

Context-Free Grammars and Languages

Teodor Rus

`rus@cs.uiowa.edu`

The University of Iowa, Department of Computer Science

Limitations of finite automata

- There are languages, such as $\{0^n 1^n \mid n \geq 0\}$ that cannot be recognized by finite automata or specified by 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.

Historical notes

- Context-free grammars were first used to study human languages;
- One way of understanding the relationship between syntactic categories (such as noun, verb, preposition, etc) and their respective phrases leads naturally to recursion;
- This is because noun phrases may occur inside the verb phrases and vice versa.

Fact

Context-free grammars can capture important aspects of these relationships.

Important application

Context-free grammars are used as basis for programming language 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 a scanners, parsers, and code generators;
- The implementation of any programming language is preceded by a context-free grammar that specifies it.

Context-free languages

- The collection of languages specified by context-free grammars are called *context-free languages*;
- Context-free languages include regular languages and many others;
- Here we will study the formal concepts of context-free grammar and context-free language.

Notations

- We abbreviate the phrase *context-free grammar* to CFG;
- We abbreviate the phrase *context-free language* to CFL;
- We abbreviate the concept of a *CFG specification rule* to the tuple $lhs \longrightarrow rhs$ where *lhs* stands for *left hand side* and *rhs* stands for *right hand side*.

More on specification rules

- The *lhs* of a specification rule is also called *variable* and is denoted by capital letters;
- The *rhs* of a specification rule is also called a specification pattern and consists of a string of variables and constants;
- The variables that occur in a specification pattern are also called *nonterminal symbols*; the constants that occur in a specification pattern are also called *terminal symbols*.

CFG: Informal

A CFG grammar consists of a collection of specification rules where one variable is designated as *start symbol* or *axiom*.

Example: the CFG G_1 has the following specification rules:

$$A \longrightarrow 0A1$$

$$A \longrightarrow B$$

$$B \longrightarrow \#$$

Observations

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

More terminology

- The specification rules of a CFG are also called *productions* or *substitution rules*;
- Nonterminals used in the specification rules defining a CFG may be strings;
- Terminals in the specification rules defining a CFG are constant strings.

Terminals

- Terminals used in CFG specification rules are analogous to the input alphabet of an automaton;
- Example terminals used in CFG-s are: letters of an alphabet, numbers, special symbols, and strings of such elements;

Notation: strings used to denote terminals in CFG specification rules are quoted.

Language specification

A CFG is used as a language specification mechanism by generating each string of the language in following manner:

1. Write down the start rule $lhs \rightarrow rhs$ where lhs is the axiom and rhs of one of its specification pattern;
Note: usually $lhs \rightarrow rhs$ is the top rule, unless specified otherwise;
2. Find a variable that occurs in the rhs and replace it with the rhs of a rule that has this variable as the lhs ;
3. Repeat step 2 until no variables remain in the string thus generated.

Example string generation

Using CFG G_1 we can generate the string 000#111 as follows:

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

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*.

