Lecture 35

Undecidable Problems About CFLs

In this lecture we show that a very simple problem about CFLs is undecidable, namely the problem of deciding whether a given CFG generates all strings.

It is decidable whether a given CFG generates any string at all, since we know by the pumping lemma that a CFG G that generates any string at all must generate a short string; and we can determine for all short strings x whether $x \in L(G)$ by the CKY algorithm.

This decision procedure is rather inefficient. Here is a better one. Let $G = (N, \Sigma, P, S)$ be the given CFG. To decide whether $L(G)$ is nonempty, we will execute an inductive procedure that marks a nonterminal when it is determined that that nonterminal generates some string in Σ^*—any string at all—and when we are done, ask whether the start symbol S is marked.

At stage 0, mark all the symbols of Σ. At each successive stage, mark a nonterminal $A \in N$ if there is a production $A \to \beta \in P$ and all symbols of β are marked. Quit when there are no more changes; that is, when for each production $A \to \beta$, either A is marked or there is an unmarked symbol of β. This must happen after a finite time, since there are only finitely many symbols to mark.

It can be shown that A is marked by this procedure if and only if there is a string $x \in \Sigma^*$ such that $A \xrightarrow[G]{*} x$. This can be proved by induction, the implication $\Rightarrow$ by induction on the stage that A is marked, and the implication $\Leftarrow$ by induction on the length of the derivation $A \xrightarrow[G]{*} x$.

Then $L(G)$ is nonempty iff there exists an $x \in \Sigma^*$ such that $S \xrightarrow[G]{*} x$ iff S is marked.

Believe it or not, this procedure can be implemented in linear time, so it is in fact quite easy to decide whether $L(G) = \varnothing$. See also Miscellaneous Exercise 134 for another approach.

The finiteness problem for CFLs is also decidable (Miscellaneous Exercise 135).

Valid Computation Histories

In contrast to the efficient algorithm just given, it is impossible to decide in general for a given CFG G whether $L(G) = \Sigma^*$. We will show this by a reduction from the halting problem.

The reduction will involve the set VALCOMPS(M, x) of *valid computation histories* of a Turing machine M on input x, defined below. This set is also useful in showing the undecidability of other problems involving CFLs, such as whether the intersection of two given CFLs is nonempty or whether the complement of a given CFL is a CFL.

Recall that a *configuration* α of a Turing machine M is a triple (q, y, n) where q is a state, y is a semi-infinite string describing the contents of the tape, and n is a nonnegative integer describing the head position.

We can encode configurations as finite strings over the alphabet

$$\Gamma \times (Q \cup \{-\}),$$

where Q is the set of states of M, Γ is the tape alphabet of M, and $-$ is a new symbol. A pair in $\Gamma \times (Q \cup \{-\})$ is written vertically with the element of Γ on top. A typical configuration (q, y, k) might be encoded as the string

$$\begin{array}{cccccccc} \vdash & b_1 & b_2 & b_3 & \cdots & b_k & \cdots & b_m \\ - & - & - & - & \cdots & q & \cdots & - \end{array}$$

which shows the nonblank symbols of y on the top and indicates that the machine is in state q scanning the kth tape cell. Recall that the *start configuration* of M on input x is

$$\begin{array}{ccccc} \vdash & a_1 & a_2 & \cdots & a_n \\ s & - & - & \cdots & - \end{array}$$

where s is the start state of M and $x = a_1 a_2 \cdots a_n$.

A *valid computation history* of M on x is a list of such encodings of configurations of M separated by a special marker $\# \notin \Gamma \times (Q \cup \{-\})$; that is, a string

$$\#\alpha_0\#\alpha_1\#\alpha_2\#\cdots\#\alpha_N\#$$

such that

- α_0 is the start configuration of M on x;
- α_N is a halting configuration; that is, the state appearing in α_N is either the accept state t or the reject state r; and
- α_{i+1} follows in one step from α_i according to the transition function δ of M, for $0 \leq i \leq N-1$; that is,

  $$\alpha_i \xrightarrow[M]{1} \alpha_{i+1}, \quad 0 \leq i \leq N-1,$$

  where $\xrightarrow[M]{1}$ is the next configuration relation of M.

In other words, the valid computation history describes a halting computation of the machine M on input x, if M does indeed halt. If M does not halt on x, then no such valid computation history exists.

