Context Free Grammars

Regular languages can be specified in terms of finite automata that accept or reject strings, equivalently, in terms of regular expressions, which strings are to match.

This section introduces a new, *generative* means of specifying sets of strings.

Context-free grammars (CFG): A way of generating words

Ingredients of a CFG:

$$(\{\text{ variables }\}, \{\text{ terminals }\}, \{\text{ productions (or rules) }\}, \text{start symbol})$$

The start symbol is a special variable.

A CFG generates strings over the alphabet $\Sigma = \{\text{ terminals }\}$.

Example. $G = (\{A, B\}, \{0, 1\}, \mathcal{R}, A)$ where $\mathcal{R}$ consists of three rules:

$$\begin{cases} A & \rightarrow & 0\,A\,1 \\ A & \rightarrow & B \\ B & \rightarrow & \epsilon \end{cases}$$

How to generate strings using a CFG

1. Set w to be the start symbol.
2. Choose an occurrence of a variable X in w if any, otherwise STOP.
3. Pick a production whose lhs is X, replace the chosen occurrence of X in w by the rhs.
4. GOTO 2.

Example $G = (\{A, B\}, \{0, 1\}, \{A \rightarrow 0\,A\,1 \mid B, \quad B \rightarrow \epsilon\}, A)$ generates $\{0^i\,1^i : i \geq 0\}$.

$$\begin{aligned} A &\Rightarrow 0\,A\,1 \\ &\Rightarrow 0\,0\,A\,1\,1 \\ &\Rightarrow 0\,0\,B\,1\,1 \\ &\Rightarrow 0\,0\,\epsilon\,1\,1 = 0^2\,1^2 \end{aligned}$$

Such sequences are called *derivations*.

Definition: Context-free Grammar

A *context-free grammar* is a 4-tuple $G = (V, \Sigma, \mathcal{R}, S)$ where

(i) V is a finite set of *variables* (or *non-terminals*)

(ii) Σ (the alphabet) is a finite set of *terminals*

(iii) $\mathcal{R}$ is a finite set of *productions*. A *production* (or *rule*) is an element of $V \times (V \cup \Sigma)^*$, written $A \longrightarrow w$.

(iv) $S \in V$ is the *start symbol*.

We define a binary relation $\Rightarrow$ over $(\{ V \cup \Sigma \})^*$ by: for each $u, v \in (\{ V \cup \Sigma \})^*$, for each $A \longrightarrow w$ in $\mathcal{R}$

$$u\, A\, v \Rightarrow u\, w\, v$$

We write $\Rightarrow^*$ for the reflexive and transitive closure of $\Rightarrow$.

The *language of the grammar*, written $L(G)$, is $\{ w \in \Sigma^* : S \Rightarrow^* w \}$.

Examples

Palindromes over $\{ a, b, c \}$: generated by $(\{ S, T \}, \{ a, b, c \}, \mathcal{R}, S)$ where $\mathcal{R}$ consists of eight rules:

$$\begin{array}{lcl} S & \to & a\,T\,a \mid b\,T\,b \mid c\,T\,c \mid T \\ T & \to & S \mid a \mid b \mid c \mid \epsilon \end{array}$$

Note: Use $|$ to save writing. The above can be simplified: one variable suffices.

Well-balanced parentheses: generated by $(\{ S \}, \{ \textbf{(}, \textbf{)} \}, \mathcal{R}, S)$ where $\mathcal{R}$ consists of

$$S \to \textbf{(}\,S\,\textbf{)} \mid S\,S \mid \epsilon$$

E.g. **(( ) (( ))) ( )**

Exercise. Prove that the grammar generates precisely all well-balanced parentheses.

Regular languages are context-free

A language is *context-free* just in case it is generated by some CFG.

A CFG is *right-linear* if every rule is either of the form $R \longrightarrow wT$ or of the form $R \longrightarrow w$ where w ranges over strings of terminals, and R and T over variables.

Theorem. A language is regular iff it is generated by a right-linear CFG.

Proof idea. Say a CFG is *strongly right-linear* if each rule has one of the following forms: $R \longrightarrow aT$, $R \longrightarrow T$ or $R \longrightarrow \epsilon$ where a ranges over terminals, and R and T over variables.

