E. A. Burfine
June 1966
C.G.T.M. #9

~~MASTER COPY~~

~~do not remove from file~~

AN OUTLINE OF THE BASIC THEORY OF FORMAL LANGUAGES

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)

<u>Definition</u>: 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.

   A, <u>Integer</u>, <identifier>

2. $\{\alpha | \alpha = a^n b^n, n \geq 1\}$ S, a, b

   $S \rightarrow aSb | ab$

<u>Definition</u>: $\mathcal{V}^*$ is the set of all strings generated by concatenation of the elements of $\mathcal{V}$. (This includes the null string Λ.)

<u>Nomenclature</u>: Denote all members of $\mathcal{V}$ by either upper or lower case latin letters and strings in $\mathcal{V}^*$ by lower case greek letters.

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

Examples:

1. Backus Normal Form

   <bound pair list>::= <bound pair>

   <bound pair list>::= <bound pair list>,<bound pair>

   abbreviation for the above two rules

   <bound pair list>::= <bound pair>|
   <bound pair list>,<bound pair>

2. $A \rightarrow aaA$

3. $ABA \rightarrow A\alpha A$

4. <number>::= <digit> <number>|<digit>

   <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 $\varphi\alpha\psi$ is under consideration, then rewrite $\varphi\alpha\psi$ as $\varphi\beta\psi$.

Denote the set of syntactic rules for a grammar as Φ.

Definition: ϑ_N is the nonterminal vocabulary defined by

$$\vartheta_N = \{A \mid \exists \varphi, \psi, \alpha, \alpha \in \vartheta^* \wedge \varphi\ A\ \psi \rightarrow \varphi\ \alpha\ \psi \in \Phi\}$$

Example:

ALGOL	all metalinguistic variables
$\{a^n b^n\}$	$\{S\}$
as defined above	

Definition: ϑ_T is the terminal vocabulary defined by $\vartheta_T = \vartheta - \vartheta_N$.

Definition: A Grammar (phrase structure syntax) G is the ordered quadruplet $G(\vartheta, \Phi, \vartheta_T, S)$ where

ϑ is the vocabulary

ϑ_T is the terminal vocabulary

Φ is the set of syntax rules

S is the initial symbol ($S \in \vartheta_N$).

The initial symbol $S \in \vartheta_N$ is chosen arbitrarily.

Examples:

1. $\vartheta = \{S, a, b\}$
   $\vartheta_T = \{a, b\}$
   $\Phi = \{S \rightarrow aSb,\ S \rightarrow ab\}$
   $G_1(\vartheta, \Phi, \vartheta_T, S)$.
2. $\vartheta = \{S, A, a, b\}$
   $\vartheta_T = \{a, b\}$
   $\Phi = \{S \rightarrow ASb,\ S \rightarrow ab,\ Aa \rightarrow aa\}$
   $G_2(\vartheta, \Phi, \vartheta_T, S)$

Definition: A string α directly generates a string β ($\alpha \dot{\rightarrow} \beta$) $\leftrightarrow$

$\alpha = \gamma\delta\epsilon \quad \beta = \gamma\eta\epsilon \quad \exists \gamma, \epsilon \quad \delta \rightarrow \eta \in \Phi \quad \alpha, \beta, \gamma, \delta, \epsilon, \eta \in \vartheta^*$

Example:

Using G_2

$AAASbbb \dot{\rightarrow} AAAabbbb$

Definition: A string α generates a string β ($\alpha \overset{*}{\rightarrow} \beta$) $\leftrightarrow$

$\alpha = \alpha_0, \alpha_1 \ldots \alpha_n = \beta \quad \in \vartheta^*$

$\alpha_{i-1} \dot{\rightarrow} \alpha_i \quad (i=1, 2, \ldots n)$

Example:

Using G_2

1. $S \overset{*}{\Rightarrow} a^n b^n \ (n \geq 1)$
2. $A^{n-1}ab^n \overset{*}{\Rightarrow} a^n b^n$
3. <program> $\overset{*}{\Rightarrow}$ <u>Begin</u> X : = 1 <u>End</u>

<u>Definition</u>: A string α is terminal $\leftrightarrow \alpha = \Lambda$ or head $(\alpha) \in \vartheta_T$ and tail (α) is terminal. Head and tail are the usual string operators.

