Lecture 27

The Cocke–Kasami–Younger Algorithm

Given a CFL A and a string $x \in \Sigma^*$, how do we tell whether x is in A? If A is a deterministic CFL and we are given a deterministic PDA for it, we can easily write a program to simulate the PDA. In fact, this is a good way to do it in practice and is done frequently in compilers; we saw an example of this in Lecture 26.

What if A is not a deterministic CFL, or even if it is but we are not given a deterministic PDA for it? If A is given by a CFG G, we can first convert G to Chomsky normal form ($A \to BC$ or $A \to a$) so that each production either produces a terminal or increases the length of the sentential form, then try all derivations of length $2|x| - 1$ to see if any of them produce x. Unfortunately, there might be exponentially many such derivations, so this is rather inefficient. Alternatively, we might produce an equivalent NPDA and try all computation sequences, but again because of the nondeterminism there may be exponentially many to try.

Here is a cubic-time algorithm due to Cocke, Kasami, and Younger [65, 125]. It is an example of the technique of *dynamic programming*, a very useful technique in the design of efficient algorithms. It determines for each substring y of x the set of all nonterminals that generate y. This is done inductively on the length of y.

For simplicity, we will assume that the given grammar G is in Chomsky normal form. One can give a more general version of the algorithm that works for grammars not in this form.

We illustrate the algorithm with an example. Consider the following grammar for the set of all nonnull strings with equally many a's and b's:

$$S \to AB \mid BA \mid SS \mid AC \mid BD,$$
$$A \to a, \qquad B \to b,$$
$$C \to SB, \qquad D \to SA.$$

We'll run the algorithm on the input string $x = aabbab$. Let n be the length of the string (here $n = 6$). Draw $n + 1$ vertical lines separating the letters of x and number them 0 to n:

$$\begin{array}{l} |a|a|b|b|a|b| \\ 0\ 1\ 2\ 3\ 4\ 5\ 6 \end{array}$$

For $0 \le i < j \le n$, let x_{ij} denote the substring of x between lines i and j. In this example, $x_{1,4} = abb$ and $x_{2,6} = bbab$. The whole string x is $x_{0,n}$. Build a table T with $\binom{n}{2}$ entries, one for each pair i, j such that $0 \le i < j \le n$.

$$\begin{array}{cccccccc} 0 & & & & & & & \\ - & 1 & & & & & & \\ - & - & 2 & & & & & \\ - & - & - & 3 & & & & \\ - & - & - & - & 4 & & & \\ - & - & - & - & - & 5 & & \\ - & - & - & - & - & - & 6 \end{array}$$

The i, jth entry of T, denoted T_{ij}, refers to the substring x_{ij}.

We will fill in each entry T_{ij} of T with the set of nonterminals of G that generate the substring x_{ij} of x. This information will be produced inductively, shorter substrings first.

We start with the substrings of length one. These are the substrings of x of the form $x_{i,i+1}$ for $0 \le i \le n - 1$ and correspond to the table entries along the top diagonal. For each such substring $c = x_{i,i+1}$, if there is a production $X \to c \in G$, we write the nonterminal X in the table at location $i, i + 1$.

$$\begin{array}{cccccccc} 0 & & & & & & & \\ A & 1 & & & & & & \\ - & A & 2 & & & & & \\ - & - & B & 3 & & & & \\ - & - & - & B & 4 & & & \\ - & - & - & - & A & 5 & & \\ - & - & - & - & - & B & 6 \end{array}$$

In this example, B is written in $T_{3,4}$ because $x_{3,4} = b$ and $B \to b$ is a production of G. In general, $T_{i,i+1}$ may contain several nonterminals, because there may be several different productions with $c = x_{i,i+1}$ on the right-hand side. We write them all in the table at position $T_{i,i+1}$.

Now we proceed to the substrings of length two. These correspond to the diagonal in T immediately below the top diagonal we just filled in.

For each such substring $x_{i,i+2}$, we break the substring up into two nonnull substrings $x_{i,i+1}$ and $x_{i+1,i+2}$ of length one and check the table entries $T_{i,i+1}$, $T_{i+1,i+2}$ corresponding to those substrings. These entries occur immediately above and to the right of $T_{i,i+2}$. We select a nonterminal from each of these locations (say X from $T_{i,i+1}$ and Y from $T_{i+1,i+2}$) and look to see if there are any productions $Z \to XY$ in G. For each such production we find, we label $T_{i,i+2}$ with Z. We do this for all possible choices of $X \in T_{i,i+1}$ and $Y \in T_{i+1,i+2}$.

