# PTC (Fall 2018) – Assignment 3

徐翔哲 161250170

2018 年 12 月 19 日

# Problem 1

Consider the (deterministic) Turing machine $M$ given by

$$M = (\{q_0, q_1, q_2\}, \{a, b\}, \{a, b, B\}, \delta, q_0, B, \{q_2\})$$

which has exactly four transitions defined in it, as described below.

1. $\delta(q_0, a) = (q_0, B, R)$
2. $\delta(q_0, b) = (q_1, B, R)$
3. $\delta(q_1, b) = (q_1, B, R)$
4. $\delta(q_1, B) = (q_2, B, R)$

Please answer the following questions:

a. Specify the execution trace of $M$ on the input string $abb$.

b. Provide a regular expression for the language of the Turing machine.

c. Suppose we added the transition $\delta(q_1, a) = (q_0, B, R)$ to the above machine, provide a regular expression for the language of the resulting Turing machine.

### a

$q_0aab \vdash q_0ab \vdash q_0b \vdash q_1B \vdash q_2B$

### b

$a^*b^+$

### c

$a^*b^+(a^+b^+)^*$

# Problem 2

Please design TM′s to decide following languages:

a. $L_1 = \{1^m \times 1^n = 1^{mn} \mid m, n \in \mathbb{N}^+\}$ (e.g. $11 \times 111 = 111111 \in L_1$, but $1 \times 1 = 11 \notin L_1$)

b. $L_2 = \{ww \mid w \in \{a, b\}^*\}$

**a**

First, let's define a TM $M_1$ which will convert $\times 1^n = 1^{n'}$ to $\times 1^n = 1^{n'-n}$ where $n' \geq n > 0$

$$M_1 = (\{p_0, p_R, p_a, p_{a'}, p_d, p_L, p_f, p_e\}, \{1, 0, \times, =\}, \{1, 0, \times, =, B, a\}, \delta, p_0, B, \{p_e\})$$

At the beginning , the head should at $\times$ Then we will enter $p_a$, which will change the first 1 to a when the head goes right.
$\delta(p_0, \times) = (p_a, \times, R)$
$\delta(p_a, a) = (p_a, a, R)$
$\delta(p_a, 1) = (p_R, a, R)$

The state $p_R$ moves the head to the first blank at the right of the input, and then switches to state $p_d$, which will delete a 1.
$\delta(p_R, X) = (p_R, X, R)$, where $X = \{=, 1\}$
$\delta(p_R, B) = (p_d, B, L)$
$\delta(p_d, 1) = (p_L, B, L)$
The state $p_L$ will moves to $\times$ and then enters $p'_a$
$\delta(p_L, X) = (p_L, X, L)$, where $X = \{1, =, a\}$
$\delta(p_L, \times) = (p_a, \times, R)$

Then the state $p_{a'}$ will act as $p_a$, change a 1 to a, then move to the right and delete a 1 ...... except that $p_{a'}$ will go to the state $p_f$ if it sees no 1 before the =, under which condition $p_a$ would halt without accepting.
$\delta(p_{a'}, a) = (p_{a'}, a, R)$
$\delta(p_{a'}, 1) = (p_R, a, R)$
$\delta(p_{a'}, =) = (p_f, =, L)$

The state $p_f$ will change all the a back to 1.
$\delta(p_f, a) = (p_f, 1, L)$
$\delta(p_f, \times) = (p_e, \times, R)$
Then we construct the TM $M$ to describe the language $L_1$

$$M = (M_1.states \cup \{q_0, q_R, q_{call}, q_L, q_1, q_e, q_{check}, q_f\}, \{1, 0, \times, =\}, \{1, 0, \times, =, B, a\}, M_1.\delta \cup \delta, q_0, B, \{q_f\})$$

The state $q_0$ will remove the first 1, and goes to state $q_R$
$\delta(q_0, 1) = (q_R, B, R)$
The state $q_R$ will move to the $\times$ and then enters $q_{call}$
$\delta(q_R, 1) = (q_R, 1, R)$
$\delta(q_R, \times) = (q_{call}, \times, L)$
The state $q_{call}$ will call the $M_1$
$\delta(q_call, 1) = (p_0, 1, R)$

When the M returns from $M_1$, state $q_L$ will move the head to the blank at the left end of the input and then enters $q_1$

$\delta(q_L, 1) = (q_L, 1, L)$

$\delta(q_L, \times) = (q_L, \times, L)$

$\delta(q_L, B) = (q_1, B, R)$

$q_1$ acts just like $q_0$ except that when it finds no 1 before $\times$, it will switch to $q_e$

$\delta(q_1, 1) = (q_R, B, R)$

$\delta(q_1, \times) = (q_e, \times, R)$

$q_e$ and $q_{check}$ will check whether the RHS of the = is blank.

