CSCI 340: Computational Models

Context-Free Grammars

Chapter 12

Department of Computer Science

Syntax as a Method for Defining Languages

Originally needed a way to write complicated expressions on one line

$$\frac{\frac{1}{2}+9}{4+\frac{8}{21}+\frac{5}{3+\frac{1}{2}}}$$

vs.

$$(((1/2)+9)/(4+(8/21)+(5/(3+(1/2)))))$$

Syntax to Machine Executable Code

- Conversion from high-level language to machine-executable code is done by a **compiler**
- Must determine the **order** of instructions executed
- Must determine the underlying **meaning**

Example: *Arithmetic Expressions*

1. Any number is in the set AE
2. If x and y are in the set AE, then so are:
   $(x) \quad -(x) \quad (x+y) \quad (x-y) \quad (x*y) \quad (x/y) \quad (x**y)$

Sample Input:

$$((3+4)*(6+7))$$

$((3+4)*(6+7))$

Rule Expansion:

3 is in *AE*
4 is in *AE*
$(3+4)$ is in *AE*
6 is in *AE*
7 is in *AE*
$(6+7)$ is in *AE*
$((3+4)*(6+7))$ is in *AE*

Algorithmic Conversion:

`LOAD` 3 into R1
`LOAD` 4 into R2
`ADD` contents of R1 and R2 into R3
`LOAD` 6 into R4
`LOAD` 7 into R5
`ADD` contents of R4 and R5 into R6
`MUL` contents of R3 and R6 into R7

In order to do *any* of this, we need to **parse** the expression. In the case of *AE*, this is a *generative grammar*

Syntax-Defining Languages – English?

1. A <u>*sentence*</u> can be a <u>*subject*</u> followed by a <u>*predicate*</u>
2. A <u>*subject*</u> can be a <u>*noun-phrase*</u>
3. A <u>*noun-phrase*</u> can be an <u>*adjective*</u> followed by a <u>*noun-phrase*</u>
4. A <u>*noun-phrase*</u> can be an <u>*article*</u> followed by a <u>*noun-phrase*</u>
5. A <u>*noun-phrase*</u> can be a <u>*noun*</u>
6. A <u>*predicate*</u> can be a <u>*verb*</u> followed by a <u>*noun-phrase*</u>
7. A <u>*noun*</u> can be

   apple *bear* *cat* *dog*
8. A <u>*verb*</u> can be

   eats *follows* *gets* *hugs*
9. An <u>*adjective*</u> can be

   itchy *jumpy*
10. An <u>*article*</u> can be

    a *an* *the*

The itchy bear hugs the jumpy dog

sentence	
subject *predicate*	Rule 1
noun-phrase *predicate*	Rule 2
noun-phrase *verb* *noun-phrase*	Rule 6
article *noun-phrase* *verb* *noun-phrase*	Rule 4
article *adjective* *noun* *verb* *noun-phrase*	Rule 3
article *adjective* *noun* *verb* *article* *noun-phrase*	Rule 5
article *adjective* *noun* *verb* *article* *adjective* *noun-phrase*	Rule 4
article *adjective* *noun* *verb* *article* *adjective* *noun*	Rule 3
the *adjective* *noun* *verb* *article* *adjective* *noun*	Rule 10
the itchy *noun* *verb* *article* *adjective* *noun*	Rule 9
the itchy bear *verb* *article* *adjective* *noun*	Rule 7
the itchy bear hugs *article* *adjective* *noun*	Rule 8
the itchy bear hugs the *adjective* *noun*	Rule 10
the itchy bear hugs the jumpy *noun*	Rule 9
the itchy bear hugs the jumpy dog	Rule 7

Grammar Nonsense

Given the rules listed, we can construct the following:

itchy itchy itchy itchy bear

This is gross but possible. We could rewrite some of our grammar!

noun-phrase → *adjective* * *noun*

We can also have our own number of dumb sentences, but it's still *valid*. Because we don't consider semantics, diction, or any sense – really – we call this a "formal language"

Arithmetic Expression

$$
\begin{aligned}
\underline{\text{Start}} &\rightarrow (\underline{\text{AE}}\,) \\
\underline{\text{AE}} &\rightarrow (\underline{\text{AE}}\; + \underline{\text{AE}}\,) \\
\underline{\text{AE}} &\rightarrow (\underline{\text{AE}}\; - \underline{\text{AE}}\,) \\
\underline{\text{AE}} &\rightarrow (\underline{\text{AE}}\; * \underline{\text{AE}}\,) \\
\underline{\text{AE}} &\rightarrow (\underline{\text{AE}}\; / \underline{\text{AE}}\,) \\
\underline{\text{AE}} &\rightarrow (\underline{\text{AE}}\; {*}{*} \underline{\text{AE}}\,) \\
\underline{\text{AE}} &\rightarrow (\underline{\text{AE}}\,) \\
\underline{\text{AE}} &\rightarrow\; -(\underline{\text{AE}}\,) \\
\underline{\text{AE}} &\rightarrow\; -(\underline{\text{ANY-NUMBER}}\,) \\
\underline{\text{ANY-NUMBER}} &\rightarrow \underline{\text{FIRST-DIGIT}} \\
\underline{\text{FIRST-DIGIT}} &\rightarrow \underline{\text{FIRST-DIGIT}}\; \underline{\text{OTHER-DIGIT}} \\
\underline{\text{FIRST-DIGIT}} &\rightarrow 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9 \\
\underline{\text{OTHER-DIGIT}} &\rightarrow 0 \quad 1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6 \quad 7 \quad 8 \quad 9
\end{aligned}
$$

