11. Hierarchy of LA-grammar

11.1 Generative capacity of unrestricted LAG

11.1.1 Generative capacity of unrestricted LA-grammar

Unrestricted LA-grammar accepts and generates all and only the recursive languages.

11.1.2 Theorem 1

Unrestricted LA-grammar accepts and generates *only* the recursive languages.

Proof: Assume an input string of finite length n. Each word in the input string has a finite number of readings (> 0).

Combination step 1: The finite set of start states ST_S and all readings of the first word w_1 result in a finite set of well-formed expressions WE_1 = {(ss' rp_S) | ss' ϵ ($\mathrm{W}^+ \times \mathrm{C}^+$)}.

Combination step n: Combination step k-1, $\mathrm{k} > 1$, has produced a finite set of well-formed expressions WE_k = {(ss' rp_i) | i ϵ n, ss' ϵ ($\mathrm{W}^+ \times \mathrm{C}^*$) and the surface of each ss' has length k}. The next word w_{k+1} has a finite number of readings.

Therefore, the Cartesian product of all elements of WE_k and all readings of the current next word will be a finite set of pairs. Each pair is associated with a rule package containing a finite set of rules. Therefore, combination step k will produce only finitely many new sentence starts. The derivation of this finite set of new sentence starts is decidable because the categorial operations are defined to be total recursive functions.

Q.E.D.

11.1.3 Theorem 2

Unrestricted LA-grammar accepts and generates *all* recursive languages.

Proof: Let L be a recursive language with the alphabet W. Because L is recursive, there is a total recursive function ϱ: $W^* \rightarrow \{0,1\}$, i.e., the characteristic function of L. Let LAG^L be an LA-grammar defined as follows:

The set of word surfaces of LAG^L is W.

The set of category segments $C =_{def} W \cup \{0,1\}$.

For arbitrary e, f ϵ W^+, [e (f)] ϵ LX if and only if e = f.

LX $=_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)], . . . }
$ST_S =_{def}$ {[(seg_c) {r_1, r_2}]}, where seg_c ϵ {a, b, c, d, . . . }
r_1: (X) (seg_c) $\Rightarrow$ (X seg_c) {r_1, r_2}
r_2: (X) (seg_c) $\Rightarrow$ ϱ (X seg_c) { }
$ST_F =_{def}$ {[(1) rp_2]}

After any given combination step, the rule package rp_1 offers two choices: application of r_1 to continue reading the input string, or application of r_2 to test whether the input read so far is a well-formed expression of L. In the latter case, the function ϱ is applied to the concatenation of the input categories, which are identical to the input surfaces. If the result of applying r_2 is [(1) rp_2], the input surface is accepted; if it is [(0) rp_2], it is rejected.

Since the categorial operations of LAG^L can be any total recursive function, LAG^L may be based on ϱ, the characteristic function of L. Therefore, LAG^L accepts and generates any recursive language.

Q.E.D.

11.1.4 Definition of the class of A-LAGs.

The class of A-LAGs consists of unrestricted LA-grammars and generates *all* recursive languages.

11.2 LA-hierarchy of A-, B-, and C-LAGs

11.2.1 Parameters of complexity

- The *amount* of computation per rule application required in the worst case.
- The *number* of rule applications relative to the length of the input needed in the worst case.

11.2.2 Main approaches to restricting LA-grammar

R1: Restrictions on the form of categorial operations in order to limit the maximal amount of computation required by arbitrary rule applications.
R2: Restrictions on the degree of ambiguity in order to limit the maximal number of possible rule applications.

11.2.3 Possible restrictions on categorial operations

R1.1: Specifying upper bounds for the *length* of categories;
R1.2: Specifying restrictions on *patterns* used in the definition of categorial operations.

11.2.4 Definition of the class of B-LAGs.

The class of *bounded* LA-grammars, or B-LAGs, consists of grammars where for any complete well-formed expression E the length of intermediate sentence start categories is bounded by $k \cdot n$, where n is the length of E and k is a constant.

