Dashboard / My courses / COSC261-2022 / Exam / Final Exam 2022

Question **1**

Not complete

Marked out of 6.00

Consider the regular expression r = `((q|n)fc*|h)(ku|s*)*(b*|wp*|z)`

Give a regular expression that generates { reversed(x) | x in L(r) }, where reversed(x) is the reverse of string x (that is, the string read backwards).

Regular expression format:

Any lower-case letter is a terminal. A bar `|` is used for alternatives and a star `*` is used for repetition. Parentheses `( )` are used to specify precedence. Without parentheses the usual precedence rules of regular expressions apply.

More precisely, your answer must be a regular expression according to the following BNF:

Regex = `a-z` | `(`*Regex*`)` | *Regex*`*` | *Regex*`|`*Regex* | *Regex Regex*

The above expression `((q|n)fc*|h)(ku|s*)*(b*|wp*|z)` is a syntactically correct example. Your answer should be formatted similarly.

Precheck tests whether your answer uses the correct format.

Note: Check will normally be fast, but might take a while in some cases (especially if the answer is unnecessarily complex).

Answer: (penalty regime: 25, 50, ... %)

```
(z|p*w|*b)*(*s|uk)(h|*cf(n|q))
```

Precheck Check

Precheck only

	Got
✗	syntax error

Information

The following questions ask you to write CFGs. An example illustrates the encoding you have to use for submission. Assume your CFG has the productions

S → aTbA | AcT
A → a | AT | ε
T → bc

Then you'd submit the following:

```
S=aTbA|AcT
A=a|AT|_
T=bc
```

You can use any single capital letter as a non-terminal; `S` is the start symbol. Any lower-case letter and any digit is a terminal. An underscore _ denotes the empty string ε. There must be exactly one line for each non-terminal. A bar `|` is used to separate the choices on the right-hand side. Right-hand sides must not be empty; at least one option must be given. Each choice in a right-hand side must not be empty; it may be _ for ε. Productions must not contain spacing. In summary, every production must conform to the following Python regular expression:

```
^[A-Z]=[A-Za-z0-9_]+(\|[A-Za-z0-9_]+)*$
```

Please use the Precheck button to partially check the syntax of your CFG representation before submitting it to a question using the Check button. A precheck attracts no marks and no penalty; you can use it any number of times and also after incorrect submissions. If your representation fails the precheck, it will fail the full check as well. If it passes the precheck, it may or may not pass the full check.

Question **2**
Not complete
Marked out of 8.00

Give a regular grammar that generates the language `{ na, nmzd, nmzg, wf, ε }` over the alphabet Σ = `{ a, d, f, g, m, n, w, z }`.

Answer: (penalty regime: 25, 50, ... %)

```
S=NA|NMZD|NMZG|WF|_
N=n|_
A=aA|_
M=mM|_
Z=zZ|_
D=dD|_
G=gG|_
W=wW|_
F=fF|_
```

Precheck Check

Precheck only

	Got
✗	The CFG is not a regular grammar.

Question **3**

Answer saved

Marked out of 8.00

Consider the CFG G_1 induced by the following productions:

```
A = AB | DAa | ε
B = AA | aBa | DaC
C = BC | Ba
D = aA | ab
```

How many strings of length 2 or less are generated by each non-terminal of G_1?

Each of your answers must be an integer.

Make sure your answers pass the Precheck. There is no Check since this question will be marked later.

Precheck tests whether each of your answers is an integer.

Answer: (penalty regime: 0 %)

A:	3
B:	2
C:	2
D:	2

Precheck

Information

The following questions ask you to construct finite automata. Draw the automaton in the provided box. If the same transition can be taken under two or more symbols, list the symbols separated by vertical bars in the transition's label. For example, the label `0|1` indicates that a transition can be taken under 0 or under 1. Any spaces in a transition label will be ignored.

Please use the Precheck button to partially check your automaton before submitting it to a question using the Check button. A precheck attracts no marks and no penalty; you can use it any number of times and also after incorrect submissions. If your representation fails the precheck, it will fail the full check as well. If it passes the precheck, it may or may not pass the full check.

