I. INTRODUCTION

A. Two Classes of Languages

1. Natural languages -- example: English
   Attempts to define these languages use transformational grammars, composed of
   a. Phrase structures
   b. Transformational rules with a structural index and a structural change
   c. Lexicon

2. Artificial languages -- examples: Fortran, ALGOL
   Formal definition in terms of
   a. Phrase structure
   b. Semantic interpretation rules

B. Basic Problems in Linguistics

1. Formal definition of languages
2. Recognition (Decision) Problem: determine if a string is in the language.
3. Generation Problem: generate all strings in the language
4. Analysis Problem: determine all structures of a string in a language

Examples of English sentences with more than one structure:
1. The Danes like good food as well as Americans.
2. Geo. Murphy is the one to watch.
3. I just can't see flying kites.
II. GRAMMAR (PHRASE STRUCTURE SYNTAX)

Definition: The vocabulary \( \mathcal{V} \) is the set of all symbols used in the definition of a grammar.

Examples:
1. ALGOL: all visible characters, delimiters, word delimiters and meta-linguistic variables.
   
   \[ \text{A, Integer, <identifier>} \]

2. \( \{ \alpha | \alpha = a^n b^n, n \geq 1 \} \)
   \[ S, a, b \]
   \[ S \rightarrow Sb | ab \]

Definition: \( \mathcal{V}^* \) is the set of all strings generated by concatenation of the elements of \( \mathcal{V} \). (This includes the null string \( \Lambda \).)

Nomenclature: Denote all members of \( \mathcal{V} \) by either upper or lower case latin letters and strings in \( \mathcal{V}^* \) by lower case greek letters.

Definition: A syntax rule (production) is a rewriting rule \( \alpha \rightarrow \beta \), \( \alpha, \beta \in \mathcal{V}^* \) on a string.

Examples:
1. Backus Normal Form

   \[ \text{<bound pair list>::= <bound pair>} \]
   \[ \text{<bound pair list>::= <bound pair list>,<bound pair>} \]
   
   abbreviation for the above two rules
   \[ \text{<bound pair list>::= <bound pair>} | \]
   \[ \text{<bound pair list>,<bound pair>} \]

2. \( A \rightarrow aaA \)
3. \( ABA \rightarrow AOB \)
4. \( <\text{number}>::= <\text{digit} <\text{number}><\text{digit}> \]
   \[ <\text{digit}>::= 0|1|2|3|4|5|6|7|8|9 \]

A syntax rule can be interpreted as follows: if \( \alpha \rightarrow \beta \) is a syntax rule and the string \( \varphi \xi \) is under consideration, then rewrite \( \varphi \xi \) as \( \varphi \beta \gamma \).

Denote the set of syntactic rules for a grammar as \( \mathcal{G} \).
Definition: \( \mathcal{V}_N \) is the nonterminal vocabulary defined by
\[
\mathcal{V}_N = \{ \mathcal{A} | \exists \alpha, \psi, \omega, \gamma : \alpha, \psi, \omega, \gamma \in \mathcal{V}^* \land \mathcal{A} \psi \rightarrow \phi \land \mathcal{A} \omega \rightarrow \phi \}
\]

Example:

**ALGOL**

all metalinguistic variables

\{s_{\text{tnn}}\} \quad \{s\}

as defined above

**Definition:** \( \mathcal{V}_T \) is the terminal vocabulary defined by \( \mathcal{V}_T = \mathcal{V} - \mathcal{V}_N \).

**Definition:** A Grammar (phrase structure syntax) \( G \) is the ordered quadruplet

\( G(\mathcal{V}, \phi, \mathcal{V}_T, S) \) where

- \( \mathcal{V} \) is the vocabulary
- \( \mathcal{V}_T \) is the terminal vocabulary
- \( \phi \) is the set of syntax rules
- \( S \) is the initial symbol \( (S \in \mathcal{V}_N) \).

The initial symbol \( S \in \mathcal{V}_N \) is chosen arbitrarily.

**Examples:**

1. \( \mathcal{V} = \{ S, a, b \} \)
   \( \mathcal{V}_T = \{a, b\} \)
   \( \phi = \{S \rightarrow aSb, S \rightarrow ab\} \)
   \( G_1(\mathcal{V}, \phi, \mathcal{V}_T, S) \).

