https://bolprocessor.org/misc/texts/AFCET87.pdf

Proceedings of the 6th AFCET/INRIA Conference, Antibes, 1987b:

353-366. GENERATION AND RECOGNITION GRAMMARS FOR RHYTHMIC PHRASES

Bernard Bel
Knowledge Representation and Processing Group, National
for Scientific Research 31,
ch Joseph Aiguier
13402 MARSEILLE cedex 9

Abstract:

A cognitive anthropology project on the percussion languages of northern India brings together expert musicians, an analyst, and an automaton simulating the generation and recognition of rhythmic compositions. Knowledge and hypotheses about each type of composition are formulated using production rules. The grammars are then modified until the correspondence between the set of generated phrases and musical practice is deemed satisfactory. This presentation describes the formal representation and the algorithms used for the synthesis and analysis of phrases, drawing on examples from the musical repertoire.

Keywords:

cognitive anthropology - formal languages - phrase recognition - syntax - learning - canonical derivation - rhythm

Abstract:

In a cognitive anthropological project focusing on drumming languages in North India, an interaction has been established between expert musicians, an analyst, and an automaton that simulates the generation and recognition of rhythmic compositions. Production rules are used to represent knowledge and hypotheses regarding each compositional type. Grammars are then modified to align the set of generated sentences with musical practice. This paper describes the data representation and algorithms used for the synthesis and parsing of sentences, supported by examples drawn from the musical repertoire.

Keywords:

cognitive anthropology - formal languages - sentence recognition - syntax - machine learning - canonical derivation - rhythm

1. Introduction

The field to which this article refers is that of the qa'ida, a term meaning 'rules' or 'system' and denoting a type of composition used

by musicians in northern India. These pieces are performed on the tabla, a two-part percussion instrument struck with the hands. The use of generative grammars for the representation of rhythmic languages stems from the importance of the concepts of permutation, substitution, and context—notions that are easily translated into production rules.
Experience also shows that this formalism is accessible to analysts who are not computer scientists and even to certain musicians [Kippen 86].

Based on a given grammar, an automaton (the Bol Processor) synthesizes phrases that are then evaluated by the musicians. By adjusting weights, certain rules can be made more likely than others, thereby exploring the boundaries of the language. After a period of fine-tuning, the grammar may produce only correct variations of the qa'ida. It must then be ensured that the language it generates contains all possible variations. To this end, the musician is asked to compose variations that are subjected to a membership test. Any incorrect evaluation leads to a further modification of the grammar.

We have presented elsewhere [Bel 87b] a learning-based "laboratory" approach: segmentation, formulation of specific rules, followed by generalization of grammars through inductive reasoning. In the experimental context, however, the analyst must start with a grammar as general as possible, representing their (incomplete) knowledge of the language. This knowledge is generally described procedurally, as a sentence of the language can be viewed as a derivation from the start symbol involving several transformational grammars. This idea is reinforced by the use of a deterministic membership-testing algorithm, which employs a canonical derivation applicable to a subclass of contextual languages. However, it is necessary to introduce non-determinism when a sentence is subjected to analysis without information about its structure: the Bol Processor's inference engine uses a collection of templates for this purpose, allowing it to insert structural symbols into the sentence.

Using an example from the musical repertoire, we show how a grammar can be modified to account for accentuation and context constraints. We conclude with a rule validation technique that involves a deductive reasoning model and control mechanisms analogous to those of machine learning.

This study is being conducted in collaboration with J.R. Kippen, an ethnomusicologist in the Department of Social Anthropology at the University of Belfast.

2. Notation of rhythmic compositions

Once a certain level of expertise is reached, tabla instruction is primarily conducted through the use of bowls—onomatopoeic terms that represent both the sounds of the instrument and the actions required to produce them. This means it is possible to analyze the sound-production system and the content of the language—reduced to its written representation—separately.

There exists an idempotent application of the alphabet of bowls, which we call the mirror, and which maps an open sound to its closed version obtained by muffling the instrument's resonance. In the examples we will discuss later, the alphabet of the language and the mirror application are defined as follows:

The symbol "-" represents a silence. Beats are marked by spaces or tabs. The bowls located within a beat all have the same rhythmic value.

