Languages, their Generators and Parsers

Language is the set of **finite length strings** “over” a finite alphabet.

CFG is a **Context Free Grammar** — interesting languages are infinite, so we can’t just write down all the possible strings of the language. We need a way to **generate** a language, and a away to check the correctness (syntax) of some string over the alphabet. **CFG**s provide both of these tools.

Languages, their Generators and Parsers

Language is the set of **finite length strings** “over” a finite alphabet.
“over” means *generated by*, alphabet is no longer letters as in REs, now “alphabet” means *“words” or more generally sequences of symbols*.
The set of strings that constitute a language may be **unbounded** (“infinite”), but the strings themselves are finite.

CFG is a **Context Free Grammar** — interesting languages are infinite, so we can’t just write down all the possible strings of the language. We need a way to **generate** a language, and a away to check the correctness (syntax) of some string over the alphabet. **CFG**s provide both of these tools.

Context Sensitive Grammars do the same, but are more complicated to analyse (because they define both the **syntax** and the **semantics** of a language). Not used in the nuts and bolts of compilers.

Context Free Grammars

A compact representation of a language defined with four terms:

1. A finite set of **non-terminal** symbols N
2. A finite **alphabet of terminals** Σ, an “end-of-file” marker \$, and the empty string symbol λ
3. A finite set of **productions** (rewriting rules) P
4. A **start or goal symbol** S that begins the process of *derivations*

$$
\begin{array}{rcl}
S & \rightarrow & A \; M \; \$ \\
A & \rightarrow & B \; C \\
 & \mid & C \; M \\
B & \rightarrow & b \; g \; h \\
C & \rightarrow & s \; t \\
 & \mid & \lambda \\
M & \rightarrow & m \\
 & \mid & n \\
 & \mid & p
\end{array}
$$

Context Free Grammars

A compact representation of a language defined with four terms:

...but not the representation of **all** languages!

1. A finite set of **non-terminal** symbols N
2. A finite **alphabet of terminals** Σ, an "end-of-file" marker \$, and the empty string symbol λ
   (the \$ marker enforces **"finite length strings"** of the language)
3. A finite set of **productions** (rewriting rules) P
4. A **start or goal symbol** S that begins the process of *derivations*

$$P \left\{ \begin{array}{lcl} S & \rightarrow & A\ M\ \$ \\ A & \rightarrow & B\ C \\ & | & C\ M \\ B & \rightarrow & b\ g\ h \\ C & \rightarrow & s\ t \\ & | & \lambda \\ M & \rightarrow & m \\ & | & n \\ & | & p \end{array} \right.$$

$$N = \{S, A, B, C, M\} \qquad \Sigma = \{b, g, h, s, t, m, n, p\}$$

Context Free Grammars

A compact representation of a language defined with four terms:

...but not the representation of **all** languages!

1. A finite set of **non-terminal** symbols N
2. A finite **alphabet of terminals** Σ, an "end-of-file" marker \$, and the empty string symbol λ
   (the \$ marker enforces **"finite length strings"** of the language)
3. A finite set of **productions** (rewriting rules) P
4. A **start or goal symbol** S that begins the process of *derivations*

$$\left.\begin{array}{lcl} S & \rightarrow & A\ M\ \$ \\ A & \rightarrow & B\ C \\ & | & C\ M \\ B & \rightarrow & b\ g\ h \\ C & \rightarrow & s\ t \\ & | & \lambda \\ M & \rightarrow & m \\ & | & n \\ & | & p \end{array}\right\} P$$

Combining these four elements notationally, a grammar is $G(N, \Sigma, P, S)$ and its vocabulary $V = N \cup \Sigma$. Implicitly, $N \cap \Sigma = \emptyset$.

NOTATION, Notation, notation

Recall: N is a set of grammar non-terminals, Σ is a set of grammar terminals, P is the set of grammar production rules

- Augmented Σ: $\Sigma_\$ = \Sigma \cup \{\$\} = \Sigma \cup \{\$\}$
  The set of language terminals along with end-of-input marker. $\Sigma_\$$ will come in handy for several critical algorithms. We will also occasionally need Σ_λ.
- Uppercase Latin (X, K, $TAIL$, $EXPR$): symbols in N
- lower case Latin and punctuation (`x`, k, `while`, `def`, `?`, `!`): elements of Σ
- Capital Script Latin ($\mathcal{X}, \mathcal{K}$): sets of symbols from $N \cup \Sigma$
- Greek letters (α, β, γ): $(N \cup \Sigma)^*$ (where * is the Kleene operator of REs)
  So a (possibly empty) **sequence** of symbols from the grammar

Derivations

A CFG is a *recipe for generating strings* of a language.

