Example: Context-free Grammar

Context-free grammar is defined by a set of recursive substitution rules:

Example 19 (Grammar G1)

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

Grammar consists of **substitution rules** (*productions*)

- Left-hand side has a *variable*
- Then an arrow
- Right-hand side is a string
- String consists of (i) **variables** and (ii) **terminals**

One variable is the so-called **start variable**

- Convention: start variable is the first rule

Derivation of strings

The basic steps to *derive* a string:

1. Write down the start variable
2. Repeat until all symbols are terminals:
   - Find a variable and replace it with an appropriate string

For example, G1 generates strings such as

- 00#11
- 000#111

- L(G) denotes the language of grammar G
  - In this case, $L(G1) = \{0^n\#1^n \mid n \geq 0\}$.
- L(G) defined by CFG G is a *context-free language*

Parse tree

Derivation can be depicted as a parse tree:

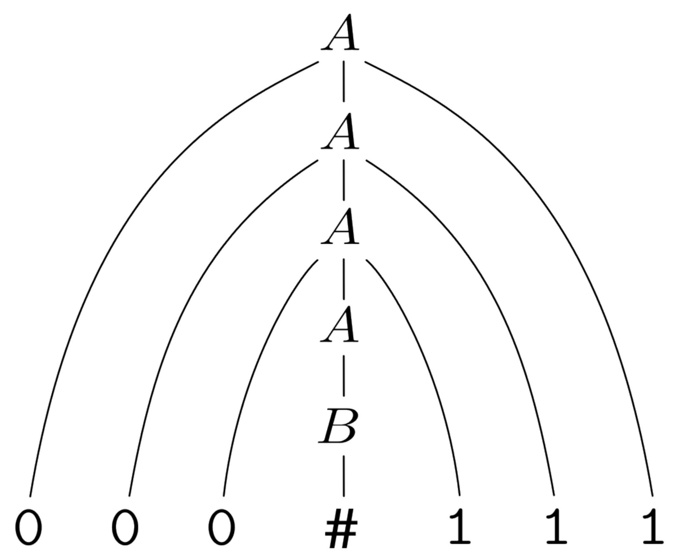


Figure: Figure 2.1 (from book).

Example: G2

Grammar G2:

<SENTENCE> → <NOUN-PHRASE><VERB-PHRASE>
<NOUN-PHRASE> → <CMPLX-NOUN> | <CMPLX-NOUN><PREP-PHRASE>
<VERB-PHRASE> → <CMPLX-VERB> | <CMPLX-VERB><PREP-PHRASE>
<PREP-PHRASE> → <PREP><CMPLX-NOUN>
<CMPLX-NOUN> → <ARTICLE><NOUN>
<CMPLX-VERB> → <VERB> | <VERB><NOUN-PHRASE>
<ARTICLE> → a | the
<NOUN> → boy | girl | flower
<VERB> → touches | likes | sees
<PREP> → with

- 10 variables
- alphabet has 27 symbols (English alphabet + space)
- 18 rules

```
a boy sees
the boy sees a flower
a girl with a flower likes the boy
```

Definition

Definition 18 (CFG (Def. 2.2))

A **context-free grammar** *is a 4-tuple* (V, Σ, R, S), *where*

- V *is a finite set called the variables,*
- Σ *is a finite set, disjoint from* V, *called the terminals,*
- R *is a finite set of rules, with each rule being a variable and a string of variables and terminals, and*
- $S \in V$ *is the start variable*

Terminology:

- uAv **yields** uwv, $uAv \Rightarrow uwv$, if $A \rightarrow w$
- u **derives** v, $u \overset{*}{\Rightarrow} v$, if
  1. $u = v$, or
  2. a sequence $u_1, \ldots, u_k$ exists for $k \geq 0$ s.t.

$$u \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$$

Example 3

Example 20 (Example 2.3)

Consider grammar G3 $= (\{S\}, \{a, b\}, R, S)$ where R is the set of rules

$$S \rightarrow aSb \mid SS \mid \epsilon$$

Questions:

- What kind of strings G3 generates?
- If "SS" from the right-hand side is omitted, what would be the strings?

