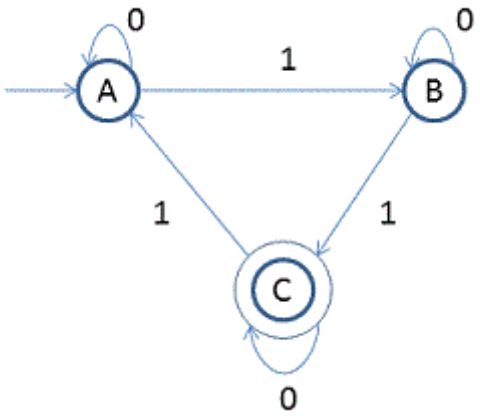
Final

Deterministic Finite Automata

The questions in this group refer to the following DFA:



We suggest you first examine the automaton and determine intuitively what language it accepts.

Answer

Let's construct the language that is accepted by the automaton, as per lectures we have a starting state `A` and our (single) end state is `C` and thus we need to construct the arc R `k`-paths for k=1...3 for `AC` and `CC` because `AC` is to get from the starting state to the end state and `CC` is to get from `C` to `C` again.

1. k = 0, $R_{AC} = \emptyset$, as there is no way to go with 0 hop from `A` → `C`
2. k = 1, $R_{AC} = \emptyset$, as before there is no way to get with 1 hop from `A` → `C`
3. k = 2, $R_{AC} = 0^+10^+1$, to get with 2 hops from `A` → `C`
4. k = 3, $R_{CC} = (0^+ + 10^+10^+1)$, to get with 3 hops from `A` → `C`

Now the final language

L is $R_{AC}\ R_{CC} = (0^+10^+1)(0^+ + 10^+10^+1)$

Part A

Which of the following strings is accepted by this DFA? In my instance I had the following options:

1. `1011`
2. `01110110`
3. `0010101001`
4. `ϵ`

Based L we generated, the correct answer is **option 3**.

Part B

How many strings of length 4 are accepted? In my instance I had the following options:

1. 0
2. 2
3. 4
4. 6

Based on L we generated, the correct answer is **option 4**.

Part C

Which of the following regular expressions takes the automaton from state `A` to state `B` without passing through state `C`?

1. 0^*1
2. $0^*(0+1)0^*$
3. 0^*10^*
4. $0^*1^*0^*$

Based on the automaton we have, we can easily see that the correct answer is **option 3**.

Part D

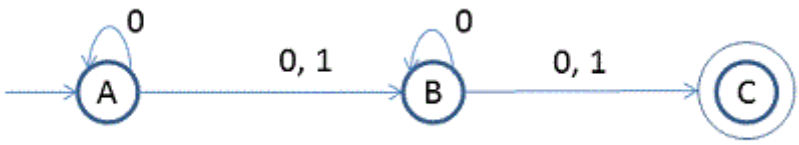
Which of the following regular expressions generates the same language as the automaton accepts?

1. $0^*10^*10^*(10^*10^*10^*)^*$
2. $(1^*0^*10^*10^*)^*$
3. $0^*10^*1(0^*10^*10^*1)^*$
4. $(0^*1)^*(0^*10^*10^*10^*)^*$

Based on the language L and the automaton above we can see that the correct answer is **option 1**.

Nondeterministic Finite Automata

The questions in this group refer to the following DFA:



We suggest you first examine the automaton and determine intuitively what language it accepts.

Answer

Part A

Which of the following strings is NOT accepted by this NFA?

1. `00`
2. `10`
3. `110`
4. `0101`

We can easily see by plugging each of the solutions to the automaton that the string that is not accepted by the NFA is **option 3**.

Part B

How many paths labeled 001 are there?

1. 1
2. 2
3. 3
4. 4

We can easily see that the number of paths `001` is 3 thus the correct answer is **option 3**.

Part C

Which of the following regular expressions DOES NOT generate the same language as the automaton accepts?

1. $0^*(0+1)0^*(0+1)$
2. $(0+1)(0+1)0^*$
3. $0^*(0+10^*)(0+1)$
4. $(0+1)00^*+00^*10^*+0^*10^*1$

We can easily see from the automaton that the one which fails to even be parsed by it is $(0+1)(0+1)0^*$
and thus the correct answer is **option 2**.

Regular and Context-Free Languages

Classify each of the following languages as either

1. regular
2. context-free but not regular, or
3. not context free

All languages are over the alphabet `{0, 1, 2}`.

Answer

Part A

The set of strings that, when treated as a base-3 number, is evenly divisible by 7.

1. Regular
2. Context-free but not regular
3. Not context free

The correct answer is **option 1** as the language is regular.

Part B

The set of strings that, when treated as a base-3 number, is a prime.

