Module 5

Context-free grammars and languages

Building a connection between human language and formal language

CS 360: Introduction to the Theory of Computing
Spring 2024

Collin Roberts
University of Waterloo

Topics for this module

- Context-free grammars and languages
- Parsing words in CFGs
- Ambiguity in CFGs

CFGs and CFLs are the second major topic of this course. They have applications to parsing, primarily, but pop up in bioinformatics often. They also have had applications in (essentially) political theory.

- Yes, really.

Context-free grammars: preliminaries

Context-free grammars:

- More complicated structure than regular expressions
- Can encode more interesting languages
- Constrained in how one part of the input relates to another (without context: here, the name of language class means something important)

We will have an automaton for CFGs, called pushdown automata, which we will discuss in Module 6.

Context-free grammars: the idea

Basic idea:

- Construct words in a language by applying rules to symbols.
- We are finished when all of the symbols in the word are fleshed out.
- An example will help to explain this.

Example:

- $S \rightarrow$ subject verb object
- $S \rightarrow$ subject verb
- subject $\rightarrow$ "billy" or "john" or "lucia"
- verb $\rightarrow$ "wants" or "broke"
- object $\rightarrow$ "the toy" or "a dog"

Here, sentences have forms like:
"billy wants a dog" or "john broke"

A more typical example

Another example with $\Sigma = \{0, 1\}$ (with the form we will use):

- $S \to AB$
- $A \to 0A \mid \varepsilon$
- $B \to B1 \mid \varepsilon$

Here,

- S generates AB,
- then we generate a subword from A,
- and follow with a subword generated by B.

Details on each step:

- The word generated from A is from 0^*,
- and the word generated from B is from 1^*.
- Some words in this CFG's language: $\varepsilon, 0000001111, \ldots$.
- Its language: 0^*1^*.

Formal definition of a CFG

Formal definition: a context-free grammar (CFG) is a 4-tuple:
$G = (V, T, P, S)$, where

- V is a finite set of variables, usually denoted by capital letters.
- T is a finite alphabet, called terminals. T is disjoint from V.
  T is the alphabet for the CFG's language.
- P is a finite set of productions, which are definitions for the variables.
- Each production P is of the form $A \to \alpha$, where
  - A is a variable, and
  - α is a string of symbols from V and T, therefore $\alpha \in (V \cup T)^*$, OR α may equal ε, as we saw in the example.
  - Notation: If we have productions $A \to \alpha$ and $A \to \beta$, shorthand these two statements as $A \to \alpha \mid \beta$.
- S = Start variable: the symbol from V where the derivation of a word begins.

Formalizing the previous example

Our example from before, which generated 0^*1^*:
$G = (V, T, P, S)$, where

- $V = \{A, B, S\}$
- $T = \{0, 1\}$
- $P = \{S \to AB, A \to 0A \mid \varepsilon, B \to B1 \mid \varepsilon\}$
- $S = S$

Now, we need to define what it means for a word to be in the language of a CFG.
Derivations: produce a string in T^* by applying rules in P to strings we get from S.

Derivations

Suppose that β and γ are strings from $(V \cup T)^*$.
We will define several kinds of derivations:

- One-step derivations:
  Notation: $\beta \Rightarrow \gamma$ if we can produce γ from β by applying one production from P.
  (In detail, for strings $x,\ y$, if $\beta = xAy$ and $\gamma = x\alpha y$, and if there is a production $A \rightarrow \alpha$ in P, then $\beta \Rightarrow \gamma$.)
- k-step derivations:
  Notation: $\beta \overset{k}{\Rightarrow} \gamma$ if we can produce γ from β by applying k productions from P, in sequence.
  Formally, defined by induction:
  - Base case: $\beta \overset{0}{\Rightarrow} \beta$ for any β.
  - Inductive case: If $\beta \overset{k}{\Rightarrow} \gamma$ and $\gamma \Rightarrow \alpha$, then $\beta \overset{k+1}{\Rightarrow} \alpha$.
- Many-step derivations:
  Notation: $\beta \overset{*}{\Rightarrow} \gamma$ if $\beta \overset{k}{\Rightarrow} \gamma$ for some finite k.

Many-step derivations

Proper definition:

- Suppose that there is a sequence of strings $\gamma_1, \gamma_2, \ldots, \gamma_k$ for some $k \geq 1$ such that:
  - $\alpha = \gamma_1$,
  - $\beta = \gamma_k$,
  - For all i from 2 to k, $\gamma_{i-1} \underset{G}{\Rightarrow} \gamma_i$.
- Then, we can say that $\alpha \overset{*}{\underset{G}{\Rightarrow}} \beta$.

The subscript G indicates the grammar, and can safely be omitted if no confusion is possible about which grammar we mean.

Leftmost and rightmost derivations

To derive a word in a grammar G, our rule for $\overset{*}{\underset{G}{\Rightarrow}}$ requires only that at each step, we expand a single variable. It does not specify which variable.

- This can make proving theorems difficult.
- In a leftmost derivation for α, we require that the variable expanded is the leftmost variable remaining in α.
  If that is the case, then we say that $\alpha \underset{lm}{\Rightarrow} \beta$.
- If it is a multi-step derivation, we use the notation $\alpha \overset{*}{\underset{lm}{\Rightarrow}} \beta$, to indicate that at each step, we expand the leftmost variable.
- Similarly, we can define rightmost derivations, which we denote using $\alpha \underset{rm}{\Rightarrow} \beta$ and $\alpha \overset{*}{\underset{rm}{\Rightarrow}} \beta$.
- We could show that $\alpha \overset{*}{\underset{rm}{\Rightarrow}} \beta$ if and only if $\alpha \overset{*}{\underset{lm}{\Rightarrow}} \beta$, and $\alpha \overset{*}{\underset{rm}{\Rightarrow}} \beta$ if and only if $\alpha \overset{*}{\Rightarrow} \beta$. (We will not, but we could.)