2. \( \mathcal{V} = \{ S, A, a, b \} \)
   \( \mathcal{V}_T = \{a, b\} \)
   \( \phi = \{S \rightarrow aSb, S \rightarrow ab, Aa \rightarrow aa\} \)
   \( G_2(\mathcal{V}, \phi, \mathcal{V}_T, S) \).

**Definition:** A string \( \alpha \) directly generates a string \( \beta \) \((\alpha \rightarrow \beta)\) \iff

\[ \alpha = \gamma \delta e \quad \beta = \eta \in \mathcal{V}^* \land \gamma, \delta, \varepsilon, \eta \in \mathcal{V}^* \]

**Example:**

Using \( G_2 \)

\( AAASbbb \rightarrow AAAabbbb \)

**Definition:** A string \( \alpha \) generates a string \( \beta \) \((\alpha \# \beta)\) \iff

\[ \alpha = \alpha_0, \alpha_1 \ldots \alpha_n = \beta \in \mathcal{V}^* \]

\[ \alpha_{i-1} \rightarrow \alpha_i \quad (i=1, 2, \ldots n) \]
Example:

Using $G_2$

1. $S \rightarrow a^n b^n \ (n \geq 1)$
2. $a^{n-1} a b^n \rightarrow a^n b^n$

3. $<\text{program}> \rightarrow \text{Begin } x := 1 \text{ End}$

Definition: A string $\alpha$ is terminal if $\alpha = \Lambda$ or head $(\alpha) \in \mathcal{V}_T$ and tail $(\alpha)$ is terminal. Head and tail are the usual string operators.

Examples:

<table>
<thead>
<tr>
<th>Terminal strings</th>
<th>Nonterminal</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALCOL: \text{BEGIN } x := y \text{ END}</td>
<td>ALCOL: $&lt;\text{body}&gt;$ \text{ END}</td>
</tr>
<tr>
<td>[ G_2 : a^n b^n ]</td>
<td>[ A^{n-1} a b^n ]</td>
</tr>
</tbody>
</table>

Definition: A language $L$ (simple phrase structure language) defined by $G$ is

$$L(G) = \{ \alpha | S \rightarrow^{*} \alpha \land \alpha \text{ is terminal} \}$$

Example:

1. $L(G_1) = \{ \alpha | \alpha = a^n b^n \land n \geq 1 \}$
2. $L(G_2) = \{ \alpha | \alpha = a^n b^n \land n \geq 1 \}$
3. All ALCOL programs generable from the syntax as specified in the revised report, i.e., all error free ALCOL programs.

III. EQUIVALENCE OF GRAMMARS

Definition: Two grammars $G_1$ and $G_2$ are weakly equivalent if $L(G_1) = L(G_2)$.

Clearly $G_1 \equiv G_2$ in the above example.

IV. HIERARCHY OF GRAMMAR

Definition: A grammar $G(\mathcal{V}, \mathcal{F}, \mathcal{V}_T, S)$ is type 1 if every production of $G$ satisfies restriction 1.

Definition: A language is type 1 if it can be generated by a grammar of type 1. In terms of weak equivalence $L$ is type 1 if $L \equiv L(G_1)$ which is type 1 and $L \equiv L(G_1)$.

Restriction 0: (unrestricted rewriting system) A production satisfies restriction 0 if it is of the form $qA \rightarrow q\gamma$.

Theorem: If a language $L$ is generated by an unrestricted grammar $G$, then $L$ is also generated by a type 0 grammar $G_1$.

(Restriction 0 is not a restriction on the languages which may be formed, but on the structure of the languages.)
Restriction 1: (Context Sensitive) A production satisfies restriction 1 if it is of the form \( \phi A \gamma \to \phi \alpha \gamma \quad l(\alpha) \geq 1 \) (no erasure) 
\( (l \) is the length function on strings.)