Examples:

	terminal strings	nonterminal
ALGOL:	<u>BEGIN</u> x:=y <u>END</u>	<u>BEGIN</u> <body> <u>END</u>
G_2 :	$a^n b^n$	$A^{n-1}ab^n$

<u>Definition</u>: A language $\mathcal{L}$ (simple phrase structure language) defined by G is

$$\mathcal{L}(G) = \{\alpha \mid S \overset{*}{\Rightarrow} \alpha \wedge \alpha \text{ is terminal}\}$$

Example:

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

III. EQUIVALENCE OF GRAMMARS

<u>Definition</u>: Two grammars G_1 and G_2 are weakly equivalent $\leftrightarrow \mathcal{L}(G_1) = \mathcal{L}(G_2)$. Clearly $G_1 \overset{W}{=} G_2$ in the above example.

IV. HIERARCHY OF GRAMMAR

<u>Definition</u>: A grammar $G(\vartheta, \Phi, \vartheta_T, S)$ is type i if every production of G satisfies restriction i.

<u>Definition</u>: A language is type i if it can be generated by a grammar of type i. In terms of weak equivalence

$$\mathcal{L} \text{ is type } i \leftrightarrow \exists G_1 \text{ which is type } i \wedge \mathcal{L} \overset{W}{=} \mathcal{L}_i(G_1)$$

<u>Restriction 0</u>: (unrestricted rewriting system) A production satisfies restriction 0 if it is of the form $\varphi A \psi \rightarrow \varphi \alpha \psi$

<u>Theorem</u>: If a language $\mathcal{L}$ is generated by an unrestricted grammar G, then $\mathcal{L}$ 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 $\varphi A \psi \rightarrow \varphi \alpha \psi$ $\quad \ell(\alpha) \geq 1$ (no erasure)

(ℓ is the length function on strings.)

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

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

Examples:

1. FS $\{(ab)^n\}$ $\quad \vartheta$: S,X $\quad \vartheta_T$: a,b

   Φ : $S \rightarrow aX$, $X \rightarrow bS$, $X \rightarrow b$

2. CF $\{a^n b^n\}$ $\quad \vartheta$: S $\quad \vartheta_T$: a,b

   Φ : $S \rightarrow aSb$, $S \rightarrow ab$

3. CS $\{a^n b^n a^n\}$ $\quad \vartheta$: S,U,X,E,A,B,C

   Φ :

$S \rightarrow UXE$	$BA \rightarrow CA$
$W \rightarrow AXE$	$CA \rightarrow CB$
$X \rightarrow BW$	$CB \rightarrow AB$
$X \rightarrow B$	
(generate loop)	(reverse symbols)
$BE \rightarrow Ba$	$Bb \rightarrow bb$
$Ba \rightarrow ba$	$aA \rightarrow aa$
$UA \rightarrow aA$	$UBE \rightarrow aBE$
$aA \rightarrow aa$	$aBE \rightarrow aBa$
$aE \rightarrow aa$	$aBa \rightarrow aba$
(start terminals)	(propagate terminals)

Theorem: For both grammars and languages type 0 $\supseteq$ type 1 $\supseteq$ type 2 $\supseteq$ type 3.

Theorem: There are type i languages which are not type i + 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) is type 0 but not type 1 (R).

V. EQUIVALENT RESTRICTIONS ON GRAMMARS (Note: $\Lambda \notin \mathcal{V}_N$)

Type 0

1. $\varphi \to \psi \qquad \varphi \neq \Lambda$
2. $\varphi A \psi \to \varphi \alpha \psi$

Type 1 Context Sensitive

1. $\varphi A \psi \to \varphi \alpha \psi \quad \ell(\alpha) \geq 1$
2. $\varphi A \psi \to \varphi \alpha \psi \quad \alpha \neq \Lambda$
3. $\varphi \to \psi \quad \ell(\varphi) \leq \ell(\psi) \quad \varphi \neq \Lambda$
4. $\varphi A \to \varphi \alpha$
   $A \varphi \to \alpha \varphi \quad \ell(\alpha) \geq 1$

Type 2 Context Free

1. $A \to \psi \qquad \psi \neq \Lambda$
2. $A \to \psi \qquad \ell(\psi) \geq 1$
3. $A \to aB_1 \ldots B_n$
   $A \to a$
4. $A \to aB_1B_2$
   $A \to aB_1$
   $A \to a$
5. $A \to B_1B_2$
   $A \to a$

Type 3 Finite State

1. $A \to aB$
   $A \to a$
2. $A \to Ba$
   $A \to a$
3. $A \to aB$
   $A \to B$
   $A \to a$

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

Theorem: There are LCS languages which are not CF.

Open Question: Is LCS an equivalent notation to CS?

