Module 7

Properties of context-free languages

What are the boundaries of being context free?

CS 360: Introduction to the Theory of Computing
Spring 2024

Collin Roberts
University of Waterloo

Topics for this module

- Normal forms for context-free grammars
- The pumping lemma for context-free languages, which is the tool to prove a given language is not context-free
- Closure properties for context-free languages
- Decision algorithms for context-free languages

Somewhat surprising: context-free languages do not have all of the same closure properties as regular languages.

Normal forms

- Normal is not a value statement; it means that the grammar satisfies a very simple form.
- Theorem: For any context-free grammar G, there is a context-free grammar G' such that $L(G) = L(G')$ (with the possible exception of ε), where all rules of the grammar G' are of one of the following two forms:
  - $A \to BC$ where A, B and C are variables
  - $A \to a$, where A is a variable and a is a terminal
- Grammars in this form are in Chomsky Normal Form, or CNF.

Why do we care?

- Suppose that we want an upper bound for the number of production steps in the derivation of a given word.
- We cannot do this if derivations can be of arbitrary length.
- changing the grammar to a grammar in CNF improves the situation.
- Also, CNF can make ambiguity in a grammar obvious (though not always).

Idea:

- At each step: either the number of terminals or the string length increases by 1.
- All derivations for a given word will have the same length.

What has to be forbidden?

We need to replace many kinds of now-forbidden rules:

- $A \rightarrow B$
- $A \rightarrow ab$
- $A \rightarrow Ab$
- $A \rightarrow ABC$
- $A \rightarrow \varepsilon$

Some of these simplifications are easier than others; none is especially hard.

First, removing ε rules

A variable A in a grammar G is nullable if $A \overset{*}{\underset{G}{\Rightarrow}} \varepsilon$.

If A is nullable, and there is a rule in the grammar $B \to AC$, we add a rule $B \to C$ to the grammar, and the language of the grammar does not change.

- Previous derivation: $B \Rightarrow AC \overset{*}{\Rightarrow} C \cdots$
- New derivation $B \Rightarrow C \cdots$

We can do this for *any* nullable variable.

- The only word lost is ε, in the case where S is nullable.

Identifying nullable variables

To identify nullable variables, apply this test:

- If there exists a rule $A \to \varepsilon$, then A is nullable.
- If there exists a rule $A \to B_1 B_2 \cdots B_m$, and all B_i are nullable, then A is nullable.
- No other variables are nullable.

Any variable identified by this test is certainly nullable, because the test gives an explicit derivation $A \overset{*}{\Rightarrow} \varepsilon$.

We need to show that every nullable variable A is discovered by this test.

Suppose that there is a k-step derivation $A \overset{k}{\Rightarrow} \varepsilon$. The argument is by induction on k, the length of the derivation.

- Base case: if $k = 1$, then there is a rule $A \to \varepsilon$, and so the above test discovers that A is nullable.
- Induction case: The induction hypothesis is that for any strictly shorter derivation $B \overset{*}{\Rightarrow} \varepsilon$, the test discovers that B is nullable.
- The first step in the derivation of ε from A is $A \Rightarrow B_1 \cdots B_m$, where all the derivations $B_i \overset{*}{\Rightarrow} \varepsilon$ have fewer than k steps. (No terminals can occur in the first step, as the derivation ends in the empty word.)
- So by the induction hypothesis, the test discovers the B_is are all nullable, and hence that A is nullable also.

Dealing with nullable variables

We now must add new rules to the grammar corresponding to the nullable variables.

- Suppose $A \to aBcD$, with B and D nullable.
- Add new rules: $A \to acD$, $A \to aBc$, and $A \to ac$ to the grammar, corresponding to the cases where B generates ε, where D does, and where both do.

In general: from a rule with m nullable variables on the right hand side, add at most $2^m - 1$ new rules, removing each possible subset of the list of nullable variables. (There are 2^m ways of including / excluding the m nullable variables, and we already have the original rule in which all m of them are included.)

Then, remove null productions $A \to \varepsilon$ from the grammar.

ε productions are not necessary

Theorem Let G_1 be the grammar constructed in this way from the original grammar G. Then either $L(G_1) = L(G)$ or $L(G_1) \cup \{\varepsilon\} = L(G)$.

