Bol Processor Grammars

Bernard Bel, James Kippen

groupe
représentation
et traitement
des
connaissances

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE
31, chemin Joseph Aiguier
F-13402 MARSEILLE CEDEX 9 (France)

téléphone: *(33) 91 22 40 64*
telex: *CNRSMAR 43 0225 F*
télécopie: *(33) 91 71 08 08*
e-mail: *grtc@frmop11.bitnet*

Bol Processor grammars

Bernard Bel & Jim Kippen

Abstract

Bol Processor grammars are an extension of unrestricted generative grammars allowing a simple representation of string "patterns", here taken to mean repetitions and homomorphic transformations. These have been successfully applied to the simulation of improvisatory techniques in traditional drum music, using a production-rule system called "Bol Processor BP1". The basic concepts and parsing techniques of BP1 are presented.

A new version of Bol Processor, namely "BP2", has been designed to serve as a aid to rule-based composition in contemporary music. Extensions of the syntactic model, such as metavariables, remote contexts, substitutions and programmed grammars, are briefly introduced.

Keywords

Formal grammars, pattern languages, membership test, ethnomusicology.

GRTC / 456 / Janv. 1991

Bernard Bel & Jim Kippen

In M. Balaban, K. Ebcioglu & O. Laske (Eds.) "Understanding AI with music"

AAAI Press, 1992, pp.366-400

Bol Processor grammars

Bernard Bel[1] **& Jim Kippen**[2]

Abstract

Bol Processor grammars are an extension of unrestricted generative grammars allowing a simple representation of string "patterns", here taken to mean repetitions and homomorphic transformations. These have been successfully applied to the simulation of improvisatory techniques in traditional drum music, using a production-rule system called "Bol Processor BP1". The basic concepts and parsing techniques of BP1 are presented.

A new version of Bol Processor, namely "BP2", has been designed to serve as a aid to rule-based composition in contemporary music. Extensions of the syntactic model, such as metavariables, remote contexts, substitutions and programmed grammars, are briefly introduced.

Keywords

Formal grammars, pattern languages, membership test, ethnomusicology.

---

1 Groupe Représentation et Traitement des Connaissances (GRTC)
Centre National de la Recherche Scientifique
31, chemin Joseph Aiguier, F-13402 Marseille Cedex 9.
E-mail: bel@frmop11.bitnet Fax: (033) 91 71 08 08

2 Faculty of Music
University of Toronto
Edward Johnson Building
Toronto, Ontario, Canada M5S 1A1.
E-mail: kippenj@utorepas.bitnet Fax: (416) 978 5771

CONTENTS

Bol Processor grammars

Bernard Bel & Jim Kippen

Although in the field of computer-aided composition considerable time has been devoted to new sound generation techniques, many studies of structural models have been restricted to the analysis of Western staff-notated music. Beyond this domain (particularly in ethnomusicology) there have been early attempts to borrow concepts from linguistics on the basis of assumed parallels between language and traditional musical systems. Some provocative papers, however, resulted in the virtual rejection of purely abstract speculations that Feld had termed "the hollow shell of formalism":

> Only Blacking [...] and Lindblom and Sundberg [...] have dealt explicitly with basic theoretical issues... The rest of the literature ignores issues like the empirical comparison of models, a metatheory of music, evaluation procedures, and the relation of the models to the phenomena they supposedly explain.
>
> [Feld 1974:210]

The project which initiated the work presented in this chapter has been an in-depth study of the music of a group of drummers in North India. [Kippen 1988] The oral tradition encountered differed markedly from the few descriptions of improvisational formulae in the limited and often unreliable literature. This meant a shift in focus from purely analytical models (imbedding a hypothetical description of musicians' competence) to models of musical performance. Analytical models attempt to describe structural regularity in a particular musical piece (e.g. [Jackendoff & Lerdahl 1982]) or in a set of related variations. In the first case, the structure that is discovered by the analyst (possibly with the help of a computer) claims to reflect an underlying musical (or perceptive) system which may be called the **competence** of an "ideal listener". In the analysis of variations, a model of the competence of an "ideal musician" is sought. However, both approaches rely on epistemological considerations (borrowed from linguistics) that do not take into account individual "deviations", i.e. models of **performance**: the originality of a particular musical work, or simply the creativity of a musician. In this respect, music is more comparable to *literature* (or poetry) than to language.[3]

Our experimental work[4] on traditional *tabla* drumming was mainly focussed on computer simulations of improvisation schemata called *qa'ida*[5]. A computer program

---

[3] Regarding aesthetic communication, see for instance [Laske 1973b]. Theories of individual compositions viewed as products of rule-governed artistic creativity are named *counter-grammars* by Laske [1973a:359].

[4] Initial work was part of a research scheme by the *International Society for Traditional Arts Research* (ISTAR, New Delhi) generously funded by the *Ford Foundation* and the *National Centre for the Performing Arts* (NCPA, Bombay). Kippen's work was also supported by the *Leverhulme Trust* and the *Economic and Social Research Council* in the UK.

[5] *Qa'ida* may be thought of as "theme and variations" in which the variations are variously referred to as *palta*, *vistar*, *bal*, *penc*, etc.

called **Bol Processor (BP1)** was implemented on the Apple™ IIc, i.e. the only computer (at the time, 1982-85) portable enough to be taken to locations where the interaction with expert musicians could take place "in context". A detailed description of the BP1's *modus operandi* may be found in [Kippen & Bel 1989a]. Results and problems arising from the BP methodology have been discussed in [Kippen 1987]. The transfer of knowledge from human informants to machines using automatic rule generation is presented in [Kippen & Bel 1989b-d] and [Bel 1990a].

This chapter is an introduction to the Bol Processor grammatical model. We first demonstrate the scope and limitations of conventional generative grammars on the basis of a simple musical example borrowed from the *qa'ida* repertoire. In response to these limitations we introduce a rule format for the representation of "string patterns" (repetitions and homomorphic transformations). Then the method for parsing unrestricted BP grammars is outlined, and additional features allowing a control of derivations are shown. In the end we describe the main features of a new version of Bol Processor, namely **BP2**, meant to be used as a compositional tool dealing with a wider variety of musical systems.

1. Generative grammars in Bol Processor BP1

It is assumed that, in some musical systems, elementary musical objects ("atoms") may be transcribed with the aid of a non-empty finite set of symbols Vt, namely the **alphabet** of **terminal symbols**. The set of all finite strings over an alphabet A is notated A*. A music sequence may therefore be represented as string belonging to Vt*.[6] Any properly defined subset of Vt* is a **formal language**.

For example, many pieces of *tabla* music are transliterated with onomatopoeic syllables representing sounds and strokes on the drums, called *bol*s[7]. (Using onomatopoeic languages for the transmission and occasionally the performance of traditional drum music is a common practice both in Asia and in Africa.) A terminal alphabet used in many *tabla* compositional types is for instance

Vt = {tr, kt, dhee, tee, dha, ta, ti, ge, ke, na, ra, -}

in which the hyphen indicates a silence (or the prolongation of a resonant stroke).[8] Symbols "tr" and "kt" are shorthand for "tira" and "kita". In general, any finite set of labels for **sound-objects** (e.g. notes) may be used as a terminal alphabet for music.

1.1 The scope of work with the Bol Processor

A number of ethnomusicologists have attempted to use generative grammars to represent sets of "acceptable" variations of a musical theme. The relevance of a concept like "acceptability" should of course be understood in relation to the musical system under study. Players of the *tabla* themselves claim that there is a precise system underlying "correct" variations although its rules are generally not explicit. The main motivations of our project, therefore, have been (1) to make rules explicit for some compositional types,

---

[6] A string representation of non-sequential structures is proposed in part B of "*Symbolic and sonic representations of sound-object structures*" in this volume.

[7] From the verb *bolna*, "to speak", in North Indian languages; for this reason the machine was named "Bol Processor".

[8] A system for transliterating tabla strokes to non-ambiguous symbolic representations has been proposed by Kippen [1988:xvi-xxiii].

and (2) to check the consistency of musicians' assessments of correctness both in both teaching and performance situations.

The compositional type most fundamental to an understanding of composition and improvisation in *tabla* playing is the *qa'ida*, the "theme and variations" form *par excellence*. Not only do beginners learn *qa'ida*s, usually with sets of "fixed variations" composed by their teachers (thus providing models of the crucial art of improvisation), but advanced players use them too, particularly in solo performances, to demonstrate their technical mastery and mental skills. Furthermore, musicians postulate that unless one can improvise on *qa'ida* themes, one is not adequately equipped to improvise on any of the other theme and variations forms.

1.2 The hierarchy of generative grammars and formal languages

Throughout this chapter we will refer to Chomsky's hierarchy of generative grammars. A generative grammar is an ordered fourtuple (Vt, Vn, S, F) in which Vt is an alphabet of terminal symbols, Vn an alphabet of variables (with $Vt \cap Vn = \varnothing$), S a distinguished symbol of Vn, and F a finite set of rewriting rules P —> Q such that P and Q are strings over the alphabet ($Vt \cup Vn$) and P contains at least one symbol from Vn. We call P and Q the **left** and **right argument** of the rule respectively.

We use the notation |X| to designate the length of a string X. The empty string is notated λ. One of many (equivalent) ways of defining the hierarchy of generative grammars is:

> Type 0 (**phrase-structure**): unrestricted
>
> Type 1 (**length-increasing** or **context-sensitive**):
>
> $|P| \leq |Q|$ except possibly for the rule S —> λ
>
> Type 2 (**context-free**): $|P| = 1$ and $|P| \leq |Q|$ except possibly for the rule S —> λ
>
> Type 3 (**regular** or **finite-state**): every rule has form either
>
> X —> a Y
>
> or Z —> b, where *X, Y*, and *Z* are variables and *a* and *b* terminal symbols.

Every type-*n* grammar generates a formal language of type *n'*, for some $n' \geq n$. This yields a proper hierarchy of language classes. [Révész 1985:7] For instance, it can be proved that every finite language is regular, hence context-free, etc., so that it may be generated by a grammar of any type.

Throughout this chapter small letters are used for terminal symbols, and variables start with a capital letter.

1.3 Context-free grammars for *qa'ida*s

As a result of some basic observations about the structure of *qa'ida*s — the regular alternation of fixed and variable sections, and the predominance of permutation and substitution as improvisatory devices — it was thought that formal language models would be suited to the construction of grammatical models.

