York University
CSE 2001 Fall 2017 – Midterm A
Instructor: Jeff Edmonds

1. Consider the alphabet $\Sigma = \{a, b\}$. Prove or disprove the following:

   (a) $bbabbaab \in \Sigma^* a \Sigma^* b \Sigma^* a \Sigma^*$.

   - Answer: The following shows that the string IS in the language.

Σ^*	a	Σ^*	b	Σ^*	a	Σ^*
bb	a	ϵ	b	b	a	ab

   (b) $baab \in \Sigma^* a \Sigma^* b \Sigma^* a \Sigma^*$.

   - Answer: It is NOT in the language. To be in the language the string must contain two a characters somewhere in the string that have a b somewhere between them. Here the string has only two a's and they do not have a b between them.

2. Short Answers: What is the 'N' in NFA for? How is it's Transition Function δ different than that of an DFA and what effect does this have on its computation? Given an NFA M and a string α, how do you know whether or not the string is accepted by the machine.

   - Answer: The 'N' in NFA is for "Nondeterministic", which means that what the machine does next is not set in stone, i.e. not determined, but there is a choice. The transition function, $\delta(q_{current}, c) = \{q_{next}\}$, specifies is a set of states, one of which will be the next state. Jeff talks of a Fairy God Mother helping to know which way to go. M accepts α if and only if there is a path in M from the start state to one of the accept states with the labels along this path concatenating to give the string α.

3. First Order Logic: We state as follows that single-tape Turing machines can compute everything that multi-tape can.

   $\forall$ multi-tape TM M_{multi}, $\exists$ a single tape TM M_{single}, $\forall$ inputs I, $M_{multi}(I) = M_{single}(I)$.

   How is the *First Order Logic Game* game played for this statement?

   Explain how this game states whether M_{single} is likely to have more states or fewer than M_{multi}.

   - Answer: First Order Logic Game:
     The adversary gives us an arbitrary M_{multi}.
     We construct M_{single}
     (Knowing M_{multi} and hence M_{single} can be much bigger than M_{multi})
     The adversary gives us an arbitrary input I
     (Knowing M_{single} and hence I can be much bigger than M_{single})
     Prove $M_{multi}(I) = M_{single}(I)$.

4. **Turing Machine:** (25 marks) Write all the transition rules for a Turing Machine that solves the computational problem that takes as input a string of characters from $\{0, 1, ..., 9\}$ and replaces every 9 with the character that appeared two cells before it in the string. (If the first or second is a 9, replace it with a zero.)
   Input: 94398279293193
   Output: 04348272293133

   Pseudo Code:

```
0: Put head on first character.
   $c$ = getCharAtHead()
   $pp = 0$ and $prev = 0$
1: loop    ⟨loop−invariant⟩: The variables $pp$ and $prev \in \{0, ..., 9\}$ remember the previous
                two characters read and we are looking at a new character $c$.
```

```
    $pp = prev$      In preparation for shifting head update previous two characters.
    $prev = c$
    if($c = 9$)      then this character must be replaced by the previous previous character.
        write $pp$
    elseif( $c =$blank )
        Halt
    else
        Nothing to do
    Move right to next character
    $c =$ getCharAtHead()
end loop
```

(a) Translate this code into Turing Machine Transitions. Recall the states are named with the line number and the value of each variable.

(b) How many states does your TM have?

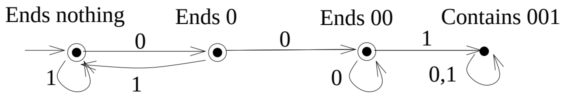
- Answer: The first thing to know is the form of the transition function is $\delta(q, c) = \langle q', c', direction\rangle$, i.e. when in current state is q and the character on the tape being seen by the head is c, then transition to state q', write c' onto the tape, and move the head in the direction indicated.
  The second thing to know is that the names of the states should indicate which line the pseudo code is in and what values are being saved on pooh bear's black board.

