Theory of Computation Assignment no. 6

Goktug Saatcioglu

(1) $A = \{a^i b^j \mid i \neq j^2\}$. Let $x = a^{i^2}$ where $i \geq 0$ and $y = a^{j^2}$ where $j \geq 0$. If $i \neq j$, we claim that x and y are not equivalent with respect to A. Let $z = b^j$, then $xz \in A$ but $yz \notin A$. Thus, x and y are not equivalent and A has an infinite number of non-equivalent words with respect to A. Therefore, A is not regular.

(2) a. Assume that $w \sim_A w'$ which means $[w]_A = [w']_A$. Now consider the extension of δ, the δ^* function for $\delta^*([\lambda]_A, wa)$ where $w \in \Sigma^*$ and $a \in \Sigma$.

$$\begin{aligned}
\delta^*([\lambda]_A, wa) &= \delta(\delta^*([\lambda]_A, w), a) && \text{by definition of } \delta^* \\
&= \delta(\delta^*([\lambda]_A, w'), a) && \text{by our assumption that } w \sim_A w' \\
&= \delta^*([\lambda]_A, w'a) && \text{by definition of } \delta^* \\
&\implies [wa]_A = [w'a]_A \implies wa \sim_A w'
\end{aligned}$$

Thus, we have shown that for any $w \in \Sigma^*$, if $w \sim w'$ then for every $a \in \Sigma$ it also holds that $wa \sim_A w'a$. The intuition behind this is if we run a DFA M on words w and w' from the initial state independently from each other they will end up in the same state if the words are equivalent with respect to M. Since we assume that $w \sim_A w'$ we know that when M is run on w and w' from the initial state independently from each other, they will end up in the same state. Then for all $a \in \Sigma$, if we run a they should again end up in the same state as there is only a single transition to take because M is a DFA. Thus, running M on wa and $w'a$ will also end up in the same state if $w \sim_A w'$.

b. We show that $\delta^*([\lambda]_A, w) = [w]_A$ for every $w \in \Sigma^*$ using induction on the length of w, say n.
Base cases.
$n = 0 \implies |w| = 0 \implies w = \lambda$.

$$\begin{aligned}
\delta^*([\lambda]_A, w) &= \delta^*([\lambda]_A, \lambda) && w = \lambda \\
&= q_0 && \text{by definition of } \delta^* \\
&= [\lambda]_A && \text{since } q_0 = [\lambda]_A \\
&= [w]_A && \text{since } \lambda = w
\end{aligned}$$

$n = 1 \implies |w| = 1 \implies w = a$.

$$\begin{aligned}
\delta^*([\lambda]_A, w) &= \delta^*([\lambda]_A, a) && w = a \\
&= [\lambda a]_A && \text{by definition of } \delta^* \\
&= [a]_A && \\
&= [w]_A && \text{since } a = w
\end{aligned}$$

Since $LHS = RHS$ for both cases, the base cases hold.
Inductive step. For some $n \geq 0$, assume that $\delta^*([\lambda]_A, w) = [w]_A$ where $|w| = n$. Now for $k = n$, consider a w' such that $|w'| = k + 1$ and $w' = wa$, where $w \in \Sigma^*$ such that $|w| = k$ and $a \in \Sigma$.

Now we evaluate $\delta^*([\lambda]_A, w')$.

$$
\begin{aligned}
\delta^*([\lambda]_A, w') &= \delta^*([\lambda]_A, wa) && w' = wa \\
&= \delta(\delta^*([\lambda]_A, w), a) && \text{by definition of } \delta^* \\
&= \delta([w]_A, a) && \text{by the induction hypothesis} \\
&= [wa]_A && \text{by definition of } \delta \\
&= [w']_A && wa = w'
\end{aligned}
$$

Conclusion. By the principle of induction, we see that for every $w \in \Sigma^*$, $\delta^*([\lambda]_A, w) = [w]_A$. $\square$

(3) (iii) The sequence of vertices and associated data configurations the PDA goes through in a recognizing computation on input $abba$ is as follows:

$$\to p \xrightarrow[\text{Push } \$]{\lambda} q \xrightarrow[\text{Push } A]{\text{Read } a} q \xrightarrow[\text{Pop } A]{\text{Read } b} q \xrightarrow[\text{Pop } \$, \text{ Push } \$]{\lambda} r \xrightarrow[\text{Push } B]{\text{Read } b} r \xrightarrow[\text{Pop } A]{\text{Read } a} r \xrightarrow[\text{Pop } \$]{\lambda} s.$$

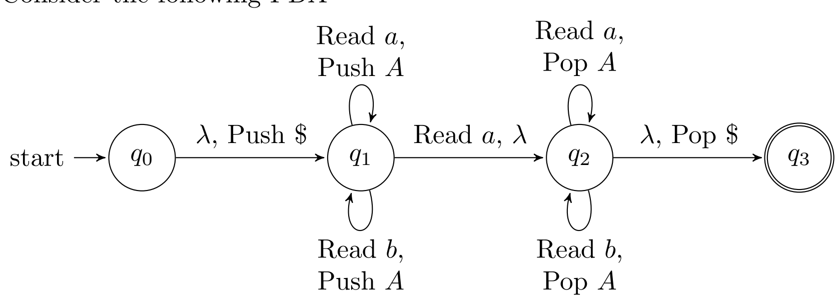
There are other computation paths that can be followed on this input but they are trivial paths. One such path would be:

$$\to p \xrightarrow[\text{Push } \$]{\lambda} r \xrightarrow[\text{Pop } \$, \text{ Push } \$]{\lambda} q \xrightarrow[\text{Push } A]{\text{Read } a} q \xrightarrow[\text{Pop } A]{\text{Read } b} q \xrightarrow[\text{Pop } \$, \text{ Push } \$]{\lambda} r \xrightarrow[\text{Push } B]{\text{Read } b} r \xrightarrow[\text{Pop } A]{\text{Read } a} r \xrightarrow[\text{Pop } \$]{\lambda} s.$$

