CHOMSKY HIERARCHY

Jie Jiang

October 4, 2009
Avram Noam Chomsky

- Born on December 7, 1928, currently professor emeritus of linguistics at MIT.
- American linguist, philosopher, cognitive scientist, political activist, author, and lecturer.
- One of the fathers of modern linguistics.
- Created the theory of generative grammar, which has undergone numerous revisions and has had a profound influence on linguistics.
- Sparked the cognitive revolution in psychology.
Biography

- From 1945, studied philosophy and linguistics at the University of Pennsylvania.
- PhD in linguistics from University of Pennsylvania in 1955.
- 1956, appointed full Professor at MIT, Department of Linguistics and Philosophy.
- 1966, Ferrari P. Ward Chair; 1976, Institute Professor; currently Professor Emeritus.
Contribution

- **Linguistics**
  - Transformational grammars
  - Generative grammar
  - Language acquisition

- **Computer Science**
  - Chomsky hierarchy
  - Chomsky Normal Form
  - Context Free Grammars

- **Psychology**
  - Cognitive Revolution (1959)
  - Universal grammar
Universal grammar

- Noam Chomsky’s approach to the study of language is known as universal grammar:
  - the human brain contains a limited set of rules for organizing language. In turn, there is an assumption that all languages have a common structural basis. This set of rules is known as universal grammar
  - an innate set of linguistic principles shared by all humans
  - attempts to explain language acquisition in general, not describe specific languages
  - proposes a set of rules intended to explain language acquisition in child development
Marcel-Paul Schützenberger

- French mathematician and Doctor of Medicine.
- Professor of the Faculty of Sciences, University of Paris
- Full Member of French Academy of Sciences.
Biography

- First trained as a physician, doctorate in medicine in 1948.
- PhD in mathematics in 1953
- Professor at the University of Poitiers, 1957-1963
- Director of research at the CNRS, 1963-1964
- Professor in the Faculty of Sciences at the University of Paris, 1964-1996
Contribution

- **Formal languages with Noam Chomsky**
  - Chomsky-Schützenberger hierarchy
  - Chomsky-Schützenberger theorem

- **Automata**
  with Samuel Ellenberger

- **Biology and Darwinism**
  - Mathematical critique of neodarwinism (1966)
In computational practice, data are encoded in the computer's memory as strings of bits or other symbols appropriate for manipulation by a computer.

How to understand the mathematics of strings of symbols?

**Definition**: An *alphabet* is a finite, nonempty set of symbols. We use $\Sigma$ to denote this alphabet.

- the Roman alphabet $\{a, b, \ldots, z\}$.
- binary alphabet $\{0, 1\}$:
  - the alphabet particularly pertinent to the theory of computation.

Note: any object can be in an alphabet.

In a formal point of view, an alphabet is simply a finite set of any sort.

For simplicity, however, we use as symbols only letters, numerals, and other common characters such as $\$, or $\#$. 
A *string* is a finite sequence of symbols from $\Sigma$. We simply juxtapose the symbols to denote a string.

- *watermelon* is a string over the alphabet $\{a, b, \ldots, z\}$.
- 0111011 is a string over $\{0, 1\}$.

A string has no symbols at all
  $\Rightarrow$ the **empty string**
  $\Rightarrow$ denoted by $\epsilon$.

We generally use $u, v, x, y, z,$ and Greek letters to denote strings.

- Example: we use $w$ as a name for the string $abc$. 
The **length** of a string $s$, denoted $|s|$, is the number of symbols in it.

- $|101| = 3$
- $|\epsilon| = 0$

A string $w \in \Sigma^*$, can be considered as a function $w : \{1, \ldots, |w|\} \rightarrow \Sigma$, the value of $w(j)$, where $1 \leq j \leq |w|$, is the symbol in the $j$th position of $w$.

- Example: if $w = \text{accordion}$, then $w(3) = w(2) = c$, and $w(1) = a$.