  (a) Turing Machine Transitions Start State: $q_{\langle line=1,pp=0,prev=0\rangle}$
  $\forall x, y \in \{0, 1, ..., 9\}$ and $\forall c \in \{0, 1, ..., 8\}$
  $$\delta(q_{\langle line=1,pp=x,prev=y\rangle}, 9) = \left\langle q_{\langle line=1,pp=y,prev=9\rangle}, x, right\right\rangle$$
  $$\delta(q_{\langle line=1,pp=x,prev=y\rangle}, blank) = \left\langle q_{\langle line=halt\rangle}, blank, stay\right\rangle$$
  $$\delta(q_{\langle line=1,pp=x,prev=y\rangle}, c) = \left\langle q_{\langle line=1,pp=y,prev=c\rangle}, c, right\right\rangle$$

  (b) It has $10 \times 10 + 1$ states counting the halt state.

5. DFA: Draw a DFA for the following language.
$L = \{\alpha \in \{0, 1\}^* \mid \alpha$ does NOT contain the substring 001$\}$.
Example: The strings 111$\underline{001}$00 contains 001 and hence in not in the language. The strings 111010100 does not so is.
Be sure to label the states of your DFA with names indicating what the DFA "knows" when in that state.

- Answer:



6. Bumping Lemma is stated as follows:
$[\exists$ infinite set S, $\forall \alpha, \beta \in S$ with $\alpha \neq \beta$, $\exists \zeta$, $L(\alpha\zeta) \neq L(\beta\zeta)] \Rightarrow [\forall$ DFA M, $\exists$ an input I, $M(I) \neq L(I)]$.

(a) No need for a full proof, but give reasonable intuition to why it is true.

- Answer:
  Answer from 2017: The conclusion $[\forall$ DFA M, $\exists$ an input I, $M(I) \neq L(I)]$ is that there is no DFA computing L.
  Break each string into a first and a last name. Here S is a set of first names with α and β being examples, while ζ is a last name. After a DFA M reads one of the first names in S, it must remember precisely which of these it has read. It does this by going into a

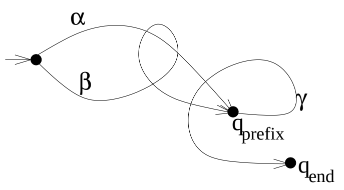
different state for each one. Hence, the number of states M has must be at least the size of $|S|$, which in this case is infinite.
If, on the other hand, M were confused to whether the first name it read was α or β, then it would also be confused as to whether the final string is $\alpha\zeta$ or $\beta\zeta$, where ζ is some last name. The statement of the claim is that the language L distinguishes between every pair $\alpha, \beta \in S$ of first names, namely that there is such a ζ such that the language L gives different answers to $\alpha\zeta$ and $\beta\zeta$. Hence, M will give the wrong answer to one of these.

Answer from 1999:

Theorem 1 *If language L distinguishes the prefixes S, then any DFA that decides L requires at least $|S|$ states.*
The intuition is that after the DFA has read in some prefix in S, it needs enough memory to remember which one it has seen or equivalently it needs enough states to distinguish between them.

Corollary 2 *If language L distinguishes the prefixes S, where S contains an infinite number of prefixes, then L is not a regular language, because any DFA deciding it requires an infinite number of states.*

Proof of Theorem ??:



By way of contradiction, assume that the DFA M decides L and has fewer than $|S|$ states. For each prefix $\alpha \in S$, run M on it and place it on the state of M at which the machine ends up. The *pigeon hole principle* states that you can't put $n+1$ pigeons into n pigeon holes without putting more than one pigeon in the same hole. By this principle, there is a state of M which we will denote q_{prefix} that receives two prefixes $\alpha, \beta \in S$. Because the prefixes α and β can be distinguished by L, there exists a postfix ζ such that $\alpha\zeta$ and $\beta\zeta$ are given different answers by L. I claim that the state q_{end} the M ends on when run on $\alpha\zeta$ is the same as when run on $\beta\zeta$. The reason is that after reading α and β, M is in the same state q_{prefix} and then it continues in the same way from there when reading ζ. This last state q_{end} is either an accept state or is not. Hence, it cannot give the correct answer on both input instances $\alpha\zeta$ and $\beta\zeta$, which have different answers. ■

(b) For each of these languages, either prove that it is not regular or prove that it is. Give the intuition.

i. $L = \{a,b\}^n\#\{u,v\}^{n \text{ or } (n+1)}$
ii. $L = \{a,b\}^m\#\{u,v\}^{n \text{ or } (n+1)}$

- Answer: The first is not regular because a DFA would need to count and remember the number of a or b in order to match it up with the number of u or v.

