Grammars for R.e sets

Video Lecture "Unrestricted grammars and Turing Machines", related practice problems and their solutions are on http://cse.iitkgp.ac.in/ abhij/course/theory/FLAT/Spring20/

1 Revisiting Chomsky hierarchy

In the beginning of this course, we had had a brief look at the Chomsky hierarchy. At that point of time, we only intuitively understood grammars. Now, we have a far better understanding and so it is time to revisit the Chomsky hierarchy to formalise definitions. For a tuple (Γ, Σ, P, S) that usually denotes a grammar, we obtain the following classification depending on the nature of the set P of productions:

1. Type 3 grammars: These are regular sets, which are known to have left-linear grammars (Every production is of the type $A \rightarrow Bw$ or $A \rightarrow w$ where A, B are non-terminals and $w \in \Sigma^*$) as well as right-linear grammars (Every production is of the type $A \rightarrow wB$ or $A \rightarrow b$ where A, B are non-terminals and $w \in \Sigma^*$). Type 3 grammars have an equivalence with finite automata: A set has a Type 3 grammar if and only if it is accepted by a finite automaton.

2. Type 2 grammars: These are CFGs. Due to Chomsky Normal form we can assume that each production is of the form $A \rightarrow b$ or $A \rightarrow B_1B_2$ with A, B_1, B_2 being nonterminals and $b \in \Sigma \cup \{\epsilon\}$. Type 2 grammars have an equivalence with PDAs: A set has a Type 2 grammar if and only if it is accepted by a nondeterministic PDA.

3. Type 1 grammars: These are called context sensitive grammars. We have not covered this type in this course. Productions for this type of grammar are of the form $uXw \rightarrow uvw$ where u, v, w are arbitrary strings of $(\Sigma \cup \Gamma)^*$, with v non-null, and X a single nonterminal. In other words, X may be replaced by v but only when it is surrounded by u and w; i.e., in a particular context. There is an equivalent machine model for these grammars called linear bounded automata. This is just for your general knowledge and to complete the picture.

4. Type 0 grammars: Here the productions are of the form $\alpha \rightarrow \beta$, where $\alpha, \beta \in (\Sigma \cup \Gamma)^*$ and $\alpha \neq \epsilon$. We will see in this Lecture that Type 0

grammars are equivalent to recursively enumerable sets and they have equivalent Turing machines accepting them.

There is a reason why the numbering is done from 3 to 0. It is mainly to denote that the Type 0 grammars are unrestricted, or have zero restrictions! As we move towards Type 3, more and more restrictions are included on the form of the productions.

Recall that $\{a^n b^n | n \geq 0\}$ cannot have a Type 3 grammar but has a Type 2 grammar as it is a CFL. Similarly, we have seen that $\{a^n b^n c^n | n \geq 0\}$ cannot have a Type 2 grammar but by the end of this Lecture we will be convinced that this language has a Type 0 grammar as it is a r.e set.

2 A Type 0 Grammar has a Turing Machine accepting it

For a Type 0 grammar $G = (\Gamma, \Sigma, P, S)$ we construct a nondeterministic 2-tape Turing machine M. This is equivalent to a deterministic 1-tape Turing machine. On the first tape, M writes down the input string w from Σ^*. In the second tape, the intermediate sentential form α will be stored. Initially, the second tape of M contains the start symbol S. Then M does the following:

1. Nondeterministically guess a position i between 1 and $|\alpha|$ - this can be done by nondeterministically choosing to either move right or stay in the same place (which involves a consecutive right-left move combination).
2. Nondeterministically select a production $\beta \rightarrow \gamma$ from P.
3. If β is a string that starts at the i^{th} position of α then α is transformed so that β is replaced by γ. Depending on the comparison between the lengths of β and γ, some parts of the string may need to shift to the left or right in order to complete this task.
4. when α is a sentence, then M compares α to w and accepts if the two are the same.

On one hand, using induction, it is possible to show that if w belongs to $L(G)$ then there is a sequence of guesses that will mimic the derivation of w from S.

On the other hand, if w is in $L(M)$ then there is a sequence of guesses that imply a derivation from S to w (Use induction to formally prove this). Thus, $w \in L(G)$ as well.

Therefore, $L(G) = L(M)$.

3 The language of a Turing Machine has a Type 0 Grammar

Now, given a Turing machine $M = (Q, \Sigma, \Gamma, \text{"}left{-}endmarker\text{"} = \vdash, \delta, s, \text{"}blank{-}space\text{"} = B, t, r)$, we design a Type 0 grammar $G = (\Gamma', \Sigma, P, S)$ deriving exactly $L(M)$. Notice that M takes an input x and has the ability to overwrite on x. If we wish to kind of mimic M's operations when building G we must ultimately derive x from S. One way of doing this is to make two copies of x. One copy stays intact no matter what production we apply, while on the other copy the productions try to simulate transitions applied by M. At some point M accepts the string by entering the state t. At this point, there should be productions in G that forget the second copy and produce the original x as the final sentence.

We also need to remember the current state and tape head position for M while deriving the input x in the grammar G. It is possible to do this by somehow modifying the copy of x on which the grammar is simulating M. For example if at some point the tape contents in $\gamma = a_1a_2\ldots a_\ell$, the state is q and the tape head is at a_i then we can remember all this in the string $\gamma' = a_1a_2\ldots a_{i-1}qa_i\ldots a_\ell$. The exact formulation will be similar to this idea.