Example 4

Example 21 (Example 2.4)

Consider grammar G4 $= (V, \Sigma, R, \langle \text{EXPR} \rangle)$

- V is $\{\langle \text{EXPR} \rangle, \langle \text{TERM} \rangle, \langle \text{FACTOR} \rangle\}$
- Σ is $\{a, +, \times, (,)\}$
- The rules R are

$$\begin{aligned}
\langle \text{EXPR} \rangle &\to \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle \\
\langle \text{TERM} \rangle &\to \langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle \\
\langle \text{FACTOR} \rangle &\to (\langle \text{EXPR} \rangle) \mid a
\end{aligned}$$

- E.g., strings $a + a \times a$ and $(a + a) \times a$ can be generated with grammar G4
- The parse trees are shown in Figure 2.5

Example 4 (continued)

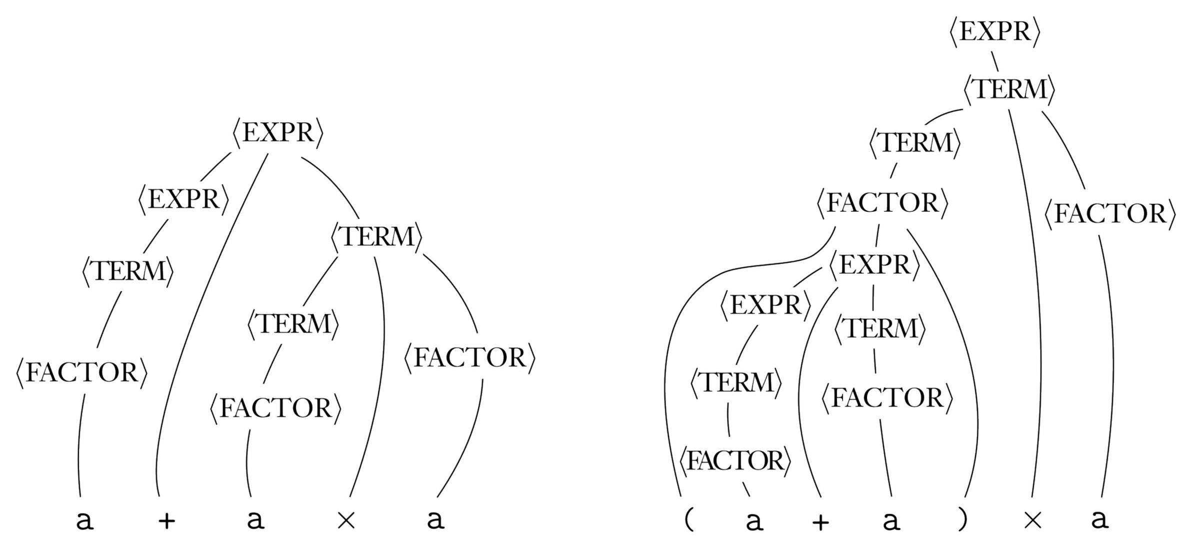


Figure: *G4* illustrated (Figure 2.5 from book).

Note:

- Compilers parse programs, G4 illustrates how arithmetic expressions are parsed
- Note the precedence rules: multiplication before addition
  - Unless overwritten with parentheses
- G4 obeys the standard precedence rules!

Designing CFLs (1)

Many CFLs are the union of simpler CFLs

- Construct CFGs for each piece first (smaller problem)
- Combine them by
  - Combine their rules
  - Add new rule:

$$S \to S_1 \mid S_2 \mid \ldots \mid S_k$$

where the S_i are starting variables of individual cases

Example:

$$\{0^n 1^n \mid n \geq 0\} \cup \{1^n 0^n \mid n \geq 0\}$$

Designing CFLs (2)

CFG for a regular language

- Every regular language has a DFA
  - DFA can be converted to a CFG (next slide)
- Any regular language can be expressed by an RE
  - REs can be converted to a CFG (see later)

CFGs and CFLs are more "powerful" than regular languages

Designing CFLs: From DFA to CFG

An arbitrary DFA can be converted to an equivalent CFG.
Let 5-tuple $(Q, \Sigma, \delta, q_0, F)$ define the DFA, see Def. 1.5.

