Non CFG and Turing Machines

2.17) Use the results of Problem 2.8 to give another proof that every regular language is context free, by showing how to convert a regular expression directly to an equivalent context free grammar.

From problem 2.8 we know that the class of context free languages is closed under the regular operations union, concatenation and star. From the definition of regular expressions given in Section 1.3 of Sipser we know that a regular expression consists of a series of union, concatenation and star applied to finite sets of characters from the alphabet $\mathbf{\Sigma}$.

Given a finite set of characters $c = \{a_1...a_n\} \subseteq \mathbf{\Sigma}$ we can construct a variable S which accepts elements in the set c, as shown below.

$$S \rightarrow a_1|...|a_n$$

Given variables such as S for each finite set in the regular expression we can construct a new variable, using the results found in Problem 2.8, for each union, concatenation and star operation in the regular expression.

2.3) Use the pumping lemma to show that the following languages are not context free.

a) $L = \{0^n1^n0^n1^n | n \geqslant 0\}$

Let $s = 0^p1^p0^p1^p$. If the language L is context free then s can be pumped.

$$s_i = \mathrm{uv}^i\mathrm{xy}^iz$$

$$|\mathrm{vy}| > 0$$

$$|\mathrm{vxy}| \leqslant p$$

Case 1: If x and y contain only a single character from the alphabet then the resulting string will be unbalanced.

Case 2: If x or y contain a mixture of 0's and 1's then the resulting string will contain too many sets of 0's and 1's

d) $L = \{t_1\#t_2\#...\#t_k | k \geqslant 2, \text{each } t_i \in \{a, b\}^*, \text{and } t_i = t_j \text{ for some } i \neq j\}$

Let $s = a^p\mathrm{b}a^p\#a^p\mathrm{b}a^p$. If the language L is context free then s can be pumped.

$$s_i = \mathrm{uv}^i\mathrm{xy}^iz$$

$$|\mathrm{vy}| > 0$$

$$|\mathrm{vxy}| \leqslant p$$

Case 1: # is pumped in either v or y, in this case pumping down, i.e. $i = 0$, results in a string with no #'s which violates the condition $k \geqslant 2$ therefore #'s can not be pumped.

Case 2: Only one substring, t_1 or t_2 is pumped and the other remains unchanged, the resulting string contains two substrings which are unequal.

Case 3: Pump a's in at the end of t_1 with v and at the start of t_2 with y. Resulting substrings are unequal since t_1 contains more a's before the b and t_2 contains more a's after the b.

All 3 possible cases result in a string s_i that is not in L, therefore the language L cannot be pumped and is not context free.

2.32) Let $\Sigma = \{1, 2, 3, 4\}$ and $C = \{w \in \Sigma^* | \text{num}(1) = \text{num}(2) \wedge \text{num}(3) = \text{num}(4)\}$ Show that C is not context free.

Let $s = 1^p 3^p 2^p 4^p$. If the language C is context free then s can be pumped

$$s_i = \text{uv}^i\text{xy}^i z$$

$$|\text{vy}| > 0$$

$$|\text{vxy}| \leqslant p$$

We know that v or y must contain something, However the constraint $|\text{vxy}| \leqslant p$ means that it is not possible to pump 1's and 2's or 3's and 4's at the same time. Therefore the resulting string is going to contain to have either num(1) $\neq$ num(2) or num(3) $\neq$ num(4) and is therefore not in the language C so the language C is not context free.

3.2) This exercise concerns TM M_1 whose description and state diagram appear in Example 3.9. In each of the parts, give the sequence of configurations that M_1 enters when startd on the indicated input string.

a. 11

$q_1 11 \rightarrow \text{xq}_3 1 \rightarrow x1q_3\sqcup \rightarrow x1\sqcup q_{\text{reject}}$

b. 1#1

$q_1 1\#1 \rightarrow \text{xq}_3\#1 \rightarrow x\#q_5 1 \rightarrow \text{xq}_6\#x \rightarrow q_7 x\#x \rightarrow \text{xq}_1\#x \rightarrow x\#q_8 x \rightarrow x\#q_8\sqcup \rightarrow x\#x\sqcup q_{\text{accept}}\sqcup$

c. 1##1

$q_1 1\#\#1 \rightarrow \text{xq}_3\#\#1 \rightarrow x\#q_5\#1 \rightarrow x\#\#q_{\text{reject}}1$

d. 10#11

$q_1 10\#11 \rightarrow \text{xq}_1 0\#11 \rightarrow x0q_1\#11 \rightarrow x0\#q_5 11 \rightarrow x0q_6\#x1 \rightarrow \text{xq}_7 0\#x1 \rightarrow q_7 x0\#x1 \rightarrow \text{xq}_1 0\#x1 \rightarrow \text{xxq}_2\#x1 \rightarrow \text{xx}\#q_4 x1 \rightarrow \text{xx}\#\text{xq}_4 1 \rightarrow \text{xx}\#x1q_{\text{reject}}\sqcup$

e. 10#10

$q_1 10\#10 \rightarrow \text{xq}_3 0\#10 \rightarrow x0q_3\#10 \rightarrow x0\#q_5 10 \rightarrow x0q_6\#x0 \rightarrow \text{xq}_7 0\#x0 \rightarrow q_7 x0\#x0 \rightarrow \text{xq}_1 0\#x0 \rightarrow \text{xxq}_2\#x0 \rightarrow \text{xx}\#q_4 x0 \rightarrow \text{xx}\#\text{xq}_4 0 \rightarrow \text{xx}\#q_6\text{xx} \rightarrow \text{xxq}_6\#\text{xx} \rightarrow \text{xq}_7 x\#\text{xx} \rightarrow \text{xxq}_1\#\text{xx} \rightarrow \text{xx}\#q_8\text{xx} \rightarrow \text{xx}\#\text{xq}_8 x \rightarrow \text{xx}\#\text{xxq}_8\sqcup \rightarrow \text{xx}\#\text{xx}\sqcup q_{\text{accept}}\sqcup$