1. Regular
2. Context-free but not regular
3. Not context free

The correct answer is **option 3** as when applying the pumping lemma we can deduce that the language not context free.

Part C

The set of strings with an equal number of 0's, 1's, and 2's.

1. Regular
2. Context-free but not regular
3. Not context free

The correct answer is **option 3** as when applying the pumping lemma we can see that the language is not context free.

Part D

The set of strings with more 0's than 1's (and any number of 2's).

1. Regular
2. Context-free but not regular
3. Not context free

The correct answer is **option 2** as the language is context-free yet not regular.

Part E

The set of strings in which every 0 and every 1 is immediately followed by a 2.

1. Regular
2. Context-free but not regular
3. Not context free

The correct answer is **option 1** as the language is indeed regular.

Closure Properties

For each of the operations below, check all and only the classes of languages that are closed under that operation. Note: to get credit for a part, you must get ALL the classes for that part correct.

Answer

Part A

Reversal with respect to the following languages:

- The regular languages
- The context-free languages
- The recursive languages
- The recursively enumerable languages

The reversal property, as per lectures, is closed for *all* the above languages.

Part B

Intersection with respect to the following languages:

- The regular languages
- The context-free languages
- The recursive languages
- The recursively enumerable languages

The intersection property, as per lectures, is closed for regular, recursive, and recursively enumerate languages but **not** for context-free.

Part C

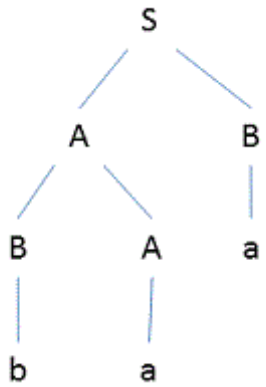
he operation that takes each string in the language and deletes all the odd-numbered positions. For example, if L = {0123, 01234, 000000, 11111} then the result of the operation is {13, 000, 11}.

- The regular languages
- The context-free languages

This operation, as per lectures, holds only for the regular and **not** for the context free languages.

Context-Free Grammars

Given the following parse tree T that parses a context-free grammar G - answer the following questions for the
parse tree T and grammar G.



Answer

Part A

How many interior nodes are in T?

1. 2
2. 3
3. 4
4. 5

This easily deduced from the parse tree as the number of *interior* nodes are 5, hence the correct answer is
option 4. Recall that an interior node is a non-terminal category of the parsed grammar of which we have 5 in the
provided tree.

Part B

Which of the following strings is DEFINITELY in the language L(G)?

1. abab
2. baba
3. baabb
4. ε

We can easily see that the correct answer is **option 1**, but before we validate it let's create the transition
rules for the tree T and grammar G:

- S → AB
- A → BA | a

- B → b | a

Now let's us expand **option 1**: S → AB → BAB → aBAB → abAB → abaB → abab which is our answer. In our derivation
we assume that in this instance we use left-most expansion.

Part C

Which of the following sentential forms is NEITHER a sentential form in the leftmost derivation corresponding to T NOR a sentential form in the rightmost derivation corresponding to T?

1. baB
2. Baa
3. aAB
4. Aa

Recall than sentential form is any string that can be derived from the start symbol which includes the non-terminals
that happen in intermediate steps as well. There are *left* and *right* sentential forms which are used in leftmost and
rightmost derivations respectively.
Based on the above definition of sentential forms we can easily see that `aAB` is neither a sentential form
in leftmost and rightmost derivations with respect to T.

CYK Algorithm

When we apply the CYK algorithm to the grammar

- `S` -> `AB`
- `A` -> `BB` | `a`
- `B` -> `AA` | `b`

and the input `babba`,
What is the value of X_{24}, that is, the set of variables that derive the string from positions
2 to 4 of the input? (Check all and only the members of X_{24}).

Answer

The full CYK table for the aforementioned string is shown below

`X_{15} = {S,A,B}`				
`X_{14} = {A}`	`X_{25} = {A,B}`			
`X_{13} = {}`	`X_{24} = {B}`	`X_{35} = {B}`		
`X_{12} = {}`	`X_{23} = {S}`	`X_{34} = {A}`	`X_{45} = {}`	
`X_{11} = {B}`	`X_{22} = {A}`	`X_{33} = {B}`	`X_{44} = {B}`	`X_{55} = {A}`

We can easily see that the correct answer to the posed question is X_{24} = {B}.

Grammar Simplification

The following questions are based on the following grammar `G`:

`S` -> `AB` | `b`

`A` -> `aA` | `ϵ`

`B` -> `CD`

`C` -> `bA` | `AA`

`D` -> `AB` | `BC`

For each question, to get credit you must check all and only the symbols that have the stated property.

