Chapter 8

Recursively Enumerable Languages

Recursively Enumerable Languages

- *Recursively enumerable* (r.e.) languages are those that can be accepted by a TM
- *Recursive* languages are those that can be decided by a TM
- Only in the second case are we guaranteed an answer to the question: Given a string *x*, is *x* an element of the language?
- We will study the relationships between these two kinds of languages, and we will see that there are languages that are not r.e., as well as languages that are r.e. but not recursive

Recursively Enumerable and Recursive

- Definition 8.1: A TM T with input alphabet Σ *accepts* a language $L \subseteq \Sigma^*$ if it accepts the strings in L and no others
- *T decides L* if T computes the characteristic function $\chi_L : \Sigma^* \to \{0,1\}$ that has the value 1 at strings in L and the value 0 otherwise
- In both cases, the issue is whether the input string is an element of L. The second approach may be more informative, however, because a TM accepting L may not return an answer if the string is not in L

Recursively Enumerable and Recursive (cont'd.)

- Theorem 8.2: Every recursive language is recursively enumerable
- Theorem 8.3: If $L \subseteq \Sigma^*$ is accepted by a TM T that halts on every input string, then L is recursive
- Theorem 8.4: If L_1 and L_2 are both recursively enumerable languages over Σ, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursively enumerable
- Theorem 8.5: If L_1 and L_2 are both recursive languages over Σ, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursive

Recursively Enumerable and Recursive (cont'd.)

- Theorem 8.6: If L is a recursive language over Σ, then its complement L' is also recursive
- Theorem 8.7: If L is a recursively enumerable language, and its complement L' is also recursively enumerable, then L is recursive

Enumerating a Language

- Definition 8.8: Let T be a k-tape Turing machine for some $k \geq 1$, and let $L \subseteq \Sigma^*$. We say *T enumerates L* if it operates such that the following conditions are satisfied:
  - The tape head on the first tape never moves to the left, and no nonblank symbol printed on tape 1 is subsequently modified or erased
  - For every $x \in L$, there is some point during the operation of T when tape 1 has contents $x_1 \# x_2 \# \ldots \# x_n \# x \#$ for some $n \geq 0$, where the x_i's are also elements of L and $x_1, x_2, \ldots, x_n, x$ are distinct
  - If L is finite, then nothing is printed after the # following the last element of L

Enumerating a Language (cont'd.)

- Theorem 8.9: For every language $L \subseteq \Sigma^*$, L is recursively enumerable if and only if there is a TM enumerating L, and L is recursive if and only if there is a TM that enumerates the strings in L in canonical order

Enumerating a Language (cont'd.)

- Proof: we need to show these four things
  - If there is a TM that accepts L then there's one that enumerates L
  - If there is a TM that enumerates L then there's one that accepts it
  - If there is a TM that decides L, then there is a TM that enumerates L in canonical order
  - If there is a TM that enumerates L in canonical order then there is a TM that decides L

Enumerating a Language (cont'd.)

- To prove these four statements, we'll make use of the Church-Turing sequence and just describe an algorithm, rather than specifying a TM in detail
- We start with the third statement
  - If T decides L, then for any string x, we can give x to T and wait for it to give an answer
  - The algorithm for enumerating L is to consider the strings of Σ^* in canonical order, and for each one, add it to the output only if T says "yes"

Enumerating a Language (cont'd.)

- For statement 1, we consider the strings in canonical order, but "waiting for T to give an answer" might mean waiting forever, so we make repeated passes
  - On each pass, we examine one more string, and for each string that we're still unsure about, we consider one more step in T's processing of that string
- Statement 2 is easy
  - If T enumerates L then an algorithm to accept L is: watch the computation of T and accept x precisely if T lists x (this algorithm never returns an answer if $x \notin L$)
- Statement 4 is almost as easy:
  - Watch the computation until either x or a string greater than x is listed; in the first case, x is in L, and in the second case it isn't

More General Grammars

- Definition 8.10: An *unrestricted* grammar is a 4-tuple $G=(V, \Sigma, S, P)$, where V and Σ are disjoint sets of variables and terminals, respectively
  - Just as in a CFG, S is the start variable and P is a set of productions. Here, however, a production can have the form $\alpha \rightarrow \beta$, where $\alpha, \beta \in (V \cup \Sigma)^*$ and α is any string containing at least one variable
- It will turn out that unrestricted grammars correspond to recursively enumerable languages, just as CFGs correspond to PDAs and regular grammars to FAs

More General Grammars (cont'd.)