Compositions often use repetitions of motifs, as well as a repetition created using the mirror application. To notate these structures, we use a system of parentheses and pointers, as well as a genre marker (open/closed). For example:

- (= dha te te dha) (= - -) (: dha te te dha) (: - -) (: dha te te

dha) (= dha ge dhi na ghi na) * (: ta ke ti na ki na)

A parenthesis containing "=" is called a master parenthesis. Those containing ":" are called slave parentheses and contain a pointer to a master parenthesis located to their left. The content of the parenthesis following the "*" symbol is a sequence of closed sounds. Multiple levels of parentheses can be nested. After each rewriting, the inference engine updates the pointers of the slave parentheses, taking into account structural changes.

The notation for tempo variations, which are not used in the example cited in the appendix, is explained elsewhere [Bel 87b].

3. Grammars and Language Classes

The analyst can program type-0 grammars (recursively enumerable languages) on the Bol Processor. However, for the sentence membership test to be algorithmic, these languages must also be recursive, i.e., recognizable by a Turing machine [Kain 81 p.111]. This class includes that of context-sensitive languages (type 1).

Type 1 rules are of the length-increasing form: LCR--> LDR where $|D| \geq |C|$

The grammars formed in this way are weakly equivalent to strictly contextual grammars [Salomaa 73], in which $|C| = 1$. The strictly type-0 rules used in rhythmic language grammars are always of the form:

A ... A (N times) --> AN

where AN represents a variable that will be derived as follows:

AN => ... => a1...aN

where the symbols ai are terminals of the language. These derivations are therefore equivalent to those of monotonic-increasing grammars.

Languages are generally described by successive derivations in multiple phrase-structure transformational grammars, a description that reflects the multi-step nature of the synthesis or recognition process. We define these grammars as follows:

A phrase-structure transformational grammar G is an ordered quintuple (VE, VN, VT, VD, F) such that:

$V = VE \cup VN \cup VT \cup VD$, $VT \neq \emptyset$, $VD \neq \emptyset$
{VE, VN, VT, VD} is a partition of V,
and F is a finite set of ordered pairs (LCR, LDR) such that: $C \in (VN \cup VT \cup VD)+$, $L \in V^*$, $R \in V^*$, and $D \in (VN \cup VT)+$

VD is the start alphabet, VT the end alphabet, VN the intermediate alphabet (the variables), VE the alphabet outside G, and F the set of production rules.

The initial alphabet of a grammar is necessarily included in the union of the terminal alphabets of the preceding grammars, except for the first one where VD = {S}. If the language is described by n grammars, the terminal alphabet of Gn is included in the terminal alphabet of the language. We also agree that the external alphabet of G1 is empty and that every symbol in the external alphabet of a grammar Gi is a terminal of a grammar Gj with j < i.

Terminal symbols that do not represent bowls (see 2) are called structural symbols:

brackets and repetition pointers: (=) (:)
the mirror symbol: *
integers (1..99) representing the current speed of
special symbols chosen from: {+ : ; = }

4. Sentence generation

Sentence generation uses the following algorithm:

```
Procedure GENERATE
start
n := 1
while there exists a grammar Gn, repeat: start
      while a rule is applicable, repeat: start
            select an applicable rule select a
            derivation position
            if the left-hand side of the rule is recognized
                  then replace the found occurrence with the term


right
            end
      n := n+1
      end
```

end

4.1 Rule Selection

In ORD grammars, the rule chosen is, by default, the rule previously applied; otherwise, it is the first candidate rule in ascending order. In LIN and RND grammars, a rule is chosen by a draw weighted by the weights of the candidate rules.

4.2 The derivation position

If the rule contains the LEFT instruction, or if it belongs to a LIN grammar, the derivation position is the leftmost position that allows the left-hand side of the rule to match the sentence currently being derived. In all other cases, the position is random.

5. Sentence recognition (LIN grammars)

The deterministic membership test algorithm is as follows:

(1) Given a transformational grammar G, swap the left-hand and right-hand sides of all the rules in G. Let G' be the resulting grammar (dual grammar [Salomaa 73]).