Question **4**

Not complete

Marked out of 8.00

Note: there are different ways to construct an answer to this question, some of which might take too much time. If you cannot think of a method that is fast enough, consider working on other questions first.

Consider the DFA M_3 with start state q0, accept states {q1} and the following transition table:

δ_3	0	1
q0	q0	q1
q1	q1	q0

Consider the DFA M_4 with start state p0, accept states {p0,p1} and the following transition table:

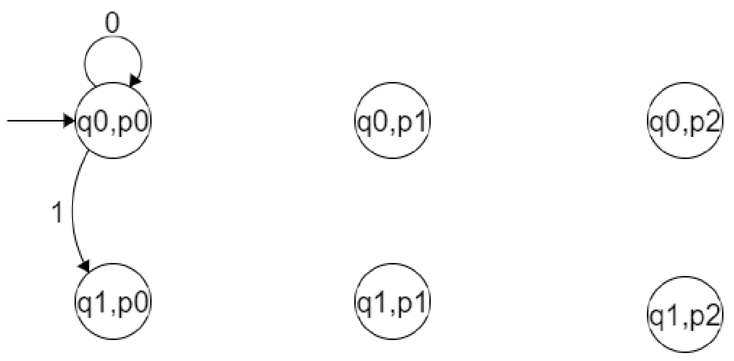
δ_4	0	1
p0	p1	p2
p1	p1	p0
p2	p2	p1

Construct a DFA that accepts the symmetric difference of $L(M_3)$ and $L(M_4)$.

The symmetric difference of two languages is the set of all strings that are in exactly one of the two languages.

Answer: (penalty regime: 25, 50, ... %)

Help | Clear



Precheck | Check

Question **5**

Incorrect

Marked out of 8.00

Consider the DFA M_5 with start state q0, accept states {q2,q3} and the following transition table:

δ_5	a	b	c	d
q0	q0	q1	q3	q2
q1	q1	q0	q2	q3
q2	q1	q2	q3	q0
q3	q3	q2	q0	q1

Consider the DFA M_6 with start state p0, accept states {p1,p3} and the following transition table:

δ_6	a	b	c	d
p0	p0	p3	p2	p1
p1	p0	p1	p3	p2
p2	p1	p0	p3	p2
p3	p2	p0	p1	p3

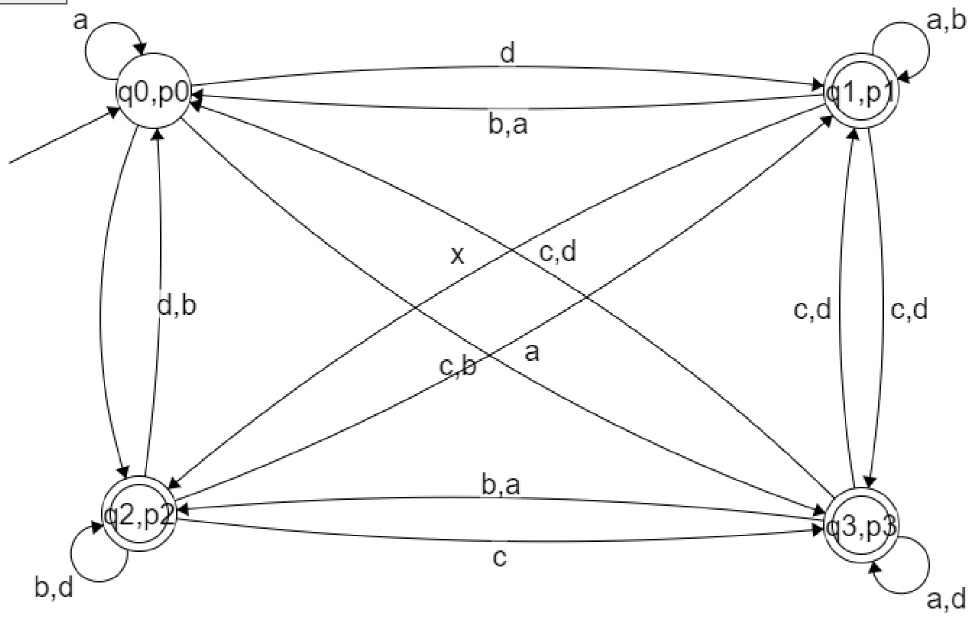
Construct an NFA that accepts the concatenation $L(M_5)L(M_6)$. The NFA must not contain ε-transitions.

