Pattern grammars in formal representations of musical structures

Bernard Bel

PATTERN GRAMMARS IN FORMAL REPRESENTATIONS OF MUSICAL STRUCTURES

Bernard Bel[1]
Groupe Représentation et Traitement des Connaissances (GRTC)
Centre National de la Recherche Scientifique
31, chemin Joseph Aiguier, F-13402 Marseille Cedex 9

Abstract

This paper introduces several formal models of pattern representation in music. *Polyvalent multimodal grammars* describe partially overlapping sound events as found in polyphonic structures. *Bol Processor grammars* are characterizations of sequential events in terms of substring repetitions, homomorphisms, etc. Parsing techniques, stochastic production and recent developments of BP grammars are briefly described.

Keywords: pattern languages, formal grammars, stochastic automata, music analysis

> *La musique est le suprême mystère des sciences de l'homme, celui contre lequel elles butent et qui garde la clé de leur progrès.*
>
> (Lévi-Strauss 1964:27)

Music representation is a vast subject covering two separate topics: sound models and musical structures. Although in the field of computer-aided composition much work has been devoted to new sound generation techniques, many studies of structural models are still limited to the analysis of Western tonal music. Beyond this domain (particularly in ethnomusicology) there has been a tendency to borrow concepts from linguistics on the basis of assumed parallels between language and traditional musical systems. Feld's provocative paper (1974), however, introduced to ethnomusicologists an epistemological dilemma which resulted in the virtual rejection of purely abstract speculations that Feld 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).

'Precomputer' studies have been limited to descriptions of 'frozen' data, thereby ignoring models that would produce musical scores:

---

[1] Bernard Bel is a computer scientist with a background in electronics. Since 1979 he has been collaborating with anthropologists, musicologists and musicians on a scientific study of North Indian melodic and rhythmic systems. In 1981 he built a sophisticated real-time *Melodic Movement Analyser* (MMA) with the aid of which he developed automatic transcription and analysis of raga music (in collaboration with Dr. Wim van der Meer). In 1986 he joined GRTC, an AI laboratory of the *Centre National de la Recherche Scientifique* in Marseille. At present he is working on algorithms for automatic rule generation derived from hypotheses on training methods in traditional *tabla* (in collaboration with ethnomusicologist Dr. Jim Kippen). He is also involved in research on computer music at the *Laboratoire Musique et Informatique de Marseille* (MIM).

> Our theory of music is...based on structural considerations; it reflects the importance of structure by concerning itself not with the composition of pieces but with assigning structures to already existing pieces.
> (Jackendoff & Lerdahl 1982:85)

In fact, computers make it possible not only to check analytical models against large data sample sets but also to adjust grammars with a view to improving the machine's ability to compose music or to assist a human composer by suggesting channels of 'musical thinking' unknown to him/her. This is — broadly speaking — the 'expert system' approach (Kippen & Bel 1989c)[2]. Several teams are working on Western tonal music along these lines, trying to set-up validations of the structural models proposed by Ruwet and Jackendoff & Lerdahl (Camilleri 1989).

Research in analysis and composition of music by means of computers implies the choice of adequate representational models of music. This paper deals with issues relevant to both approaches. Two models of musical patterns imbedded in formal grammars are introduced: *Polymodal Multivalent Grammars* (PMG) and *Bol Processor* (BP) grammars. PMGs have been proposed by Vecchione (1978ff) as a general syntactic model of music which lends itself to a paradigmatic analysis based on *resemblance*, and may therefore be used as an aid to music composition[3]. BP grammars have been designed by Kippen & Bel (1984) for an experimental study of improvisation/evaluation schemata in North Indian *tabla* drumming. The Bol Processor was first implemented in assembly language on the Apple IIc for fieldwork with traditional musicians[4]. Recent developments of this model in the light of contemporary music theory and practice are briefly introduced in §2.5[5].

A detailed description of the BP's modus operandi is found in Kippen & Bel (1989a). The problem of transferring knowledge from informants to machines using automatic rule generation — the present focus of the author's project — is discussed in Kippen & Bel (1989b,c,d).

1. Pattern grammars, languages, and musical structures

1.1. Definitions and notations

It is assumed that elementary musical events ('atoms') may be transcribed with the aid of a finite set of symbols, namely the *alphabet* Vt of *terminal symbols*. For example, many pieces of *tabla* music belonging to the *qa'ida* repertoire may be transliterated with Vt = {*dhin, tin, dhee, tee, dha, ta, ghi, ki, ge, ke, ka, na, ne, ra, -*}, in which the hyphen denotes a silence[6]. Each symbol is a quasi-onomatopoeic mnemonic representation of a musical event (stroke or *bol*[7]). In the same way, any finite set of elementary sound events (e.g. notes) may be viewed as a terminal alphabet for music.

---

[2] Another approach, which will not be discussed here, is the discovery of a system for music composition or improvisation from musical scores (Laske 1984, Kippen & Bel 1989b).

[3] Vecchione's model takes in consideration polyphonic time structures and other features like modes of space location, modes of significance, modes of voice/instrument processing, modes of performance, etc.

[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). Subsequently, Kippen's work was also supported by the *Leverhulme Trust* in the UK.

[5] An outcome of discussions in the 'OC' study group involving composers and scientists at *Laboratoire Musique et Informatique* de Marseille (MIM).

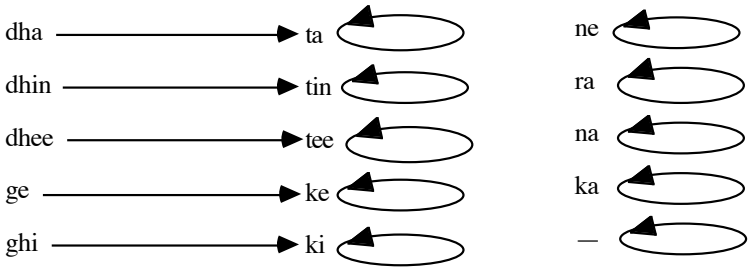
[6] A system of transliterating tabla strokes to non-ambiguous symbolic representations has been proposed by Kippen (1988:xvi-xxiii).

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

A *monodic sequence* is a string on Vt. The set of finite strings on alphabet Vt is notated Vt*. The length of string s is |s|. For instance, |*dhee na ge na*| = 4. The nil string is λ, hence |λ| = 0. The set on non-nil finite strings on Vt is Vt^+. In set notation, $Vt^+ = Vt^* - \{\lambda\}$. The number of elements in a set S (the *cardinal* of S) is notated card(S). ∅ denotes the empty set, hence card(∅) = 0.

To describe languages we may need an enumerable set of *variables* or *non-terminal* symbols Vn such that Vn ∩ Vt = ∅. Throughout this paper, variables will be represented with upper-case characters sometimes followed with numbers denoting paradigmatic variants or metric durations, e.g. X2, TA14, etc...

Tabla strokes are either *open* (voiced) or *closed* (unvoiced). Any sequence of unvoiced strokes may be seen as the closed (*mirror*) image of a sequence of open strokes through a mapping of Vt* to itself. This mapping *mir* is a λ-free homomorphism of Vt* to itself whose reduction to Vt is an idempotent mapping of Vt to itself, for instance:



ta is notated as mir(*dha*), and *teenakena* as mir(*dheenagena*).

Homomorphisms are suitable for describing any structural relationship that is built on transformations of the terminal symbols, e.g. transposition, octave shift, etc. in tonal music.

1.2. Pattern grammars

Leaving aside its parallels with language, music may be seen as an algorithmic (non-random) signal sequence. Since automatic (i.e. automata-generated) sequences lie somewhere between periodicity and chaos, yet closer to periodicity (Allouche 1987:264), a workable hypothesis is one that envisages the description of all kinds of musical events as generated by automata, or, equivalently, formal grammars.

Yet both the atomistic theory of music — "la mélodie n'est autre chose qu'une succession de sons déterminés" (Vincent d'Indy) — and the classical 'scheme' model — "ABA is a Lied pattern" — are one-level conceptualizations of music. The atomistic conception, which was followed by Schönberg, Webern, etc.[8], resulted in attempts to proclaim *set theory* as the foundation of tonal (cf. Babbitt) and atonal (cf. Forte) music. As put by Roads (1984:10):

> With this approach, abstract properties — such as invariance relationships under formal operations — may be studied, while properties which are not amenable to such an approach are often ignored (e.g. timbral and spatial relationships, articulation factors, clusters and clouds of sound, dramatic and expressive qualities, and so on).

