CSE 322 - Introduction to Formal Methods in Computer Science
Closure Properties of Context-Free Languages

Dave Bacon
Department of Computer Science & Engineering, University of Washington

Previously we showed how regular operations were closed under certain operations: union, intersection, the $*$ operation, concatenation, etc. Do similar results hold for CFLs? Well it will turn out for some of these operations, CFLs are indeed closed. Lets see how we can get at a few of these by introduction the a notion called *substitution.*

Let Σ be some alphabet. Suppose that for every symbol $a \in \Sigma$, we choose a language L_a. Now L_a can be a language over any alphabet, it doesn't have to be over the alphabet Σ. The alphabets of L_a do not have to match for different as. By choosing such a set of languages we have in effect defined a function on Σ. We call this a *substitution.* Lets call this function s, so that $s(a)$ is the language L_a.

If w is a string from Σ^* such that $w = a_1a_2 \dots a_n$, where $a_i \in \Sigma$, then we define $s(w)$ as the language of all strings $x_1x_2 \cdots x_n$ such that $x_i \in s(a_i)$. Extending this even more we can define $s(L)$ as the union of all $s(w)$ for all strings w in L.

Example: Let $s(0) = \{a^nb^nc^n | n \geq 1\}$ and $s(1) = \{aaa, bbb, ccc\}$. Let $w = 01$. Then $s(01) = \{a^nb^nc^n | n \geq 1\} \circ \{aaa, bbb, ccc\}$ which is just $s(01) = \{a^nb^nc^ns^3 | n \geq 1, s \in \{a, b, c\}\}$. If $L = L(0^*)$ is the language of all strings made up of no 1s, then $s(L) = (s(0))^*$. This is $s(L) = \{a^{n_1}b^{n_1}c^{n_1}a^{n_2}b^{n_2}c^{n_2} \dots a^{n_k}b^{n_k}c^{n_k} | k \geq 0, \forall 1 \leq i \leq k, n_i \geq 1\}$

Okay having defined substitution we can now turn to our main result. We claim that if L is a context-free language over an alphabet Σ, and s is a substitution on Σ such that $s(a)$ is a context-free language for all $a \in \Sigma$, then $s(L)$ is a context-free language.

The basic idea behind how this work is that we can take the CFG for L and replace the terminals in this CFG with the start variables of relevant substitutions. In other words if the CFG for L has a terminal a, then we replace this terminal by the start symbol for $s(a)$ in all rules of the CFG.

Let's be a little more formal about this. Let $G = (V, \Sigma, R, S)$ be a CFG for the language L and $G_a = (V_a, \Sigma_a, R_a, S_a)$ be a CFG for the languages $s(a)$, where $a \in \Sigma$. Since we can call variables whatever we like, let us assume that all of the variables in V and V_a, $a \in \Sigma$ are different. That is V and V_a, $a \in \Sigma$ are all disjoint.

Okay so now we will construct a new grammar $G' = (V', \Sigma', R', S')$ for $s(L)$. This new grammar will have

1. $V' = \bigcup_{a \in \Sigma} V_a \bigcup V$
2. $\Sigma' = \bigcup_{a \in \Sigma} \Sigma_a$
3. R' is made up of the union of all rules from every R_a, unioned with all rules from R, but with each terminal $a \in \Sigma$ replaced by S_a everywhere that a occurs in R.
4. $S' = S$.

Okay so now a formal proof would proceed by showing that a string is in $L(G')$ if and only if w is in $s(L)$. If w is in $s(L)$, then there is some string $x = a_1a_2 \dots a_n$ in L and strings x_i in $s(a_i)$ such that $w = x_1x_2 \dots x_n$. Then we can see that G' will generate this string. The portion of G' that comes from the rules of G with S_a substitute for each a which generate a string $S_{a_1}S_{a_2} \dots S_{a_n}$. Then the part of the derivation for each $S_{a_i} \overset{*}{\Rightarrow} x_i$ proceeds from the second part of the derivation. If w is in $L(G')$, then the parse tree for w must be a tree which has an upper structure made up entirely of derivations from G with the old terminals replaced by trees made up entirely of derivations from the relevant G_a. This implies that if w is in $L(G')$ we can find $S_{a_1}S_{a_2} \dots S_{a_n}$ in the derivation of the top part of the tree and then proceeding downwards each of these must derive a string in $s(a_i)$.

Okay so now lets apply this substitution theorem to show how CFLs are closed under union, concatenation and $*$.