**Occurrences** of a symbol in a string: the symbol $\sigma \in \Sigma$ occurs in the $j$th position of the string $w \in \Sigma^*$ if $w(j) = \sigma$. 
Operations on strings

- **Concatenation**: $x \circ y$ (or simply $xy$)
  - Formally, $w = x \circ y$ iff $|w| = |x| + |y|$, $w(j) = x(j)$ for $j = 1, \ldots, |x|$, and $w(|x| + j) = y(j)$ for $j = 1, \ldots, |y|$.
  - Example: $01 \circ 001 = 01001$, $\text{beach} \circ \text{boy} = \text{beathboy}$
  - Note: $w \circ \epsilon = \epsilon \circ w$ for any string $|w|$
  - Associative: $(wx)y = w(xy)$ for any strings $w$, $x$ and $y$.

- A string $v$ is a **substring** of a string $w$ iff there are strings $x$ and $y$ such that $w = xvy$.
  - Note: every string is a substring of itself. ($x = \epsilon$ and $y = \epsilon$)
  - $\epsilon$ is a substring of every string.

- If $w = xv$ for some $x$, then $v$ is a **suffix** of $w$.
- If $w = vy$ for some $y$, then $v$ is a **prefix** of $w$.
- Example: $\text{road}$ is a prefix of $\text{roadrunner}$, a suffix of $\text{abroad}$, and a substring of both these and of $\text{broader}$.
Operations on strings

**Definition** $w^i$:
for each string $w$ and each natural number $i$, the string $w^i$ is defined as

\[
\begin{align*}
    w^0 &= \epsilon, \text{ the empty string} \\
    w^{i+1} &= w^i \circ w, \text{ for each } i \geq 0
\end{align*}
\]

**Definition by induction**: $w^{i+1}$ is defined in terms of $w^i$.

- Example: $(do)^2 = ?$
- $(do)^2 = (do)^1 \circ do, (i = 1)$
- $(do)^1 = (do)^0 \circ do, (i = 0)$
- $(do)^0 = \epsilon$
- $\Rightarrow (do)^2 = (e \circ do) \circ do = dodo$
Operations on strings

Definition $w^R$: reversal of a string $w$, denoted by $w^R$, is defined as

- If $|w| = 0$, then $w^R = w = \epsilon$
- If $|w| = n + 1 \geq 0$, then $w = ua$ for some $a \in \Sigma$, and $w^R = au^R$

The string "spelled backwards":

- Example: $reverse^R = esrever$.

Note: $(wx)^R = x^Rw^R$, for any strings $w$ and $x$. 
Languages definition

- \( \Sigma^* \) denotes the set of all sequences of strings that are composed of zero or more symbols of \( \Sigma \).
- \( \Sigma^+ \) denotes the set of all sequences of strings composed of one or more symbols of \( \Sigma \).
  \[ \Sigma^+ = \Sigma^* - \{ \epsilon \} \]
- A language is a subset of \( \Sigma^* \).
- A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols.
  - \( \Sigma^* \), \( \emptyset \) and \( \Sigma \) are languages.
  - \( \{ aba, czr, d, f \} \) is a language over \( \{ a, b, \ldots, z \} \).
- The infinite languages are specified by the scheme:
  \[ L = \{ w \in \Sigma^* : w \text{ has property } P \} \]
  - \( \{ w \in \{ 0, 1 \}^* : w \text{ has an equal number of 0's and 1's} \} \)
  - \( \{ w \in \Sigma^* : w = w^R \} \)
Operations on languages

- **Concatenation**: if \( L_1 \) and \( L_2 \) are languages over \( \Sigma \), their concatenation is \( L = L_1 \circ L_2 \), or simply \( L = L_1 L_2 \), where

  \[
  L = \{ w \in \Sigma^* : w = x \circ y \text{ for some } x \in L_1 \text{ and } y \in L_2 \}
  \]