Language of a grammar

We will work with the language of a grammar.

- Given a context-free grammar $G = (V, T, P, S)$, its language is:

$$L(G) = \{w \in T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}.$$

  These are the words we can derive from the start variable S, with no variables from V remaining in them.
- A language L is context free if it is the language of a context-free grammar G.

Why context-free languages?

- It does not matter where we see a symbol.
- Suppose P includes this production: $A \rightarrow ab$
- Given any string $x_1 A x_2$, we can change the A into ab.
- That is, we can do this substitution regardless of the context of A, which is what x_1 and x_2 are.
- Think about mad-libs from when you were a kid; they were funny because you were making sentences that were silly. They did not have context.
- Proper English (or indeed, any human language) has context.

An example

$G = (V, T, P, S)$, where

- $V = \{S\}$,
- $T = \{0, 1\}$,
- $P = \{S \rightarrow 0S1 \mid S1 \mid \varepsilon\}$, and
- $S = S$.

We do not need to specify V, T or S when they are obvious.

- $G : S \rightarrow 0S1 \mid S1 \mid \varepsilon$.

The language of this CFG

Let $L = \{0^i 1^j$, where $0 \leq i \leq j\}$

- Proof using the Pumping Lemma that L is not regular:
  - Let $n > 0$.
  - Let $x = 0^n 1^n \in L$ be our long word.
  - Write $x = uvw$, where $|uv| \leq n$ and $v \neq \varepsilon$.
  - Then $v = 0^k$ for some $1 \leq k \leq n$, so $uv^2w \notin L$.

Theorem: $L(G) = L$.

- To prove the Theorem, we must show $L \subseteq L(G)$ and $L(G) \subseteq L$.

Proving that $L \subseteq L(G)$

We will start with $L \subseteq L(G)$. Suppose $x \in L$. We must show x can be generated by G. The proof is by induction on $|x|$.

- Base case: $|x| = 0$. But then $x = \varepsilon$, and $S \to \varepsilon$ is a rule in the grammar G. Therefore we have $S \stackrel{*}{\Rightarrow} x$. So the base case holds.
- Inductive case: Let $|x| = n \geq 1$. Then $x \neq \varepsilon$.
- The induction hypothesis is that, for any $w \in L$, such that $|w| < |x|$, we have that $w \in L(G)$, i.e. that $S \stackrel{*}{\Rightarrow} w$.
- Since $x \in L$, we know that $x = 0^i 1^j$ for some $i \leq j$.

Finishing the proof that $L \subseteq L(G)$

We have these two cases:

1. $i = 0$, so $x = 1^j = 1^{j-1}1$ ($j \geq 1$ as we are not in the base case).
   - Then $w = 1^{j-1} \in L$ by the definition of L.
   - As $|w| < |x|$, we have that $w \in L(G)$ by the induction hypothesis.
   - So $S \stackrel{*}{\Rightarrow} w$.
   - Then derive x in G via

     $$S \Rightarrow S1 \stackrel{*}{\Rightarrow} w1 = x.$$

   - This shows that $x \in L(G)$ in this case.
2. $x = 0^i 1^j$, and $i > 0$.
   - Then $w = 0^{i-1}1^{j-1}$ is also in L, as $i \leq j$ implies $i - 1 \leq j - 1$.
   - As $|w| < |x|$, we have that $w \in L(G)$ by the induction hypothesis.
   - So $S \stackrel{*}{\Rightarrow} w$.
   - Then derive x in G via

     $$S \Rightarrow 0S1 \stackrel{*}{\Rightarrow} 0w1 = x.$$

   - This shows that $x \in L(G)$ in this case.

This finishes the proof that $L \subseteq L(G)$.

Other half of the proof

Now we must show: if $x \in L(G)$, then $x \in L$.

Proof: by induction on k, the number of steps in the derivation of x.

- Base case ($k = 1$): Then $x = \varepsilon$, which is a member of L. So the base case holds.
- Inductive case ($k \geq 2$): Assume every word $w \in L(G)$ with fewer than k steps in its derivation is in L.
  - As we are not in the base case, the first step in the derivation of x must be $S \to 0S1$ or $S \to S1$.
  - If the first step is $S \to 0S1$,
    - Write $x = 0w1$ where $S \stackrel{*}{\Rightarrow} w$ in fewer than k steps.
    - By the induction hypothesis, we know that $w \in L$.
    - Write $w = 0^i 1^j$, where $i \leq j$.
    - Then $x = 0w1 = 0(0^i 1^j)1 = 0^{i+1} 1^{j+1} \in L$.
  - If the first step is $S \to S1$,
    - Write $x = w1$ where $S \stackrel{*}{\Rightarrow} w$ in fewer than k steps.
    - By the induction hypothesis, we know that $w \in L$.
    - Write $w = 0^i 1^j$, where $i \leq j$.
    - Then $x = w1 = (0^i 1^j)1 = 0^i 1^{j+1} \in L$.
- This finishes the proof that $L(G) \subseteq L$.

Put them together: $L(G) \subseteq L$ and $L \subseteq L(G)$. So $L = L(G)$ as claimed.

This take a bit of care to set up, but theorems like this one are not hard to prove.

Another CFL

Let's look at another CFL:
Consider $L = \{w \in \{0,1\}^* \mid n_0(w) = n_1(w)\}$.

