CSE396 Lecture Thu. 3/11: The Meanings of Context-Free Grammars

[Picking up from the first example]

Example

$G = S \to 0S1 \mid \epsilon$ Here the start symbol S is the only variable, and $\Sigma = \{0, 1\}$. Some derivations:

- $S \implies \epsilon$. This is a one-step derivation using the rule $S \to \epsilon$. The empty string counts as a terminal string since it belongs to Σ^* (as well as to $(\Sigma \cup V)^*$). So $\epsilon \in L(G)$.
- $S \implies 0S1 \implies 01$. The first step used the rule $S \to 0S1$, then we used the ϵ-rule. (Not all grammars have ϵ-rules, and we will later want to eliminate them from those that do.)
- $S \implies 0S1 \implies 00S11 \implies 0011$.

This is enough to give the idea of why $L(G) = \left\{0^n 1^n : n \geq 0\right\}$.

Example

$G = S \to \epsilon \mid (S) \mid SS$. Again S is the only variable, but $\Sigma = \{(,)\}$ instead of $\{0, 1\}$.

The first two rules are much the same as in the first grammar (the order of writing the possible right-hand sides does not matter, and the only reason the order of writing rules for different variables might matter is if you need to put rule(s) for the start symbol first in order to say which it is). But the third rule "expands" by having two (or more) variables on the RHS.

- We can derive ϵ, $()$, $(())$, $((()))$, etc., much as in the previous grammar.
- But we can also do $S \implies SS \implies (S)S \implies ()S \implies ()(S) \implies ()()$ to get other kinds of balanced-parenthesis expressions.
- In fact, $L(G) = BAL$.

Example

$G' = S \to \epsilon \mid (S) \mid (S)S$ is another grammar involving parentheses.

- Anything G' can derive can be derived in G because the rule $S \to (S)S$ in G' can be **simulated** by the two steps $S \implies SS \implies (S)S$ in G.
- Hence $L(G') \subseteq L(G)$. Since we already asserted that $L(G) = BAL$, this means $L(G') \subseteq BAL$. We therefore say that G' is **sound** for BAL.
- Is $L(G') = BAL$, which would follow if $L(G') \supseteq BAL$? The latter I call G' being **comprehensive** for BAL.
- The combination of being sound and comprehensive just means being *correct*.
- In fact, yes, G' is comprehensive---but that is often not as easy to prove as soundness.

Example

$G'' = S \to \epsilon \mid 0S \mid \$S \mid \$SDS$ generates all strings in the spears-and-dragons game with unlimited spears in which the "Player" survives. We can make it look more like G' by changing "spear" to (and

"dragon" to) and ignoring 0 for "empty room":

$G'' \ = \ S \ \rightarrow \ \epsilon \ | \ (S \ | \ (S)S.$

Then $L(G'')$ is the language of parenthesis expressions that can be properly closed by appending zero or more right parens.

Example with more than one variable:

$S \ \rightarrow \ \epsilon \mid aB \mid b\ A$
$A \ \rightarrow a \mid aS \mid bAA$
$B \ \rightarrow \ b \mid bS \mid aBB$

What is the language? Think of the variables as saying:

- S: "Every string I derive has equal #s of a's and b's---and I derive all such strings."
- A: "Every string I derive has one more a than b---and I derive all such strings."
- B: "Every string I derive has one more b than a---and I derive all such strings."

Every rule is **sound** in the sense that if each variable on the right-hand side does what it says, then the variable on the left-hand side fulfills its promise. This means in particular that L_S, which is the language of G, is a subset of $EQ \ = \ \left\{x \ \in \ \{a,b\}^* : \ \#a(x) \ = \#b(x)\right\}$. Thus we can say that G is **sound** for the target language EQ. If it is also **comprehensive**, meaning $L(G) \supseteq EQ$, then in this case it will be **correct**, of course meaning $L(G) = EQ$. Yes, G is correct---but comprehensiveness is often a lot harder to prove than soundness, because you can't just examine individual rules but have to use all the rules in concert. [One fact that helps in this case is that G is in **Greibach normal form** (Sheila Greibach, UCLA), meaning that the RHS of every rule begins with a terminal letter---with $S \rightarrow \epsilon$ as an allowed exception. This form helps you **parse** strings in a left-to-right manner.]