Fig.1 is an example of variation on a well-known *qa'ida* theme. A theme may itself be viewed as one particular variation — here in Fig.1 the kernel of the theme is represented in the first line with a variation of it in the second. It should be read from left to right as plain

text. The durations of all syllables (including "tr" or "kt" as composites) are identical. Syllables are grouped into beats, therefore we may say that this piece has a **stroke density** of four strokes per beat. Since each line contains four beats, the total metric duration of the piece is sixteen beats (anything from eight to twelve seconds in performance depending on interpretation).

dha ti dha ge	na dha tr kt	dha ti dha ge	dhee na ge na
dha tr kt dha	ti dha ge na	dha ti dha ge	tee na ke na
ta ti ta ke	na ta tr kt	ta ti ta ke	tee na ke na
dha tr kt dha	ti dha ge na	dha ti dha ge	dhee na ge na

Fig.1 The theme of a *qa'ida*

Some strokes on the *tabla* have a voiced (resonating) and an unvoiced (dampened) version. Here, the cadential string "dha ti dha ge dhee na ge na" is repeated at the end of each line in its voiced as well as partly-voiced ("dha ti dha ge tee na ke na") and fully-unvoiced ("ta ti ta ke tee na ke na") transformations. The complete mapping of voiced to unvoiced strokes in this *qa'ida* will be shown in Fig.7.

The piece in Fig.1 belongs to a set of acceptable variations that may be very large although it is certainly finite since all pieces are bound by the metric cycles, i.e. a duration of sixteen or thirty-two beats. A complex variation is given in appendix 2. However, let us for a while consider only the first lines of a subset of ten simple variations:

dha tr kt dha	tr kt dha ge	dha ti dha ge	dhee na ge na
dha tr kt dha	tr kt dha dha	dha ti dha ge	dhee na ge na
dha ti dha tr	kt dha tr kt	dha ti dha ge	dhee na ge na
dha tr kt dha	ti - dha ti	dha ti dha ge	dhee na ge na
dha tr kt dha	ti dha tr kt	dha ti dha ge	dhee na ge na
ti - dha ti	dha dha tr kt	dha ti dha ge	dhee na ge na
ti dha tr kt	dha dha tr kt	dha ti dha ge	dhee na ge na
tr kt dha ti	dha dha tr kt	dha ti dha ge	dhee na ge na
tr kt tr kt	dha dha tr kt	dha ti dha ge	dhee na ge na
tr kt dha tr	kt dha ge na	dha ti dha ge	dhee na ge na

Fig.2 The first lines of ten variations of the *qa'ida*

This set can easily be described using a regular/finite-state (type-3) grammar. A regular grammar can be represented as a **finite acceptor**, i.e. a directed graph in which *X*, *Y*, and *Z* are state labels, and *a* and *b* transition labels (see Fig.3). Using a rule X —> a Y to rewrite "X" as "a Y" is equivalent to jumping from state *X* to state *Y* following the transition labelled *a*. The second type of rule, Z —> b, is represented as a transition from state *Z* to an **accepting state** *nil* (see Fig.3).[9]

[9] We call "finite acceptor" a kind of finite automaton with a univocal mapping of the set of states to the set {"acceptable", "unacceptable"}. This suggests that other mappings can be envisaged, see for instance [Allouche 1987].

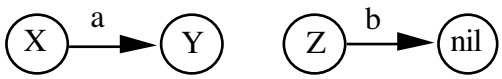

Fig.3 Basic transitions in a finite acceptor

The state from which all paths originate is labelled *S*, the initial symbol in the grammar. To analyze a string, each of its component symbols (from left to right) is used as a "road sign". The string is grammatically correct if it is possible to move from *S* to an accepting state following all the road signs. For example, a finite-state acceptor recognizing exactly the ten examples given in Fig.2 could be the one shown in Fig.4.

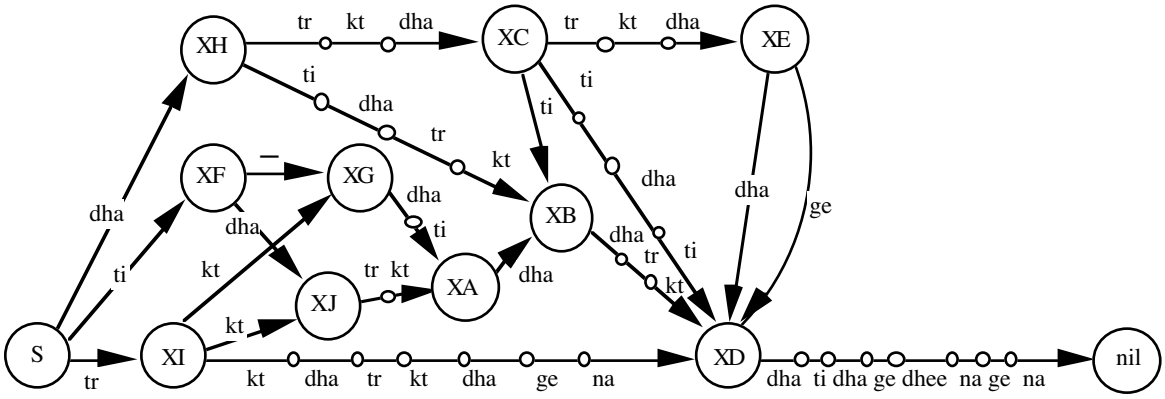

Fig.4 A finite acceptor for the language shown Fig.2

The purpose of this representation is twofold: (1) it serves as a "classifier" telling whether or not a given string belongs to the set of original examples in Fig.2, and (2) it can be used to generate any string belonging to the set. However, it is not unique and its musical relevance will even be questioned in §1.4. To simplify the representation, only those states which are (diverging or converging) nodes of the graph have been labelled. Other states appear as small circles. This suggests an alternate equivalent representation using a "two-layer acceptor" as in Fig.5.

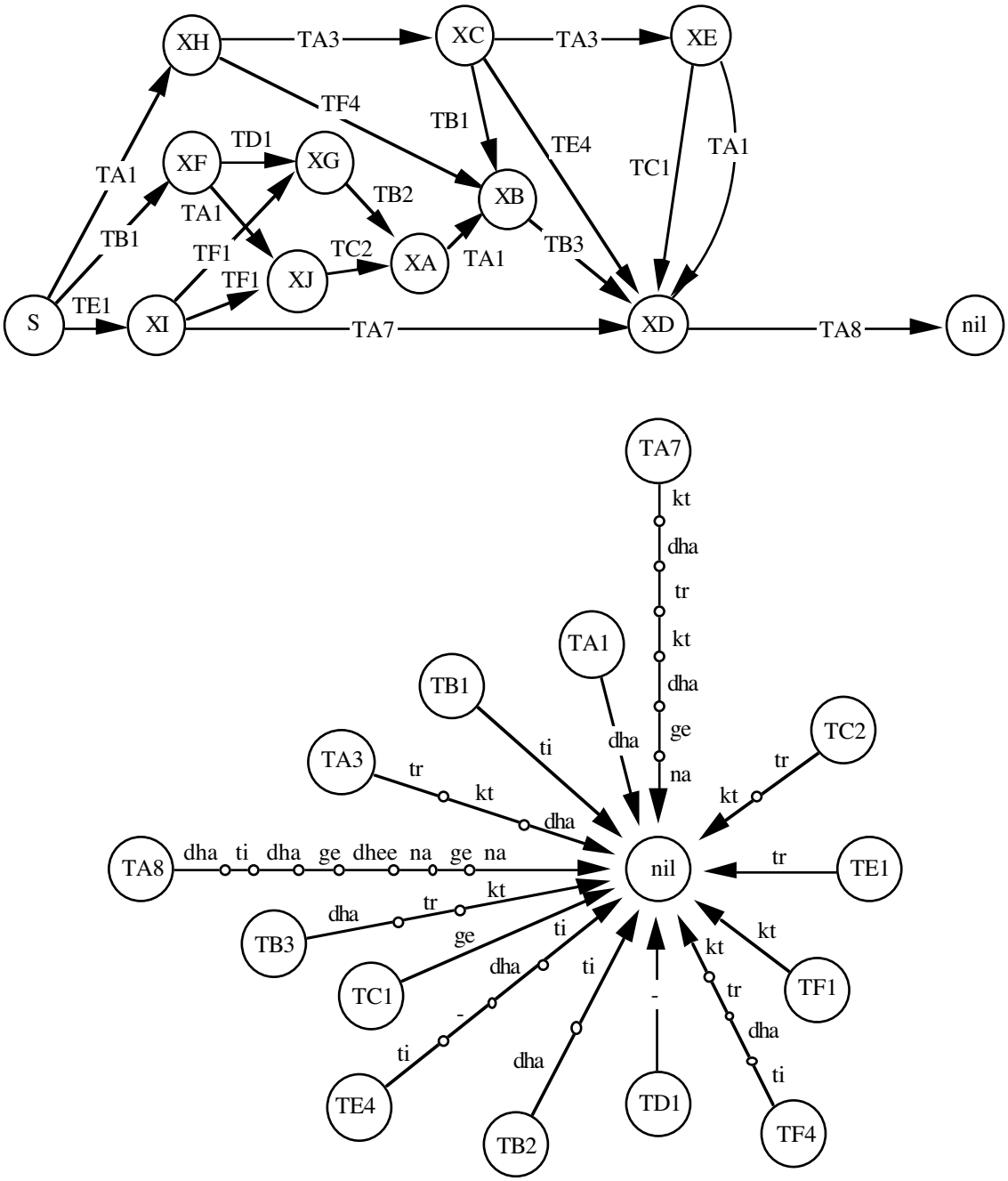

Fig.5 An equivalent "two-layer finite acceptor"

This new acceptor is equivalent to the generative grammar of Fig.6. The mapping of transitions to grammar rules is self-explanatory.

