1. Finite automata and regular languages:

1a. Design a finite automaton for recognizing binary sequences that have odd number of 0s and odd number of 1s. Assume that the input strings contain only symbols 0 and 1. The easiest is to have 4 states:

- the desired (final) state oo in which we have odd number of 0s and odd number of 1s,
- the state eo, in which we have even number of 0s and odd number of 1s, and
- similarly defined states oe (odd-even) and ee (even-even).

You just need to describe transitions between these states, and which states are final. Show, step-by-step, how your automaton will accept the string 1000.

1b. On the example of this automaton, show how the word 1000 can be represented as xyz in accordance with the pumping lemma.

1c. Use a general algorithm to describe a regular expression corresponding to the finite automaton from the Problem 1a. (If you are running out of time, it is Ok not to finish, just eliminate the first state.)

1d-e. Some of the words from the corresponding language (but not all of them) The resulting language can be described by a regular expression $1^*(00)^*$. Use a general algorithm to transform this regular expression into a finite automaton: first a non-deterministic one, then a deterministic one.
\[ R_{ns,oe} = R_{ns,oe} \cup (R_{ns,ee} \cup R_{ee,ee} \cup R_{ee,oe}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{ns,oo} = R_{ns,oo} \cup (R_{ns,ee} \cup R_{ee,ee} \cup R_{ee,oo}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{ns,co} = R_{ns,co} \cup (R_{ns,ee} \cup R_{ee,ee} \cup R_{ee,co}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{ns,nf} = R_{ns,nf} \cup (R_{ns,ee} \cup R_{ee,ee} \cup R_{ee,nf}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oe,oe} = R_{oe,oe} \cup (R_{oe,ee} \cup R_{ee,ee} \cup R_{ee,oe}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oe,oo} = R_{oe,oo} \cup (R_{oe,ee} \cup R_{ee,ee} \cup R_{ee,oo}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oe,co} = R_{oe,co} \cup (R_{oe,ee} \cup R_{ee,ee} \cup R_{ee,co}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oe,oe} = R_{oe,oe} \cup (R_{oe,ee} \cup R_{ee,ee} \cup R_{ee,oe}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oe,nf} = R_{oe,nf} \cup (R_{oe,ee} \cup R_{ee,ee} \cup R_{ee,nf}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oo,oe} = R_{oo,oe} \cup (R_{oo,ee} \cup R_{ee,ee} \cup R_{ee,oo}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oo,oo} = R_{oo,oo} \cup (R_{oo,ee} \cup R_{ee,ee} \cup R_{ee,oo}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oo,co} = R_{oo,co} \cup (R_{oo,ee} \cup R_{ee,ee} \cup R_{ee,co}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oo,oe} = R_{oo,oe} \cup (R_{oo,ee} \cup R_{ee,ee} \cup R_{ee,oe}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{oo,nf} = R_{oo,nf} \cup (R_{oo,ee} \cup R_{ee,ee} \cup R_{ee,nf}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{ko,oe} = R_{ko,oe} \cup (R_{ko,ee} \cup R_{ee,ee} \cup R_{ee,oe}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{ko,oo} = R_{ko,oo} \cup (R_{ko,ee} \cup R_{ee,ee} \cup R_{ee,oo}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{ko,co} = R_{ko,co} \cup (R_{ko,ee} \cup R_{ee,ee} \cup R_{ee,co}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{ko,oe} = R_{ko,oe} \cup (R_{ko,ee} \cup R_{ee,ee} \cup R_{ee,oe}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]

\[ R_{ko,nf} = R_{ko,nf} \cup (R_{ko,ee} \cup R_{ee,ee} \cup R_{ee,nf}) \]
\[ \emptyset \cup (\emptyset \cup \emptyset^* \emptyset) = \emptyset \]
1. c) non-deterministic

\[
\text{1}^* (\text{00})^* \]