$\delta(q_e, 1) = (q_e, 1, R)$

$\delta(q_e, =) = (q_{check}, =, R)$

$\delta(q_{check}, B) = (q_f, B, L)$

## b

We will use multi-track TMs in this problem.

First, define a TM $M_1$ which will compare whether two strings are equal to each other. The beginning of the second string will be marked as <x,c>, the beginning before the first string is marked as <z,B>, where c $\in$ { a,b }. Other related cells on the second track is initialized to *.

$$M_1(\{q_0, q_c, q_a, q_b, q_{da}, q_{db}, q_L, q_f, q_e\}, \{a, b\}, \{a, b, x, *, z, B\}, \delta, q_0, B, \{q_e\})$$

$q_0$ will move the head to the end of the first string. $\delta(q_0, < *, X >) = (q_0, < *, X >, R)$, where $X \in \{a, b\}$

$\delta(q_0, < x, X >) = (q_c, < x, X >, L)$, where $X \in \{a, b\}$

$q_c$ will remove a character from the tail of the first string and remember this string in its state.

$\delta(q_c, < *, a >) = (q_a, < *, B >, R)$

$\delta(q_c, < *, b >) = (q_b, < *, B >, R)$

$\delta(q_a, < *, X >) = (q_a, < *, X >, R)$

$\delta(q_b, < *, X >) = (q_b, < *, X >, R)$

$q_a$ and $q_b$ will move the head to the end of the second string and remember to delete a or b respectively.

$\delta(q_a, < *, B >) = (q_{da}, < *, B >, L)$

$\delta(q_b, < *, B >) = (q_{db}, < *, B >, B)$

$q_{da}$ and $q_{db}$ will delete one a or one b respectively.

$\delta(q_{da}, < *, a >) = (q_L, < *, B >, L)$

$\delta(q_{da}, < x, a >) = (q_L, < x, B >, L)$

$\delta(q_{db}, < *, b >) = (q_L, < *, B >, L)$

$\delta(q_{db}, < x, b >) = (q_L, < x, B >, L)$

$q_L$ will move the head to the ending blanks of the first string.

$\delta(q_L, < *, X >) = (q_L, < *, X >, L)$, where $X \in \{a, b\}$

$\delta(q_L, < x, X >) = (q_c, < x, X >, L)$, where $X \in \{a, b\}$

$q_c$ will find the next char to remove, when it meets z on the second track, the TM goes to a new state $q_f$

$\delta(q_c, < *, B >) = (q_c, < *, B >, L)$

$\delta(q_c, < z, B >) = (q_f, < z, B >, R)$

$q_f$ will check whether the second string has been reduced to blanks, if so, it will change to the accepting state$q_e$.

$\delta(q_f, < *, B >) = (q_f, < *, B >, R)$

$\delta(q_f, < x, B >) = (q_e, < x, B >, R)$

Then we define the TM M for language $L_2$.

$M(M_1.states \cup \{p_0, p_1, p_{odd}, p_{even}, p_{back}, p_{fwd}, p_R, p_{last}, p_{next}, p_{markZ}\}, \{a, b\}, \{a, b, x, *, t, z, B\}, M_1.\delta \cup \delta, p_0, B, \{q_e\})$

M will accept the empty strings.

$\delta(p_0, < B, B >) = (q_e, < B, B >, R)$

Firstly, marked the first blank at the left of the input as x.

$\delta(p_0, < B, X >) = (p_1, < B, X >, L)$

$\delta(p_1, < B, B >) = (p_{odd}, < x, B >, R)$

If this is the odd indexed(start from 1) char from the input, then the state will change to even and vice versa.