- L consists of all binary strings with as many 0s as 1s.

Theorem: L is context free.
Proof: We will show that L is the language of the grammar:
$G = (V, T, P, S)$, where

- $V = \{S\}$,
- $T = \{0,1\}$,
- $P = \{S \to \varepsilon \mid 0S1 \mid 1S0 \mid SS\}$, and
- $S = S$.

To prove this, we must show that the languages are subsets of each other, in both directions.

Proof that $L(G) \subseteq L$

First, we will show that $L(G) \subseteq L$.

- Let $w \in L(G)$ be arbitrary, i.e. assume $S \overset{n}{\Rightarrow} w$, for some n.
- The proof is by induction on n.
- Base case ($n = 1$): Then $w = \varepsilon$ (the only one-step derivation in G is $S \to \varepsilon$). Note that $\varepsilon \in L$, so the base case holds.
- Inductive case ($n \geq 2$): The induction hypothesis is that for every $x \in L(G)$ which is derived in fewer than n steps, we have $x \in L$.

Inductive step in one direction

Consider the first step in a derivation of w. We have these cases (which are exhaustive, as we are no longer in the base case):

- $\underline{S \to 0S1}$: Write $w = 0x1$, where $S \overset{n-1}{\Rightarrow} x$. The induction hypothesis applies to x, so $n_0(x) = n_1(x)$. Hence, by construction, $n_0(w) = n_1(w)$ too. Thus $w \in L$.
- $\underline{S \to 1S0}$: Write $w = 1x0$, where $S \overset{n-1}{\Rightarrow} x$. The induction hypothesis applies to x, so $n_0(x) = n_1(x)$. Hence, by construction, $n_0(w) = n_1(w)$ too. Thus $w \in L$.
- $\underline{S \to SS}$: Write $w = xy$, where $S \overset{*}{\Rightarrow} x$ and $S \overset{*}{\Rightarrow} y$. The induction hypothesis applies to x, so $n_0(x) = n_1(x)$. The induction hypothesis also applies to y, so $n_0(y) = n_1(y)$. Hence, by construction, $n_0(w) = n_1(w)$ too. Thus $w \in L$.

We have shown that in all cases, if $w \in L(G)$ then $w \in L$. Thus the containment $L(G) \subseteq L$ is established.

The other direction: Show that $L \subseteq L(G)$.

Let $w \in L$ be arbitrary. We must show that $w \in L(G)$.
The proof is by induction on $|w|$.

- Base case ($|w| = 0$): Then $w = \varepsilon$. Then $w \in L(G)$ via the production $S \to \varepsilon$ in G.
- Inductive case ($|w| \geq 1$): The induction hypothesis is that for all words $x \in L$ with $|x| < |w|$, we have $x \in L(G)$.
- Note that all words in L have even length, since they have equal numbers of 0s and 1s.
- We will have four cases, depending on the two outside letters of w.

Four cases of the induction proof

1. $w = 0x1$.
   - Since $w \in L$, therefore $n_0(w) = n_1(w)$ by the definition of L.
   - But then $n_0(x) = n_1(x)$ by construction, and so $x \in L$.
   - Also by construction, $|x| < |w|$.
   - Thus by the induction hypothesis, $x \in L(G)$, i.e. $S \overset{*}{\Rightarrow} x$.
   - Then we can derive w in G via

$$S \Rightarrow 0S1 \overset{*}{\Rightarrow} 0x1 = w, \text{ so } w \in L(G).$$

2. $w = 1x0$. Same argument, instead using the rule $S \to 1S0$ to start.
3. $w = 0x0$. We will use the rule $S \to SS$ here to produce w (see below).
4. $w = 1x1$. Here, we will use the rule $S \to SS$ analogously (see below).

For cases 3 and 4, must show that w can be decomposed into two parts $w = yz$, both of which are in $L(G)$.

Finishing cases 3 and 4

Claim: If $w = 0x0$ and $n_0(w) = n_1(w)$, then we can write $w = yz$, where $n_0(y) = n_1(y)$, $y \neq \varepsilon$, and $z \neq \varepsilon$.

Why is this useful?

- Decompose w into two parts that are both shorter and in L.
- By the inductive hypothesis, they are then both also in $L(G)$.
- Use the $S \to SS$ rule to start the derivation.

Why is it true?

- Basic idea: look at the balance between the number of 1s and 0s in prefixes of w.
- The 1-letter prefix of w is 0, which has one more 0 than 1s.
- The $(|w| - 1)$-letter prefix of w is $0x$, which has one fewer 0 than 1s. (Why? We will wind up evening out at the last letter, which is 0.)
- This balance between 0s and 1s shifts by 1 each letter. It eventually goes from $+1$ to -1.
- So at some point strictly in between, the balance is zero.

End of cases 3 and 4

Definitions:

- Let p_i be the i-letter prefix of w.
- Let $b_i = n_0(p_i) - n_1(p_i)$.

With this in mind:

- $b_1 = 1$, since $p_1 = 0$.
- $b_{|w|-1} = -1$, since $p_{|w|-1} = 0x$, where $0x0 \in L$, and
- $b_{|w|} = 0$, since $p_{|w|} = 0x0 \in L$.
- For any i, $b_i = b_{i-1} \pm 1$, depending on whether the ith character is 1 or 0.
- Since we must go from $+1$ to -1 by steps of 1, there must be some i satisfying $1 < i < |w| - 1$ such that $b_i = 0$.
- Decompose $w = yz$, then, taking y to be the i-letter prefix of w and z the rest of w.
- The two substrings y and z are both shorter (and both in L by construction), so we can use $S \to SS \overset{*}{\Rightarrow} Sz \overset{*}{\Rightarrow} yz = w$ as our derivation for w.