- We will not show both directions of the proof (See Theorem 7.9 in the text).
- Here is the proof that if $w \in L(G)$ and $w \neq \varepsilon$, then $w \in L(G_1)$ (i.e. a proof that $L(G) \setminus \{\varepsilon\} \subseteq L(G')$).
- We will show more generally that if $A \underset{G}{\overset{*}{\Rightarrow}} w$, then $A \underset{G_1}{\overset{*}{\Rightarrow}} w$.
- This is sufficient because we may then take $A = S$.
- The proof is by induction on k, the number of steps in the derivation $A \underset{G}{\overset{k}{\Rightarrow}} w$.
- Base ($k = 1$):
  - Then $A \to w$ is a production in G.
  - Since $w \neq \varepsilon$, therefore $A \to w$ is a production in G_1 also.
  - Therefore we have $A \underset{G_1}{\overset{*}{\Rightarrow}} w$, as required.

The inductive case

- Now suppose that we have a k-step derivation $A \overset{k}{\underset{G}{\Rightarrow}} w$, for $k > 1$.
- The induction hypothesis is that for all derivations $A \overset{\ell}{\underset{G}{\Rightarrow}} x$ with $\ell < k$, we have $A \overset{*}{\underset{G_1}{\Rightarrow}} x$.
- The first step in the derivation of w in G is $A \Rightarrow B_1 B_2 \cdots B_m$, where each B_i is a variable or a terminal.
- At least one variable remains after the first step, as we are not in the base case.
- Write $w = w_1 w_2 \cdots w_m$, where $B_i \overset{*}{\underset{G}{\Rightarrow}} w_i$ for all i. (If B_i is a terminal, say $B_i = w_i$, then $B_i \overset{*}{\underset{G}{\Rightarrow}} w_i$ trivially.)
- Some of the w_i may be ε, but not all, as $w \neq \varepsilon$. Let $C_1, \ldots, C_n$ be the B_i that correspond to the non-ε subwords of w.
- Since the other B_is are nullable, by construction there exists a derivation in G_1 that starts with $A \Rightarrow C_1 \cdots C_n$.
- Each C_i yields its corresponding w_i in G, in fewer than k steps.
- So, by induction, $C_i \overset{*}{\underset{G_1}{\Rightarrow}} w_i$, for all i. Then derive w in G_1 via

$$A \underset{G_1}{\Rightarrow} C_1 \cdots C_n \overset{*}{\underset{G_1}{\Rightarrow}} w_1 C_2 \cdots C_n \overset{*}{\underset{G_1}{\Rightarrow}} \cdots \overset{*}{\underset{G_1}{\Rightarrow}} w_1 \cdots w_n = w.$$

Next transformation: one-variable transformations

- We want to get rid of productions of the form $S \rightarrow A$, with only one variable on the right hand side.
- Such productions are called unit productions.

Why?

- One reason: avoid cycles like $S \Rightarrow A \Rightarrow B \Rightarrow S \Rightarrow \cdots$.

Easy:

- Basic idea: find all of the variables we can get to from a given variable.
- If $S \stackrel{*}{\Rightarrow} A$, then add all of A's productions directly to S's productions.

Finding unit pairs

Variables (A, B) are a unit pair if $A \stackrel{*}{\Rightarrow} B$.
We can find unit pairs by a simple recursive definition:

- (A, A) is a unit pair for any pair A.
- If (A, B) is a unit pair and there is a rule $B \rightarrow C$ in our grammar, where C is a variable, then (A, C) is a unit pair.
- No other pairs are unit pairs.

Easy proof (another induction, which we will not do; it is Theorem 7.11 in the text) that this method finds all unit pairs.

Removing unit productions

If $S \overset{*}{\Rightarrow} A$ in our grammar G, add the productions for A to the productions for S.
Then, remove all unit productions.
Denote the new grammar by G_1.

- Any production that previously used the derivation in G starting from $S \overset{*}{\Rightarrow} A \Rightarrow B_1 B_2 \cdots B_m$ can now use the rule $S \to B_1 B_2 \cdots B_m$ in the new grammar G_1.
- This shows that $L(G) \subseteq L(G_1)$.
- Now, consider a derivation of a word w in $L(G_1)$.
  - Suppose we use a rule $S \to B_1 B_2 \cdots B_m$ in G_1 for a variable S that came from a rule $A \to B_1 B_2 \cdots B_m$ in G, where (S, A) is a unit pair in G.
  - Take derivation $S \underset{G}{\overset{*}{\Rightarrow}} A \Rightarrow B_1 B_2 \cdots B_m$.
  - Then the rest of derivation follows; any word we can derive in G_1, we can also derive in G.
- This shows that $L(G_1) \subseteq L(G)$.
- Therefore we have $L(G_1) = L(G)$.

