Modelling improvisatory and compositional processes

Bernard Bel

Journal of Language Design

Modelling improvisatory and compositional processes

Length: 6625 words

Author: Bernard Bel

Affiliation/address:

Groupe Représentation et Traitement des Connaissances
Centre National de la Recherche Scientifique
31, ch. J. Aiguier, F-13402 Marseille Cedex 9
E-mail: bel@grtc.cnrs-mrs.fr   Fax (033) 91 71 08 08

Abbreviated title (47 chars):

`Modelling improvisatory and compositional processes`

Modelling improvisatory and compositional processes

Abstract

An application of formal languages to the representation of musical processes is introduced. Initial interest was the structure of improvisation in North Indian *tabla* drum music, for which experiments have been conducted in the field as far back as 1983 with an expert system called the Bol Processor, BP1. The computer was used to generate and analyze drumming patterns represented as strings of onomatopeic syllables, bols, by manipulating formal grammars. Material was then submitted to musicians who assessed its accuracy and increasingly more elaborate and sophisticated rule bases emerged to represent the musical idiom.

Since several methodological pitfalls were encountered in transferring knowledge from musician to machine, a new device, named QAVAID, was designed with the capability of learning from a sample set of improvised variations supplied by a musician. A new version of Bol Processor, BP2, has been implemented in a MIDI studio environment to serve as a aid to rule-based composition in contemporary music. Extensions of the syntactic model, such as substitutions, metavariables, and remote contexts, are briefly introduced.

Keywords

Formal grammars, pattern languages, knowledge acquisition, cognitive anthropology, ethnomusicology.

Music, like mathematics but unlike language, is not intelligible unless it is grammatical: its form is its content. As a product of “the unchanging human mind” and body in the context of different cultures, music reflects both man’s biological structure and the patterns of interaction that have been institutionalized as systems of relationships in culture. (John Blacking, [11])

[…] to my mind, any community of musicological practice which excludes from consideration living musicians and restricts itself to accounts of frozen results of musical action, fails to be an inspiring community of inquiry about music. (Otto Laske, [22])

A number of musicologists have attempted to use generative grammars to represent sets of “acceptable” variations of a musical theme. It must be understood that the relevance and reliability of assessments for “acceptability” depend dramatically on musical contexts and individual musicians, so that it is unrealistic to proclaim the existence of a universally valid “musical grammar”.

Drum improvization in North Indian classical music tends to follow a precise system, the rules of which are generally not explicit and conveyed informally to students, much like a natural language [16]. A strong initial motivation of the work presented here has been the challenge of modelling a knowledge relying exclusively on oral transmission and making use of a “speech notation” (onomatopeic syllables representing elementary sounds or finger movements) that could unambiguously be transcribed on a computer. The very first version of the *Bol Processor* software (BP1), in 1982, was a customized wordprocessor allowing a real-time transcription of drumming sequences thanks to a mapping of keyboard strokes to the vocabulary of onomatopeic syllables (*bol*s, from the verb *bolna*, to speak) used by musicians for the transmission (and occasionally the performance) of musical pieces.

Analytical work was then undertaken with players of the *tabla*, a North Indian two-piece drum set, with the objectives of (1) making rules explicit for some compositional types, and (2) checking the consistency of musicians' assessments of correctness in both teaching and performance situations.

In this paper we introduce the basic techniques that have been used for encoding musical pieces and "improvization schemata" called *qa'ida*. It should be kept in mind that the focus of our project gradually shifted from a strict musicological perspective to an enquiry on knowledge acquisition techniques (including automatic inductive generalization) and cognitive aspects of musical expertise in the domain under study. In addition, we are now paying attention to extensions of grammar models that may be used in computer-assisted composition.

Several publications have given detailed accounts of this study. We first focussed on music representation issues that should be carefully studied by musicologists attempting to adapt our methodology to other musical domains [21]. The syntactic models underlying both versions of Bol Processor (BP1 and BP2, running on Apple™ IIc and Macintosh™ respectively) have been described extensively in [6] and [10]. In [17], [18], [20], [28], the methodology of data collection using Bol Processor BP1 has been discussed. Automatic knowledge acquisition using inductive inference techniques (the QAVAID system) is introduced in [19], while [3] addresses the problem of language inference from the point of view of automata theory. Applications of BP2 in a "sound-object" environment are found in [6], [7], [8], and the general concept of "time-object" is introduced in [9].

