IV. BP Grammars[81]

We refer to "BP grammars" as a class of formal grammars belonging to the family of rewriting systems, allowing for a comprehensible representation of syntactic forms used in "percussion languages": contexts, repetitions, and repetitions with homomorphic image. These grammars are presented here as an extension of the notion of pattern proposed by Angluin82.

For these grammars, a deterministic *membership* **test** algorithm was implemented in the first version of the *Bol Processor* (BP1). The test can only be applied if the rules are partially ordered. This order corresponds to a pattern recognition heuristic ("priority to the largest aggregates of symbols") and can be managed by the inference engine. In production mode, BP grammars can therefore be considered a form of **declarative** knowledge, whereas in analysis, the **procedural** aspect is emphasized. The deterministic membership test algorithm thus allows, by following the step-by-step analysis, for an explanation of failure and, consequently, for considering a generalization of the grammar if necessary.

1. *Historical* Survey

The "atomistic" conception of music—"melody is nothing more than a succession of specific sounds" (Vincent d'Indy)—is at the origin of Schoenberg's serialism,[83] followed by Webern, and then by the proponents of the stochastic approach:[84] Elisabeth Shannon,[85] Meyer,[86] Coons & Kraehenbuehl,[87] Fuchs,[88] Moles,[89] Xenakis,[90] Philippot,[91] etc.

Another way of understanding a musical idiom is to assume the existence of deep structures, in other words, a set of syntagmatic relations , and

---

81 The first four paragraphs of this chapter were presented as a paper at *the Workshop on AI & Music*, *11th International Joint Conference on Artificial Intelligence* (IJCAI). See Bel 1989a. The following paragraphs were presented at the 6th AFCET Congress—*Pattern Recognition and Artificial Intelligence* (Bel 1987b). A simplified English version (Bel 1991b) will be published.

82 1980b

83 See Chemillier 1990b for a discussion of the formal aspects of serial theory.

84 References provided by Vecchione.

85 1951

86 1956; 1957

87 1958

88 1961

89 1958

90 1963

91 1970

paradigms that allow us to characterize the productions of a musical idiom when these are represented by discrete structures.

The idea of formalizing a "musical syntax" is very old. Among the models that do not use the formalism of rewriting rules, we can cite W.A. Mozart's Musical Dice Game (*Musikalisches Würfelspiel*), and then, in the 20th century, the works of Schenker[92], which still serve as the foundation for many theoretical approaches. **Generative theory of** tonal music[93] is a synthesis of work in formal linguistics, psychology, and musicology: starting from a tonal monody, the model allows for the determination of both a surface structure (using rules of metrical structure and grouping rules, the latter borrowed from Gestalt psychology) and a deep structure (Schenkerian prolongational reduction and prosodic structure inspired by Cooper and Meyer's theory of rhythm94). The term "generative," in this context, thus refers to **the analysis** of a single piece, and not to the musical production of a set of variants:

> Our theory of music is therefore based on structural considerations; it reflects the importance of structure by focusing not on the composition of pieces but on assigning structures to already existing pieces.[95]

For a given piece, there are several possible analyses. In a second step, we therefore apply higher-order rules (preference rules[96]), taking into account consonances/dissonances, tonal cadences, etc., to determine the *best* analysis.[97]

The theory of Lerdahl and Jackendoff, which the authors later began to extend to atonal music, remains one of the most accomplished attempts to model tonal music.[98] It is the basis for much current research, some of which aims to experimentally evaluate its psychological validity, while other work focuses on designing expert systems for composition[99] or analysis[100]. A critique and extension of generative theory were formulated by Célestin Deliège.[101]

Research into formal grammars used to model **compositional** (or improvisational) **activity** has inspired some work in the field of ethnomusicology.[102] Feld, however, has spoken out against what he calls *"the hollow shell of formalism*" in musicology's borrowings from generative linguistics. Regarding Lidov's work, he writes in particular:

---

92 1935
93 Jackendoff & Lerdahl 1982, pp. 92 ff.; Lerdahl & Jackendoff 1983.
94 1960
95 Jackendoff & Lerdahl 1982, p. 85
96 *op. cit.,* pp. 99–102.
97 Thus adopting Chomsky's hypothesis of an "ideal subject."
98 It is also to this work on music that we owe the introduction of preference rules in formal linguistics.
99 For example, Camilleri 1984.
100 In particular, to simulate the perception of rhythm: see Jones et al. 1990.
101 1983; 1984.
102 Laloum & Rouget 1965; Lidov 1972; etc., cited by Feld 1974.

> […] I find the approach absurdly formal and totally pretentious, not because I can't appreciate his mathematics, but because he purports to do a theoretical task and then says nothing about what kind of ethnomusicological theory he is talking about. Yet worse, the music is considered as nothing more than one-dimensional transcriptions—a set of abstractions which a mathematician may retranslate into other abstractions of another logical order. This method of analysis is based on the completely false assumption that transcriptions of music have some sort of objective reality [,…][103]

In the same article, Feld quotes Blacking regarding what might be called an epistemological foundation of scientific ethnomusicology:

> Analytic tools cannot be borrowed freely and used as shortcuts to greater achievements in ethnomusicological research, as can electronic devices such as the tape recorder: they must emerge from the nature of the subject studied.[104]

A key issue with theories of performance or competence is their *explanatory adequacy*. According to Laske[105], a theory of performance can be:

- *observationally adequate* if it provides a comprehensible account of the data;
- *des*criptively *adequate* if the output is designed in accordance with a *well-formedness condition*, where applicable, with a measure of grammaticality;
- *explanatory* adequate if and only if it is capable of defining the internal structure of the utterance in relation to the rules of competence on which that utterance depends—at least partially.