For reference, the box below is initialised with the two given DFAs (M_5 above M_6). You may modify these to construct your answer or delete them and construct your answer from scratch. Reset answer will show the two DFAs again but discard your previous work, so you should be quite sure your strategy works before implementing it.

Answer: (penalty regime: 25, 50, ... %)

Reset answer

Help Clear



Precheck Check

	Got
✗	`a transition from state q0,p0 to state q2,p2 has no label`

Your code must pass all tests to earn any marks. Try again.

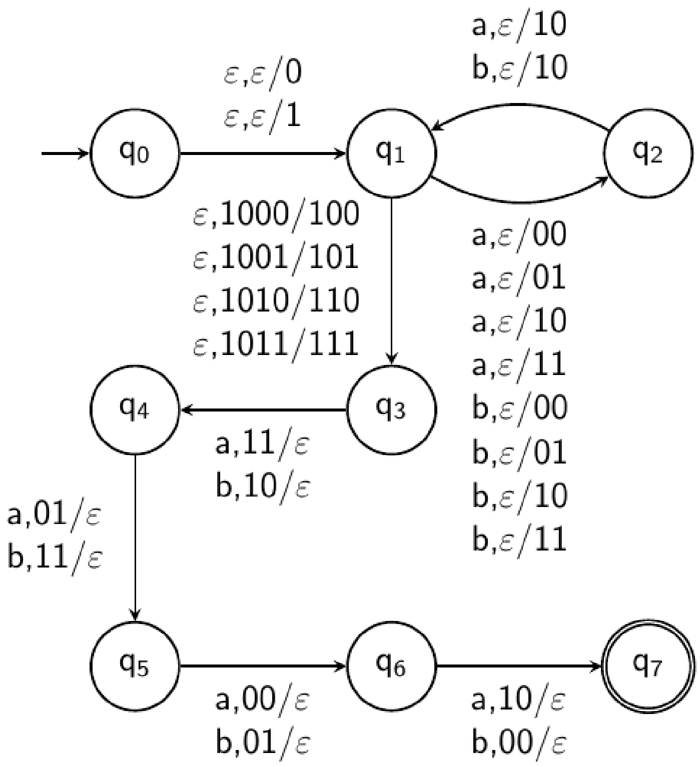
Question **6**

Precheck results

Marked out of 8.00

Note: you might find this question more challenging than it appears at first sight. Consider working on other questions first before you spend too much time on this one.

Consider the following PDA M_7:



Give an accepting sequence of configurations for input `babaaaab` using M_7.

Answer format:

- Separate the configurations by commas without extra spacing.
- Use `e` instead of ε.
- For example, using a different PDA, the configuration `(q1,xyz,20202021)` means that the PDA is in state q_1 and the remaining input is xyz and the stack contains 20202021 with 2 at the top of the stack and 1 at its bottom.
- For another example, using a different PDA, the configuration sequence $(q_1,ba,110) \rightarrow (q_0,a,\varepsilon) \rightarrow (q_0,a,1) \rightarrow (q_0,\varepsilon,01)$ would be written as `(q1,ba,110),(q0,a,e),(q0,a,1),(q0,e,01)`

Make sure your answer passes the Precheck. There is no Check since this question will be marked later.

Precheck tests whether your answer uses the correct format.

Answer: (penalty regime: 0 %)

```
configuration sequence:
(q0,babaaaab,e),(q2,abaaab,001),(q1,baaaab,00110),(q2,aaaab,0011010),(q1,aaab,001101010),(q3,aaab,00110110),(q4,aab,001101),(q5
```

Precheck

Precheck only

	Got
✓	Good

Question **7**

Answer saved

Marked out of 8.00

A scanner recognises the following tokens:

FROM	=	`from`
TO	=	`to`
MUL	=	`*`
MULBEC	=	`*=`
EXPBEC	=	`**=`
EXP	=	`**`
DBLEQ	=	`==`
EQ	=	`=`
POINT	=	`.`
DOLLAR	=	`$`
LBRACK	=	`[`
RBRACK	=	`]`
DIGIT	=	`0-9`
NUM	=	DIGIT$^+$
FLOAT	=	NUM POINT NUM
MONEY	=	DOLLAR FLOAT
LETTER	=	`a-z`
ID	=	LETTER$^+$
ARRAY	=	ID LBRACK (ID | NUM) RBRACK
MAGIC	=	`42`
EXCEPT	=	`except`

In this sequence of regular definitions,

- all uppercase characters belong to non-terminals (naming the tokens),
- all lower-case characters and all digits belong to terminals,
- all symbols on the right-hand sides of the first 12 rules (up to RBRACK) belong to terminals,
- all other symbols are part of the language defining the tokens, and
- spacing does not indicate space characters in the input.

At each call, the scanner first skips any whitespace characters and then tries to match as much of the remaining input as possible. The token matching the most characters is returned (and the matched characters are consumed). If the same number of characters matches two tokens, the first according to the above list is preferred. If no tokens match the start of the remaining input, the scanner yields a special ERROR token and skips one character.

Give the sequence of tokens returned by the scanner for the following input:

```
except $876.029 42 **== from edn[128.236^ to[441] ***== )except[frr] $701
```

Answer format:

- Separate the tokens by commas without extra spacing.
- Each token must be either ERROR or one of the names on the left-hand sides of the above token definitions.
- In actual scanners, some token types are annotated with values; do not include values here.
- For example, using a different string, a token sequence could be `POINT,ERROR,NUM`

Make sure your answer passes the Precheck. There is no Check since this question will be marked later.

Precheck tests whether your answer uses the correct format.

Answer: (penalty regime: 0 %)

```
token sequence:
ID,MONEY,NUM,EXPBEC,EQ,FROM,ID,LBRACK,FLOAERROR,ARRAY,EXP,MULBEC,EQ,ERROR,ARRAY,DOLLAR,NUM
```

Precheck

Precheck only

	Got

	Got
✓	Good

Question **8**

Incorrect

Marked out of 8.00

The following extended-BNF rule describes materials:

Material = `yellow` `blue` Ceramic [`green` Glass] (`brown` Wood | `pink` Concrete `white` Plastic)* `red`

Implement a Python 3 function `material()` that recognises materials as part of a recursive-descent parser (for a language with further extended-BNF rules, which are parsed by other functions).

Follow the structure of recursive-descent parsers used in the lecture notes/assignment. Your code does not need to construct a syntax tree. Your code should not do any error-handling beyond that automatically provided by using the scanner. Your code may use only the following functions, which have their usual meaning in a recursive-descent parser:

```
consume(*expected_tokens)
lookahead()
ceramic()
concrete()
glass()
plastic()
wood()
```

These functions will be provided at the global level and you should not implement them. In particular, `consume` and `lookahead` are global functions, not methods of a scanner class.

Your code may use only the following tokens (note that tokens are simply Python strings):

```
'blue'
'brown'
'green'
'pink'
'red'
'white'
'yellow'
```

Your code should not use any other strings (in particular, no docstrings).

Precheck will test whether your answer is syntactically correct Python 3 code, defines exactly one function called `material`, calls only functions listed above and uses only strings listed above (the tokens). It then combines your answer with a scanner, embeds it into a recursive-descent parser and runs it with some test inputs. A failure at that stage typically means your code is not following the structure of recursive-descent parsers. This does **not** catch some errors such as parsing the inputs in a wrong way.

Time left 0:00:53

Check will additionally test that your parser works as expected on some inputs (which may or may not be materials according to the above EBNF rule). If your answer is incorrect, an example input will be given for which your parser does not work as expected and a penalty will apply.