Remaining bad kinds of rules

For $A \to B_1 B_2 \cdots B_m$, where $m > 2$, create a cascading sequence of rules:

- Only two symbols on right hand side for each rule.
- If we take the first rule for A, then we will produce (eventually) all of $B_1 B_2 \cdots B_m$.

This is not hard. Create $m - 2$ new variables $C_1, \ldots, C_{m-2}$, and these rules:

$$\begin{array}{rcl} A & \to & B_1 C_1 \\ C_1 & \to & B_2 C_2 \\ C_2 & \to & B_3 C_3 \\ & \vdots & \\ C_{m-2} & \to & B_{m-1} B_m \end{array}$$

The new derivation is: $A \Rightarrow B_1 C_1 \Rightarrow B_1 B_2 C_2 \stackrel{*}{\Rightarrow} B_1 B_2 \cdots B_m$.
If some of the B_i are terminals, then some of the rules we have just added are still are not allowed in a CNF grammar.
We will correct this in the next (and last) step.

The last step

In Chomsky Normal Form, a grammar has two kinds of rules:

- $A \to BC$, for variables A, B and C
- $A \to a$, for variables A and terminals a

If we start with an arbitrary grammar, and we:

- Remove ε-productions
- Remove unit productions
- Remove long productions

then the only possible remaining obstacle to being in CNF is that we might still have rules of the form $A \to bc$ or $A \to Bc$, with one terminal on the right hand side of the arrow, but two symbols.

The last step, finished

This is easy:

- For a rule of the form $A \to bc$:
  - Add two new variables:
    - X_b, and
    - X_c.
  - Add three new productions:
    - $A \to X_b X_c$,
    - $X_b \to b$, and
    - $X_c \to c$.
- For a rule of the form $A \to Bc$:
  - Add the variable: X_c.
  - Add the productions:
    - $A \to BX_c$
    - $X_c \to c$

The new variables are only used in these derivations, so they do not change the language of the grammar.
The new grammar fits the desired framework.

Chomsky Normal Form algorithm

From a general CFG:

- Remove ε-productions.
  - Find nullable variables.
  - Change rules using them
  - Then remove all ε-productions.
- Remove one-variable productions.
  - Find unit pairs (A, B) for each variable A.
  - Add B's rules to A.
  - Then remove one-variable productions
- Remove long productions.
  - Create cascading sequence of definitions.
- Remove terminals from two-letter rules.
  - Create a new variable for each terminal, and substitute it into the rules

Why do we care?

Theorem: Let G be a CNF grammar. Let $w \in L(G)$ be arbitrary. Then any derivation of w in G takes $2|w| - 1$ steps.
Proof: by induction on $|w|$. We will instead prove that for any variable A in G, if $A \overset{*}{\Rightarrow} w$, then the derivation must be of length $2|w| - 1$ steps. This is sufficient because we may then take $A = S$.

- The grammar cannot make ε, so the base case is $|w| = 1$.
- Base ($|w| = 1$):
  - I claim that the only step in the derivation is $A \to w$.
  - There are no nullable variables, so if we instead started with a rule of form $A \to BC$, we would have to produce at least 2 letters in the end.
  - So the only derivation of a 1-letter word takes 1 step.
  - Since $1 = 2(1) - 1$, therefore the base case holds.

Second half of the induction proof

- <u>Induction ($|w| > 1$):</u>
  - The induction hypothesis is that for all words x satisfying $A \overset{*}{\Rightarrow} x$ and $|x| < |w|$, the derivation of x takes $2|x| - 1$ steps.
  - As we are not in the base case, the first step in the derivation of w must be of the form $A \rightarrow BC$.
  - We know that $B \overset{*}{\Rightarrow} w_1$ and $C \overset{*}{\Rightarrow} w_2$, where $w = w_1 w_2$, and neither of w_1 or w_2 is ε.
  - Since w_1 and w_2 are both shorter than w, by the induction hypothesis, the derivations for them are of lengths $2|w_1| - 1$ and $2|w_2| - 1$.
  - So the overall derivation, first using the $A \rightarrow BC$ rule, and then the derivations for w_1 and for w_2, takes $2|w_1| - 1 + 2|w_2| - 1 + 1 = 2(|w_1| + |w_2|) - 1 = 2|w| - 1$ steps.

Pumping lemma: review, and the CFL version

Another use for CNF grammars: creation of a CFL pumping lemma.
How did the pumping lemma work for regular languages?

