

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:

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

$$\left\{ \begin{array}{l} A \rightarrow 0 A 1 \\ A \rightarrow B \\ B \rightarrow \epsilon \end{array} \right.$$

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 $G = (\{A, B\}, \{0, 1\}, \{A \rightarrow 0 A 1 \mid B, B \rightarrow \epsilon\}, A)$ 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 $A \rightarrow w$.
- (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 $A \rightarrow w$ 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:

$$\begin{aligned} S &\rightarrow aT a \mid bT b \mid cT c \mid T \\ T &\rightarrow S \mid a \mid b \mid c \mid \epsilon \end{aligned}$$

Note: Use \mid to save writing. The above can be simplified: one variable suffices.

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

$$S \rightarrow (S) \mid SS \mid \epsilon$$

E.g. $((()())())$

Exercise. Prove that the grammar generates precisely all well-balanced parentheses. [*Hint.* Define the “balance index” of a string, and argue by induction on length.]

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 $R \rightarrow wT$ or of the form $R \rightarrow w$ 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: $R \rightarrow aT$, $R \rightarrow T$ or $R \rightarrow \epsilon$ 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 $R \rightarrow wT$ with $w = a_0a_1 \cdots a_n$ we add n fresh variables, say V_1, \dots, V_n , and replace the rule $R \rightarrow wT$ by the rules

$$R \rightarrow a_0V_1, \quad V_1 \rightarrow a_1V_2, \quad \dots \quad V_n \rightarrow a_nT$$

" \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

$$q \rightarrow a q' \in \mathcal{R} \iff \delta(q, a) = q'$$

$$q \rightarrow \epsilon \in \mathcal{R} \iff q \in F$$

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

$$R \xrightarrow{a} R' \text{ (i.e. } R' \in \delta(R, a)) \iff R \rightarrow a R' \in \mathcal{R}$$

$$R \xrightarrow{\epsilon} R' \text{ (i.e. } R' \in \delta(R, \epsilon)) \iff R \rightarrow R' \in \mathcal{R}$$

$$R \in F \iff R \rightarrow \epsilon \in \mathcal{R}$$

□

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

Parse trees

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

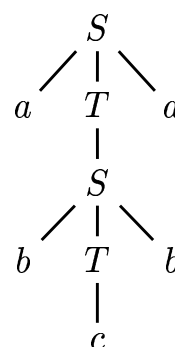
Parse trees are *ordered* trees: the children at each node are ordered.

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



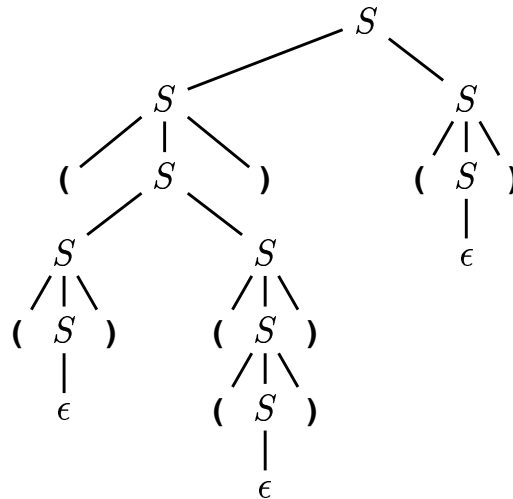
Example Parentheses

$$S \rightarrow (S) \mid SS \mid \epsilon$$

derivation

$$\begin{aligned}
 S &\Rightarrow \overline{SS} \\
 &\Rightarrow \overline{(\overline{S})S} \\
 &\Rightarrow \overline{(\overline{S}\overline{S})S} \\
 &\Rightarrow^2 \overline{((\overline{S})(\overline{S}))\overline{S}} \\
 &\Rightarrow^3 \overline{((\overline{\epsilon})(\overline{(\overline{S}))})\overline{(\overline{S})}} \\
 &\Rightarrow^2 \overline{((\overline{\epsilon})(\overline{(\overline{\epsilon}))})\overline{(\overline{\epsilon})}} \\
 &= \overline{((\overline{()})\overline{()})\overline{()}}
 \end{aligned}$$

parse tree

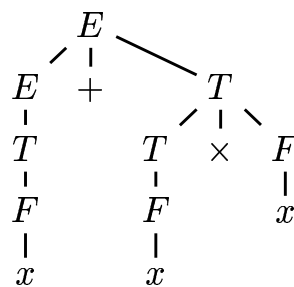


Example: Arithmetic expressions

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

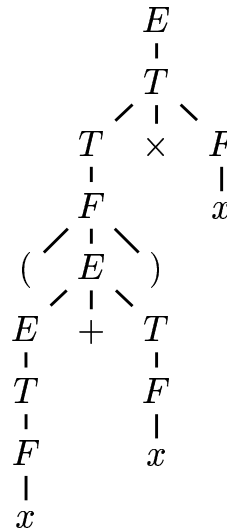
$$E \rightarrow E + T \mid T \quad T \rightarrow T \times F \mid F \quad F \rightarrow (E) \mid x$$

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



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

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



Example: A small English language

- $\langle \text{SENTENCE} \rangle \rightarrow \langle \text{NOUN-PHRASE} \rangle \langle \text{VERB-PHRASE} \rangle$
- $\langle \text{NOUN-PHRASE} \rangle \rightarrow \langle \text{CMPLX-NOUN} \rangle \mid \langle \text{CMPLX-NOUN} \rangle \langle \text{PREP-PHRASE} \rangle$
- $\langle \text{VERB-PHRASE} \rangle \rightarrow \langle \text{CMPLX-VERB} \rangle \mid \langle \text{CMPLX-VERB} \rangle \langle \text{PREP-PHRASE} \rangle$
- $\langle \text{PREP-PHRASE} \rangle \rightarrow \langle \text{PREP} \rangle \langle \text{CMPLX-NOUN} \rangle$
- $\langle \text{CMPLX-NOUN} \rangle \rightarrow \langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle$
- $\langle \text{CMPLX-VERB} \rangle \rightarrow \langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN-PHRASE} \rangle$
- $\langle \text{ARTICLE} \rangle \rightarrow a \mid the$
- $\langle \text{NOUN} \rangle \rightarrow boy \mid girl \mid flower$
- $\langle \text{VERB} \rangle \rightarrow touches \mid like \mid see$
- $\langle \text{PREP} \rangle \rightarrow with$

10 variables, 9 terminals and 18 rules.

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

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

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

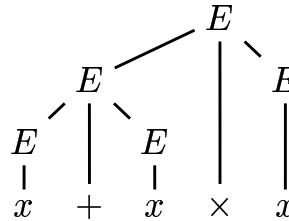
The string $x + x \times x$ has two essentially different derivations:

$ \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} $	$ \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} $
---	---

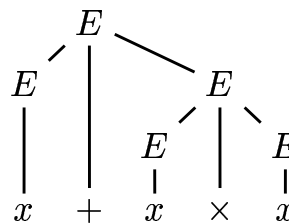
Example: Ambiguous arithmetic expressions

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

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



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



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.)

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 $\{ 0^i 1^j 2^k : i = j \vee j = k \}$ is inherently ambiguous.

Chomsky Normal Forms

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

$$A \rightarrow BC$$

$$A \rightarrow a$$

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 CFG is generated by a CFG in Chomsky normal form.