Example: Palindromes and Even Palindromes

The CFG $G \ = \ S \ \rightarrow \ aSa \mid bSb \mid \epsilon \mid a \mid b$ (with $\Sigma = \{a,b\}$) makes $L(G) = PAL$. If we take away the terminal rules $S \rightarrow a$ and $S \rightarrow b$, leaving $G' \ = \ S \ \rightarrow \ aSa \mid bSb \mid \epsilon$, then we get $L(G')$ equal to the language of even-length palindromes: EVENPAL = $\left\{x \cdot x^R : \ x \in \{a,b\}^*\right\}$.

Non-Example: The Double-Word Language

Recall DOUBLEWORD = $\left\{x \cdot x : \ x \in \{a,b\}^*\right\}$. This looks even simpler than EVENPAL. However, there does not exist a CFG G such that $L(G) =$ DOUBLEWORD. We will later prove this via the CFL Pumping Lemma when we hit section 2.3 (after jumping over most of section 2.2). As a self-study

challenge, however, see if you can figure out the language of the grammar

$$
\begin{aligned}
S &\rightarrow AB \mid BA \\
A &\rightarrow aAa \mid aAb \mid bAa \mid bAb \mid a \\
B &\rightarrow aBa \mid aBb \mid bBa \mid bBb \mid b \ .
\end{aligned}
$$

(The final period is just punctuation, not part of Σ.) As a warmup, can this grammar ever derive a string of odd length? We will come back to this later.

What is a "Language", Anyhow?

We have defined *language* to mean *a set of strings*, but when we try to apply that to human languages, that's like equating English with the set of words in some reference dictionary. The real unit of human language---true in all cultures---is the *sentence* rather than the *word*. How sentences are formed and interpreted is what we call the "rules of grammar."

Indeed, what is IMHO remarkable is the lack of rules for units higher than a sentence. We have the notion of "paragraph" but it is highly flexible. Newspapers keep them short, blogs try to, but some famous novels run paragraphs for pages and pages. You may have been taught that an essay is composed of an introduction, body, and conclusion, and there are prescribed formats of kinds of business letters, for instance. But if you violated those higher-level rules, it wouldn't make what you wrote unintelligible. Your boss would still get it.

Whereas, meant out be really what to may impossible it structure readers you Yoda
text you or and violate of even order phrase rules if word the for figure .

What struck Noam Chomsky in the 1950s was that although different human languages have different rules for sentences, the natures of those rules are much the same. To a (debatably) large extent, they can be given as CFG rules. One result was an effort toward systemabstrimplification of how grammar was taught in schools. When I was in primary school, I recall a book that had

S -→ N V

The intent, rendered more accurately in BNF style as in the text, was

```
<sentence>  ::=  <noun-phrase> <verb-phrase>
```

That rule applies to the great majority of sentences in English--where `<verb-phrase>` can expand to allow direct and indirect objects and other forms that can involve more noun phrases. Does every full sentence follow that rule? At least every non-interrogative sentence? Think about it!

In English we can further expand:

```
<noun-phrase> ::=  <noun>  |  <article> <noun>
                           |  <adjective> <noun-phrase>
                           |  <noun-phrsase> <prep-phrase>
```

The rule `<noun-phrase> ::= <adjective> <noun-phrase>` allows you to put one or any number of adjectives before a noun---with the zero option coming in if you use one of the first two rules immediately. In "extended BNF" notation you can use square brackets to indicate optional stuff and braces for zero-or-more (just like Kleene star), so we could write more compactly:

```
<noun-phrase>  ::=  [<article>] {<adjective>} <noun> {<prep-phrase>}
```

(Actually, this is not equivalent to the above grammar---it fixes an error in the placement of articles that actually requires having a separate variable saying the article is optional with the rule `<art-opt-noun-phrase> ::= <article> <noun-phrase>  |  <noun-phrase>`.)