  Example: \( \Sigma = \{0, 1\} \),
  \[
  L_1 = \{ w \in \Sigma^* : w \text{ has an even number of 0's} \},
  \]
  \[
  L_2 = \{ w \in \Sigma^* : w \text{ starts with a 0 and the rest of the symbols are 1's} \}
  \]

  \[
  \Rightarrow L_1 \circ L_2 = \{ w \in \Sigma^* : w \text{ has an odd number of 0's} \}
  \]
Operations on languages

- **Kleene star** of a language $L$, denoted by $L^*$, is the set of all strings obtained by concatenating zero or more strings from $L$:

  $$L^* = \{w \in \Sigma^* : w = w_1 \circ \cdots \circ w_k \text{ for some } k \geq 0 \text{ and some } w_1, \ldots, w_k \in L\}$$

- Example: $\Sigma = \{0, 1\}$,
  $L = \{01, 1, 100\}$, $\Rightarrow$ $110001110011 \in L^*$
  since $110001110011 = 1 \circ 100 \circ 01 \circ 1 \circ 100 \circ 1 \circ 1$, and each of these strings is in $L$s.
A **grammar** can be regarded as a device that enumerates the sentences of a language.

Types of grammars

- **Prescriptive**: prescribes authoritative norms for a language
- **Descriptive**: attempts to describe actual usage rather than enforce arbitrary rules
- **Formal**: a precisely defined grammar, such as context-free
- **Generative**: a formal grammar that can generate natural language expressions
Formal grammars

- Two broad categories of formal languages: **generative** and **analytic**
- A **generative** grammar formalizes an algorithm that generates valid strings in a language
- An **analytic** grammar is a set of rules to reduce an input string to a boolean result that indicates the validity of the string in the given language
- A generative grammar describes how to **write** a language, and an analytic grammar describes how to **read** it (a parser).
How to determine whether a string is valid within a given language?

A grammar is a set of formation rules that describe which strings formed from the alphabet of a language are syntactically valid within the language.

- Grammars are composed of:
  - a set of string rewriting rules
  - an assigned start symbol

The language described by a grammar is the set of strings that can be generated by applying these rules arbitrarily, starting with the start symbol.

Grammars could be thought of language generators or string recognizers for languages.
Assume the alphabet consists of $a$ and $b$, the start symbol is $S$, rules are:

1. $S \rightarrow aSb$
2. $S \rightarrow ba$

Operations: start with $S$, and choose a rule to apply to it

- choose rule 1, obtain the string $aSb$
- choose rule 1 again, replace $S$ with $aSb \Rightarrow aaSbb$
- repeated at will until all occurrences of $S$ are removed, and only symbols from the alphabet remain (i.e., $a$ and $b$).
- choose rule 2, replace $S$ with $ba \Rightarrow aababb$
- $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aababb$

The language of the grammar is the set of all the strings that can be generated using this process:

$$\{ba, abab, aababb, aaababbb, \ldots\}$$
Grammars

- Defined as a quadruple $\langle V_t, V_n, P, S \rangle$, where:
  - $V_t$ = the *terminal vocabulary*
    i.e. the words of the language: finite (but productive ...)
  - $V_n$ = the *non-terminals*
    i.e. the set of categories, used for the benefit of the grammar to express generalizations over items in $V_t$
  - $P$ = the *set of productions*,
    i.e. rules in the grammar: finite, and of the form
    $$\alpha \rightarrow \beta$$
    where $\alpha, \beta \in (V_n \cup V_t)$
  - $S$ = the *distinguished symbol*, or ‘start’ symbol (‘sentence’)
    - $S \in V_n$,
    - $P$ must include at least one rule $\alpha \rightarrow \beta$ where $\alpha = S$. 
A grammar is *generative* if it predicts (explicitly defines, or characterizes) *all and only all* strings $\in$ the language.