(2) Knowing that the language is generated by applying the transformational grammars G1, ... Gn in that order, we perform the canonical right derivation of the string parsed by G'n, ... G'1.

(3) The sentence belongs to the language if the derivation ends with the start symbol S.

If a word u is derived from a word v by canonical right derivation in a grammar without deletion rules, then v is derived from u by canonical left derivation in the dual grammar [Hart 76 p.245].

This algorithm offers several advantages: it is memory-efficient, and the number of rewrites is less than the length of the analyzed string. The only challenge is defining the canonical derivation, as well as the algorithm’s scope of application—that is, the class of languages for which it recognizes every sentence.

5.1 Right-contextual canonical derivation: definition Let G be a contextual grammar. The derivation of G:

W0 => W1 => ... => Wn

is context-sensitive rightmost canonical if:

$\forall$ i $\in$ [0, n-1], Wi = XiLiCiRiYi and Wi+1 = XiLiDiRiYi by applying the rule fi:

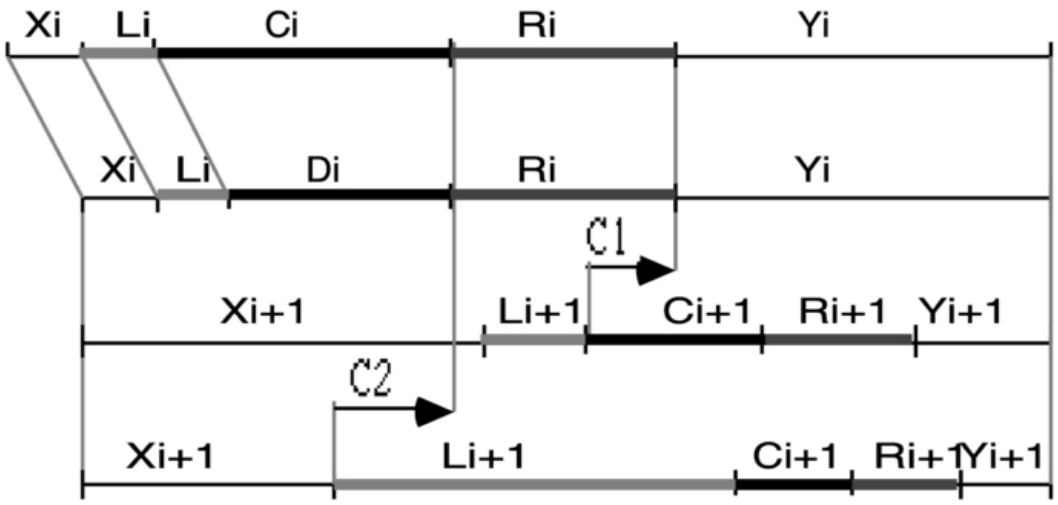
LiCiRi --> LiDiRi,

and at least one of the following conditions is satisfied:

|Ci+1Ri+1Yi+1| > |Yi| (C1)

|Li+1Ci+1Ri+1Yi+1| > |RiYi| (C2)

The figure below illustrates conditions (C1) and (C2), which we will now justify.



Suppose that conditions (C1) and (C2) are not satisfied. The rule fi+1 could have been applied before fi, since Li+1Ci+1Ri+1 would be a substring of RiYi. On the other hand, applying the rule fi+1 would only modify the string Yi without affecting the context Ri. In this case, the order of fi and fi+1 could be reversed. This reversal would be justified since the symbols rewritten by fi+1 are to the right of those rewritten by fi.

For a strictly contextual grammar, |Ci| = |Ci+1| = 1, and condition (C1) can be written as |Ri+1Yi+1| ≥ |Yi|, which amounts to Hart's definition [Hart 80 p.82]. On the other hand, if (C1) or (C2) holds, then the condition |Li+1Ci+1Ri+1Yi+1| > |Yi|, which defines the canonical right derivation of a type-0 grammar without deletion rules, also holds [Hart 76 p.246].

