

Introduction to Automata Theory, Language and computation

Shen Lu

shenlu@mail.usf.edu

Department of Computer Science and
Engineering

University of South Florida

Outline

- Formalism
- Derivations
- Backup-Naur FORM
- Left-most and right-most derivations

Productions

- A *production* has the form *variable (head)* \rightarrow *string of variables and terminals (body)*.
- **Convention:**
 - A, B, C,... and also S are variables.
 - a, b, c,... are terminals.
 - ..., X, Y, Z are either terminals or variables.
 - ..., w, x, y, z are strings of terminals only.
 - α , β , γ ,... are strings of terminals and/or variables.

Informal Comments

- A *context-free grammar* is a notation for describing languages.
- It is more powerful than finite automata or RE's, but still cannot define all possible languages.
- Useful for nested structures, e.g., parentheses in programming languages.

Derivations – Formalism

- We say $\alpha A \beta \Rightarrow \alpha \gamma \beta$ if $A \rightarrow \gamma$ is a production.
- **Example:** $S \rightarrow 01$; $S \rightarrow 0S1$.
- $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$.

Informal Comments – (2)

- Basic idea is to use “variables” to stand for sets of strings (i.e., languages).
- These variables are defined recursively, in terms of one another.
- Recursive rules (“productions”) involve only concatenation.
- Alternative rules for a variable allow union.

Example: CFG for $\{ 0^n 1^n \mid n \geq 1 \}$

- Productions:

$S \rightarrow 01$

$S \rightarrow 0S1$

- **Basis**: 01 is in the language.
- **Induction**: if w is in the language, then so is $0w1$.

CFG Formalism

- *Terminals* = symbols of the alphabet of the language being defined.
- *Variables* = *nonterminals* = a finite set of other symbols, each of which represents a language.
- *Start symbol* = the variable whose language is the one being defined.

Example: Formal CFG

- Here is a formal CFG for $\{ 0^n 1^n \mid n \geq 1 \}$.
- Terminals = $\{0, 1\}$.
- Variables = $\{S\}$.
- Start symbol = S .
- Productions =
 - $S \rightarrow 01$
 - $S \rightarrow 0S1$

Derivations – Intuition

- We *derive* strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the body of one of its productions.
- That is, the “productions for A ” are those that have head A .

Derivations – Intuition

- We *derive* strings in the language of a CFG by starting with the start symbol, and repeatedly replacing some variable A by the body of one of its productions.
- That is, the “productions for A ” are those that have head A .

Iterated Derivation

- \Rightarrow^* means “zero or more derivation steps.”
- **Basis:** $\alpha \Rightarrow^* \alpha$ for any string α .
- **Induction:** if $\alpha \Rightarrow^* \beta$ and $\beta \Rightarrow \gamma$, then $\alpha \Rightarrow^* \gamma$.

Example: Iterated Derivation

- $S \rightarrow 01; S \rightarrow 0S1.$
- $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111.$
- Thus $S \Rightarrow^* S; S \Rightarrow^* 0S1; S \Rightarrow^* 00S11; S \Rightarrow^* 000111.$

Sentential Forms

- Any string of variables and/or terminals derived from the start symbol is called a *sentential form*.
- Formally, α is a sentential form iff $S \Rightarrow^* \alpha$.

Language of a Grammar

- If G is a CFG, then $L(G)$, the *language of G* , is $\{w \mid S \Rightarrow^* w\}$.
- **Example:** G has productions $S \rightarrow \varepsilon$ and $S \rightarrow 0S1$.
- $L(G) = \{0^n 1^n \mid n \geq 0\}$.

Context-Free Languages

- A language that is defined by some CFG is called a *context-free language*.
- There are CFL's that are not regular languages, such as the example just given.
- But not all languages are CFL's.
- **Intuitively**: CFL's can count two things, not three.

BNF Notation

- Grammars for programming languages are often written in BNF (*Backus-Naur Form*).
- Variables are words in <...>; **Example:** <statement>.
- Terminals are often multicharacter strings indicated by boldface or underline; **Example:** **while** or WHILE.

BNF Notation – (2)

- Symbol $::=$ is often used for \rightarrow .
- Symbol $|$ is used for “or.”
 - A shorthand for a list of productions with the same left side.
- **Example:** $S \rightarrow 0S1 \mid 01$ is shorthand for $S \rightarrow 0S1$ and $S \rightarrow 01$.