In this paper we present a few essential aspects of knowledge representation in Bol Processor (BP1 and BP2), addressing readers who may already be conversant with formal language theory and its applications to other artistic disciplines. We support the idea that models based on generative grammars (and their associated automata) have little adequacy to descriptions of real world. Each knowledge domain calls for extensions of the syntactic model, depending on the way data is encoded and how it is used.

1. Music string-encoding and generative grammars

It is assumed that, in some musical systems, elementary musical "objects" may be transcribed with the aid of a non-empty finite set of symbols Vt, namely the **alphabet** of **terminal symbols**. In this context, a music sequence is represented as a string belonging to Vt*, the set of all finite strings over Vt. Any properly defined subset of Vt* is a **formal language**.

Consider for example pieces of *tabla* music transliterated with their onomatopoeic syllables (*bol*s). A terminal alphabet used in some *tabla* compositional types is

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

in which the hyphen indicates a silence (or the prolongation of a resonant stroke). Symbols *tr* and *kt* are shorthand for *tira* and *kita*.

A system for transliterating tabla strokes to non-ambiguous symbolic representations has been presented in [16]. Onomatopeic transliteration is a practice commonly encountered in Indian and African traditional drumming and dance.

Throughout this paper we will refer to Chomsky's hierarchy of **generative grammars** [13]. 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 $S \longrightarrow \lambda$

Type 2 (**context-free**): $|P| = 1$ and $|P| \leq |Q|$ except possibly for the rule $S \longrightarrow \lambda$

Type 3 (**regular** or **finite-state**): every rule has form either

$X \longrightarrow a\ Y$

or $Z \longrightarrow b$, 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 (See [25] p.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.

2. Context-free grammars for *qa'ida*s

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.

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 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.8.

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. Variations are derived from the “theme”, and involve the repetition, permutation, or substitution of bols. Students are taught that changes occurring in the first half of the structure must be reflected in the second; variations, too, are subject to voiced/unvoiced transformations as can be seen in the following three variations (changes have been italicised; hyphens in the third variation represent silences of one unit, though effectively they elongate the preceding syllable):

dhatidhage	nadhatrkt	dhatidhage	dheenagena
dhatrktdha	*tidhagena*	dhatidhage	teenakena
tatitake	natatrkt	tatitake	teenakena
dhatrktdha	*tidhagena*	dhatidhage	dheenagena

dhatidhatr	*ktdhatidha*	*trktdhage*	*nadhagena*
dhatidhage	nadhatrkt	dhatidhage	teenakena
tatitatr	*kttatita*	*trkttake*	*natakena*
dhatidhage	nadhatrkt	dhatidhage	dheenagena

dhagenadha	*trktdhage*	*nadhatrkt*	*dhatrktdha*
tidha-dha	*tidhagena*	dhatidhage	teenakena
takenata	*trkttake*	*natatrkt*	*tatrktta*
tidha-dha	*tidhagena*	dhatidhage	dheenagena

Fig.2 Three variations of the *qa'ida*

Let us now consider only the first lines of a subset of ten simple variations, as shown in Fig.3.

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

This set can easily be described using a type-3 (finite-state) grammar equivalently represented as a **finite automaton**, i.e. a directed graph in which *X*, *Y*, and *Z* are state labels, and *a* and *b* transition labels (see Fig.4). 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*.

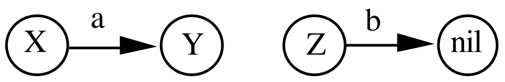

Fig.4 Basic transitions in a finite automaton

We call **finite acceptor** a kind of finite automaton with a univocal mapping of the set of states to the set {"acceptable", "unacceptable"}, thereby suggesting that other mappings can be envisaged (see for instance [1]). 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 acceptor recognizing exactly the ten examples given in Fig.3 could be the one shown in Fig.5.