- We can continue to use much of the notation developed for CFGs, but one important difference is that the assumption $S \Rightarrow^* xAy \Rightarrow^* z$ no longer implies that $z = xwy$ for some string w
- Theorem 8.13: For every unrestricted grammar G, there is a Turing machine T with $L(T) = L(G)$
  - Proof: construct a TM that accepts $L(G)$
  - It will be simpler to build a nondeterministic TM

More General Grammars (cont'd.)

- The TM will contain all the variables and terminals in G and works as follows
  - It moves the tape head to the blank square following the input string
  - During the second phase of its operation, T treats this blank square as if it were the beginning of the tape, and the input string is undisturbed
- T simulates a derivation in G nondeterministically as follows:

More General Grammars (cont'd.)

- First the symbol *S* is written in the square following the blank
- Each subsequent step involves
  - Choosing a production $\alpha \rightarrow \beta$ in the grammar
  - Selecting an occurrence of α, if there is one, in the string currently on the tape
  - Replacing the occurrence of α by β, which may mean moving the rest of the string to the right or left
- The final phase is to compare the two strings on the tape and to accept if and only if they are equal

More General Grammars (cont'd.)

- Theorem 8.14: For every TM T with input alphabet Σ, there is an unrestricted grammar generating the language $L(T) \subseteq \Sigma^*$
  - Proof: for simplicity assume that $\Sigma = \{a, b\}$. The grammar will have three types of productions:
  - Those of type 1 are

    $S \to S(\Delta\Delta) \mid T \qquad T \to T(aa) \mid T(bb) \mid q_0(\Delta\Delta)$

    which generate strings $q_0(\Delta\Delta)(\sigma_1\sigma_1)(\sigma_2\sigma_2)\ldots(\sigma_k\sigma_k)$ followed by zero or more copies of $(\Delta\Delta)$. Each pair of σ's is thought of as two copies of a symbol in $x \in \Sigma^*$

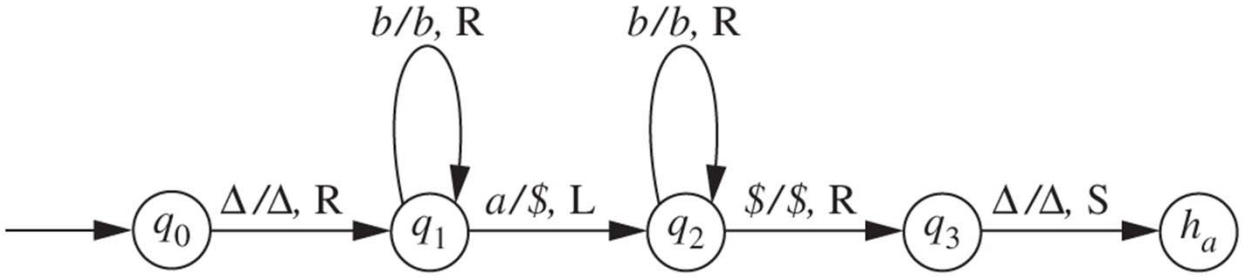
More General Grammars (cont'd.)

- Proof (cont'd.)
  - Productions of type 2 allow the moves of T to be simulated on the second copy x_2 of x, while keeping the first copy x_1 unchanged. (The additional variables in the string, which include state names, Δ, and parentheses, prevent the derivation from producing a string of terminals before it has been determined whether T accepts x.)
  - Productions of type 3 allow everything in the string except x_1 to be erased, *provided* that the computation of T reaches h_a

More General Grammars (cont'd.)