Example derivation tree

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

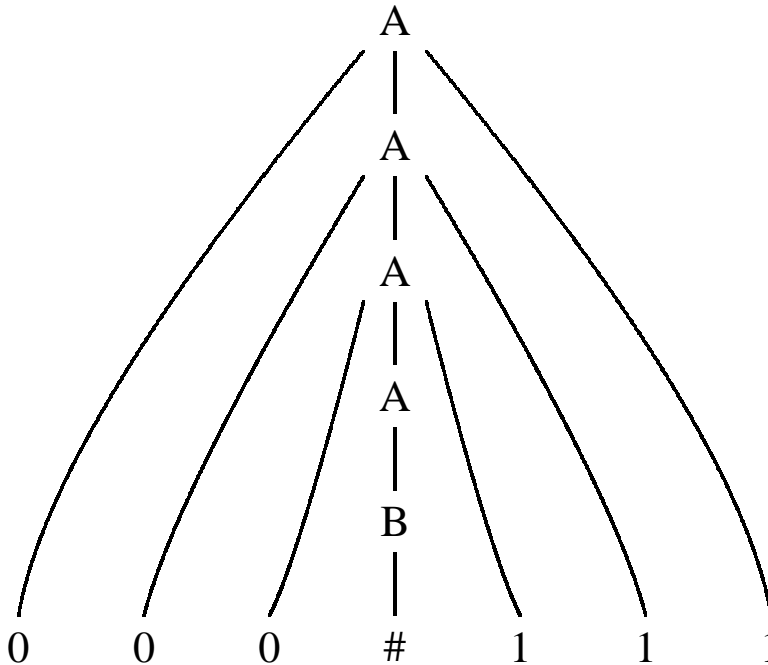


Figure 1: Derivation tree for $000\#111$

Facts

- 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, $L(G_1) = \{0^n \# 1^n \mid n \geq 0\}$;
- The language generated by a context-free grammar is called a Context-Free Language, CFL.

More notations

- To distinguish nonterminal from terminal strings we often enclose nonterminals in angular parentheses, \langle, \rangle , and terminals in quotes “,”;
- If two or more rules have the same *lhs*, as in the example $A \rightarrow 0A1$ and $A \rightarrow B$, we may compact them using the form
$$lhs \rightarrow rhs_1 | rhs_2 | \dots | rhs_n$$
where $|$ is used with the meaning of an “or”.

Example compaction

The rules $A \rightarrow 0A1$ and $A \rightarrow B$ may be written
 $A \rightarrow 0A1|B$.

CFG G_2

The CFG G_2 specifies a fragment of English

$\langle SENTENCE \rangle$	\longrightarrow	$\langle NounPhrase \rangle \langle VerbPhrase \rangle$
$\langle NounPhrase \rangle$	\longrightarrow	$\langle CpNoun \rangle \langle CpNoun \rangle \langle PrepPhrase \rangle$
$\langle VerbPhrase \rangle$	\longrightarrow	$\langle CpVerb \rangle \langle CpVerb \rangle \langle PrepPhrase \rangle$
$\langle PrepPhrase \rangle$	\longrightarrow	$\langle Prep \rangle \langle CpNoun \rangle$
$\langle CpNoun \rangle$	\longrightarrow	$\langle Article \rangle \langle Noun \rangle$
$\langle CpVerb \rangle$	\longrightarrow	$\langle Verb \rangle \langle Verb \rangle \langle NounPhrase \rangle$
$\langle Article \rangle$	\longrightarrow	"a" "the"
$\langle Noun \rangle$	\longrightarrow	"boy" "girl" "flower"
$\langle Verb \rangle$	\longrightarrow	"touches" "likes" "sees"
$\langle Prep \rangle$	\longrightarrow	"with"

Facts

- The CFG G_2 has ten variables (capitalized and in angular brackets) and 9 terminals (written in quotes in the standard English alphabet) plus a space character;
- Also, the CFG G_2 has 18 rules;
- Examples strings that belongs to $L(G_2)$ are:

a boy sees

the boy sees a flower

a girl with a flower likes the boy

Example derivation with G_2

$\langle SENTENCE \rangle \Rightarrow \langle NounPhrase \rangle \langle VerbPhrase \rangle$
 $\Rightarrow \langle CpNoun \rangle \langle VerbPhrase \rangle$
 $\Rightarrow \langle Article \rangle \langle Noun \rangle \langle VerbPhrase \rangle$
 $\Rightarrow a \langle Noun \rangle \langle VerbPhrase \rangle$
 $\Rightarrow a \text{ boy } \langle VerbPhrase \rangle$
 $\Rightarrow a \text{ boy } \langle CpVerb \rangle$
 $\Rightarrow a \text{ boy } \langle Verb \rangle$
 $\Rightarrow a \text{ boy sees}$

Note:

1. It is sufficient to mark one of terminals or non-terminals using special symbols. Here we use angular brackets for non-terminals.
2. Terminals in this derivation are not quoted.

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

Example CFG grammar

$G_1 = (\{A, B\}, \{0, 1, \#\}, R, A)$ where R is:

$$A \longrightarrow 0A1$$

$$A \longrightarrow B$$

$$B \longrightarrow \#$$

Direct derivation

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

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 .

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

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

Observations

- Often we specify a grammar by writing down only its rules;
- We can identify the variables as the symbols that appear only as the *lhs* of the rules;
- Terminals are the remaining strings used in the rules.