Thus, $w \in L(G)$. (This works for Case 4 also, swapping 0 and 1.)

So we are done!

- We were given the language $L = \{w \in \{0,1\}^* \mid n_0(w) = n_1(w)\}$, and told to show it is context free.
- We gave a grammar G, and asserted that $L(G) = L$.
  - We showed that $L(G) \subseteq L$, by showing that any word derived in G had an equal number of 0's and 1's.
  - We showed that $L \subseteq L(G)$, by showing that any word with an equal number of 0's and 1's could be derived in G.
- Hence, L is context free.
- (You can easily prove that L is not regular, using the pumping lemma on $0^n 1^n$.)

Parse trees

We can show how a word is generated from a grammar, using a parse tree.

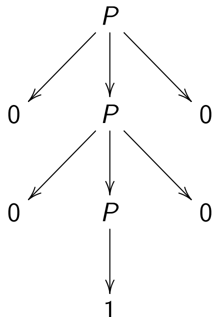
- Root of the tree: A variable in the grammar
- Internal nodes of the tree: variables generated by productions in the grammar
- Leaves of the tree: variables, terminals, or ε, generated by productions

We will show this first with a simple grammar for palindromes:
$G : P \rightarrow \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$.

Palindrome parse trees

Generate 00100 in this grammar:

- First, we replace P with $0P0$.
- Then we replace P with $0P0$ again.
- Finally, we replace P with 1.



Note: A valid parse tree does not have to finish the whole derivation!

Formal definition for parse trees

A formal definition for parse trees: Given a context-free grammar G, a rooted tree is a parse tree for a derivation in G if:

- The root of the tree is a variable in G.
- The interior nodes are labeled by variables in G.
- The leaves of the tree are either labeled by variables of G, terminals in the grammar G, or ε. If a leaf is labeled by ε, then it must be the only child of its parent.
- If there is an internal node labelled A, whose children are labelled $B_1, B_2, \ldots, B_k$, then there must be a production $A \to B_1 B_2 \cdots B_k$ in the grammar.

Any subtree of a parse tree is also a parse tree, unless it is a leaf.

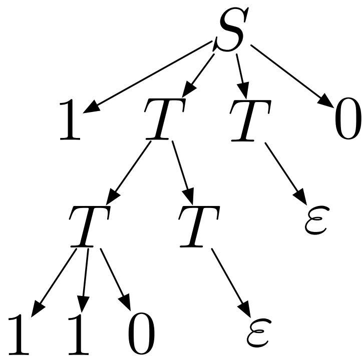
The yield of a parse tree

Parse trees show how we generate a string in the grammar.

- Given a parse tree, the yield of the tree is the concatenation of the symbols at the leaves of the tree, in order from left to right. (Don't include ε in the concatenation.)
- Full parse trees are trees whose root is the start variable in the grammar, and for which all leaves are labelled with terminals or ε.
- In these trees, the yield corresponds to a word in the language of the grammar.

An example of a more complicated parse tree

Here is a full parse tree yielding the string 11100 in the grammar
$G : S \rightarrow 1TT0; T \rightarrow TT \mid S \mid \varepsilon \mid 110 \mid 100.$

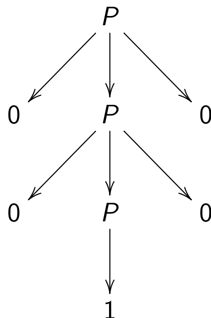


The relationship between leftmost derivations and parse trees

Derivations correspond to parse trees:

- From leftmost derivation from S to a string α, build a parse tree starting with S.
- At each step, add children to the leftmost leaf of the tree labeled with a variable, corresponding to next production in the leftmost derivation.

Example with the palindrome grammar in the derivation $P \rightarrow 0P0 \rightarrow 00P00 \rightarrow 00100$:



Parse trees correspond to derivations

From a parse tree, we can construct a derivation:

- Start with the root of the tree being the start variable in the derivation.
- Then, follow a depth-first, left-most search in the tree.
- Each time we encounter a variable, expand it according to the rule that generates its children.
- If we encounter terminals, do nothing.

This can be made much more formal

We are not going to make a formal proof here.

- You should be able to imagine: both directions are induction proofs.
- To go from parse trees to derivations, we would perform structural induction on the complexity of the tree.

Far more interesting: some words can have different parse trees.

Ambiguity in context-free languages

Context-free grammars can be used to generate expressions in mathematical notation or programming languages.

- Example: $G : S \to S + S \mid S * S \mid (S) \mid a$
- This grammar, for example, generates a or $a + a$ or $(a + a * a)$.

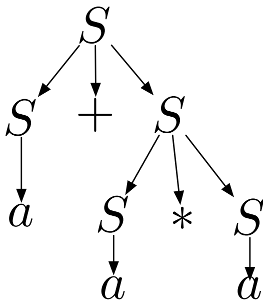
What is the leftmost derivation for the expression $a + a * a$?

- $\mathbf{S} \Rightarrow \mathbf{S} + S \Rightarrow a + \mathbf{S} \Rightarrow a + \mathbf{S} * S \Rightarrow a + a * \mathbf{S} \Rightarrow a + a * a$
- or ...
- $\mathbf{S} \Rightarrow \mathbf{S} * S \Rightarrow \mathbf{S} + S * S \Rightarrow a + \mathbf{S} * S \Rightarrow a + a * \mathbf{S} \Rightarrow a + a * a$

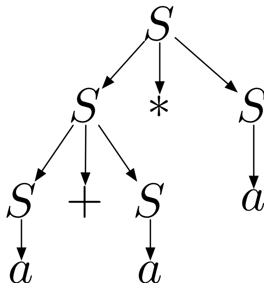
Quite different!