  We have to design the infinite set S of first names. An obvious set of first names would be $S = \{a,b\}^n$. But the problem with this is that $\alpha = a^3$ and $\beta = b^3$ are definitely not distinguished. It is better to have a smaller set of first names, say $S = a^n$.

  Concern: If $\alpha = a^3$, $\beta = a^4$, and $\zeta = u^4$, then both strings are yeses, namely $L(\alpha\zeta) = L(a^3u^{3+1}) = yes$ and $L(\beta\zeta) = L(a^4u^4 = yes$.

  Avoiding Concern:
  Let $S = \{a^{3n} \mid n \geq 0\}$.
  Note then α and β can't be one apart like in the concerning example.
  Let $\alpha = a^{3i}$ and $\beta = b^{3j}$ be arbitrary strings in S with $i \neq j$.

Let $\zeta = \#u^{3i}$.
Note $L(\alpha\zeta) = L(a^{3i}\#u^{3i}) = yes$, because we designed it this way.
We want to prove that $L(\beta\zeta) = L(a^{3j}\#u^{3i}) = no.$
In order for this to be a yes, because the number of a's is $3j$, the number of b's needed would be $3j$ or $3j+1$. But the number of b's we have is $3i$. Note that neither $3j$ or $3j+1$ is equal to $3i$ because i and j are different and are integers.
Hence, $L(\beta\zeta) = L(a^{3j}\#u^{3i}) = no.$
Hence S is an infinite set of strings distinguished by the language.
Hence no DFA computes L.

Address the Concern:
Let $S = \{a^n \mid n \geq 0\}$.
Let $\alpha = a^i$ and $\beta = b^j$ be arbitrary strings in S with $i \neq j$.
Assume without loss of generality that $i > j$. Let $\zeta = \#u^{i+1}$.
Note $L(\alpha\zeta) = L(a^i\#u^{i+1}) = yes$, because we designed it this way.
We want to prove that $L(\beta\zeta) = L(a^j\#u^{i+1}) = no.$
In order for this to be a yes, because the number of a's is j, the number of b's needed would be j or $j+1$. But the number of b's we have is $i+1$. Not that neither j or $j+1$ is equal to $i+1$ because $i > j$.
Hence, $L(\beta\zeta) = L(a^j\#u^{i+1}) = no.$
Hence S is an infinite set of strings distinguished by the language.
Hence no DFA computes L.

The second language $L = \{a,b\}^m\#\{u,v\}^{n \text{ or } (n+1)}$ is regular because the two blocks are not linked. A regular expression for it is $\{a,b\}^*\#\{u,v\}^*$.

7. For each of these languages, either give a context free grammar for it or argue that it can't be done.

   (a) $L = 0^{2n+7}\ a^m\{x,y\}^s 1^{2m}\{u,v\}^{3n+8}$

   (b) $L = 0^{2m+7}\ a^n\{x,y\}^s 1^{2m}\{u,v\}^{3n+8}$

   - Answer: The following is a CFG for the first.
     We give non-terminal names to the blocks as follows.
     $S = 0^{2n+7}\ a^m\{x,y\}^s 1^{2m}\{u,v\}^{3n+8}$
     $U = \{u,v\}$
     $T = a^m\{x,y\}^s 1^{2m}$
     $X = \{x,y\}^s$.
     The grammar rules are then as follows.
     $S \Rightarrow 00SUUU$
     $S \Rightarrow 0000000TUUUUUUUU$
     $U \Rightarrow u|v$
     $T \Rightarrow aT11$
     $T \Rightarrow X$
     $X \Rightarrow xX|yX$
     $X \Rightarrow \epsilon$

     The second cannot be expressed by a CFG because the blocks with parameters $mnsmn$ have the $m-m$ link crossing over the $n-n$ link.