

Context-Free Grammars (CFG)

- There are languages, such as $\{0^n 1^n \mid n \geq 0\}$ that cannot be described (specified) by finite automata or regular expressions.
- Context-free grammars provide a more powerful mechanism for language specification.
- Context-free grammars can describe features that have a recursive structure making them useful beyond finite automata.

22c:131 Limits of Computation

Hantao Zhang

<http://www.cs.uiowa.edu/~hzhang/c131>

The University of Iowa
Department of Computer Science

Example

- CFG G_1 has the following specification rules:

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

- Nonterminals of CFG G_1 are $\{A, B\}$ and A is the start symbol.
- Terminals of CFG G_1 are $\{0, 1, \#\}$.

Formal Definition of a CFG

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

1. V is a finite set of symbols called the *variables* or *nonterminals*.
2. Σ is a finite set of symbols, disjoint from V , called *terminals*.
3. R is a finite set of *rules* (or specification rules) of the form $lhs \rightarrow rhs$, where $lhs \in V$, $rhs \in (V \cup \Sigma)^*$.
4. $S \in V$ is the start variable.

Example derivation tree

The derivation tree of the string 000#111 using CFG G_1 is in Figure 1

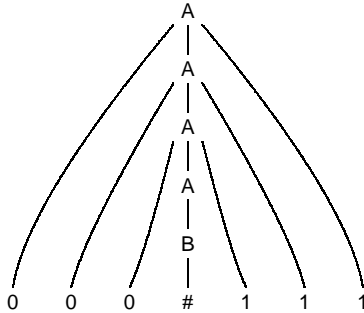


Figure 1: Derivation tree for 000#111

Language specification

A grammar is used for a language specification by generating each string of the language in following manner:

1. Write down the start variable; it is the *lhs* of the first specification rules, unless specified otherwise.
2. Find a variable that is written down and a rule whose *lhs* is that variable. Replace the written down variable with the *rhs* of that rule.
3. Repeat step 2 until no variables remain in the string thus generated

Note: The sequence of substitutions used to obtain a string using a CFG is called a *derivation* and may be represented by a tree called a *derivation tree* or a *parse tree*.

CFG G_2

The CFG G_2 specifies a fragment of English

$\langle \text{SENTENCE} \rangle$	\rightarrow	$\langle \text{NOUN - PHRASE} \rangle \langle \text{VERB - PHRASE} \rangle$
$\langle \text{NOUN - PHRASE} \rangle$	\rightarrow	$\langle \text{CP - NOUN} \rangle \mid \langle \text{CP - NOUN} \rangle \langle \text{PREP - PHRASE} \rangle$
$\langle \text{VERB - PHRASE} \rangle$	\rightarrow	$\langle \text{CP - VERB} \rangle \mid \langle \text{CP - VERB} \rangle \langle \text{PREP - PHRASE} \rangle$
$\langle \text{PREP - PHRASE} \rangle$	\rightarrow	$\langle \text{PREP} \rangle \langle \text{CP - NOUN} \rangle$
$\langle \text{CP - NOUN} \rangle$	\rightarrow	$\langle \text{ARTICLE} \rangle \langle \text{NOUN} \rangle$
$\langle \text{CP - VERB} \rangle$	\rightarrow	$\langle \text{VERB} \rangle \mid \langle \text{VERB} \rangle \langle \text{NOUN - PHRASE} \rangle$
$\langle \text{ARTICLE} \rangle$	\rightarrow	$a \mid the$
$\langle \text{NPUN} \rangle$	\rightarrow	$boy \mid girl \mid flower$
$\langle \text{VERB} \rangle$	\rightarrow	$touches \mid likes \mid sees$
$\langle \text{PREP} \rangle$	\rightarrow	$with$

Note

- All strings of terminals generated in this way constitute the language specified by the grammar
- We write $L(G)$ for the language generated by the grammar G . Thus, $L(G_1) = \{0^n \# 1^n \mid n \geq 0\}$.
- A language generated by a context-free grammar (CFG) is called a Context-Free Language, CFL.

