EXERCISES

1. Prove that the set of all real numbers is not countable.
2. Prove that the set of all languages that are not recursively enumerable is not countable. (S)
3. Let L be a finite language. Show that then L^+ is recursively enumerable. Suggest an enumeration procedure for L^+.
4. Let L be a context-free language. Show that L^+ is recursively enumerable and suggest an enumeration procedure for it.
5. Show that if a language is not recursively enumerable, its complement cannot be recursive.
6. Show that the family of recursively enumerable languages is closed under union. (S)
7. Is the family of recursively enumerable languages closed under intersection?
8. Show that the family of recursive languages is closed under union and intersection.
9. Show that the families of recursively enumerable and recursive languages are closed under reversal.
10. Is the family of recursive languages closed under concatenation?
11. Prove that the complement of a context-free language must be recursive. (S)
12. Let L_1 be recursive and L_2 recursively enumerable. Show that $L_2 - L_1$ is necessarily recursively enumerable.
13. Suppose that L is such that there exists a Turing machine that enumerates the elements of L in proper order. Show that this means that L is recursive.
14. If L is recursive, is it necessarily true that L^+ is also recursive? (S)
15. Choose a particular encoding for Turing machines, and with it, find one element of the language $\overline{L}$ in Theorem 11.3.
16. Let S_1 be a countable set, S_2 a set that is not countable, and $S_1 \subset S_2$. Show that S_2 must then contain an infinite number of elements that are not in S_1.
17. In Exercise 16, show that in fact $S_2 - S_1$ cannot be countable.
18. Why does the argument in Theorem 11.1 fail when S is finite? (S)
19. Show that the set of all irrational numbers is not countable.

11.2 Unrestricted Grammars

To investigate the connection between recursively enumerable languages and grammars, we return to the general definition of a grammar in Chapter 1. In Definition 1.1 the production rules were allowed to take any form,

but various restrictions were later made to get specific grammar types. If we take the general form and impose no restrictions, we get unrestricted grammars.

Definition 11.3

A grammar $G = (V, T, S, P)$ is called **unrestricted** if all the productions are of the form

$$u \to v,$$

where u is in $(V \cup T)^+$ and v is in $(V \cup T)^*$.

In an unrestricted grammar, essentially no conditions are imposed on the productions. Any number of variables and terminals can be on the left or right, and these can occur in any order. There is only one restriction: λ is not allowed as the left side of a production.

As we will see, unrestricted grammars are much more powerful than restricted forms like the regular and context-free grammars we have studied so far. In fact, unrestricted grammars correspond to the largest family of languages so we can hope to recognize by mechanical means; that is, unrestricted grammars generate exactly the family of recursively enumerable languages. We show this in two parts; the first is quite straightforward, but the second involves a lengthy construction.

Theorem 11.6

Any language generated by an unrestricted grammar is recursively enumerable.

Proof: The grammar in effect defines a procedure for enumerating all strings in the language systematically. For example, we can list all w in L such that

$$S \Rightarrow w,$$

that is, w is derived in one step. Since the set of the productions of the grammar is finite, there will be a finite number of such strings. Next, we list all w in L that can be derived in two steps

$$S \Rightarrow x \Rightarrow w,$$

and so on. We can simulate these derivations on a Turing machine and, therefore, have an enumeration procedure for the language. Hence it is recursively enumerable. ■

This part of the correspondence between recursively enumerable languages and unrestricted grammars is not surprising. The grammar generates strings by a well-defined algorithmic process, so the derivations can be done on a Turing machine. To show the converse, we describe how any Turing machine can be mimicked by an unrestricted grammar.

We are given a Turing machine $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ and want to produce a grammar G such that $L(G) = L(M)$. The idea behind the construction is relatively simple, but its implementation becomes notationally cumbersome.

Since the computation of the Turing machine can be described by the sequence of instantaneous descriptions

$$q_0 w \overset{*}{\vdash} x q_f y, \tag{11.3}$$

we will try to arrange it so that the corresponding grammar has the property that

$$q_0 w \overset{*}{\Rightarrow} x q_f y \tag{11.4}$$

if and only if (11.3) holds. This is not hard to do; what is more difficult to see is how to make the connection between (11.4) and what we really want, namely,

$$S \overset{*}{\Rightarrow} w$$

for all w satisfying (11.3). To achieve this, we construct a grammar which, in broad outline, has the following properties:

1. S can derive $q_0 w$ for all $w \in \Sigma^+$.
2. (11.4) is possible if and only if (11.3) holds.
3. When a string $x q_f y$ with $q_f \in F$ is generated, the grammar transforms this string into the original w.

The complete sequence of derivations is then

$$S \overset{*}{\Rightarrow} q_0 w \overset{*}{\Rightarrow} x q_f y \overset{*}{\Rightarrow} w. \tag{11.5}$$

The third step in the above derivation is the troublesome one. How can the grammar remember w if it is modified during the second step? We solve this by encoding strings so that the coded version originally has two copies of w. The first is saved, while the second is used in the steps in (11.4). When a final configuration is entered, the grammar erases everything except the saved w.

To produce two copies of w and to handle the state symbol of M (which eventually has to be removed by the grammar), we introduce variables V_{ab}

and V_{aib} for all $a \in \Sigma \cup \{\square\}$, $b \in \Gamma$, and all i such that $q_i \in Q$. The variable V_{ab} encodes the two symbols a and b, while V_{aib} encodes a and b as well as the state q_i.