11.2.5 Rule schemata with constant categorial operations

r$_i$: (seg$_1$...seg$_k$ X) cat$_2$ $\Rightarrow$ cat$_3$ rp$_i$
r$_i$: (X seg$_1$...seg$_k$) cat$_2$ $\Rightarrow$ cat$_3$ rp$_i$
r$_i$: (seg$_1$...seg$_m$ X seg$_{m+1}$...seg$_k$) cat$_2$ $\Rightarrow$ cat$_3$ rp$_i$

11.2.6 Rule schema with nonconstant categorial operation

r$_i$: (X seg$_1$...seg$_k$ Y) cat$_2$ $\Rightarrow$ cat$_3$ rp$_i$

11.2.7 Definition of the class of C-LAGs.

The class of *constant* LA-grammars, or C-LAGs, consists of grammars in which no categorial operation co$_i$ looks at more than k segments in the sentence start categories, for a finite constant k.

11.2.8 The hierarchy of A-LAGs, B-LAGs, and C-LAGs

The class of A-LAGs accepts and generates all recursive languages, the class of B-LAGs accepts and generates all context-sensitive languages, and the class of C-LAGs accepts and generates many context-sensitive, all context-free, and all regular languages.

11.3 Ambiguity in LA-grammar

11.3.1 Factors determining the number of rule applications

The number of rule application in an LA-derivation depends on

1. the length of the input;
2. the number of rules in the rule package to be applied in a certain combination to the analyzed input pair;
3. the number of readings existing at each combination step.

11.3.2 Impact on complexity

- Factor 1 is grammar-independent and used as the length n in formulas characterizing complexity .
- Factor 2 is a grammar-dependent constant.
- Only factor 3 may push the total number of rule applications beyond a linear increase. Whether for a given input more than one rule in a rule package may be successful depends on the input conditions of the rules.

11.3.3 Regarding factor 3: Possible relations between the input conditions of two rules

1. *Incompatible* input conditions: if there exist no input pairs which are accepted by both rules.

Examples:	(a X) (b)	(a X) (b)
	(c X) (b)	(a X) (c)

2. *Compatible* input conditions: if there exists at least one input pair accepted by both rules and there exists at least one input pair accepted by one rule, but not the other.

Examples:	(a X) (b)
	(X a) (b)

3. *Identical* input conditions: if all input pairs are either accepted by both rules or rejected by both rules.

11.3.4 Definition of unambiguous LA-grammars

An LA-grammar is unambiguous if and only if (i) it holds for all rule packages that their rules have *incompatible* input conditions and (ii) there are no lexical ambiguities.

11.3.5 Definition of syntactically ambiguous LA-grammars

An LA-grammar is syntactically ambiguous if and only if (i) it has at least one rule package containing at least two rules with *compatible* input conditions and (ii) there are no lexical ambiguities.

11.3.6 +global syntactic ambiguity

A syntactic ambiguity is called +global if it is a property of the whole sentence.

Example: Flying air planes can be dangerous.

11.3.7 –global syntactic ambiguity

A syntactic ambiguity is called -global if it is a property of only part of the sentence.

Example: The horse raced by the barn fell.

11.3.8 Role of the ±global distinction

In LA-grammar, the difference between +global and –global ambiguities consists in whether more than one reading survives to the end of the sentence (example 11.3.6) or not (example 11.3.7). The ±global distinction has no impact on complexity in LA-grammar and is made mainly for linguistic reasons.

11.3.9 +recursive syntactic ambiguity

An ambiguity is +recursive, if it originates within a recursive loop of rule applications.

Examples: the C-LAGs for WW^R (cf. 11.5.6) and WW (cf. 11.5.8), which are –global, and for SubsetSum (cf. 11.5.10), which are +global.

11.3.10 -recursive syntactic ambiguity

An ambiguity is –recursive, if none of the branches produced in the ambiguity split returns to the state which caused the ambiguity.

Examples: the C-LAG for $a^kb^kc^md^m \cup a^kb^mc^md^k$ (cf. 11.5.3), which is +global, and the C-LAGs for natural language in Chapter 17 and 18, which exhibit both +global and –global ambiguities.