S	—>	TE1 XI			
XI	—>	TA7 XD			
XD	—>	TA8			
XI	—>	TF1 XJ	TA7	—>	kt dha tr kt dha ge na
XJ	—>	TC2 XA	TC2	—>	tr kt
XA	—>	TA1 XB	TE1	—>	tr
XB	—>	TB3 XD	TF1	—>	kt
XI	—>	TF1 XG	TF4	—>	ti dha tr kt
XG	—>	TB2 XA	TD1	—>	-
S	—>	TA1 XH	TB2	—>	dha ti
XH	—>	TF4 XB	TE4	—>	ti - dha ti
XH	—>	TA3 XC	TC1	—>	ge
XC	—>	TE4 XD	TB3	—>	dha tr kt
XC	—>	TA3 XE	TA8	—>	dha ti dha ge dhee na ge na
XE	—>	TA1 XD	TA3	—>	tr kt dha
XE	—>	TC1 XD	TB1	—>	ti
XC	—>	TB1 XB	TA1	—>	dha
S	—>	TB1 XF			
XF	—>	TA1 XJ			
XF	—>	TD1 XG			

Fig.6 A grammar equivalent to the two-layer acceptor

Some variable labels have numbers indicating the metrical values of terminal strings which are derived therefrom. Thus, "TA7" denotes a string of seven strokes. Although this information is not used by the Bol Processor, it facilitates checks of the grammar when variations are of fixed length; here for instance the sum of metrical values along each path of the upper acceptor is 16 (meaning four beats in stroke density 4).

This grammar is **context-free** (type-2) although it generates a finite (type-3) language. Rules shown on the right side of Fig.6 are called **lexical rules**. Their right arguments are chunks of strokes that have been repeated several times in the examples of Fig.2, presumably "words" of the language although they do not bear any semantic value.

The segmentation of musical pieces into "significant chunks" has been discussed in great detail by musicologists, e.g. Ruwet [1972] and Nattiez [1976]. In brief, lexical rules define the vocabulary of the piece as presumably perceived by musicians, a task that requires some presupposed knowledge about what is "meaningful" and what is not [Kippen & Bel 1989b:203]. In fact, the vocabulary displayed by the grammar Fig.6 was not assessed as correct, therefore another grammar was constructed on the basis of musicians' comments that eventually yielded a correct segmentation of the variations. (See [Kippen & Bel 1989b:210])

A context-free grammar in the format of Fig.6 may be viewed as a combination of two "**transformational subgrammars**" corresponding to the two automata shown Fig.5. The term "transformational" is borrowed from formal language theory [Kain 1981:24], not linguistics. To generate a variation, the first subgrammar (rules on the left side) is used until no further derivation is possible, for instance:

S
TE1 XI
TE1 TA7 XD
TE1 TA7 TA8

Then rules of the second subgrammar (right side of Fig.6) are applied in an arbitrary order:

TE1 TA7 TA8
TE1 kt dha tr kt dha ge na TA8
tr kt dha tr kt dha ge na TA8
tr kt dha tr kt dha ge na dha ti dha ge dhee na ge na

The module that takes care of (enumeratively or randomly) selecting rules in order to generate variations is part of the **inference engine** of the Bol Processor. The other part of the inference engine is the parsing module that will be described in §3.

1.4 Generalizing grammars

Despite some musicians' claims that they follow systematic procedures in constructing sets of variations, it is difficult to discover what that system may be as they rarely create more than six to eight variations in any given situation. Furthermore, few players are able or willing to describe in words what procedures they follow, and even then we have observed that there is a considerable discrepancy between what is said and what is actually done.

Therefore, the main problem in formalizing improvisation schemata as found in *qa'ida* is the inference of a grammar recognizing large sets of correct variations on the basis of a small set of typical examples. This problem belongs to **structural pattern**[10] **recognition** (see for instance [Fu 1982], [Miclet 1984]), where a pattern classifier is inferred from a relatively small set of training examples. The classifier is expected to assign correct class numbers to a (possibly infinite) number of new patterns. Finite acceptors (equivalently, regular grammars) are sorts of string classifiers working with only two classes: "acceptable" and "unacceptable".[11] In addition, they can also generate any string belonging to the "acceptable" class.

Given a sample set of correct and incorrect variations, i.e. **positive** and **negative instances** of the language, a learner (human or mechanical) should be able to construct a grammar that eventually accepts/generates all possible correct variations and rejects the incorrect ones. We can imagine that this learning process is strictly **incremental**: each time a new (positive or negative) instance is supplied, the learner is expected to adjust the currently guessed grammar accordingly. This inference process belongs to **inductive generalization** because the new grammar may become smart enough to recognize/generate variations that have not yet been supplied as examples.[12]

Take for instance the finite acceptor of Fig.4 and suppose that the eleventh example supplied by the musician were:

ti dha dha ti dha dha tr kt dha ti dha ge dhee na ge na

---

[10] The word "pattern" is used here in a less restrictive sense than in §2.

[11] These can also be adapted to deal with more than two classes. Probabilistic grammars, for instance, may be viewed as pattern classifiers on an arbitrary number of classes. See [Booth & Thompson 1973].

[12] See [Case & Lynes 1982] for a formal introduction.

A new acceptor recognizing this example could be obtained by merging states XJ and XG. This would yield an acceptor recognizing/generating one more variation that has not yet been assessed by the musician:

ti - tr kt dha dha tr kt dha ti dha ge dhee na ge na

If this variation is incorrect, then other inferences may be tried, for example constructing a path of two transitions labelled "dha" and "ti" connecting XJ to XA.

Any inference, therefore, requires an assessment of newly generated variations. When training a pattern classifier, negative instances are supplied so that incorrect inferences may eventually be detected. In music teaching or performance situations this is generally not the case. Therefore the validity of generalizations is assessed by instructing the Bol Processor to generate variations that are submitted to informants: any variation which they reject is then considered as a negative instance.

1.5 Implications of the generalization technique on the grammatical model

Regular grammars (equivalently, finite acceptors) have been used extensively in syntactic pattern recognition. (See for instance [Fu 1982].) A major reason is that there exist relatively efficient inductive inference methods for the identification of regular (type-3) languages. **Identification in the limit** means that, once a finite number of positive and negative instances of an unknown language have been supplied to the learner, the currently guessed grammar no longer requires modification, and this grammar recognizes exactly the target language.[13] Gold's theorem [1967] says that (1) any enumerable class of decidable languages (see §3) can be identified in the limit using positive and negative instances of the language, and (2) no class of formal languages containing all finite languages and at least one infinite language may be identified in the limit using exclusively positive instances.

An implication of Gold's theorem is that there is little scope for identifying languages in the absence of negative instances. Angluin [1980a] has given useful characteristic properties of languages that can be identified from exclusively positive sample sets.

An algorithm for inferring context-free grammars (in the format shown in §1.3) has been designed [Bel 1990a] and successfully tried in support to the analytical work with the Bol Processor. (See the **QAVAID**[14] system in [Kippen & Bel 1989b].) This approach provides interesting results insofar as variations may be considered sequences of words taken from an unknown vocabulary: the algorithm yields the vocabulary and a grammar determining the segmentation and deep structure of any variation.

There are human limits as to the number of examples that can be supplied to the machine, let alone problems of time/space complexity in the grammatical inferencer. Therefore, even though the identification of finite languages like *qa'ida* could be performed mechanically, part of the generalization process is still left to human experts for the sake of heuristic efficiency. In the absence of a music theory able to account formally for the scores of several hundreds of *qa'ida*s performed by a group of musicians, the major part of the work is being accomplished by a human analyst. This has obvious implications for the grammatical model used for representing music. It is difficult, for instance, to reconcile one's musical intuitions with a context-free grammar in the restrictive format shown in

---

[13] More flexible learning criteria have been defined, among others, by Case & Lynes [1982].

[14] "Question-Answer-Validated Analytical Inference Device". This acronym is also a word meaning "grammar" in Arabic/Urdu — *Qava'id* is the plural of *qa'ida*

§1.3, all the more so if the number of rules is bound to exceed a few hundred. This reflects an obvious discrepancy between machine- and human-oriented representations.

For this reason, we developed a grammatical model that is general and inclusive of representations aimed at limiting the number of rules needed to account for significant musical ideas. Only now that many *qa'ida*s have been identified could we envisage looking for a unified restrictive grammar format encompassing this whole musical system. Meanwhile, Bol Processor BP1 supports generative grammars in unrestricted (type-0) format. It is the task of the analyst to stick to rule formats enabling the machine to generate and recognize the same language. As indicated in §3, parsing variations is generally not possible with arbitrary grammar formats.

2. Pattern rules in BP grammars

So far we have dealt only with permutations of "words". In order to find an appropriate representation of periodic structures (systematic repetitions, etc.) we developed the idea of **pattern rules**.

We call a **string pattern** any element of (Vn ∪ Vt)*, i.e. a string containing variables and terminal symbols. Every variable in a string pattern may in turn be replaced with another arbitrary string pattern. Replacing all occurrences of a variable with the same non-empty string is called a **substitution**.

Consider for example the alphabets

$$Vt = \{a, b, c, d, \ldots, z\} \quad Vn = \{A, B, C, \ldots, Z\}$$

and a string pattern "A a b A B c B". Substitutions of this pattern may be:

A **a b** A B **c** B	Original string pattern
C e C a **b** C e C c d **c** c d	Substitute "A" with "C e C" and "B" with "c d".
f g h e f g h **a b** f g h e f g h c d **c** c d	Substitute "C" with "f g h".
etc…	

If *p* is a string pattern and *s* a substitution, then s(p) is a **derivation** of *p*. A string pattern containing no variable is called a **terminal derivation**. The set of all terminal derivations of *p* is called the **pattern language** generated by *p*. [Angluin 1980b]

We felt it would be interesting to combine the representational power of pattern languages (in terms of periodicity) with the versatility of generative grammars. Generative grammars are rather counterintuitive for the representation of string-patterns.[15] Consider for instance the following grammar (proposed by [Salomaa 1973:12]) generating the language derived from string pattern "X X" over terminal alphabet Vt = {a,b}.

[15] The class of pattern languages is properly included in the class of unrestricted (type 0) languages, but it is not comparable with any other class.

```
S      —> A B C
A B    —> a A D
A B    —> b A E
D a    —> a D
D b    —> b D
E a    —> a E
E b    —> b E
D C    —> B a C
E C    —> B b C
a B    —> B a
b B    —> B b
A B    —> λ
C      —> λ
```

This grammar is non-restricted (type-0). In addition, most derivations of the starting symbol "S" halt on a string that still contains variables; therefore it is difficult to control the generative process so that only terminal strings are produced.