Fact. Each right-linear CFG is equivalent to a strongly right-linear one.

For each rule $R \longrightarrow wT$ with $w = a_0 a_1 \cdots a_n$ we add n fresh variables, say $V_1, \ldots V_n$, and replace the rule $R \longrightarrow wT$ by the rules

$$R \longrightarrow a_0 V_1, \quad V_1 \longrightarrow a_1 V_2, \quad \ldots \quad V_n \longrightarrow a_n T$$

"$\Rightarrow$": We map DFAs $M = (Q, \Sigma, \delta, q_0, F)$ to strongly right-linear CFGs $G_M = (Q, \Sigma, \mathcal{R}, q_0)$ where

$$\begin{aligned} q \longrightarrow a\, q' \in \mathcal{R} \quad &\iff \quad \delta(q, a) = q' \\ q \longrightarrow \epsilon \in \mathcal{R} \quad &\iff \quad q \in F \end{aligned}$$

"$\Leftarrow$": We map strongly right-linear CFGs $G = (V, \Sigma, \mathcal{R}, S)$ to NFAs $N_G = (V, \Sigma, \delta, S, F)$ where

$$\begin{aligned} R \xrightarrow{a} R' \text{ (i.e. } R' \in \delta(R, a)) \quad &\iff \quad R \rightarrow a\, R' \ \in \ \mathcal{R} \\ R \xrightarrow{\epsilon} R' \text{ (i.e. } R' \in \delta(R, \epsilon)) \quad &\iff \quad R \rightarrow R' \ \in \ \mathcal{R} \\ R \in F \quad &\iff \quad R \rightarrow \epsilon \ \in \ \mathcal{R} \end{aligned}$$

$\square$

Easy exercise. Give a right-linear CFG that generates $0^* \, 1^* \, 0^*$.

Parse trees

Parse trees: Each derivation determines a *parse tree*.

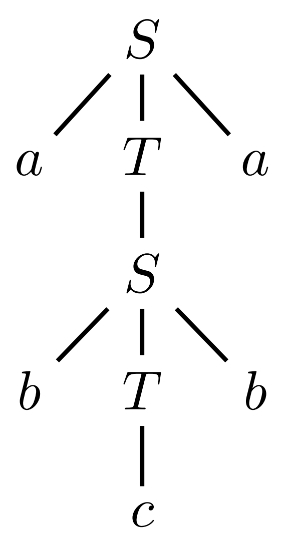
Parse trees are *ordered* trees: the children at each node are ordered.

The parse tree of a derivation abstracts away from the order in which variables are replaced in the sequence.

derivation

$$\begin{aligned} S &\Rightarrow a\,T\,a \\ &\Rightarrow a\,S\,a \\ &\Rightarrow a\,b\,T\,b\,a \\ &\Rightarrow a\,b\,c\,b\,a \end{aligned}$$