- To illustrate the productions of types 2 and 3, consider the sample transition diagram of T below
- Productions for the first transition, for example, are
  $q_0(\Delta\Delta) \rightarrow (\Delta\Delta)q_1 \quad q_0(a\Delta) \rightarrow (a\Delta)q_1 \quad q_0(b\Delta) \rightarrow (b\Delta)q_1$
- Productions for the third transition look like

  $(\sigma\ \rho)\ q_1(\tau\ a) \rightarrow q_2\ (\sigma\ \rho)\ (\tau\ \$)$, where $\sigma, \tau \in \{a, b, \Delta\}$ and ρ is any tape symbol of T. These are more complex because the transition involves a move to the left



More General Grammars (cont'd.)

- We show the moves corresponding to the simulation of T on the input ba. At each step, the underlined portion is the part used in the next production.

$$\begin{aligned}
&\underline{q_0(\Delta\Delta)}(bb)(aa)(\Delta\Delta) \vdash (\Delta\Delta)\underline{q_1(bb)}(aa)(\Delta\Delta)\\
&\vdash (\Delta\Delta)\underline{(bb)q_1(aa)}(\Delta\Delta) \vdash (\Delta\Delta)\underline{q_2(bb)}(a\$)(\Delta\Delta)\\
&\vdash (\Delta\Delta)(bb)\underline{q_2(a\$)}(\Delta\Delta) \vdash (\Delta\Delta)(bb)(a\$)\underline{q_3(\Delta\Delta)}\\
&\vdash (\Delta\Delta)(bb)(a\$)h_a(\Delta\Delta)
\end{aligned}$$

- The appearance of h_a in the derivation is what will allow everything except x_1, the original input string, to disappear

More General Grammars (cont'd.)

- Productions like $(\sigma_1\,\sigma_2)\,h_a \rightarrow h_a\,(\sigma_1\,\sigma_2)\,h_a$ and $h_a\,(\sigma_1\,\sigma_2) \rightarrow h_a\,(\sigma_1\,\sigma_2)\,h_a$ allow the copies of h_a to "propagate"
- Productions like

  $h_a\,(a\,\sigma_2) \rightarrow a \quad h_a\,(b\,\sigma_2) \rightarrow b \quad h_a\,(\Delta\,\sigma_2) \rightarrow \Lambda$

  allow us to eliminate everything but ba, which is accepted
- Here is the rest of the derivation:

$$
\begin{array}{rcl}
(\Delta\Delta)(bb)(a\$)h_a(\Delta\Delta) & \vdash & (\Delta\Delta)(bb)h_a(a\$)h_a(\Delta\Delta) \\
\vdash (\Delta\Delta)\,h_a(bb)\,h_a(a\$)h_a(\Delta\Delta) & \vdash & h_a\,(\Delta\Delta)h_a(bb)h_a(a\$)h_a(\Delta\Delta) \\
\vdash \quad h_a(bb)h_a(a\$)h_a(\Delta\Delta) & \vdash & b\,h_a\,(a\$)h_a(\Delta\Delta) \\
\vdash \quad bah_a(\Delta\Delta) & \vdash & ba
\end{array}
$$

Context-Sensitive Languages and the Chomsky Hierarchy

- Definition: A context-sensitive grammar (CSG) is an unrestricted grammar in which no production is length-decreasing
  - In other words, every production is of the form $\alpha \rightarrow \beta$, where $|\beta| \geq |\alpha|$
- A language is a context-sensitive language (CSL) if it can be generated by a CSG
- CSGs cannot have Λ-productions, and CSLs cannot include Λ
- We think of CSLs as a generalization of CFLs

Context-Sensitive Languages and the Chomsky Hierarchy (cont'd.)