To overcome these limitations we developed an extension of the rule format that we call **pattern rules**. [Bel 1990c:41-43]

Informally, a pattern rule generating the "X X" pattern language (on any terminal alphabet) would be the following:

```
S  —> (= X) (: X)
```

in which brackets indicate that all derivations of the occurrences of "X" must be identical. The leftmost expression "(= X)" is the **reference** and "(: X)" its **copy**. We call brackets containing "=" or ":" **pattern delimiters**. There may be several copies of the same reference, e.g.:

```
S  —> (= A) (= B) (: A) (: B) (: A)
```

Repetitions may not always be strict. In many musical systems a number of transformations affecting terminal symbols have been proposed. In §1.3, for instance, we suggested that strokes on the *tabla* may be either voiced or unvoiced. Fig.7 shows the mapping of the corresponding voiced/unvoiced transformation, which stands for all *qa'ida*s using these strokes.

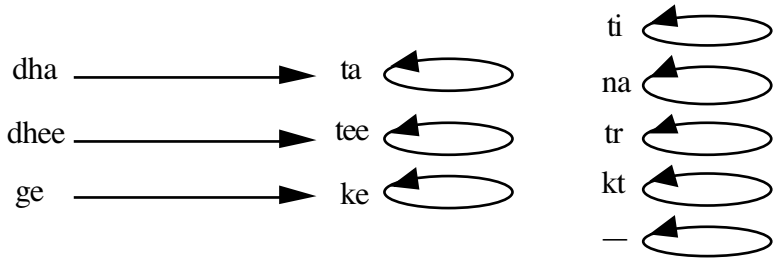

Fig.7 A "voiced/unvoiced" mapping in *tabla* music

This mapping may be extended to any string over the terminal alphabet Vt (yielding a λ-free homomorphism, see [Révész 1985:10]). For instance, the unvoiced image of "dha ge na" is "ta ke na". To indicate a homomorphic transformation we insert a special symbol (a **homomorphic marker**) before the pattern delimiter, indicating the part of the string in which the transformation must be performed. The marker used for the voiced/unvoiced transformation is an asterisk. For instance, the grammar

S —> (= D) * (: D)
D —> dha ge dhee na ge na

yields the terminal derivation

(= dha ge dhee na ge na) * (: ta ke tee na ke na)

which is internally represented with the help of a **master-slave assignment pointer**. (See Fig.8)

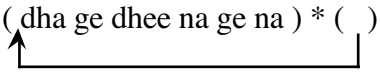

Fig.8 Master-slave assignment pointer

This internal representation is economical in terms of memory space. The algorithm for rewriting assignment pointers in string patterns is presented in [Bel 1990c:43-45]. It applies to multilayered representations as well.

The variation given in Fig.1 may be represented:

(= dha ti dha ge	na dha tr kt	dha ti dha ge	dhee na ge na)
(= dha tr kt dha	ti dha ge na)	(= dha ti dha ge	tee na ke na)
* (: ta ti ta ke	na ta tr kt	ta ti ta ke	tee na ke na)
(: dha tr kt dha	ti dha ge na)	(= dha ti dha ge	dhee na ge na)

This variation is produced by the grammar shown in Fig.9, using rules 1, 4, 7, 8, etc.

[1] S	—>	(=A16) (=V8) A'8 *(:A16) (:V8) A8
[2] S	—>	(=V16) A'16 *(:V16) A16
[3] S	—>	(=V24) A'8 *(:V24) A8
[4] A16	—>	dha ti dha ge na dha tr kt dha ti dha ge dhee na ge na
[5] A'16	—>	dha ti dha ge na dha tr kt dha ti dha ge tee na ke na
[6] A8	—>	dha ti dha ge dhee na ge na
[7] A'8	—>	dha ti dha ge tee na ke na
[8] V8	—>	*... define permutations of eight strokes*
[9] V16	—>	*... define permutations of sixteen strokes*
etc…		

Fig.9 A grammar with three pattern rules

Rules defining V8, V16 and V24 may be context-free as shown in §1.3. An elaborated version of the grammar of this *qa'ida* is given in appendix 1.

We call a **Bol Processor grammar** any type-0 grammar containing pattern rules.

3. Parsing variations with BP grammars

When designing Bol Processor BP1 we wanted a machine that could not only generate variations which would be submitted to musicians, but also one that could evaluate musical pieces proposed by musicians as correct examples. In this process (called a **membership test**), the inference engine verifies that a given variation is well-formed according to the rules of the grammar. The membership test is performed by the **parsing module** of the inference engine. Only when a grammar agrees with the expert after both generating and

parsing variations can one then suggest that it reflects the musician's knowledge of the "language".

One of the major reasons why the BP could be used successfully for modelling part of the musical behaviour of traditional musicians was its ability to display compositional and analytical processes in two different ways. Information on how correct variations can be composed is generally supplied higgledy-piggledy. This fits well with a grammatical representation in which the order of rules is arbitrary. On the other hand, if a musician/musicologist is prompted to describe how a variation is analyzed, he/she is likely to come up with an ordered set of instructions such as:

> In the analytical process, the first significant chunks to be recognized are those appearing in fixed positions (e.g. the cadential part "dha ti dha ge dhee na ge na", see §1.3). Then variable parts are analyzed. Large chunks or "words" are recognized first.

This type of description indicates a human preference for a **data-driven** (i.e. bottom-up) procedure that starts from the string under analysis and rewrites it until a "success/failure" flag is obtained (see appendix 2).

The main problem with the membership test arises when there is disagreement between a human expert and the machine regarding the acceptability of a variation. Usually the variation is a correct one, and yet the machine rejects it thereby pointing to a defect in the grammar. In order to understand why the variation has been rejected, the analyst should be able to repeat the procedure step by step. This may lead to a clue provided that the trace reflects an intuitive analytical process. (See for instance appendix 2.) For instance, tracing a parsing procedure is difficult if the procedure is non-deterministic, i.e. if it backtracks on failures. Therefore we opted for a **deterministic** parsing procedure.

Another problem is purely theoretical. The class of languages for which there exist membership tests (i.e. **decidable** languages) contains the class of context-sensitive (type-1) languages, but there are also unrestricted (type-0) languages that are undecidable. In addition, membership tests for classes of languages properly containing the class of context-free (type-2) languages often lead to inefficient parsing procedures. (See [Loeckx 1970], [Révész 1970].) The design of an efficient procedure, therefore, implies restrictions on the grammar format.[16]

3.1 BP1 parsing procedure

The procedure which we implemented in Bol Processor BP1 is data-driven, deterministic, and quite efficient in terms of time/space complexity.[17] Informally, the algorithm is the following:

> 1) If G is a transformational grammar, its **dual** is obtained by swapping the left and right argument in every rule.
> 2) Given G_1, ..., G_n, the subgrammars of the language, and their respective duals G'_1, ..., G'_n, the membership test is the result of the **context-sensitive canonic rightmost derivation** of the string under analysis by G'_n, ..., G'_1 in this order.
> 3) If the membership test has yielded S, the starting symbol, the input string is "accepted". It is rejected in any other case.

---

[16] Restrictions on BP grammars are not described here. See [Bel 1987a-b].

[17] The number of derivation steps needed for parsing a string is smaller than its length; space complexity is linear.

Informally, the grammar is turned "upside down" and rules in each subgrammar are applied in reverse. If for instance a string "dha tr kt dha ti dha ge na" is generated using the subgrammar

(1) V3 —> dha ge na
(2) V2 —> tr kt
(3) V1 —> dha

then during its analysis by BP1 the dual subgrammar

(1') dha ge na —> V3
(2') tr kt —> V2
(3') dha —> V1

should be used. This introduces ambiguity because, if rule (3') is used first, then the string is rewritten "V1 tr kt V1 ti V1 ge na" so that no further rule is applicable. Indeed this goes against the intuitive reasoning that suggests "large chunks should be recognized first". Therefore the parsing procedure is expected to impose a (partial) ordering of rules and set preferences for the position of derivations. This is accomplished by the canonic rightmost derivation.

3.2 Canonic rightmost derivation

The concept of context-sensitive canonic (leftmost) derivation was defined by Hart [1980:82] for *strictly* context-sensitive grammars, i.e. grammars in which all the left arguments of rules contain no more than one variable. We first extended this definition to all length-increasing grammars. Then we adapted it to the dual grammars in which rules are always length-decreasing, i.e. their right arguments are shorter than their left ones.

Let W_i be the string under derivation after the i-th derivation step. (Each step corresponds to the application of a rule of the grammar.) The context-sensitive rule used for the i-th derivation may be written

$$L_iC_iR_i \text{ —> } L_iD_iR_i$$

in which L_i and R_i are the left and right contexts respectively. C_i is the part of the string W_i that will be rewritten as D_i. Since the rule is length-decreasing, $|D_i| \leq |C_i|$.

Context-sensitive rightmost derivation

Let G be a length-decreasing grammar. The derivation in G:
$W_0 => W_1 => ... => W_n$
is **context-sensitive rightmost** iff:
$\forall i \in [0, n-1]$, $W_i = X_iL_iC_iR_iY_i$,
$W_{i+1} = X_iL_iD_iR_iY_i$ after having applied rule f_i: $L_iC_iR_i$ —> $L_iD_iR_i$,
the next rule to be applied will be f_{i+1}: $L_{i+1}C_{i+1}R_{i+1}$ —> $L_{i+1}D_{i+1}R_{i+1}$
such that $W_{i+1} = X_{i+1}L_{i+1}D_{i+1}R_{i+1}Y_{i+1}$,
and at least one of the two following conditions is satisfied:
(C1) $|C_{i+1}R_{i+1}Y_{i+1}| > |Y_i|$
(C2) $|L_{i+1}C_{i+1}R_{i+1}Y_{i+1}| > |R_iY_i|$

The diagram and commentary of Fig.10 illustrate conditions **C1** and **C2**.