$\delta(p_{odd}, < B, X >) = (p_{even}, < *, X >, R)$, where $X \in \{a, b\}$

$\delta(p_{even}, < B, X >) = (p_{back}, < t, X >, L)$, where $X \in \{a, b\}$

When M makes every two steps right, it will move x forward for one step. Then when the first blank at right is met, the x will be moved to the end of the first string.

$\delta(p_{back}, < *, X >) = (p_{back}, < *, X >, L)$, where $X \in \{a, b\}$

$\delta(p_{back}, < x, X >) = (p_{fwd}, < *, X >, R)$, where $X \in \{a, b, B\}$

$\delta(p_{fwd}, < *, X >) = (p_R, < x, X >, R)$, where $X \in \{a, b\}$

$\delta(p_R, < *, X >) = (p_R, < *, X >, R)$, where $X \in \{a, b\}$

$\delta(p_R, < t, X >) = (p_{odd}, < *, X >, R)$, where $X \in \{a, b\}$

Then we move x forward for one more step.

$\delta(p_{odd}, < B, B >) = (p_{last}, < *, B >, R)$

$\delta(p_{last}, < *, X >) = (p_{last}, < *, X >, L)$, where $X \in \{a, b\}$

$\delta(p_{last}, < x, X >) = (p_{next}, < *, X >, R)$, where $X \in \{a, b\}$

$\delta(p_{next}, < *, X >) = (p_{markZ}, < x, X >, L)$, where $X \in \{a, b\}$

$\delta(p_{markZ}, < *, X >) = (p_{markZ}, < *, X >, L)$, where $X \in \{a, b\}$

Finally, we marked the blank at the left of the input as z, and call $M_1$.

$\delta(p_{markZ}, < *, B >) = (q_0, < z, B >, R)$

# Problem 3

A *useless state* in a Turing machine is one that is never entered on any input string. Consider the problem of determining whether a state in a Turing machine is useless. Formulate this problem as a language and show it is decidable or undecidable. (**Hint**: consider the language $E_{\text{TM}}$)

Construct a TM M' as follows. Given any input s, if s can be interpreted as $< M, w >$, where M is a TM, w is a string, then we input it to the algorithm for ATM. If ATM accepts $< M, w >$, M' will enter a state $QF$, where $QF \notin$ the state set of ATM.
It's obvious that the TM M' will enter the state $QF$ if and only if the ATM accepts $< M, w >$. So if we can decide whether or not $QF$ is a useless state, then we can decide ATM. Since ATM is undecidable, so useless state is undecidable.

# Problem 4

Show that the following questions are decidable:

a. The set $L$ of codes for TM's $M$ such that, when started with the blank tape will eventually write some nonblank symbol on its tape. (**Hint**: If $M$ has $m$ states, consider the first $m$ transitions that it makes)

b. The set $L$ of codes for TM's that never make a move left on any input.

c. The set $L$ of pairs $(M, w)$ such that TM $M$, started with input $w$, never scans any tape cell more than once.

### a

Let's construct a directed graph(N,E,P), where N is the set of all the nodes, E is the set of edges and P is a set for some special nodes. For each transition $\delta(p, X) = (q, Y, D)$, if X is B, we add a node pXq. Also, if Y is not B, then we also add this node to the set P. Specially, we add a node $p_0Bp_0$, where $p_0$ is the start state of M. For each pair of nodes, say, $p_1Bq_1$, $p_2Bq_2$, if $q_1$=$p_2$ and $p_2$ is not a final state , add an edge ($q_1$, $p_2$).

Then we traverse the sub-graph which can be access from $p_0Bp_0$. When we meet any node in P, then we accept this M. Otherwise, if we never meet any node in P after visiting all the reachable node, we reject this M.

### b

Let's construct a directed graph(N,E,P), where N is the set of all the nodes, E is the set of edges and P is a set for some special nodes. Specially, we add a node $p_0Bp_0$, where $p_0$ is the start state of M. For each transition $\delta(p, X) = (q, Y, D)$, we add a node pXq. Also, if D is L, then we also add this node to the set P. For each pair of nodes, say, $p_1Xq_1$, $p_2Yq_2$, if $q_1$=$p_2$ and $p_2$ is not a final state, add an edge ($q_1$, $p_2$).