The first starts out with an addition, the second with a multiplication!
Consider their parse trees:

$\mathbf{S} \Rightarrow \mathbf{S} + S \Rightarrow a + \mathbf{S} \Rightarrow$
$a + \mathbf{S} * S \Rightarrow a + a * \mathbf{S} \Rightarrow a + a * a$



$\mathbf{S} \Rightarrow \mathbf{S} * S \Rightarrow \mathbf{S} + S * S \Rightarrow$
$a + \mathbf{S} * S \Rightarrow a + a * \mathbf{S} \Rightarrow a + a * a$



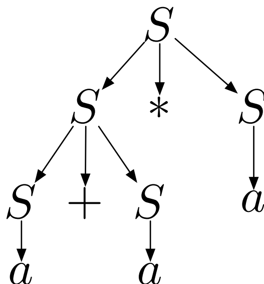
Fundamentally different!

Two derivations with same parse tree

$\mathbf{S} \Rightarrow S*\mathbf{S} \Rightarrow S+S*\mathbf{S} \Rightarrow S+\mathbf{S}*a \Rightarrow \mathbf{S}+a*a \Rightarrow a+a*a$ is not really different from
$\mathbf{S} \Rightarrow S*\mathbf{S} \Rightarrow \mathbf{S}*a \Rightarrow S+\mathbf{S}*a \Rightarrow \mathbf{S}+a*a \Rightarrow a+a*a$.

- Both start with a multiplication.
- Consider them as derivations, where we see how each letter in the end happened, and they are the same.

Both derivations correspond to the same parse tree:



Ambiguity in CFGs

A grammar G is ambiguous if there is a word w in $L(G)$ with more than one leftmost derivation (or parse tree).

- Ambiguity is not a good thing.
- Why? When we parse a word, we want to know how the word was produced: this gives a sense of its meaning.
- With an ambiguous grammar, there may be more than one way.
- This is annoying.

Examples are common in computer languages

An annoying example, from CS.

- The grammar:
  $G : S \rightarrow SS \,|\texttt{if test} : S \,|\texttt{if test} : S \texttt{ else:} S \,|\texttt{foo}$
  generates "programs" like:
  `if test: foo else: foo if test: foo`
- But what about: `if test: if test: foo else: foo`
- To which `if` does the `else` belong?
- This is called the dangling else problem.

Some languages make this easy

In Python, it is really easy to tell by the indentation:

```
if test:
   if test:
      foo
   else:
       foo
```

```
if test:
   if test:
      foo
else:
   foo
```

But what about in other languages?

The fundamental problem: two parse trees

Of course, the problem is that there are two derivations for
`if test: if test: foo else: foo`
And this is the annoyance of ambiguity.
Ambiguous grammars for computer language are bad: we should have exactly one parse for a program.

- Note! This is not the way the English language works!!
- Example: "Time flies like an arrow. Fruit flies like a banana."
- One problem with computer translation of human speech: ambiguity in grammars.

How to get rid of ambiguity?

Inherent ambiguity

Sometimes we cannot get rid of ambiguity.

- Example: $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$ is a CFL, since it is the union of two CFLs. (We will see the closure rules for CFLs in Module 7.)
- $w = a^i b^i c^i$ is in L for any i.
- Must be derivable in two ways:
  - one since w is of the form $a^i b^i c^k$
  - one since w is of the form $a^k b^i c^i$

A context-free language is inherently ambiguous if all grammars for it are ambiguous.
Sometimes, we will have an ambiguous grammar for a language that is not inherently ambiguous. It is best to fix that if we can.

How can we remove ambiguity

Important difference between ambiguity and inherent ambiguity:

- A context-free grammar G is ambiguous if there exists a word $w \in L(G)$ with two different parse trees in G.
- A context-free language L is inherently ambiguous if for all grammars G whose language is L, the grammar G is ambiguous.

Many languages have ambiguous and unambiguous grammars. We would prefer the latter!

How can we avoid ambiguity?

One way: Say the same message in a different way so there is no possible ambiguity:

- This grammar:
  $G : S \rightarrow SS \,|\texttt{if test} : S \,|\texttt{if test} : S \;\texttt{else:}\, S \,|\texttt{foo}$
  is ambiguous.
- It makes sentences like:
  `if test: if test: foo else: foo`
  in two different ways.

But we can communicate the same message differently:

- $G : S \Rightarrow SS \,|\texttt{if test} : S \;\; \texttt{end}|\; \texttt{if test} : S \;\texttt{else:}\, S \;\texttt{end}|\texttt{foo}$
- This new grammar makes sentences like:
  - `if test: if test: foo end else: foo end`
  - `if test: if test: foo else: foo end end`
- Here, it is completely clear with which `if` the `else` pairs.

Languages which are not inherently ambiguous, with obvious grammars that are ambiguous

Can we remove ambiguity from ambiguous grammars?