In our example, for $x_{0,2} = aa$, we find only $A \in T_{0,1}$ and A in $T_{1,2}$, so we look for a production with AA on the right-hand side. There aren't any, so $T_{0,2}$ is the empty set. Let's write $\varnothing$ in the table to indicate this.

For $T_{1,3}$, we find A immediately above and B to the right, so we look for a production with AB on the right-hand side and find $S \to AB$, so we label $T_{1,3}$ with S. We continue in this fashion until all the $T_{i,i+2}$ are filled in.

0						
A	1					
$\varnothing$	A	2				
–	S	B	3			
–	–	$\varnothing$	B	4		
–	–	–	S	A	5	
–	–	–	–	S	B	6

Now we proceed to strings of length three. For each such string, there are two ways to break it up into two nonnull substrings. For example,

$$x_{0,3} = x_{0,1}x_{1,3} = x_{0,2}x_{2,3}.$$

We need to check both possibilities. For the first, we find $A \in T_{0,1}$ and $S \in T_{1,3}$, so we look for a production with right-hand side AS. There aren't any. Now we check $T_{0,2}$ and $T_{2,3}$. We find $\varnothing$ in $T_{0,2}$, so there is nothing more to check. We didn't find a nonterminal generating $x_{0,3}$, so we label $T_{0,3}$ with $\varnothing$.

For $x_{1,4} = x_{1,2}x_{2,4} = x_{1,3}x_{3,4}$, we find $A \in T_{1,2}$ and $\varnothing$ in $T_{2,4}$, so there is nothing here to check; and we find $S \in T_{1,3}$ and $B \in T_{3,4}$, so we look for a production with right-hand side SB and find $C \to SB$; thus we label $T_{1,4}$

with C.

0						
A	1					
$\varnothing$	A	2				
$\varnothing$	S	B	3			
–	C	$\varnothing$	B	4		
–	–	–	S	A	5	
–	–	–	–	S	B	6

We continue in this fashion, filling in the rest of the entries corresponding to strings of length three, then strings of length four, and so forth. For strings of length four, there are three ways to break them up, and all must be checked. The following is the final result:

0						
A	1					
$\varnothing$	A	2				
$\varnothing$	S	B	3			
S	C	$\varnothing$	B	4		
D	S	$\varnothing$	S	A	5	
S	C	$\varnothing$	C	S	B	6

We see that $T_{0,6}$ contains S, the start symbol, indicating that

$$S \xrightarrow[G]{*} x_{0,6} = x,$$

so we conclude that x is generated by G.

In this example, there is at most one nonterminal in each location. This is because for this particular grammar, the nonterminals generate disjoint sets. In general, there may be more than one nonterminal in each location.

A formal description of the algorithm is given below. One can ascertain from the nested loop structure that the complexity of the algorithm is $O(pn^3)$, where $n = |x|$ and p is the number of productions of G.

```
for i := 0 to n − 1 do                          /* strings of length 1 first */
  begin
    T_{i,i+1} := ∅;                             /* initialize to ∅ */
    for A → a a production of G do
      if a = x_{i,i+1} then T_{i,i+1} := T_{i,i+1} ∪ {A}
  end;
for m := 2 to n do                              /* for each length m ≥ 2 */
  for i := 0 to n − m do                        /* for each substring */
    begin                                       /* of length m */
      T_{i,i+m} := ∅;                           /* initialize to ∅ */
      for j := i + 1 to i + m − 1 do            /* for all ways to break */
        for A → BC a production of G do         /* up the string */
          if B ∈ T_{i,j} ∧ C ∈ T_{j,i+m}
            then T_{i,i+m} := T_{i,i+m} ∪ {A}
    end;
```

Closure Properties of CFLs

CFLs are closed under union: if A and B are CFLs, say generated by grammars G_1 and G_2 with start symbols S_1 and S_2, respectively, one can form a grammar generating $A \cup B$ by combining all productions of G_1 and G_2 along with a new start symbol S and new productions $S \to S_1$ and $S \to S_2$. Before combining the grammars, we must first make sure G_1 and G_2 have disjoint sets of nonterminals. If not, just rename the nonterminals in one of them.