[8] later by followers of the stochastic approach: Betty Shannon (1951), Meyer (1956-7), Coons & Kraehenbuehl (1958), Fuchs (1961), Moles (1958), Xenakis (1963), Philippot (1970)

Ruwet (1966,1972:100-34) and Nattiez (1976:239-78) — both involved in surface, Schenkerian and prosodic analysis — and later Jackendoff & Lerdahl (1983)[9], introduced linguistic/semiotic descriptions in which several layers of music constituents may be assigned to a single (monodic or homophonic) sequence.

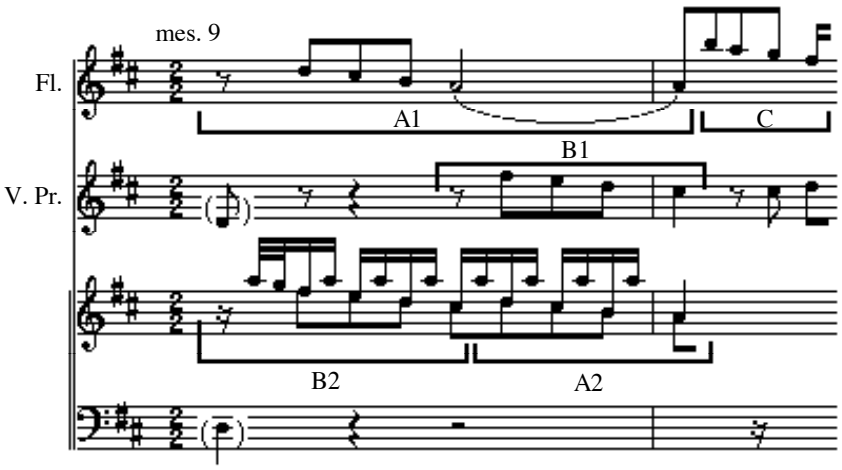
Unfortunately, these sequential models have been unable to account for the structural organization of overlapping constituents as found in polyphonic musical structures:

> At the present stage of development of the theory, we are treating all music as essentially homophonic (...) For the more contrapuntal varieties of tonal music, where this condition does not obtain, our theory is inadequate. We consider an extension of the theory to account for polyphonic music to be of great importance. However, we will not attempt to treat such music here except by approximation. (Lerdahl & Jackendoff 1983:37)

To achieve this, it is necessary to disconnect the set of *musical constituents* from the set of the *time-span intervals* which support them[10]. The set of constituents is always structured as a *tree* (a hierarchy of partitions), whereas the set of time-span intervals is structured as an *join-semilattice* (Birkhoff 1984:22) which may or may not be tree-shaped. Musical structure is therefore determined by a morphism between these two structures: a bijective morphism defines a sequential structure whereas injective morphisms define non-sequential structures (Vecchione 1984:139ff, 1985:207ff).

The idea of dissociating musical constituents from their time-span intervals comes in response to Xenakis' (1963, 1972:57) suggestion to separate *out-time structures*, the *structure of time* and the resulting *in-time structure*, i.e. the Cartesian product of the former two.

Consider for instance the 9th measure (2nd theme of the first movement) in J.S. Bach's fifth *Brandenburg Concerto* in D major:



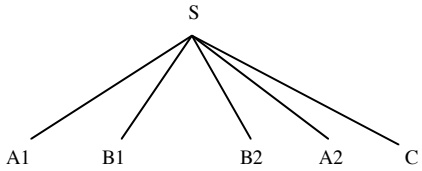
(Vecchione 1984:137)

The set of constituents may be structured as a tree

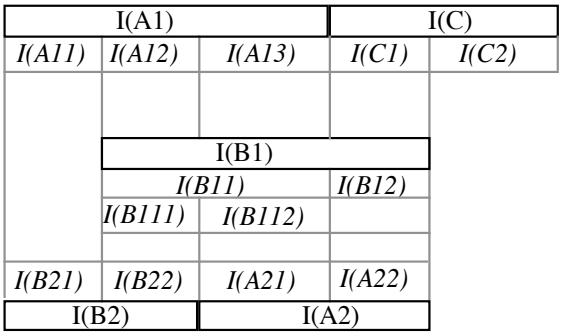
---

[9] a model recognising four components: metrical structure, grouping structure, time-span reduction and Schenkerian prolongational reduction — the latter was criticised and expanded by Deliège (1983-4).

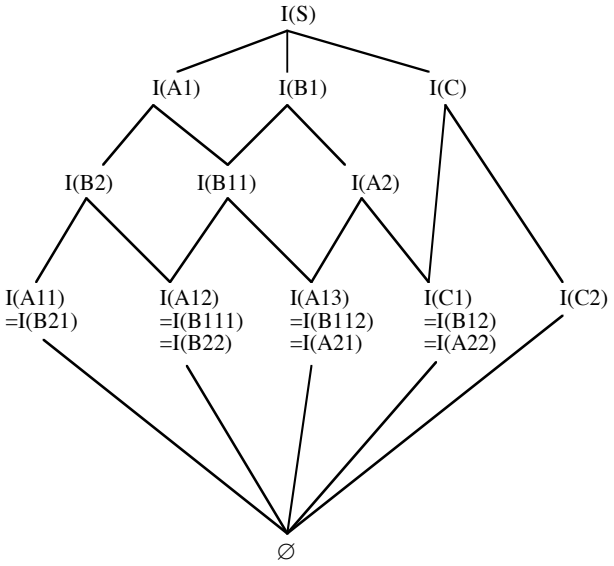
[10] This hypothesis has not been considered by all researchers in the field. See for instance Chemillier (1986:2, 1987,2:380ff). Interestingly, Xenakis introduced three structures in his early work (1963:190-1,200): in-time, out-time structures and structure of time, the former being a mapping between the latter two. Yet in 1972 he did not deal any more with the structure of time.



corresponding to the following time-span intervals



which are structured as a lattice:



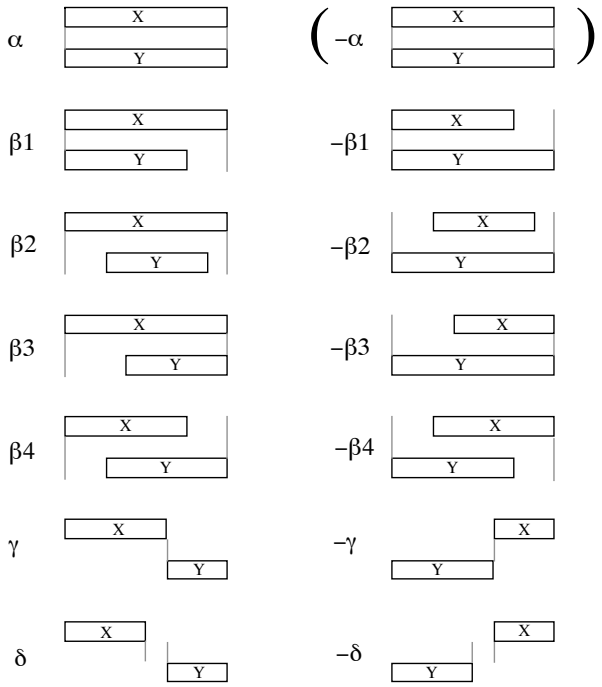
Note that if the universal lower bound '∅' is not represented, the structure becomes a join-semilattice with universal upper bound I(S), the *minimal elements* of which are arranged in a strict sequence: I(A11)=I(B21), I(A12)=I(B111)=I(B22), I(A13)=..., I(C2).

The weakness of this representation lies in the fact that some of the time-span intervals have been chopped into smaller intervals, e.g. I(B11), I(B12) that do not necessarily support meaningful musical constituents. The same problem arises in set representations, e.g. Chemillier & Timis (1988) who developed a formal language for polyphonic music based on *trace languages* and *partially commutative free monoïds* (Mazurkiewicz 1984, Chemillier 1987).