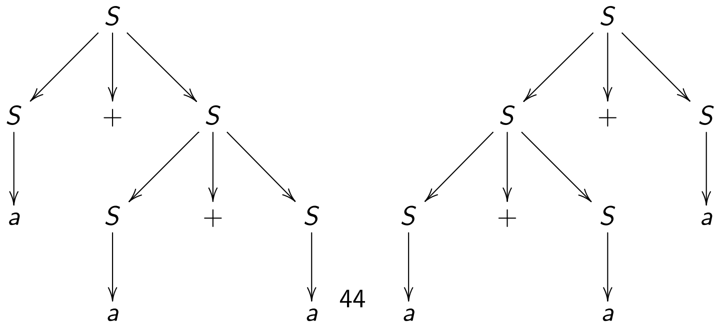
- By definition, yes, but it is not always easy.
- We will do it for algebraic expressions:

Consider the language of the grammar

$$G : S \to S * S \mid S + S \mid (S) \mid a$$

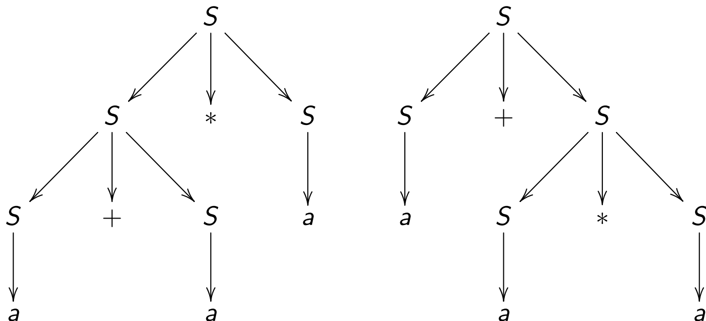
Where does this acquire ambiguity? Two ways:

1. We can derive $a + a + a$ in two different ways: no sense of order in associativity.



Or through lack of operator precedence

2. We can derive $a + a * a$ in two different ways: no sense of operator precedence.



Removing ambiguity

Idea: Separate expressions being multiplied, added or parenthesized

- If we have the sum of two things, we cannot use that as a factor in a product.
- The terms being multiplied in a product have to be either products themselves, or a or a parenthesized expression.
- If we have the sum of three things, we have to have build the first two as a sum, and then sum this with the third.

A new grammar

This idea suggests a grammar G' with three variables:

- E for expressions (which is the start variable): $E \to E + T \mid T$
- T for terms, which can be added together to make expressions: $T \to T * F \mid F$
- F for factors, which can be multiplied into products: either the single terminal a or a parenthesized expression: $F \to (E) \mid a$

(Why not $E \to T + T \mid T$? Because then there is no unique way to derive $a + a + a$. We need to make sure that there is exactly one way of generating every expression!)
This grammar includes the order of operations. It is also unambiguous and generates the same language.

Statement of our result

Theorem: If G is the grammar

$$S \to S + S \mid S * S \mid (S) \mid a,$$

and G' is the grammar with rules:

- $E \to E + T \mid T,$
- $T \to T * F \mid F,$
- $F \to (E) \mid a,$

then G' is unambiguous, and $L(G') = L(G)$.

Not quick: two lemmas

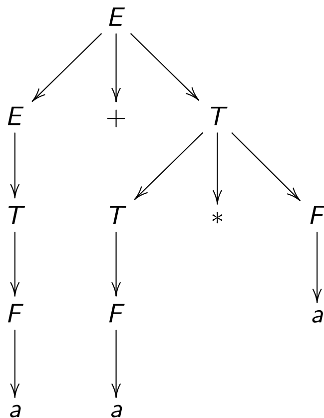
This is not going to be a short proof.
First, consider G':

- If $T \stackrel{*}{\Rightarrow} x$, then x has no $+$ character outside of parentheses.
- If $F \stackrel{*}{\Rightarrow} x$, then x has no $+$ or $*$ characters outside parentheses.
- If x has no $+$ or $*$ characters outside parentheses, then $F \stackrel{*}{\Rightarrow} x$.
- If x has no $+$ characters outside parentheses, then $T \stackrel{*}{\Rightarrow} x$ or $F \stackrel{*}{\Rightarrow} x$.

We will use these lemmas quite a lot.

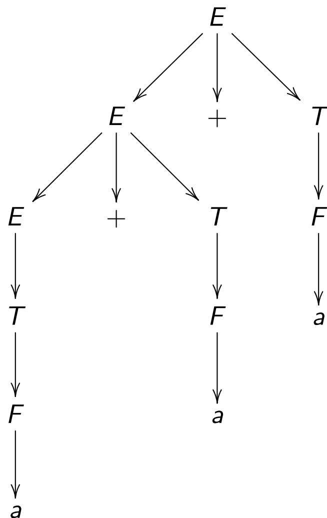
A formerly ambiguous example

In this new grammar, $a + a * a$ has only one parse tree:



The other formerly ambiguous example

And $a + a + a$, similarly, has only one parse tree.



First: G' is unambiguous

We will prove something stronger:

Lemma: If, for some string x containing only alphabet symbols, we have $E \overset{*}{\Rightarrow} x$ or $T \overset{*}{\Rightarrow} x$ or $F \overset{*}{\Rightarrow} x$, then there is a unique left-most derivation for that string from E, T or F, respectively.

Proof: by induction on $|x|$:

- Base case: If $|x| = 1$, then $x = a$.
  - If $E \overset{*}{\Rightarrow} a$, then there is only one possible derivation: $E \Rightarrow T \Rightarrow F \Rightarrow a$.
  - If $T \overset{*}{\Rightarrow} a$, then there is only one possible derivation: $T \Rightarrow F \Rightarrow a$.
  - If $F \overset{*}{\Rightarrow} a$, then there is only one possible derivation: $F \Rightarrow a$.
- Inductive case: Assume that for any string which is strictly shorter than x that we can derive from E, T or F in G', there exists a unique left-most derivation for that string from the variable E, T or F respectively in G'.
- Now, let's look at $x \in L(G')$ and figure out how we got to where we are.