Restriction 2: (Context Free) A production satisfies restriction 2 if it is of the form \( A \to \alpha \quad l(\alpha) \geq 1 \)

Restriction 3: (Finite State) A production satisfies restriction 3 if it is of the form \( A \to aB \) or \( A \to a \)

Examples:

1. **FS**
   \[ \mathcal{V}: S,X \]
   \[ \mathcal{V}_T: a,b \]
   \[ \{(ab)^n\} \]
   \[ \phi: S \to aX, X \to bS, X \to b \]

2. **CF**
   \[ \mathcal{V}: S \]
   \[ \mathcal{V}_T: a,b \]
   \[ \{a^n b^n\} \]
   \[ \phi: S \to aSb, S \to ab \]

3. **CS**
   \[ \mathcal{V}: S,U,X,E,A,B,C \]
   \[ \{a^n b^n c^n\} \]
   \[ \phi: S \to UXE \]
   \[ E \to CA \]
   \[ W \to AXE \]
   \[ C \to CB \]
   \[ X \to E \]
   \[ aA \to aA \]
   \[ ba \to ba \]
   \[ UA \to aA \]
   \[ UE \to aE \]
   \[ aA \to aa \]
   \[ aE \to aa \]
   \[ aB \to aB \]
   \[ aE \to aE \]
   \[ (generate loop) \]
   \[ (reverse symbols) \]
   \[ X \to B \]

Theorem: For both grammars and languages type 0 \( \geq \) type 1 \( \geq \) type 2 \( \geq \) type 3.

Theorem: There are type 1 languages which are not type 1 + 1 \( (i = 0,1,2) \).

Example:

1. \( \{a^n b^n a^n\} \) is CS but not CF
2. \( \{a^n b^n\} \) is CF but not FS
3. The set of all theorems in first order predicate calculus (RE) are type 0 but not type 1 (R).
V. EQUIVALENT RESTRICTIONS ON GRAMMARS (Note: \( \Lambda \notin L_n \))

Type 0
1. \( \phi \rightarrow \psi \quad \phi \notin \Lambda \)
2. \( \phi \Lambda \psi \rightarrow \phi \Lambda \psi \)

Type 1 Context Sensitive
1. \( \phi \Lambda \psi \rightarrow \phi \Lambda \psi \quad \ell(\alpha) \geq 1 \)
2. \( \phi \Lambda \psi \rightarrow \phi \Lambda \psi \quad \alpha \notin \Lambda \)
3. \( \phi \rightarrow \psi \quad \ell(\phi) \leq \ell(\psi) \quad \phi \notin \Lambda \)
4. \( \phi \Lambda \rightarrow \phi \Lambda \alpha \quad \ell(\alpha) \geq 1 \)
   \[ A \phi \rightarrow \alpha \phi \]

Type 2 Context Free
1. \( A \rightarrow \psi \quad \psi \notin \Lambda \)
2. \( A \rightarrow \psi \quad \ell(\psi) \geq 1 \)
3. \( A \rightarrow aB_1 \ldots B_n \)
   \[ A \rightarrow a \]
4. \( A \rightarrow aB_1B_2 \)
   \[ A \rightarrow aB_1 \]
   \[ A \rightarrow a \]
5. \( A \rightarrow B_1B_2 \)
   \[ A \rightarrow a \]

Type 3 Finite State
1. \( A \rightarrow aB \)
   \[ A \rightarrow a \]
2. \( A \rightarrow Ba \)
   \[ A \rightarrow a \]
3. \( A \rightarrow aB \)
   \[ A \rightarrow B \]
   \[ A \rightarrow a \]

Definition: A grammar is Left Context Sensitive if all productions are of the form \( \phi \Lambda \rightarrow \phi \Lambda \alpha \quad \ell(\alpha) \geq 1 \).

Theorem: There are LCS languages which are not CF.

Open Question: Is LCS an equivalent notation to CS?
VI. AUTOMATA, LANGUAGES AND COMPUTABILITY

Definition: A set of strings \( \Gamma \) is \( K \)-Decidable \iff there exists an automaton \( \alpha \) satisfying restriction \( K \) such that
\[
\alpha : \mathcal{V}^* \rightarrow \{0,1\} \text{ and } \Gamma = \{ \alpha \in \mathcal{V}^* \mid \alpha(\alpha) = 1 \}
\]

Definition: A set of strings \( \Gamma \) is \( K \)-Generable \iff there exists an automaton \( \alpha \) satisfying restriction \( K \) such that
\[
\alpha : \mathcal{V}^* \rightarrow \Gamma U \{ \Lambda \}
\]