- A regular language L has a DFA with n states, for some n.
- Once a word x in L is of length at least n, the path through the DFA for x reuses a state. So $x = uvw$, where $\hat{\delta}(\hat{\delta}(q_0, u), v) = \hat{\delta}(q_0, u)$.
- Hence, uw must be in L, as must $uvvw$ and all of uv^*w.

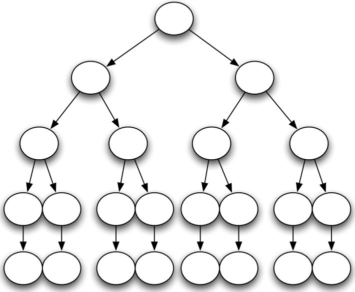
We used this to prove a given language is not regular:

- If for all n, there is a word x in L longer than n letters...
- such that for any decomposition $x = uvw$ with $v \neq \varepsilon$ and $|uv| \leq n$...
- $uv^*w \not\subseteq L$, then L is not regular.

Toward a pumping lemma for CFGs

Suppose we have a CNF grammar G with p variables.
Consider a parse tree in that grammar for a word $z \in L(G)$, where $|z| = k$.

- Each internal node corresponds to a derivation, therefore given k leaves, there are $2k - 1$ internal nodes (grammar is in CNF).
- The parse tree (excluding leaves) is binary (grammar is in CNF).
- The height of the tree (number of edges in the longest path from the root of the tree to a leaf) is at least $1 + \log_2 k$.



In detail,

Theorem 7.17: Suppose we have a parse tree according to a CNF grammar G and suppose the yield of the tree is a word w. If the height of the tree is ℓ, then $|w| \leq 2^{\ell-1}$.

- The proof is by induction on ℓ.
- Base ($\ell = 1$): The length of a path is one less than the number of nodes on the path (count the edges).
- Thus a tree with height 1 consists of only a root and a leaf.
- Therefore $|w| = 1$, and $1 \leq 2^{1-1} = 2^0 = 1$ holds.
- Induction ($\ell > 1$):
- The induction hypothesis is that any parse tree of height $q < \ell$ has yield of length at most 2^{q-1}.
- The root of the tree must use a production of the form $A \to BC$ (as we are not in the base case).
- The induction hypothesis applies to the subtrees rooted at B and C, so these subtrees have yields of lengths at most $2^{\ell-2}$.
- The yield of the tree is the concatenation of the yields of these two subtrees, thus its length is at most $2^{\ell-2} + 2^{\ell-2} = 2^{\ell-1}$.

In detail (completed),

The Theorem implies that, for a word $z \in L(G)$ with length at least 2^p, a parse tree for z has height at least $p+1$.

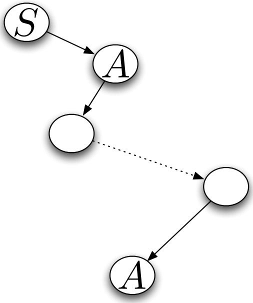
- Suppose that $2^p \leq |z|$.
- Then by the Theorem we have $2^p \leq |z| \leq 2^{\ell-1}$, where ℓ is the height of a parse tree for z.
- Then we must have $p \leq \ell - 1$, or in other words $p+1 \leq \ell$.

The Theorem also implies that the height of a parse tree for a word of length k is at least $1 + \log_2 k$.

- By the Theorem we have $k \leq 2^{\ell-1}$, where ℓ is the height of a parse tree.
- Then we must have $\log_2 k \leq \ell - 1$, or in other words $\log_2 k + 1 \leq \ell$.

Repeated variables on the parse tree

Now in a parse tree of height at least $p+1$, there must be a repeated variable on a path from root to any terminal on the bottom tree level.



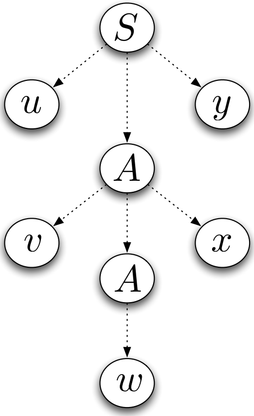
- There are p variables in grammar.
- There are $p+1$ variables and one terminal on a path starting from the root.
- By the pigeonhole principle, there must be a repeated variable.

What does that mean?

