Context-Free Grammars and Languages

Limitations of finite automata

- There are languages, such as $\{0^n1^n | n \geq 0\}$ that cannot be described (specified) by NFAs or REs

Limitations of finite automata

- There are languages, such as $\{0^n1^n | n \geq 0\}$ that cannot be described (specified) by NFAs or REs
- Context-free grammars provide a more powerful mechanism for language specification

Limitations of finite automata

- There are languages, such as $\{0^n1^n | n \geq 0\}$ that cannot be described (specified) by NFAs or REs
- 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

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 to natural recursion

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 to natural recursion
- This is because noun phrases may occur inside the verb phrases and vice versa.

Note

Context-free grammars can capture important aspects of these relationships

Important application

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

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

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 a scanners, parsers, and code generators

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 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

- The collection of languages specified by context-free grammars are called context-free languages
- Context-free languages include regular languages and many others

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 grammars and context-free languages

Notations

- Abbreviate the phrase context-free grammar to CFG.

Notations

- Abbreviate the phrase context-free grammar to CFG.
- Abbreviate the phrase context-free language to CFL.

Notations

- Abbreviate the phrase context-free grammar to CFG.
- Abbreviate the phrase context-free language to CFL.
- 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

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

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

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:

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:

$$\begin{array}{rcl} A & \longrightarrow & 0A1 \\ A & \longrightarrow & B \\ B & \longrightarrow & \# \end{array}$$

Note

- Nonterminals of CFG G_1 are $\{A, B\}$ and A is the axiom

Note

- Nonterminals of CFG G_1 are $\{A, B\}$ and 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

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

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

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.

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.
- 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:

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 variable; it is the lhs of one of the specifi cation rules, the top rule, unless specifi ed otherwise

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 variable; it is the lhs of one of the specifi cation rules, the top rule, unless specifi ed 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

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 variable; it is the lhs of one of the specifi cation rules, the top rule, unless specifi ed 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

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

Example string generation

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

A⇒0A1⇒00A11⇒000A111⇒000B111⇒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

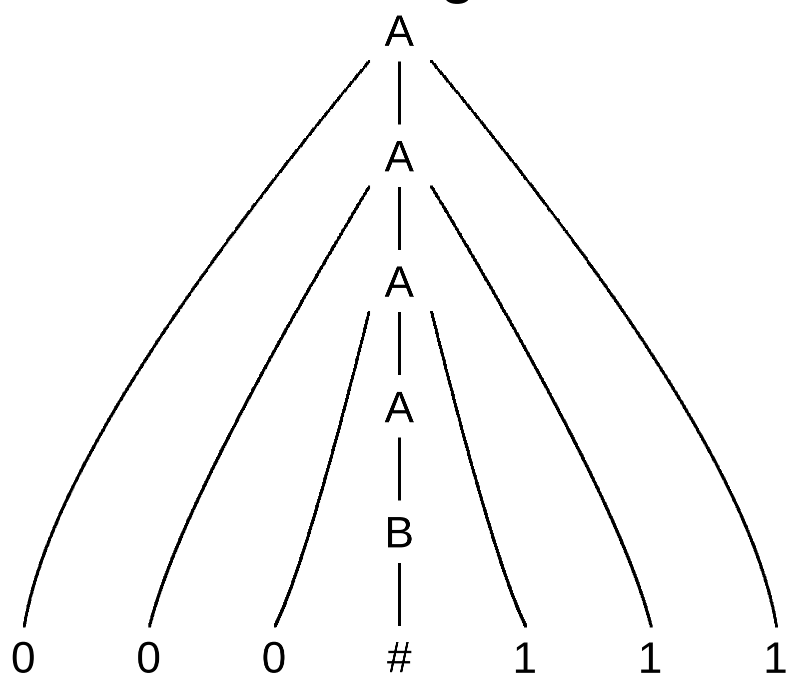

Figure 1: Derivation tree for `000#111`

Note

- All strings of terminals generated in this way constitute the language specified by the grammar

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, $L(G_1) = \{0^n\#1^n | n \geq 0\}$.

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, $L(G_1) = \{0^n\#1^n | n \geq 0\}$.
- 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 “,".

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 $lhs \rightarrow rhs_1|rhs_2| \ldots rhs_n$ where $|$ is used with the meaning of an “or".