A formal grammar that, for a given (finite) corpus of utterances, allows one to decide whether an arbitrary utterance belongs to the corpus or not, has no explanatory capacity. To reach the level of explanatory adequacy, one must in fact have a model capable of generating new correct utterances or of predicting whether an utterance belongs to the language. Such a model must therefore lend itself to generalization, an operation that the analyst can attempt to perform themselves if the representation is sufficiently explicit.[106]

2. *Pattern* gramm*ars*

2.1 Definitions and Notations

Let Vt be a finite terminal alphabet of **terminal symbols**, and Vn a countable alphabet of non-terminal symbols (**variables**). If we denote *by* H the set of *λ-free* homomorphisms of (Vn ∪ Vt)* onto itself, then every element of H whose restriction to Vt is the mapping

---

[103] Feld 1974, pp.209–10.
[104] Blacking 1972, p.1. Cited by Feld 1974.
[105] 1972a, p.5. Laske adopts here a classification of competence theories proposed by Chomsky (1964).
[106] Another approach consists of implementing an inductive inference technique. (See Chapter VI)

An identical mapping is called a **substitution**. If the image set of a substitution is Vt*, that substitution is said to **be terminal**. Any substitution whose restriction to Vn is a bijection from Vn to Vn is a *renaming of variables*.

A pattern is any element of (Vn ∪ Vt)*. If *p* is a pattern and *s* is a substitution, then s(p) is called a **derivation** of *p*. A pattern that contains no variables is called **a terminal derivation** or **a phrase**. Two patterns, *p* and *q*, are **equivalent** (written as p ≈ q) if and only if there exists a renaming of variables *r* such that p = r(q). Another binary relation (denoted p ≤ q) is defined as follows: *p* is **less general than** (or **more specific than**) *q* if and only if for a substitution *s*, p = s(q). Since the substitution is a non-erasing homomorphism, p
≤ q implies |p| ≥ |q|.

The **language** generated by the pattern *p* is the set of terminal derivations of *p*, namely L(p) = {s□ ∈ Vt+ : s ≤ p}. It can be proven that:

≤ is transitive
p ≤ q => L(q) ⊇L(p)
p ≈ q ⇔ p ≤ q and q ≤ p

However, the question of finding an effective procedure to decide whether L(q) ⊇ L(p), given two arbitrary patterns *p* and *q*, appears to be open.[107]

2.2 Application to a musical example

The sets of variations of *the qa'ida* presented in Chapter III §2 can, at first glance, be described using the following motifs *p1* and *p2*, with the variables X, Y, and Z:

Simple variations:

p1 = X tagetirakitadhin--dhagenadhatigegenakateeneteenakena
Y tagetirakitadhin--dhagenadhatigegenakadheenedheenagena

Double variations:

p2 = dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena
tagetirakitadhin--dhagenadhatigegenakateeneteenakena
Z dhatigegenakateeneteenakena
tagetirakitadhin--dhagenadhatigegenakateeneteenakena
tin--takenata--takenatatikekenakateeneteenakena
taketirakitatin--takenatatikekenakateeneteenakena
Z dhatigegenakateeneteenakena
tagetirakitadhin--dhagenadhatigegenakadheenedheenagena

Let L be the language representing all acceptable variations of *the qa'ida*, L1 the subset of simple variations, and L2 the subset of double variations. We obviously have: L = L1 ∪ L2. L(p1) and L(p2) are the languages generated respectively by *p1* and *p2*. Let us assume (based on an experimental study) that L(p1) and L(p2) contain non-trivial subsets of L1 and L2.

Ideally, we are looking for patterns *that are descriptive* of L1 and L2. A pattern *d* is **descriptive** of a language L if L(d) ⊇ L and if, for any pattern *q* such that L(q) ⊇ L, L(q) is not strictly contained in L(d).

L is in fact a finite set. For example, in this *qa'ida*, all acceptable strings of Vt* have lengths less than or equal to 32 × 6 = 192 symbols, which

[107] Angluin 1980b, pp. 49–52.

which allows one to compute an upper bound for card(L).[108] However, all that is known about L is a set S^+ **of** *positive* instances and a set S^- of *negative* instances. The set $S = S^+ \cup S^-$ is called a presentation. A pattern *p* **is** said *to* match a presentation if and only if *p* is descriptive of S^+ and $S^- \cap L(p) = \varnothing$. The pattern *p* **is** a *tight fit* **on** S **if** L(p) = S+. Clearly, any pattern that is *a tight fit* on S is descriptive of S^+ and matches S.

There is an efficient procedure for inferring a descriptive pattern from a set of examples S^+, but the problem is NP-hard in most cases.[109] Furthermore, the class of pattern languages is not closed under most basic set operations: union, complement, and intersection. Consequently, it is not easy to systematically construct a pattern description of a formal language.

2.3 *Restricted pattern* languages (RPL)

Given a pattern *p*, any subclass of L(p) can be defined by restricting the acceptable derivations of p. For example, we can define the subclass of length *n*: $L_n(p) = \{s \in Vt^+ : s \leq p \text{ and } |s| = n\}$. In the example considered, obviously |X| = |Y| = 2 and |Z| = 12. Furthermore, we can write Y = mir(X).

Another way to formulate constraints is to use **rewriting rules**. We denote

$$p \rightarrow q$$

to indicate that any acceptable derivation of *q* is an acceptable derivation of *p*, in other words q ≤ p. If we denote *by S* the set of all substitutions, p —> q denotes the subset

$$S_{p\rightarrow q} \quad \text{such that } \forall s \in S_{p\rightarrow q},\ s(p) = q.$$

Since the problem of determining whether p ≤ q for two arbitrary numbers p and q is NP-complete,[110] we must limit ourselves to **well-defined rules**:

Definition:

> The rule p → q is well-formed if and only if $p \in Vn^+$, $q \in (Vt \cup Vn)^+$, |q| ≥ |p|, and no variable appears more than once in p.

Theorem:

> If the rule p → q is well-formed, then L(p) ⊇ L(q).

Proof:

It suffices to find a substitution *s* such that q = s(p), which implies q ≤ p. One procedure for constructing *s* is as follows:

---

[108] The number of strings of length at most *n* in an alphabet of cardinality k > 1 is indeed:

$$\frac{k \cdot (k^n - 1)}{k - 1}$$

[109] Angluin 1980b, pp. 54–55.

[110] *Op. cit.,* p. 52

(1) replace the |p|-1 leftmost variables of *p* with the |p|-1 leftmost variables of *q*;

(2) replace the rightmost variable in *p* with the string formed by the |q|-|p|+1 rightmost symbols of *q*.