Repeated variables, in the derivation

One derivation of the word z in G is of the form:

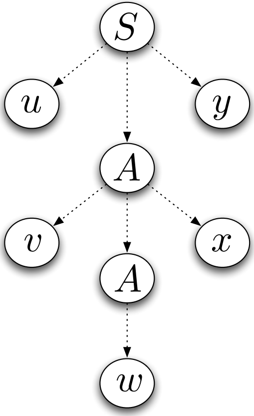
$$\begin{aligned} S &\overset{*}{\Rightarrow} uAy \\ &\overset{*}{\Rightarrow} uvAxy \\ &\overset{*}{\Rightarrow} uvwxy = z \end{aligned}$$



Making a pumping lemma

Important: $A \stackrel{*}{\Rightarrow} vAx$ and $A \stackrel{*}{\Rightarrow} w$. So $A \stackrel{*}{\Rightarrow} vAx \stackrel{*}{\Rightarrow} vvAxx \stackrel{*}{\Rightarrow} v^iAx^i \stackrel{*}{\Rightarrow} v^iwx^i$, for any choice of $i \geq 0$! This will give our pumping lemma for CFLs.

Note: it cannot be the case that $vx = \varepsilon$, as a non-trivial repetition of A occurs, and unit productions $A \stackrel{*}{\Rightarrow} A$ are not allowed in a CNF grammar.



One more trick

Choose a pair of repeated variables near the bottom of the parse tree.

- By assumption $|z| \geq 2^p$, so that a parse tree for z has height at least $p+1$.
- So there exists a terminal in z with a path of length at least $p+1$ above it.
- By the pigeonhole principle, there is a (non-trivially) repeated variable (Say A) in this path, no more than $p+1$ levels above the leaf.
- Then we have $A \overset{*}{\Rightarrow} vAx$ and $A \overset{*}{\Rightarrow} w$, i.e.
  - the yield of the subtree rooted at the lowest A is the word w, and
  - the yield of the subtree rooted at the second lowest A is the word vwx.
- By construction the subtree rooted at the second lowest A has height at most $p+1$.
- Applying Theorem 7.17, we have $|vwx| \leq 2^{(p+1)-1} = 2^p$.
- As the repetition of A is non-trivial, therefore v and x are not both ε (unit productions $A \overset{*}{\Rightarrow} A$ are not allowed in a CNF grammar).

A full statement of the CFL pumping lemma

Lemma: Let G be a CFG in Chomsky Normal form, with p variables.

- Any word $z \in L(G)$ of length at least 2^p can be decomposed as $z = uvwxy$, where
- $|vwx| \leq 2^p$,
- v and x are not both ε, and
- and for all nonnegative i, $uv^iwx^iy \in L(G)$.

As with the pumping lemma for regular languages, we can remove the dependency on a specific choice of CFG for the CFL, since all CFLs have a CNF grammar.

Revised version of the pumping lemma

Let L be a context-free language.

- There exists an $n > 0$ such that any word $z \in L$ where $|z| \geq n$ can be decomposed as $z = uvwxy$, where
- $|vwx| \leq n$,
- v and x are not both ε, and
- for all nonnegative i, $uv^iwx^iy \in L$.

What does this say about non-context-free languages?

Contrapositive of the pumping lemma

Let L be a language.

- Suppose that for any $n > 0$, there exists a word $z \in L$ with $|z| \geq n$ such that:
  - For any decomposition $z = uvwxy$, where $|vwx| \leq n$ and $|vx| > 0$,
  - it is not true that that uv^iwx^iy is in L for all nonnegative integers i.
- Then L is not context free.

This is analogous to the Pumping Lemma for regular languages, except:

- Rather than being decomposed into $x = uvw$,
- and having all words in uv^*w be in L,
- we now have this 5-partite decomposition.

One last rephrasing

Let L be a language.

- Suppose that for any $n > 0$, there exists a word $z \in L$ with $|z| \geq n$ such that:
  - For any decomposition $z = uvwxy$, where $|vwx| \leq n$ and $|vx| > 0$,
  - There exists an $i \geq 0$ such that that uv^iwx^iy is not in L.
- Then L is not context free.

(This just gets rid of one round of double negation.)

Turning it into English

Let L be a language. Suppose that for any $n > 0$:

- there exists a word $z \in L$ with $|z| \geq n$, such that
- for any decomposition $z = uvwxy$, where $|vwx| \leq n$ and $|wx| > 0$,
- there exists an $i \geq 0$ such that $uv^iwx^iy \notin L$.

Then L is not context free.

Suppose that for any definition of long:

- There is a long word
- For which every decomposition
- is not pumpable.

This gives us a recipe for proving that a given language is not context free.

To prove a language is not context free

Our recipe:

- Find a long word.
- Look at its decompositions.
- Show they cannot be pumped.