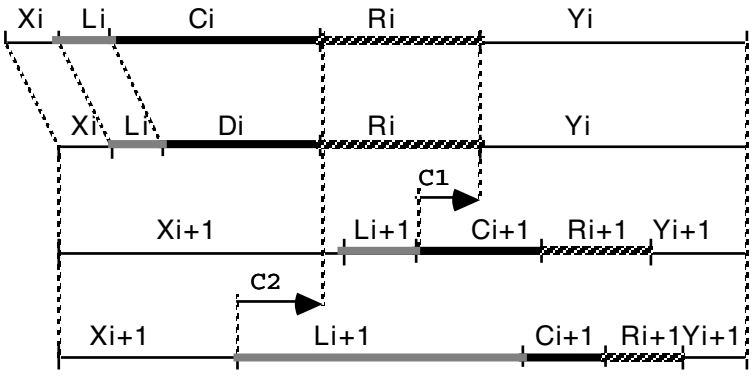

Fig.10 Context-sensitive rightmost derivation

Suppose that neither conditions **C1** nor **C2** are satisfied. Since **C2** is not true, rule f_{i+1} could have been applied before f_i as $L_{i+1}C_{i+1}R_{i+1}$ would be a substring of R_iY_i. Besides, since **C1** is not true, applying rule f_{i+1} would only modify Y_i without changing the context R_i. In such a case the order of application of f_i and f_{i+1} might have been inverted. This change in the order would have been justified since all symbols rewritten by f_{i+1} are to the right of those rewritten by f_i.

This derivation is **canonic** because if **C1** or **C2** is satisfied for all derivations then the possible choice of every f_i was unique.

An indication of how the context-sensitive rightmost derivation is used to handle ambiguity in the parsing procedure is given below. Let W_i be the string under derivation. Candidate rules are those whose left arguments[18] are substrings of W_i. Consider two candidate rules:

$\mathbf{f_i}$ $L_iC_iR_1$ —> $L_iD_iR_i$
$\mathbf{f'_i}$ $L'_iC'_iR'_i$ —> $L'_iD'_iR'_i$
given $X_iL_iC_iR_iY_i = X'_iL'_iC'_iR'_iY'_i = W_i$

The selection criterion is: f_i will have priority over f'_i if one of the following conditions (in this order) is satisfied:

(D1) $|X_iL_iC_i| > |X'_iL'_iC'_i|$
(D2) $|X_iL_iC_iR_i| > |X'_iL'_iC'_iR'_i|$
(D3) $|L_iC_iR_i| > |L'_iC'_iR'_i|$
(D4) $i > i'$

It can be proved that, if **D1** is not satisfied (i.e. $|X_iL_iC_i| = |X'_iL'_iC'_i|$), **D2** is no longer relevant and the ambiguity between f_i and f'_i may therefore be handled by **D3** and **D4**. [Bel 1987a:8] **D3** makes a decision on the basis of the length of the left arguments of the two rules, and **D4** is a final arbitrary decision that takes into account the order in which the rules appear in the grammar. In the BP membership test, rules are tried from the bottom to the top of the grammar.

To save computation time, **D3** is not evaluated by the inference engine, so that the following partial ordering of rules is imposed on the grammar:

---

[18] Since we are considering the dual grammar, these are the right arguments of original rules.

"Chunk" rule

> In a BP subgrammar, the right argument of rule f_i may not be a substring of the right argument of f_j such that $j < i$.

This conforms to the intuitive statement that "large chunks should be recognized first". Rules in the subgrammar shown in §3.1 should therefore be arranged in either way:

(1) V1	—> dha	(1) V1	—> dha	(1) V2	—> tr kt
(2) V3	—> dha ge na	(2) V2	—> tr kt	(2) V1	—> dha
(3) V2	—> tr kt	(3) V3	—> dha ge na	(3) V3	—> dha ge na

In a practical implementation, this partial ordering of rules may be computed by the machine when exiting the rule editor.

3.3 Templates

Variations can only be parsed when represented along with their pattern delimiters, homomorphic markers, etc. However, the Bol Processor is meant to perform membership tests on large amounts of data comprising examples to which it is not always easy to assign a structure, as for instance the variation analyzed in appendix 2. Understandably, one could expect the machine to complete the missing information on the basis of structural knowledge contained in pattern rules of the grammar. To this effect, the inference engine of the Bol Processor has been given the ability to generate **templates** from a grammar, i.e. a list of possible structures in which dots are used to represent the locations of imaginary unitary terminal symbols. For instance, the templates generated by the first three rules of the grammar Fig.9 are shown in Fig.11. It may be noticed that each template contains exactly sixty-four dots since all variations have a metric duration of sixteen beats in stroke density 4.

```
[1]  (=.....)(=.....).....*(:.....)(:.....).....
[2]  (=.....).....*(:.....).....
[3]  (=.....).....*(:.....).....
```

Fig.11 Templates

Templates are stored along with the grammar file. When analyzing a variation the Bol Processor attempts to superimpose it on each template in turn. The membership test is tried whenever the variation matches a template. (See appendix 2) Several templates may lead to successful membership tests, thereby pointing at structural ambiguity.

4. Controlling the generation process

We present a few additional features of Bol Processor grammars that were introduced to help the musicologist formalize certain musical ideas in a straightforward and intuitive manner. These are meant to control the selection of candidate rules in generation, although this selection is also taken into consideration by the parsing module.

Other features allowing a control of the position of derivation have been discussed at length in [Kippen & Bel 1989a].

4.1 Negative context

Negative context is a practical way of writing a rule when all but one variable/terminal symbol are allowed as context. See for instance rule [28] in subgrammar 5, appendix 1:

[28] #ti V V V V —> #ti ti dha tr kt

This rule means that "V V V V" may be rewritten as "ti dha tr kt" only if it is not preceded by "ti". This reflects the idea that it is not acceptable to duplicate "ti" in a sequence (both for technical and aesthetic reasons). Procedures for matching and rewriting expressions with (possibly several) negative context(s) are described in detail in [Bel 1990c:55-60].

4.2 Wild cards

Wild cards are metavariables notated "?" in BP syntax. These are used by the inference engine when it looks for candidate rules in the generation or parsing process. A wild card may be matched with any variable or terminal symbol. In subgrammar 6, appendix 1, wild cards are used to define the locations of "fixed chunks".

4.3 Special structural symbols

Special symbols like "+" and ";" are used as position markers in variations. For instance, in the grammar of appendix 1, ";" indicates the end of a variation while "+" is the end of a line of four beats. These symbols appear are used as contexts for controlling the selection of candidate rules, see for instance subgrammar 3. Special symbols also appear in templates. It may be noticed that templates [16] and [20] are identical except that [16] contains a "+" marker after forty-eight dots, i.e. at the end of the third line of the variation. (Each line represents four beats of the metric cycle, see §1.3.) When parsing a variation, therefore, template [16] assumes that the thirteenth beat should fall on the beginning of a word, which is not the case with template [20].

4.4 Stochastic production

Since the actual sets of correct variations of a *qa'ida* are very large, the only realistic way of checking the generative precision of a grammar is to instruct the Bol Processor to produce randomly chosen variations. If the variation is assessed as correct by the expert, the procedure is invoked again and another variation (sometimes the same one) is generated. The grammar is considered "correct" if it has been in full agreement with the expert over a sufficient number of work sessions.

Since the correctness of a grammar can never be fully assessed in this way — indeed, like musicians themselves, machines may be allowed casual mistakes — it is important to enable the stochastic production process to generate variations from a wide and representative subset of the language. This can be achieved by weighting the decisions of the inference engine. Weights (and their associated probabilities) are used to direct the Bol Processor's production along paths more likely to be followed by musicians. The use of weighted rules resulted in a marked improvement in the quality of the generated music. This went a long way towards solving the problem of musical credibility encountered in earlier experiments with BP1, a problem that arose from the complete randomness of the generative process.

The stochastic model in Bol Processor is inspired from probabilistic grammars/automata, [Booth & Thompson 1973] the difference being that a **weight** rather than a probability is

attached to every rule. The probability of a rule is computed each time it is a candidate in a generation process. (Candidate rules are those whose left argument matches a substring of the string under derivation.) Before any derivation, the inference engine calculates the sum *W* of weights of all candidate rules. If the weight of a candidate rule is 0 then its probability remains 0; in any other case its probability is the ratio of its weight to the sum *W*. Consider for example the set of rules

[1]	<100>	V3 —>	dhagena
[2]	<100>	V3 —>	dhatrkt
[3]	<50>	V3 —>	dha–
[4]	<5>	V3 —>	dhati-

in which the sum of weights is W = 100+100+50+5 = 255. The probability of choosing candidate rule [4] in the derivation of a string containing "V3" is therefore

$$\frac{5}{255} = 0.0196$$

In some context-free grammars — those that fulfil a "consistency" condition defined by Booth & Thompson [1973:442-447] — weights may also be used for computing the probability of occurrence of each variation generated by the grammar. Grammars in the format shown in Fig.6 are consistent for any weight assignment. This probability is displayed by Bol Processor BP1 for each variation which has been generated or parsed, thereby yielding a graduation of its acceptability. Another remarkable feature of consistent grammars is that rule probabilities can be inferred from a subset of the language. [Maryanski & Booth 1977:525] This leads to an interesting method for weighting the rules — even in inconsistent grammars as demonstrated in [Kippen & Bel 1989a] with the *qa'ida* of appendix 1. The method is the following: a grammar is given along with a sample set of the language that it recognizes (for instance variations taken from a performance of an expert musician). Let all rule weights be set to 0; then analyze every variation of the sample set, incrementing by one unit the weights of all rules used in the parse.[19] Rules that have not been used in the parse of the sample set may then be scrutinized to check whether they are incorrect or whether they point to unexplored parts of the language. To this effect, their weights are set to a high value so that the Bol Processor is likely to select them and generate variations that may then be assessed.

5. Bol Processor BP2

We now introduce the syntactic model of the new version of Bol Processor (BP2).[20] Other important features, such as polymetric structures and the representation of sound-objects, are presented in another chapter of this volume.

Since BP2 is meant to be part of a computer environment for rule-based composition, the grammar format has been (and is still being) extended on the basis of requests formulated by composers. These may be based on well-defined compositional ideas, or simply exploratory.

---

[19] Unlike the algorithm by Maryanski and Booth [1977], this method works on arbitrary sample sets even if some rules are not used in the parse.

[20] A prototype version of BP2 for Macintosh computers is available as shareware from authors.