Then we traverse the sub-graph which can be access from $p_0Bp_0$. When we meet any node in P, then we reject this M. Otherwise, if we never meet any node in P after visiting all the reachable node, we accept this M.

**c**

Suppose that the ID of M is $q_0w$ at the beginning.

**case 1**

If there exists $\delta(q_0, c) = (q', Y, L)$, where c is the first character of w, q' is an arbitrary state and Y is an arbitrary type character, then we construct a graph(N,E,P) as follows. Forall $\delta(p, B) = (q, Y, D)$, add a node pBq in N. If D is R, also add this node in P. Specially, we add a node $q_0Xq'$. For each pair of nodes, say, $p_1Xq_1$, $p_2Yq_2$, if $q_1$=$p_2$ and $p_2$ is not a final state, add an edge ($q_1$, $p_2$).

Then we traverse the sub-graph which can be access from $q_0Bq'$. If we meet any node in P, we reject this (M,w).

**case2**

If there exists $\delta(q_0, c) = (q', Y, R)$, where c is the first character of w, q' is an arbitrary state and Y is an arbitrary type character, then M cannot move any step to left. Suppose the length of w is n. Then we simulate M for n steps. If M moves to left in any step, we reject this (M,w). If M halts within n steps not moving to left, then we accept.

**case3**

If M doesn't halt within n steps, the head must point to the B at the right of w, suppose the state now is $q_x$.
We construct a graph(N,E,P) as follows. Forall $\delta(p, B) = (q, Y, D)$, add a node pBq in N. If D is L, also add this node in P. Specially, we add a node $q_xBq_x$. For each pair of nodes, say, $p_1Xq_1$, $p_2Yq_2$, if $q_1$=$p_2$ and $p_2$ is not a final state, add an edge ($q_1$, $p_2$).

Then we traverse the sub-graph which can be access from $q_xBq_x$. If we meet any node in P, we reject this (M,w). Otherwise, we accept this (M,w).

# Problem 5

If a pushdown automaton has $k$ stacks, we call it $k-$PDA. Clearly, $0-$PDA is NFA, $1-$PDA is PDA, and $1-$PDA is more powerful than $0-$PDA.

1. What is the difference between the express ability of 2-PDA and 1-PDA. Please clarify your argument. Prove the (un)equivalence.
2. How about 3-PDA and 2-PDA.

## 1

It's obvious that 2-PDA and 1-PDA are not equivalence. Since 2-PDA can accept languages like $a^n b^n c^n$ while 1-PDA cannot. Also, it's trivial to show that 2-PDA can express any language 1-PDA can accept. So 2-PDA is more powerful than 1-PDA.

## 2

3-PDA and 2-PDA are equivalence. I'll show that both 3-PDA and 2-PDA are equivalence to TM.
Firstly, I'll prove that every language accepted by 2-PDA can be accepted by TM.
We choose a 2-type TM. The first type contains the input, and the second type is a simulation for the stack. For any transition $\delta'(q, a, X) = (p, Y)$, supposed that the length of Y is k, we will get these transitions for the TM:
If Y=$\epsilon$, for the first type: $\delta(q, a) = (p, a, R)$; for the second type: $\delta(q, X) = (p, B, L)$
If Y$\neq \epsilon$, for the first type:
$\delta(q, a) = (p, a, R)$ for the second type: $\delta(q, X) = (q_{k-1}, Y[k-1], R)$
Also, for each integer i from 0 to k-2, exists $\delta(q_{i+1}, B) = (q_i, Y[i], R)$
finally, add $\delta(q_0, B) = (p, B, L)$
Similarly, we can simulate 3 stack using a 3-type TM.
Now we have shown that any language accepted by 2-PDA and 3-PDA can be accepted by a TM.
Then we'll prove that 2-PDA can simulate a TM.
At the beginning, we just push all the input into stack A, and pop the character from stack A and push this character into stack B one by one, until we meet the stack-bottom marker of A. Then the stack B holds characters at the right of the TM head, the stack A holds characters at the left of the TM head.
For each transition in TM noted as $\delta'(p, X) = (q, Y, D)$:
If D is L, add $\delta(p, \epsilon, A, X) = (q, \epsilon, AY)$, where A is any symbol at the top of A except for the stack-bottom marker of A, noted as $\perp_A$.
When the top of A is $\perp_A$, add $\delta(p, \epsilon, \perp_A, X) = (q, \perp_A, BY)$
If D is R, add $\delta(p, \epsilon, A, X) = (q, YA, \epsilon)$.
In addition, if X is B(the blank symbol in M), also add $\delta(p, \epsilon, A, \perp_B) = (q, YA, \perp_B)$ (D=R) or $\delta(p, \epsilon, A, \perp_B) = (q, \epsilon, AY\ \perp_B)$(D=L & A $\neq \perp_A$) and $\delta(p, \epsilon, \perp_A, \perp_B) = (q, \perp_A, BY\ \perp_B)$(D=L & A = $\perp_A$);