11.3.11 Role of the ±recursive distinction

The ±recursive distinction is crucial for the analysis of complexity because it can be shown that in LA-grammars with nonrecursive ambiguities the maximal number of rule applications per combination step is limited by a grammar-dependent constant.

11.3.12 Theorem 3

The maximal number of rule applications in LA-grammar with only –recursive ambiguities is

$$(n - (R - 2)) \cdot 2^{(R-2)}$$

for $n >$ (R - 2), where n is the length of the input and R is the number of rules in the grammar.

Proof: Parsing an input of length n requires (n – 1) combination steps. If an LA-grammar has R rules, then one of these rules has to be reapplied after R combination steps at the latest. Furthermore, the maximal number of rule applications in a combination step for a given reading is R.

According to the definition of –recursive ambiguity, rules causing a syntactic ambiguity may not be reapplied in a time-linear derivation path (reading). The first ambiguity-causing rule may produce a maximum of R-1 new branches (assuming its rule package contains all R rules except for itself), the second ambiguity causing rule may produce a maximum of R – 2 new branches, etc. If the different rules of the LA-grammar are defined with their maximally possible rule packages, then after R – 2 combination steps a maximum of $2^{(R-2)}$ readings is reached.

Q.E.D.

11.4 Complexity of grammars and automata

11.4.1 Choosing the primitive operation

The Griffith and Petrick data is not in terms of actual time, but in terms of "primitive operations." They have expressed their algorithms as sets of nondeterministic rewriting rules for a Turing-machine-like device. Each application of one of these is a primitive operation. We have chosen as our primitive operation the act of adding a state to a state set (or attempting to add one which is already there). We feel that this is comparable to their primitive operation because both are in some sense the most complex operation performed by the algorithm whose complexity is independent of the size of the grammar and the input string.

J. Earley 1970, p. 100

11.4.2 Primitive operation of the C-LAGs

The primitive operation of C-LAGs is a rule application (also counting unsuccessful attempts).

11.5 Subhierarchy of C1-, C2-, and C3-LAGs

11.5.1 The subclass of C1-LAGs

A C-LAG is a C1-LAG if it is not recursively ambiguous. The class of C1-languages parses in linear time and contains all deterministic context-free languages which can be recognized by a DPDA without ε-moves, plus context-free languages with –recursive ambiguities, e.g. $a^k b^k c^m d^m \cup a^k b^m c^m d^k$, as well as many context-sensitive languages, e.g. $a^k b^k c^k$, $a^k b^k c^k d^k e^k$, $\{a^k b^k c^k\}^*$, L_{square}, L^k_{hast}, a^{2^i}, $a^k b^m c^{k \cdot m}$, and $a^{i!}$, whereby the last one is not even an index language.

11.5.2 C1-LAG for context-sensitive a^{2^i}

LX $=_{def}$ {[a (a)]}
ST$_S$ $=_{def}$ {[(a) {r$_1$}]}

r$_1$:	(a)	(a)	⇒ (aa)	{r$_2$}
r$_2$:	(aX)	(a)	⇒ (Xbb)	{r$_2$, r$_3$}
r$_3$:	(bX)	(a)	⇒ (Xaa)	{r$_2$, r$_3$}

ST$_F$ $=_{def}$ {[(aa) rp$_1$], [(bXb) rp$_2$], [(aXa) rp$_3$]}.