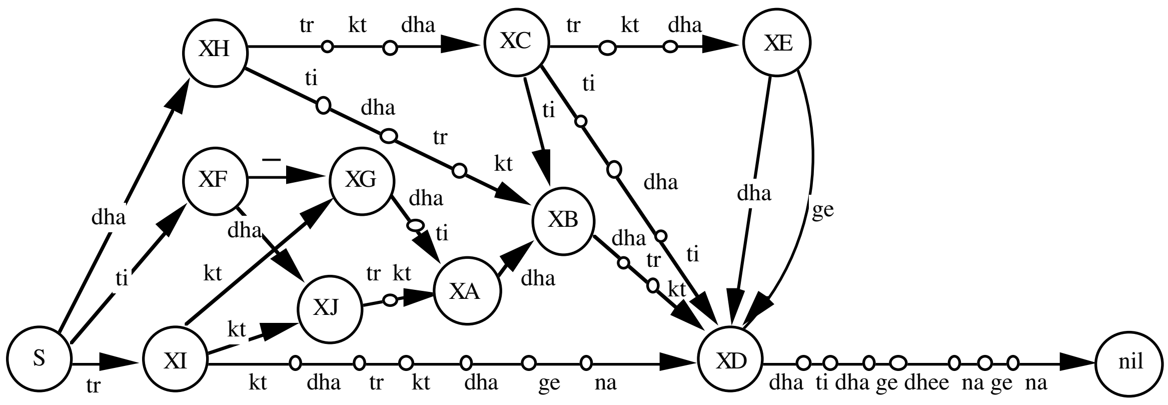

Fig.5 A finite acceptor for the language shown Fig.3

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.3, 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 later. 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.6.