Answer

Let's start by performing a grammar simplification on `G`

Part A

Given `G`, check all and only the nullable symbols.

1. `S`
2. `A`
3. `B`
4. `C`
5. `D`

Recall that a *nullable* symbol is the one that holds `A` ->* `ϵ`, which in our case is it easy to see that

1. `A` -> `ϵ`
2. `C` -> `bA` | `AA` ->* `ϵ`

This the nullable symbols are `A` and `C`.

For brevity, the elimination of `ϵ`-productions for `G` provided above forming `G'` is:

`S` -> `AB` | `B` | `b`

`A` -> `aA` | `a`

`B` -> `CD`

`C` -> `bA` | `b` | `AA` | `A`

`D` -> `AB` | `B` | `BC`

Strategy: we see that `A` -> `ϵ`, thus we go and replace all instances of `A` with `ϵ` to create the new states,

these are then merged - as shown above - into our previous states of `G` to form a non-`ϵ` version of `G`.

Based on the above, the correct answer are **option 2** and **option 4**

Part B

Given `G`, check all and only the variables that derive a terminal string.

1. `S`
2. `A`
3. `B`
4. `C`
5. `D`

We can easily see that in grammar `G` provided above the symbols that derive a terminal string are the following:

1. `S` -> `b` (directly)
2. `A` -> `ϵ` (directly)
3. `C` -> `bA` | `AA` ->* `ϵ` (transitively)

Thus the correct options are **option 1, 2, 4**

Part C

Now, modify the grammar to eliminate all productions that involve one or more symbols that do not derive a terminal string. Given the new grammar `G'`, check all and only the variables that appear in some sentential form derived from S using this modified grammar.

1. `S`
2. `A`
3. `B`
4. `C`
5. `D`

Recall from our grammar `G'` found previously (and the original `G` for that matter), the only symbols that
are useful to keep are only those that derive *terminal* strings. We can see that in the `ϵ`-free grammar `G'`
that these symbols are `S` and `A`.
Moreover, we now have to eliminate the unit productions which are

1. `S` -> `B`
2. `C` -> `A`
3. `D` -> `B`

The simplified grammar is as follows:
`S` -> `AB` | `CD` | `b`
`A` -> `aA` | `a`
`B` -> `CD`
`C` -> `bA` | `b` | `AA` | `aA` | `a`
`D` -> `AB` | `CD` | `BC`
Now, we have to eliminate the useless productions and to do that we need to first find a set with the terminal
symbols as such:

- {`a`, `b`}

Then, we need to find the reachable states for these terminal symbols, namely `S`, `A`, and `C` which are then
added to the above set as shown below

- {`a`, `b`, `S`, `A`, `C`}

We can see that no other other states can be added and thus `B` and `D` are *redundant*; hence, all states and
transitions involving these have to be removed - the resulting grammar is only
`S` -> `b`
Which is also the correct answer to this question, as it is already in sentential form. Thus, the correct
answer is **option 1** (and only that).

Homomorphisms

Let h be the homomorphism defined by

- h(a) = 01 and
- h(b) = 1.

Now let L1 = h({aba, bab}), that is, h applied to the language consisting of two strings, aba and bab.

Also let L2 = h^{-1}({010, 101}).

Answer

Part A

How many string are in L1?
We can easily see that there are only two (2) strings in L1 - this is because both strings can be mapped to L1
using h, which results into two strings; namely: h(aba) → 01101, h(bab) → 1011

Part B

How many strings are in L2?
And in a similar fashion we can see that there is only one (1) string in L2. In this instance only one string can be
mapped to L2 using h^{-1}, namely: h^{-1}(101) → ba as 01101 cannot be mapped using h^{-1}.

Turing Machines

The Turing machine M has states `q`, `p`, and `f`. State `q` is the start state,
and `f` is the accepting state. The input alphabet is `{0,1}` and the tape alphabet
is `{0,1,B}`; `B` is the blank. The transitions of `M` are:
`δ(q,0) = (p,0,R)`
`δ(q,1) = (p,0,R)`
`δ(p,0) = (p,0,R)`
`δ(p,1) = (q,1,L)`
`δ(p,B) = (f,0,R)`
Thus the state transition table for `M` is the following

State	Tape Symbol	Move
`q`	`0`	`(p, 0, R)`
`q`	`1`	`(p, 0, R)`
`p`	`0`	`(p, 0, R)`
`p`	`1`	`(p, 0, L)`
`p`	`B`	`(f, 0, R)`

Answer

Part A

What happens when given to `M` as described above input `w1` = `100`?