Now let us formalise this idea.

We take $\Gamma' = ((\Sigma \cup \{\epsilon\}) \times \Gamma) \cup \{S, A_2, A_3\}$.

The productions in P are:

1. $S \to sA_2$
2. $A_2 \to [a,a]A_2$ for each $a \in \Sigma$
3. $A_2 \to A_3$
4. $A_3 \to [\epsilon, B]A_3$
5. $A_3 \to \epsilon$
6. $q[a,X] \to [a,Y]p$ for each $a \in \Sigma \cup \{\epsilon\}$, $q,p \in Q$, $X,Y \in \Gamma$, and such that $\delta(q,X) = (p,Y,R)$ is a transition for M.
7. $[b,Z]q[a,X] \to p[b,Z][a,Y]p$ for each $a,b \in \Sigma \cup \{\epsilon\}$, $q,p \in Q$, $X,Y,Z \in \Gamma$, and such that $\delta(q,X) = (p,Y,L)$ is a transition for M.
8. $[a,X]t \to tat, t[a,X] \to tat, t \to \epsilon$ for each $a \in \Sigma \cup \{\epsilon\}, X \in \Gamma$.

Now let us see how this works.

For the input x, using productions 1 and 2 we can derive

$$S \implies {}^* s[x_1,x_1][x_2,x_2]\ldots[x_n,x_n]A_2$$

.

Now, if M accepts x then there is a constant c such that M does not go beyond c cells on its tape. If we use production 3, then production 4 c times, and finally production 5 once we derive

$$S \implies {}^* s[x_1,x_1][x_2,x_2]\ldots[x_n,x_n][\epsilon,B]^c$$

.

Note that after this only productions $6, 7$ can be used until the accepting state t is generated. Note that production 6 aims at simulating a right tape-head move while production 7 aims at simulating a left tape-head move by M. As the first component of the variables in $(\Sigma \cup \{\epsilon\} \times \Gamma)$ never changes, we are preserving a copy of the input x throughout all these productions.

Using induction on the number of moves, it can be shown that in M if $(s, x, 0) \rightarrow_M^* (q, X_1X_2 \ldots X_{r-1}X_r \ldots X_\ell, r)$ then

$$s[x_1, x_1][x_2, x_2] \ldots [x_n, x_n][\epsilon, B]^c \implies {}_G^*[x_1, X_1][x_2, X_2] \ldots [x_{r-1}, X_{r-1}]q[x_r, X_r] \ldots [x_{n+c}, X_{n+c}]$$

Here, for all $j > n, x_j = \epsilon$ and for all $j > \ell, X_j = B$.

Finally, if the following has been derived:

$$s[x_1, x_1][x_2, x_2] \ldots [x_n, x_n][\epsilon, B]^c \implies {}_G^*[x_1, X_1][x_2, X_2] \ldots [x_{r-1}, X_{r-1}]t[x_r, X_r] \ldots [x_{n+c}, X_{n+c}]$$

then we apply production 8 multiple times to get the following:

$$[x_1, X_1][x_2, X_2] \ldots [x_{r-1}, X_{r-1}]t[x_r, X_r] \ldots [x_{n+c}, X_{n+c}] \implies {}_G^* x_1x_2 \ldots x_n$$

.

This means that if $x \in L(M)$ then $x \in L(G)$.

Lastly, we need to show that any sentence generated by G must be accepted by M. Use the definitions of productions of G as well as an induction on the number of steps of derivation to formally show that G must simulate an accepting computation of M if it derives a sentence from S.

Thus, we have shown that $L(G) = L(M)$.

This Section and the previous one completes the proof that a set has a Type 0 grammar if and only if it is an r.e set, and has a Turing machine accepting it.

4 Example

In the previous section, we have seen an algorithm to convert a Turing machine into a Type 0 grammar. The aim was to mimic the Turing machine as productions. Sometimes, depending on the problem, this mimicking can be done with a smaller/more intuitive set of productions. This is the case in the following example.

Consider $\{a^{2^n} | n \geq 0\}$. We have seen that this is recursively enumerable. Now we see a Type 0 grammar for this language.

The set of productions are the following:

1. $S \rightarrow ACaB$,
2. $Ca \rightarrow aaC$,
3. $CB \rightarrow DB$,

4. $CB \to E$,

5. $aD \to Da$,

6. $AD \to AC$,

7. $aE \to Ea$,

8. $AE \to \epsilon$.

The nonterminals A, B will behave as left and right endmarkers for the sentential form a^{2^n}. C can be thought of as the tape head that moves through the current string of a's generated between A and B and doubling their numbers by using production 2. When the right endmarker is reached then C changes to D or E through productions 3 or 4. If D is chosen, then using production 5, it moves left till left endmarker A is reached. At this point, production 6 is used to change D to C and repeat the doubling of the current string. Suppose E is chosen, then in production 4 the right endmarker B is removed and then by production 7 E moves to the left. When it hits the left endmarker A, it removes A. The sentential form this obtained is a^{2^n}. You can use induction on the number of steps of derivation to give a formal proof.