11.5.3 C1-LAG for ambiguous $a^k b^k c^m d^m \cup a^k b^m c^m d^k$

LX $=_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)]}
ST$_S$ $=_{def}$ {[(a) {r$_1$, r$_2$, r$_5$}]}

r$_1$: (X)	(a)	$\Rightarrow$ (a X)	{r$_1$, r$_2$, r$_5$}
r$_2$: (a X)	(b)	$\Rightarrow$ (X)	{r$_2$, r$_3$}
r$_3$: (X)	(c)	$\Rightarrow$ (c X)	{r$_3$, r$_4$}
r$_4$: (c X)	(d)	$\Rightarrow$ (X)	{r$_4$}
r$_5$: (X)	(b)	$\Rightarrow$ (b X)	{r$_5$, r$_6$}
r$_6$: (b X)	(c)	$\Rightarrow$ (X)	{r$_6$, r$_7$}
r$_7$: (a X)	(d)	$\Rightarrow$ (X)	{r$_7$}

ST$_F$ $=_{def}$ {[ε rp$_4$], [ε rp$_7$]}

11.5.4 The Single Return Principle (SRP)

A +recursive ambiguity is single return, if exactly one of the parallel paths returns into the state resulting in the ambiguity in question.

11.5.5 The subclass of C2-LAGs

A C-LAG is a C2-LAG if it is SR-recursively ambiguous. The class of C2-languages parses in polynomial time and contains certain nondeterministic context-free languages like WW^{R} and $\mathsf{L}^{\infty}_{hast}$, plus context-sensitive languages like WW, $\mathsf{W}^{\mathrm{k}\geq 3}$, $\{\mathsf{WWW}\}^{*}$, and $\mathsf{W}_1\mathsf{W}_2\mathsf{W}_1^{\mathrm{R}}\mathsf{W}_2^{\mathrm{R}}$.

11.5.6 C2-LAG for context-free WW^R

LX $=_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)] . . . }
ST$_S$ $=_{def}$ {[(seg$_c$) {r$_1$, r$_2$}]}, where seg$_c$ ϵ {a, b, c, d, . . . }
r$_1$: (X) (seg$_c$) $\Rightarrow$ (seg$_c$ X) {r$_1$, r$_2$}
r$_2$: (seg$_c$ X) (seg$_c$) $\Rightarrow$ (X) {r$_2$}
ST$_F$ $=_{def}$ {[ε rp$_2$]}

11.5.7 Derivation structure of the worst case in WW^R

```
rules:              analyses:

  2                   a  $  a
  1  2  2             a  a  $  a  a
  1  1  2  2  2       a  a  a  $  a  a  a
  1  1  1  2  2       a  a  a  a  $  a  a
  1  1  1  1  2       a  a  a  a  a  $  a
  1  1  1  1  1       a  a  a  a  a  a  $
```

11.5.8 C2-LAG for context-sensitive WW

LX $=_{def}$ {[a (a)], [b (b)], [c (c)], [d (d)] . . . }

ST$_S$ $=_{def}$ {[(seg$_c$) {r$_1$, r$_2$}]}, where seg$_c$ ϵ {a, b, c, d, . . . }

r$_1$:	(X)	(seg$_c$)	$\Rightarrow$	(X seg$_c$)	{r$_1$, r$_2$}
r$_2$:	(seg$_c$ X)	(seg$_c$)	$\Rightarrow$	(X)	{r$_2$}

ST$_F$ $=_{def}$ {[ε rp$_2$]}

11.5.9 C2-LAG for context-sensitive $W_1W_2W_1^RW_2^R$

LX $=_{def}$ {[a (a)], [b (b)]}

ST$_S$ $=_{def}$ {[(seg$_c$) {r$_{1a}$}], [(seg$_c$) {r$_{1b}$}]}, where seg$_c$, seg$_d$ ϵ {a, b}

r$_{1a}$:	(seg$_c$)	(seg$_d$)	$\Rightarrow$	(# seg$_c$ seg$_d$)	{r$_2$, r$_3$}
r$_{1b}$:	(seg$_c$)	(seg$_d$)	$\Rightarrow$	(seg$_d$ # seg$_c$)	{r$_3$, r$_4$}
r$_2$:	(X)	(seg$_c$)	$\Rightarrow$	(X seg$_c$)	{r$_2$, r$_3$}
r$_3$:	(X)	(seg$_c$)	$\Rightarrow$	(seg$_c$ X)	{r$_3$, r$_4$}
r$_4$:	(X seg$_c$)	(seg$_c$)	$\Rightarrow$	(X)	{r$_4$, r$_5$}
r$_5$:	(seg$_c$ X #)	(seg$_c$)	$\Rightarrow$	(X)	{r$_6$}
r$_6$:	(seg$_c$ X)	(seg$_c$)	$\Rightarrow$	(X)	{r$_6$}