Since no variable appears twice in *p*, there is no constraint on *q* that makes these substitutions impossible.[111] For example, a substitution of XYZ yielding abcX would be {X/a, Y/b, Z/cX}.■

Let L(p) be a pattern language and R a set of well-formed rules. Every rule (p → q) defines a set of substitutions $S_{p \to q}$. To construct a sentence of **the restricted pattern language** $L_R(p)$, proceed as follows:

(a) Select a subset R_o of R;

(b) let $S_o = \cap_i(S_{pi->qi})$ such that $(p_i \to q_i) \in R_o$;

(c) if $S_o \neq \varnothing$ and S_o is finite, then $S_o = \{s\}$ such that the sentence is w =

s(p). The selection of R_o calls for some comments:

(1) If a variable X does not appear on the left-hand side of any selected rule, then it can be replaced by any q $\in (Vt \cup Vn)^+$. In this case S_o is infinite and consequently $L_R(p) = \varnothing$.

(2) There are subsets of R for which $S_o = \varnothing$, that is, **aborted derivations**. For example, if a variable X appears on the left-hand side of two distinct rules in R_o, such as X → q_i and X → q_j, then $S_{X \to qi} \cap S_{X \to qj} = \varnothing$, and thus $S_o = \varnothing$.

(3) In all cases other than (1) and (2), every variable appears in a pattern with a single derivation, so that card(S_o) = 1.

The following example is intended to clarify points (2) and (3). An RPL for double variations would be L_R (p2) with:

p2 = P192 and R is the set of rules:
(1) P192 —> C Z D A E Z D B, where E = mir(C)
(2) A —> tagetirakitadhin--dhagena D
(3) B —> tagetirakitadhin--dhagenadhatigegenakadheenedheenagena
(4) C —> dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena A
(5) D —> dhatigegenakateeneteenakena
(6) Z —> tirakitatirakitatirakita (7) Z —> ZA ZB (8) Z —> ZC ZD ZE ZF
(9) ZC ZD —> ZG ZH (10) ZD ZE —> ZG ZH (11) ZE ZF → ZG ZH
(12) ZG ZH —> dhagenadhin-- (13) ZG ZH —> dhagenadha--
(14) ZC —> dhin-- (15) ZD —> dhin-- (16) ZE —> dhin-- (17) ZF —> dhin--
(18) ZC —> dha-- (19) ZD —> dha-- (20) ZE —> dha-- (21) ZF —> dha--
(22) ZA —> dheenedheenedheene (23) ZB —> dheenedheenedheene
(24) ZA —> dha-dha-dha- (25) ZB —> dha-dha-dha-

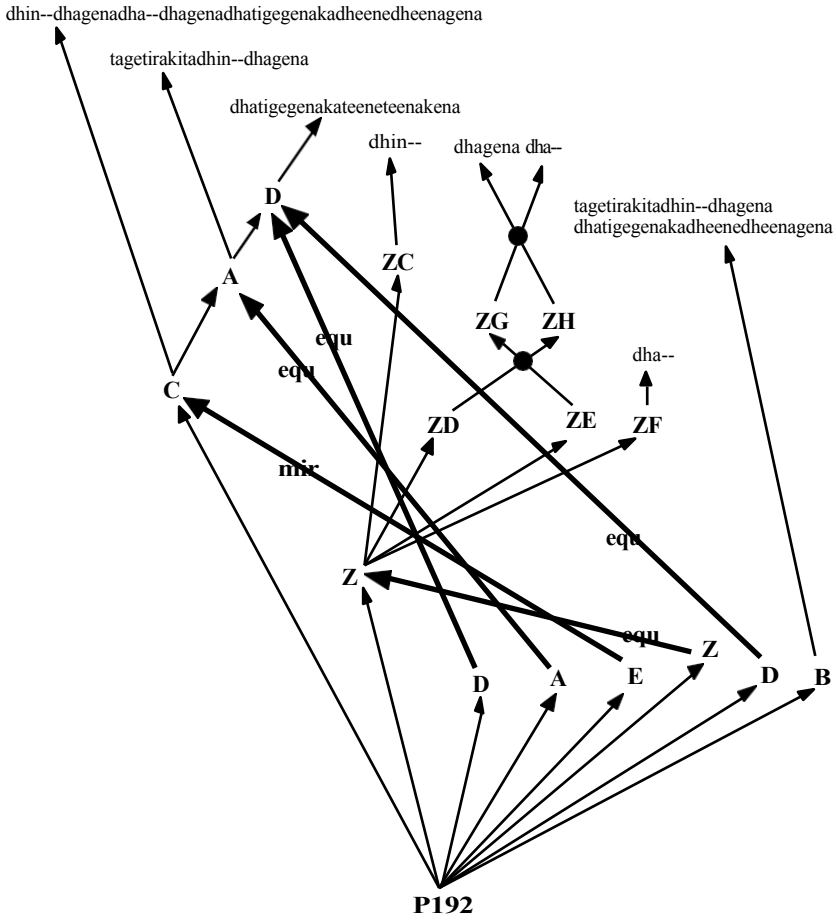
If we choose R_o ={1,2,3,4,5,8,10,13,14,21}, we obtain the phrase:

---

111 This procedure is inspired by Makanin's algorithm (Makanin 1977, Roussel 1987).

dhin--dhagena dha--dhagena dhatigegenaka dheenedheenagena **C**
tagetirakita dhin--dhagena dhatigegenaka teeneteenakena
dhin-- dhagena dha--dha-- dhatigegenaka teeneteenakena **Z D**
tagetirakita dhin--dhagena dhatigegenaka teeneteenakena **A** tin--
takena ta--takena tatikekenaka teeneteenakena **E**
taketirakita tin--takena tatikekenaka teeneteenakena
dhin--dhagena dha--dha-- dhatigegenaka teeneteenakena **Z D**
tagetirakita dhin--dhagena dhatigegenaka dheenedheenagena **B**

which can be represented by the following (context-sensitive) syntactic graph, where 'equ' and 'mir' denote strict repetition and mirror repetition structures:



The substitution sets for rules 10 and 14 can be represented by:

$$Sp10\text{->}q10 = \{..., ZD\ ZE/ZG\ ZH, ...\}$$

and

$$S_{p14\text{->}q14} = \{..., ZC/dhin\text{--}\ ,...\}$$

where '...' denotes the (countable) set of all possible substitutions of the strings in $(Vt \cup Vn)^+$, with the exception of those that have already been specified (ZD, ZE, and ZC). We see that

$$Sp10\text{->}q10 \cap Sp14\text{->}q14 = \{..., ZD\ ZE/ZG\ ZH, \ldots, ZC/dhin\text{--}\ ,...\}.$$

The result of the intersections is:

S_o = {P192/C Z D A E Z D B, A/tagetirakitadhin--dhagena D,
B/tagetirakitadhin--dhagenadhatigegenakadheenedheenagena,
C/dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena A, D/dhatigegenakateeneteenakena,
Z/ZC ZD ZE ZF, ZC/dhin--, ZD ZE/ZG ZH, ZF/dha--, ZG ZH/dhagenadha--}

where each derivation is explicit. Consequently, $S_o$contains a unique substitution that produces the sentence in question.

If , then , we select , R(o)= {1,2,3,4,5,8,10,13,14,15,21} , and , we obtain $S_{p10->q10}$ = {..., ZD ZE/ZG ZH, ...}, $S_{p15->q15}$ = {..., *ZD/dhin--* ,...}, and $S_{p20->q20}$ = {..., *ZE/dha--* ,...}. Let's call $S_{15,20} = S_{p15->q15} \cap S_{p20->q20}$= {..., *ZD/dhin--* ,..., *ZE/dha--* ,...}. For any substitution s $\in S_{15,20}$, s(ZD ZE) = s(ZD) s(ZE) = *dhin--dha--*. Therefore, ZD ZE can never be substituted for ZG ZH, which contradicts $S_{p10->q10}$. In this case, $S_o = \varnothing$ and the derivation fails.

Note: One could also say that $S_o$is the maximal (and thus most specific) conjunctive expression using predicates (pi = qi) such that (pi → qi) $\in R$.

Theorem:

The class of finite languages coincides exactly with that of the RPLs.

Proof:

Any subset of the (finite) set of rules in an RPL generates at most one terminal derivation. The set of terminal derivations is therefore finite. Conversely, for any finite language L, there exists a *non-imbedding right-linear* grammar without recursive rules that generates exactly L. Let G = (Vt, Vn, S, R) where S is the start symbol and R is a finite set of rules of

form A → a or A → aB such that A, B ∈ Vn, A ≠ B, and a ∈ Vt. It is easy to see that G exactly generates the RPL LR(p) such that p = S, which completes the proof.■

Corollary:

The class of RPLs is recursive and closed under union, concatenation, and intersection.

We construct an effective procedure to compute L_R(p) = L_{R1}(p1) ∪ L_{R2}(p2) as follows:

Suppose that L_{R1}(p1) and L_{R2}(p2) are defined as:
p1 = S1 and $R1$ = {S1 → q1, ...}
p2 = S2 and $R2$ = {S2 → q2, ...} in which the variables of $R2$ have been renamed so that no variable appears in both $R1$ and $R2$. We construct L_R(p) defined by:
p = S and R = {S → S1, S → S2} ∪ $R1$ ∪ $R2$.
The set L_R(p) of terminal derivations of S is the union of the sets of terminal derivations of S1 and S2; therefore, L_R(p) = L_{R1}(p1) ∪ L_{R2}(p2). Furthermore, since $R1$ and $R2$ containonly well-formed rules, R also contains only well-formed rules. Therefore, L_R (p) is an RPL.

RPLs form a class of formal languages that is closed under union, and there exists a procedure for constructing a language from its subsets. The class

is also recursive, and consequently the membership test is algorithmic. These properties are useful for constructing a representation model that allows for descriptive generalizations.

3. BP grammars

3.1 *Pattern* rules

The rewriting rules in §2.3 can be replaced by **pattern rules**:

P96 —> X A Y B *with* mirror(X,Y) and length(X,24)
P192 —> C Z1 D A E Z2 D B *with* equal(Z2,Z1) and length(Z1,12) and mirror(C,E)

These rules consist of a standard rewriting part and the predicates *mirror*, *length,* and *equal,* which express constraints.

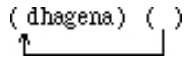
It is clear that if two variables linked by an *equal/mirror* predicate appear on the right-hand side of a rule, the variable on the far left denotes a string that serves as a reference, and the other denotes a copy of that reference. For example, with the following rules:

A —> B C where equal(B,C)
B —> dhagena

the terminal derivation *dhagenadhagena* can be written as

(= dhagena) (: dhagena)

where the first parenthesis (marked with "=") serves as the reference (**master**), and the next one (marked with ":") as the copy (**slave**). In a computer implementation, the second parenthesis actually contains a pointer to the reference:



The same notation is used in the rules, for example:

A —> (= B) (: B)

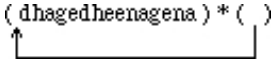
As for homomorphisms, a reserved symbol is used to indicate that the content of the following parenthesis (master or slave) is a homomorphic image. For example, the mirror of percussion languages is denoted by "*". Given the mirror homomorphism defined in Chapter III §2, in the following grammar

S —> (= D) * (: D)
D —> dhagedheenagena

the unique terminal derivation is

(= dhagedheenagena) * (: taketeenakena)

and its internal representation:



Using these conventions, it is possible to simplify the formalism for representing RPLs:

(1) If a variable appears multiple times in an RPL rule, it must be marked with repetition markers. For example,
A —> B C B C B should be written as: A —> (=B) (=C) (:B) (:C) (:B).