---

1. Define variable R_i for each state q_i
2. Add rule $R_i \rightarrow aR_j$ if $\delta(q_i, a) = q_j$, $a \in \Sigma$
3. Add rule $R_i \rightarrow \epsilon$ if $q_i \in F$ (accept state)
4. Define R_0 as the start variable (corresponding to q_0)

Why the above procedure yields a CFG that generates the given regular language?

Designing CFLs: From Regexp to CFG

We can construct a CFG for an arbitrary Regexp as follows:

---

1. If RE is a single operand, $w = \epsilon$ or $w \in \Sigma$, add $\langle \mathrm{RE} \rangle \to w$
2. If RE is $\emptyset$, do nothing
3. If RE is a union, $R1 \cup R2$, add $\langle \mathrm{RE} \rangle \to \langle \mathrm{R1} \rangle \mid \langle \mathrm{R2} \rangle$
4. If RE is a concatenation, $R1 \circ R2$, add $\langle \mathrm{RE} \rangle \to \langle \mathrm{R1} \rangle \langle \mathrm{R2} \rangle$
5. If RE is a star, $R1^*$, add $\langle \mathrm{RE} \rangle \to \langle \mathrm{R1} \rangle \langle \mathrm{RE} \rangle \mid \epsilon$

Compare with the definition of the Regexps Def. 1.52

Designing CFLs (3)

CFL with two substrings that are linked

- Prime example $\{0^n1^n \mid n \geq 0\}$
- Infinite memory needed to verify "pairs"?
- No, instead use a rule of form

$$R \rightarrow uRv$$

Recursive structures

- Strings may contain certain recursive structures
- See example 2.4:
  - Anytime symbol a appears, an entire parenthesized expression might appear instead
  - ...recursively
- Use a variable to generate such recursive structures

Regular operations

Theorem 22

The class of context-free languages is closed under the regular operations, union, concatenation, and star.

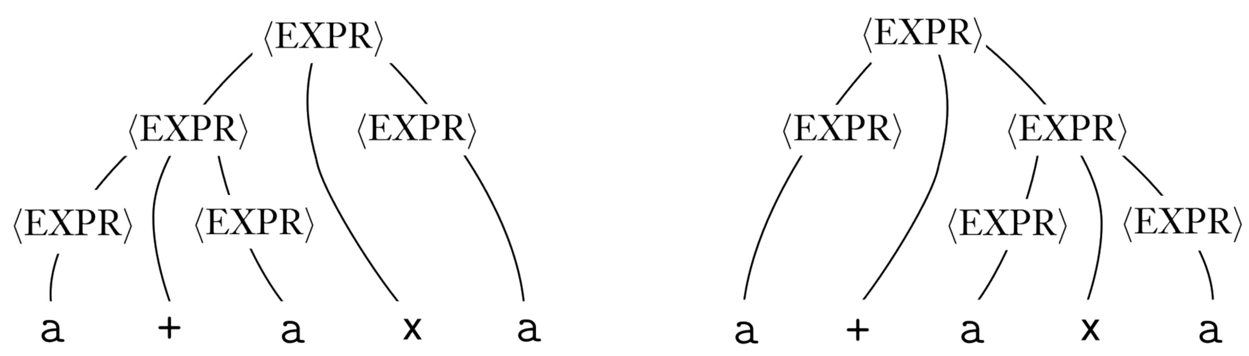
Proof:
Left as an tutorial exercise. $\square$

Example 5

Example 23 (Grammar G5)

$$\langle \text{EXPR} \rangle \to \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle | \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle | (\langle \text{EXPR} \rangle) | a$$

G5 generates the string $a + a \times a$ ambiguously:



- *G5* **does not** capture the usual precedence relations
  - it may group the $+$ before the $\times$ or vice versa
- *G4* generates exactly the same language
  - but every generated string has a unique parse tree!
- *G4* is **unambiguous**, whereas *G5* is **ambiguous**