Another aspect of the present research, therefore, is an attempt to deal with time as a *topological* entity: considering non-Euclidian metric durations is necessary when dealing with contemporary music, most extra-European systems, and European music prior to the 15th

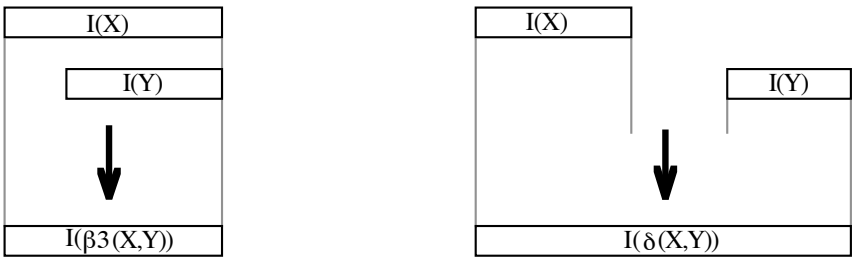
Century. Wiggins & al. (1988) pursued this line of thinking in a logic representation of musical constituents, defining an ordered set of 'times' and a linearly ordered commutative group 'duration'. Another fruitful approach is based on a single set of open intervals (of rational numbers) carrying two binary relations: inclusion and precedence (Van Benthem 1983:58-79).

In the linguistic/semiotic model, applying a rule in a formal grammar amounts to a step in string *concatenation*. In the description of polyphony, however, thirteen distinct operations are possible (Van Benthem 1984:70, Vecchione 1984:150ff):



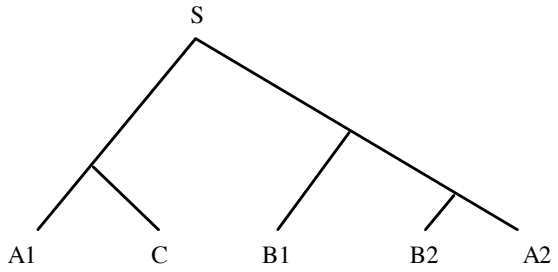
Comparisons of time-span intervals may be worked out by calculus or with reference to the fact that that the ear may perceive event X as starting or ending before, simultaneously, or after event Y.

If I(X) and I(Y) are the time-span intervals supporting X and Y respectively, operations α, β1, ..., δ yield a constituent X∪Y, the time-span interval of which is the smallest interval covering both I(X) and I(Y), e.g.



Note that γ is a strict sequence and both α and −α strict superimpositions. The latter two are identical.

If a binary-tree structure is assigned to constituents A1, A2, B1, B2 and C in the Bach example, e.g.



then S is the result of the following operation:

S = β1(γ(A1,C),-β3(B1,γ(B2,A2)))

Obviously this description does not retain the entire topological information: the combination of A1 and B1 might as well have been β4(A1,B1) or β2(A1,B1).

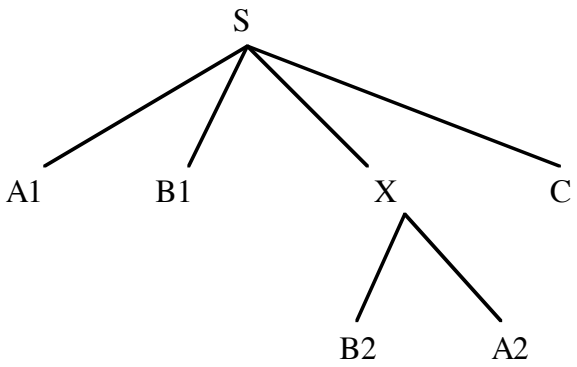
If more information must be made explicit, then alternative formulae, built on all the supposingly 'meaningful' binary trees — in terms of musical analysis — may reduce ambiguity:

S = β1(γ(A1,C),-β3(B1,γ(B2,A2))) = -β4(α(β4(A1,B1),γ(B2,A2)),C) = ... (up to 5! = 120 formulae).

Vecchione (1984:185ff) implicitly used time operators as *predicates* (i.e. Boolean operators) in his syntactic model ('Ω-syntax'), actually using the same notation for operators and their corresponding predicates. In this way, the first expression of S may be written:

S —> β1(X,Y)
X —> γ(A1,C)
Y —> -β3(B1,Z)
Z —> γ(B2,A2)

Rules in this format may contain more than one predicate, thereby allowing hierarchical descriptions more general than binary trees, e.g.

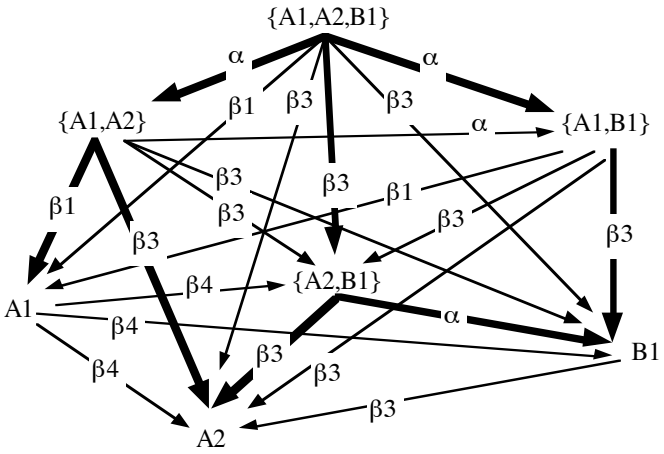


resulting in a more compact (and less ambiguous) representation:

S —> β4(A1,B1) -β1(A1,X) γ(A1,C) -β3(B1,X) -β4(B1,C) -β4(X,C)
X —> γ(B2,A2)

where the right side of the first rule is a conjunction of predicates[11].

In many cases only part of the information on time-span intervals is musically significant. If E = {A1,A2,B1,B2,C}, P(E), the set of all subsets of E, is a complete lattice which may be represented as a graph in which each node indicates a subset. Edges marking the inclusion relation X ⊃ Y may be labeled with the predicate which is verified on (X,Y), in fact always α, β1, or β3. The graph may then be complemented until it is complete. Other temporal relations: β2, β3, β4, γ, δ (and their symmetric counterparts) are visible in the complete graph. Consider for instance the subgraph concerning A1, A2 and B1 in the example:



Bold edges are those belonging to the lattice.

If only part of the information about time is known (i.e. significant), it may be completed with the aid of a *transitivity table*: given f(X,Y) and g(Y,Z), the table yields f*g(X,Z). In many cases the result is a disjunction of predicates; in some cases it is even unknown (hence it is a disjunction of the thirteen predicates). Part of the 13x13 square table is:

f \ g	−β2	β2	β4	γ	δ
−β2	−β2	α ∨ β1 ∨ β2 ... (unknown)	−β1 ∨ −β2 ∨ β4 ∨ γ ∨ δ	δ	δ
β2	α ∨ β1 ∨ −β1 ∨ β2 ∨ −β2 ∨ β3 ∨ −β3 ∨ β4 ∨ −β4	β2	β2 ∨ β3 ∨ β4	β2 ∨ β3 ∨ β4	β2 ∨ β3 ∨ β4 ∨ γ ∨ δ
β4	−β1 ∨ −β2 ∨ β4	β2 ∨ β3 ∨ β4 ∨ γ ∨ δ	β4 ∨ γ ∨ δ	δ	δ
γ	−β1 ∨ −β2 ∨ β4	δ	δ	δ	δ
δ	−β1 ∨ −β2 ∨ β4 ∨ γ ∨ δ	δ	δ	δ	δ

f * g | f(X,Y) ∧ g(Y,Z) —> f*g (X,Z)

[11] These examples of tree structures do not claim any musical relevance.

For example, $\beta2(X,Y) \wedge \gamma(Y,Z) \rightarrow [\beta2 \vee \beta3 \vee \beta4]\ (X,Z)$. The graph with incomplete information about time may be computed as a *constraint network* (Allen & Kautz 1985:255ff). The model makes no assumptions on relations that have deliberately been left unknown. Rather it is a enumeration of all structures consistent with the input data.