Answer: (penalty regime: 25, 50, ... %)

Reset answer

```python
def material():
    # recursive-descent parser for the extended-BNF rule
    #   Material = yellow blue Ceramic [green Glass] (brown Wood | pink Concrete white Plastic)* red
    # your code goes here
    consume(yellow)
    consume(blue)
    ceramic()
    if (lookahead() == green):
        consume(green)
        glass()
    while lookahead() in [brown, pink]:
        if (lookahead() == brown):
            consume(brown)
            wood()
        else:
            consume(pink)
            concrete()
            consume(white)
            plastic()
    consume(red)
```

Precheck Check

	Got
✗	The parser does not work as expected for some inputs; here is one such input: yellow blue Ceramic red

Question **9**

Correct

Marked out of 8.00

The following are instructions of the Java virtual machine:

`sipush` *n*	push the integer constant *n* on the stack
`iadd`	pop two integers from the stack and push their sum on the stack
`imul`	pop two integers from the stack and push their product on the stack
`iload` *n*	push the value stored at relative location *n* on the stack
`istore` *n*	pop a value from the stack and store it at relative location *n*

Compilation of a program **P** without optimisation resulted in the following instruction sequence for the Java virtual machine:

```
iload 2
sipush 9
iload 1
iadd
iadd
iload 0
imul
sipush 4
imul
istore 1
```

Assume relative location 0 for variable `c`, location 1 for variable `r` and location 2 for variable `w`. Other variables have higher locations and should not be used.

Give the program **P** for which this instruction sequence was generated.

The program should conform to the following syntax:

Program = *Statements*
Statements = *Statement* (`;` *Statement*)*
Statement = *If* | *While* | *Assignment*

If = `if` *Comparison* `then` *Statements* `end`
While = `while` *Comparison* `do` *Statements* `end`
Assignment = identifier `:=` *Expression*

Comparison = *Expression Relation Expression*
Relation = `=` | `!=` | `<` | `<=` | `>` | `>=`

Expression = number | identifier | *Factor Arithmetic Factor*
Factor = number | identifier | `(`*Factor Arithmetic Factor*`)`
Arithmetic = `+` | `-` | `*` | `/`

Extra spaces in your answer will be ignored. Identifiers contain only lower-case letters. Numbers are represented by non-negative integers in decimal notation without extra leading zeros.

Precheck tests whether your answer conforms to this syntax without penalty.

Check compiles your answer. If it is not syntactically correct a penalty applies, so do a Precheck first. If it is syntactically correct but does not yield the above instruction sequence, the generated instruction sequence is shown and a penalty applies.

Answer: (penalty regime: 25, 50, ... %)

```
r := ((w + (9 + r)) * c) * 4
```

Precheck Check

	Got
✓	correct

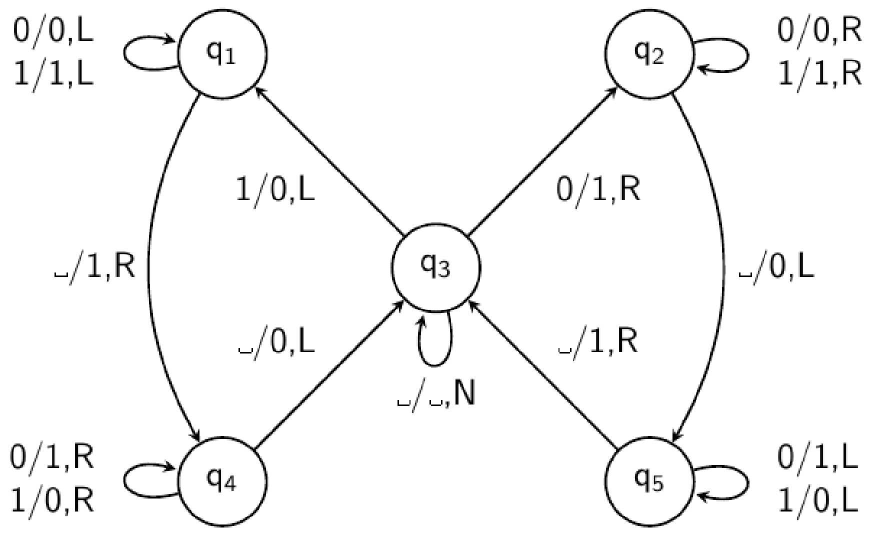
Passed all tests! ✓

Question **10**

Precheck results

Marked out of 8.00

Consider the following Turing machine M_8:



The start state is not shown but specified below. There are no accept or reject states, so the TM does not halt.

Assume M_8 starts in state q_3, the tape content is `1010` (with infinitely many blanks to the left and to the right), and M_8's head is under the fourth non-blank symbol (counting from left to right).