The language of a grammar $G$, denoted as $L(G)$, is defined as all those strings over $\Sigma$ that can be generated by $G$.

$$L(G) = \{ w \in \Sigma^* : S \Rightarrow^*_G w \}$$
Consider the grammar $G$ where $V_n = \{S, B\}$, $V_t = a, b, c$, $S$ is the start symbol, and $P$ consists of:

1. $S \rightarrow aBSc$
2. $S \rightarrow abc$
3. $Ba \rightarrow aB$
4. $Bb \rightarrow bb$

Some examples of the derivation of strings in $L(G)$ are:

1. $S \Rightarrow_2 abc$
2. $S \Rightarrow_1 aBSc \Rightarrow_2 aBabcc \Rightarrow_3 aaBbcc \Rightarrow_4 aabbc$
3. $S \Rightarrow_1 aBSc \Rightarrow_1 aBaBSc \Rightarrow_2 aBaBabccc \Rightarrow_3 aaBBabccc \Rightarrow_3 aaBabBbccc \Rightarrow_3 aaaBBbcc \Rightarrow_4 aaaBbbccc \Rightarrow_4 aabbbcc$
### Chomsky Hierarchy

<table>
<thead>
<tr>
<th>Type</th>
<th>Grammar</th>
<th>Language</th>
<th>Automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Finite State</td>
<td>Regular</td>
<td>Finite</td>
</tr>
<tr>
<td>2</td>
<td>Context-Free</td>
<td>C-F</td>
<td>Pushdown</td>
</tr>
<tr>
<td>1</td>
<td>Context-Sensitive</td>
<td>C-S</td>
<td>Linear-Bounded</td>
</tr>
<tr>
<td>0</td>
<td>General Rewrite</td>
<td>Unrestricted</td>
<td>Turing Machines</td>
</tr>
</tbody>
</table>

- Occasionally referred to as Chomsky–Schützenberger hierarchy
- Described by Noam Chomsky in 1956
- Hierarchy of grammars
  - Type 3 ⊂ Type 2 ⊂ Type 1 ⊂ Type 0
### Type-3 grammars

<table>
<thead>
<tr>
<th>Type</th>
<th>Grammar</th>
<th>Language</th>
<th>Automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Finite State</td>
<td>Regular</td>
<td>Finite</td>
</tr>
<tr>
<td>2</td>
<td>Context-Free</td>
<td>C-F</td>
<td>Pushdown</td>
</tr>
<tr>
<td>1</td>
<td>Context-Sensitive</td>
<td>C-S</td>
<td>Linear-Bounded</td>
</tr>
<tr>
<td>0</td>
<td>General Rewrite</td>
<td>Unrestricted</td>
<td>Turing Machines</td>
</tr>
</tbody>
</table>

- **regular grammars**: $A \rightarrow a$, and $A \rightarrow aB$
  - a single nonterminal on the left-hand side.
  - a single terminal, possibly followed (or preceded, but not both) by a single nonterminal on the right-hand side.
  - $S \rightarrow \epsilon$ is allowed if $S$ is NOT on the right side of any rule
- generate the **regular languages**:
  - can be decided by *finite state automata*.
  - can be obtained by *regular expressions* (search patterns & lexical structure of programming languages)
## Type-2 grammars

<table>
<thead>
<tr>
<th>Type</th>
<th>Grammar</th>
<th>Language</th>
<th>Automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Finite State</td>
<td>Regular</td>
<td>Finite</td>
</tr>
<tr>
<td>2</td>
<td>Context-Free</td>
<td>C-F</td>
<td>Pushdown</td>
</tr>
<tr>
<td>1</td>
<td>Context-Sensitive</td>
<td>C-S</td>
<td>Linear-Bounded</td>
</tr>
<tr>
<td>0</td>
<td>General Rewrite</td>
<td>Unrestricted</td>
<td>Turing Machines</td>
</tr>
</tbody>
</table>

- **context-free grammars:** $A \rightarrow \gamma$
  - a single nonterminal on the left-hand side.
  - a string of terminals and nonterminals on the right-hand side.
  (less restrictive than Type-3)

- generate the **context-free languages**:
  - can be recognized by *non-deterministic pushdown automata*.
  - theoretical basis for the syntax of most programming languages.
### Type-1 grammars