Unambiguous predicate representation requires $n(n-1)/2$ predicates in a rule containing n variables. The following matrix lay out is self-explanatory:

	A1	B1	X	C
A1	α	$\beta4$	$-\beta1$	γ
B1	$-\beta4$	α	$-\beta3$	$-\beta4$
X	$\beta1$	$\beta3$	α	$-\beta4$
C	$-\gamma$	$\beta4$	$\beta4$	α

$$S \rightarrow \begin{pmatrix} \alpha & \beta4 & -\beta1 & \gamma \\ -\beta4 & \alpha & -\beta3 & -\beta4 \\ \beta1 & \beta3 & \alpha & -\beta4 \\ -\gamma & \beta4 & \beta4 & \alpha \end{pmatrix} [A1,B1,X,C]$$

or equivalently:

$$S \rightarrow T\ [A1,B1,X,C] \qquad \text{with} \qquad T = \begin{pmatrix} \alpha & \beta4 & -\beta1 & \gamma \\ -\beta4 & \alpha & -\beta3 & -\beta4 \\ \beta1 & \beta3 & \alpha & -\beta4 \\ -\gamma & \beta4 & \beta4 & \alpha \end{pmatrix}$$

Vecchione's general format for context-sensitive *time-predicate rules* (which he calls 'Ω-syntax rules') is the following (1985:276):

$$\theta q\ [A\ J\ B] \rightarrow \theta r\ [C\ K\ D]$$

in which θq, θr are matrixes of time predicates, and A, B, C, D possible contexts. A, B, C, D, J, K are strings of variables corresponding to 'significant' units[12]. In general, $|K| \geq |J|$ to ensure decidability. If A=C and B=D the rule is context-sensitive in the common sense. The following subclass of rules

$$\theta q\ [C\ J\ D] \rightarrow \theta q\ [E\ J\ F]$$

may be called *change of context*. Depending on whether changes occur in the time-predicate matrix, in the medium part of the list ('J') or in its right and left contexts, eight modes of transformation can be defined and related to musical transformations[13] (Vecchione 1985:277ff).

---

[12] Derived from a meaningful segmentation.

[13] This should only be regarded as an initial step towards a theory of musical transformations.

In grammars, unary or *n*-ary predicates may also be used to express relationships other than those on time-span intervals: location in time and space, modes of significance, voice or instrument processing, etc. Grammars containing several classes of predicates (taking their arguments in several sets of values) have been called *Polymodal Multivalent Grammars* (PMG) (Vecchione 1985:257ff).

Time predicates are equivalent to the thirteen primitive relations in Allen & Kautz's model of naive temporal reasoning (1985), while Oppo (1984) considered only seven relations given the fact that, for any time predicate ω, $\omega(X,Y) = -\omega(Y,X)$. In a PMG it is necessary to use the thirteen values to obtain a canonic rule format that is meaningful for computing rule resemblances (Bel & Vecchione 1989).

Formulae using time operators and conjunctive expressions of time predicates may be matched or evaluated using tree unification as implemented in the Prolog language (Colmerauer 1984). The Prolog II-III implementation of generation/parsing techniques for PMGs is under study at the MIM laboratory (Risch 1988).

PMGs belong to a class of formal grammars which we call *pattern grammars*:

> A pattern grammar is an ordered 6-tuple (Vt, Vn, A, S, F, P) such that:
> Vt is the terminal alphabet, Vn a finite set of variables, $Vt \cap Vn = \varnothing$ and $S \in Vn$;
> A is any enumerable set;
> P is a set of predicates, i.e. applications of $(Vn \cup \mathrm{A})^n$ to {true, false} where n is a positive integer;
> F is a finite number of triplets (p,q,R) such that $p,q \in (Vn \cup Vt)^+$ and R a conjunctive expression using predicates in P whose arguments are variables occurring in q.

Triplets of F can be notated: p —> q *where* R . In many cases, A is the set of positive integers; consequently, predicates may accept numeric arguments, see for instance §2.1.

In the field of pattern recognition, Evans (1969) first proposed an effective procedure for inferring pattern grammars from examples. This author has achieved a similar goal using top-down parsing and postponed unification in a Prolog II program (Bel 1988).

The pattern grammar concept may be further elaborated depending on its actual implementation (rewriting systems, logic, etc.) Contextual information may be conveyed in a sophisticated way, e.g. *feature-grammars* or *metamorphosis grammars* (Colmerauer 1978). Metamorphosis grammars are easier to parse than context-sensitive grammars (Véronis 1986). An example of logic representation implicitly conveying contextual information may be found in Wiggins & al. (1988).

We shall see (§2.1) that *Bol Processor grammars* are a combination of pattern grammars and a sequential model derived from *pattern languages*. The following (§1.3—1.4) is a formal description of the sequential model which may be skipped by readers unfamiliar with formal language theory.

1.3. Pattern languages

If H denotes the set of all λ-free homomorphisms (with respect to concatenation) of $(Vn \cup Vt)^*$ to itself, an element of H which is the identity when restricted to Vt is called a *substitution*. If the image set of a substitution is Vt*, the substitution is *terminal*. A substitution which is a bijection of Vn to itself when restricted to Vn is called a *renaming of variables*.

A *pattern* is any element of $(Vn \cup Vt)^*$. If p is a pattern and s a substitution, then s(p) is a *derivation* of p. A pattern containing no variable is a *terminal derivation* or a *sentence*. Two patterns, p and q, are *equivalent* (i.e. $p \approx q$) if and only if there exists a renaming of variables r such that $p = r(q)$. Another binary relation (notated $p \leq q$) is defined as follows: p is *less*

general than (or *more specific than*) q iff for some substitution s, p = s(q). Since a substitution is a λ-free homomorphism, $p \leq q \Rightarrow |p| \geq |q|$.

The *language* generated by a pattern p is the set of terminal derivations of p, namely $L(p) = \{s \in Vt^+ : s \leq p\}$. It can be proved that:

$\leq$ is transitive
$p \leq q \Rightarrow L(q) \supseteq L(p)$
$p \approx q \Leftrightarrow p \leq q$ and $q \leq p$

However, the question of finding an effective procedure to decide whether $L(q) \supseteq L(p)$ given arbitrary patterns p and q appears to be open (Angluin 1980:49-52).

The mirror mapping (see 1.1) can be extended to the set of patterns with the following definition:

$\forall p,q \in (Vn \cup Vt)^*, p = mir(q) \Leftrightarrow \forall s \in Vt^+, s \leq q \Rightarrow mir(s) \leq p$

A transcription of a few variations of a *qa'ida* is given below[14]:

[1]	**dhin--dhagena**	**dha--dhagena**	**dhatigegenaka**	**dheenedheenagena**
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	tin--takena	**ta--takena**	**tatikekenaka**	**teeneteenakena**
	tagetirakita	dhin--dhagena	dhatigegenaka	dheenedheenagena
[2]	**dhin--dhatige**	**genakadhin--**	**tirakitatira**	**kitatirakita**
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	tin--tatike	**kenakatin--**	**tirakitatira**	**kitatirakita**
	tagetirakita	dhin--dhagena	dhatigegenaka	dheenedheenagena
[3]	dhin--dhagena	dha--dhagena	dhatigegenaka	dheenedheenagena
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	dhin--dhagena	**dha-dha-dha-**	dhatigegenaka	teeneteenakena
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	tin--takena	ta--takena	tatikekenaka	teeneteenakena
	taketirakita	tin--takena	tatikekenaka	teeneteenakena
	dhin--dhagena	**dha-dha-dha-**	dhatigegenaka	teeneteenakena
	tagetirakita	dhin--dhagena	dhatigegenaka	dheenedheenagena
[4]	dhin--dhagena	dha--dhagena	dhatigegenaka	dheenedheenagena
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	dheenedheenagena	**dheenedha-dheene**	dhatigegenaka	teeneteenakena
	tagetirakita	dhin--dhagena	dhatigegenaka	teeneteenakena
	tin--takena	ta--takena	tatikekenaka	teeneteenakena
	taketirakita	tin--takena	tatikekenaka	teeneteenakena
	dheenedheenagena	**dheenedha-dheene**	dhatigegenaka	teeneteenakena
	tagetirakita	dhin--dhagena	dhatigegenaka	dheenedheenagena

Tabulation has been used to denote beats; each beat comprises six strokes (*bol*s)[15].

Conventionally, the first item is the *theme* of the *qa'ida* whereas [2], [3] and [4] are called *variations*. The problem may be defined as the description of the whole set of possible variations of this theme. One obvious feature of this sample sequence is that some variations are

---

[14] *Qa'ida* is an Urdu word meaning 'rules' or 'system', and is derived from the Arabic word *qava'id* meaning 'grammar'. Tabla players often refer to the concept of grammaticality to denote correct arrangements of 'words' built on a limited vocabulary of *bol*s.