Theorem: \( K \)-Decidable \( \subseteq \) \( K \)-Generable

Languages and Automata equivalent:

<table>
<thead>
<tr>
<th>Language</th>
<th>Strings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. type 0</td>
<td>Recursive Enumerable</td>
</tr>
<tr>
<td></td>
<td>Recursive Generable</td>
</tr>
<tr>
<td></td>
<td>(Turing Machine)</td>
</tr>
<tr>
<td>2. type 1</td>
<td>Recursive</td>
</tr>
<tr>
<td></td>
<td>Recursive Decidable</td>
</tr>
<tr>
<td>3. type 2</td>
<td>Push Down Store</td>
</tr>
<tr>
<td>4. type 3</td>
<td>Finite State</td>
</tr>
</tbody>
</table>

Theorem: Finite State Decidable = Finite State Generable

Push Down Store Decidable \( \subseteq \) Push Down Store Generable

Push Down Store Generable = Recursive Decidable

Recursive Decidable \( \subseteq \) Recursive Generable

VII. PARSING AND AMBIGUITIES

Definition: \( \beta \) directly reduces to \( \alpha \Rightarrow \alpha \rightarrow \beta \).

Definition: \( \beta \) reduces to \( \alpha \Rightarrow \alpha \rightarrow \beta \).

Definition: A parse of a string \( \alpha \) into the symbol \( A \) is a sequence of syntactic rules \( p_1, p_2, \ldots, p_n \in \phi \) such that \( p_j \) directly reduces \( \alpha_{j-1} \) into \( \alpha_j \) for \( j = 1, \ldots, n \) and \( \alpha = \alpha_0, \alpha_n = A \).

Examples: Parse of \( \alpha \) into \( S \).
\[
\alpha = \text{aaaa}, \mathcal{V} = \{\beta\}, \mathcal{V}_T = \{\alpha\}
\]

Examples:
1. \( \phi = \{ p_1 : S \rightarrow a, p_2 : S \rightarrow aS \} \)

Diagrams:
2. $\phi = \{ p_1 : S \to a, p_2 : S \to Sa \}$

3. $\phi = \{ p_1 : S \to A, p_2 : S \to aSa \}$

4. $\phi = \{ p_1 : S \to a, p_2 : S \to SS \}$

(one parse)
Definition: The Canonical form of the section of the parse reducing $\alpha_i$ into $\alpha_k$ where

$$\begin{align*}
\alpha_k &= A_1A_2 \cdots A_m \\
\alpha_i &= \beta_1\beta_2 \cdots \beta_m \\
A_\ell &\Rightarrow \beta_\ell \quad (\ell = 1 \cdots m)
\end{align*}$$

is $\{p_1\} \{p_2\} \cdots \{p_m\}$ where the sequence $\{p_\ell\}$ is the canonical form of the section reducing $\beta_\ell$ into $A_\ell$, and

1. $\{p_\ell\}$ is empty if $\beta_\ell = A_\ell$
2. $\{p_\ell\}$ is canonical if $A_\ell \Rightarrow \beta_\ell$.

Example: The canonical form of the parse in 4. with $\phi = \{p_1: S \rightarrow a, p_2: S \rightarrow SS\}$

$$\{\{\{p_1\}\}\{\{p_1\}\}\{p_1\}\}\{p_1\}\}$$

Definition: The canonical parse is a parse which proceeds strictly from left to right in a sentence and reduces a leftmost part of the sentence as far as possible* before proceeding further to the right.

Example: $$\{\{\{p_1\}\}\{\{p_1\}\}\{p_1\}\}\{p_1\}\}$$

Definition: A language $\mathcal{L}$ is ambiguous if for any terminal string $\alpha \in \mathcal{L}$, there exists more than one canonical parse.

Theorem: There is no algorithm which decides for an arbitrary syntax, whether it is unambiguous.