Let $\Delta = \{\#\} \cup (\Gamma \times (Q \cup \{-\}))$. Then a valid computation history of M on x, if it exists, is a string in Δ^*. Define

$$\text{VALCOMPS}(M,x) \overset{\text{def}}{=} \{\text{valid computation histories of } M \text{ on } x\}.$$

Then $\text{VALCOMPS}(M,x) \subseteq \Delta^*$, and

$$\text{VALCOMPS}(M,x) = \varnothing \iff M \text{ does not halt on } x. \tag{35.1}$$

Thus the complement of VALCOMPS(M,x), namely

$$\sim\text{VALCOMPS}(M,x) = \Delta^* - \text{VALCOMPS}(M,x),$$

is equal to Δ^* iff M does not halt on x.

The key claim now is that $\sim$VALCOMPS(M,x) is a CFL. Moreover, without knowing whether or not M halts on x, we can construct a CFG G for $\sim$VALCOMPS(M,x) from a description of M and x. By (35.1), we will have

$$L(G) = \Delta^* \iff M \text{ does not halt on } x.$$

Since we can construct G effectively from M and x, this will constitute a reduction

$$\sim\text{HP} \leq_{\text{m}} \{G \mid G \text{ is a CFG and } L(G) = \Delta^*\}.$$

By Theorem 33.3(i), the latter set is not r.e., which is what we want to show.

To show that $\sim$VALCOMPS(M,x) is a CFL, let us carefully write down all the conditions for a string $z \in \Delta^*$ to be a valid computation history of M on x:

(1) z must begin and end with a #; that is, it must be of the form

$$\#\alpha_0\#\alpha_1\#\cdots\#\alpha_N\#,$$

where each α_i is in $(\Delta - \#)^*$;

(2) each α_i is a string of symbols of the form

$$\begin{array}{c} a \\ - \end{array} \quad \text{or} \quad \begin{array}{c} a \\ q \end{array}$$

where exactly one symbol of α_i has an element of Q on the bottom and the others have $-$, and only the leftmost has a $\vdash$ on top;

(3) α_0 represents the start configuration of M on x;

(4) a halt state, either t or r, appears somewhere in z (by our convention that Turing machines always remain in a halt state once they enter it, this is equivalent to saying that α_N is a halt configuration); and

(5) $\alpha_i \xrightarrow[M]{1} \alpha_{i+1}$ for $0 \le i \le N-1$.

Let

$$A_i = \{x \in \Delta^* \mid x \text{ satisfies condition } (i)\}, \quad 1 \le i \le 5.$$

A string in Δ^* is in VALCOMPS(M,x) iff it satisfies all five conditions listed above; that is,

$$\text{VALCOMPS}(M,x) = \bigcap_{1 \le i \le 5} A_i.$$

A string is in $\sim$VALCOMPS(M,x) iff it fails to satisfy at least one of conditions (1) through (5); that is, if it is in at least one of the $\sim A_i$, $1 \le i \le 5$. We show that each of the sets $\sim A_i$ is a CFL and show how to obtain a CFG G_i for it. Then $\sim$VALCOMPS(M,x) is the union of the $\sim A_i$, and we know how to construct a grammar G for this union from the G_i.

The sets A_1, A_2, A_3, and A_4 are all regular sets, and we can easily construct right-linear CFGs for their complements from finite automata or regular expressions. The only difficult case will be A_5.

The set A_1 is the set of strings beginning and ending with a #. This is the regular set

$$\#\Delta^*\#.$$

To check that a string is in A_2, we need only check that between every two $\#$'s there is exactly one symbol with a state q on the bottom, and $\vdash$ occurs on the top immediately after each $\#$ (except the last) and nowhere else. This can easily be checked with a finite automaton.

The set A_3 is the regular set

$$\begin{array}{ccccccc} \# & \vdash & a_1 & a_2 & \cdots & a_n & \# \; \Delta^* \\ & s & - & - & \cdots & - & \end{array}$$

To check that a string is in A_4, we need only check that t or r appears someplace in the string. Again, this is easily checked by a finite automaton.

Finally, we are left with the task of showing that $\sim A_5$ is a CFL. Consider a substring $\cdots \#\alpha\#\beta\# \cdots$ of a string in Δ^* satisfying conditions (1) through (4). Note that if $\alpha \xrightarrow[M]{1} \beta$, then the two configurations must agree in most symbols except for a few near the position of the head; and the differences that can occur near the position of the head must be consistent with the action of δ. For example, the substring might look like

$$\begin{array}{ccccccccccccccccccccccc} \cdots & \# & \vdash & a & b & a & a & b & a & b & b & \# & \vdash & a & b & a & b & b & a & b & b & \# & \cdots \\ & & - & - & - & - & q & - & - & - & - & & - & - & - & p & - & - & - & - & - & & \end{array}$$