parse tree



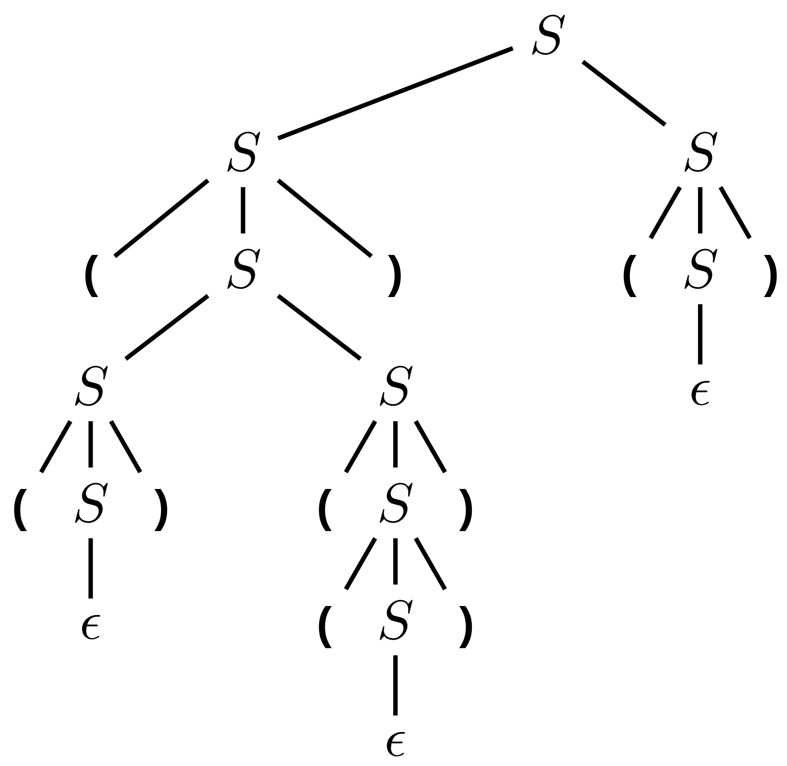
Example Parentheses

$$S \rightarrow \mathbf{(}\,S\,\mathbf{)} \;\mid\; S\,S \;\mid\; \epsilon$$

derivation

$$\begin{array}{rcc}
S & \Rightarrow & \underline{S}\,S \\
 & \Rightarrow & \overline{\mathbf{(}\underline{S}\mathbf{)}}\,S \\
 & \Rightarrow & \mathbf{(}\overline{\underline{S}\,\underline{S}}\mathbf{)}\,S \\
 & \Rightarrow^2 & \mathbf{(}\overline{\mathbf{(}\underline{S}\mathbf{)}}\,\overline{\mathbf{(}\underline{S}\mathbf{)}}\mathbf{)}\,\underline{S} \\
 & \Rightarrow^3 & \mathbf{((}\overline{\epsilon}\mathbf{)(}\overline{\mathbf{(}\underline{S}\mathbf{)}}\mathbf{))}\overline{\mathbf{(}\underline{S}\mathbf{)}} \\
 & \Rightarrow^2 & \mathbf{((}\epsilon\mathbf{)((}\overline{\epsilon}\mathbf{)))(}\overline{\epsilon}\mathbf{)} \\
 & = & \mathbf{(())((()))()}
\end{array}$$

parse tree

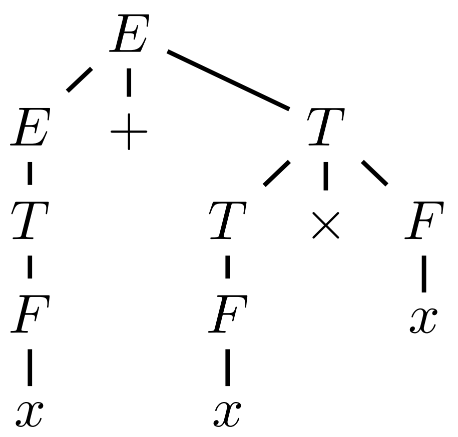


Example: Arithmetic expressions

$(\{E,T,F\},\{+,\times,(,),x\},\mathcal{R},E)$ where $\mathcal{R}$ consists of 6 rules:

$E \rightarrow E+T \mid T \qquad T \rightarrow T \times F \mid F \qquad F \rightarrow (E) \mid x$

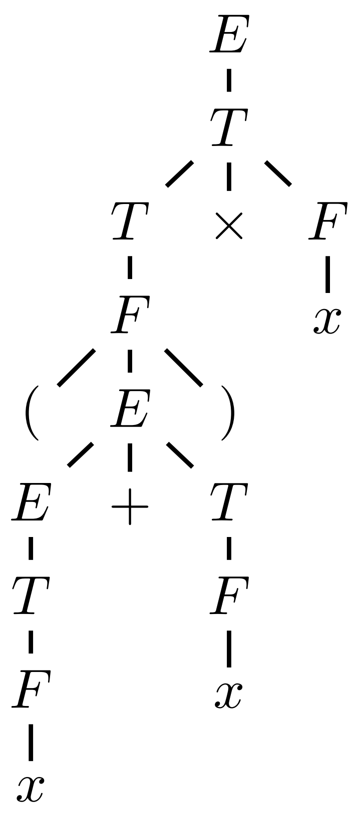
$$\begin{aligned}
E &\Rightarrow E+\underline{T} \\
&\Rightarrow \underline{E}+T\times F \\
&\Rightarrow T+\underline{T}\times F \\
&\Rightarrow \underline{T}+F\times F \\
&\Rightarrow F+F\times \underline{F} \\
&\Rightarrow F+F\times x \\
&\Rightarrow^{*} x+x\times x
\end{aligned}$$