Under such conditions, the correctness of a grammar is assessed exclusively on the basis of the music it generates. The composer may instruct the machine to generate many variations and then select those he/she wants to keep. Selected item may even be edited further using a music sequencer. Only in improvisational situations is a grammar expected to generate only "good" variations. For this reason we did not feel it necessary to implement a parsing module in BP2. This allows the user complete freedom as to the grammar format that will best reflect his/her intuitions, even if the grammar is undecidable.

5.1 Metavariables

Metavariables are a set of ordered tokens notated "?1", "?2", etc. Each may match a variable or a terminal symbol. If a rule like

```
?1 ?1 ?2 ?3        —> ?1 ?3 ?2 ?1
```

is applied, BP2 scans the string under derivation looking for a sequence of four variables or terminal symbols in which the first two occurrences are identical, e.g. "A A B C"; it then rewrites it swapping the second and fourth occurrences, i.e. "A C B A". Metavariables are local to the rule in which they appear. A typical application is proposed in appendix 3.

5.2 Remote context

Formal (e.g. Chomsky-type) grammars make it difficult (although theoretically possible) to control productions on the basis of a "remote context", i.e. the occurrence of a string located anywhere to the left or right side of the derivation position. Therefore a special syntax of remote contexts is available in BP2.

Remote contexts are represented between ordinary brackets in the left argument of a rule.[21] For instance, a rule like

(a b c) X Y (c d) —> X e f

means that "X Y" may be rewritten as "X e f" only if "a b c" is found somewhere before "X Y" in the string under derivation, and "c d" somewhere after "X Y". Note that "X" itself is a left context in the sense of conventional generative grammars.

A remote context may contain any string in BP syntax, including string patterns and metavariables. (See for instance appendix 3) It may also be negative. For instance,

#(a b c) X —> c d e

means that "X" may be rewritten as "c d e" only if not preceded by "a b c" in the string under derivation.

[21] These brackets are distinct from pattern delimiters that contain either "=" or ":".

5.3 Context-sensitive substitutions

Substitutions may be viewed as a "parallel rewriting" of the string under derivation: in a single step, each symbol of the string is replaced with another string of symbols (defined by a rule). All occurrences of the same symbol are replaced likewise. Constant-length substitutions, in which all replacement strings have identical lengths, yield (infinite) sequences whose structure is intermediary between periodicity and chaos. [Allouche 1987] This property is useful for the design of rhythmic structures, as shown by Allouche & Mouret [1988]. A less restrictive formalism that we call "context-sensitive substitution" has been implemented in BP2. Substitution rules can also be weighted, use remote contexts, etc.

For instance, the substitution grammar

Substitution

S	—>	A B B A B B A
A B	—>	a B
B A	—>	B b
A A	—>	c c
A B A	—>	A d A
B B A	—>	B e A
B B	—>	f B

produces "a f e a f e b" in two derivation steps. The first derivation yields "A B B A B B A", which is then derived as illustrated in Fig.12.

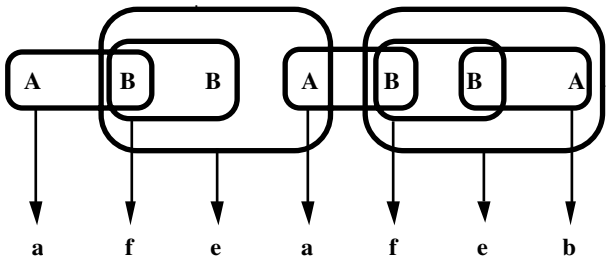

Fig.12 A context-sensitive substitution

In appendix 3, subgrammar 4 uses a context-sensitive substitution to realize the **sonological interpretation**[22] of strings of "A", "B", "C", "D".

[22] See §1 of "*Symbolic and sonic representations of sound-object structures*" in this volume.

5.4 Programmed grammars

When a generative grammar is used to derive a string, rule order is intrinsically predetermined by the availability of variables in the string under derivation. This process is generally non-deterministic because there may be several candidate rules. The idea of a programmed grammar, as suggested by Laske [1973a:365], is to impose an extrinsic ordering of rules reflecting a certain manner in which the generation process is envisaged by the composer. Programmed grammars in BP2 use flags taking positive integer values. Suppose we need to generate random variations of length 12 containing "a" and "b", yet with approximately two times more "a" than "b". We may write:

```
[1]   S                  —>   X X X X X X X X X X X X   /flag1/  /flag1/  /flag2/
[2]   /flag1/  X         —>   a  /flag2/
[3]   /flag2/  X         —>   b  /flag1/  /flag1/
```

The first rule generates "X X X X X X X X X X X X" and three flags. More precisely, "flag1", being generated twice, is set to integer value 2, while "flag2" is set to 1. These are used as conditions for the application of rules [2] and [3]: a rule may be applied only if the value of its condition flag is strictly positive. Each time the rule is applied the value of its condition flag is decremented. For example, rule [2] generates "a" while it decrements "flag1" and increments "flag2" with the effect of validating [3] as a candidate rule. Rule [3] decrements "flag2", generates "b" and increments "flag1" by 2 units, which will make it possible to apply rule [2] twice.

6. Conclusion

Work with the Bol Processor has been beneficial in finding a workable compromise between general formal language models whose mathematical properties are well established (although they often bear little musical relevance) and ad hoc representations fulfilling the requirements of only particular musical tasks. Our problem is not so much finding a universal abstract representation of music but identifying certain forms of musical "thinking" that may be rendered operative in the design of tools for computer-aided music creation, performance, and analysis.

Bol Processor grammars lend themselves to descriptions of music that may be **normative** when dealing with highly constrained systems (e.g. improvisation in traditional music) or **empirical** when applied to the modelling of compositional processes.

We are currently engaged in two projects that originated from this field of investigation. The first is a study of the inference of formal languages under the control of human experts (see [Kippen & Bel 1989b], [Bel 1990a]). The second one is the design of software tools for rule-based music composition/improvisation (see [Bel 1990d]).

References

Allouche, J.P. 1987

Automates finis en théorie des nombres. *Expositiones Mathematicae* 5:239-66.

Allouche, J.P. and Mouret, A. 1988

Libertés non anarchiques, automates finis et champs matriciels. *Colloque International "Structures Musicales et Assistance Informatique"*, Marseille:45-50.

Angluin, D. 1980a

Inductive inference of formal languages from positive data. *Information & Control*, 45, 2:117-35.

1980b

Finding patterns common to a set of strings. *Journal of Computer and System Sciences* 21:46-62.

Bel, B. 1990a

Inférence de langages réguliers. In *Proceedings of "Journées Françaises de l'Apprentissage" (JFA)*, 5-27. Lannion: CNET.

1990b

Grammaires BP pour des airs de sonneurs de cloches. Technical Report, Laboratoire Musique et Informatique de Marseille.

1990c

Acquisition et Représentation de Connaissances en Musique. Thèse de Doctorat en Sciences, Faculté des Sciences de St-Jérôme, Université Aix-Marseille III, Marseille.

1990d

Bol Processor BP2: reference manual. ISTAR France, Marseille.

1987a

Les grammaires et le moteur d'inférences du Bol Processor, Internal Report n°237, GRTC, Centre National de la Recherche Scientifique, Marseille.

1987b

Grammaires de génération et de reconnaissance de phrases rythmiques. In *Proceedings of "6ème Congrès AFCET: Reconnaissance des Formes et Intelligence Artificielle"*, 353-366. Paris: Dunod Informatique.

Booth, T.L. and Thompson, R.A. 1973

Applying Probability Measures to Abstract Languages. *IEEE Transactions on Computers*, C-22, 5:442-450.

Case, J. and Lynes, C. 1982

Machine Inductive Inference and Language Identification. In *Proceedings of the International Colloquium on Algorithms, Languages and Programming (ICALP)*. Springer Verlag:107-115.

Feld, S. 1974

Linguistic Models in Ethnomusicology. *Ethnomusicology*, 18,2:197-217.

Fu, K.S. 1982

Syntactic Pattern Recognition and Applications. Englewood Cliffs: Prentice Hall.

Gold, E.M. 1967

Language Identification in the Limit. *Information and Control*, 10:447-74.

Hart, J.M. 1980

Derivation Structures for Strictly Context-Sensitive Grammars. *Information and Control*, 45:68-89.

Jackendoff, R. and Lerdahl, F. 1982

A Grammatical Parallel between Music and Language. In *Music, Mind and Brain: the Neuropsychology of Music*, ed. M. Clynes, 83-117. New York: Plenum Press.

Jaulin, B. 1980

L'art de sonner les cloches. In *Sons et Musique*. Paris: Belin:100-111.

Kain, R.Y. 1981

Automata Theory: Machines and Languages. Malabar: Krieger.

Kippen, J. 1988

The Tabla of Lucknow: a Cultural Analysis of a Musical Tradition. Cambridge: Cambridge University Press.

1987

An ethnomusicological approach to the analysis of musical cognition. *Music Perception*, 5:173-95.

Kippen, J. and Bel, B. 1989a

Modelling music with grammars: formal language representation in the Bol Processor. In *Computer Representations and Models in Music*, eds. A. Marsden and A. Pople. London:Academic Press. Forthcoming.

1989b

The identification and modelling of a percussion "language", and the emergence of musical concepts in a machine-learning experimental set-up. *Computers and Humanities* 23, 3:199-214

1989c

Can a computer help resolve the problem of ethnographic description? *Anthropological Quarterly*, 62, 3:131-144.

1989d

From word-processing to automatic knowledge acquisition: a pragmatic application for computers in experimental ethnomusicology. *ALLC/ICCH Conference*, Toronto, 6-10 June. Forthcoming.

Laske, O. 1973a

In Search of a Generative Grammar for Music. *Perspectives of New Music*, Fall-Winter 1973, Spring-Summer 1974:351-378.

1973b

On the Understanding and Design of Aesthetic Artifacts. In *Musik und Verstehen*, eds. P. Faltin and H.P. Reinecke, 189-216. Köln: Arno Volk.

Loeckx, J. 1970

The Parsing for General Phrase-Structure Grammars. *Information and Control*, 16:443-464.

Maryanski, F.J. and Booth, T.L. 1977

Inference of Finite-State Probabilistic Grammars. *IEEE Transactions on Computers*, C-26, 6:521-536

[Some anomalies of this paper are corrected in B.R. Gaine's paper "Maryanski's Grammatical Inferencer", *IEEE Transactions on Computers*, C-27, 1, 1979:62-64]

Miclet, L. 1984

Méthodes structurelles pour la reconnaissance des formes. Paris: Eyrolles.