- Definition 8.18: A linear-bounded automaton (LBA) is a nondeterministic TM with this exception:
  - There are two extra tape symbols, [and]
  - The initial configuration of M corresponding to input x is $q_0[x]$
  - During its computation, M is not permitted to replace either of these brackets or to move its tape head to the left of the [or to the right of the]

Context-Sensitive Languages and the Chomsky Hierarchy (cont'd.)

- Theorem 8.19: If $L \subseteq \Sigma^*$ is a CSL, then there is an LBA that accepts L
  - We can follow the proof of Theorem 8.13, except that instead of being able to use the space to the right of the input string on the tape, the LBA must use the space between the two brackets
  - It can do this by converting individual symbols into symbol pairs so as to simulate two tape "tracks"

Context-Sensitive Languages and the Chomsky Hierarchy (cont'd.)

- If $L \subseteq \Sigma^*$ is accepted by a LBA M, then there is a CSG generating L - $\{\Lambda\}$
- The proof is similar to that of Theorem 8.14
  - For details, see book
- The four levels of language we have seen so far correspond to the *Chomsky Hierarchy*

Context-Sensitive Languages and the Chomsky Hierarchy (cont'd.)

- The Chomsky Hierarchy

Type	Languages (Grammars)	Form of Productions	Accepting Device
3	Regular	$A \to aB, \quad A \to \Lambda$	Finite Automaton
2	Context-free	$A \to \alpha$	Pushdown Automaton
1	Context-sensitive	$\alpha \to \beta$ with $\lvert\beta\rvert \geq \lvert\alpha\rvert$	LBA
0	Unrestricted	$\alpha \to \beta$	Turing machine

Context-Sensitive Languages and the Chomsky Hierarchy (cont'd.)

- Theorem 8.22: Every CSL is recursive
- Proof: Let G be a CSG generating L
  - Theorem 8.19 says that there is an LBA M accepting L
  - It suffices to show that there is a nondeterministic TM T_1 accepting L such that no input string can possibly cause T_1 to loop forever
  - We may consider M to be a nondeterministic TM T, which begins by inserting the markers [and] in the squares where they would be already if we were thinking of T as an LBA

Context-Sensitive Languages and the Chomsky Hierarchy (cont'd.)

- T_1 is constructed as a modification of T
- T_1 also begins by placing the markers on the tape
- Just like T, T_1 performs the first iteration in a simulated derivation in G by writing S in the first position of the second track
- Before the second iteration, however, it moves to the blank portion of the tape after the right marker and records the string S obtained in the first iteration

Context-Sensitive Languages and the Chomsky Hierarchy (cont'd.)

- In each subsequent iteration it performs the following four steps (the first three the same as *T*)
  - It selects a production $\alpha \rightarrow \beta$
  - It attempts to select an occurrence of α in the current string; if it is unable to, it compares the current string to the input *x* and accepts if they're equal, rejects if they're not

Context-Sensitive Languages and the Chomsky Hierarchy (cont'd.)

- Four steps (cont'd.)
  - If it can select an occurrence of α, it replaces it by β, and rejects if this would result in a string longer than the input
  - If it successfully replaces α by β, it compares the new current string with the strings it has written in the portion of the tape to the right of]; it rejects if there's a match, and otherwise writes the new string after the most recent entry

Not Every Language is Recursively Enumerable

- We will now consider languages over an alphabet Σ and TMs with input alphabet Σ
  - We will show that there are more languages than TMs to accept them. It follows that there must be many languages not accepted by any TM
- The first step is to explain how it makes sense to talk about one infinite set being larger than another
- We'll formulate two definitions
  - What it means for two sets to be the same size
  - What it means for one to be larger than another

Not Every Language is Recursively Enumerable (cont'd.)

- For finite sets, this is easy, because we know how to say that one *number* is equal to or bigger than another.
- Definition 8.23: In general, two sets A and B are the same size if there is a bijection $f: A \to B$. A is larger than B if some subset of A is the same size as B but A itself is not
- Definition 8.24: A set A is *countably infinite* if there is a bijection $f: \mathbb{N} \to A$, or a list $a_0, a_1, \ldots$ of elements of A such that every element of A appears exactly once in the list
  - A is *countable* if A is either finite or countably infinite

Not Every Language is Recursively Enumerable (cont'd.)