Turing Machines

Question 3.8 Give implementation-level descriptions of Turing machines that decide the following languages over the alphabet

Note: An implementation level description is defined by Siper (Page 159) as an implementation "...in which we use English prose to describe the way that the Turing machine moves its head and the way that it stores data on its tape."

b) $\{w \,|\, w \,\text{contains twice as many 0's as 1's}\}$

1. Scan the tape and mark the first 1 which has not been marked. If no unmarked 1's are found go to stage 5. Otherwise move the head back to the start of the tape

2. Scan the tape until an unmarked 0 is found, mark the 0, if no 0's are found reject

3. Scan the tape once more until an unmarked zero is found, mark the 0, if no 0's are found reject

4. Move the head back to the start of the tape and go to stage 1

5. Move the head back to the start of the tape. Scan the tape to see if any unmarked 0's are found. If none are found accept, otherwise reject.

c) {w | w does not contain twice as many 0's as 1's}

1. Scan the tape and mark the first 1 which has not been marked. If no unmarked 1's are found go to stage 5. Otherwise move the head back to the start of the tape

2. Scan the tape until an unmarked 0 is found, mark the 0, if no 0's are found accept

3. Scan the tape once more until an unmarked zero is found, mark the 0, if no 0's are found accept

4. Move the head back to the start of the tape and go to stage 1

5. Move the head back to the start of the tape. Scan the tape to see if any unmarked 0's are found. If none are found reject, otherwise accept.

Countability & Decidability

Question 4.2 Consider the problem of determining whether a DFA and a regular expression are equivalent. Express this problem as a language and show that it is decidable.

$A = \{\langle R, B\rangle \,|\, R \,\text{is a regular expression and}\, B \,\text{is an implementation of}\, R \,\text{on a DFA}\}$

Proof the following TM M_1 decides language A

1. Convert regular expression R to an equivalent NFA, by using the procedure for this conversion given in Theorem 1.54

2. Convert the NFA to an equivalent DFA, denoted A, using the procedure for this conversion given in Theorem 1.39

3. Run TM F from Theorem 4.5, on $\langle A, B \rangle$ if it accepts, accept. If F rejects, reject.

Question 4.6 Let $\mathcal{B}$ contain all infinite sequences from $\{0, 1\}$. Show that $\mathcal{B}$ is uncountable.

As usual, a diagonalisation argument will solve this one for us. First we assume that $\mathcal{B}$ is countable. If $\mathcal{B}$ is to be countable, it can be indexed by $\mathbb{N}$ to give a list $\langle b_1, b_2, b_3, ... \rangle$ containing every element of $\mathcal{B}$. Now we construct a new infinite bit sequence s where the i^{th} bit of s is the opposite of the i^{th} bit of b_i.

By construction we see that there is no i where $s = b_i$, since s differs from every b_i in at least one location. Consequently s isn't in the list and is hence not in $\mathcal{B}$; however, we also note that s is an infinite sequence of bits and so must be contained in $\mathcal{B}$ by definition. From this contradiction we conclude that no such list can exist and $\mathcal{B}$ is not countable.

Question 4.7 Let $T = \{(i, j, k) | i, j, k \in \mathcal{N}\}$. Show that T is countable.

Here it suffices to construct a function $f :: T \to \mathbb{N}$ which is uniquely invertible. Remember the unique factorisation theorem, which states that every number in $\mathbb{N}$ is expressible as a unique product of primes. Consequently we may choose three arbitrary primes $p \neq q \neq r$ and define our function $f(i, j, k) = p^i q^j r^k$. From the unique prime factorisation theorem we see that any choice of i, j, k uniquely determines a value of f, and so f is invertible.

Question 4.22 A *useless state* in a pushdown automaton is never entered on any input string. Consider the problem of determining whether a PDA has any useless states. Formulate this problem as a language and show that it's decidable.

Let $\mathcal{P}$ be the set of all pushdown automata. Let the language $\mathcal{U} = \{x \in \mathcal{P} | x \text{ has a useless state}\}$. To show that $\mathcal{U}$ is decidable we design a Turing machine which accepts only strings in $\mathcal{U}$. From the book we know that the question of whether a PDA has an empty language is decidable, and we may reduce the question of whether a given state q is useless to this question by making q the only accept state and then determining whether the resulting PDA has an empty language. If it does, then q is a useless state.

Our Turing machine may therefore solve the question of whether there are *any* useless states by performing this test for each state in order.

Question 5.13 A *useless state* in a turing machine is never entered on any input string. Consider the problem of determining whether a TM has any useless states. Formulate this problem as a language and show that it's undecidable.

The language would be $\mathcal{L} = \{x | x \text{ is a Turing machine and } x \text{ has a useless state}\}$. If we had a decider for this language we could employ it to solve the halting problem thus: given a Turing machine T with input I for which we wish to solve the halting problem, we construct a new machine T' which ignores its input, cycles through every state except a special state q, and then executes T on I. Once T has terminated T' enters state q.

Now if we can determine that T' has a useless state that state must be q, and consequently T does not halt with input I. Otherwise T' has no useless states and must have halted on I. Since this question is undecidable, $\mathcal{L}$ must also be undecidable.