Similarly, CFLs are closed under set concatenation: if A and B are CFLs generated by grammars G_1 and G_2 as above, one can form a grammar generating $AB = \{xy \mid x \in A,\ y \in B\}$ by combining G_1 and G_2 with a new start symbol S and new production $S \to S_1 S_2$.

CFLs are closed under asterate: if A is a CFL generated by a grammar with start symbol S_1, then A^* is generated by the same grammar but with new start symbol S and new productions $S \to S_1 S \mid \epsilon$.

CFLs are closed under intersection with regular sets: if A is a CFL and R is regular, then $A \cap R$ is a CFL (Homework 7, Exercise 2). This can be shown by a product construction involving a PDA for A and a DFA for R similar to the construction we used to show that the intersection of two regular sets is regular. This property is useful in simplifying proofs that certain sets are not context-free. For example, to show that the set

$$A = \{x \in \{a, b, c\}^* \mid \#a(x) = \#b(x) = \#c(x)\}$$

is not context-free, intersect it with $a^*b^*c^*$ to get

$$A \cap a^*b^*c^* = \{a^n b^n c^n \mid n \geq 0\},$$

which we have already shown is not context-free.

Techniques similar to those used in Lecture 10 to show that the regular sets are closed under homomorphic images and preimages can be used to show the same results for CFLs (Miscellaneous Exercise 79).

CFLs are *not* closed under intersection:

$$\{a^m b^m c^n \mid m, n \geq 0\} \cap \{a^m b^n c^n \mid m, n \geq 0\} = \{a^n b^n c^n \mid n \geq 0\}.$$

The product construction does not work for two CFLs; intuitively, there is no way to simulate two independent stacks with a single stack.

We have also shown in Lecture 22 that the family of CFLs is not closed under complement: the set $\{ww \mid w \in \{a,b\}^*\}$ is not a CFL, but its complement is.

Closure Properties of DCFLs

A *deterministic context-free language* (DCFL) is a language accepted by a deterministic PDA (DPDA). These automata were introduced in Supplementary Lecture F. A DPDA is like an NPDA, except that its transition relation is single-valued (i.e., is a function). We also need to include a special right endmarker $\dashv$ so that the machine can tell when it reaches the end of the input string. The endmarker is not necessary for an NPDA, because an NPDA can guess nondeterministically where the end of the input string is.

Most of the important examples of CFLs we have seen have been DCFLs. For example,

$$\{a^n b^n \mid n \geq 0\}$$

is a DCFL. The shift-reduce parser of Lecture 26 was also a DPDA.

Every DCFL is a CFL, but not vice versa. The set

$$\{a,b\}^* - \{ww \mid w \in \{a,b\}^*\}$$

is an example of a CFL that is not a DCFL. We showed in Lecture 22 that this set is a CFL, but its complement is not. This implies that neither set is accepted by any DPDA, since as we showed in Supplementary Lecture F, the family of DCFLs is closed under complement. Thus, unlike the case of finite automata, deterministic PDAs are strictly less powerful than nondeterministic ones.

DCFLs are not closed under union. For example, consider the union

$$\{a^m b^n c^k \mid m \neq n\} \cup \{a^m b^n c^k \mid n \neq k\}.$$

Each set is a DCFL. The union is a CFL—an NPDA could guess nondeterministically which condition to check for—but it is not a DCFL. If it were, then its complement

$$\sim\{a^m b^n c^k \mid m \neq n\} \cap \sim\{a^m b^n c^k \mid n \neq k\}$$

would be. But then intersecting with the regular set $a^* b^* c^*$ would give

$$\{a^n b^n c^n \mid n \geq 0\},$$

which is not even context-free.

Similarly, DCFLs are not closed under reversal, although proving this is a little harder. The set

$$\{ba^m b^n c^k \mid m \neq n\} \cup \{ca^m b^n c^k \mid n \neq k\}$$

over the alphabet $\{a, b, c\}$ is an example of a DCFL whose reversal is not a DCFL (Miscellaneous Exercise 93).

Historical Notes

The CKY algorithm first appeared in print in Kasami [65] and Younger [125], although Hopcroft and Ullman [60] credit the original idea to John Cocke.

Closure properties of CFLs were studied by Scheinberg [110], Ginsburg and Rose [46, 48], Bar-Hillel, Perles, and Shamir [8], and Ginsburg and Spanier [49].

See p. 180 for the history of DPDAs and DCFLs.