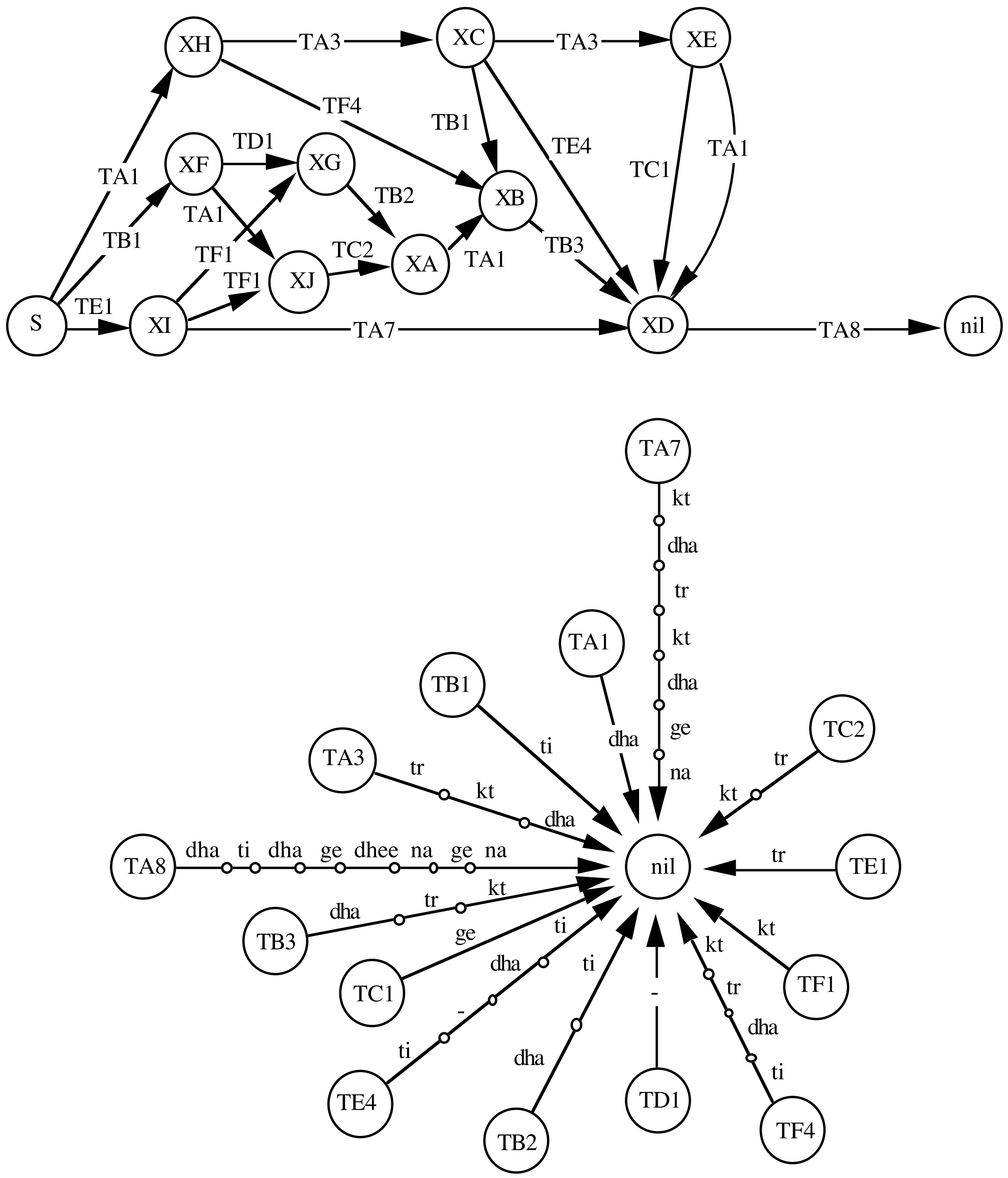

Fig.6 An equivalent "two-layer finite acceptor"

This new acceptor is equivalent to the generative grammar of Fig.7. The mapping of transitions to grammar rules is self-explanatory.

S	—>	TE1 XI
XI	—>	TA7 XD
XD	—>	TA8
XI	—>	TF1 XJ
XJ	—>	TC2 XA
XA	—>	TA1 XB
XB	—>	TB3 XD
XI	—>	TF1 XG
XG	—>	TB2 XA
S	—>	TA1 XH
XH	—>	TF4 XB
XH	—>	TA3 XC
XC	—>	TE4 XD
XC	—>	TA3 XE
XE	—>	TA1 XD
XE	—>	TC1 XD
XC	—>	TB1 XB
S	—>	TB1 XF
XF	—>	TA1 XJ
XF	—>	TD1 XG

TA7	—>	kt dha tr kt dha ge na
TC2	—>	tr kt
TE1	—>	tr
TF1	—>	kt
TF4	—>	ti dha tr kt
TD1	—>	-
TB2	—>	dha ti
TE4	—>	ti - dha ti
TC1	—>	ge
TB3	—>	dha tr kt
TA8	—>	dha ti dha ge dhee na ge na
TA3	—>	tr kt dha
TB1	—>	ti
TA1	—>	dha

Fig.7 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.7 are called **lexical rules**. Their right arguments are chunks of strokes that have been repeated several times in the examples of Fig.3, 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. [24], [26]). 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. In fact, the vocabulary displayed by the grammar Fig.7 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 [19] page 210).

A context-free grammar in the format of Fig.7 may be viewed as a combination of two **transformational subgrammars** corresponding to the two automata shown Fig.6. The term “transformational” is borrowed from formal language theory ([15], page 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.7) 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 a **parsing module** described in detail in [8].

3. A few syntactic extensions

3.1 Pattern rules

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 **string pattern** any element of $(Vn \cup Vt)^*$, 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**.

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*. (See [2])