Example: Arithmetic expressions

$$E \to E + T \mid T \qquad T \to T \times F \mid F \qquad F \to (\, E\,) \mid x$$

$$\begin{aligned}
E &\Rightarrow \underline{T} \\
&\Rightarrow \underline{T} \times F \\
&\Rightarrow \underline{F} \times F \\
&\Rightarrow (\, \underline{E}\,) \times F \\
&\Rightarrow (\, \underline{E} + T\,) \times F \\
&\Rightarrow (\, \underline{T} + T\,) \times F \\
&\Rightarrow (\, \underline{F} + T\,) \times F \\
&\Rightarrow (\, x + \underline{T}\,) \times \underline{F} \\
&\Rightarrow^* (\, x + x\,) \times x
\end{aligned}$$



Example: A small English language

$$
\begin{array}{rcl}
\langle\text{SENTENCE}\rangle & \longrightarrow & \langle\text{NOUN-PHRASE}\rangle\ \langle\text{VERB-PHRASE}\rangle \\
\langle\text{NOUN-PHRASE}\rangle & \longrightarrow & \langle\text{CMPLX-NOUN}\rangle \mid \langle\text{CMPLX-NOUN}\rangle\ \langle\text{PREP-PHRASE}\rangle \\
\langle\text{VERB-PHRASE}\rangle & \longrightarrow & \langle\text{CMPLX-VERB}\rangle \mid \langle\text{CMPLX-VERB}\rangle\ \langle\text{PREP-PHRASE}\rangle \\
\langle\text{PREP-PHRASE}\rangle & \longrightarrow & \langle\text{PREP}\rangle\ \langle\text{CMPLX-NOUN}\rangle \\
\langle\text{CMPLX-NOUN}\rangle & \longrightarrow & \langle\text{ARTICLE}\rangle\ \langle\text{NOUN}\rangle \\
\langle\text{CMPLX-VERB}\rangle & \longrightarrow & \langle\text{VERB}\rangle \mid \langle\text{VERB}\rangle\ \langle\text{NOUN-PHRASE}\rangle \\
\langle\text{ARTICLE}\rangle & \longrightarrow & \text{a} \mid \text{the} \\
\langle\text{NOUN}\rangle & \longrightarrow & \text{boy} \mid \text{girl} \mid \text{flower} \\
\langle\text{VERB}\rangle & \longrightarrow & \text{touches} \mid \text{like} \mid \text{see} \\
\langle\text{PREP}\rangle & \longrightarrow & \text{with}
\end{array}
$$

10 variables, 9 terminals and 18 rules.

Example: A small English language

$$
\begin{array}{lcl}
\langle\text{SENTENCE}\rangle & \Rightarrow & \langle\text{NOUN-PHRASE}\rangle\ \langle\text{VERB-PHRASE}\rangle \\
 & \Rightarrow & \langle\text{CMPLX-NOUN}\rangle\ \langle\text{PREP-PHRASE}\rangle\ \langle\text{VERB-PHRASE}\rangle \\
 & \Rightarrow & \langle\text{ARTICLE}\rangle\ \langle\text{NOUN}\rangle\ \langle\text{PREP-PHRASE}\rangle\ \langle\text{VERB-PHRASE}\rangle \\
 & \Rightarrow & \text{a girl}\ \langle\text{PREP}\rangle\ \langle\text{CMPLX-NOUN}\rangle\ \langle\text{VERB-PHRASE}\rangle \\
 & \Rightarrow & \text{a girl with}\ \langle\text{CMPLX-NOUN}\rangle\ \langle\text{VERB-PHRASE}\rangle \\
 & \Rightarrow & \text{a girl with}\ \langle\text{ARTICLE}\rangle\ \langle\text{NOUN}\rangle\ \langle\text{VERB-PHRASE}\rangle \\
 & \Rightarrow & \text{a girl with a flower}\ \langle\text{VERB-PHRASE}\rangle \\
 & \Rightarrow & \text{a girl with a flower}\ \langle\text{CMPLX-VERB}\rangle \\
 & \Rightarrow & \text{a girl with a flower}\ \langle\text{VERB}\rangle\ \langle\text{NOUN-PHRASE}\rangle \\
 & \Rightarrow & \text{a girl with a flower likes}\ \langle\text{CMPLX-NOUN}\rangle \\
 & \Rightarrow & \text{a girl with a flower likes}\ \langle\text{ARTICLE}\rangle\ \langle\text{NOUN}\rangle \\
 & \Rightarrow & \text{a girl with a flower likes the boy}
\end{array}
$$