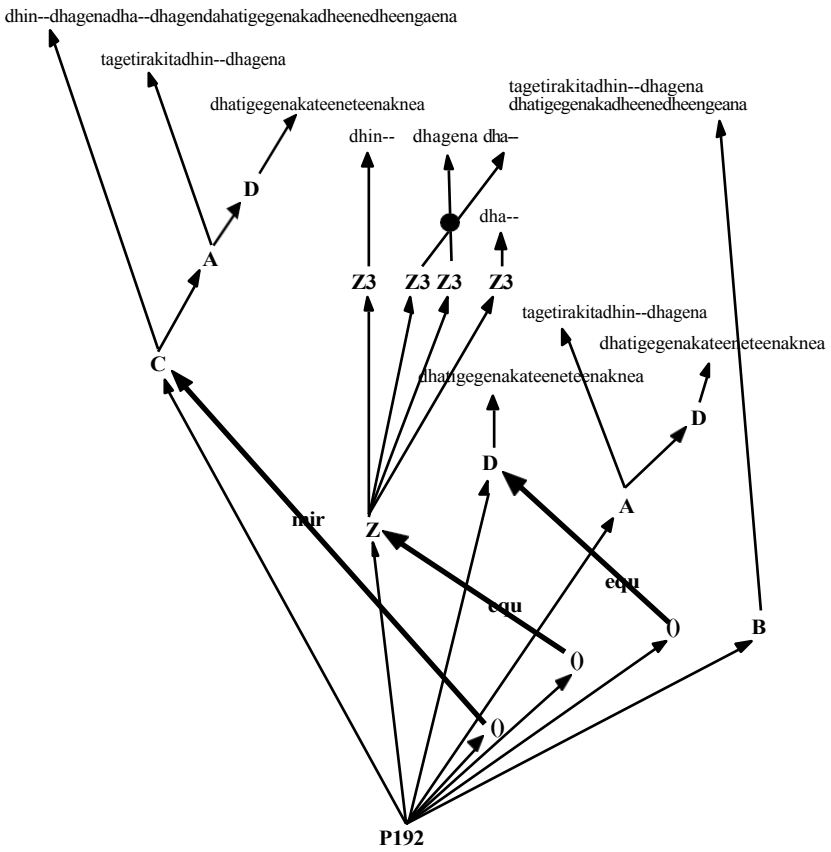
(2) The same variable may appear twice in the left-hand side of a rule (provided these two occurrences are not linked by a repetition marker). Thus, the rule A A —> q is correct, but not the rule (=A) (:A) —> q.

(3) Mirrors are represented by an asterisk.

With these conventions, the BP grammar generating the RPL language defined in §2.3 is:

(1) P192 —> (=C) (=Z) (=D) A * (:C) (:Z) (:D) B
(2) A → tagetirakitadhin--dhagena D
(3) B —> tagetirakitadhin--dhagenadhatigegenakadheenedheenagena
(4) C —> dhin--dhagenadha--dhagenadhatigegenakadheenedheenagena A
(5) D —> dhatigegenakateeneteenakena
(6) Z —> tirakitatirakitatirakita
(7) Z —> Z6 Z6
(8) Z —> Z3 Z3 Z3 Z3
(12) Z3 Z3 —> dhagenadhin--
(13) Z3 Z3 —> dhagenadha--
(14) Z3 —> dhin--
(18) Z3 —> dha--
(22) Z6 —> dheenedheenedheene
(24) Z6 —> dha-dha-dha-

in which rules 9, 10, 11, 15, 16, 17, 20, 21, and 23 have been removed. The syntactic graph of the sentence generated in §2.3 is now:

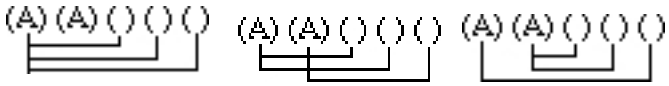


3.2 Master-slave assignment

The pattern representation defined in §3.1 may be ambiguous. For example, the expression

(= A) (= A) (: A) (:A) (:A)

can represent various pointer assignments:



etc…

Only the first assignment is taken into account in the BP2 version of *the Bol Processor*. In the BP1 version, all assignments can be performed using the editor. In both cases, the ambiguity is resolved even if the assignment performed is not explicitly visible on the screen.

The assignment of a parenthesis is identified by a pointer (a positive integer or zero) *i,* which we call **the parenthesis marker**, calculated as follows:

1) Master parenthesis: i = 0 (**"marker** zero")
2) Slave parenthesis: (i-1) is the number of zero markers separating the slave parenthesis from the master parenthesis to which it is attached.

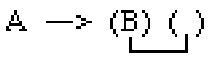
The values of *i* for the slave parentheses in the three previous assignments are as follows:

(=A) (=A) (2) (2) (2) (=A) (=A) (2) (2) (1) (=A) (=A) (1) (1) (2) .

We now show how these assignments are recalculated during rewrites. Suppose the following rule is applied to the previous expression:

A —> (= B) (:B)

at the second derivation position, i.e., on the second occurrence of A. The rule is written as



or alternatively:

A —> (=B) (1)

After rewriting, we get:

(=A) (=(=B) (:B)) (:A) (:A) (:A)

for the first assignment, or

(=A) (=(=B) (:B)) (:A) (:A) (:(=B) (:B))

for the second, or finally

(=A) (=(=B) (:B)) (:(=B) (:B)) (:(=B) (:B)) (:A)

for the third.

These formulas can be represented more simply using the markers, respectively:

(=A) (=(=B) (1)) (**3**) (**3**) (**3**) (=A) (=(=B) (1)) (**3**) (**3**) (**2**) (=A) (=(=B) (1)) (**2**) (**2**) (**3**)

It is easy to see that the markers shown in bold have been incremented by one after replacing A with (=B) (1).

The algorithm for reassigning markers in the workstring X(i) after a rewrite is as follows:

```
inmark = n2 - n1; /* n1 and n2 represent the number of zero markers
   in the left and right sides of the rule, respectively.
   */
i0 = position of the rightmost symbol rewritten by the rule; q
= 0;
For (i = i0; i < length of the expression)
|    If (X[i] is a zero marker)
|    then
|    |    q = q + 1;
|    else
|    |    If (X[i] is a slave marker and X[i] ≥ q)
|    |    then
|    |    |    X[i] = X[i] + inmark;
|    |    Endif
|    EndIf
|    i = i + 1; Endif
```

It is easy to see that only the tokens pointing to master parentheses to the left of the rewritten part are incremented by *inmark*.