This would occur if $\delta(q,a) = (p,b,L)$. We can check that $\alpha \xrightarrow[M]{1} \beta$ by checking for all three-element substrings u of α that the corresponding three-element substring v of β differs from u in a way that is consistent with the operation of δ. *Corresponding* means occurring at the same distance from the closest $\#$ to its left. For example, the pair

$$\begin{array}{ccccccc} a & a & b & \quad & a & b & b \\ - & q & - & & p & - & - \end{array}$$

occurring at a distance 4 from the closest $\#$ to their left in α and β, respectively, are consistent with δ, since $\delta(q,a) = (p,b,L)$. The pair

$$\begin{array}{ccccccc} a & b & b & \quad & a & b & b \\ - & - & - & & - & - & - \end{array}$$

occurring at distance 7 are consistent (any two identical length-three substrings are consistent, since this would occur if the tape head were far away). The pair

$$\begin{array}{ccccccc} a & b & a & \quad & a & b & a \\ - & - & - & & - & - & p \end{array}$$

occurring at distance 2 are consistent, because there exists a transition moving left and entering state p.

We can write down all consistent pairs of strings of length three over Δ. For any configurations α and β, if $\alpha \xrightarrow[M]{1} \beta$, then all corresponding substrings of length three of α and β are consistent. Conversely, if all corresponding substrings of length three of α and β are consistent, then $\alpha \xrightarrow[M]{1} \beta$. Thus, to check that $\alpha \xrightarrow[M]{1} \beta$ does *not* hold, we need only check that there exists a substring of α of length three such that the corresponding substring of β of length three is not consistent with the action of δ.

We now describe a nondeterministic PDA that accepts $\sim A_5$. We need to check that there exists i such that α_{i+1} does *not* follow from α_i according to δ. The PDA will scan across z and guess α_i nondeterministically. It then checks that α_{i+1} does not follow from α_i by guessing some length-three substring u of α_i, remembering it in its finite control, and checking that the corresponding length-three substring v of α_{i+1} is not consistent with u under the action of δ. It uses its stack to check that the distance of u from the last # is the same as the distance of v from the last #. It does this by pushing the prefix of α_i in front of u onto the stack and then popping as it scans the prefix of α_{i+1} in front of v, checking that these two prefixes are the same length.

For example, suppose $\delta(q, a) = (p, b, R)$ and z contains the following substring:

$$\begin{array}{ccccccccccccccccccccccc}
\cdots & \# & \vdash & a & b & a & a & b & a & b & b & \# & \vdash & a & b & a & a & b & b & b & b & \# & \cdots \\
 & & - & - & - & - & - & - & q & - & - & & - & - & - & - & - & p & - & - & - & &
\end{array}$$

Then z does not satisfy condition (5), because δ said to go right but z went left. We can check with a PDA that this condition is violated by guessing where the error is and checking that the corresponding length-three subsequences are not consistent with the action of δ. Scan right, pushing symbols from the # up to the substring

$$\begin{array}{ccc}
b & a & b \\
- & q & -
\end{array}$$

(we nondeterministically guess where this is). Scan these three symbols, remembering them in the finite control. Scan to the next # without altering the stack, then scan and pop the stack. When the stack is empty, we are about to scan the symbols

$$\begin{array}{ccc}
b & b & b \\
p & - & -
\end{array}$$

We scan these and compare them to the symbols from the first configuration we remembered in the finite control, and then we discover the error.

We have given a nondeterministic PDA accepting $\sim A_5$. From this and the finite automata for $\sim A_i$, $1 \le i \le 4$, we can construct a CFG G for their union $\sim \mathrm{VALCOMPS}(M, x)$, and

$$L(G) = \Delta^* \iff M \text{ does not halt on } x.$$

If we could decide whether G generates all strings over its terminal alphabet, it would answer the question of whether M halts on x. We have thus reduced the halting problem to the question of whether a given grammar generates all strings. Since the halting problem is undecidable, we have shown:

Theorem 35.1 *It is undecidable for a given CFG G whether or not $L(G) = \Sigma^*$.*

Many other simple problems involving CFLs are undecidable: whether a given CFL is a DCFL, whether the intersection of two given CFLs is a CFL, whether the complement of a given CFL is a CFL, and so on. These problems can all be shown to be undecidable using valid computation histories. We leave these as exercises (Miscellaneous Exercise 121).

Historical Notes

Undecidable properties of context-free languages were established by Bar-Hillel et al. [8], Ginsburg and Rose [47], and Hartmanis and Hopcroft [56]. The idea of valid computation histories is essentially from Kleene [67, 68], where it is called the *T-predicate.*