The first step in (11.5) can be achieved (in the encoded form) by

$$S \to V_{\square\square}S \,|\, SV_{\square\square} \,|\, T, \tag{11.6}$$

$$T \to TV_{aa} | V_{a0a}, \tag{11.7}$$

for all $a \in \Sigma$. These productions allow the grammar to generate an encoded version of any string q_0w with an arbitrary number of leading and trailing blanks.

For the second step, for each transition

$$\delta(q_i, c) = (q_j, d, R)$$

of M, we put into the grammar productions

$$V_{aic}V_{pq} \to V_{ad}V_{pjq}, \tag{11.8}$$

for all $a, p \in \Sigma \cup \{\square\}$, $q \in \Gamma$. For each

$$\delta(q_i, c) = (q_j, d, L)$$

of M, we include in G

$$V_{pq}V_{aic} \to V_{pjq}V_{ad}, \tag{11.9}$$

for all $a, p \in \Sigma \cup \{\square\}$, $q \in \Gamma$.

If in the second step, M enters a final state, the grammar must then get rid of everything except w, which is saved in the first indices of the V's. Therefore, for every $q_j \in F$, we include productions

$$V_{ajb} \to a, \tag{11.10}$$

for all $a \in \Sigma \cup \{\square\}$, $b \in \Gamma$. This creates the first terminal in the string, which then causes a rewriting in the rest by

$$cV_{ab} \to ca, \tag{11.11}$$

$$V_{ab}c \to ac, \tag{11.12}$$

for all $a, c \in \Sigma \cup \{\square\}$, $b \in \Gamma$. We need one more special production

$$\square \to \lambda. \tag{11.13}$$

This last production takes care of the case when M moves outside that part of the tape occupied by the input w. To make things work in this case, we must first use (11.6) and (11.7) to generate

$$\square \ldots \square q_0 w \square \ldots \square,$$

representing all the tape region used. The extraneous blanks are removed at the end by (11.13).

The following example illustrates this complicated construction. Carefully check each step in the example to see what the various productions do and why they are needed.

Example 11.1 Let $M = (Q, \Sigma, \Gamma, \delta, q_0, \square, F)$ be a Turing machine with

$$\begin{aligned} Q &= \{q_0, q_1\}, \\ \Gamma &= \{a, b, \square\}, \\ \Sigma &= \{a, b\}, \\ F &= \{q_1\}, \end{aligned}$$

and

$$\begin{aligned} \delta(q_0, a) &= (q_0, a, R), \\ \delta(q_0, \square) &= (q_1, \square, L). \end{aligned}$$

This machine accepts $L(aa^*)$.

Consider now the computation

$$q_0aa \vdash aq_0a \vdash aaq_0\square \vdash aq_1a\square, \tag{11.14}$$

which accepts the string aa. To derive this string with G, we first use rules of the form (11.6) and (11.7) to get the appropriate starting string,

$$S \Rightarrow SV_{\square\square} \Rightarrow TV_{\square\square} \Rightarrow TV_{aa}V_{\square\square} \Rightarrow V_{a0a}V_{aa}V_{\square\square}.$$

The last sentential form is the starting point for the part of the derivation that mimics the computation of the Turing machine. It contains the original input $aa\square$ in the sequence of first indices and the initial instantaneous description $q_0aa\square$ in the remaining indices. Next, we apply

$$V_{a0a}V_{aa} \to V_{aa}V_{a0a},$$

and

$$V_{a0a}V_{\square\square} \to V_{aa}V_{\square0\square},$$

which are specific instances of (11.8), and

$$V_{aa}V_{\square0\square} \to V_{a1a}V_{\square\square}$$

coming from (11.9). Then the next steps in the derivation are

$$V_{a0a}V_{aa}V_{\square\square} \Rightarrow V_{aa}V_{a0a}V_{\square\square} \Rightarrow V_{aa}V_{aa}V_{\square0\square} \Rightarrow V_{aa}V_{a1a}V_{\square\square}.$$

Theorem 11.7

The sequence of first indices remains the same, always remembering the initial input. The sequence of the other indices is

$$0aa\Box, a0a\Box, aa0\Box, a1a\Box,$$

which is equivalent to the sequence of instantaneous descriptions in (11.14).

Finally, (11.10) to (11.13) are used in the last steps

$$V_{aa}V_{a1a}V_{\Box\Box} \Rightarrow V_{aa}aV_{\Box\Box} \Rightarrow V_{aa}a\Box \Rightarrow aa\Box \Rightarrow aa.$$

The construction described in (11.6) to (11.13) is the basis of the proof of the following result.

Theorem 11.7 For every recursively enumerable language L, there exists an unrestricted grammar G, such that $L = L(G)$.

Proof: The construction described guarantees that

$$x \vdash y,$$

then

$$e(x) \Rightarrow e(y),$$

where $e(x)$ denotes the encoding of a string according to the given convention. By an induction on the number of steps, we can then show that

$$e(q_0 w) \overset{*}{\Rightarrow} e(y)$$

if and only if

$$q_0 w \overset{*}{\vdash} y.$$

We also must show that we can generate every possible starting configuration and that w is properly reconstructed if and only if M enters a final configuration. The details, which are not too difficult, are left as an exercise. ■

These two theorems establish what we set out to do. They show that the family of languages associated with unrestricted grammars is identical with the family of recursively enumerable languages.