

Homework 8: Solutions

November 17, 2008

Problem 1. Recall the operations *NOPREFIX* and *NOEXTEND*.

$NOPREFIX(A) = \{w \in A \mid \text{no proper prefix of } w \text{ is a member of } A\}$.

$NOEXTEND(A) = \{w \in A \mid w \text{ is not a proper prefix of any string in } A\}$.

- Show that the class of CFLs is not closed under *NOPREFIX* operation.
- Show that the class of CFLs is not closed under *NOEXTEND* operation.

Solution

- Consider the language $L = L_1 \cup L_2$ where $L_1 = \{a^n b^m c \mid n \neq m, n, m \geq 1\}$ and $L_2 = \{a^n b^m c^m \mid n, m \geq 1\}$. If we consider a string of L_1 , $a^n b^m c$, its proper prefixes are strings containing only *as* and *bs* and all strings in L_1 and L_2 contain at least 1 *c*. Hence every string in L_1 is in *NOPREFIX*(L). If we consider a string in L_2 , $a^n b^m c^m$, it is not in *NOPREFIX*(L) iff there is a proper prefix of it which is in L , since no proper prefix of it is in L_2 (the number of *bs* will not be equal to the number of *cs* in any proper prefix), the proper prefix will have to come from L_1 and hence the $n \neq m$. So the strings in L_2 which are in *NOPREFIX*(L) are $\{a^n b^n c^n \mid n \geq 1\}$. Therefore $NOPREFIX(L) = L_1 \cup \{a^n b^n c^n \mid n \geq 1\}$.

L is context-free since L_1 and L_2 are both context-free and context-free languages are closed under union. However *NOPREFIX*(L) is not context-free. Otherwise $NOPREFIX(L) \cap L(a^* b^* c c^*) = \{a^n b^n c^n \mid n \geq 2\}$ is context-free which is a contradiction. Therefore there exists a context-free language L such that *NOPREFIX*(L) is not context-free, hence context-free languages are not closed under *NOPREFIX* operation.

- Consider the language $L = L_1 \cup L_2$ where $L_1 = \{a^n b^m c^p \mid n \neq m, n, m, p \geq 1\}$ and $L_2 = \{a^n b^m c^m \mid n, m \geq 1\}$. Consider $a^n b^m c^p \in L_1$, this string is not in *NOEXTEND*(L) since $a^n b^m c^{p+1}$, an extension of the string is in L . Now consider a string $a^n b^m c^m$, any extension of this string in L should

belong to L_1 (since any extension will have more cs than bs). Hence this string will not be in $NOEXTEND(L)$ iff an extension of it belongs to L_1 iff $n \neq m$. Therefore strings of the form $a^n b^n c^n$ belong to $NOEXTEND(L)$. Hence $NOEXTEND(L) = \{a^n b^n c^n \mid n \geq 1\}$. As before L is context-free but $NOEXTEND(L)$ is not and hence context-free languages are not closed under the $NOEXTEND$ operation.

□

Problem 2. If A and B are languages, define $A \diamond B = \{xy \mid x \in A \text{ and } y \in B \text{ and } |x| = |y|\}$. Show that if A and B are regular languages, then $A \diamond B$ is a CFL.

Solution Let $M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$ be a DFA recognizing A and $M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ be a DFA recognizing B . We construct a PDA P recognizing $A \diamond B$. The idea behind the construction is the following. Given a word w we simulate M_1 on a prefix x of w and store the length of x in the stack (its height). We then run M_2 on the rest of the string y (note $w = xy$) and check that $|y| = |x|$, which is essentially done by popping a symbol from the stack for every symbol of y read. Note that we need to non-deterministically guess where to break w into x and y . Formally $P = (Q, \Sigma, \Gamma, \delta, q_0, F)$, where:

- $Q = Q_1 \cup Q_2 \cup \{q_0, q_f\}$ (Q_1 and Q_2 can be assumed to be disjoint).
- $\Gamma = \{0, \$\}$.
- $(q_{01}, \$) \in (q_0, \epsilon, \epsilon)$.
- $(\delta_1(q, a), 0) \in \delta(q, a, \epsilon)$ iff $q \in Q_1$.
- $(\delta_2(q, a), \epsilon) \in \delta(q, a, 0)$ iff $q \in Q_2$.
- $(q_f, \epsilon) \in \delta(q, \epsilon, \$)$ iff $q \in F_2$.
- $(q_{02}, \epsilon) \in \delta(q, \epsilon, \epsilon)$ iff $q \in F_1$.
- q_0 .
- $F = \{q_f\}$.

A string $w \in A \diamond B$ iff $w = xy$, $|x| = |y|$, $x \in A$ and $y \in B$ iff $w = xy$, $|x| = |y|$, $q_{01} \xrightarrow{x} q_1$ where $q_1 \in F_1$ and $q_{02} \xrightarrow{y} q_2$ where $q_2 \in F_2$ (1). It is easy to see from the construction that $(q_0, \epsilon) \xrightarrow{\epsilon} (q_{01}, \$)$, $(q_{01}, \$) \xrightarrow{x} (q_1, 0^{|x|}\$)$, $(q_1, 0^{|x|}\$) \xrightarrow{\epsilon} (q_{02}, 0^{|x|}\$)$, $(q_{02}, 0^{|x|}\$) \xrightarrow{y} (q_2, \$)$ and $(q_2, \$) \xrightarrow{\epsilon} (q_f, \epsilon)$ (*). On the other hand any string which is accepted by P has to be of the form xy where $|x| = |y|$ and has to go through a sequence of steps of the above form. Hence $w \in L(P)$ iff $w = xy$, $|x| = |y|$ and there is a sequence of transitions of the form (*) iff (1). Therefore $L(P) = L(A \diamond B)$.