Direct derivation

- If $u, v, w \in (V \cup \Sigma)^*$ (i.e., are strings of variables and terminals) and $A \rightarrow w \in R$ (i.e., is a rule of the grammar) then we say that uAv yields uwv , written $uAv \Rightarrow uwv$
- We may also say that uwv is directly derived from uAv using the rule $A \rightarrow w$

Example derivation with G_2

$\langle SENTENCE \rangle \Rightarrow \langle NOUN - PHRASE \rangle \langle VERB - PHRASE \rangle$
 $\Rightarrow \langle CP - NOUN \rangle \langle VERB - PHRASE \rangle$
 $\Rightarrow \langle ARTICLE \rangle \langle NOUN \rangle \langle VERB - PHRASE \rangle$
 $\Rightarrow a \langle NOUN \rangle \langle VERB - PHRASE \rangle$
 $\Rightarrow a \text{ boy } \langle VERB - PHRASE \rangle$
 $\Rightarrow a \text{ boy } \langle CP - VERB \rangle$
 $\Rightarrow a \text{ boy } \langle VERB \rangle$
 $\Rightarrow a \text{ boy sees}$

Language Specified by G

If $G = (V, \Sigma, R, S)$ is a CFG then the language specified by G (or the language of G) is a CFL

$$L(G) = \{w \in \Sigma^* \mid S \xRightarrow{*} w\}$$

Derivation

- We write $u \xRightarrow{*} v$ if $u = v$ or if a sequence $u_1, u_2, \dots, u_k \in (V \cup \Sigma)^*$ exists, for $k \geq 0$, and $u_1 \Rightarrow u_2 \Rightarrow \dots \Rightarrow u_k \Rightarrow v$
- We may also say that u_1, u_2, \dots, u_k, v is a derivation of v from u_1

Important application

Context-free grammars are used as basis for compiler design and implementation.

- Context-free grammars are used as specification mechanisms for programming languages
- Designers of compilers use such grammars to implement compiler's components, such as scanners, parsers, and code generators
- The implementation of almost any programming language is preceded by a context-free grammar that specifies it

More examples of CFGs

- Consider the grammar $G_3 = (\{S\}, \{a, b\}, \{S \rightarrow aSb \mid SS \mid \epsilon\}, S)$
- $L(G_3)$ contains strings such as $abab, aaabbb, aababb;$

Note: if one think of a and b as (and) then we can see that $L(G_3)$ is the language of all strings of properly nested parentheses

Example Derivation with G_4

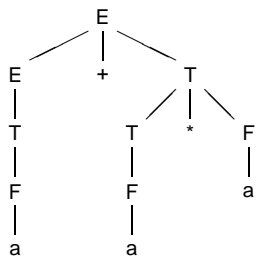


Figure 2: Derivation tree for $a+a*a$

Arithmetic Expressions

- Consider the grammar $G_4 = (\{E, T, F\}, \{a, +, *, (\,)\}, R, E)$ where R is:

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid a$$

- $L(G_4)$ is the language of arithmetic expressions

Design Techniques

- Many CFG are unions of simpler CFGs. Hence the suggestion is to construct smaller, simpler grammars first and then to join them into a larger grammar
- The mechanism of grammar combination consists of putting all their rules together and adding the new rules $S \rightarrow S_1 \mid S_2 \mid \dots \mid S_k$ where the variables $S_i, 1 \leq i \leq k$, are the start variables of the individual grammars and S is a new variable

Designing CFGs

- As with the design of automata, the design of CFGs requires creativity.
- CFGs are even trickier to construct than finite automata because “we are more accustomed to programming a machine than we are to specify programming languages”.

Second Design Technique

- Constructing a CFG for a regular language is easy if one can first construct a DFA for that language
- Conversion procedure:
 1. Make a variable R_i for each state q_i of DFA
 2. Add the rule $R_i \rightarrow aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in the DFA
 3. Add the rule $R_i \rightarrow \epsilon$ if q_i is an accept state of the DFA
 4. If q_0 is the start state of the DFA make R_0 the start variable of the CFG.

Theorem: Every regular language is context-free.

Example Grammar Design

Design a grammar for the language

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

1. Construct the grammar $S_1 \rightarrow 0S_11 \mid \epsilon$ for the language $\{0^n 1^n \mid n \geq 0\}$
2. Construct the grammar $S_2 \rightarrow 1S_20 \mid \epsilon$ for the language $\{1^n 0^n \mid n \geq 0\}$
3. Put them together adding the rule $S \rightarrow S_1 \mid S_2$ thus getting

$$S \rightarrow S_1 \mid S_2$$

$$S_1 \rightarrow 0S_11 \mid \epsilon$$

$$S_2 \rightarrow 1S_20 \mid \epsilon$$

Fourth Design Technique

- In a complex language, strings may contain certain structures that appear recursively.
- Example: in arithmetic expressions any time the symbol a appear, the entire parenthesized expression may appear.
- To achieve this effect one needs to place the variable generating the structure (E in case of G_4) in the location of the rule corresponding to where the structure may recursively appear as in: $E \rightarrow E + T$.