ST$_F$ $=_{def}$ {[ε rp$_5$], [ε rp$_6$]}

11.5.10 C3-LAG for SubsetSum.

LX $=_{def}$ {[0 (0)], [1 (1)], [# (#)]}

ST$_S$ $=_{def}$ {[(seg$_c$) {r$_1$, r$_2$}]}, where seg$_c$ ϵ {0, 1}

seg$_c$ ε {0, 1}

r$_1$:	(X)	(seg$_c$)	$\Rightarrow$	(seg$_c$ X)	{r$_1$, r$_2$}
r$_2$:	(X)	(#)	$\Rightarrow$	(# X)	{r$_3$, r$_4$, r$_6$, r$_7$, r$_{12}$, r$_{14}$}
r$_3$:	(X seg$_c$)	(seg$_c$)	$\Rightarrow$	(0 X)	{r$_3$, r$_4$, r$_6$, r$_7$}
r$_4$:	(X #)	(#)	$\Rightarrow$	(# X)	{r$_3$, r$_4$, r$_6$, r$_7$, r$_{12}$, r$_{14}$}
r$_5$:	(X seg$_c$)	(seg$_c$)	$\Rightarrow$	(0 X)	{r$_5$, r$_6$, r$_7$, r$_{11}$}
r$_6$:	(X 1)	(0)	$\Rightarrow$	(1 X)	{r$_5$, r$_6$, r$_7$, r$_{11}$}
r$_7$:	(X 0)	(1)	$\Rightarrow$	(1 X)	{r$_8$, r$_9$, r$_{10}$}
r$_8$:	(X seg$_c$)	(seg$_c$)	$\Rightarrow$	(1 X)	{r$_8$, r$_9$, r$_{10}$}
r$_9$:	(X 1)	(0)	$\Rightarrow$	(0 X)	{r$_5$, r$_6$, r$_7$, r$_{11}$}
r$_{10}$:	(X 0)	(1)	$\Rightarrow$	(0 X)	{r$_8$, r$_9$, r$_{10}$}
r$_{11}$:	(X #)	(#)	$\Rightarrow$	(# X)	{r$_3$, r$_4$, r$_6$, r$_7$, r$_{12}$, r$_{14}$}
r$_{12}$:	(X 0)	(seg$_c$)	$\Rightarrow$	(0 X)	{r$_4$, r$_{12}$, r$_{14}$}
r$_{13}$:	(X 0)	(seg$_c$)	$\Rightarrow$	(0 X)	{r$_{11}$, r$_{13}$, r$_{14}$}
r$_{14}$:	(X 1)	(seg$_c$)	$\Rightarrow$	(1 X)	{r$_{11}$, r$_{13}$ r$_{14}$}

ST$_F$ $=_{def}$ {[(X) rp$_4$]}

11.5.11 Types of restriction in LA-grammar

0. LA-type A: no restriction
1. LA-type B: The length of the categories of intermediate expressions is limited by $\mathsf{k} \cdot \mathsf{n}$, where k is a constant and n is the length of the input ($R1.1$, amount).
2. LA-type C3: The form of the category patterns results in a constant limit on the operations required by the categorial operations ($R1.2$, amount).
3. LA-type C2: LA-type C3 and the grammar is at most SR-recursively ambiguous ($R2$, number).
4. LA-type C1: LA-type C3 and the grammar is at most –recursively ambiguous ($R2$, number).

11.5.12 LA-grammar hierarchy of formal languages

restrictions	types of LAG	languages	complexity
LA-type C1	C1-LAGs	C1 languages	linear
LA-type C2	C2-LAGs	C2 languages	polynomial
LA-type C3	C3-LAGs	C3 languages	exponential
LA-type B	B-LAGs	B languages	exponential
LA-type A	A-LAGs	A languages	exponential