In conclusion, rewriting in a BP grammar is done in two steps:

1) Rewriting symbols and markers;

2) Correcting markers using the previous algorithm.

The formalism we have introduced for representing patterns therefore does not alter the rewriting procedure (step 1): tokens are copied as arbitrary symbols from the alphabet Vn. In what follows, we can therefore treat any BP grammar as a type-0 formal grammar, noting that step 2 must be performed after each rewriting.

4. BP transformational grammars (*BP transformational grammars*)

This term is used here in a more restrictive sense than in linguistics. The idea, borrowed from Kain112, is to consider every derivation as a sequence of derivations across multiple grammars; the initial symbols AD of a grammar, except for "S," the start symbol, are terminal symbols of a higher-level grammar.

Definition

> A **transformational grammar** (without deletion rules) G is an ordered quintuple (Ve,Vn,Vt,Vd,F) such that:

112 1981, p.24

$V = Ve \cup Vn \cup Vt \cup Vd$, $Vt \neq \varnothing$, $Vd \neq \varnothing$

(Ve, Vn, Vt, Vd) is a partition of a finite alphabet V,

and F is a finite set of ordered pairs (LCR, LDR) such that: C $\in$

$(Vn \cup Vt \cup Vd)^+$, L $\in V^*$, R $\in V^*$, and D $\in$ (Vn $\cup$ Vt $\cup$ Vd)

Vd is the initial alphabet, Vt the terminal alphabet, Vn the intermediate alphabet (the variables), Ve the alphabet outside G, and F the set of production rules.

The initial alphabet of a grammar is necessarily included in the union of the terminal alphabets of the preceding grammars, except for the first one where Vd = {S}. If the language is described by *n* grammars, the terminal alphabet of G_n is included in the terminal alphabet of the language. We also agree that the external alphabet of G_1 is empty and that every symbol in the external alphabet of a grammar G_i is a terminal of a grammar G_j with j < i. Terminal symbols that do not represent bowls are called **structural symbols**. We will see later (Chapter V §5) how these symbols are handled by the *templates*. The structural symbols recognized by *the Processor* BP1 are:

brackets and repeat markers: **(=) (:)**

the mirror symbol: *****

the integers (1..99) representing the current speed of the

special symbols chosen from: {**+ : ; =** }

5. Control of derivations in the grammatical BP (*derivation control in a BP grammar*)

Various control strategies have been implemented in the BP1 and BP2 versions of *the Bol Processor*. A detailed justification of these strategies has been published.[113] We first present derivations in "production" mode. "Recognition" mode is discussed in §6.

5.1 'RND' *grammars*

The generation of sentences with stochastic control of rule order and derivation position uses the following procedure:

```
Procedure GENERATE
Start
  As long as (a rule is applicable),
  |    select a candidate rule; select a
  |    derivation position;
  |    If (the left-hand side of the rule is
  |    recognized) then
  |    |    Replace the found occurrence with the right-hand
  |    side; End
  Endwhile
End
```

[113] Kippen & Bel 1990.

A rule is selected via a draw weighted by the weights of the candidate rules (see Chapter V §6). The derivation position is random. The BP1 inference engine can also generate all sentences of the language in order.

This control mode is indicated by placing the 'RND' (*random*) instruction above the rules of the transformational grammar.

5.2 *Controlling the derivation position*: *'LEFT' and 'RIGHT' rules*

It is often necessary—and in any case faster—to take the leftmost or rightmost occurrence of the left-hand side of the selected rule in the working string. This can be controlled on a per-rule basis by placing the corresponding directive, 'LEFT' or 'RIGHT', at the beginning of the rule.

5.3 'LIN' *gramm*ars

In production, checking the derivations of a 'LIN' grammar is identical to that of an 'RND' grammar in which all rules are of the 'LEFT' type.

5.4 'ORD' gramm*ars*

In an 'ORD' grammar, rules are selected in the order in which they appear in the grammar. Each rule is applied until saturation, then the grammar is parsed again to find the first candidate rule. The derivation position is chosen as with the 'LEFT' directive.

'ORD' grammars are used primarily in cases where no stochastic choice—neither regarding the candidate rule nor the derivation position—is necessary.

6. BP grammar *membership test for a BP grammar*)

An example of syntactic analysis using a BP grammar (extended formalism) is provided in the appendix.

6.1 Al*gorith*m

The general principle of the membership test algorithm is as follows:

1) Given a transformational grammar G, swap the left-hand and right-hand sides of all the rules in G. Let G' be the grammar thus obtained (**dual grammar**).
2) Since the language is generated by applying the transformational grammars G_1, ..., $G_n$in that order, the membership test is the result of the **canonical right derivation** of the string analyzed by G'_n, ... G'_1 in that order.
3) The string belongs to the language if and only if the membership test ends with the sentence symbol S.

The permutation of arguments in the rules justifies the use of double arrows in grammars: <—>

This algorithm has multiple advantages. First, it allows us to use essentially the same procedures as in synthesis to perform the membership test. Furthermore, it is deterministic, memory-efficient, and the number of rewrites is less than the length of the analyzed string.

The only difficulty lies in defining a canonical derivation for context-sensitive grammars. By *"right-hand derivation*," we mean a derivation that rewrites the rightmost non-terminal symbol in the chain. This definition is ambiguous: is the context part of the symbols being rewritten? If the answer to this question is 'yes,' it is not possible to provide a general solution to the problem of ambiguity in contextual grammar. However, a more general definition of canonical derivations[114] allows us to define constraints on the production rules that guarantee the grammar will not be ambiguous.