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. Despite this we felt it would be interesting to combine the representational power of pattern languages (in terms of periodicity) with the versatility of generative grammars, as the latter are rather counterintuitive for the representation of string-patterns. Consider for instance the following grammar proposed by Salomaa ([27], page 12) generating the language derived from string pattern "X X" over terminal alphabet Vt = {a,b}.

```
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**. 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.8 shows the mapping of the corresponding voiced/unvoiced transformation, which stands for all *qa'ida*s using these strokes.

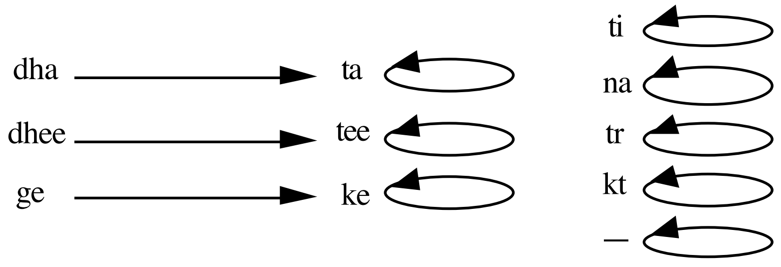

Fig.8 A "voiced/unvoiced" mapping in *tabla* music

This mapping may be extended to any string over the terminal alphabet Vt, yielding a λ-free homomorphism [25]. 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.9)

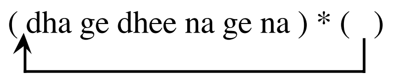

Fig.9 Master-slave assignment pointer

This internal representation is economical in terms of memory space. The algorithm for rewriting assignment pointers in string patterns is presented in [6] (pages 43-45). It applies to multilayered representations as well.

The theme given in Fig.1 may now 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 piece is produced by the grammar shown in Fig.10, 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.10 A grammar with three pattern rules

The same grammar produces many variations, including the three ones shown in Fig.2. Rules defining V8, V16 and V24 may be context-free like the ones in Fig.7.

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

3.2 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:

```
#ti  V4  —>  #ti ti dha tr kt
```

This rule means that *V4* 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 [6] (pages 55-60).

3.3 Wild cards

Wild cards are metavariables notated "?1", "?2", etc. 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. An example will be found in the grammar Fig.11 below.

3.4 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. (These brackets are distinct from pattern delimiters that contain either “=” or “:”) 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. 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.

Below is a typical grammar using remote contexts. This grammar is inspired by **tintinnabulation**, the traditional art of ordering peals of church bells in north of France and England [14]. 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 [4]. Negative remote contexts make sure that the permutation newly generated is occurring for the first time. Permutations are performed on wildcards (“?1”, “?2”…) that may be instantiated as *A, B, C* or *D*. 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* and *D*) are erased. “λ” is the null string. 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 description of this grammar and its variants is available in [4].

```
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 level
SUB (Context-sensitive substitutions)
A B —> do3 B
A #B —> do4 #B
B —> sol4
C —> re5
D A —> mi4 A
D #A —> mi5 #A
```

Fig.11 A grammar with remote contexts

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.

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, [12] 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 in [12] — 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.7 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 [23]. This leads to an interesting method for weighting the rules — even in inconsistent grammars, as demonstrated in [21] (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. 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. Automatic Knowledge Acquisition

Despite the comprehensiveness of the descriptive language for grammatical rules, grammars became too unwieldy when dealing with compositions of middle complexity (in terms of length of structure, variety of syllables, interchangeability of chunks, etc.), not to mention high complexity. Whenever analytical or generative computations blocked owing to the inadequacy of the rule base, then determining what to modify was both difficult and problematic. Since grammars were arranged in hierarchical segments representing notional sequences of musical decision-making processes the modification of one rule could (and often did) have implications for the rest of the grammar. Therefore, simply adjusting one component was tantamount to treating the symptoms of the disorder, not the cause of the breakdown. There was no way of finding out in a sound and systematic manner the most efficient and structurally correct procedure for modifying a grammar; and in experimental situations where musicians were waiting patiently for the researcher to press the magic button and produce ever new and varied improvisations for their delectation (they were indeed fascinated by the technology and deeply interested in the experiments), the researcher was often forced to settle for the simplest solution to the problem — the computational equivalent of a band-aid.

An alternative approach to the problem of knowledge transfer was introduced in 1988 with a system that automated the analytical/generative process, thus effectively circumventing the researcher as "interpreter" of musical knowledge and grammatical rules [3] [19]. This system adapted grammatical inference and machine-learning techniques to extract rules from a given sample of variations (improvisations) on a rhythmic theme, mapping them to an expandable finite-state automaton resembling a network of converging and diverging paths (see Fig.5 supra). The computer system then "generalized" by merging states and constructing new paths within the model, and sought verification of its decisions by a question-answer sequence with the informant (i.e. the provider of the musical samples). This process gave rise to the name of the system: QAVAID (*Question-Answer Validated Analytical Inference Device*), an acronym meaning "grammar" in Urdu, the language of tabla-player informants.

6. BP2 — a tool for computer-aided composition

It was felt that the formal model embedded in BP1 could be expanded to encompass more general musical structures, and in this form could be of some benefit as a tool for rule-based music composition. The new implementation (BP2) deals with "sound-objects" handled (1) at the symbolic level (where each object is represented by an arbitrary symbol, and symbols arranged in strings, tables, trees, etc.) and (2) the lower level of "elementary events". These events are messages dispatched to a sound processor to trigger and control sound synthesis processes. For instance, a conventional note is a simple sound-object represented as a NoteOn/NoteOff sequence in the MIDI format. Objects are ordered on symbolic (virtual) time whereas events are mapped to points on physical time. This approach widely compensates the rigidity of the timing of computer-generated musical pieces.

A major development of the representation model has been "polymetric expressions", thereby meaning (incomplete) string descriptions of concurrential processes. A polymetric expression can be fully determined ("expanded") by an algorithm inferring a strict ordering of objects along symbolic time. For instance, given the information that a sequence of five sound-objects: a, b, c, d, e, should be superimposed on another sequence of three objects: f, g, h, the algorithm would suggest the ordering of events shown Fig.12.