*without blocking

VIII. PROGRAMMING LANGUAGES

Definition: The ordered sextuple $\mathcal{L}_p (\mathcal{V}, \phi, \mathcal{V}_T, \mathcal{E}, \mathcal{V}, \mathcal{E})$ is the programming language defined by

1. $\mathcal{V}$ the vocabulary
2. $\phi$ the set of syntax rules
3. $\mathcal{V}_T$ terminal vocabulary
4. $S$ initial symbol
5. $\mathcal{V}$ semantic interpretation rules
6. $\mathcal{E}$ environment
Example:
\[ v : \{ \langle \text{program}, \langle \text{statement}, \langle \text{expression}, \langle \text{term}, \langle \text{factor}, a \rangle \rangle \rangle \rangle \} \]

\[ \mathcal{L}_T : \{ \} \quad S = \langle \text{program} \rangle \]

\[ \psi : \{ p_1 : \langle \text{program} \rangle := \langle \text{statement} \rangle, \]
\[ p_2 : \langle \text{statement} \rangle := a \leftarrow \langle \text{expression} \rangle, \]
\[ p_3 : \langle \text{expression} \rangle := \langle \text{expression} \rangle + \langle \text{term} \rangle, \]
\[ p_4 : \langle \text{expression} \rangle := \langle \text{term} \rangle, \]
\[ p_5 : \langle \text{term} \rangle := \langle \text{term} \rangle \times \langle \text{factor} \rangle, \]
\[ p_6 : \langle \text{term} \rangle := \langle \text{factor} \rangle, \]
\[ p_7 : \langle \text{factor} \rangle := a, \]
\[ p_8 : \langle \text{factor} \rangle := (\langle \text{expression} \rangle) \} \]

\[ w : \{ s_1 : \text{NOP} \]
\[ s_2 : \text{STO} \ a \]
\[ s_3 : \text{ADD} \]
\[ s_4 : \text{NOP} \]
\[ s_5 : \text{NEG} \]
\[ s_6 : \text{NEG} \]
\[ s_7 : \text{STK} \ a \]
\[ s_8 : \text{NOP} \} \]

\[ \mathcal{E} : \text{Memory of stack computer} \]

**Definition:** The meaning of a sentence \( a \) in a language \( \mathcal{L}_p \) is \( S_1 S_2 \ldots S_m \) where a parse of \( a \) into \( S \) is \( p_1 p_2 \ldots p_m \) and \( S_1 \) corresponds to \( p_i (i = 1 \ldots m) \).

**Example:**
\[ a + a + a \]
\[ \{\{\{\{p_7\}g\}4\}\{p_7\}i\}3\{2\}1\} \quad \text{(canonical form)} \]
\[ p_7 \ p_6 \ p_4 \ p_7 \ p_6 \ p_3 \ p_2 \ p_2 \quad \text{(parse)} \]
\[ S_7 \ S_6 \ S_4 \ S_7 \ S_6 \ S_3 \ S_2 \ S_1 \quad \text{(meaning)} \]

\[ \text{NOP} \ \text{NOP} \ \text{STK} \ a \quad \text{STK} \ a \ \text{ADD} \ \text{STO} \ a \]

- 10 -
IX. PRECEDENCE RELATIONS AND PRECEDENCE GRAMMARS

Definition: The left set $L(A)$ of $A \in \mathcal{N}$ is

$$L(A) = \{ S \mid \exists \alpha \in (A, S \alpha) \},$$

$$R(A) = \{ S \mid \exists \alpha \in (A, \alpha S) \}.$$

Definition: For $S_i$ and $S_j \in \mathcal{N}$

$$S_i = S_j \iff \exists \alpha \in \mathcal{N} \text{ p: } U \rightarrow S_i S_j \beta$$

$$S_i < S_j \iff \exists \alpha \in \mathcal{N} \text{ p: } U \rightarrow S_i U \beta \text{ } S_j \in L(U')$$

$$S_i > S_j \iff \exists \alpha \in \mathcal{N}$$

$$(p: U \rightarrow S_i U \beta) \text{ } S_j \in R(U) \text{ and } U \rightarrow S_i U \beta \text{ } S_j \in L(U')$$

Definition: A grammar $G$ is a simple precedence grammar $\iff$ at most one relation holds between ordered pairs of symbols of $\mathcal{N}$.

Definition: A language is a simple precedence phrase structure programming language $\iff$ it is defined by a precedence grammar.

Theorem: There exists a simple algorithm for simple precedence phrase structure programming languages which obtains the unique canonical parse of a string in the language.
Bibliography