Nattiez, J.J. 1976

Fondements d'une sémiologie de la musique. Paris: Union Générale d'Editions 10-18.

Révész, G. 1985

Introduction to Formal Languages. New York: McGraw-Hill.

1971

Unilateral Context Sensitive Grammars and Left-to-right Parsing. *Journal of Computer and System Sciences*, 5:337-352.

Ruwet, N. 1972

Langage, musique, poésie. Paris: Seuil.

Salomaa, A. 1973
Formal Languages. New York: Academic Press.

Appendix 1 — a BP1 grammar for a *qa'ida*

Below is the grammar of a *qa'ida* taught by the late Afaq Husain Khan of Lucknow. It was elaborated in many work sessions and later revised on the basis of the musician's actual performance. An interesting feature of this improvisation schemata is the possible change of tempo within a single variation. Tempi are notated by way of integer numbers following slashes. For instance, "/4" indicates a stroke density of 4 strokes per beat.[23] A variation with a change of tempo may be found in appendix 2. Rules [18] to [20] in subgrammar 2 indicate a change to density 6 followed with a change back to density 4.

Subgrammar 1

```
[1]   S  —>  /4 (= + S64 ;) Default stroke density is 4
[2]   S  —>  ...  Other possible structures
```

Subgrammar 2

```
[1]   S64  —>  L16 + S48
[2]   S64  —>  L14  S50
[3]   S64  —>  L12  S52
[4]   S64  —>  L24  S40
[5]   S48  —>  M16 + S32
[6]   S48  —>  M14  S34
[7]   S48  —>  M40  O8
[8]   S50  —>  M18 + S32
[9]   S50  —>  M34 + S16
[10]  S50  —>  M18 + * (= + N14 )  O18
[11]  S16  —>  O16
[12]  S50  —>  V10  A'8T  * (= N18 +) + O16
[13]  S50  —>  M20  * (= N14 +) + O16
[14]  S52  —>  V28  A8T  O18
[15]  S52  —>  M20 + S32
[16]  S40  —>  M8 + S32
[17]  S32  —>  * (= + N16 +) + S16
[18]  S32  —>  * (= /6 + V12 /4 A8 +) + O16
[19]  S32  —>  * (= /6 + A16  V8 + /4) + O16
[20]  S32  —>  * (= /6 + V24 + /4) + O16
[21]  S32  —>  * (= /8 + N'16 + A16 + /4) + O16
[22]  S32  —>  * (= + N14 )  O18
[23]  S34  —>  O34
[24]  S34  —>  * (= N18 +) + S16
[25]  S16  —> /8 + A16  O16 ; /4
```

Subgrammar 3

```
[1]   M16  —>  V16
[2]   N18  —>  V18
[3]   (= + L16  —>  (= + A16
[4]   (= + L16  —>  (= + V16
[5]   (= + L14  —>  (= + V10  A'6T
[6]   M14  —>  A'16T
[7]   (= + L14  —>  (= + A16T
[8]   (= + L14  —>  (= + A'16T
[9]   (= + L12  —>  (= + A16TT
[10]  (= + L24  —>  (= + V24
[11]   + M16 + * —> + V10  A'6 + *
```

---

[23] This notation of tempo is elaborated in §6.1 of "*Symbolic and sonic representations of sound-object structures*" in this volume.

[12] + M16 —> + A'16
[13] M8 + * —> A'8 + *
[14] M16 + * —> V8 A'8 + *
[15] M14 —> V8 A'8T
[16] M18 + * —> V10 A'8 + *
[17] M18 + * —> V18 + *
[18] M34 + * —> V28 A'6 + *
[19] M34 + * —> V26 A'8 + *
[20] M20 + * —> V12 A'8 + *
[21] M20 —> V20
[22] M40 —> V8 A'8T V26
[23] * (= /8 + N'16 —> * (= /8 + V16
[24] * (= /8 + N'16 —> * (= /8 + N16
[25] N16 + —> V12 A4 +
[26] N16 + —> V8 A8 +
[27] N16 + —> V10 A6 +
[28] * (= + N16 —> * (= + A16
[29] M14 —> V8 A8T
[30] * (= + N14 —> * (= + V8 A8T
[31] * (= + N14 —> * (= + A16T
[32] N14 + —> V6 A8 +
[33] O8 ; —> A8 ;
[34] O34 ; —> V26 A8 ;
[35] O18 ; —> V10 A8 ;
[36] O16 ; —> V8 A8 ;
[37] + O16 ; —> + A16 ;

Subgrammar 4

[1] V30 —> V V V28
[2] V28 —> V V V26
[3] V26 —> V V V24
[4] V24 —> V V V V V20
[5] V20 —> V V V18
[6] V18 —> V V V16
[7] V16 —> V V V V V12
[8] V12 —> V V V10
[9] V10 —> V V V8
[10] V8 —> V V V6
[11] V6 —> V V V V V V

Subgrammar 5

[1] ? V —> ? -
[2] V —> dha
[3] V V —> trkt
[4] V V —> dheena
[5] V V —> teena
[6] V V —> dhati
[7] V V —> gena
[8] V V —> dhage
[9] + V V —> +tidha
[10] - V V —> -tidha
[11] kt V V —> kttidha
[12] na V V —> natidha
[13] ge V V —> getidha
[14] - V V —> -ti-
[15] kt V V —> ktti-
[16] ge V V —> geti-
[17] na V V —> nati-
[18] V V V —> dhagena
[19] V V V —> teenake
[20] V V V —> dheenage
[21] V V V —> dhatrkt
[22] V V V —> trktdha
[25] V V V V —> tidhagena

[26] V V V V —> dhagedheena
[27] V V V V —> teena-ta
[28] #ti V V V V —> #ti tidhatrkt
[29] V V V V —> dheenagena
[30] V V V V —> teenakena
[31] V V V V V —> dhagenadheena
[32] V V V V V —> dhagenateena
[33] V V V V V —> dhagenadhati
[34] V V V V V —> dhatrktdhati
[35] V V V V V —> dhatidhatrkt
[36] V V V V V —> dhatidhagena
[37] V V V V V V —> dheenagedhatrkt
[38] V V V V V V —> genagedhatrkt
[39] V V V V V V —> dhagedheenagena
[40] V V V V V V —> dhageteenakena
[41] #ti V V V V V V —> #ti tidhagedheenage
[42] V V V V V V —> teenakegenage
[43] V V V V V V V V —> dhagenagenanagena
[44] V V V V V V V V —> dhatidhagedheenagena
[45] V V V V V V V V —> dhatidhageteenakena
[46] V V V V V V V V A8 —> dhatrktdhatidhatrkt A8
[47] V V V V V V V V A'8 —> dhatrktdhatidhatrkt A'8

Subgrammar 6

[1] + ? ? ? ? ? ? ? ? ? ? A'6T —> + ? ? ? ? ? ? ? ? ? ? dhageteena
[2] + ? ? ? ? ? ? ? ? ? ? A6T —> + ? ? ? ? ? ? ? ? ? ? dhagedheena
[3] + ? ? ? ? ? ? ? ? ? ? A'6T —> + ? ? ? ? ? ? ? ? ? ? dhageteena
[4] + ? ? ? ? ? ? ? ? A8T —> + ? ? ? ? ? ? ? ? dhatidhagedheena
[5] A8T ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ; —> dhatidhagedheena ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ? ;
[6] + ? ? ? ? ? ? ? ? A'8T —> + ? ? ? ? ? ? ? ? dhatidhageteena
[7] + ? A'8T —> + ? dhatidhageteena
[8] A'6 + —> dhageteenakena +
[9] A'8 + —> dhatidhageteenakena +
[10] A4 + —> dheenagena +
[11] A6 + —> dhagedheenagena +
[12] A8 + —> dhatidhagedheenagena +
[13] A8 ; —> dhatidhagedheenagena ;
[16] + A16TT —> + dhatidhagenadhatrktdhatidhage
[17] + A16T #ge —>+ dhatidhagenadhatrktdhatidhagedheena #ge
[18] + A'16T #ke —>+ dhatidhagenadhatrktdhatidhageteena #ke
[19] + A16 —> + dhatidhagenadhatrktdhatidhagedheenagena
[20] + A'16 —> + dhatidhagenadhatrktdhatidhageteenakena

Templates generated by this grammar:

[1] /4 (= +.....+.....+ * (= +.....+) +.....;)
[2] /4 (= +.....+.....+ * (= +.....+) + /8 +.....; /4 ;)
[3] /4 (= +.....+.....+ * (= /6 +..... /4.....+) +.....;)
[4] /4 (= +.....+.....+ * (= /6 +.....+ /4) +.....;)
[5] /4 (= +.....+.....+ * (= /8 +.....+.....+ /4) +.....;)
[6] /4 (= +.....+.....+ * (= +.....) .....;)
[7] /4 (= +.....+.....;)
[8] /4 (= +.....+..... * (=.....+) +.....;)
[9] /4 (= +.....+..... * (=.....+) + /8 +.....; /4 ;)
[10] /4 (= +.....+ * (= +.....+) +.....;)
[11] /4 (= +.....+ * (= +.....+) + /8 +.....; /4 ;)
[12] /4 (= +.....+ * (= /6 +..... /4.....+) +.....;)
[13] /4 (= +.....+ * (= /6 +.....+ /4) +.....;)
[14] /4 (= +.....+ * (= /8 +.....+.....+ /4) +.....;)
[15] /4 (= +.....+ * (= +.....) .....;)
[16] /4 (= +.....+.....;)
[17] /4 (= +.....+ /8 +.....; /4 ;)
[18] /4 (= +..... * (=.....+) +.....;)
[19] /4 (= +..... * (=.....+) +.....;)

[20] /4 (= +.....;)

Appendix 2 — parsing a variation

The following is part of the trace showing the parsing of a variation by the grammar above. This demonstrates template matching (see §3.3) and the canonic rightmost derivation (see §3.1). The complete process took less than 2 minutes on an Apple IIc. The variation is:

/4 dhatidhage	nadhatrkt	dhatidhage	dheenatrkt
dhadhatrkt	dhatidha-	dhatidhage	teena-ta
teena-ta	titakena	tatitake	teenakena
/8 dhatidhagenadhatrkt	dhatidhagedheenagena	gena-dhatidhagena	dhatidhagedheenagena