An example taking the optional article, the zero option for adjectives, and one prepositional phrase is "the cat in the hat". We could extend it to be "the cat in the hat with a bat." It is curious that those phrases are modifiers like adjectives are but come after the noun. We could have said, "the hat-wearing, bat-carrying cat."

Let's just use N for noun, N_P for noun-phrase, and A for adjective. The rule

$$N_P \rightarrow A\,N_P \mid N$$

places no limit on the number of adjectives we can have before a noun. It might seem sensible to have a limit like 3 or 4, but it is actually both *simpler* not to impose a limit, and more indicative of how we talk---or can talk---especially in the heat of the moment. For instance,

"You are a dirty rotten stinking lying skunk!"

This applies $N_P \rightarrow AN_P$ four times before terminating with $N_P \rightarrow N$ at the word "skunk". Now French has a different rule, basically $N_P \rightarrow N_P\,A$ so that adjectives come after the noun (but not exclusively, as we'll see). Let's try insulting "Pepé Le Pew" by translating this to French on Google...

[Try the above on Google Translate. You may get some surprises, such as GT thinking that "lying" means lying down. Change "lying" to "fibbing" or "untruthful". Then try translating the French back to English (but if you get the word "putain" in the French, don't). See if you can get something that keeps coming back the same when you go back and forth, so that GT's French and English agree on what is being said. The most relevant part is that the adjective for "dirty", *sale*, will generally stay in front.]

What English and French share is not the vocabulary or rules but the sameness of the nature of the rules. That sameness extends to non-Indo-European languages. Isolated language communities were

found to have rules that can be modeled to a similarly large extent by CFG rules. The CFG rules don't catch everything, but they catch a lot, and they appear to matter to our brains in a way that ***precedes*** the meaning of the words. Chomsky's famous sentence to illustrate this is:

Colorless green ideas sleep furiously.

It makes poetic sense, despite the first two words contradicting each other, and the last two words... Whereas, there are times when even if we completely know in advance what a speaker is going to say we can still get uptight if we have to wait...

Chomsky's "Rationalist Thesis" is that our acquisition of language is not wholly a product of picking it up from our childhood environment---that our brains are "pre-wired" for it. The question of how far the language facility is genetic remains controversial (not to mention its embroilment in larger questions about intelligence), but despite evidence about particular genes and their effects, there does not seem (to me) to be a clear genetic blueprint. The "Rationalist Thesis" does include the option that our mental sensitivity to CFG rules is inherent in mathematics itself, and that is a helpful standpoint to consider for understanding this course: the class **CFL** of context-free languages, like its proper subclass **REG** of regular languages, is **salient**. Together with the class **CSL** of **context-sensitive languages** (which we will only mention in May) and the class **RE** of **computably enumerable** languages (which we will see earlier, by April 1!), these form the rungs of the **Chomsky Hierarchy**.

The fact that Chomsky fiddled around with **CSL** should tell you that CFGs were soon found to be well short of perfect for describing *human* languages. CFGs turned out, however, to be "da bomb" for *programming* languages. We can get a taste of why.

Example: Numerical Expressions

Let's use BNF notation style for this one, and let's call the start symbol E.

E ::= $(E+E)$ | $(E-E)$ | $(E*E)$ | (E/E) | <variable> | <constant>
<variable> ::= *any alphanumeric legal identifier*
<constant> ::= *any legal numeric literal.*

The terminal alphabet Σ includes the parentheses, the operator symbols +,-,*,/, and whatever letters and digits and other punctuation are allowed in variables and constants. We don't want to have to specify the last of these. What we could do is treat the *tokens* <var> and <const> as if they were members of Σ. The text gets around this issue by pretending that a is the only variable and ignoring constants, but being "a little more real" won't hurt us. What we actually do is allow <var> and <const> to derive any legal identifier or constant in one step.