- A **rewrite** is when a production rule $A \rightarrow \alpha$ replaces A with α; rewriting with the special rule $A \rightarrow \lambda$ deletes A.
- Each rewrite is a step in the **derivation** of some string of the language.
- If we begin at S, the grammar's **start symbol**, the set of all possible (terminal only) derived strings is the **context free language** of the grammar, $L(G)$.

$$
\begin{array}{ccl}
S & \Rightarrow & A\,M\,\$ \\
S & \Rightarrow & C\,M\,M\,\$ \\
S & \Rightarrow & \lambda\,M\,M\,\$ \\
S & \Rightarrow & m\,M\,\$ \\
S & \Rightarrow & m\,p\,\$ \\
 & & m\,p
\end{array}
$$

Derivations

A CFG is a *recipe for generating strings* of a language.

- A **rewrite** is when a production rule $A \rightarrow \alpha$ replaces A with α; rewriting with the special rule $A \rightarrow \lambda$ deletes A.
- Each rewrite is a step in the **derivation** of some string of the language.
- If we begin at S, the grammar's **start symbol**, the set of all possible (terminal only) derived strings is the **context free language** of the grammar, $L(G)$.

$$
\begin{array}{lcl}
S & \Rightarrow & A\,M\,\$ \\
S & \Rightarrow & C\,M\,M\,\$ \\
S & \Rightarrow & \lambda\,M\,M\,\$ \\
S & \Rightarrow & m\,M\,\$ \\
S & \Rightarrow & m\,p\,\$ \\
 & & m\,p
\end{array}
$$

The process of **derivation** can also be used to validate the **syntax** of some specific $\Sigma^*\$$ — by the end of the course this is what you'll remember most : (

…recall Σ^* are all possible strings made of language terminals, the \$ suffix indicates **finite** sequences.

Derivation Notation and Sentential Forms

$\Rightarrow$ means "derives in one derivation step" If $A \to \lambda$, then $\alpha A \beta \Rightarrow \alpha\,\beta$.

...recall lower Greek letters are $(N \cup \Sigma)^*$

$\Rightarrow^+$ derives in one or more derivation step(s) $A \Rightarrow^+ m$.

$\Rightarrow^*$ derives in zero or more derivation steps $S \Rightarrow^* A\,M\,\$$ $S \Rightarrow^* m\,p$.

...the latter two from the `slide grammar` .

Derivation Notation and Sentential Forms

$\Rightarrow$ means "derives in one derivation step" If $A \rightarrow \lambda$, then $\alpha A \beta \Rightarrow \alpha\beta$.

...recall lower Greek letters are $(N \cup \Sigma)^*$

$\Rightarrow^+$ derives in one or more derivation step(s) $A \Rightarrow^+ m$.

$\Rightarrow^*$ derives in zero or more derivation steps $S \Rightarrow^* A\ M\ \$$ $S \Rightarrow^* m\ p$.

...the latter two from the `slide grammar` .

β is a **sentential form** of a CFG if $S \Rightarrow^* \beta$,[1]
$SF(G)$ is the set of all sentential forms of grammar G (typically a CFG), and now we can formalize the **language of a grammar**:

$$L(G) = \{w \in \Sigma^* | S \Rightarrow^+ w\} = SF(G) \cap \Sigma^*$$

[1]...more accurately: $S \Rightarrow^* \beta\ \$$ or $S \Rightarrow^+ \beta$, but all texts seem to play fast and loose with these notations :(I try to be as consistent as possible.

Sentential Forms and $L(G)$ by Example (take two)

A CFG is a *recipe for generating strings* of a language, it can also be used to *verify the syntax* of a finite string from Σ^*.

- β is a **sentential form** of a CFG if $S \Rightarrow^* \beta$,
  $SF(G)$ is the set of all sentential forms of grammar G,
  ...so all the symbol sequences to the left of $\Rightarrow^*$s are **sentential forms**.
- Finite strings from Σ^* that **are also** sentential forms, they also define the language L of a grammar G,

$$L(G) = SF(G) \cap \Sigma^*$$

so $m\ p$ is in the language of this grammar.

$$\begin{array}{lcl}
S & \Rightarrow & A\ M\ \$ \\
S & \Rightarrow & C\ M\ M\ \$ \\
S & \Rightarrow & \lambda\ M\ M\ \$ \\
S & \Rightarrow & m\ M\ \$ \\
S & \Rightarrow & m\ p\ \$ \\
 & & m\ p
\end{array}$$

Sentential Forms and $L(G)$ by Example (take two)

A CFG is a *recipe for generating strings* of a language, it can also be used to *verify the syntax* of a finite string from Σ^*.