Ambiguity

Say that two derivations are *essentially different* if they determine distinct parse trees. In some CFGs, the *same* string may have essentially different derivations. Call these CFGs ambiguous.

Examples: Ambiguous arithmetic expressions

$$E \;\rightarrow\; E + E \;\mid\; E \times E \;\mid\; x$$

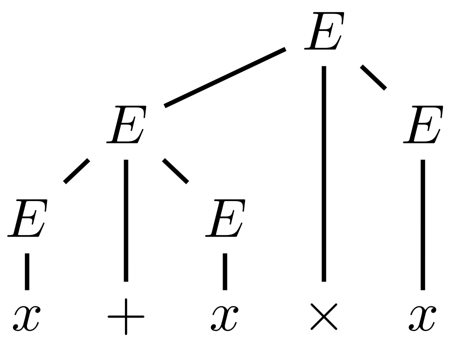
The string $x + x \times x$ has two essentially different derivations:

$$\begin{aligned} E &\Rightarrow \underline{E} \times E \\ &\Rightarrow \underline{E} + E \times E \\ &\Rightarrow x + \underline{E} \times E \\ &\Rightarrow x + x \times \underline{E} \\ &\Rightarrow x + x \times x \end{aligned} \qquad\qquad \begin{aligned} E &\Rightarrow \underline{E} + E \\ &\Rightarrow x + \underline{E} \\ &\Rightarrow x + \underline{E} \times E \\ &\Rightarrow x + x \times \underline{E} \\ &\Rightarrow x + x \times x \end{aligned}$$

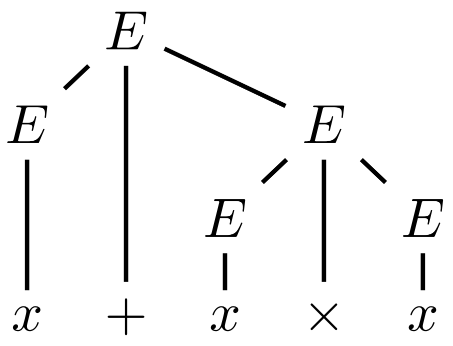
Example: Ambiguous arithmetic expressions

$$E \;\longrightarrow\; E + E \;\mid\; E \times E \;\mid\; x$$

$$\begin{aligned}
E &\Rightarrow \underline{E} \times E \\
&\Rightarrow \underline{E} + E \times E \\
&\Rightarrow x + \underline{E} \times E \\
&\Rightarrow x + x \times \underline{E} \\
&\Rightarrow x + x \times x
\end{aligned}$$



$$\begin{aligned}
E &\Rightarrow \underline{E} + E \\
&\Rightarrow x + \underline{E} \\
&\Rightarrow x + \underline{E} \times E \\
&\Rightarrow x + x \times \underline{E} \\
&\Rightarrow x + x \times x
\end{aligned}$$



Leftmost derivations

A *leftmost derivation* is one in which at every step, the leftmost occurring variable is the one chosen for replacement.

Example. The two derivations for $x + x \times x$

Definition. A CFG G is *ambiguous* just in case there is some word in $L(G)$ which has two (or more) different leftmost derivations.

Note: Each parse tree of a string identifies a leftmost derivation of it. There is a 1-1 correspondence between parse trees and leftmost derivations.

Exercise. Prove that the 6-rule arithmetic expressions on page 8 is unambiguous.

In general the questions of whether a given CFG is ambiguous, or whether two CFGs are equivalent, are very difficult to answer. (In fact these are *undecidable* decision problems.)