```
        {a b c d e, f g h}          a _ _ b _ _ c _ _ d _ _ e _ _

or      {f g h, a b c d e}          f _ _ _ _ g _ _ _ _ h _ _ _ _
```

Incomplete representation	Complete representation
(polymetric expression)	(computed by algorithm)

Fig.12 Polymetric expression (left) and "expanded" representation (right)

The same algorithm is able to process multilayered polymetric expressions [5].

Sound-objects may be assigned properties that help determining their actual durations and locations on physical time. They may be performed in "striated" time (with regular or irregular beats) or in "smooth" time (with no beat). Durations depend on the "local tempo" and on metrical properties stipulating the acceptable range of object contraction/dilation. In striated time, local tempo is the time interval separating two streaks (beats); in smooth time it is computed from the duration of the object preceding the current one in the sequence.

Object location makes use of "time pivots" (anchoring points) that are first located on time streaks. This pivot-synchronization technique is inspired from Marco Stroppa (IRCAM). The topological properties of each object are constraints stipulating whether or not its time interval may be overlapped, truncated, whether or not it must be contiguous to that of neighbouring objects, etc. Some objects need to be relocated or truncated until all constraints are fulfilled. Constraint-satisfaction is handled by a fast algorithm [8].

Since the constraint-satisfaction algorithm does not rely on specific musical concepts it may be applied to various applications in which similar metrical/topological properties are assigned to "time-objects", such as multimedia performance, robotics or multiple-screen video editing. In the latter application, time-objects are predetermined picture sequences. A string of time-objects is the script of a "story" displayed on a single video monitor. The story may be predefined or edited at the time of the performance. Overlapping time-objects in a story may be displayed as dissolved sequences. The time-setting algorithm makes it possible to build many different scenari relating several stories on different screens [9].

An external control can be exerted on the inference engine, grammars and the interpretation module. Thus, Bol Processor BP2 can be operated and synchronized by instructions received from its MIDI interface. Messages on the different MIDI channels may be used for communicating between machines or for controlling several sound processors. It is possible to perform/repeat an item, modify tempo, adjust rule weights, etc., using a MIDI keyboard and controllers, a MIDI sequencer or even another BP2. Real-time interaction is similar to the situation of several musicians improvizing together while communicating information about parameters like tempo or compositional strategies through conventional (audible or inaudible) messages.

~~Note~~

~~A shareware version of BP2 and a "Quick Start" manual are available from author or may be retrieved electronically by ANONYMOUS FTP or TELNET from INFO.UMD.EDU (128.8.10.29), directory ReadingRoom / NewsLetters / EthnoMusicology / Archive.~~

Acknowledgement

Many musicological and methodological points in this paper have been elicited and written down in collaborative work with ethnomusicologist James Kippen, Faculty of Music, University of Toronto (Canada).

References

[1] J.P. Allouche, *Automates finis en théorie des nombres*, Expositiones Mathematicae 5 (1987) 239-266.