[15] In the onomatopoeic system used by North Indian drummers, Vt is practically a *prefix* code mapping strokes/sounds to the English lower-case alphabet. In other words, any string of Vt* (like *dheenedheenagena*) can be identified deterministically from left to right as a unique sequence of *bol*s (*dhee*/*ne*/*dhee*/*na*/*ge*/*na*). Therefore strings of Vt* are usually represented without spaces.

of the same duration as the theme (16 beats) while others are double (32 beats). Doubling the length is an improvisation technique commonly used by tabla players of the Lucknow style (Kippen 1988:166-67). These sets of variations may be roughly described with the following patterns, where X, Y, Z are variables:

Single variations:

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

Double variations:

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

Let L be the language representing all acceptable variations of this *qa'ida*, L1 the subset of single variations, and L2 the subset of double variations: $L = L1 \cup L2$. L(p1) and L(p2) are the languages generated by p1 and p2 respectively. We will agree (on the basis of much larger sample sets) that L(p1) and L(p2) contain non trivial subsets of L1 and L2. An elaborated description of L1 and L2 has been published in Bel (1987b).

Ideally, the search is for *descriptive* patterns of L1 and L2. A pattern d is descriptive of a language L if $L(d) \supseteq L$ and for every pattern q such that $L(q) \supseteq L$, L(q) is not a proper subset of L(d).

L is always a finite set: in this *qa'ida*, for example, all acceptable strings of Vt* have lengths shorter than or equal to 32 x 6 = 192 symbols, therefore $card(L) \leq (1+card(Vt))^{192}$. Yet, all we know about L is a set of positive and negative instances (correct and incorrect variations) called the *presentation*. The subsets of positive and negative instances are notated S^+ and S^-.

A pattern p *matches* a presentation $S = S^+ \cup S^-$ iff p is descriptive of S^+ and $S^- \cap L(p) = \varnothing$. A pattern p is a *tight fit* of S if $L(p) = S^+$. Evidently, any tight fit of S is descriptive of S^+ and matches S.

There is an effective procedure to infer a descriptive pattern from a set of positive instances S^+, but the problem appears to be NP-complete in most cases (Angluin 1980:54-55). In addition, the class of pattern languages is not closed under any of the basic set operations: union, complement, and intersection. Therefore it is not possible to construct systematically a pattern description of a language. In order to achieve such an analytical task it is necessary to define more suitable classes of languages.

1.3. Restricted pattern languages (RPL)

If p is a pattern, a subclass of L(p) may be defined with the aid of restrictions on acceptable derivations of p. For example, we may define the subclass of strings in L(p) with specified length n: $L_n(p) = \{s \in Vt^+ : s \leq p \text{ and } |s| = n\}$. In the example above, evidently $|X| = |Y| = 24$ and $|Z| = 12$. In addition, we can write that Y = mir(X).

There are other ways to formalize specifications with the aid of *rewriting rules*: we notate p —> q, the fact that *every acceptable derivation of q is an acceptable derivation of p*, i.e. $q \leq p$. If S is the set of all substitutions, p —> q denotes the subset $S_{p->q}$ such that $\forall s \in S_{p->q}$, s(p) = q. Since the problem of deciding whether $p \leq q$ given arbitrary patterns p and q is NP-complete (Angluin 1980:52), we must restrict to *well-formed rules* :

Definition:

Rule p —> q is well-formed iff $p \in Vn^+$, $q \in (Vt \cup Vn)^+$, $|q| \geq |p|$, and no variable occurs twice in p.

Theorem:

If rule p —> q is well-formed then $L(p) \supseteq L(q)$.

Proof:

All we need to find is a substitution s such that q = s(p) that yields $q \leq p$. A procedure for finding s is the following:
(1) substitute the |p|-1 leftmost variables in p with the |p|-1 leftmost symbols in q
(2) substitute the rightmost variable in p with the string made of the |q|-|p|+1 rightmost symbols in q.
Since no variable occurs twice in p, there are no restrictions on q that render substitutions impossible. This procedure may in fact be derived from Makanin's algorithm (see Makanin 1977 or Roussel 1987). For example, a substitution of XYZ yielding abcX is {X/a, Y/b, Z/cX}.

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

(a) select a subset R_o of the set of rules R .
(b) Let $S_o = \cap_i (S_{pi->qi})$ such that $(p_i \text{ —> } q_i) \in R_o$.
(c) If $S_o \neq \varnothing$ and S_o is finite, then $S_o = \{s\}$ such that the sentence is w = s(p).

The selection of R_o requires the comments:

(1) If a variable X does not appear in any premise of the selected set of rules, then it may be substituted to any $q \in (Vt \cup Vn)^+$. Consequently, S_o is infinite and $L_R(p) = \varnothing$.
(2) There are subsets of R that yield $S_o = \varnothing$, i.e. *aborted derivations*. For example, if a variable X appears as the premise of two distinct rules in R_o, X —> q_i and X —> q_j, then $S_{X->qi} \cap S_{X->qj} = \varnothing$, hence $S_o = \varnothing$.
(3) In cases other than (1) and (2) every variable occurs in a pattern with a unique derivation, so that $card(S_o) = 1$.

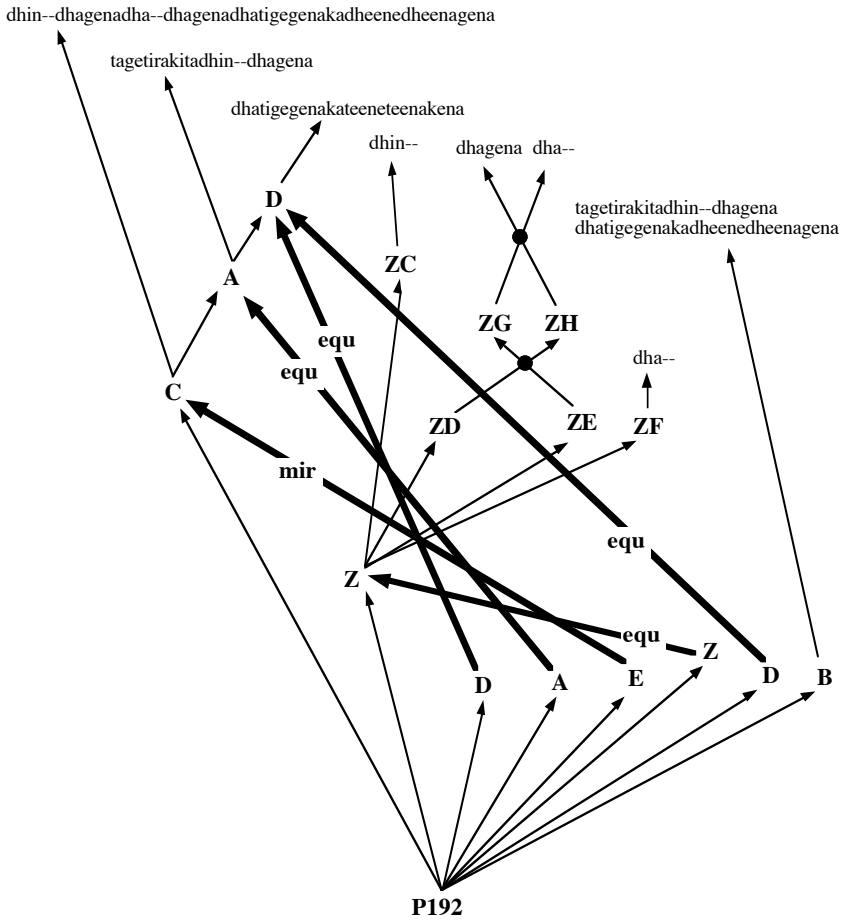
The following example will clarify points (2) and (3). A RPL for double-length variations would be $L_R(p2)$ where:

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

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

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

which may be represented with the following context-sensitive syntactic graph, where 'equ' and 'mir' indicate repetition and mirror structures:



Sets of substitutions for rules 10 and 14 may be represented as $S_{p10->q10}$ = {..., ZD ZE/ZG ZH, ...} and $S_{p14->q14}$ = {..., ZC/*dhin--* ,...}, where '...' indicates the (denumerable) possible substitutions of all strings of $(Vt \cup Vn)^+$ except the specified ones (ZD ZE and ZC). We find that $S_{p10->q10} \cap S_{p14->q14}$ = {..., ZD ZE/ZG ZH, ... , ZC/dhin-- ,...}.

The result of all intersections is:

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

where every derivation is explicit. Therefore S_o contains a unique substitution that yields the sentence.