Many similar such paths can be created by performing trivial operations until getting to q and performing the Read a, Push A operation, then performing such trivial operations between q and r before performing the Read b, Push B operation and then finally looping back and forth between q and r without reading anything until we choose to get to s using Pop \$. Even though these paths are trivial, they are possible and valid.

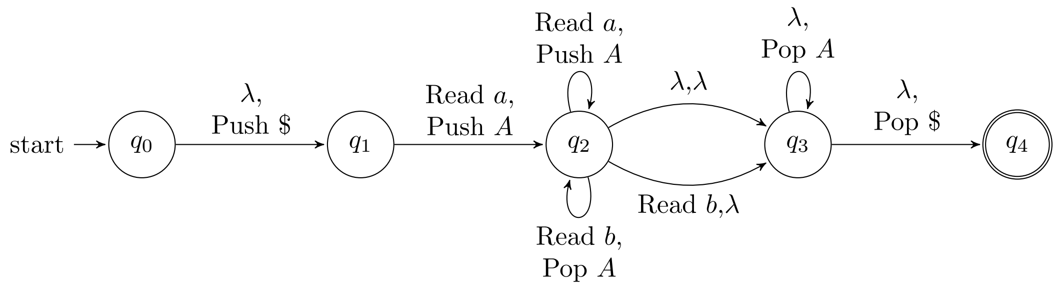
(iv) The language accepted by this PDA is $\{w \in \{ab, ba\}^*\}$.

(4) (i) Consider the following PDA



where q_0 is the empty string λ and empty stack, q_1 reads some $\{a, b\}^*$ and the stack contains $|q_1|$ number of A's, q_2 has read a single a and then reads some $\{a, b\}^*$ as the stack empties out $|q_1|$ number of A's, and q_3 is the empty string λ and empty stack with the number of A's pushed in q_1 equaling the number of B's popped in q_2. This works since we want an equal number of letters on the right and left of the middle a. We use the fact that if the number of letters right and left of a is odd then w is odd and, similarly, if the number of letters right and left of a is even then w is odd. Furthermore, in both cases a will end up in the middle. Thus, this PDA will start pushing a \$ (shielding) sign onto the stack and then A's onto the stack for some n number of letters it sees which means the stack will contain n number of A's. When a middle a is seen nothing is pushed or popped from the stack and every letter seen after the middle a will mean we pop an A from the stack. If we can pop n number of letters we will only end up with the \$ (shielding) sign on the stack and we can pop this to end up at the accepting state.

(vi) Consider the following PDA



where q_0 is the empty string λ and empty stack, q_1 is the state where the shielding symbol \$ is pushed onto the stack and a first a is read such that A is pushed onto the stack (the necessity of this is explained below), q_2 reads some amount of a's and b's and pushs A onto the stack for every a and pops A from the stack for every b such that for every initial string $\#a(w) \geq \#b(w)$, q_3 is the state where a final letter is read and now we can keep popping A's until we get to the shielding symbol, and q_4 is the empty string and empty stack where we have recognized the language specified by the question. The way I interpreted this question is that for every $1 \leq k < |w|$ length "initial string" of w, (i.e. $abab$ has initial strings a, ab, and aba) $\#a(w) \geq \#b(w)$. For us to recognize this language we need to ensure that every inputs starts with an a and we count the occurrence of this a by pushing an A onto the stack. In this case we don't need to use shielding and the reason for this will be explained below. Then for every next a we see we push an a and for every next b we see we pop a B. This gaurantees that as we read letters $\#a(w) \geq \#b(w)$ and something like abb will not be recognized by the PDA. However, we know that $aabbb$ should be recognized as the whole word w is not an initial string and thus, we add a λ transition for reading a b that takes us to the state where we can continually pop A's until we get to the shielding symbol. In this state, if we can't pop A's until we get to \$, then we know that the word is not in the language and otherwise we pop \$ and get to the accepting state.

(5) (v) The set of strings generated by this grammar involve the concatenation of two different set of strings. The first set is the set of strings such that there are some j amount of words in the form $c^j d^j$ where j can be 0 which is preceeded by m amount of a's and suceeded by i amount of b's where i can be 0. The second set is an o amount of word in the form $c^k d^k$ where k can be 0. In terms of set notation, S can be described as

$$\{(a^i(c^j d^j)b^i)(c^k d^k) \mid i, j, k \geq 0\}.$$

(6) (i) The context free grammar is as follows:

$$\begin{aligned} S &\to TST \mid a \\ T &\to a \mid b \end{aligned}$$