Inherently ambiguous languages

Sometimes the language generated by an ambiguous grammar has an equivalent unambiguous grammar.

Some languages can only be generated by ambiguous grammars. They are called *inherently ambiguous*.

Exercise [*Hard*]. Prove that $\{\, 0^i\, 1^j\, 2^k \,:\, i = j \lor j = k \,\}$ is inherently ambiguous.

Chomsky Normal Forms

A CFG is in *Chomsky normal form* is every rule has one of the forms:

$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow a \end{aligned}$$

where a is any terminal, and A, B and C are any variables, except that B and C may not be the start variable. In addition we permit $S \rightarrow \epsilon$ where S is the start variable.

Theorem. Any CFL can be generated by a CFG in Chomsky normal form.

Complementation and context-free languages

Context-free languages are not closed under complementation.

Proof Idea The set $A = \{\, w\, w : w \in \{0,1\}\,\}$ is not context-free, but its complement $B = \{0,1\}^* \setminus A$ is context-free, as it is generated by the grammar $(\{\,S, A, B, C\,\}, \{\,0, 1\,\}, \mathcal{R}, S)$

whith rule set $\mathcal{R} = \{$

$$\begin{array}{lcl} S & \rightarrow & AB \mid BA \mid A \mid B\,, \\ A & \rightarrow & CAC \mid 0\,, \\ B & \rightarrow & CBC \mid 1\,, \\ C & \rightarrow & 0 \mid 1 \;\} \end{array}$$

This grammar generates

all strings of odd length starting with productions $S \rightarrow A$ or $S \rightarrow B$ or

strings of the form $x0yu1v$ or $u1vx0y$ where $x, y, u, v \in \{0,1\}^*$, $|x| = |y|$ and $|u| = |v|$. None of these strings can be of the form $w\,w$.

The Pumping Lemma for context-free languages

Pumping Lemma for CFGs. If L is a context free language, then there is a number p – the *pumping length* – such that if $w \in L$ of length at least p, then w may be divided into five pieces, $w = u\,x\,y\,z\,v$, satisfying:

- for each $i \geq 0$, $u\,x^i\,y\,z^i\,v \in L$
- $|xz| > 0$
- $|x\,y\,z| \leq p$.

This can be used to show that the following languages are not context-free

- $\{\,w\,w : w \in \{0,1\}\,\}$ **Idea**: use $w = 0^p\,1^p\,0^p\,1^p$
- $\{\,0^n 1^n 0^n : n \geq 0\,\}$ **Idea**: use $w = 0^p\,1^p\,0^p$

Concluding remarks for Part One
Outlook on Part Two

Regular languages

- Finite automata, regular expressions and right-linear grammars
- Pumping Lemma and Myhill-Nerode Theorem

Contex-free languages

- Context-free grammars and non-deterministic push down automata
- Pumping Lemma for context-free languages

Decidable and recursively enumerable languages

- Turing machines [and unrestricted grammars]
- Halting Problem and other undecidable problems

Language	**regular**	**context-free**	**semi-decidable/r.e.**
Grammar	right-linear	context-free	unrestricted
rules	$A \to wB$, $A \to w$	$A \to \alpha$	$\alpha \to \beta$
Machine	DFA or NFA	NPDA	Turing machine
memory	finite	finite + one stack	unrestricted
Other Descr$^{\mathrm{n}}$	Regular expression Myhill-Nerode-Thm		λ-calculus μ-recursive functions Hugs, Oberon, . . .
Example	$\{\, w : w$ contains $10 \,\}$ $\{\, 0^n 1^m : n, m \geq 0 \,\}$	$\{\, w\, w^R \,\}$ $\{\, 0^n 1^n : n \geq 0 \,\}$	$\{\, 0^n 1^m 0^{n+m} : n \geq 0 \,\}$ $\{\, Mx : M$ halts on $x \,\}$ HP
Counterexample	$\{\, 0^n 1^n : n \geq 0 \,\}$	$\{\, 0^n 1^n 0^n : n \geq 0 \,\}$ $\{\, w\, w \,\}$	$\{\, M : M$ halts on every $x \,\}$ TP
Application	Tokenizer Model checker	Parser	General computing