Give the sequence of the first 9 configurations for M_8 (this includes the starting configuration as specified above).

Answer format:

- Separate the configurations by commas without extra spacing.
- Use `e` instead of ε.
- Use _ for the blank symbol.
- In a configuration (x,q,y) omit blanks at the beginning of x and at the end of y, but show all other blanks.
- For example, using a different TM, the configuration `(00,q9,_00)` means that the tape contains `00_00` (and blanks everywhere else) and the TM is in state q_9 and its head is under the cell with the middle blank symbol.
- For another example, using a different TM, the configuration sequence $(\varepsilon,q_0,101) \rightarrow (1,q_0,01) \rightarrow (10,q_0,1) \rightarrow (101,q_0,\varepsilon) \rightarrow (10,q_1,1) \rightarrow (1,q_1,00) \rightarrow (1,q_a,10)$ would be written as `(e,q0,101),(1,q0,01),(10,q0,1),(101,q0,e),(10,q1,1),(1,q1,00),(1,qa,10)`

Make sure your answer passes the Precheck. There is no Check since this question will be marked later.

Precheck tests whether your answer uses the correct format.

Answer: (penalty regime: 0 %)

```
configuration sequence:
(101,q3,0),(1011,q2,e),(101,q5,10),(10,q5,100),(1,q5,0000),(e,q5,11000),(e,q5,_01000),(1,q3,01000),(11,q2,1000)
```

Precheck

Precheck only

	Got
✓	Good

Information

The following questions ask you to construct TMs. Draw the automaton in the provided box. Use the format `a/b,X` for transition labels as described in the lecture notes. You may combine two or more such labels into one by listing them separated by vertical bars as, for example, in `a/b,X | c/d,Y`. Any spaces in a transition label will be ignored.

There must be exactly one accept state, which must not have any outgoing transitions. Omit the reject state and all transitions to the reject state. If a state has no transition under a certain symbol, this implicitly results in a transition to the reject state.

The input alphabet is given in each question. The tape alphabet comprises the input alphabet and the blank symbol _ denoted by the underscore character. Some questions may advise to use additional tape symbols.

Your TM may take only a certain number of steps until it halts. The bound depends on the problem and the input; it is chosen in a generous way. If your TM exceeds the bound, you will have to devise a more efficient solution.

Please use the Precheck button to partially check your TM before submitting it to a question using the Check button. A precheck attracts no marks and no penalty; you can use it any number of times and also after incorrect submissions. If your representation fails the precheck, it will fail the full check as well. If it passes the precheck, it may or may not pass the full check.

Question **11**

Correct

Marked out of 8.00

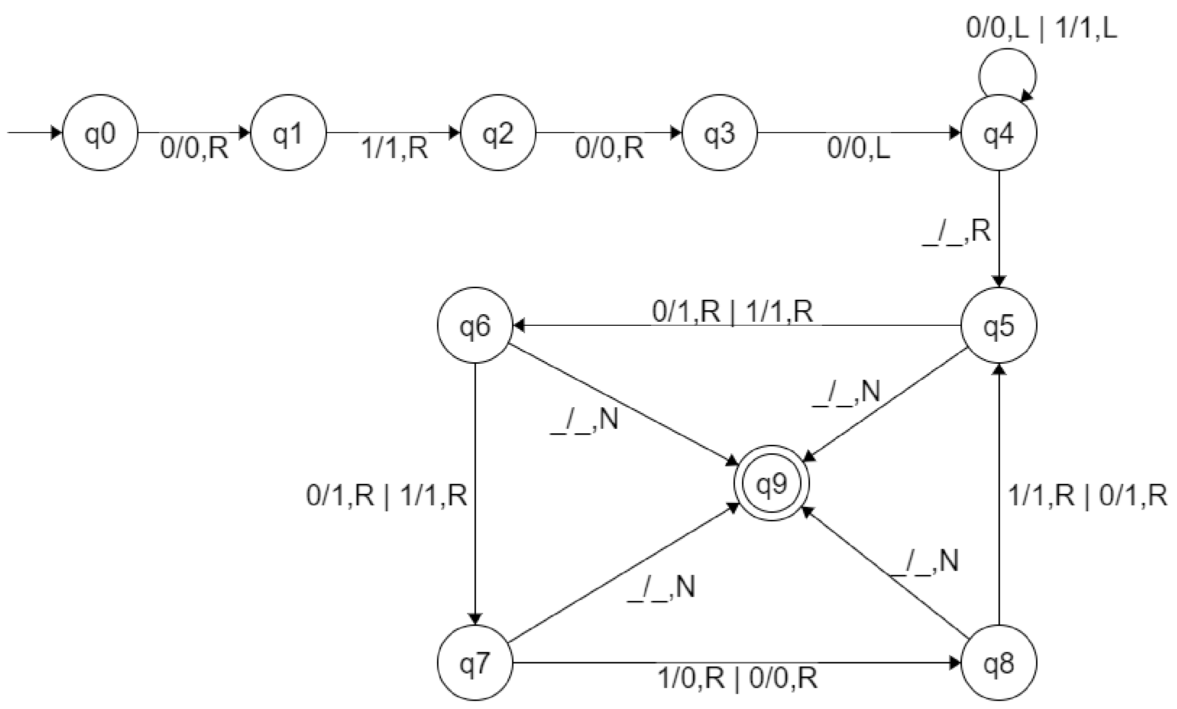
Construct a TM M_9 which behaves as follows. The input to M_9 is a string w over {0,1}.