BNF Notation – Kleene Closure

- Symbol ... is used for “one or more.”
- **Example:** $\langle \text{digit} \rangle ::= 0|1|2|3|4|5|6|7|8|9$
 $\langle \text{unsigned integer} \rangle ::= \langle \text{digit} \rangle \dots$
- **Translation:** Replace $\alpha \dots$ with a new variable A and productions $A \rightarrow A\alpha \mid \alpha$.

Example: Kleene Closure

- Grammar for unsigned integers can be replaced by:

$U \rightarrow UD \mid D$

$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

BNF Notation: Optional Elements

- Surround one or more symbols by [...] to make them optional.
- **Example:** $\langle \text{statement} \rangle ::= \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement} \rangle [; \text{else } \langle \text{statement} \rangle]$
- **Translation:** replace $[\alpha]$ by a new variable A with productions $A \rightarrow \alpha \mid \varepsilon$.

Example: Optional Elements

- Grammar for if-then-else can be replaced by:

$S \rightarrow iCtSA$

$A \rightarrow ;eS \mid \varepsilon$

BNF Notation – Grouping

- Use {...} to surround a sequence of symbols that need to be treated as a unit.
 - Typically, they are followed by a ... for “one or more.”
- **Example:** <statement list> ::= <statement> [{;<statement>}...]

Translation: Grouping

- Create a new variable A for $\{\alpha\}$.
- One production for A : $A \rightarrow \alpha$.
- Use A in place of $\{\alpha\}$.

Example: Grouping

$L \rightarrow S \{ ;S \} \dots$

- Replace by $L \rightarrow S [A \dots]$ $A \rightarrow ;S$
 - A stands for $\{ ;S \}$.
- Then by $L \rightarrow SB$ $B \rightarrow A \dots \mid \epsilon$ $A \rightarrow ;S$
 - B stands for $[A \dots]$ (zero or more A's).
- Finally by $L \rightarrow SB$ $B \rightarrow C \mid \epsilon$ $C \rightarrow AC \mid A$
 $A \rightarrow ;S$
 - C stands for $A \dots$.

Leftmost and Rightmost Derivations

- Derivations allow us to replace any of the variables in a string.
 - Leads to many different derivations of the same string.
- By forcing the leftmost variable (or alternatively, the rightmost variable) to be replaced, we avoid these “distinctions without a difference.”

Leftmost Derivations

- Say $wA\alpha \Rightarrow_{lm} w\beta\alpha$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.
- Also, $\alpha \Rightarrow_{lm}^* \beta$ if α becomes β by a sequence of 0 or more \Rightarrow_{lm} steps.

Example: Leftmost Derivations

- Balanced-parentheses grammar:

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

- $S \Rightarrow_{lm} SS \Rightarrow_{lm} (S)S \Rightarrow_{lm} (())S \Rightarrow_{lm} (())()$
- Thus, $S \Rightarrow_{lm}^* (())()$
- $S \Rightarrow SS \Rightarrow S() \Rightarrow (S)() \Rightarrow (())()$ is a derivation, but not a leftmost derivation.

Rightmost Derivations

- Say $\alpha Aw \Rightarrow_{rm} \alpha \beta w$ if w is a string of terminals only and $A \rightarrow \beta$ is a production.
- Also, $\alpha \Rightarrow_{rm}^* \beta$ if α becomes β by a sequence of 0 or more \Rightarrow_{rm} steps.

Example: Rightmost Derivations

- Balanced-parentheses grammar:
 - $S \rightarrow SS \mid (S) \mid ()$
- $S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$
- Thus, $S \Rightarrow_{rm}^* (())()$
- $S \Rightarrow SS \Rightarrow SSS \Rightarrow S()S \Rightarrow ()()S \Rightarrow ()()()$ is neither a rightmost nor a leftmost derivation.

IN-class Exercise

- **function =**
 - **if(condition) then**
 - **statement**
 - **else**
 - **statement**
- **the grammar for this if-then-else is**
 - $S \rightarrow \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement} \rangle \text{ A}$**
 - $A \rightarrow ;eS \mid \epsilon \text{ (;else}\langle \text{statement} \rangle|\epsilon)$**
- **please design an automata to simulate the compiler.**
- **how to write a compiler program for if-then-else statement**