VI. AUTOMATA, LANGUAGES AND COMPUTABILITY

<u>Definition</u>: A set of strings Γ is <u>K-Decidable</u> $\leftrightarrow$ there exists an automaton $\mathfrak{A}$ satisfying restriction K such that

$$\mathfrak{A} : \vartheta^* \to \{0,1\} \text{ and}$$
$$\Gamma = \{\alpha \in \vartheta^* \mid \mathfrak{A}(\alpha) = 1\}$$

<u>Definition</u>: A set of strings Γ is K-Generable $\leftrightarrow$ there exists an automaton $\mathfrak{A}$ satisfying restriction K such that

$$\mathfrak{A} : \vartheta^* \to \Gamma \cup \{\Lambda\}$$

<u>Theorem</u>: K-Decidable $\subseteq$ K-Generable

Languages and Automata equivalent

	<u>language</u>	<u>strings</u>
1.	type 0	Recursive Enumerable Recursive Generable (Turing Machine)
2.	type 1	Recursive Recursive Decidable
3.	type 2	Push Down Store
4.	type 3	Finite State

<u>Theorem</u>: Finite State Decidable = Finite State Generable

Push Down Store Decidable $\subset$ Push Down Store Generable

Push Down Store Generable = Recursive Decidable

Recursive Decidable $\subset$ Recursive Generable

VII. PARSING AND AMBIGUITIES

<u>Definition</u>: β <u>directly reduces</u> to $\alpha \leftrightarrow \alpha \xrightarrow{\cdot} \beta$.

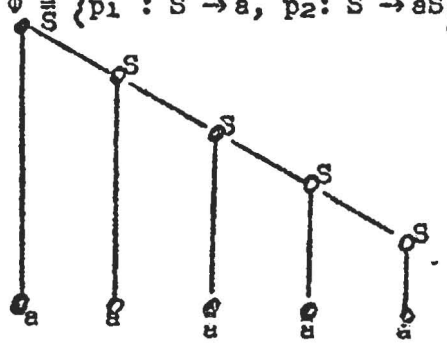
<u>Definition</u>: β <u>reduces</u> to $\alpha \leftrightarrow \alpha \xrightarrow{*} \beta$.

<u>Definition</u>: A <u>parse</u> of a string α 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 α_{j-1} into α_j $j = 1 \ldots n$ and $\alpha = \alpha_0$, $\alpha_n = A$.

Examples: Parse of α into S.

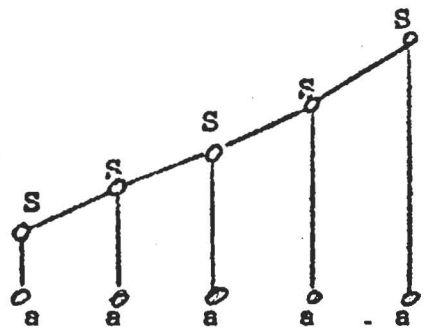
α = aaaaa, $\vartheta = \{S\}$, $\vartheta_T = \{a\}$

1. $\Phi \equiv \{p_1 : S \to a,\ p_2 : S \to aS\}$



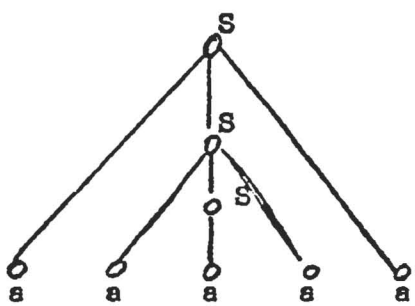
p_1, p_2, p_2, p_2, p_2

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



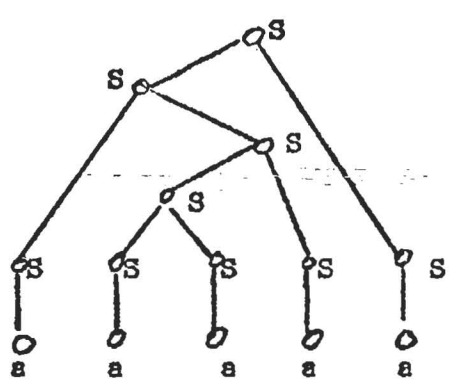
p1,p2,p2,p2,p2

3. $\Phi = \{p_1 : S \to a,\ p_2 : S \to aSa\}$



p1,p2,p2

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



p1,p1,p1,p2,p1,p2,p2,p1,p2
(one parse)