Example: mathematical expressions

The following CFG generates a more complete set of arithmetic expressions than G5:

$$
\begin{aligned}
\langle \text{EXPR} \rangle &\rightarrow \textit{number} \\
\langle \text{EXPR} \rangle &\rightarrow (\langle \text{EXPR} \rangle) \\
\langle \text{EXPR} \rangle &\rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \\
\langle \text{EXPR} \rangle &\rightarrow \langle \text{EXPR} \rangle - \langle \text{EXPR} \rangle \\
\langle \text{EXPR} \rangle &\rightarrow \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \\
\langle \text{EXPR} \rangle &\rightarrow \langle \text{EXPR} \rangle / \langle \text{EXPR} \rangle
\end{aligned}
$$

Example

Suppose $\Sigma = \{\epsilon, (,), x, +, -, \times, 0, \ldots, 99\}$, and the CFG G has the following production rules:

$$
\begin{aligned}
\langle \text{EXPR} \rangle &\to (\langle \text{EXPR} \rangle) \times (\langle \text{EXPR} \rangle) \\
\langle \text{EXPR} \rangle &\to x + \langle \text{A} \rangle \\
\langle \text{EXPR} \rangle &\to x - \langle \text{A} \rangle \\
\langle \text{A} \rangle &\to 0|1|2|\ldots|99
\end{aligned}
$$

Questions:

- What are the variables of the CFG G
- What mathematical functions correspond to G?

Ambiguity

Definition 19

A derivation of a string w in a grammar G is a **leftmost derivation** *if at every step the leftmost remaining variable is the one replaced.*

Definition 20 (Def. 2.7)

A string w is derived ambiguously in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ambiguous if it generates some string ambiguously.

- For some ambiguous grammars an unambiguous grammar generating the same language exists
- However, some CFLs can be generated only by ambiguous grammars
  - Such languages are called **inherently ambiguous**

Example

Example 24

The strings of form

$$\{0^n 1^n 2^m 3^m \mid n, m \geq 0\},$$

i.e., the corresponding language clearly belongs to the class of context-free languages.

Write down a formal description of the corresponding CFG!

Chomsky Normal Form

Definition 21 (Def. 2.8)

A context-free grammar is in **Chomsky normal form** *if every rule is of the form*

$$A \to BC$$
$$A \to a$$

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 the rule $S \to \epsilon$, where S is the start variable.

Theorem 25 (Theorem 2.9)

Any context-free language can be generated by a context-free grammar in Chomsky normal form.

In other words:

- For any CFL, a CFG in Chomsky normal form exists
  (cf. for any regular language, a DFA exists)

We proof this by constructing a compliant CFG from an arbitrary one!

Proof:

1. **Start state**: Add a new start state S_0 and rule $S_0 \rightarrow S$. Thus the new start state does not appear on the right-hand side.
2. ϵ-**rules**: Remove an ϵ-rule $A \rightarrow \epsilon$, where A is not start state. Then for each rule with A on the right-hand side, we add a new rule with A deleted. This means that
   - If $R \rightarrow uAv$, add $R \rightarrow uv$
   - If $R \rightarrow uAvAw$, add $R \rightarrow uvAw$, $R \rightarrow uAvw$, $R \rightarrow uvw$
   - If $R \rightarrow A$, add $R \rightarrow \epsilon$ (unless it was previously removed)

   Repeat until no ϵ-rules exists (except for the start state)
3. **Unit rules**: Remove a unit rule $A \rightarrow B$. Then, for every rule $B \rightarrow u$, add a new rule $A \rightarrow u$ (unless this was a unit rule previously removed) Repeat until no unit rules left.
4. **Long rules**: Replace all rules of form $A \rightarrow u_1 u_2 \ldots u_k$, where $k \geq 3$ and each u_i is a variable or terminal symbol, by
$$A \rightarrow u_1 A_1, \qquad A_1 \rightarrow u_2 A_2, \quad \ldots \quad A_{k-2} \rightarrow u_{k-1} u_k$$
5. Replace any terminal symbols in $A \rightarrow u_i u_j$ with a new variable U_i and rule $U_i \rightarrow u_i$.

□

Summary

- Regular languages have limitations
  - They “cannot count”
  - *Some* recursive structures however are still impossible . . .
- CFGs are defined by a set of (recursive) rules
  - They generate *context-free languages* (CFLs)
  - The class of regular languages is a subset of CFLs
- Every CFG can be converted to Chomsky normal form

Thanks!