Alternate Solution If A and B are regular languages over Σ , then so is $L_1 = \{x\#y \mid x \in A, y \in B\}$ where $\# \notin \Sigma$. This is because $L_1 = A \cdot \{\#\} \cdot B$ and regular languages are closed under concatenation. Let $L_2 = \{x\#y \mid x, y \in \Sigma^*, |x| = |y|\}$. L_2 is context-free since it is generated by the grammar $(\{S\}, \Sigma \cup \{\#\}, R, S)$ where R is $S \rightarrow aSb$ for all $a, b \in \Sigma$ and $S \rightarrow \#$. Since the intersection of a context free language and a regular language is context-free, we have $L_3 = L_1 \cap L_2$ is context-free. But $L_3 = \{x\#y \mid x, y \in \Sigma^*, x \in A, y \in B, |x| = |y|\} = A \diamond B$. Therefore if A and B are regular $A \diamond B$ is context-free. \square

Problem 3. Let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ be a Turing machine. Construct a type 0 grammar $G = (V, \Sigma, R, S)$ such that $L(G) = L(M)$.

Solution We want to build a grammar G which generates w if and only if w is accepted by M . We do this by simulating M on w . An intermediate string will correspond to a configuration of M , if we reach an accepting configuration, then the grammar will output w . However initially we need to guess w , and remember it so that we can output it if it is in the language. The grammar does the following. It first generates w , and makes a copy of it. It then simulates M on the second copy. If the second part ever reaches an accepting computation, it erases the second part and outputs w . Otherwise either the second part reaches a reject configuration, and there are no rules to continue the derivation, or the turing machine never halts on w in which case the derivation continues never yielding a string of terminals.

The set of non-terminals $V = \{S, S_1, \$, *, \#, M, Z, B, O, \} \cup Q \cup (\Gamma \setminus \Sigma)$.

The following are the rules. The first two rules generate a string of the form $\#w\$*$, where $w \in \Sigma^*$. Next we generate $\#w\$q_0w*$, where q_0w corresponds to the initial configuration.

- $S \rightarrow S_1\$*$.
- $S_1 \rightarrow S_10 \mid S_11 \mid \#$.
- $\#0 \rightarrow \#0MZ, Z0 \rightarrow 0Z, Z1 \rightarrow 1Z, Z\$ \rightarrow \$Z, Z* \rightarrow B0*, 0B \rightarrow B0, 1B \rightarrow B1, \$B \rightarrow B\$$.
- $\#1 \rightarrow \#1MO, O0 \rightarrow 0O, O1 \rightarrow 1O, O\$ \rightarrow \$O, O* \rightarrow B1*, 0B \rightarrow B0, 1B \rightarrow B1, \$B \rightarrow B\$$.
- $MB0 \rightarrow 0MZ, MB1 \rightarrow 1MO, MB\$ \rightarrow \q_0 .

The last three rules correspond to copying the w bit by bit between $\$$ and $*$. M is like a marker, Z remembers that 0 was read and moves forward, similarly O remembers that 1 was read and moves forward, when the end $*$ is reached the character being remembered is written and B corresponds to moving backwards. Finally q_0 is inserted.

The following rules correspond to simulating the turing machine M on w on the part following the $\$$. Each rule that can be applied corresponds to rewriting the part between $\$$ and $*$ to obtain the next configuration. Observe that the next configuration depends on a constant number of bits around the marker which is the state the TM is in.

- If $\delta(q, a) = (q', a', L)$, then there is a rule $bqa \rightarrow q'ba'$ for every b which is not $\$$, and a rule $bqa \rightarrow bq'a'$ if $b = \$$.
- If $\delta(q, a) = (q', a', R)$, then there is a rule $qa \rightarrow a'q'$.
- $* \rightarrow \sqcup*$.

The following rules correspond to erasing the second part and accepting w when an accepting configuration is reached, i.e, if we reach an accepting configuration $\#w\$q_{accept}\beta*$, then we want to have rules which will derive w from it.

- $q_{accept}a \rightarrow q_{accept}$, $aq_{accept} \rightarrow q_{accept}$ for all $a \in \Gamma$.
- $\$q_{accept}* \rightarrow F$, $0F \rightarrow F0$, $1F \rightarrow F1$, $\#F \rightarrow \epsilon$.

$q_{accept}a \rightarrow q_{accept}$ can be used to erase β and $aq_{accept} \rightarrow q_{accept}$ can be used to erase α . Then we reach the configuration $\$q_{accept}*$ which is replaced by F to obtain $\#wF$, then F moves left and when it reaches $\#$ erases it.

Alternate Solution Alternately, we could start with an accepting configuration and simulate the turing machine backwards, if we reach an initial configuration which is of the form q_0w then we could output w by dropping off the q_0 .

Formally $G = (V, \Sigma, R, S)$. $V = \{S, A, \$, *\} \cup \Gamma \cup Q$. The rules in R are given by:

$S \rightarrow \$Aq_{accept}A*$, $A \rightarrow aA \mid \epsilon$ for all $a \in \Gamma$. S generates an accepting configuration.

If $\delta(q, a) = (q', a', R)$, then add a rule $a'q' \rightarrow qa$, and if $\delta(q, a) = (q', a', L)$, then add rules $q'ba' \rightarrow bqa$ for $b \in \Gamma$, and $\$q'a' \rightarrow \qa . The above rules simulate the turing machine backwards.

$\sqcup* \rightarrow *$, $* \rightarrow \epsilon$, $\$q_0 \rightarrow \epsilon$. If a configuration with $q_0w\sqcup^i$ is reached, then we could have started with q_0w and reached an accepting configuration, in other words, w is accepted by the TM, hence we drop the blanks and q_0 , to generate w .

□