Definition: The Canonical form of the section of the parse reducing α_i into α_k where

$$\alpha_k = A_1 A_2 \ldots A_m$$

$$\alpha_i = \beta_1 \beta_2 \ldots \beta_m$$

$$A_\ell \overset{*}{\underset{p}{\rightarrow}} \beta_\ell \qquad (\ell = 1 \ldots m)$$

is $\{p_1\}\ \{p_2\} \ldots \{p_m\}$ where the sequence $\{p_\ell\}$ is the canonical form of the section reducing β_ℓ into A_ℓ, and

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

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

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

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:

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

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 $\mathcal{L}_p\ (\mathcal{V}, \Phi, \mathcal{V}_T, S, \Psi, \mathcal{E})$ is the programming language defined by

1. $\mathcal{V}$ the vocabulary
2. Φ the set of syntax rules
3. $\mathcal{V}_T$ terminal vocabulary
4. S initial symbol
5. Ψ semantic interpretation rules
6. $\mathcal{E}$ environment

Example:

$\mathcal{V}$: {<program>,<statement>,<expression>,<term>,<factor>, a}

$\mathcal{V}_T$: {a} S = <program>

Φ : {p1: <program>::= <statement>,

p2: <statement>::= a ← <expression>,

p3: <expression>::= <expression> + <term>,

p4: <expression>::= <term>,

p5: <term>::= <term>× <factor>,

p6: <term>::= <factor>,

p7: <factor>::= a,

p8: <factor>::= (<expression>)}.

Ψ : {S_1: NOP

S_2: STO a

S_3: ADD

S_4: NOP

S_5: MUL

S_6: NOP

S_7: STK a

S_8: NOP }.

$\mathcal{E}$: Memory of stack computer

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

Example: a ← a + a

{{{{{{{p7}6}4}{{{p7}6}5}2}1} (canonical form)

p7 p6 p4 p7 p6 p5 p2 p1 (parse)

S_7 S_6 S_4 S_7 S_6 S_5 S_2 S_1 (meaning)

NOP NOP NOP NOP

STK a STK a ADD STO a

IX. PRECEDENCE RELATIONS AND PRECEDENCE GRAMMARS

<u>Definition</u>: The <u>Left set</u> L(<u>Right set</u> R) of $A \in \mathcal{V}_N$ is

$$L(A) = \{S \mid \exists\, \alpha (A \overset{*}{\Rightarrow} S\alpha)\},$$
$$R(A) = \{S \mid \exists\, \alpha (A \overset{*}{\Rightarrow} \alpha S)\}.$$

<u>Definition</u>: For S_i and $S_j \in \mathcal{V}$

$$S_i \doteq S_j \leftrightarrow \exists_p \in \Phi \ni \; p: U \rightarrow \alpha\, S_i S_j \beta$$

$$S_i < S_j \leftrightarrow \exists_p \in \Phi \ni \; p: U \rightarrow \alpha\, S_i U_\ell \beta \;\; S_j \in L(U_\ell)$$

$$S_i > S_j \leftrightarrow \exists_p \in \Phi \ni$$

$$(p: U \rightarrow \alpha U_r S_j \beta \wedge S_i \in R\,(U_r))$$

$$\vee (p: U \rightarrow \alpha U_r U_\ell \beta \wedge S_i \in R\,(U_r) \wedge S_j \in L(U_\ell))$$

<u>Definition</u>: A grammar G is a simple precedence grammar $\leftrightarrow$ at most one relation holds between ordered pairs of symbols of $\mathcal{V}_N$.

<u>Definition</u>: A language is a <u>simple precedence phrase structure programming language</u> $\leftrightarrow$ it is defined by a precedence grammar.

<u>Theorem</u>: 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

1. Bach, Emmon, An Introduction to Transformational Grammars, Holt, Rinehart and Winston, Inc., San Francisco, 1964.
2. Braffort, P. (Ed.), Computer Programming and Formal Systems, North Holland Publishing Co., Amsterdam, 1963.
3. Chomsky, N., Syntactic Structures, Mouton and Co., The Hague, 1965.
4. Chomsky, N., On Certain Formal Properties of Grammars, Information and Control, Vol. 2, 137-167, 1959.
5. Davis, Martin, Computability and Unsolvability, McGraw-Hill, New York, 1958.
6. Glantz, Richard S., Left Context Sensitive Languages, Draft.
7. Wirth, Niklaus and Weber, Helmut, Euler: A Generalization of ALGOL and its Formal Definition, Technical Report CS20, Computer Science Department, Stanford University, April 27, 1965.