Or, formally:

- For given $n > 0$, find a word $z \in L$ at least n letters long.
- Look at all decompositions $z = uvwxy$, with $|vwx| \leq n, vx \neq \varepsilon$.
- Say something useful about the decompositions
- For each decomposition, find an i such that uv^iwx^iy is not in L.
- Then the language L is not context free.

An example: $L = \{a^i b^i c^i \mid i \geq 0\}$

I claim that the language $L = \{a^i b^i c^i \mid i \geq 0\}$ is not context-free.

- For each $n > 0$, find a word $z \in L$ that is at least n letters long.
  We choose the long word $z = a^n b^n c^n$. Clearly $z \in L$.
- Consider decompositions $z = uvwxy$ with $|vwx| \leq n$ and $vx \neq \varepsilon$.
- Say something useful about all such decompositions.
  - All such decompositions have one or two types of letters in vwx, but not all 3.
  - (Why? The smallest consecutive substring with all 3 symbols is $ab^n c$; it has length $n + 2$.)
  - In particular, vx omits one or two letters of the set $\{a, b, c\}$.
- For each decomposition, find an i such that $uv^i wx^i y$ is not in L.
  - Consider uwy (i.e. take $i = 0$). Observe that uwy does not have the same number of a's, b's and c's, since one of these letters is not in vx, and at least one is!
  - Hence, $uwy = uv^0 wx^0 y$ is not in L. (Neither is $uvvwxxy$).

We have shown that z cannot be pumped, and hence, L is not context free.

Another example

Consider $L = \{a^i b^j c^k \mid i < j, i < k\}$.

- Let $n > 0$ be arbitrary.
- Long word: $z = a^n b^{n+1} c^{n+1}$.
- Consider decompositions $z = uvwxy$ with $|vwx| \leq n$ and $vx \neq \varepsilon$.
  In all of them, vx has either no a's, or has a's but no c's.
  - Case 1: No a's.
    Then uwy has fewer b's or fewer c's than z, but there are not fewer a's.
    So uwy does not have fewer a's than both b's and c's, and therefore uwy is not in L.
  - Case 2: a's, but no c's.
    $uvvwxxy$ has as at least as many a's as c's, so $uvvwxxy$ is not in L.
- So no decomposition of our long word $z = a^n b^{n+1} c^{n+1}$ can be pumped.

And, thus, L is not context free.

One last example

Somewhat surprising, maybe:
$L = \{ss \mid s \in \{a, b\}^*\}$.
L includes words like *aa* or *abbabb* or ε or *abaaba*.

- For a given $n > 0$, find a long word. We will use $z = a^n b^n a^n b^n$. (This choice might not be so obvious.)
- Decompose into $z = uvwxy$, with $|vwx| \leq n$ and $vx \neq \varepsilon$. Then uwy must have at least one a or one b removed from one of the two copies of the identical string.
- But when we remove vx from $uvwxy$ to form uwy, and lose a letter from the copied word, we cannot lose the corresponding letter on the other side; it is too far away.
- Therefore $uwy \notin L$.
- So L is not context-free.

(See Example 7.21 of the text for all the gory details.)

A bit surprising

Surprising: The very similar-looking $L = \{ss^R \mid s \in \{a, b\}^*\}$, of even-length palindromes, is context free, with this grammar:

- $S \to aSa \mid bSb \mid \varepsilon$

However the previous example is still not context-free; we cannot keep all the information available whenever it is needed. (PDAs, which only recognize CFLs, have trouble with doing this.)

Closure rules

- Regular languages are closed under concatenation, Kleene star, union, intersection, complement, reversal and more.
- For CFLs, the above statement is not true. CFLs are:
  - Closed under union, concatenation, Kleene star and reversal.
  - Not closed under intersection or complementation.

The easy ones

Union:

- Grammar $G_1 : S_1 \to \cdots$
- Grammar $G_2 : S_2 \to \cdots$ (with all different variables)
- New grammar: $G : S \to S_1 | S_2 \cdots$

Concatenation:

- Grammar $G_1 : S_1 \to \cdots$
- Grammar $G_2 : S_2 \to \cdots$ (with all different variables)
- New grammar: $G : S \to S_1 S_2 \cdots$

Kleene star:

- Grammar $G_1 : S_1 \to \cdots$
- New grammar: $G : S \to \varepsilon | S_1 S$

Reversal

Reversal is not hard, either.