If we select R_o = {1,2,3,4,5,8,10,13,14,15,21} we get $S_{p10->q10}$ = {..., ZD ZE/ZG ZH, ...}, $S_{p15->q15}$ = {..., ZD/*dhin*-- ,...}, and $S_{p20->q20}$ = {..., ZE/*dha*-- ,...}. Let us call $S_{15,20} = S_{p15->q15} \cap S_{p20->q20}$ = {..., ZD/*dhin*-- ,..., ZE/*dha*-- ,...}. For any s $\in S_{15,20}$, s(ZD ZE) = s(ZD) s(ZE) = *dhin--dha*--. Therefore ZD ZE may never be substituted to ZG ZH, which contradicts $S_{p10->q10}$. Consequently, $S_o = \varnothing$ and the derivation is aborted.

Remark: we may also define S_o as a maximal (hence *most specific*) consistent conjunctive expression using predicates ($p_i = q_i$) such that (p_i —> q_i) $\in R$.

Theorem:

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

Proof:

Each subset of the (finite) set of rules in a RPL yields at most one terminal derivation. Therefore the set of terminal derivations is finite. Conversely, for any finite language there exists a non-imbedding right-linear grammar G that generates exactly L. Let G = (Vt, Vn, S, R) where S is the starting symbol and R a finite set of rules in format A —> a or A —> aB such that A,B ∈ Vn, A≠B, and a ∈ Vt. It is easy to prove that G generates exactly the RPL $L_R(p)$ such that p = S, which completes the proof.

Corollary:

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

We can define an effective procedure for constructing $L_R(p) = L_{R1}(p1) \cup L_{R2}(p2)$ as follows:

Assume that $L_{R1}(p1)$ and $L_{R2}(p2)$ are defined with:
p1 = S1 and R1 = {S1 —> q1, ...}
p2 = S2 and R2 = {S2 —> q2, ...} in which we renamed variables in R2 so that no variable occurs in both R1 and R2 . We construct $L_R(p)$ defined with:
p = S and R = {S —> S1, S —> S2} ∪ R1 ∪ R2.
The set $L_R(p)$ of terminal derivations of S is the union of the sets of terminal derivations of S1 and S2, therefore $L_R(p) = L_{R1}(p1) \cup L_{R2}(p2)$. In addition, since R1 and R2 contain only well-formed rules, R contains also well-formed rules. Consequently, $L_R(p)$ is a RPL.

RPLs form a class of formal languages that is closed under union, and there is even a procedure for building a language from its subsets. In addition, the class is recursive and therefore membership tests can be computed. These properties are useful for building representational models that lend themselves to descriptive generalization.

2. Bol Processor grammars

2.1. Pattern rules

Since stroke sequences in North Indian drumming are homophonic, then in any pattern xyz the time predicates γ(x,y), γ(y,z) and δ(x,z) are implicit. Therefore the only predicates that need to be made explicit are those denoting repetitions (*equal*) and mirror substitutions (*mirror*).

For example the rewriting rules defining P96 and P192 in §1.4 may be represented with the following pattern grammar:

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

In this example the meaning of predicates *mirror*, *length* and *equal* is self-explanatory.

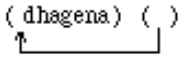
It is clear that if two variables linked with an *equal/mirror* predicate appear in the conclusion of a rule, the leftmost variable denotes a sequence of strokes that has been played first and served as a reference. For example, using the following set of rules:

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

the terminal derivation *dhagenadhagena* is such that the first *dhagena* has served as a reference for the second. Thus we may notate this sentence:

(= dhagena) (: dhagena)

with the meaning that the parenthesis with an equal sign denotes the reference, and that with a colon is the copy. In fact, in the machine implementation, the content of the second parenthesis is merely a pointer to the reference:



We use the same convention in rules, e.g.:

A —> (= B) (: B)

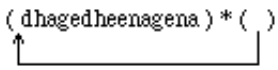
As to notating mirrors, we use an asterisk to indicate that the following sequence between brackets is unvoiced. For example, given the mirror mapping of §1.1, in the following grammar:

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

the (only) terminal derivation is

(= dhagedheenagena) * (: taketeenakena)

and its internal representation:



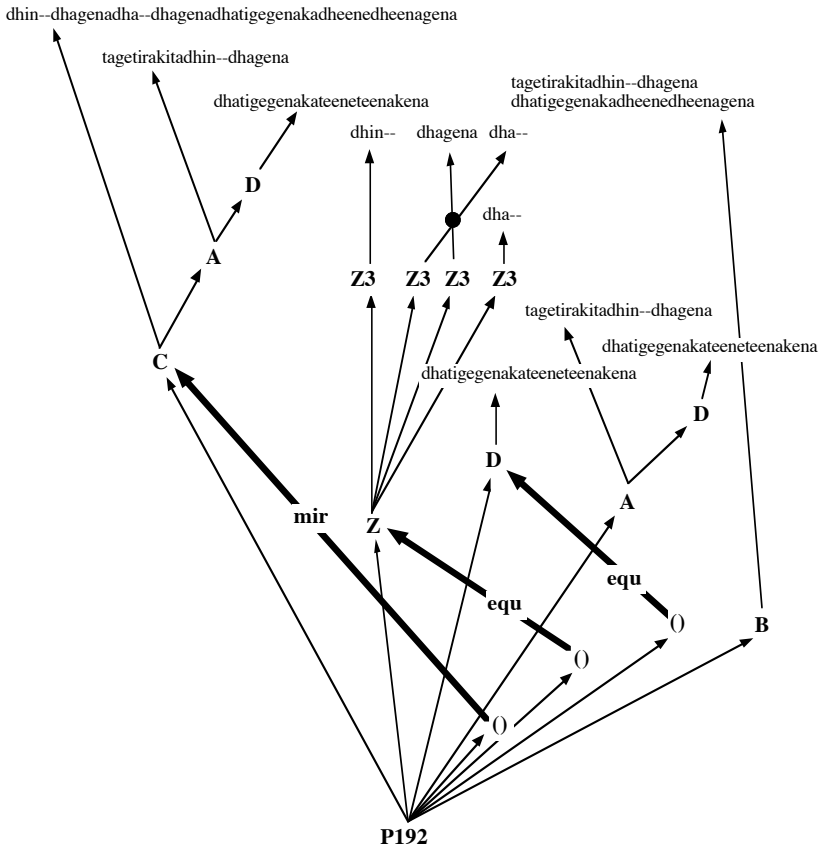
Using these conventions we can simplify the format of rewriting rules in RPLs:

(1) If a variable appears several times in the RPL rewriting rule, it shall be represented with repetition markers. For instance,
A —> B C B C B shall be represented A —> (=B) (=C) (:B) (:C) (:B).
(2) The same variable may appear twice in a rule premise (if not linked with repetition markers), e.g. A A —> q is correct but not (=A) (:A) —> q.
(3) Mirrors shall be represented with an asterisk.

Using these conventions, the BP grammar generating the RPL language defined under §1.3 is:

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

in which rules 9, 10, 11, 15, 16, 17, 20, 21 and 23 have been deleted. This representation is more compact although it generates the same language. The syntactic graph of the sentence generated in §1.4 is now:



In BP representations, a grammar is generally layered in several *transformational* 'subgrammars'. The term 'transformational' is used here as it is defined in formal language theory (Bel 1987a:356), not linguistics. In a subgrammar there is more than one starting symbol, and a derivation sequence consists of transforming a string of starting symbols P1 to a string of terminal symbols P2. Symbols that are terminal to a subgrammar may become the starting symbols of the next subgrammar(s) to be applied, and so on (Kippen & Bel 1989a:3).

2.2. The membership test in Bol Processor grammars

BP grammars may be viewed as *declarative* statements on the languages they generate. Nevertheless, a membership test is a recognition procedure by which a given string is checked against the grammar. Since the membership test algorithm implemented in the BP is bottom-up deterministic, emphasis is put on a *procedural* description. Discriminating compositional and analytical processes has been one of the major reasons why the BP could be used for modelling part of the musical 'behaviour' of traditional musicians[16].

A membership test should provide an explanation of its failures by pointing to a deficiency of the grammar, i.e. an easy step by step trace. For these reasons the test implemented in the BP is bottom-up deterministic. Informally, the algorithm is the following:

[16] This corroborates Laske's view (1984:168) on analysis versus composition.

(1) If G is a transformational grammar, its *dual* is obtained by swapping the premise and conclusion in every rule.
(2) If $G_1, G_2, ... G_n$ are the subgrammars of the language, a membership test is the result of the *context-sensitive canonic rightmost derivation* of the sentence by $G_n, ... G_2, G_1$.
(3) If the membership test has yielded S, the starting symbol, the input sentence is assessed 'correct'. It is 'incorrect' in any other case.