\[
\begin{align*}
\epsilon & \quad 1 \quad \epsilon \\
0 & \quad \rightarrow & \quad 0 \\
0 & \quad \rightarrow & \quad 0 \\
\end{align*}
\]

\[
\text{1}^* \quad \rightarrow \quad \text{00}^* \quad \epsilon \\
\begin{align*}
\epsilon & \quad \rightarrow & \quad \epsilon & \quad \rightarrow & \quad 0 \\
0 & \quad \rightarrow & \quad 0 & \quad \rightarrow & \quad 0 \\
\end{align*}
\]

\[
\text{1}^* (\text{00})^* \quad \rightarrow \quad \text{1}^* \quad \rightarrow \quad \text{00}^* \quad \epsilon \\
\begin{align*}
\epsilon & \quad 1 \quad \epsilon & \quad \epsilon & \quad \epsilon & \quad 0 \\
0 & \quad \rightarrow & \quad 0 & \quad \rightarrow & \quad 0 \\
\end{align*}
\]

1e. deterministic

\[
\begin{align*}
1, 2, 4, 5 & \quad \rightarrow \quad 2, 3, 4, 5 \\
\omega, 7 & \quad \rightarrow \quad 5, 8 \\
\end{align*}
\]
2. Beyond finite automata: pushdown automata and context-free grammars:

2a. A team is above average if in the championship, it has more wins (W) than losses (L). For example, a team with a record W:L:W is above average. Prove that the language consisting of all sequences of W and L that make the team above average is not regular.

2b. Use a general algorithm to transform the finite automaton from the Problem 1a into a context-free grammar (CFG). Show, step-by-step, how this CFG will generate the word 1000.

2c. For the context-free grammar from the Problem 2b, show how the word 1000 can be represented as uvxyz in accordance with the pumping lemma.

2d. Use a general algorithm to translate the CFG from 2b into Chomsky normal form.

2e. Use a general algorithm to translate the CFG from 2b into an appropriate push-down automaton. Explain, step-by-step, how this automaton will accept the word 1000.

2f. Use the general stack-based algorithms to show:
   - how the compiler will transform a Java expression 8 / 4 / 2 into inverse Polish notation, and
   - how it will compute the value of this expression.

2a. Proof by contradiction. Let's assume that L is regular and let's derive a contradiction from this assumption. Since L is regular, by pumping lemma, there exists an integer p such that every word from L whose length ≥ p can be represented by xyz, where len(y) > 0, len(xy) ≤ p and for all i, xy^iz ∈ L.

Let's take w = L^p W^{p+1} = L...LW...W, length(xy) ≤ p.

We xyz starts with xy so x's is in L's. In xy^2z = we add x's but not w's. So xy^2z has the same or more than w's, so xy^2z is not an element or language L. But by pumping lemma xy^2z ∈ L. We got a contradiction, so our assumption that L is regular is false, L is not regular.
Z.b.

A is ee
B is oc
C is eo
D is oo

1000

A

C

D

O

2C.

A

C

D

Z

Y

X


uvx y yz = 100000

file://Q:/cs3350.18a/final.html
2d. A → 0B  
    A → 1C  
    B → 0A  
    B → 1D  
    C → 0D  
    C → 1A  
    D → 0C  
    D → 1B  
    D → E

    Pre: S_0 → A

    Step 1: B → 1  C → 0  
    S_0 → OB  S_0 → 1C

    Step 2: V_0 → 0  V_1 → 1  
    A → V_0 B  A → V_1 C  
    B → V_0 A  B → V_1 D  
    C → V_0 D  C → V_1 C  
    D → V_0 B  D → V_1 B  
    S_0 → V_0 B  S_0 → V_1 C  
    B → 1  C → 0
2c. $S \rightarrow \epsilon, \epsilon \rightarrow \$ \quad E, \epsilon, \rightarrow A
\quad A \rightarrow 0B
\quad A \rightarrow 1C
\quad B \rightarrow 0A
\quad B \rightarrow 1D
\quad C \rightarrow 0D
\quad C \rightarrow 1A
\quad D \rightarrow 0C
\quad D \rightarrow 1B
\quad D \rightarrow \epsilon