Trying templates:

[1] /4 (=+dhatidhagenadhatrktdhatidhagedheenatrkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+dhatidhagenadhatrktdhatidhagedheenagena;)
Incomplete template: failed…

[2] /4 (=+dhatidhagenadhatrktdhatidhagedheenatrkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+ /8 +dhatidhagenadhatrktdhatidhagedheenagenagena-dhatidhagenadhatidhagedheenagena ; /4 ;) *Acceptable template…*

Trying a membership test on this template:

/4 (=+dhatidhagenadhatrktdhatidhagedheenatrkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+ /8 +A16 gena-dhatidhagenadhatidhagedheenagena; /4 ;)
/4 (=+ A16T trkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+ /8 + A16 gena-dhatidhagenadhatidhagedheenagena; /4 ;)
/4 (=+ A16T trkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+ /8 + A16 gena-dhatidhagena A8 ; /4 ;)
/4 (= + A16T trkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+ /8 +A16 gena-dhatidhagena A8 ; /4 ;)
/4 (= + A16T trkt+dhadhatrktdhatidha- A'8T -ta+*(=+teena-tatitakena A8+)+ /8 +A16 gena-dhatidhagena A8 ; /4 ;)
/4 (= +A16T trkt+dhadhatrktdhatidha- A'8T -ta+*(=+teena-tatitakena A8+)+ /8 +A16 gena- VVVVV A8 ; /4 ;)
/4 (= +A16T trkt+dhadhatrktdhatidha- A'8T -ta+*(=+teena-tatitakena A8+)+ /8 +A16 gena VVVVVV A8 ; /4 ;)
/4 (= +A16T trkt+dhadhatrktdhatidha- A'8T -ta+*(=+teena-tatitakena A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trkt+dhadhatrktdhatidha- A'8T -ta+*(=+teena- VVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trkt+dhadhatrktdhatidha- A'8T -ta+*(=+teena VVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trkt+dhadhatrktdhatidha- A'8T -ta+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trkt+dhadhatrktdhatidha- A'8T V ta+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trkt+dhadhatrktdhatidha V A'8T V ta+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trkt+dhadhatrktdhati VV A'8T V ta+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trkt+dha VVVVVVV A'8T V ta+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trkt+VVVVVVVV A'8T V ta+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T VV+VVVVVVVV A'8T V ta+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T VV+VVVVVVVV A'8T V ta+*(=+VVVVVVVV A8+)+ /8 +A16 VVV6 A8 ; /4 ;)
/4 (= +A16T VV+VVVVVVVV A'8T V ta+*(=+VVVVVVVV A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VV+VVVVVVVV A'8T V ta+*(=+VV V6 A8+)+ /8 +A16 V8 A8 ; /4 ;)

/4 (= +A16T VV+VVVVVVVV A'8T V ta+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VV+VVV6 A'8T V ta+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VV+V8 A'8T V ta+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VV+V8 A'8T V ta+*(=+V8 A8+)+ /8 +A16 O16 ; /4 ;)
/4 (= +A16T VV+V8 A'8T V ta+*(=+N16+)+ /8 +A16 O16 ; /4 ;)
/4 (= +A16T VV+M14 V ta+*(=+N16+)+ /8 +A16 O16 ; /4 ;)
/4 (= +L14 VV+M14 V ta+*(=+N16+)+ /8 +A16 O16 ; /4 ;)
/4 (= +L14 VV+M14 V ta+*(=+N16+)+S16 ;)
/4 (= +L14 VV+M14 V ta+S32 ;) *Membership test failed…*

Trying more templates:

…
[5] (=+dhatidhagenadhatrktdhatidhagedheenatrkt+dhadhatrktdhatidha-dhatidhageteena-ta+*(= /8 +teenatatitakenatatitaketeenakena+dhatidhagenadhatrktdhatidhagedheenagena+ /4)+gena-dhatidhagenadhatidhagedheenagena;) *Acceptable template…*

Trying membership test on this template: failed…

Trying more templates:

…
[11] (=+dhatidhagenadhatrktdhatidhagedheenatrktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+ /8 +dhatidhagenadhatrkt dhatidhagedheenagena gena-dhatidhagena dhatidhagedheenagena ; /4 ;) *Acceptable template…*

Trying membership test on this template:

/4 (= +dhatidhagenadhatrktdhatidhagedheenatrktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+ /8 +A16 gena-dhatidhagenadhatidhagedheenagena; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+ /8 +A16 gena-dhatidhagenadhatidhagedheenagena; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakenatatitaketeenakena+)+ /8 +A16 gena-dhatidhagena A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+ /8 +A16 gena-dhatidhagena A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+ /8 +A16 gena-VVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+ /8 +A16 gena VVVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena-tatitakena A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena- VVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+teena VVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhageteena-ta+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhatidhage VVVV+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha-dhati VVVVVV+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha- VVVVVVVV+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhatidha VVVVVVVVV+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trktdhadhatrktdhati VVVVVVVVVV+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T trktdha VVVVVVVVVVVVVVV+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T VVVVVVVVVVVVVVVVVV+*(=+VVVVVVVV A8+)+ /8 +A16 VVVVVVVV A8 ; /4 ;)
/4 (= +A16T VVVVVVVVVVVVVVVVVV+*(=+VVVVVVVV A8+)+ /8 +A16 VVV6 A8 ; /4 ;)
/4 (= +A16T VVVVVVVVVVVVVVVVVV+*(=+VVVVVVVV A8+)+ /8 +A16 V8 A8 ; /4 ;)

```
/4 (= +A16T VVVVVVVVVVVVVVVVVV+*(=+VV V6 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VVVVVVVVVVVVVVVVVV+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VVVVVVVVVVVV V6+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VVVVVVVVVV V8+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VVVVVVVV V10+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VVVVVV V12+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T VV V16+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T V18+*(=+V8 A8+)+ /8 +A16 V8 A8 ; /4 ;)
/4 (= +A16T V18+*(=+V8 A8+)+ /8 +A16 O16 ; /4 ;)
/4 (= +A16T V18+*(=+N16+)+ /8 +A16 O16 ; /4 ;)
/4 (= +A16T M18+*(=+N16+)+ /8 +A16 O16 ; /4 ;)
/4 (= +L14 M18+*(=+N16+)+ /8 +A16 O16 ; /4 ;)
/4 (= +L14 M18+*(=+N16+)+S16 ;)
/4 (= +L14 M18+S32 ;)
/4 (= +L14 S50 ;)
/4 (= +S64 ;)
 S    Membership test successful...
```

Trying more templates, etc...

Appendix 3 — a BP2 grammar

This grammar is inspired by tintinnabulation, the art of ordering peals of church bells in England [Jaulin 1980]. The grammar is expected to build a sequence chaining distinct permutations of four sounds: “A”, “B”, “C” and “D”. All acceptable changes from one permutation to the next are listed in subgrammar 2: these rules restrict changes of positions of a given sound yielding a structure in which an average periodicity of 4 is suggested but never clearly shown. Negative remote contexts make sure that the permutation newly generated is occurring for the first time. Variable “Cut” is used to separate permutations. When no further rule in subgrammar 2 is applicable, the inference engine jumps to subgrammar 3 in which all remaining variables (except of course “A”, “B”, “C”, “D”) are erased. Subgrammar 4 uses a context-sensitive substitution to replace variables with notes. Following standard solfège, “re4” stands for “D octave 4”. Notes proposed here are arbitrary pitches unrelated to the traditional tuning of bells.

A detailed presentation of this grammar and its variants is available in [Bel 1990b].

Subgrammar 1

```
S —> A B C D   Cut   B A C D   X12
S —> A B C D   Cut   A C B D   X23
S —> A B C D   Cut   A B D C   X34
S —> A B C D   Cut   B A D C   X1234
```

Subgrammar 2

```
#(?1 ?2 ?4 ?3) ?1 ?2 ?3 ?4  X12 —> ?1 ?2 ?3 ?4   Cut   ?1 ?2 ?4 ?3   X34
#(?2 ?1 ?4 ?3) ?1 ?2 ?3 ?4  X12 —> ?1 ?2 ?3 ?4   Cut   ?2 ?1 ?4 ?3   X1234
#(?2 ?1 ?4 ?3) ?1 ?2 ?3 ?4  X34 —> ?1 ?2 ?3 ?4   Cut   ?2 ?1 ?4 ?3   X1234
#(?2 ?1 ?3 ?4) ?1 ?2 ?3 ?4  X34 —> ?1 ?2 ?3 ?4   Cut   ?2 ?1 ?3 ?4   X12
#(?2 ?1 ?4 ?3) ?1 ?2 ?3 ?4  X23 —> ?1 ?2 ?3 ?4   Cut   ?2 ?1 ?4 ?3   X1234
#(?2 ?1 ?3 ?4) ?1 ?2 ?3 ?4  X1234 —> ?1 ?2 ?3 ?4   Cut   ?2 ?1 ?3 ?4    X12
#(?1 ?3 ?2 ?4) ?1 ?2 ?3 ?4  X1234 —> ?1 ?2 ?3 ?4   Cut   ?1 ?3 ?2 ?4    X23
#(?1 ?2 ?4 ?3) ?1 ?2 ?3 ?4  X1234 —> ?1 ?2 ?3 ?4   Cut   ?1 ?2 ?4 ?3    X34
```

Subgrammar 3

```
Cut    —> λ
X12    —> λ
X23    —> λ
X34    —> λ
X1234 —> λ
```

Subgrammar 4: ***sonological interpretation***
(Context-sensitive substitutions)

```
A B —> do3 B
A #B —> do4 #B
B —> sol4
C —> re5
D A —> mi4 A
D #A —> mi5 #A
```

A sequence generated by this grammar is:

do3 sol4 re5 mi5	sol4 **do4** mi5 re5	sol4 mi4 **do4** re5	mi5 sol4 re5 **do4**
mi5 re5 sol4 **do4**	re5 mi4 **do4** sol4	re5 **do4** mi5 sol4	**do4** re5 sol4 mi5

The pseudo-periodicity of all occurrences of “do”, “sol”, “re” and “mi” is clearly visible in this sequence. Tabulations delimit permutations.

Index