Three cases for the induction

We will examine x and see what it has outside of parentheses:

1. There is a $+$ symbol outside of parentheses in x.
2. There is no $+$ symbol outside of parentheses in x, but there is a $*$ outside of parentheses.
3. There is neither a $+$ symbol nor a $*$ symbol outside of parentheses in x.

Case 1

Assume there is a $+$ symbol outside of parentheses in x.
Basic idea: Writing $x = y + z$, argue that we must start with the rule $E \to E + T$ where $E \overset{*}{\Rightarrow} y$ and $T \overset{*}{\Rightarrow} z$, and then argue that y and z must have unique derivations.

- If x has a $+$ outside of parentheses, then $T \overset{*}{\not\Rightarrow} x$.
- But since $x \in L(G'), E \overset{*}{\Rightarrow} x$.
- So the first rule used in any derivation of x in G' must be $E \to E + T$.
- We know that $x \in L(G')$, so there must be some way, now, to generate $x = y + z$, where $E \overset{*}{\Rightarrow} y$ and $T \overset{*}{\Rightarrow} z$.
- But by our induction hypothesis, the ways of generating y and z are unique, so for this decomposition $x = y + z$, there exists exactly one way of generating x.

Finishing up the first case

Could we choose y and z in different ways?

- No: we know that y is before the last $+$ symbol outside parentheses in x.
- If not, then z has a $+$ symbol outside parentheses, and thus by a Lemma z cannot be generated from T.
- But we know that $T \stackrel{*}{\Rightarrow} z$.

So, for this case, the only leftmost derivation for x is:
$E \Rightarrow E + T \stackrel{*}{\Rightarrow} y + T \stackrel{*}{\Rightarrow} y + z$, and x is unambiguously derived.

Case 2

Assume x has no $+$ outside parentheses, but does have a $*$ outside parentheses.

- We cannot use the $E \to E + T$ rule, so we must use $E \to T$ instead.
- But since there is a $*$ outside parentheses, by a Lemma, $F \stackrel{*}{\not\Rightarrow} x$.
- So we must use the $T \to T * F$ rule instead. We are going to thus be decomposing x into $x = y * z$.
- As in the proof for Case 1 (since $F \stackrel{*}{\Rightarrow} z$), z must have no $*$ outside parentheses, so it is uniquely chosen.
- Again, for $x = y * z$, y and z have unique derivations by induction, and hence there is a unique derivation for x.

Case 3

Assume x has neither * nor $+$ outside parentheses.

- But then $x = (y)$ for some y, since we are **not** in the base case, namely the case where $x = a$.
- We must start the derivation with $E \Rightarrow T \Rightarrow F \Rightarrow (E)$.
- Then we follow the derivation for y, which is unique by the induction hypothesis.

This handles all three possibilities for a word in $L(G')$. In all cases, they have a unique derivation.
Hence, the grammar G' is unambiguous.

Proof that languages are equal

That is the proof that the new grammar is unambiguous.
What about the proof that $L(G) = L(G')$?

- First, we will show that $L(G') \subseteq L(G)$.
- That is, if $x \in L(G')$, then $x \in L(G)$.

Proof by induction on $|x|$:

- Base case: If $|x| = 1$, then $x = a$, since that is the only 1-letter word in $L(G')$, and it is the only 1-letter word in $L(G)$, too.
- Inductive case: inductive hypothesis is that all words in $L(G')$ shorter than x are also in $L(G)$.
- Consider the unique derivation of x in G'.

Inductive proof that $L(G') \subseteq L(G)$

Consider a word $x \in L(G')$ with $|x| > 1$.
How is it derived? Three cases:

1. x derives from the rule $E \to E + T$.
2. x derives from $E \Rightarrow T \Rightarrow T * F \Rightarrow \cdots$
3. x derives from $E \Rightarrow T \Rightarrow F \Rightarrow (E) \Rightarrow \cdots$

First case: x derives from $E \to E + T$

Case 1: x derives from the rule $E \to E + T$.

- Then $x = y + z$, where $E \stackrel{*}{\Rightarrow} y$ and $T \stackrel{*}{\Rightarrow} z$.
- We can derive z in G' using $E \Rightarrow T \stackrel{*}{\Rightarrow} z$.
- Both y and z are in $L(G')$ and are shorter than x, so they are both in $L(G)$ by our inductive hypothesis.
- Using the $S \to S + S$ rule in G, we can thus derive x in $L(G)$.

Now, we just have to finish the other two cases.

The other two cases for $L(G') \subseteq L(G)$

Case 2: x derives from $E \Rightarrow T \Rightarrow T * F \Rightarrow \cdots$

- Then $x = y * z$, where $T \stackrel{*}{\Rightarrow} y$ and $F \stackrel{*}{\Rightarrow} z$.
- Since $T \stackrel{*}{\Rightarrow} y$, we can derive y via $E \Rightarrow T \stackrel{*}{\Rightarrow} y$, and therefore $E \stackrel{*}{\Rightarrow} y$.
- Since $F \stackrel{*}{\Rightarrow} z$, we can derive z via $E \Rightarrow T \Rightarrow F \stackrel{*}{\Rightarrow} z$, and therefore $E \stackrel{*}{\Rightarrow} z$.
- So y and z are in $L(G')$ and are shorter than x.
- By the inductive hypothesis, y and z are in $L(G)$, and we can derive x in G by
$$S \Rightarrow S * S \stackrel{*}{\Rightarrow} y * S \stackrel{*}{\Rightarrow} y * z = x.$$

Case 3: x derives from $E \Rightarrow T \Rightarrow F \Rightarrow (E) \Rightarrow \cdots$.