<table>
<thead>
<tr>
<th>Type</th>
<th>Grammar</th>
<th>Language</th>
<th>Automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Finite State</td>
<td>Regular</td>
<td>Finite</td>
</tr>
<tr>
<td>2</td>
<td>Context-Free</td>
<td>C-F</td>
<td>Pushdown</td>
</tr>
<tr>
<td>1</td>
<td>Context-Sensitive</td>
<td>C-S</td>
<td>Linear-Bounded</td>
</tr>
<tr>
<td>0</td>
<td>General Rewrite</td>
<td>Unrestricted</td>
<td>Turing Machines</td>
</tr>
</tbody>
</table>

- **context-sensitive grammars**: $\alpha A\beta \rightarrow \alpha \gamma \beta$
  - $\alpha$ and $\beta$ may be empty.
  - $A$ is a single nonterminal. (less restrictive than Type-2)
  - $\gamma$ is a nonempty string of nonterminals and terminals. (less restrictive)
  - $S \rightarrow \epsilon$ is allowed if $S$ is NOT on the right side of any rule.

- generate the **context-sensitive languages**:
  - can be recognized by *linear bounded automata* (a nondeterministic Turing machine whose tape is bounded by a constant times the length of the input).
## Type-0 grammars

<table>
<thead>
<tr>
<th>Type</th>
<th>Grammar</th>
<th>Language</th>
<th>Automata</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Finite State</td>
<td>Regular</td>
<td>Finite</td>
</tr>
<tr>
<td>2</td>
<td>Context-Free</td>
<td>C-F</td>
<td>Pushdown</td>
</tr>
<tr>
<td>1</td>
<td>Context-Sensitive</td>
<td>C-S</td>
<td>Linear-Bounded</td>
</tr>
<tr>
<td>0</td>
<td>General Rewrite</td>
<td>Unrestricted</td>
<td>Turing Machines</td>
</tr>
</tbody>
</table>

- **unrestricted grammars:** $\alpha \rightarrow \beta$
- **generate the recursively enumerable languages:**
  - can be recognized by *Turing machines.*
Exercise 1

- Given $\Sigma = \{0, 1\}$, please provide the specifications for the following languages:
  - The language consists of all strings begin with 0.
    - $\{0\}\{0, 1\}^*$
  - The language consists of all strings begin with 0, and end with 1.
    - $\{0\}\{0, 1\}^*\{1\}$
  - The language consists of all strings with odd lengths.
    - $\{0, 1\}^{2n-1}$, $n = 1, 2, \ldots$
  - The language consists of all strings with substring of three consecutive 0.
    - $\{0, 1\}^*000\{0, 1\}^*$
  - The language consists of all strings without substring of three consecutive 0.
    - $\{001, 01, 1\}^*$
Consider the grammar $G$ where $V_n = \{S, B, C\}$, $V_t = a, b, c$, $S$ is the start symbol, and $P$ consists of:

1. $S \rightarrow aBC$
2. $S \rightarrow aSBC$
3. $aB \rightarrow ab$
4. $bB \rightarrow bb$
5. $CB \rightarrow BC$
6. $bC \rightarrow bc$
7. $cC \rightarrow cc$

Try to provide one derivation for sentence $aaabbbccc$.

$S \Rightarrow_2 aSBC \Rightarrow_2 aaSBCBC \Rightarrow_1 aaaBCBCBC \Rightarrow_3$

$aaabCBCBC \Rightarrow_5 aaabBBCCBC \Rightarrow_4 aaabbCCBC \Rightarrow_5$

$aaabbCBCC \Rightarrow_5 aaabbbBCCC \Rightarrow_4 aaabbbCCC \Rightarrow_6$

$aaabbbbcC \Rightarrow_7 aaabbbccC \Rightarrow_7 aaabbbccc$
1 Personalities
   - Noam Chomsky
   - Marcel Schützenberger

2 Languages
   - Alphabets
   - Languages

3 Grammars
   - Introduction
   - Formal definition
   - Examples

4 Chomsky Hierarchy