Example compaction

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

CFG G_2

The CFG G_2 specifi es a fragment of English

$$
\begin{array}{rcl}
\langle SENTENCE \rangle & \longrightarrow & \langle NounPhrase \rangle\ \langle VerbPhrase \rangle \\
\langle NounPhrase \rangle & \longrightarrow & \langle CpNoun \rangle | \langle CpNoun \rangle\ \langle PrepPhrase \rangle \\
\langle VerbPhrase \rangle & \longrightarrow & \langle CpVerb \rangle | \langle CpVerb \rangle\ \langle PrepPhrase \rangle \\
\langle PrepPhrase \rangle & \longrightarrow & \langle Prep \rangle\ \langle CpNoun \rangle \\
\langle CpNoun \rangle & \longrightarrow & \langle Article \rangle\ \langle Noun \rangle \\
\langle CpVerb \rangle & \longrightarrow & \langle Verb \rangle | \langle Verb \rangle\ \langle NounPhrase \rangle \\
\langle Article \rangle & \longrightarrow & a | the \\
\langle Noun \rangle & \longrightarrow & boy | girl | flower \\
\langle Verb \rangle & \longrightarrow & touches | likes | sees \\
\langle Prep \rangle & \longrightarrow & with
\end{array}
$$

Note

- The CFG G_2 has ten variables (capitalized and in angular brackets) and 9 terminals (written in the standard English alphabet) plus a space character

Note

- The CFG G_2 has ten variables (capitalized and in angular brackets) and 9 terminals (written in the standard English alphabet) plus a space character
- Also, the CFG G_2 has 18 rules

Note

- The CFG G_2 has ten variables (capitalized and in angular brackets) and 9 terminals (written 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

$$
\begin{aligned}
\langle SENTENCE\rangle &\Rightarrow \langle NounPhrase\rangle\ \langle VerbPhrase\rangle \\
&\Rightarrow \langle CpNoun\rangle\ \langle VerbPhrase\rangle \\
&\Rightarrow \langle Article\rangle\ \langle Noun\rangle\ \langle VerbPhrase\rangle \\
&\Rightarrow a\ \langle Noun\rangle\ \langle VerbPhrase\rangle \\
&\Rightarrow a\ boy\ \langle VerbPhrase\rangle \\
&\Rightarrow a\ boy\ \langle CpVerb\rangle \\
&\Rightarrow a\ boy\ \langle Verb\rangle \\
&\Rightarrow a\ boy\ sees
\end{aligned}
$$

Formal definition of a CFG

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

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

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

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 (or specification rules) of the form $lhs \rightarrow rhs$, where $lhs \in V$, $rhs \in (V \cup \Sigma)^*$

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 (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 (or grammar axiom)

Example CFG grammar

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

$$
\begin{array}{rcl}
A & \longrightarrow & 0A1 \\
A & \longrightarrow & B \\
B & \longrightarrow & \#
\end{array}
$$

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 uwv, written $uAv \Rightarrow uwv$

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 uwv, written $uAv \Rightarrow uwv$
- We may also say that uwv is directly derived from uAv using the rule $A \longrightarrow w$

Derivation

- We write $u \stackrel{*}{\Rightarrow} v$ if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k \in (V \cup \Sigma)^*$ exists, for $k \geq 0$, and $u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v$

Derivation

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

$L(G) = \{w \in \Sigma^* | S \stackrel{*}{\Rightarrow} w\}$

Note

- Often we specify a grammar by writing down only its rules

Note

- 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

Note

- 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:

More examples of CFGs

- Consider the grammar:

$$G_3 = (\{S\}, \{a, b\}, \{S \longrightarrow aSb|SS|\epsilon\}, S)$$

More examples of CFGs

- Consider the grammar:

  $G_3 = (\{S\}, \{a, b\}, \{S \longrightarrow aSb|SS|\epsilon\}, S)$

- $L(G_3)$ contains strings such as:

More examples of CFGs

- Consider the grammar:

  $G_3 = (\{S\}, \{a, b\}, \{S \longrightarrow aSb|SS|\epsilon\}, S)$

- $L(G_3)$ contains strings such as:

  abab, aaabbb, aababb;

More examples of CFGs

- Consider the grammar:

  $G_3 = (\{S\}, \{a, b\}, \{S \longrightarrow aSb|SS|\epsilon\}, S)$

- $L(G_3)$ 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:

Arithmetic expressions

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

$$
\begin{array}{rcl}
E & \longrightarrow & E + T \mid T \\
T & \longrightarrow & T * F \mid F \\
F & \longrightarrow & (E) \mid a
\end{array}
$$

Arithmetic expressions

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

$$\begin{array}{lcl} E & \longrightarrow & E + T \mid T \\ T & \longrightarrow & T * F \mid F \\ F & \longrightarrow & (E) \mid a \end{array}$$

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

Note

- The variables and constants in $L(G_4)$ are represented by the terminal a

Note

- The variables and constants in $L(G_4)$ are represented by the terminal a
- Arithmetic operations in $L(G_4)$ are addition, represented by +, and multiplication, represented by *

Note

- The variables and constants in $L(G_4)$ 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

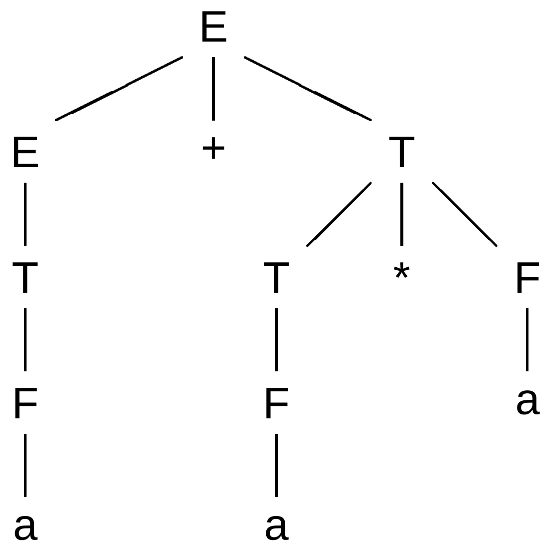

Figure 2: Derivation tree for `a+a*a`

Designing CFGs

- As with the design of automata, the design of CFGs requires creativity

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

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

Example grammar design

Design a grammar for the language

$$\{0^n1^n|n \geq 0\} \cup \{1^n0^n|n \geq 0\}$$

Example grammar design

Design a grammar for the language

$\{0^n1^n|n \geq 0\} \cup \{1^n0^n|n \geq 0\}$

1. Construct the grammar $S_1 \longrightarrow 0S_11|\epsilon$ that generates $\{0^n1^n|n \geq 0\}$

Example grammar design

Design a grammar for the language

$\{0^n1^n | n \geq 0\} \cup \{1^n0^n | n \geq 0\}$

1. Construct the grammar $S_1 \longrightarrow 0S_11|\epsilon$ that generates $\{0^n1^n|n \geq 0\}$
2. Construct the grammar $S_2 \longrightarrow 1S_20|\epsilon$ that generates $\{1^n0^n|n \geq 0\}$

Example grammar design

Design a grammar for the language

$\{0^n1^n|n \geq 0\} \cup \{1^n0^n|n \geq 0\}$

1. Construct the grammar $S_1 \longrightarrow 0S_11|\epsilon$ that generates $\{0^n1^n|n \geq 0\}$
2. Construct the grammar $S_2 \longrightarrow 1S_20|\epsilon$ that generates $\{1^n0^n|n \geq 0\}$
3. Put them together adding the rule $S \longrightarrow S_1|S_2$ thus getting

$$S \longrightarrow S_1|S_2$$
$$S_1 \longrightarrow 0S_11|\epsilon$$
$$S_2 \longrightarrow 1S_20|\epsilon$$

Second design technique

- Constructing a CFG for a regular language is easy if one can first construct a DFA for that language

Second design technique

- Constructing a CFG for a regular language is easy if one can first construct a DFA for that language
- Conversion procedure:

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

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 $R_i \longrightarrow aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in the DFA

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 $R_i \longrightarrow aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in the DFA
  3. Add the rule $R_i \longrightarrow \epsilon$ if q_i is an accept state of the DFA

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 $R_i \longrightarrow aR_j$ to the CFG if $\delta(q_i, a) = q_j$ is a transition in the DFA
    3. Add the rule $R_i \longrightarrow \epsilon$ 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.

Note

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 $\{0^n1^n | n \geq 0\}$

Third design technique

- Certain CFLs contain strings with two related substrings as are 0^n and 1^n in $\{0^n1^n|n \geq 0\}$
- Example of relationship: to recognize such a language a machine would need to remember an unbounded amount of info about one of the substrings

Note

A CFG that handles this situation uses a rule 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

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.

Note

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