- Given a grammar G, construct a new grammar G', by reversing the outputs of all of the productions in G.
- For example, if G has a production $S \to XYZ$, then add the rule $S \to ZYX$ to G'.
- (If the grammar is in CNF, this works especially easily.)
- Now for a given derivation of a word $w \in L(G)$, apply the corresponding rules in G' to generate $w^R \in L(G')$.
- Then by construction, $L(G') = L(G)^R$.
- Then since $L(G)^R$ is the language of a context-free grammar, therefore $L(G)^R$ is a context-free language.

Intersection

We have already seen a language that shows that the intersection of two CFLs is not always a CFL.
$L = \{a^i b^i c^i \mid i \geq 0\}$ (we saw that this language is not context-free).

- $L = L_1 \cap L_2$, where:
  - $L_1 = \{a^i b^i c^j \mid i, j \geq 0\}$.
  - $L_2 = \{a^i b^j c^j \mid i, j \geq 0\}$.
- L_1 and L_2 are each the concatenation of two context-free languages, so context free.
- In detail, define
  - $L_{11} = \{a^i b^i \mid i \geq 0\}$ (a CFL, with grammar $G : S \rightarrow aSb|\varepsilon$), and
  - $L_{12} = \{c^j \mid j \geq 0\} = L(c^*)$ (regular, and thus a CFL).
  - Then $L_1 = L_{11}L_{12}$.
  - And $L_{21} = \{a^i \mid i \geq 0\} = L(a^*)$ (regular, and thus a CFL).
  - $L_{22} = \{b^j c^j \mid j \geq 0\}$ (a CFL, with grammar $G : S \rightarrow bSc|\varepsilon$), and
  - Then $L_2 = L_{21}L_{22}$.

Therefore, the class of context-free languages is not closed under intersection!

Intersection with a *regular* language

If L_1 is context free and L_2 is regular, then $L_1 \cap L_2$ is context free.

- Suppose that a PDA M accepts L_1 by final state, and that a DFA D accepts L_2.
- Let R be the states of M, and $F_M \subseteq R$ be the accept states.
- Let S be the states of D, and $F_D \subseteq S$ be the accept states.
- Define a PDA, P, which accepts by final state, with
  - States $Q = R \times S$,
  - Accept states $F = F_M \times F_D$,
  - and transition function δ, defined from the transition functions δ_M for M and δ_D for D (ignoring stack manipulations for the moment):

$$\delta(a,(r,s)) = \begin{cases} \{(\delta_M(\varepsilon, r), s)\} & \text{if } a = \varepsilon \\ \{(\delta_M(a, r), \delta_D(a, s))\} & \text{if } a \neq \varepsilon \end{cases}$$

  - Manipulate the stack in P exactly as it was manipulated in M.
- Then P is a PDA, and from construction, we have that
  - P accepts w
  - if and only if M accepts w and D accepts w
  - if and only if $w \in L_1 \cap L_2$, so that $L_1 \cap L_2$ is a CFL.

Note, this construction will not work for intersection of two arbitrary CFLs: both PDAs would need editing access to the one stack.

Complementation

The following example will show that the class of context-free languages is not closed under taking complements.

Let $L_1 = \{a^i b^j c^k \mid i \neq j \text{ or } k \neq j\}$.

- Then L_1 is context-free, as it is the union of the four CFLs:
  - $L_{11} = \{a^i b^j c^k \mid i < j\} = \{a^i b^j \mid i < j\}\{c^k \mid k \geq 0\}$,
  - $L_{12} = \{a^i b^j c^k \mid i > j\} = \{a^i b^j \mid i > j\}\{c^k \mid k \geq 0\}$,
  - $L_{13} = \{a^i b^j c^k \mid j < k\} = \{a^i \mid i \geq 0\}\{b^j c^k \mid j < k\}$, and
  - $L_{14} = \{a^i b^j c^k \mid j > k\} = \{a^i \mid i \geq 0\}\{b^j c^k \mid j > k\}$.
- For example, a grammar for $\{a^i b^j \mid i < j\}$ is $G : S \to b|Sb|aSb$.

Now, consider $L_2 = L(a^* b^* c^*)'$. That is, L_2 is the set of words that are not of the form $a^i b^j c^k$, for any choice of i, j, k.

- Then L_2 is regular, as it is the complement of a regular language. (Exercise: What is a regular expression for L_2?)
- Then L_2 is a CFL.

Then $L = L_1 \cup L_2$ is context-free, as it is the union of two context-free languages.

Complementation, continued

Note that words in L are:

- of the form $a^i b^j c^k$ for some i, j, k, but not having $i = j = k$, or
- not of the form $a^i b^j c^k$, for any choice of i, j, k.