- If w starts with `0100`, then M_9 replaces the cells occupied by the input with the repeated pattern `1101`, that is, with `110111011101110...` repeating to as many symbols as the length of w, and then accepts.
- Otherwise, M_9 rejects the input without changing it.

Submit a representation of M_9 as described above.

Answer: (penalty regime: 25, 50, ... %)

Help | Clear



Precheck | Check

	Got
✓	Good

Passed all tests! ✓

Question **12**

Precheck results

Marked out of 8.00

Consider the following productions (where as usual upper-case letters are non-terminals and lower-case letters are terminals). For each production independently, in the given order, state the most specific type of grammar it can appear in.

1. `Saa = Na`
2. `E = aMaF`
3. `aRM = YbS`
4. `Q = TS`
5. `S = aEb`
6. `T = bS`
7. `RR = b`
8. `SF = Sab`

If a production can appear in a type-X grammar, the answer for it is X. If the same production can appear in grammars of different types, you must give the most specific type.

Your answer for each production must be the single character `0` or `1` or `2` or `3`. If you do not know the answer for some productions, leave the corresponding boxes empty.

Make sure your answers pass the Precheck. There is no Check since this question will be marked later. Correct answers will count positive. **Wrong answers will count negative.** Empty boxes will not count either positive or negative. The total marks will not be negative.

Precheck tests whether your answers are empty, `0`, `1`, `2` or `3`.

Answer: (penalty regime: 0 %)

```
production 1: type- 0
production 2: type- 2
production 3: type- 0
production 4: type- 2
production 5: type- 2
production 6: type- 3
production 7: type- 0
production 8: type- 1
```

Precheck

Precheck only

	Got
✓	Good

Question **13**

Answer saved

Marked out of 6.00

Consider the following statements. For each of them, state if it is true or false.

1. For every regular expression A there is some NFA with ε-transitions B such that L(A) = L(B).
2. For every regular expression A there is some TM B such that L(A) = L(B).
3. For every type-1 grammar A there is some DFA B such that L(A) = L(B).
4. For every DFA A there is some recursively enumerable language B such that L(A) = B.
5. For every deterministic PDA A there is some generalised NFA B such that L(A) = L(B).
6. For every recursively enumerable language A there is some type-1 grammar B such that A = L(B).

Your answer for each statement must be either `true` or `false`. If you do not know the answer for some statements, leave the corresponding boxes empty.

Make sure your answers pass the Precheck. There is no Check since this question will be marked later. Correct answers will count positive. **Wrong answers will count negative.** Empty boxes will not count either positive or negative. The total marks will not be negative.

Precheck tests whether your answers are empty, `true` or `false`.

Answer: (penalty regime: 0 %)

statement 1:	true
statement 2:	false
statement 3:	false
statement 4:	false
statement 5:	false
statement 6:	false

Precheck

Precheck only

	Got
✓	Good

◄ Mid-Semester Test 2022

Jump to...