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 rule premises contain no more than one variable. This author has extended this definition to all *length-increasing* grammars:

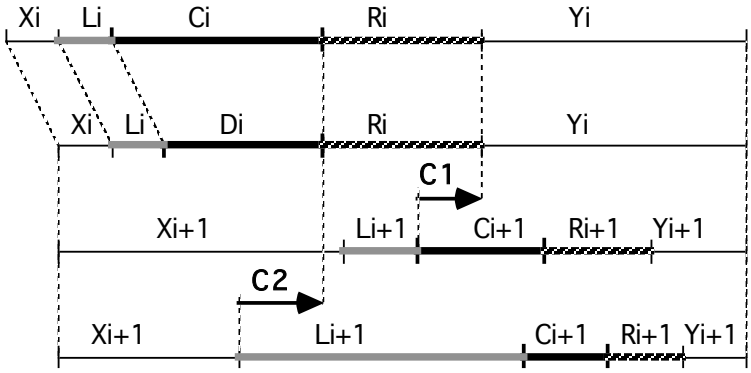
Context-sensitive canonic rightmost derivation: definition

The formal definition of the context-sensitive rightmost definition adapted from Hart (1980) and used in the membership test of LIN grammars is the following:

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

In Hart's definition (1980:82), $|C_i| = |C_{i+1}| = 1$, so that C1 may be written $|R_{i+1}\ Y_{i+1}| \geq |Y_i|$.

The diagram and commentary below illustrate conditions **C1** and **C2**:



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.

Now, an explanation of how the inference engine of the BP handles ambiguity in bottom-up parsing will be given. Suppose that for a working string W_i there are two candidate rules:

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

The selection criterion is the following: 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 (Bel 1987a:8) that once D1 has been considered, D2 is no longer relevant and ambiguity may therefore be handled by D3 and D4. D3 makes a decision on the basis of the length of the conclusions 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.

To augment efficiency, D3 is not considered by the inference engine of the BP; it is therefore the task of the analyst to take into account the following partial ordering of rules:

'Chunk' rule:

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

There are a few more restrictions on rule formats imposed by the membership test procedure (Bel 1987a-b). One is that *right contexts may only contain symbols of an alphabet external to the grammar*.

2.3. Templates

Sentences can only be parsed if they have been entered into the editor complete with structural information, i.e. brackets, asterisks, etc. Yet in view of the fact that the Bol Processor has to perform membership tests on large amounts of data comprising examples to which it is not always easy to assign a structure, this limitation is held to be unacceptable. In response to this, the inference engine of the Bol Processor is able to generate templates from a grammar, i.e. a list of possible structures in which each dot represents a terminal symbol of one unit. Templates are enumerated and stored in the grammar file, for instance:

[1] (=.....)(=.....)(=.....)*(.....)(:.....)(=.....)

[2] (=.....)(=.....)*(.....)(:.....)

[3] (=.....)(=.....)*(.....)(=.....)

In analysis the Bol Processor takes a new sentence and superimposes it on each template in strict order. A membership test is performed each time the sentence matches a template, so allowing for any structural ambiguity to be assessed. An example of template matching may be found in Kippen & Bel (1989c: appendix).

2.4. Stochastic model

The only realistic method for testing a grammar with an expert musician is to instruct the machine to produce one randomly chosen sentence at a time. If the sentence is assessed correct, the procedure is invoked again and another sentence is generated. Generally, the grammar is considered to be satisfactory if all sentences generated within a few sessions have been accepted by the expert.

Since the correctness of a grammar can never be fully assessed — indeed, like musicians themselves, machines may be allowed casual mistakes — it is important to enable the stochastic

production process to generate sentences from a wide and representative subset of the language. This can be achieved by weighting the decisions of the inference engine.

Formal grammars offer the possibility of defining consistent probabilistic models that apply to a very wide class of languages. We developed such a model as a response to the need to inhibit rules in context-free grammars. This model is derived from probabilistic grammars/automata as defined by Booth & Thompson (1973), the difference being that a *weight* — within the range [0,255] — rather than a *probability* is attached to every context-free rule. The rule probability is computed from weights as follows: if the weight is zero then the probability is zero; if the weight is positive then the inference engine calculates the sum of weights of all *candidate* rules, and the rule probability is the ratio of its own weight to the sum. Candidate rules are those whose premise is a substring of the work string. 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 the weights is 100+100+50+5 = 255. The probability of choosing rule [4] in the derivation of a string containing *V3* is therefore 5/255 = 0.0196. Using weights instead of probabilities has the advantage that is does not presuppose the sum of coefficients of all candidate rules to be 1.

Weights (and their associated probabilities) are used in modus ponens to direct the Bol Processor's production along paths more likely to be followed by musicians. In some context-free grammars — those that fulfil the *consistency* condition expressed by Booth & Thompson (1973:442) — they may be used for computing a *probabilistic sentence function*, i.e. a coefficient representing the likelihood of occurrence of each sentence in the language. Grammars that are constructed in a systematic way (or generated from sample sentences, see Kippen & Bel 1989b:205) are good examples of consistent probabilistic grammars.

Another remarkable feature of consistent grammars is that rule probabilities can be inferred from a set of sentences (Maryanski & Booth 1977:525). Given a correct grammar and a subset of the language that this grammar generates (for instance a sample sequence taken from a performance of an expert musician), rule weights are inferred as follows: let all weights be reset to zero; then analyse every sentence and increment by one unit the weights of all rules used in the derivation. (The algorithm described here is more general than the one devised by Maryanski and Booth, since the latter requires the choice of a sample set in which *all* rules have been used.)

Evidently, rules that are never used in the analysis of a sample set remain with weight (and therefore probability) zero, which inhibits their use in modus ponens. Those rules may be scrutinized to see whether they are incorrect or whether they point to fragments of the language that have not yet been explored. To test this, their weights are set to a high value so that the Bol Processor is forced to generate sentences that either have never been assessed or at least are not being considered by the informant at the time.

An example of weight inference based on the *qa'ida* introduced in §1.2 may be found in Kippen & Bel (1989a).

Using 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, a problem that arose from the complete randomness of the generative process.

2.5. Recent developments of the Bol Processor

These developments have taken place as the result of collaboration with Western composers, more specifically musicians working with digital synthesizers. Many have felt a strong need to implement general abstract models of musical structures. A new version of the Bol Processor has been implemented in C language on the Macintosh. 'BP2' accepts polyphonic structures in a format loosely related to Diener's *T-trees* (1988). Unlike T-trees, however, where each individual terminal (e.g. a note or a stroke) is viewed as an object (in an *object-oriented* programming environment), BP2 remains based on strings. Structural units of overlapping events may be represented as sets or trees. Several λ-free homomorphisms may be defined on the same alphabet. Wildcards (local variables that may be instantiated on Vn ∪ Vt) can be used in the left argument of context-sensitive rules to represent patterns. BP2 is also able to handle *metagrammars*, i.e. grammars that generate grammars.

A MIDI interface is under study so that the sentences generated may be immediately checked in the way they actually sound.

3. Conclusion

Pattern grammars (which may be imbedded in production rules, logic descriptions, tree algebra, etc.) are not only powerful tools for music analysis. They also lend themselves to descriptions of creative/evaluation processes. In the phenomenological approach, a pattern is viewed more as a *dynamic* than static relation:

> (La forme) n'est pas la somme des détails intégrés dans l'ensemble qui constitue l'oeuvre. Elle appartient au niveau des structures, c'est à dire des principes. Elle s'identifie avec le schème d'organisation qui suggère l'assemblage des parties en considération de leurs liaisons avec les lois propres du schème organisateur. La forme est liée aux principes de cohésion du système, à la réalisation d'un dynamisme particulier.
> (Vecchione 1984:11-12)

Descriptions may be *normative* (i.e. based on some kind of rigorous reasoning) when dealing with highly constrained systems (e.g. improvisation in traditional music) or *empirical* (i.e. based on multi-criteria classification) when applied to the modelling of a compositional process.

The problem lies less in finding a 'universal' abstract representation of music, than in delineating forms of musical 'thinking' that may be operative in the design of tools for computer-aided music creation and performance.

Acknowledgements

I am indebted to Prof. Bernard Vecchione and Dr. Jim Kippen for their help in providing musical examples and the *raison d'être* of this paper. Their competence in music theory, practice, and in the history of musical analysis has been essential for the difficult task of linking formal models with musical concepts.