Third Design Technique

- Certain CFLs contain strings with two related substrings as are 0^n and 1^n in $\{0^n 1^n \mid n \geq 0\}$.
- Example of relationship: to recognize such a language a machine would need to remember an unbounded amount of info about one of the substrings
- A CFG that handles this situation uses a rule of the form $R \rightarrow uRv$ which generates strings wherein the portion containing u 's corresponds to the portion containing v 's.

Ambiguity

- If a CFG generates the same string in several different ways, we say that the string is derived *ambiguously* in that grammar.
- If a CFG generates some string ambiguously we say that the grammar is *ambiguous*.
- **Example:** the grammar G_5 , whose rules are:

$$E \rightarrow E + E \mid E * E \mid (E) \mid a$$

generate ambiguously some arithmetic expressions.

Example Application

Consider the CFG $G = (\{S, B\}, \{a, b\}, \{S \rightarrow aSB \mid \epsilon, B \rightarrow bB \mid b\}, S)$.

- The following are derivations with G :
 $S \Rightarrow aSB \rightarrow aaSBB \Rightarrow aaSbBB,$
 $S \Rightarrow aSB \rightarrow aaSBB \Rightarrow aaSBbB,$
 $S \Rightarrow aSB \rightarrow aaSBB \Rightarrow aaSB,$
 $S \Rightarrow aSB \rightarrow aaSBB \Rightarrow aaBB$
which show that derivations in this grammar are quite complex
- When rewriting the string $aaSBB$ we can consider further derivations of each of its symbols in isolation
- Derivations from B are $B \Rightarrow bB \Rightarrow bbB \Rightarrow^* b^{k-1}B \Rightarrow b^k, k \geq 1$
- Therefore $S \Rightarrow aSB \Rightarrow^* aSb^k, k \geq 1$
- Hence, $L(G) = \{a^n b^m \mid n \leq m\}$.

Note

- The grammar G_5 does not capture the usual precedence relations and so groups the + before * and vice versa.
- In contrast, the grammar G_4 generates the same language, but every generated string has a unique derivation tree.
- Hence, G_5 is ambiguous and G_4 is not, i.e., G_4 is *unambiguous*.

Ambiguous expressions

Figure 3 shows two different derivation trees for $a+a*a$

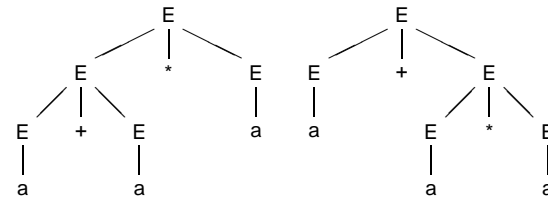


Figure 3: Two derivation trees for $a+a*a$

Fixing rule application order

Leftmost derivation: a derivation of a string w in a grammar G is a *leftmost derivation* if at every step the leftmost nonterminal is replaced.

Rightmost derivation: a derivation of a string w in a grammar G is a *rightmost derivation* if at every step the rightmost nonterminal is replaced.

Note The leftmost and rightmost derivations of a string w are unique, so they are equivalent to the derivation tree.

Note

- When a grammar generates a string ambiguously it means that the string has two different derivation trees.
- Two different derivations however, may produce the same derivation tree because they may differ in the order in which they replace nonterminals not in the rules they use.
- To concentrate on the structure of derivations we need to fix the order of rule application.

Chomsky Normal Form

- It is often convenient to simplify CFG.
- One of the simplest and most useful simplified forms of CFG is called the Chomsky normal form
- Another normal form usually used in algebraic specifications is Greibach normal form.

Inherent Ambiguity

- Some CFL can have both ambiguous and unambiguous grammars.
- Some CFL, however, can be generated only by an ambiguous grammar.
- A CFL that can be generated only by ambiguous grammars is called *inherently ambiguous*.
- **Example of inherently ambiguous language:**

$$\{0^i 1^j 2^k \mid i = j \vee j = k\}$$

Theorem 2.9

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

Proof idea:

- Show that any grammar G can be converted into Chomsky normal form.
- Conversion procedure has several stages where the rules that violate Chomsky normal form conditions are replaced with equivalent rules that satisfy these conditions.
- Order of transformations: (1) add a new start variable, (2) eliminate all ϵ -rules, (3) eliminate unit-rules, (4) convert rules.
- Check that the obtained grammar define the same language as the initial.

Definition

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

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

where a is a terminal, A, B, C are nonterminals, and B, C may not be the start variable.