Union: Let L_1 and L_2 be CFLs. Then $L_1 \cup L_2$ be the language $s(L)$ where L is the language $\{1, 2\}$ and s is the substitution defined by $s(1) = L_1$ and $s(2) = L_2$. By the theorem $s(L)$ is a CFL.

Concatenation: Let L_1 and L_2 be CFLs. Then $L_1 \circ L_2$ is the language $s(L)$ where L is the language $\{12\}$ and s is the substitution defined by $s(1) = L_1$ and $s(2) = L_2$. By the theorem $s(L)$ is a CFL.

The $*$ operation: Let L_1 be a CFL. Then L_1^* is the language L where L is the language $\{0\}^*$ and the substitution s is defined by $s(0) = L_1$.

It is interesting to think about the above and ask the question of whether CFLs are closed under intersection. After you think about it for a while you'll conclude that you can't use a construction like the one above, if this were true. Which is good, because it turns out that it is not true! CFLs are not closed under intersection (we will see an example of this later.)

I. A DIFFERENT EXAMPLE

In class we talked about the following problem

> If L is a language, and a is a symbol, then L/a i, the *quotient* of L and a, is the set of strings w such that wa is in L. For example if $L = \{a, aab, baa\}$, then $L/a = \{\varepsilon, ba\}$. Prove that is L is a CFL then L/a is CFL. Hint: it might help to have the CFG for L in Chomsky normal form.

How to solve this problem? We want to show that if L is a CFL then L/a is a CFG. Now since L is a CFG, there must exists some CFG $G = (V, \Sigma, R, S)$ such that $L(G) = L$. In order to show that L/a is a CFL, we will show how to convert this CFG, G into a new grammar, call it $G_1 = (V_1, \Sigma, R_1, S_1)$, such that this grammar has the language L/a, i.e. $L(G_1) = L/a$.

First of all we may assume, without loss of generality, that the grammar G has rules expressed in Chomsky normal form (if it isn't we can always convert this to a grammar in this form, which is why we can assume this without loss of generality.) To understand how to construct G_1, lets think about derivations using G. Since G is in Chomsky normal form, the parse tree for our derivation will be a binary tree, with the exception that for the variables which turn into terminals, these variables will only have single children. From the definition of L/a we see that we would like to be able to construct a grammar which allows similar derivations with the property that when the rightmost character in the parse tree is derived, if it is an a we would like to strip this a from the string for our new grammar. Thus it seems that need a way to keep track of the rightmost variable in our derivation. We will do this by adding to our grammar a variable for every variable in V but with this variable indicating that the variable is currently the rightmost string.

Formally, let let A_i denote all of the variables in V $(1 \leq i \leq |V|)$. Then we will define a new set of variables A_i' for our grammar G_1. In particular we let $V_1 = V \cup \{A_i' | 1 \leq i \leq |V|\}$. To be clear, our new grammar has twice as many variables as our old one. Now consider all of the rules from our original grammar (those in R) which are of the form $A_i \rightarrow A_j A_k$. If those rules are used in a derivation from a variable which is not the rightmost variable, then we want them to work as they did before. Thus R_1 will contain these rules. But now, in addition to these rules, we wan to enforce the property that the rightmost variable (the primed ones) stay on the right. Thus if $A_i \rightarrow A_j A_k$ is a rule in G, we want in G_1 the rule $A_i' \rightarrow A_j A_k'$. Finally we keep all the rules which change a variable into a terminal, but if there is a rule sending $A_i \rightarrow a$, we keep this rule but add a primed rule $A_i \rightarrow \varepsilon$. In other words the rules of R_1 are given by

$$R_1 = R \cup \{A_i' \rightarrow A_j A_k' \text{ if } A_i \rightarrow A_j A_k \in R\} \cup \{A_i' \rightarrow \varepsilon \text{ if } A_i \rightarrow a \in R\}$$

If we now define the start variable of G_1 to be the primed version of the start state of G, we will have defined a new grammar G_1 which we claim accepts L/a.

II. INTERSECTION OF A CFL AND A REGULAR LANGUAGE

A more interesting example of a sort of closure property than the one above is provided by the fact that the intersection of a CFL and a regular language is a CFL. The basic idea of how to prove this uses, instead of the CFG for a CFL, the NPDA for the CFL. One imagines taking this NPDA and a NFA for the regular language and running them in parallel. Since the NFA doesn't need to touch the stack, it won't interfere with the NPDA. Then we can make the intersection condition informed by accepting when only both the NPDA and the NFA accept a string. We won't formally prove this here, but note how this prove goes through NPDAs and not just CFGs in its derivation.