Finally, change all the occurrences like $B \perp_B$, $B \perp_A$ to $\perp_B$, $\perp_A$ respectively.
Now we have shown that 2-PDA can simulate a TM. Since 3-PDA can simulate 2-PDA by using only 2 stack, 3-PDA can simulate a TM. Then both 2-PDA and 3-PDA are equivalent to TM, which means they are equivalent.

# Problem 6

Suppose we have an encoding of context-free grammars using some finite alphabet. Consider the following two languages:

1. $L_1 = \big\{(G, A, B) \mid G$ is a (coded) CFG, $A$ and $B$ are (coded) varibles of $G$, and the sets of terminal strings derived from $A$ and $B$ are the same$\big\}$.
2. $L_2 = \big\{(G_1, G_2) \mid G_1$ and $G_2$ are (coded) CFG's, and $L(G_1) = L(G_2)\big\}$.

Answer the following questions:

a. Show that $L_1$ is polynomial-time reducible to $L_2$.

b. Show that $L_2$ is polynomial-time reducible to $L_1$.

### a

Just copy the language G twice, but mark A as the start symbol of $G_1$, mark B as the start symbol of $G_2$. The copy action will consume just O(n) time, and the modification will consume constant time. So the reduction is polynomial-time. Also, we should prove that when $L_2$ accepts, $L_1$ accepts and when $L_2$ rejects, $L_1$ rejects.
If $L_2$ accepts, that means the terminal strings derived from the start symbol of $G_1$ and $G_2$ are equivalent. Since $G_1$ and $G_2$ share the same productions in G, then they will surely derivate the same terminal strings when they're two variables in G.
If $L_2$ rejects, we can assume that there is a string w, which is in $L(G_1)$ but is not in $L(G_2)$, then variable A in G can derivate the terminal string w but variable B cannot. Then (G,A,B) should be rejected.

### b

Suppose that the start symbol of $G_1$ and $G_2$ are $S_1, S_2$ respectively. Then construct G as $S \to S_1|S_2$. Before we copy productions from $G_1$ and $G_2$, we just rename variables in $G_1$ such that variables in $G_1$ are disjoint from variables in $G_2$. Then we get an instance of $L_1$ which will be $(G, S_1, S_2)$. It's obvious that the reduction takes O(n) time. Now we're going to prove that when $L_1$ accepts, so does $L_2$, and when $L_1$ rejects, so does $L_1$.
If $L_1$ accepts, that means $S_1$ and $S_2$ can derivate the exact same terminal string set. Also, notice that all the production they can use are belongs to $G_1$ and $G_2$ respectively since we have guarantee that the variables in $G_1$ are disjoint with variables in $G_2$. So for each terminal string they can derivate in G, they can also derivate it in $G_1$ or $G_2$. So we can conclude that $L(G_1) = L(G_2)$.

If $L_1$ rejects, we can assume that there is a terminal string w that belongs to the terminal strings of $S_1$ but not belongs to the terminal strings of $S_2$. It's obvious that $S_1$ can derivate w in $G_1$. So $L(G_1) \neq L(G_2)$.

# Problem 7

As classes of languages, $\mathcal{P}$ and $\mathcal{NP}$ each have certain closure properties. Prove or disprove that $\mathcal{P}$ and $\mathcal{NP}$ are closed under each of the following operations:

a. Union.

b. Concatenation.

c. Complementation.

**Proof.**