Now, consider L'. I claim that

$$L' = \{a^i b^i c^i \mid i \geq 0\}.$$

We have

$$\begin{aligned} L' &= (L_1 \cup L_2)' \\ &\underbrace{=}_{\text{DeMorgan}} L_1' \cap L_2'. \end{aligned}$$

- $L_2' = (L(a^* b^* c^*)')' = L(a^* b^* c^*)$ is the set of words that can be written in the form $a^i b^j c^k$, for some choice of i, j, k,
- and L_1' is the set of such words for which $i = j = k$,
- and therefore our description of L' is correct.
- We have already seen that L' is not context free.

L is context free, and its complement is not context-free.

Therefore the class of context-free languages is not closed under complementation.

Contrasts with DCFLs

DCFLs are closed under complementation.

- Proving this is non-trivial.
- See the additional notes for Module 7.

Simple proof that there are context-free languages that are not DCFLs: we just saw one. L is context-free, while L' is not context-free (and hence not a DCFL)

Decision algorithms for CFLs

- Your textbook constructs a couple of efficient algorithms for transforming CFGs to CNF, or to test membership of a word in the language of the CFL.
- One, in particular, is used in bioinformatics, a lot; see section 7.4.4 for the CYK algorithm, which tests membership of a word of length n in the language of a CNF grammar in $O(n^3)$ runtime.

Lots of the analogues to the problems we saw in Module 4 for regular languages are not solvable by computers.

What we can do: membership

- Given a CFG, does its language include the word w?
  - Turn it into CNF.
  - Try all derivations of length $2|w| - 1$.
  - Does any of them derive w?
- Given a PDA, does its language include the word w?
  - Turn it into a CFG.
  - Use the algorithm for CFGs. (Note: This is an example of a reduction. Reductions will be crucial when working with Turing machines at the end of the course.)
  - We cannot just run the PDA: it might run forever!

Empty language

Given a CFG, is its language empty?

- First turn it into CNF.
- Lemma: If a CFG in CNF, G, having p variables generates any words, then it must generate a word with fewer than 2^p letters.
  Proof:
  - Assume $L(G) \neq \emptyset$.
  - Let $z_0 \in L(G)$ be arbitrary.
  - If $|z_0| < 2^p$, then we are finished.
  - Otherwise, $|z_0| \geq 2^p$ and by the proof of the Pumping Lemma, we can decompose $z_0 = u_0 v_0 w_0 x_0 y_0$, with $|v_0 w_0 x_0| \leq 2^p$ and $v_0 x_0 \neq \varepsilon$.
  - If $|u_0 w_0 y_0| < 2^p$, then we are finished.
  - Otherwise, $|u_0 w_0 y_0| \geq 2^p$ and $u_0 w_0 y_0 \in L(G)$ and so by the proof of the Pumping Lemma, we can decompose $u_0 w_0 = u_1 v_1 w_1 x_1 y_1$, with $|v_1 w_1 x_1| \leq 2^p$ and $v_1 x_1 \neq \varepsilon$.
  - Continuing in this way we obtain a sequence of words in $L(G)$ having strictly decreasing lengths: $z_0, u_0 w_0 y_0, u_1 w_1 y_1, \ldots, u_j w_j y_j, \ldots$.
  - As z_0 has finite length, after at most $|z_0| - 2^p + 1$ steps, we will obtain a word in $L(G)$ with length $< 2^p$. $\square$
- Enumerate all of them, and test membership for each.
- This is unbelievably slow, but it will work.

Undecidable problems

Other sensible problems are undecidable:

- Given two CFGs, do their languages have any words in common?
- Given two CFGs, do their languages have all words in common?
- Is the language of a CFG equal to Σ^*?
- Given two CFGs, is the language of one a subset of the other's?
- Is a given CFG ambiguous? (Note: this is about the grammar, not the language.)
- Is a given CFL inherently ambiguous?

That is, there is no algorithm for any of these problems!

- Note: We are not saying that we are waiting for an algorithm to be discovered.
- We know that no algorithm can exist to solve each of these problems.

End of module 7

- Normal forms for CFGs let us prove theorems about them, and design efficient algorithms to test membership.
- Some surprisingly simple languages are not context-free.
- The class of context-free languages is not closed under as many operations as the class of regular languages.
- Many natural CFL problems are undecidable.

End of second main unit. In the Turing machine unit, we design real algorithms, and identify the limits of real computers.