- Theorem 8.25: Every infinite set has a countably infinite subset, and every subset of a countable set is countable
  - For proof, see book
- Even though the set $\mathbb{N} \times \mathbb{N}$ seems much larger than $\mathbb{N}$, it is countable and therefore the same size as $\mathbb{N}$
- We can see this by forming a two-dimensional array with all the ordered pairs (i, j) and starting a spiral path at (0,0) that hits each ordered pair. This is a way of listing the elements of $\mathbb{N} \times \mathbb{N}$

Not Every Language is Recursively Enumerable (cont'd.)

- The set $2^{\mathbb{N}}$ is uncountable. The proof uses a *diagonalization* argument
  - For every list $A_0, A_1, \ldots$ of subsets of $\mathbb{N}$, we can define the set A as follows:

$$A = \{n \in \mathbb{N} \mid n \notin A_n\}$$

  - Then for every $n \in \mathbb{N}$, $A \neq A_n$, because n is an element of A if and only if it is not an element of A_n
  - The conclusion is that A cannot be one of the subsets in the list, which means that there can be no list containing all the subsets of $\mathbb{N}$

Not Every Language is Recursively Enumerable (cont'd.)

- Here's the reason we call this argument a diagonal argument:
  - Subsets of $\mathbb{N}$ can be represented by infinite sequences of 0's and 1's. For example,

$$\begin{array}{rl} \varnothing & \leftrightarrow\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ \ldots \\ \mathbb{N} & \leftrightarrow\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ \ldots \\ \{0, 3, 4\} & \leftrightarrow\ 1\ 0\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ \ldots \\ \{1, 3, 5, \ldots\} & \leftrightarrow\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ \ldots \end{array}$$

- Let each set A_i in the list be represented as follows:

$$A_i \ \leftrightarrow\ a_{i,0}\ a_{i,1}\ a_{i,2}\ a_{i,3}\ \ldots$$

Not Every Language is Recursively Enumerable (cont'd.)

- Then the set A was constructed by looking at the diagonal entries (underlined)

$$
\begin{aligned}
A_0 \leftrightarrow \quad & \underline{a}_{0,0} \quad a_{0,1} \quad a_{0,2} \quad a_{0,3} \quad a_{0,4} \quad \ldots \\
A_1 \leftrightarrow \quad & a_{1,0} \quad \underline{a}_{1,1} \quad a_{1,2} \quad a_{1,3} \quad a_{1,4} \quad \ldots \\
A_2 \leftrightarrow \quad & a_{2,0} \quad a_{2,1} \quad \underline{a}_{2,2} \quad a_{2,3} \quad a_{2,4} \quad \ldots \\
A_3 \leftrightarrow \quad & a_{3,0} \quad a_{3,1} \quad a_{3,2} \quad \underline{a}_{3,3} \quad a_{3,4} \quad \ldots
\end{aligned}
$$

  and reversing them. For example, A contains 0 if and only if A_0 doesn't; i.e., $A \leftrightarrow a_0 \quad a_1 \quad a_2 \quad a_3 \quad \ldots$ where a_0 is the opposite of $a_{0,0}$, and in general, a_i is the opposite of $a_{i,i}$

Not Every Language is Recursively Enumerable (cont'd.)

- Theorem 8.32: Not all languages are recursively enumerable
  - In fact, the set of languages over {0,1} that are not recursively enumerable is uncountable
- We showed that the set of subsets of $\mathbb{N}$ is uncountable, and we observed that because {0,1}* is the same size as $\mathbb{N}$, it follows that the set of languages over {0,1} is uncountable
- But the set of recursively enumerable languages over {0,1} is countable: each one can be accepted by a TM, and we can list the TMs T by listing the strings $e(T)$ that represent them