[2] D. Angluin, Finding patterns common to a set of strings, Journal of Computer and System Sciences 21 (1980) 46-62.

[3] B. Bel, *Inférence de langages réguliers*, Proc. *Journées Françaises de l'Apprentissage* (JFA 90), Lannion, France (1990) 5-27.

[4] B. Bel, *Grammaires BP pour des airs de sonneurs de cloches*, unpublished (1990).

[5] B. Bel, Time in Musical Structures, Interface 19 (2-3) (1990) 107-135.

[6] B. Bel, *Acquisition et représentation de connaissances en musique*, phD thesis, Université Marseille III (1990).

[7] B. Bel, Bol Processor BP2: Quick Start and Reference Manual, distributed by ISTAR France, 37 rue Boudouresque, F-13007 Marseille (1991).

[8] B. Bel, Symbolic and sonic representations of time-object structures, in: M. Balaban, K. Ebcioglu and O. Laske, eds., Understanding Music With AI (AAAI Press, Menlo Park, 1992).

[9] B. Bel, A fast algorithm for scheduling sequences of elementary tasks, submitted to European Conference on Artificial Intelligence, Vienna (1992).

[10] B. Bel and J. Kippen, Bol Processor grammars, in: M. Balaban, K. Ebcioglu and O. Laske, eds., Understanding Music With AI (AAAI Press, Menlo Park, 1992).

[11] J. Blacking, Ethnomusicology as a key subject in the social sciences, In Memoriam Antonio Jorge Dias 3 (1974), Lisbon (Portugal) 71-93.

[12] T.L. Booth & R.A. Thompson, Applying probability measures to abstract languages, IEEE Transactions on Computers C-22 (5) (1973) 442-450.

[13] N. Chomsky, Aspects of the Theory of Syntax (The MIT Press, Cambridge MA, 1965).

[14] B. Jaulin, *L'art de sonner les cloches*, in: *Sons et Musique* (Belin, Paris, 1980) 100-111.

[15] R.Y. Kain, Automata theory: machines and languages (Krieger, Malabar, 1981).

[16] J. Kippen, The Tabla of Lucknow: A Cultural Analysis of a Musical Tradition (Cambridge University Press, Cambridge, 1988).

[17] J. Kippen, Where does the end begin? Problems in musico-cognitive modelling, Minds and Machines 1 (1992).

[18] J. Kippen and B. Bel, Can the computer help resolve the problem of ethnographic description?, Anthropological Quarterly 62 (3) (1989) 131-144.

[19] J. Kippen and B. Bel, The identification and modelling of a percussion 'language', and the emergence of musical concepts in a machine-learning experimental set-up, Computers and the Humanities 23 (3) (1989) 199-214.

[20] J. Kippen and B. Bel, A pragmatic application for computers in experimental ethnomusicology, in: S. Hockey and N. Ide, series eds., I. Lancashire, guest ed., Research in Humanities Computing I: Papers from the 1989 ACH-ALLC Conference (Oxford University Press, Oxford, forthcoming).

[21] J. Kippen and B. Bel, Modelling music with grammars: formal language representation in the Bol Processor, in: A. Marsden and A. Pople, eds., Computer Representations and Models in Music (Academic Press, London, 1992) 207-238.

[22] O. Laske, An epistemic approach to musicology, in: G. Haus, ed., E. Nissan, series ed., Music Processing (JAI Press, forthcoming).

[23] F.J. Maryanski and T.L. Booth, Inference of Finite-State Probabilistic Grammars, IEEE Transactions on Computers C-26 (6) (1977) 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.]

[24] J.J. Nattiez, *Fondements d'une sémiologie de la musique* (Union Générale d'Editions 10-18, Paris, 1976).

[25] G. Révész, Introduction to Formal Languages (McGraw-Hill Computer Science Series, New York, 1985).

[26] N. Ruwet, *Langage, Musique, Poésie* (Seuil, Paris, 1972).

[27] A. Salomaa, Formal Languages (Academic Press, New York, 1973).

[28] B. Vecchione, *Musique et modèles: approche d'une typologie*, *Analyse Musicale* 22 (1991) 13-29.