- Then $x = (y)$, where $E \stackrel{*}{\Rightarrow} y$.
- The induction hypothesis applied to y says $S \stackrel{*}{\Rightarrow} y$.
- Therefore we can derive x in G via
$$S \Rightarrow (S) \stackrel{*}{\Rightarrow} (y) = x.$$

So we see that $L(G') \subseteq L(G)$.

Other Direction: $L(G') \supseteq L(G)$

- We have just shown that $L(G') \subseteq L(G)$.
- Now, we must show that $L(G') \supseteq L(G)$:
  if $x \in L(G)$, then $x \in L(G')$.

Proof by induction on $|x|$.

- Base case: If $|x| = 1$, then $x = a$; seen before.
- Inductive case: Assume all words shorter than x in $L(G)$ are in $L(G')$.
- Then $x = y + z$ or $x = y * z$ or $x = (y)$ for y and z also in $L(G)$.
- We will have three cases, depending on how x can be derived in G.
- As derivations in G need not be unique, we need to carefully state what the cases are.

Inductive part of the proof

Cases:

1. x has at least one derivation of the form $S \Rightarrow (S) \stackrel{*}{\Rightarrow} (y)$, where $x = (y)$.
2. x has no such derivation, but has at least one derivation of the form $S \Rightarrow S + S \stackrel{*}{\Rightarrow} y + z$, where $x = y + z$.
3. The only derivations for x start by using the production $S \rightarrow S * S$.

<u>First case:</u> x has at least one derivation of the form $S \Rightarrow (S) \stackrel{*}{\Rightarrow} (y)$, where $x = (y)$. Suppose that one derivation for x in G is:

$$S \Rightarrow (S) \stackrel{*}{\Rightarrow} (y).$$

- Then in G' we can use the derivation: $E \Rightarrow T \Rightarrow F \Rightarrow (E) \cdots$
- Since $S \underset{G}{\stackrel{*}{\Rightarrow}} y$, and y is shorter than x, our inductive hypothesis tells us that $E \underset{G'}{\stackrel{*}{\Rightarrow}} y$.
- Therefore in G' we have $E \Rightarrow T \Rightarrow F \Rightarrow (E) \stackrel{*}{\Rightarrow} (y) = x$, which witnesses the fact that $x \in L(G')$.

Second case: $S \Rightarrow S + S \stackrel{*}{\Rightarrow} y + z$, where $x = y + z$

Suppose that x has no derivation as in Case 1, but has at least one derivation in G of the form $S \Rightarrow S + S \stackrel{*}{\Rightarrow} y + z$, where $x = y + z$.

- Problem: we want to start the derivation of x in G' using the production: $E \to E + T$.
- By the induction hypothesis, $E \underset{G'}{\stackrel{*}{\Rightarrow}} y$ and $E \underset{G'}{\stackrel{*}{\Rightarrow}} z$, but to apply the desired production in G', we need to know that $T \underset{G'}{\stackrel{*}{\Rightarrow}} z$.
- Idea: Choose y as long as possible, so that z has no $+$ symbols outside of parentheses.
- Then, as above, we still have $E \stackrel{*}{\Rightarrow} y$, but now we can only use $E \Rightarrow T \stackrel{*}{\Rightarrow} z$ to derive z in G'.
- But then we do have $T \underset{G'}{\stackrel{*}{\Rightarrow}} z$, as required.
- Putting it all together, in G' we have $E \Rightarrow E + T \stackrel{*}{\Rightarrow} y + z = x$.
- This witnesses the fact that x is in $L(G')$.

Final case: $S \Rightarrow S * S$, where $x = y * z$

This last case is quite similar.

- Suppose that the only derivations for x start by using the production $S \to S * S$.
- Because x has no derivation beginning with $S \to S + S$, therefore every $+$ in x must be inside parentheses.
- If x contains an exposed $+$, then we can write $x = y + z$, with $S \overset{*}{\Rightarrow} y$ and $S \overset{*}{\Rightarrow} z$, so that we can always find a derivation for x which starts with $S \to S + S$, contradicting the case that we are in.
- Write $x = y * z$, choosing y as long as possible such that we can still derive both y and z from S in G.
- Then y is a word in $L(G)$, shorter than x, and hence by the induction hypothesis is in $L(G')$.
- But then y must be derivable in G' via $E \Rightarrow T \overset{*}{\Rightarrow} y$, since y has no exposed $+$ characters.
- Also, z must have no exposed $+$ or $*$ characters, yet still be in $L(G)$.
- Therefore z is derivable in G' by $E \Rightarrow T \Rightarrow F \overset{*}{\Rightarrow} z$.
- So finally in G' we can derive x by $E \Rightarrow T \Rightarrow T * F \overset{*}{\Rightarrow} y * F \overset{*}{\Rightarrow} y * z$.
- Thus $x \in L(G')$.

Recap of the proof

We are done: we have just finished showing that $L(G') \supseteq L(G)$.
Overall, we have shown:

- G' is unambiguous
- Words in $L(G')$ are in $L(G)$.
- Words in $L(G)$ are in $L(G')$.

(All 3 proofs by induction.)
The third proof was probably hardest: the content was to reconstruct the way in G' to derive each word of $L(G)$.
Hence, $L(G)$ is not inherently ambiguous: G' is an unambiguous grammar for it.

What does this show us?

Ambiguity can, sometimes, be removed:

- Identify the source of ambiguity
- Re-organize to identify precedence or other desired grammar rules
- These proofs are long, but useful: we really do need unambiguous grammars in practice.

End of Module 5

Topics of Module 5:

- Definitions of context-free languages and grammars
- Parse trees and derivations
- Ambiguity in context-free languages and grammars

Next module: the automata that accept context-free languages