6.2 Contextual contextual to right (*context-sensitive rightmost derivation*)

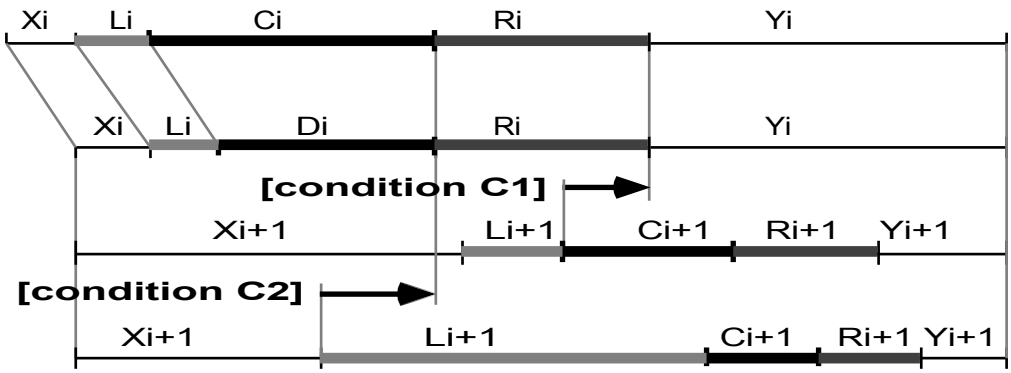
Let G be a context-sensitive grammar. The derivation of G:

$W0 \Rightarrow W1 \Rightarrow ... \Rightarrow Wn$

is **context-sensitive** *rightmost* if and only if

$\forall i \in [0, n-1]$, $W_i = X_i L_i C_i R_i Y_i$
and $W_{i+1} = X_i L_i D_i R_i Y_i$ by applying the rule f_i: $L_i C_i R_i \rightarrow L_i D_i R_i$,
and at least one of the following conditions is satisfied:
(C1) $|C_{i+1} R_{i+1} Y_{i+1}| > |Y_i|$
(C2) $|L_{i+1} C_{i+1} R_{i+1} Y_{i+1}| > |R_i Y_i|$

This definition is adapted from Hart's[115]. Hart considers only **strictly context-sensitive** grammars for which $|C_i| = |C_{i+1}| = 1$, and condition C1 can then be written as $|R_{i+1} Y_{i+1}| \geq |Y_i|$. The figure below illustrates conditions **C1** and **C2**, which will be justified later:



Suppose that conditions **C1** and **C2** are not satisfied. Since **C2** is not satisfied, the rule f_{i+1} could have been applied before f_i, since $L_{i+1}C_{i+1}R_{i+1}$ would be a substring of R_iY_i. On the other hand, since **C1** is not satisfied, the application of the

[114] Hart 1980.
[115] *op. cit.,* p. 82

rule f_{i+1} would only modify the string Y_i without affecting the context R_i. In this case, the order of f_i and f_{i+1} could be reversed. This permutation would be legitimate since the symbols rewritten by f_{i+1} are to the right of those rewritten by f_i.

6.3 Application of the canonical derivation to the recognition algorithm (*applying the canonical derivation to the membership test*)

Suppose that for a string Wi, two rules are applicable:

fi LiCiRi -> LiDiRi

f 'i L'iC'iR'i -> L'iD'iR'i

given that $X_iL_iC_iR_iY_i = X'_iL'_iC'_iR'_iY'_i = W_i$

Let us order these rules according to the following criteria: f_i will be chosen over f'_i if one of the following conditions, taken in order, is satisfied:

(D1) |XiLiCi| > |X'iL'iC'i|
(D2) |XiLiCiRi| > |X'iL'iC'iR'i|
(D3) |LiCiRi| > |L'iC'iR'i|
(D4) i > i'

Comment:

D1 ensures that the maximum number of rules qualify for the next rewriting step by satisfying condition **C2**.

D2 allows a maximum number of rules to be candidates for the next rewriting step by satisfying condition **C1**.

When **D3** is applied, relations **D1** and **D2** reduce to equalities, and consequently $R_i = R'_i$ and $L_i C'_i R'_i$ is a substring of $L_i C_i R_i$. The ambiguity is then resolved by applying the following principle:

> The longest patterns (including contexts) are recognized first.

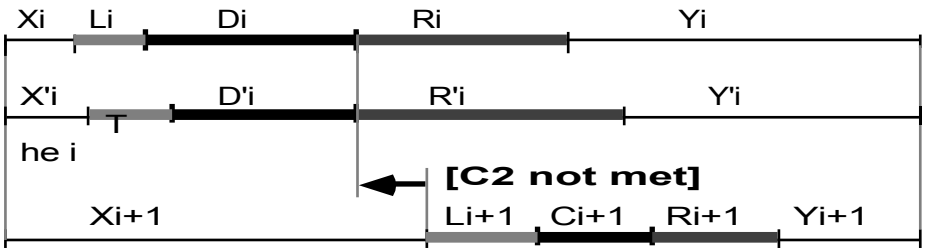
When **D3** does not resolve the ambiguity, the right-hand sides of the production rules are identical. There is indeed ambiguity, but this situation can be avoided by careful grammar design. In this case, the inference engine uses an arbitrary selection criterion: the reverse order of appearance of the rules in the grammar.

6.4 Simplifying the criteria in the *Bol Processor* (*BP*)

Suppose that after applying criterion **D1**, two rules f_i and f'_i are candidates, and thus |XiLiCi| = |X'iL'iC'i|. After rewriting by f_i and f'_i respectively, the strings would be:

$W_{i+1} = X_iL_iD_iR_iY_i$ and $W'_{i+1} = X'_iL'_iD'_iR'_iY'_i$ with $R_iY_i = R'_iY'_i$

Is it necessary to use criterion **D2** to choose between f_i and f'_i? This criterion allows for the best possible fulfillment of condition **C1** on the candidate rules f_{i+1}. Let's take the example of a rule f_{i+1} that satisfies **C1** but not **C2**:



It is clear that the situation described above cannot occur because, in that case, the rule f_{i+1} would have to be applied before f_i or f_{i+1} according to criterion **D1**. It follows that examining criterion **D2** is not useful for resolving the ambiguities arising from the application of **D1**.

Ambiguities arising from the application of criterion **D1** will therefore be immediately resolved by **D3**. In this case, $L'_iC'_iR'_i$ is not necessarily a substring of $L_iC_iR_i$. In the BP1 version, criterion **D3** is not taken into account by the inference engine, and it is ultimately **D4**—that is, the order of the rules—that resolves ambiguities. It follows that the user must adhere to the following rule:

Chunk Rule

The right-hand side of a rule f_i cannot be a substring of that of a rule f_j such that j<i.