1. Accepts.
2. Halts without accepting.
3. Does not halt.

Expanding of `w1`

1. `q100`, found `1` on tape thus `q` → `(p, 0, R)`
2. `1p00`, found `0` on tape thus `p` → `(p, 0, R)`
3. `10p0`, found `0` on tape thus `p` → `(p, 0, R)`
4. `100p`, found `0` on tape thus `p` → `(p, 0, R)`
5. `100p`, found `B` on tape thus `p` → `(f, 0, R)` and thus we **accept**

Part B

What happens when given to `M` as described above input `w2` = `001`?

1. Accepts.
2. Halts without accepting.
3. Does not halt.

Expanding of `w2`

1. `q001`, found `0` on tape thus `q` → `(p, 0, R)`
2. `0p01`, found `0` on tape thus `p` → `(p, 0, R)`
3. `00p1`, found `0` on tape thus `p` → `(p, 0, R)`
4. `00p1`, found `1` on tape thus `p` → `(p, 0, L)`
5. `0p01`, found `0` on tape thus `p` → `(p, 0, R)`

Notice that in this instance, the Turing machine will loop between states 4 and 5 indefinitely, which means that we will **not halt**.

Part C

What happens when given to `M` as described above input `w3` = `ϵ`?

1. Accepts.
2. Halts without accepting.
3. Does not halt.

This case is quite easy - as there is no transition using the empty string at the start as we can only transition when we see an empty character at the tape from state `p` and not from state `q`. Thus this fact will cause the machine to halt without accepting; hence, the correct answer is **option 2**.

Rice's Theorem

Check all and only those of the following languages that are recursive (decidable).

1. The set of Turing-machine codes for Turing machines with more states than tape symbols
2. The set of Turing-machine codes for Turing machines that accept all strings over their input alphabet
3. The set of Turing machine codes for Turing machines that both accept and reject the same input.

Answer

From the lectures, we know that the only two of the above that are are decidable - namely:

- The set of Turing-machine codes for Turing machines with more states than tape symbols
- The set of Turing machine codes for Turing machines that both accept and reject the same input.

Polynomial-Time Reductions

In the following, we use comp(L) to represent the complement of the language L. We are given the following facts:

1. There is a polynomial-time reduction of L1 to L2.
2. There is a polynomial-time reduction of comp(L2) to L3
3. L3 is in NP.

Given what we know about the relationships among P, NP, and co-NP, which of the following is DEFINITELY true?

Answer

In my instance I had the following options to select from:

1. L2 is in NP.
2. comp(L2) is in co-NP.
3. L1 is not in P.
4. comp(L1) is in NP.

Out of the options provided only **option 4** is correct since we know that L3 is in NP. We also know that languages
are closed with respect to complementation under P but we do not know if that holds under NP. Since there is a
polynomial-time reduction from comp(L2) to L3 we know that comp(L2) will be in NP. Thus, although we do not know
if L1 or L2 are indeed in NP due to the polynomial reduction of comp(L2) to L3 we can infer that comp(L1) will be
in NP as well.

Post's Correspondence Problem (PSP)

Given the following instance of the Post's Correspondence Problem (PSP)

List A	List B
011	01
110	10
01	1011

What is the length of the shortest solution to this instance of PCP?
Note: we are asking for the number of occurrences of pairs, not the length of the string that is formed by concatenating the first or second components of the pairs.

Answer

The correct answer is 3 - that is, we use each block of each list *once*; an example solution that uses each block once is shown below

	0	1	3
List A	01	011	110
List B	01	0111	10

Satisfiability

The following expression: `(x + y)(¬x + ¬z)` is in SAT. (Note: `¬` stands for logical negation.)
How many satisfying assignments does this expression have?

Answer

This has 4 individual satisfying assignments, namely:

- x = 1, y = 0, z = 0
- x = 1, y = 1, z = 0
- x = 0, y = 1, z = 0
- x = 0, y = 1, z = 1

Intractable Problems

For the purpose of this question, assume it is known that P is not equal to NP, but NP is equal to co-NP.
For each problem below, check all and only the classes to which the problem would then be known to belong.

Answer

Part A

2SAT (given a Boolean expression that is the AND of factors, each factor being the OR of two literals,
is this expression satisfiable?):

1. P
2. NP
3. co-NP

As per lectures this would be in all of the above, namely: P, NP, and co-NP.

Part B

The Tautology problem (given a Boolean expression, is it true for all possible truth assignments?):

1. P
2. NP
3. co-NP

As per lectures this would be in NP and co-NP - concretely since it is known that the boolean satisfiability
problem to be NP complete then consequently the tautology problem is in NP and specifically co-NP
(check here).

Part C

The 3-Knapsack problem (given a set of integers, can it be partitioned into THREE disjoint sets such
that the sum of the members of each set is the same?):

1. P
2. NP
3. co-NP

As per lectures, we can easily see that this problem is in NP and co-NP