Syntax as a Method for Defining Languages

Originally needed a way to write complicated expressions on one line

\[
\frac{1}{2} + 9 \quad \frac{4 + \frac{8}{21} + \frac{5}{3 + \frac{1}{2}}}{\text{vs.}}
\]

\[
(((1/2)+9)/(4+(8/21)+(5/(3+(1/2))))))
\]
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:
   $\begin{align*}
   (x) & \quad -(x) & \quad (x + y) & \quad (x - y) & \quad (x \times y) & \quad (x/y) & \quad (x \times \times y)
   \end{align*}$

Sample Input:

$$((3 + 4) \times (6 + 7))$$
\[(3 + 4) \times (6 + 7)\]

**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 \(AE\)
\(((3 + 4) \times (6 + 7))\) 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 sentence can be a subject followed by a predicate
2. A subject can be a noun-phrase
3. A noun-phrase can be an adjective followed by a noun-phrase
4. A noun-phrase can be an article followed by a noun-phrase
5. A noun-phrase can be a noun
6. A predicate can be a verb followed by a noun-phrase
7. A noun can be
   apple   bear   cat   dog
8. A verb can be
   eats   follows   gets   hugs
9. An adjective can be
   itchy   jumpy
10. An article can be
    a   an   the
The itchy bear hugs the jumpy dog

sentence
subject predicate
noun-phrase predicate
noun-phrase verb noun-phrase
article noun-phrase verb noun-phrase
article adjective noun verb noun-phrase
article adjective noun verb article noun-phrase
article adjective noun verb article adjective noun-phrase
article adjective noun verb article adjective noun
the adjective noun verb article adjective noun
the itchy noun verb article adjective noun
the itchy bear verb article adjective noun
the itchy bear hugs article adjective noun
the itchy bear hugs the adjective noun
the itchy bear hugs the jumpy noun
the itchy bear hugs the jumpy dog
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!

\[
\text{noun-phrase} \rightarrow \text{adjective} \ast \text{noun}
\]

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

Start → (AE )
AE → (AE + AE )
AE → (AE − AE )
AE → (AE * AE )
AE → (AE / AE )
AE → (AE **AE )
AE → (AE )
AE → − (AE )
AE → − (ANY-NUMBER )

ANY-NUMBER → FIRST-DIGIT
FIRST-DIGIT → FIRST-DIGIT OTHER-DIGIT
FIRST-DIGIT → 1 2 3 4 5 6 7 8 9
OTHER-DIGIT → 0 1 2 3 4 5 6 7 8 9
Generative Grammars

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

\[
\text{Non-Terminal} \rightarrow \text{Non-Terminal-1} \ldots \text{Non-Terminal-N} \\
\text{or} \\
\text{Non-Terminal} \rightarrow \text{Terminal-1} \ldots \text{Terminal-N}
\]

- The sequence of repetitive applications of rules is called a \textit{derivation} or \textit{generation} of a word.
- The grammatical rules are known as \textbf{productions}.
- \textit{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 $\Sigma$ 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: $\text{NT} \rightarrow \text{finite string of terminals and/or NT’s}$

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.
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:

$\Rightarrow aS$
$\Rightarrow aaS$
$\Rightarrow aaaS$
$\Rightarrow aaaaaS$
$\Rightarrow aaaaaaS$
$\Rightarrow aaaaaaaS$
$\Rightarrow aaaaaaa\lambda$
$= aaaaaa$

What language does this define?
More examples

Example ($\lambda \neq \Lambda$)

<table>
<thead>
<tr>
<th></th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$S \rightarrow SS$</td>
</tr>
<tr>
<td>2</td>
<td>$S \rightarrow a$</td>
</tr>
<tr>
<td>3</td>
<td>$S \rightarrow \Lambda$</td>
</tr>
</tbody>
</table>

Here, $\Lambda$ represents it can be removed from the final string, but it is neither terminal nor non-terminal.

Example

<table>
<thead>
<tr>
<th></th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$S \rightarrow aS$</td>
</tr>
<tr>
<td>2</td>
<td>$S \rightarrow bS$</td>
</tr>
<tr>
<td>3</td>
<td>$S \rightarrow a$</td>
</tr>
<tr>
<td>4</td>
<td>$S \rightarrow b$</td>
</tr>
</tbody>
</table>
Two more Examples

Example

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

Example

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

$$(a + b)^*aa(a + b)^*$$
Defining a “complicated” regular language

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. $S \rightarrow SS$</td>
</tr>
<tr>
<td>2. $S \rightarrow BS$</td>
</tr>
<tr>
<td>3. $S \rightarrow SB$</td>
</tr>
<tr>
<td>4. $S \rightarrow \Lambda$</td>
</tr>
<tr>
<td>5. $S \rightarrow USU$</td>
</tr>
<tr>
<td>6. $B \rightarrow aa$</td>
</tr>
<tr>
<td>7. $B \rightarrow bb$</td>
</tr>
<tr>
<td>8. $U \rightarrow ab$</td>
</tr>
<tr>
<td>9. $U \rightarrow ba$</td>
</tr>
</tbody>
</table>
Defining non-regular languages

Example

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

Example

1. $S \rightarrow aSb$
2. $S \rightarrow bSb$
3. $S \rightarrow \Lambda$

Example

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


Example

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

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).

\[
\begin{align*}
S & \rightarrow aS \\
S & \rightarrow \Lambda \\
S & \rightarrow X \\
S & \rightarrow Y \\
X & \rightarrow \Lambda \\
Y & \rightarrow aY \\
Y & \rightarrow bY \\
Y & \rightarrow a \\
Y & \rightarrow b \\
S & \rightarrow aS | \Lambda \\
S & \rightarrow X | Y \\
X & \rightarrow \Lambda \\
Y & \rightarrow aY | bY | a | b
\end{align*}
\]
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

\[ S \rightarrow aS \mid Sa \mid a \]