6.5 Context Dependency and *Canon*ical Derivation

Derivations in contextual grammars can be represented by syntactic trees.[116] However, these trees do not allow one to deduce the canonical form of a derivation, i.e., the order of rule application. A satisfactory solution was proposed by Hart[117]. It consists of inserting directed arcs into the syntactic tree that express two types of constraints:

$<_c$ context dependency: the rule pointed to by the end of this arc can only be applied if the symbol at its origin is present;
$<_f$ s context "release": the rule pointed to by the end of this arc is "frozen" until the rule pointed to by its origin has been applied.

We can also add directed arcs $<_d$ defined as follows to the syntax tree:

An arc connects f_i and f_j if there is no path between these nodes; $f_i <_d f_j$ if the derivation of f_i occurs to the right of that of f_j.

These arcs, when depicted together, form a *doub***ly ordered** 3-colored **graph**, and this graph allows us to deduce an order of rules that corresponds to the canonical derivation.[118] Consider, for example, the following **LIN** grammar:

f_1 S <—> EBA
f_2 B <—> CD

116 Révész 1985, p. 57
117 1980
118 Révész 1985, p.182

f_3 DA <—> DEF
f_4 EC <—> AAC

in which we have highlighted the contexts and the canonical derivation to the right of the word:

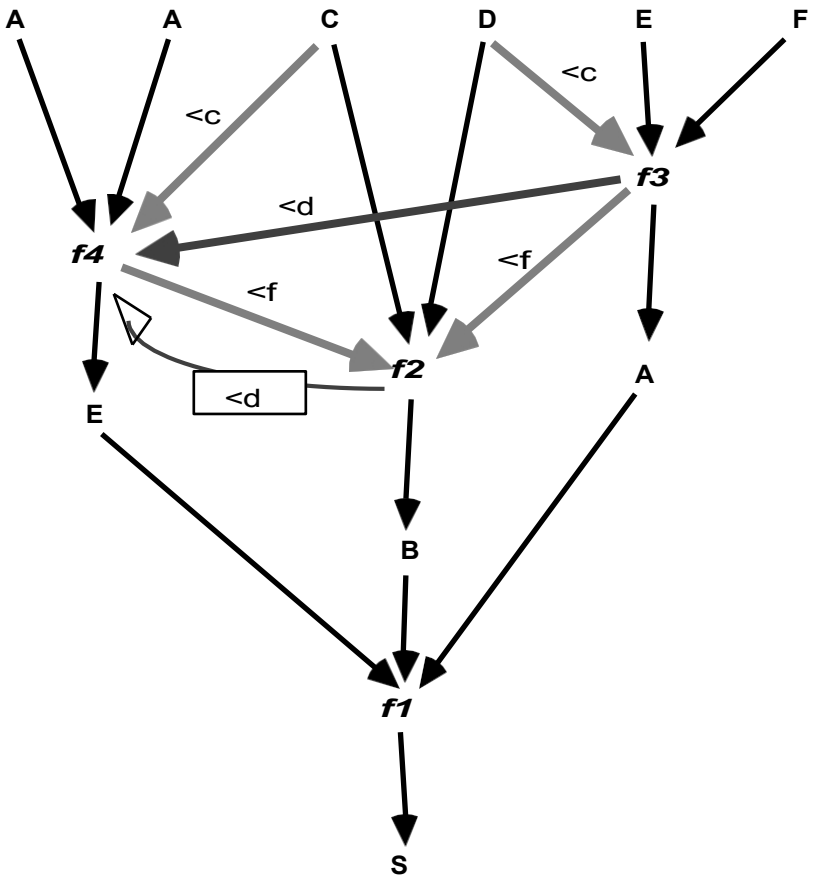
AACDEF => AACDA => ECDA => EBA => S
f_3 f_4 f_2 f_1

If we applied criterion **D1** defined in the previous paragraph, the derivation would be:

AACDEF => AACDA => AABA
f_3 f_2

Consequently, criterion **D1** is insufficient here to define the canonical right derivation. We have depicted below the *context-sensitive syntactic* graph[119] corresponding to the correct analysis. The arc labeled $<_{\mathbf{d}}$ in the box is not part of this graph, since there is already an arc $<_{\mathbf{f}}$ between f_4 and f_2. It is
clear that the freezing of rule f_2 contradicts the order imposed by constraint **D1**. It is clear, on the other hand, that the arcs $<_{\mathbf{f}}$ originating from a node f_i representing a rule with
The contexts on the left are all oriented from right to left, that is, in the same direction as the arcs $<_{\mathbf{d}}$. Finally, there is no pair of arcs ($<_{\mathbf{c}}$,$<_{\mathbf{f}}$) originating from a context that does not belong to the alphabet of a transformational grammar.

119 *ibid.*



We derive the following rule:

Context Rule

> In a LIN grammar, a right-hand context can consist only of symbols from the external alphabet of that grammar.

6.6 *Membership test for 'RND' and* 'ORD' *grammars*

Provided that the chunk and context rules are followed, 'LIN' grammars allow for the canonical derivation, in parsing, of any sentence in the language. In 'RND' and 'ORD' grammars, the order of the rules takes precedence over the derivation position (criterion **D4**). Furthermore, when a rule is selected, it is applied until saturation.

The corresponding algorithm is as follows:

```
Procedure REWRITE(i)
Start
  Search for the right-hand side of f_i in the sentence starting from the
  right; Replace the occurrence found with the left-hand side of f_(i);
End
```

```
Procedure
SATURATE(i) Start
  While (rule f_i is applicable) REWRITE(i);
  |     test = 1;
  Endwhile
End

Procedure ANALYZE
Start
  Repeat
  |     i = number of the last rule;
  |     test = 0;
  |     Repeat
  |     |     SATURATE(i);
  |     |     i = i - 1;
  |     until (i = 0) or (test = 1) until (test
  = 0)
End
```

This algorithm is acceptable when all rule contexts use only symbols from the external alphabet, and when the patterns of the right-hand sides of the rules do not partially overlap. The problem of pattern overlap is addressed elsewhere.[120]

[120] Bel 1987.