factor multop factor addOp term
→ number + number multop factor addOp term
→ number + number * factor addop term
→ number + number * number addop term
→ number + number * number – term
→ number + number * number – factor
→ number + number * number – number

(3-b)

exp → term
→ term multop factor
→ factor multop factor
→ number multop factor
→ number multop ( exp )
→ number * ( exp )
→ number * ( exp addop term )
→ number * ( exp addop term addop term )
→ number * ( term addop term addop term)
→ number * ( factor addop term addop term)
→ number * ( number addop term addop term)
→ number * ( number – term addop term)
→ number * ( number – factor addop term)
→ number * ( number – number addop term)
→ number * ( number – number + term)
→ number * ( number –number + factor )
→ number * ( number –number + number )

(3-c)

exp → exp addop term
→ term addop term
→ factor addop term
→ number addop term
→ number – term
→ number – factor
→ number – ( exp )
→ number – ( exp addop term )
→ number –( term addop term )
→ number – ( factor addop term )
→ number – ( number addop term )
→ number – ( number +term)
→ number – ( number + term multop factor )
→ number – ( number + factor multop factor)
→ number – ( number + number multop factor)
→ number – ( number + number * factor )
→number – ( number + number * number)

(4)

$$
\begin{array}{lcl}
E & \rightarrow & E \texttt{ or } T \mid T \\
T & \rightarrow & T \texttt{ and } F \mid F \\
F & \rightarrow & \texttt{not } F \mid B \\
B & \rightarrow & \texttt{true} \mid \texttt{false} \mid ( E )
\end{array}
$$

5.
https://ideone.com/bAPCYW

6.
(6-a)

| FIRST | FOLLOW | Nonterminal |
|---|---|---|
| {num,lparen} | {$} | Start |
| {num,lparen} | {$,num,lparen,rparen} | Value |
| {plus,prod} | {rparen} | Expr |
| {num,lparen,''} | {rparen} | Values |

(6-b)
Predirct(Start->Value)={num, lparen}
Predirct(Value->num)={num}
Predirct(Value->lparen Expr rparen)={lparen}
Predirct(Expr->plus Value Value)={plus}
Predirct(Expr->prod Values)={prod}
Predirct(Values->Value  Values)={num, lparen}
Predirct(Values->λ)={rparen}
(6-c)

| Nonterminal | num | lparen | rparen | plus | prod | $ |
|---|---|---|---|---|---|---|
| Start | Start->Value | Start->Value | | | | |
| Value | Value->num | Value->lparen Expr rparen | | | | |
| Expr | | | | Expr->plus Value Value | Expr->prod Values | |
| Values | Values->Value Values | Values->Value Values | Values-> '' | | | |

(6-d)

| Trace | | |
|---|---|---|
| **Stack** | **Input** | **Rule** |
| $ Start | lparen plus num lparen prod num num num rparen rparen $ | |
| $ Value | lparen plus num lparen prod num num num rparen rparen $ | Start->Value |
| $ rparen Expr lparen | lparen plus num lparen prod num num num rparen rparen $ | Value->lparen Expr rparen |
| $ rparen Expr | plus num lparen prod num num num rparen rparen $ | |
| $ rparen Value Value plus | plus num lparen prod num num num rparen rparen $ | Expr->plus Value Value |
| $ rparen Value Value | num lparen prod num num num rparen rparen $ | |
| $ rparen Value num | num lparen prod num num num rparen rparen $ | Value->num |
| $ rparen Value | lparen prod num num num rparen rparen $ | |
| $ rparen rparen Expr lparen | lparen prod num num num rparen rparen $ | Value->lparen Expr rparen |
| $ rparen rparen Expr | prod num num num rparen rparen $ | |
| $ rparen rparen Values prod | prod num num num rparen rparen $ | Expr->prod Values |
| $ rparen rparen Values | num num num rparen rparen $ | |
| $ rparen rparen Values Value | num num num rparen rparen $ | Values->Value Values |
| $ rparen rparen Values num | num num num rparen rparen $ | Value->num |
| $ rparen rparen Values | num num rparen rparen $ | |
| $ rparen rparen Values Value | num num rparen rparen $ | Values->Value Values |
| $ rparen rparen Values num | num num rparen rparen $ | Value->num |
| $ rparen rparen Values | num rparen rparen $ | |
| $ rparen rparen Values Value | num rparen rparen $ | Values->Value Values |
| $ rparen rparen Values num | num rparen rparen $ | Value->num |
| $ rparen rparen Values | rparen rparen $ | |
| $ rparen rparen | rparen rparen $ | Values-> '' |
| $ rparen | rparen $ | |
| $ | $ | |

7

A1 -> A2 β1 A1'
    | A3 β2 A1'
A2 -> A3 β2 A1' β3 A2'
    | A3 β4 A2'
A1' -> α1 A1'
    | α2 A1'
    | α3 A1'
    | ϵ
A2' -> α4 A2'
    | β1 A1' β3 A2'
    | ϵ
A3' -> α5 A3'
    | β2 A1' β3 A2' β1 A1' β5 A3'
    | β4 A2' β1 A1' β5 A3'
    | β2 A1' β5 A3'
    | β2 A1' β3 A2' β6 A3'
    | β4 A2' β6 A3'
    | ϵ