Note: the rule $S \longrightarrow \epsilon$, where S is the start variable, is not excluded.

Eliminate ϵ -rules

Repeat

1. Eliminate the ϵ rule $A \rightarrow \epsilon$ where A is not the start symbol.
2. For each occurrence of A on the *rhs* of a rule, add a new rule with that occurrence of A deleted
Example: To delete $A \rightarrow \epsilon$, replace $B \rightarrow uAv$ by $B \rightarrow uAv \mid uv$;
replace $R \rightarrow uAvAw$ by $B \rightarrow uAvAw \mid uvAw \mid aAvw \mid uvw$.
3. Replace the rule $B \rightarrow A$, (if it is present) by $B \rightarrow A \mid \epsilon$ unless the rule $B \rightarrow \epsilon$ has not been previously eliminated.

until all ϵ rules are eliminated.

Proof

Add a new start symbol S_0 and the rule $S_0 \rightarrow S$ where S was the original start symbol.

Note: this change guarantees that the start symbol does not occur on the *rhs* of any rule.

Convert all remaining rules

Repeat

1. Replace a rule $A \rightarrow u_1u_2 \dots u_k$, $k \geq 3$, where each u_i , $1 \leq i \leq k$, is a variable or a terminal, by:
 $A \rightarrow u_1A_1, A_1 \rightarrow u_2A_2, \dots, A_{k-2} \rightarrow u_{k-1}u_k$
where A_1, A_2, \dots, A_{k-2} are new variables.
2. If $k \geq 2$ replace any terminal u_i with a new variable U_i and add the rule $U_i \rightarrow u_i$.

until no rules of the form $A \rightarrow u_1u_2 \dots u_k$ with $k \geq 3$ remain.

Remove unit rules

Repeat

1. Remove a unit rule $A \rightarrow B$.
2. For each rule $B \rightarrow u$ that appears, add the rule $A \rightarrow u$, unless it was a unit rule previously removed.

until all unit rules are eliminated.

Note: u is a string of variables and terminals.

Removing ϵ rules

Removing $B \rightarrow \epsilon$:

$$\begin{aligned}S_0 &\rightarrow S \\S &\rightarrow ASA \mid aB \mid a \\A &\rightarrow B \mid S \mid \epsilon \\B &\rightarrow b\end{aligned}$$

Removing $A \rightarrow \epsilon$:

$$\begin{aligned}S_0 &\rightarrow S \\S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \mid S \\A &\rightarrow B \mid S \\B &\rightarrow b\end{aligned}$$

Example CFG conversion

Consider the grammar G_6 whose rules are:

$$\begin{aligned}S &\rightarrow ASA \mid aB \\A &\rightarrow B \mid S \\B &\rightarrow b \mid \epsilon\end{aligned}$$

After first step of transformation we get:

$$\begin{aligned}S_0 &\rightarrow S \\S &\rightarrow ASA \mid aB \\A &\rightarrow B \mid S \\B &\rightarrow b \mid \epsilon\end{aligned}$$

More unit rules

Removing $A \rightarrow B$:

$$\begin{aligned}S_0 &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\A &\rightarrow S \mid b \\B &\rightarrow b\end{aligned}$$

Removing $A \rightarrow S$:

$$\begin{aligned}S_0 &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\A &\rightarrow b \mid ASA \mid aB \mid a \mid SA \mid AS \\B &\rightarrow b\end{aligned}$$

Removing unit rule

Removing $S \rightarrow S$:

$$\begin{aligned}S_0 &\rightarrow S \\S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\A &\rightarrow B \mid S \\B &\rightarrow b\end{aligned}$$

Removing $S_0 \rightarrow S$:

$$\begin{aligned}S_0 &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\S &\rightarrow ASA \mid aB \mid a \mid SA \mid AS \\A &\rightarrow B \mid S \\B &\rightarrow b\end{aligned}$$

Note

- The conversion procedure produces several variables U_i along with several rules $U_i \rightarrow a$.
- Since all these represent the same rule, we may simplify the result using a single variable U and a single rule $U \rightarrow a$.

Converting the remaining rules

$$\begin{aligned} S_0 &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\ S &\rightarrow AA_1 \mid UB \mid a \mid SA \mid AS \\ A &\rightarrow b \mid AA_1 \mid UB \mid a \mid SA \mid AS \\ A_1 &\rightarrow SA \\ U &\rightarrow a \\ B &\rightarrow b \end{aligned}$$

Theorem

If CFG G is in Chomsky Normal Form, then for any string $w \in \Sigma^+$, the length of the derivation of w is $2|w| - 1$.