Generative Grammars

All substitutions made are always of one of the following two forms:

$$\underline{\text{Non-Terminal}} \rightarrow \underline{\text{Non-Terminal-1}} \ \dots \ \underline{\text{Non-Terminal-N}}$$

or

$$\underline{\text{Non-Terminal}} \rightarrow \text{Terminal-1} \dots \text{Terminal-N}$$

- The sequence of repetitive applications of rules is called a **derivation** or **generation** of a word.
- The grammatical rules are known as **productions**.
- *There is no guarantee the derivation will be unique*

These are known as Context-Free Grammars (or CFGs)

Context-Free Grammars

Definition

A **context-free grammar, CFG,** is a collection of three things:

1. An alphabet Σ of letters called terminals from which we are going to make strings that will be the words of a language
2. A set of symbols called non-terminals, one of which is the symbol S, standing for "start here"
3. A finite set of productions of the form:
   <u>NT</u> $\rightarrow$ finite string of *terminals* and/or <u>NT</u> 's

   where the strings of terminals and non-terminals can consist:

   - **of any mixture** of terminals or non-terminals, or
   - *the empty string.*

   One production **must** have the non-terminal S as its left side.

Non-terminals are often CAPITALIZED; terminals are usually lowercase

Context-Free Languages

Definition

The **language generated** by a CFG is the set of all strings of terminals that can be produced from the start symbol S using the productions as substitutions. A language generated by a CFG is called a **context-free language**, abbreviated **CFL**.

Other terms used:

- language defined by the CFG
- language derived from the CFG
- language produced by the CFG

Example

Let the only terminal be a and the productions be:

1. $S \rightarrow aS$
2. $S \rightarrow \lambda$

Apply Prod-1 six times and then apply Prod-2:

$$
\begin{aligned}
&\Rightarrow aS \\
&\Rightarrow aaS \\
&\Rightarrow aaaS \\
&\Rightarrow aaaaS \\
&\Rightarrow aaaaaS \\
&\Rightarrow aaaaaaS \\
&\Rightarrow aaaaaa\lambda \\
&= aaaaaa
\end{aligned}
$$

What language does this define?

More examples

Example ($\lambda \neq \Lambda$)

1. $S \longrightarrow SS$
2. $S \longrightarrow a$
3. $S \longrightarrow \Lambda$

Here, Λ represents it can be removed from the final string, but it is neither terminal nor non-terminal

Example

1. $S \longrightarrow aS$
2. $S \longrightarrow bS$
3. $S \longrightarrow a$
4. $S \longrightarrow b$

Two more Examples

Example

1. $S \longrightarrow X$
2. $S \longrightarrow Y$
3. $X \longrightarrow \Lambda$
4. $Y \longrightarrow aY$
5. $Y \longrightarrow bY$
6. $Y \longrightarrow a$
7. $Y \longrightarrow b$

Example

1. $S \longrightarrow aS$
2. $S \longrightarrow bS$
3. $S \longrightarrow \Lambda$

Perhaps a useful grammar?

Example

1. $S \rightarrow XaaX$
2. $X \rightarrow aX$
3. $X \rightarrow bX$
4. $X \rightarrow \Lambda$

Perhaps a useful grammar?

Example

1. $S \longrightarrow XaaX$
2. $X \longrightarrow aX$
3. $X \longrightarrow bX$
4. $X \longrightarrow \Lambda$

$(\mathbf{a} + \mathbf{b})^*\mathbf{aa}(\mathbf{a} + \mathbf{b})^*$

Defining a “complicated” regular language

Example

1. $S \longrightarrow SS$
2. $S \longrightarrow BS$
3. $S \longrightarrow SB$
4. $S \longrightarrow \Lambda$
5. $S \longrightarrow USU$
6. $B \longrightarrow aa$
7. $B \longrightarrow bb$
8. $U \longrightarrow ab$
9. $U \longrightarrow ba$

Defining non-regular languages

Example

1. $S \longrightarrow aSb$
2. $S \longrightarrow \Lambda$

Example

1. $S \longrightarrow aSa$
2. $S \longrightarrow bSb$
3. $S \longrightarrow \Lambda$

Example

1. $S \longrightarrow aSa$
2. $S \longrightarrow b$

EQUAL

Example

1. $S \longrightarrow aB$
2. $S \longrightarrow bA$
3. $A \longrightarrow a$
4. $A \longrightarrow aS$
5. $A \longrightarrow bAA$
6. $B \longrightarrow b$
7. $B \longrightarrow bS$
8. $B \longrightarrow aBB$

Why does this work?

Compression of Syntax

It is common for the same non-terminal to be the left side of more than one production. We introduce the symbol " | ", a vertical line, to mean disjunction (or).

$$S \to aS$$
$$S \to \Lambda$$

$$S \to aS \mid \Lambda$$

$$S \to X$$
$$S \to Y$$
$$X \to \Lambda$$
$$Y \to aY$$
$$Y \to bY$$
$$Y \to a$$
$$Y \to b$$

$$S \to X \mid Y$$
$$X \to \Lambda$$
$$Y \to aY \mid bY \mid a \mid b$$

Ambiguity

Definition

A CFG is called **ambiguous** if for at least one word in the language that it generates there are two possible derivations of the word that correspond to different *syntax trees*. If a CFG is not ambiguous, it is called **unambiguous**.

Example

$S \rightarrow aSa \mid bSb \mid a \mid b \mid \Lambda$

Example

$S \rightarrow aS \mid Sa \mid a$