CPSC 313 Final Exam **Fall 2020**
Ali Akbari

1) Prove using induction and the recursive definitions of length and reverse that the following statement holds for all Σ and all $w \in \Sigma^* : |w| = |w^r|$. Recall that $|w| = 1 + |x|$ if $w = ax$ and 0 if $w = \varepsilon$. Recall that $w^r = ax^r$ if $w = xa$ and $w^r = w$ if $w = \varepsilon$.
Let n = $|w|$.
Base (n = 0):
Let $w = \varepsilon$ then
$w \in \Sigma^*$ and $n = |w| = |\varepsilon| = \varepsilon$ then by the recursive definition of reverse,
$\varepsilon = \varepsilon^r = \varepsilon$ then by the recursive definition of length
$|\varepsilon| = |\varepsilon^r|$
$0 = 0$
Thus, this holds true for the base case.

Inductive Hypothesis:
IH: Assume that $|w| = |w^r|$ holds true for some n where $n \geq 0.$
Induction Step:
Prove this holds true for $n + 1.$
Then n + 1 = $|w| + 1$
From the recursive definition of reverse, Let $z = aw$, $z^r = aw^r$ where a is a letter and w is a word. Then by the recursive length and reverse defintions,
$|z| = 1 + |w|$
$|z^r| = 1 + |w^r|$
From the inductive hypothesis
$|w| = |w^r|$
Then,
$|z| = |z^r| = 1 + |w|$.
$|z| = |z^r|$.
$|w| + 1 = |w^r| + 1.$
This holds true for the induction step.
Thus by induction we have proven for all Σ and all $w \in \Sigma^* : |w| = |w^r|.$
■

2) Prove using induction that for an NFA that if there is a state q and letter a such that $q \in \delta(q, a)$ then for all $i \geq 0$ $q \in \delta^*(q, a^i)$. Recall that the transition function for an NFA is $\delta : Q \times \Sigma \rightarrow 2^Q$.

Base (i = 0):

$\delta^*(q, a^0) = \delta^*(q, \varepsilon) = q$

$a^0 = \varepsilon$, epsilon transition in an NFA stays in the state by definition. Thus, this holds for the base case.

Inductive Hypothesis:

IH: Assume that $q \in \delta^*(q, a^n)$ holds true for some i where $i \geq 0.$

Induction Step:

Prove this holds true for $i + 1.$

From the definition of δ^*:

$\delta^*(q, a^{i+1}) = \delta(\delta^*(q, a^i), a)$

By the inductive hypothesis:

Since $q \in$ in the set of $\delta^*(q, a^n)$, then $\delta^*(q, a^n) = q$

So, $\delta(\delta^*(q, a^i), a) = \delta^*(q, a)$

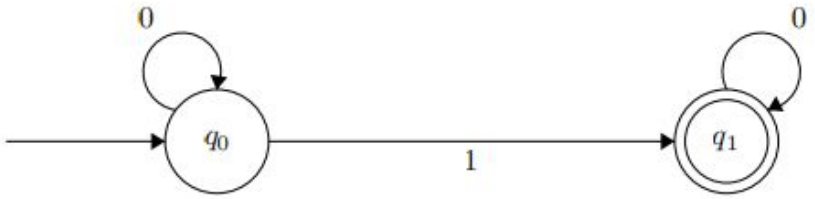
Then, $\delta^*(q, a) = q$

This holds for the induction step.

Thus by induction we have proven, $q \in \delta^*(q, a^i)$ for all $i \geq 0$.

■

3) Define a construction for a context-free grammar that produces the language of state sequences for a finite state machine for valid executions that lead to acceptance. That is: $q_0q_1q_2$ is a word in this language if the machine's initial state is $q0,\ q2 \in A$, and $\exists \sigma,\ \sigma'$ such that $\delta(q_0,\ \sigma) = q1$ and $\delta(q1,\ \sigma') = q2$. For example, suppose you had a finite state machine below:



Then words in this language of state sequences include q_0q_1 and $q_0q_0q_1q_1$ but not q_1q_1 (because it does not start with initial state), not q_0q_0 (because it does not end with an accepting state) and not $q_0q_1q_0$ (because there is no symbol that takes you from q_1 to q_0). Observe that for every word (i.e., state sequence) that is generated by this grammar, there is at least one corresponding word accepted by the machine (i.e., q_0q_1 is 1 and $q_0q_0q_1q_1$ is 010).
N.B. This is only an example. A solution to this problem is a generic construction that takes a DFA as input and produces a CFG as output such that the language of the CFG is the language of valid state sequences in M. Hint: if $\delta(q,\ a) = p$ in the DFA, what corresponding rule in the CFG should exist? What should the variables in the grammar represent?

