Closure Properties of CFL's — Substitution

If a substitution s assigns a CFL to every symbol in the alphabet of a CFL L, then $s(L)$ is a CFL.

Proof

- Take a grammar for L and a grammar for each language $L_a = s(a)$.
- Make sure all the variables of all these grammars are different.
  - ❖ We can always rename variables whatever we like, so this step is easy.
- Replace each terminal a in the productions for L by S_a, the start symbol of the grammar for L_a.
- A proof that this construction works is in the reader.
  - ❖ Intuition: this replacement allows any string in L_a to take the place of any occurrence of a in any string of L.

Example

- $L = \{0^n 1^n \mid n \geq 1\}$, generated by the grammar $S \rightarrow 0S1 \mid 01$.
- $s(0) = \{a^n b^m \mid m \leq n\}$, generated by the grammar $S \rightarrow aSb \mid A$; $A \rightarrow aA \mid ab$.
- $s(1) = \{ab, abc\}$, generated by the grammar $S \rightarrow abA$; $A \rightarrow c \mid \epsilon$.

1. Rename second and third S's to S_0 and S_1, respectively. Rename second A to B. Resulting grammars are:

   $S \rightarrow 0S1 \mid 01$
   $S_0 \rightarrow aS_0b \mid A$; $A \rightarrow aA \mid ab$
   $S_1 \rightarrow abB$; $B \rightarrow c \mid \epsilon$

2. In the first grammar, replace 0 by S_0 and 1 by S_1. The combined grammar:

   $S \rightarrow S_0SS_1 \mid S_0S_1$
   $S_0 \rightarrow aS_0b \mid A$; $A \rightarrow aA \mid ab$
   $S_1 \rightarrow abB$; $B \rightarrow c \mid \epsilon$

Consequences of Closure Under Substitution

1. Closed under union, concatenation, star.
   - ❖ Proofs are the same as for regular languages, e.g. for concatenation of CFL's L_1, L_2, use $L = \{ab\}$, $s(a) = L_1$, and $s(b) = L_2$.

2. Closure of CFL's under homomorphism.

Nonclosure Under Intersection

- The reader shows the following language $L = \{0^i1^j2^k3^l \mid i = k \text{ and } j = l\}$ not to be a CFL.
  - Intuitively, you need a variable and productions like $A \rightarrow 0A2 \mid 02$ to generate the matching 0's and 2's, while you need another variable to generate matching 1's and 3's. But these variables would have to generate strings that did not interleave.
- However, the simpler language $\{0^i1^j2^k3^l \mid i = k\}$ is a CFL.
  - A grammar:

    $S \rightarrow S3 \mid A$
    $A \rightarrow 0A2 \mid B$
    $B \rightarrow 1B \mid \epsilon$
- Likewise the CFL $\{0^i1^j2^k3^l \mid j = l\}$.
- Their intersection is L.

Nonclosure of CFL's Under Complement

- Proof 1: Since CFL's are closed under union, if they were also closed under complement, they would be closed under intersection by DeMorgan's law.
- Proof 2: The complement of L above is a CFL. Here is a PDA P recognizing it:
  - Guess whether to check $i \neq k$ or $j \neq l$. Say we want to check $i \neq k$.
  - As long as 0's come in, count them on the stack.
  - Ignore 1's.
  - Pop the stack for each 2.
  - As long as we have not just exposed the bottom-of-stack marker when the first 3 comes in, accept, and keep accepting as long as 3's come in.
  - But we also have to accept, and keep accepting, as soon as we see that the input is not in $L(0^*1^*2^*3^*)$.

Closure of CFL's Under Reversal

Just reverse the body of every production.

Closure of CFL's Under Inverse Homomorphism

PDA-based construction.

- Keep a "buffer" in which we place $h(a)$ for some input symbol a.
- Read inputs from the front of the buffer (ϵ OK).
- When the buffer is empty, it may be reloaded with $h(b)$ for the next input symbol b, or we may continue making ϵ-moves.

Testing Emptiness of a CFL

As for regular languages, we really take a representation of some language and ask whether it represents $\emptyset$.

- In this case, the representation can be a CFG or PDA.
  - ❖ Our choice, since there are algorithms to convert one to the other.
- The test: Use a CFG; check if the start symbol is useless?

Testing Finiteness of a CFL

- Let L be a CFL. Then there is some pumping-lemma constant n for L.
- Test all strings of length between n and $2n-1$ for membership (as in next section).
- If there is any such string, it can be pumped, and the language is infinite.
- If there is no such string, then $n-1$ is an upper limit on the length of strings, so the language is finite.
  - ❖ Trick: If there were a string $z = uvwxy$ of length $2n$ or longer, you can find a shorter string uwy in L, but it's at most n shorter. Thus, if there are any strings of length $2n$ or more, you can repeatedly cut out vx to get, eventually, a string whose length is in the range n to $2n-1$.

Testing Membership of a String in a CFL

Simulating a PDA for L on string w doesn't quite work, because the PDA can grow its stack indefinitely on ϵ input, and we never finish, even if the PDA is deterministic.

- There is an $O(n^3)$ algorithm (n = length of w) that uses a "dynamic programming" technique.
  - Called Cocke-Younger-Kasami (CYK) algorithm.
- Start with a CNF grammar for L.
- Build a two-dimensional table:
  - Row = length of a substring of w.
  - Column = beginning position of the substring.
  - Entry in row i and column j = set of variables that generate the substring of w beginning at position j and extending for i positions.
  - In reader, these entries are denoted $X_{j,i+j-1}$, i.e., the subscripts are the first and last positions of the string represented, so the first row is $X_{11}, X_{22}, \ldots, X_{nn}$, the second row is $X_{12}, X_{23}, \ldots, X_{n-1,n}$, and so on.

Basis: (row 1) X_{ii} = the set of variables A such that $A \rightarrow a$ is a production, and a is the symbol at position i of w.

Induction: Assume the rows for substrings of length up to $m-1$ have been computed, and compute the row for substrings of length m.

- We can derive $a_i a_{i+1} \cdots a_j$ from A if there is a production $A \rightarrow BC$, B derives any prefix of $a_i a_{i+1} \cdots a_j$, and C derives the rest.
- Thus, we must ask if there is any value of k such that
  - $i \leq k < j$.
  - B is in X_{ik}.
  - C is in $X_{k+1,j}$.

Example

In class, we'll work the table for the grammar:

$$S \rightarrow AS \mid SB \mid AB$$
$$A \rightarrow a$$
$$B \rightarrow b$$

and the string $aabb$.