W O R K I N G

1000
1 \quad s \rightarrow aux \quad 1 \quad w \quad 1 \quad 1 \quad 1 \quad 1 \quad s \quad w \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 1
\quad f
\quad s
\quad aux
\quad w
\quad 1
\quad 1
\quad 1
\quad w
\quad s
\quad w
\quad 1
\quad 0
\quad 0
\quad 1
\quad 0
\quad 0
\quad 1
\quad 0
\quad 0
\quad 1
\quad f

2d. $8/4/2$

\[
\begin{array}{c|c|c|c}
8 & 4 & 2 \\
\hline
1 & 1 & 1
\end{array}
\]

\[
\frac{8}{4/2} \rightarrow \frac{8}{4/2/2} \rightarrow \frac{8}{4/2/2} = 1
\]
3. Beyond pushdown automata: Turing machines

3a. To celebrate Cinco de Mayo, students decided to decorate the walls a classroom in green (G), white (W), and red (R), colors of the Mexican flag. To get a good image, they want to make sure that there are exactly the same number of segments of each color. So, if the walls are decorated in GGWRWR, it is good, but if it is GGWR, it is not good. Prove that the set of all "good" color combinations is not context-free and therefore, cannot be recognized by a pushdown automaton.

3b-c. Use a general algorithm to design a Turing machine that accepts exactly all sequences accepted by a finite automaton from Problem 1a. Show, step-by-step, how this Turing machine will accept the word 1000. Describe, for each step, how the state of the tape can be represented in terms of states of two stacks.

3d-e. Design Turing machines for computing $a - 1$ in unary and in binary codes. Trace both for $a = 3$, i.e., for $a = 111$ in unary code and $a = 11$ in binary code.

3a. By contradiction, let's assume $L$ is CFL. Then by pumping

**Lemma.** $\exists p \forall w \in L (\text{length}(w) \geq p \rightarrow \exists uvxyz : (w = uvxyz \text{ and } \text{length}(vy) \leq p \text{ and } \forall i (uv^nzy)^i))$

Let's take a word $w = G_p w_p R_p$ so that its length is $3p \geq p$.

Let $w \in L$. We can represent it as $uvxyz = y G_1 ... G_p W ... W R ... R$

The fragment $vxy$ cannot contain $G$'s and $w$'s and $R$'s because its length would be greater than $p$.

We have 5 options,

1. $vxy$ is in $G$'s
2. $vxy$ is in $G$'s and $W$'s
3. $vxy$ is in $W$'s
4. $vxy$ is in $W$'s and $R$'s
5. $vxy$ is in $R$'s

In case 1 we add $G$'s but not $W$'s or $R$'s, so in $uv^2xy^2z$, we have more $G$'s than $W$'s and $R$'s, so $uv^2xy^2z \notin L$.

In case 2 we add $G$'s and $w$'s but not $R$'s, so in $uv^2xy^2z$, we have more $G$'s and $W$'s than $R$'s, so $uv^2xy^2z \notin L$.

Very similar in case 3, 4 and 5...

But by pumping lemma $uv^2xy^2z \notin L$, so we have a contradiction.