References

Allen, J. F. & H. Kautz

'A Model of Naive Temporal Reasoning', in *Formal Theories of the Commonsense World*, J.R. Hobbs & R.C. Moore, Eds., Ablex, 1985:251-68

Allouche, J.P.

'Automates finis en théorie des nombres', *Expositiones Mathematicae* 5, 1987:239-66

Angluin, D.

'Finding Patterns Common to a Set of Strings', *Journal of Computer and System Sciences* 21, 1980:46-42

Bel, B.

'Les grammaires et le moteur d'inférences du Bol Processor', note n°237, GRTC, Centre National de la Recherche Scientifique, Marseille, 1987a

'Grammaires de génération et de reconnaissance de phrases rythmiques', *6ème Congrès A.F.C.E.T: Reconnaissance des Formes et Intelligence Artificielle*, Antibes, 1987b:353-66

'Inférence de grammaires dans la forme normale de Chomsky pour la description structurelle d'ensembles d'objets', 1988, unpublished

Bel, B. & B. Vecchione

'The "OC" project', *European Workshop on Artificial Intelligence and Music*, Genoa (Italy), June 22-23, 1989

Birkhoff, G.

Lattice theory, American Mathematical Society Colloqium Publications, Vol.25, Providence (USA), 1940, reprinted 1984

Booth, T.L., & R.A. Thompson

'Applying Probability Measures to Abstract Languages', *IEEE Transactions on Computers*, Vol. C-22, n°5, 1973:442-50

Camilleri, L.

'A grammar of the melodies of Schubert's Lieder', *Musical Grammars and Computer Analysis*, M. Baroni & L. Callegari, Eds., Olschki, Firenze (Italy), 1984:229-36

Camilleri, L., F. Carreras & C. Duranti

'An expert system prototype for the study of musical segmentation', *European Workshop on AI & Music*, Genoa (Italy), June 1989:14-15

Chemillier, M.

'Monoïde libre et musique: deuxième partie', *RAIRO — Informatique Théorique et Applications*, Vol.21, n°4, 1987:379-418

Chemillier, M., & D. Timis

'Toward a theory of formal musical languages', *14th International Computer Music Conference*, Cologne (FRG), 1988:175-83

Colmerauer, A.

'Metamorphosis Grammars', *Lecture Notes in Computer Science*, Vol. 63, Jan. 1978:133-89

'Equations and inequations on finite and infinite trees', *Proc. of the Second International Conference on Fifth Generation Computer Systems*, Tokyo (Japan), 1984:85-99

Coons, E. & D. Kraenhenbuehl

'Information as a measure of structure in music', *Journal of Music Theory,* 2, 1958:127-61

Dershowitz, N. & J.P. Jouannaud
'Rewrite systems', rapport de recherche n°478, LRI, Université Paris-Sud, Orsay, 1989

Diener, Glendon
'T-trees: An Active Data Structure for Computer Music', *14th International Computer Music Conference*, Cologne (FRG), 1988:184-88

Evans, T.G.
'Grammatical Inference in Pattern Analysis', *3d International Symposium on Computer and Information Sciences*, Miami Beach, (USA), 1969:183-202

Feld, S.
'Linguistic Models in Ethnomusicology', *Ethnomusicology* 18(2), May 1974:197-217

Fuchs, W.
'Musical analysis by mathematics', Gravesano Blätter 1, Gravesano (Switzerland), 1961

Hart, J.M.
'Derivation Structures for Strictly Context-Sensitive Grammars', *Information & Control* 45, 1980:68-89

Jackendoff, R., & F. Lerdahl
'A Grammatical Parallel between Music and Language', *Music, Mind and Brain: the Neuropsychology of Music*, M. Clynes, Ed., Plenum Press, New York, 1982:83-117

Kippen, J.
'An ethnomusicological approach to the analysis of musical cognition.', *Music Perception* 5(2), 1987:173-95

The Tabla of Lucknow: a Cultural Analysis of a Musical Tradition, Cambridge University Press (UK), 1988

Kippen, J., & B. Bel
'Linguistic study of rhythm: computer models of tabla language', *ISTAR Newsletter* 2, International Society for Traditional Arts Research, New Delhi, 1984:28-33

'Modelling music with grammars: formal language representation in the Bol Processor', in *Computer Representations and Models in Music*. A. Marsden and A. Pople., Eds., Academic Press, London, 1989a, forthcoming

'The identification and modelling of a percussion "language", and the emergence of musical concepts in a machine-learning experimental set-up', Computers & Humanities 23.3, 1989b:199-214

'Can a computer help resolve the problem of ethnographic description?', *Anthropological Quarterly*, 1989c, forthcoming

'From word-processing to automatic knowledge acquistion: a pragmatic application for computers in experimental ethnomusicology', *ALLC/ICCH Conference*, Toronto, 6-10 June 1989d

Laske, O.
'KEITH: a rule system for making music-analytical discoveries', *Musical Grammars and Computer Analysis*, M. Baroni & L. Callegari, Eds., Olschki, Firenze (Italy), 1984:165-99

Lerdahl, F. & R. Jackendoff
A Generative Theory of Tonal Music, MIT Press, Cambridge (USA), 1983

Lévy-Strauss, C.
Le Cru et le Cuit, Plon, Paris, 1964

Makanin, G.S.
'The problem of solvability of equations in a free semi-group', *Math. USSR Sbornik* 32, 2 , 1977, also in *American Mathematical Society*, 1978

Maryanski, F.J., & T.L. Booth
'Inference of Finite-State Probabilistic Grammars', *IEEE Transactions on Computers*, Vol. C-26, n°6, 1977:521-36
[Some anomalies of this paper are corrected in B.R. Gaine's paper 'Maryanski's Grammatical Inferencer', *IEEE Transactions on Computers*, Vol. C-27, n°1, 1979:62-64]

Mazurkiewicz, A.
'Traces, histories, graphs: instances of a process monoïd', *Lecture Notes in Computer Science* 176, 1984:115-33

Meyer, L.B.
Emotion and meaning in music, University of Chicago Press, Chicago, 1956
'Meaning in music and information theory', *Journal of Aesth. & Art Criticism,* 15, 1957

Moles, A.
Théorie de l'information et perception esthétique, Denoël, Paris, 1958

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

Oppo, F.
'Per una teoria generale del languaggio musicale', *Musical Grammars and Computer Analysis*, M. Baroni & L. Callegari, Eds., Olschki, Firenze (Italy), 1984:115-30

Philippot, M.
'Muss es sein', in *L'Arc 40: Beethoven,* A. Boucourechliev, Ed., Aix-en-Provence, 1970:82-95

Risch, V.
'Prolog II et les grammaires temporelles', *Colloque 'Structures Musicales et Assistance Informatique'*, Laboratoire Musique et Informatique (MIM), Marseille, 1988

Ruwet, N.
'Méthodes d'analyse en musicologie.', *Revue de musicologie* 20, 1966:65-90, reprinted in *Langage, Musique, Poésie*, Seuil, Paris, 1972

Roads, C.
'An overview of music representations', *Musical Grammars and Computer Analysis*, M. Baroni & L. Callegari, Eds., Olschki, Firenze (Italy), 1984:7-37

Roussel, A.
'Programmation de l'algorithme de Makanin en Prolog II', Mémoire de D.E.A. de Mathématiques et Informatique, GIA, Faculté des Sciences de Luminy, Université Marseille II, 1987

Van Benthem, J.F.A.K.
The Logic of Time, Reidel, Dordrecht-Boston-London, 1983

Vecchione, B.
'Pour une théorie générale des grammaires musicales formelles', Université de Provence Aix-Marseille I, 1978
Pour une science de la réalité musicale, Thèse de Doctorat de IIIème cycle, Université de Provence Aix-Marseille I, 1984
La réalité musicale: éléments d'épistémologie musicologique, Thèse de Doctorat d'Etat, Université Paris VIII, 1985

Véronis, J.
'Bottom-up parser generation in Prolog', note n°123, GRTC, Centre National de la Recherche Scientifique, Marseille, 1986

Wiggins, G., M. Harris & A. Smail
'An Abstract Representation for Music', Dept of AI, University of Edinburgh (UK), 1988, unpublished

Xenakis, I.
Musiques formelles, La Revue Musicale, Paris, 1963
'Vers une métamusique', in *Architecture et Musique*, Casterman, Paris, 1972:38-70