Let $G = (V,\ \Sigma,\ P,\ S)$:
$V = \{X\}$ - set of variables
$\Sigma = \{q_1, q_2 \ldots\}$ - set of terminals

A generic construction for a given DFA that produces a context-Free Grammar where the language that is produced is a valid state sequence is:
The initial state is q_0 that is given.
Any state in the DFA we have a variable and a terminal such that the transition from that state q_1 to another state q_2 , $\delta(q_1,\ 0) = q_2$ has a production $X_1 \rightarrow q_1X_2$.
Also for each final states q_1 we also have a production such that $X_1 \rightarrow q_1$ which also considers the loops to final state. Thus, this generic construction will take in a DFA and produce a CFG such that the language of the CFG is the language of valid state sequences.
■

4) Prove that the following problem is decidable by giving an algorithm to decide it: given a Turing machine T, does T ever write any non-blank symbol $\sigma \in \Gamma \setminus \{\Delta\}$ on its tape when its tape starts out as completely empty (i.e., processing ε). Give an explanation why your algorithm works and why it will always halt.

D = Does T ever write any non-blank symbol σ on its tape when tape starts out As completely empty?

Suppose a machine T' only writes blank symbols and keep count for the number of states visited. Also checks if it entered a state twice, if it enters a state twice then it halts as a cycle was detected, if the machine halts on its own, return the number of states visited.
Run the machine T upto the maximum number of steps T' took. If a non-blank symbol Is written then halt as it is decidable.
Our algorithm is the following for T_D:

```
Create T ' = T but only writes blank symbols;
MAX = (maximum_number_of_states_visited(T', w));

bool does_T_halt(TM T, WORD w, MAX ) {
        return (does_T_writes_non_blank_in_MAX_steps(T, w, MAX ));
}
```

T' is a cycle detection turing machine that returns the maximum number of states visited, which depends on the number of states and directions. So the maximum is equivalent to q x 2, as we have two directions. This is basically a counter to the number of configurations. The subroutines represents our T_D.The return statement returns true if and only if T writes a non-blank symbol while processing w. Since we know that T' returns maximum number of step we need to process, T will only run for a finite amount of steps and when it writes a non-blank symbol it halts and returns a boolean. Therefore algorithm halts making it decidble. Therefore such a Turing machine T_D can exist.

■

5) Prove that the following problem is undecidable by giving a reduction to a known undecidable problem that we have proven in the lectures: given a Turing machine T, a tape symbol $\sigma \in \Gamma$, and a word $w \in \Sigma^*$, does T ever write σ on its tape while processing w. Clearly state the steps in your reduction. If you do a preprocessing step, for example, include details such as the transition function's behaviour for any modifications.

D = Does T ever write a symbol σ on its tape while processing w?
Reduce to the problem of does T accept w. Given a T, we preprocess it to T' where we insert a new state q. For any (p, x) such that $\delta(p, x) = (h_a, x', d)$ we make $\delta(p, x) = (q, \sigma, d)$. We further add, for all $\sigma \in \Gamma$, $\delta(q, \sigma) = (h_a, \sigma, \rightarrow)$. Thus, the path to ha must transit through q. That is, the old machine enters h_a if and only if the new machine enters q.
By contradiction we first assume that D is solvable. Therefore there exist an always halting turing machine T_D that decides this problem. We will use it to solve the problem of
E ="given T and w, does T halt on w."

Our algorithm is the following for T_E:

```
bool does_T_halt_on_w(TM T, WORD w) {

Create T' = T but with a new state q;

return (does_write_symbol_while_processing_w(T', w, σ) );
}
```

The subroutines represents our T_D The return statement returns true if and only if T ' accepts w if it enters q (h_a of T) while processing w right σ. The return statements suggest that T ' halts on w. However the algorithm halts if we assume that T_D halts. In the lecture notes we are shown that if T accepts w it is also undecidble therefore it cannot halt. Therefore such a Turing machine T_D cannot exist.
■