- β is a **sentential form** of a CFG if $S \Rightarrow^* \beta$,
  $SF(G)$ is the set of all sentential forms of grammar G,
  ... so all the symbol sequences to the left of $\Rightarrow^*$s are **sentential forms**.
- Finite strings from Σ^* that **are also** sentential forms, they also define the language L of a grammar G,

$$L(G) = SF(G) \cap \Sigma^*$$

so $m\ p$ is in the language of this grammar.

$$\begin{array}{lcl}
S & \Rightarrow & A\ M\ \$ \\
S & \Rightarrow & C\ M\ M\ \$ \\
S & \Rightarrow & \lambda\ M\ M\ \$ \\
S & \Rightarrow & m\ M\ \$ \\
S & \Rightarrow & m\ p\ \$ \\
 & & m\ p
\end{array}$$

Notice that we we treat $\lambda\ M\ M\ \$ \equiv M\ M$ because λ **and** $\$$ **are not in** Σ, they are merely notational placeholders (but soon we'll see them become important algorithmic entities!)

Left or Right Most Derivations

What happens when you want to perform the next rewrite on

$$A \Rightarrow C\, M$$

Which is rewritten first, C or M?

Left or Right Most Derivations

What happens when you want to perform the next rewrite on

$$A \Rightarrow C\,M$$

Which is rewritten first, C or M?

Neither C first or M first is wrong, as long as we **always** work **left to right** ("leftmost") or right to left ("rightmost").

This is more than just a convention, the choice dictates the complexity of the language you can compile.

Which is an incredibly huge takeaway for this course!

Left or Right Most Derivations

What happens when you want to perform the next rewrite on

$$A \Rightarrow C\,M$$

Which is rewritten first, C or M?

Left-Most Derivations

- Denoted with $\Rightarrow_{lm}$, $\Rightarrow^*_{lm}$, $\Rightarrow^+_{lm}$
- Creates **left sentential forms** $\subseteq SF(G)$
- The type of “validation derivation” performed by **top down parsers**, aka “recursive descent parsing.”

Right-Most Derivations

- Denoted with $\Rightarrow_{rm}$, $\Rightarrow^*_{rm}$, $\Rightarrow^+_{rm}$
- Creates **right sentential forms** $\subseteq SF(G)$
- Performed by **bottom up parsers**; colloquially referred to as “canonical parsing”.
- Many of our favorite programming languages require right most derivations.

Left or Right Most Derivations

#	Rules
1	$S \rightarrow E\,\$$
2	$E \rightarrow PREFIX\ (\ E\ )$
3	$E \rightarrow v\ TAIL$
4	$PREFIX \rightarrow f$
5	$PREFIX \rightarrow \lambda$
6	$TAIL \rightarrow +\,E$
7	$TAIL \rightarrow \lambda$

Source string:

```
f ( v + v )
```

Left most derivation

$$
\begin{array}{llll}
(1) & S & \Rightarrow_{lm} & E\ \$ \\
(2) & S & \Rightarrow_{lm} & PREFIX\ (\ E\ )\ \$ \\
(4) & S & \Rightarrow_{lm} & f\ (\ E\ )\ \$ \\
(3) & S & \Rightarrow_{lm} & f\ (\ v\ TAIL\ )\ \$ \\
(6) & S & \Rightarrow_{lm} & f\ (\ v\ +\ E\ )\ \$ \\
(3) & S & \Rightarrow_{lm} & f\ (\ v\ +\ v\ TAIL\ )\ \$ \\
(7) & S & \Rightarrow_{lm} & f\ (\ v\ +\ v\ )\ \$
\end{array}
$$

Right most derivation

$$
\begin{array}{llll}
(1) & S & \Rightarrow_{rm} & E\ \$ \\
(2) & S & \Rightarrow_{rm} & PREFIX\ (\ E\ )\ \$ \\
(3) & S & \Rightarrow_{rm} & PREFIX\ (\ v\ TAIL\ )\ \$ \\
(6) & S & \Rightarrow_{rm} & PREFIX\ (\ v\ +\ E\ )\ \$ \\
(3) & S & \Rightarrow_{rm} & PREFIX\ (\ v\ +\ v\ TAIL\ )\ \$ \\
(7) & S & \Rightarrow_{rm} & PREFIX\ (\ v\ +\ v\ )\ \$ \\
(4) & S & \Rightarrow_{rm} & f\ (\ v\ +\ v\ )\ \$
\end{array}
$$

("Raw") Parse Trees

- Root with S, the grammar **start symbol** or **goal**.
- Interior nodes $\in N$, always **non-terminals** of the language.
- An interior node and its children is a **derivation rewrite**, the root is the LHS of a production rule, the children are the RHS.
- When a derivation is complete, all leaves $\in \Sigma + \{\$, \lambda\}$ (to say "the language terminals" is not 100% correct).
- Sentential forms are derivable from S, so all sentential forms have a parse tree.