More examples of CFGs

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

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

Arithmetic expressions

- Consider the grammar:

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

$$E \longrightarrow E + T | T$$

$$T \longrightarrow T * F | F$$

$$F \longrightarrow (E) | a$$

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

Facts

- The variables and constants in expressions of $L(G_4)$ are represented by the terminal a ;
- Arithmetic operations in $L(G_4)$ are addition, represented by $+$, and multiplication, represented by $*$;
- An examples of a derivation using G_4 is in Figure 2.

Example derivation with G_4

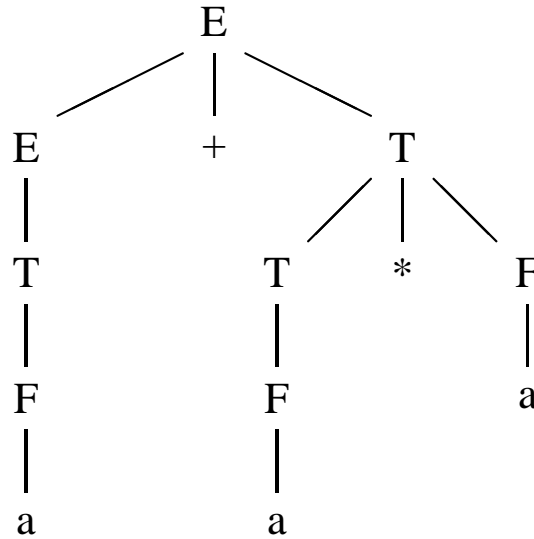


Figure 2: Derivation tree for $a+a*a$

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”.

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: $S \rightarrow S_1|S_2|\dots|S_k$ where the variables $S_i, 1 \leq i \leq k$, are the start variables of the individual grammars and S is a new variable.

Formally: let $G_1 = (V_1, T_1, R_1, S_1), \dots, G_n = (V_n, T_n, R_n, S_n)$ be given.
 $G = (V_1 \cup \dots \cup V_n \cup \{S\}, T_1 \cup \dots \cup T_n, R_1 \cup \dots \cup R_n \cup S \rightarrow S_1 | \dots | S_n, S)$
is the CFG composed of G_1, \dots, G_n .

Example grammar design

Design a CFG for the language:

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

1. Construct the grammar $S_1 \longrightarrow 0S_11 \mid \epsilon$ that generates $\{0^n 1^n \mid n \geq 0\}$;
2. Construct the grammar $S_2 \longrightarrow 1S_20 \mid \epsilon$ that generates $\{1^n 0^n \mid n \geq 0\}$;
3. Put them together adding the rule $S \longrightarrow S_1 \mid S_2$ thus getting:

$$S \longrightarrow S_1 \mid S_2$$

$$S_1 \longrightarrow 0S_11 \mid \epsilon$$

$$S_2 \longrightarrow 1S_20 \mid \epsilon$$

Second design technique

- Constructing a CFG for a regular language is easy if one can first construct a DFA for that language;
- Let $A = (Q, \Sigma, \delta, q_0, F)$ be a DFA that recognizes L . Convert A into $CFG_A = (V, \Sigma, R, R_0)$ by:
 1. Make a variable $R_i \in V$ for each state $q_i \in Q$ of the DFA;
 2. Add the rule $R_i \longrightarrow aR_j$ to R for each transition $\delta(q_i, a) = q_j, a \in \Sigma$, of A ;
 3. Add the rule $R_i \longrightarrow \epsilon$ to R if q_i is an accept state of A .
 4. If q_0 is the start state of the DFA make R_0 the start variable of CFG_A .

Observation

Verify that CFG (constructed by the conversion of a DFA into a CFG) generates the language that the DFA recognizes.

Third design technique

- Certain CFLs contain strings with two related substrings as are 0^n and 1^n in $\{0^n 1^n | n \geq 0\}$;
Note: here the relation is $|0^n| = |1^n|$; other relations may also be observed.
- To recognize such a language a machine would need to remember an unbounded amount of info about one of the substrings.

Fact

A CFG that handles this situation uses rules of the form

$$R \longrightarrow uRv$$

which generates strings wherein the portion containing u 's corresponds to the portion containing v 's.

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.

Fact

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 $E \longrightarrow E + T$ in case of G_4).