A our assumption is false, $L$ is not CFL.
3b - C.

\[ \text{Stacks} \]

\[ \text{1000} \]

1. \[ \text{1000} \]

2. \[ \text{1000} \]

3. \[ \text{1000} \]

4. \[ \text{1000} \]

5. \[ \text{1000} \]

6. \[ \text{1000} \]

- \( \text{st}, \text{1} \rightarrow \text{ee}, \text{R} \)
- \( \text{ee}, \text{0} \rightarrow \text{oe}, \text{R} \)
- \( \text{ee}, \text{1} \rightarrow \text{eo}, \text{R} \)
- \( \text{oe}, \text{0} \rightarrow \text{ee}, \text{R} \)
- \( \text{oe}, \text{1} \rightarrow \text{eo}, \text{R} \)
- \( \text{eo}, \text{0} \rightarrow \text{oo}, \text{R} \)
- \( \text{eo}, \text{1} \rightarrow \text{ee}, \text{R} \)
- \( \text{oo}, \text{0} \rightarrow \text{eo}, \text{R} \)
- \( \text{oo}, \text{1} \rightarrow \text{oe}, \text{R} \)

- \( \text{ee}, \text{1} \rightarrow \text{reject} \)
- \( \text{oe}, \text{1} \rightarrow \text{reject} \)
- \( \text{eo}, \text{1} \rightarrow \text{reject} \)
- \( \text{oo}, \text{1} \rightarrow \text{accept} \)
3d-e. a - 1

**Unary**

- Start, \(\downarrow \rightarrow R, \text{work} \)
- work, 1 \(\rightarrow R \)
- work, 1 \(\rightarrow L, \text{ready to delete} \)
- ready to delete, 1 \(\rightarrow \downarrow, \text{back, L} \)
- back, 1 \(\rightarrow L \)
- back, \(\downarrow \rightarrow \text{halt} \)

**Binary**

- Start, \(\downarrow \rightarrow \text{working, R} \)
- working, 0 \(\rightarrow 1, R \)
- working, 1 \(\rightarrow 0, \text{back, L} \)
- back, 1 \(\rightarrow L \)
- back, \(\downarrow \rightarrow \text{halt} \)

\[ a = 11 = 3 \]

\[ 2^0 = 10_2 = 2 \]
4. Beyond Turing machines: computability

4a. Formulate Church-Turing thesis. Is it a mathematical theorem? Is it a statement about the physical world?

4b. Prove that the halting problem is not algorithmically solvable.

4c. Not all algorithms are feasible, but, unfortunately, we do not have a perfect definition is feasibility. Give a current formal definition of feasibility and give two examples:
   - an example of an algorithm's running time which is feasible according to the current definition but not practically feasible, and
   - an example of an algorithm's running time which is practically feasible but not feasible according to the current definition.

4d. Briefly describe what is P, what is NP, what is NP-hard, and what is NP-complete. Is P equal to NP?

4e. Give an example of an NP-complete problem: what is given, and what we want to find.

4f. Give definitions of a recursive (decidable) language and of a recursively enumerable (Turing-recognizable) language.

4a. Church–Turing Thesis: Anything that can be computed on any physical device can also be computed on a Turing machine. It is not a mathematical theorem. It is a statement about the physical world.
4c. Feasibility = polynomial time
Definition: An algorithm \( A \) is called feasible if there exists a polynomial \( p \) such that for every input \( x \) of length \( n \), \( T_A(x) \leq p(n) \). (Time needed to solve this polynomial is less than \( p(n) \)).

* \( \exp (10^{-90} n) \) - feasible but not practically feasible
* \( 10^{90} n \) - practically feasible but not feasible

4d.

P-class or problems solvable in polynomial time

NP-class or all problems in which, once you have a candidate for a solution, you can check feasibly whether this candidate is a solution

NP-Hard: every problem from the class NP can be reduced to this problem

NP-complete: a problem is NP-complete if it is in NP and every problem from NP can be reduced to this problem

* It is unknown if \( P = NP \).

4e. Hamiltonian Path: given: input graph
find: whether input graphs contains a path \( s \) to \( t \) that goes through every node exactly once.
4f.
- **recursive decidable:** a language \( A \) is recursive decidable if there is an algorithm that given a natural number \( n \), decides whether \( n \in A \) through boolean outputs.

- **recursively enumerable:** a language set is recursively enumerable if there is an algorithm that eventually prints all elements with the use of a turing machine.