Now the above grammar generates only *fully parenthesized* expressions. It doesn't let you write $a - b + c$ or even $xy + z$. We can get them if we make the parentheses optional:

E ::= $E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid$ <var> | <const>

Now we can derive them---note I write $E \implies^2 a$ to shortcut $E \implies$ <var> $\implies a$, etc.

$$E \implies E - E \implies^2 a - E \implies a - E + E \implies^2 a - b + E \implies^2 a - b + c \ .$$

$$E \implies E * E \implies^2 x * E \implies x * E + E \implies^2 x * y + E \implies^2 x * y + z \ .$$

Does anything about these derivations trouble you? I will say that this "liberal" grammar G generates all and only legal numeric expressions, but it "tells fibs" while doing so. The **sentiential form** $a - E$ seems to say that the whole rest of the expression gets subtracted from a, but that is not how we read the expression $a - b + c$ under the **left-to-right associativity** rule. More clearly (but less insidiously), the sentiential form $x * E$ seems to say that x will multiply both terms in the expression $y + z$ derived from that E, but it only multiplies y in $xy + z$. Perhaps most insidiously, what about the expression a / bc? You might read it as if the intent were $\frac{a}{bc}$ but it will get **parsed** as $(a / b) * c$ because $/$ and $*$ have equal **precedence**---at least in C/C++/Java/Python/etc. How can we write a grammar to reflect precedence (and associativity)? The answer is to add variables for the extra **syntactic categories** "term" and "factor":

E ::= $T \mid E + T \mid E - T$
T ::= $F \mid T * F \mid T / F$
F ::= $(E) \mid$ <var> | <const>

Now if we try to imitate the first derivation above by putting the minus sign $-$ in first, we get:

$$E \implies E - T \implies a - T$$

and we're stuck: there isn't a rule with $+$ for T. To get $a - b + c$ we now must do

$$E \implies E + T \implies E - T + T \implies T - T + T \implies F - T + T \implies^2 a - T + T \implies^6 a - b + c.$$

The sentential form $T - T + T$ reads the three terms left-to-right (even though the leftmost term was derived last) at equal level, rather than grouping the last two. Likewise, the only way to derive $xy + z$ is by putting out the $+$ first rather than the $*$ first as before---in terms you may have heard already, the $+$ is the "topmost" or "outermost" operator. The derivation

$$E \implies E + T \implies T + T \implies T * F + T \implies F * F + T \implies^4 x * y + T \implies^3 x * y + z$$

now makes clear that x was never intended to multiply z. We can also still write the fully-parenthesized forms if we wish, as well as options in-between, even silly but legal ones like $(x*(y) + ((z)))$. We can also tack on more syntactic categories, such as having a <factor> involve powers. Some programming languages have a native operation for powers like $**$, but you have to be careful that it is **right-associative**: $a**b**c$ means $a**(b**c) = a^{b^c}$, not $(a**b)**c) = \left(a^b\right)^c$ because the latter just becomes a^{bc}. In practice, the part of the grammar for expressions in modern programming languages has a dozen or two dozen variables (i.e., syntactic categories). But the point is that not only is the grammar able perfectly to describe the **syntax** of the language (still falling short of checking consistency of types and the number/sequence of arguments in function/method calls, which Ada95 for one called the "semantic" phase), the grammar also is instrumental to write the compiler's **parsing** stage. So let's move on to parsing---still in section 2.1 but not intending to go into the compiler-level detail of the later section 2.4.

Parse Trees, Leftmost Derivations, and Ambiguity.

[notes to follow---most of this will be next Thursday, after the exam on Tuesday]