5.2 Rule Selection Criteria

In the Bol Processor, we have defined criteria that allow us to order the rules in order to perform the canonical right derivation during analysis. If two rules are candidates:

```
      LiCiRi      -->  LiDiRi                  (fi)
and   L'iC'iR'i   -->  L'iD'iR'i               (f'i)
where XiLiCiRiYi = X'iL'iC'iR'iY'i = Wi
```

(fi) will be chosen over (f 'i) if any of the following criteria, in order, is met:

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

We have shown [Bel 87a] that criteria (D2) and (D3) can be ignored under two conditions. The first is the chunk rule:

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

The second is a sufficient (but not necessary) condition for (D1) and (D4) to satisfy the context dependencies in the contextual syntax graph [Révész 85, p. 181; Bel 87a]:

A right-hand context can contain only symbols from the alphabet external to the grammar.

This rule justifies the use of transformational grammars, where the concept of an external alphabet is recognized (see 3).

6. Scope of application of the deterministic algorithm

We will show how to modify a grammar with overlapping patterns that does not generate a language recognized by the algorithm described in (5):

LIN			
S	<->	A A A A A A A A A A	(f1)
A	<->	A1	(f2)
A A	<->	A2	(f3)
A A A	<->	A3	(f4)
A1	<->	-	(f5)
A2	<->	dha -	(f6)
A2	<->	- dha	(f7)
A3	<->	te te dha	(f8)
A3	<->	dha te te	(f9)

The chunk rule dictates the order (f5,f6). The following correct sentence:

te te dha te te dha - dha te te

will be derived, according to criteria (D1) and (D4), as follows:

te te dha te te dha - dha te te => te te dha te te dha - A3 => te te dha te te dha - A A A =>

te te dha te te A2 A A A => te te dha te te A A A A A => te te A3 A A A A A =>

te te A A A A A A A A ... failure

The solution, for the deterministic algorithm to be applicable, consists of examining all possible configurations of concatenating two patterns with their correct segmentation:

- | -, - | dha -, - | - dha, - | te te dha, - | dha te te

dha - | -, dha - | dha -, dha - | - dha, dha - | te te dha, dha - | dha te te

- dha | -, - dha | dha -, - dha | - dha, - dha | te te dha, - dha | dha te te

te te dha | -, te te dha | dha -, te te dha | - dha, te te dha | te te dha, te te dha | dha te te

dha te te | -, dha te te | dha -, dha te te | - dha, dha te te | te te dha, dha te te | dha te te

The underlined concatenation is the one that will be incorrectly segmented and will cause the analysis to fail. Therefore, the following rules must be added:

A A A A <-> A4 (f10)
A4 <-> te te dha - (f11)

and repeat the process if necessary with the new word. This approach is very similar to that of a musicologist who breaks down phrases into simple words and then defines compound words representing broader motifs, and can also be implemented using software [Bel 87b].

The scope of application of the deterministic membership testing algorithm can now be formally defined:

The grammars of a language L that allow the use of the deterministic algorithm possess the following properties:

(1) Every rule is of the form: Li Ci Ri --> Li Di Ri (fi)
with $|Ci| \geq 1$ and $|Di| \geq 1$
(2) Let M = {Di, a right-hand side of (fi) where Di is in VT+}.
(M is the set of words)
$\exists u,v \in M$ such that $xuvy \in L$ with $x,y \in M^*$,
If there exists a v' in M such that $uv = u'v'$ with $u' \in VT+$ and
$|v'| > |v|$, then there exists $w \in$ M such that $w = uv$

If $u' \in M$, the creation of the compound word w resolves the ambiguity between uv and u'v'. Starting from a grammar that does not possess property (2), one can add rules until the property is satisfied. This problem is related to the problem of segmentation [Bel 87b].

7. Sentence recognition (RND and ORD grammars)

The algorithm below, which gives priority to the order of rules (criterion D4) over the derivation position (criterion D1), replaces the canonical right-contextual derivation in the RND and ORD grammars:

Procedure REWRITE(i)
start
search for the right-hand side of fi in the sentence starting from the right

```
replace the occurrence found with the left-hand side of
fi end

Procedure SATURATE(i)
start
while the rule fi is applicable, repeat: start
      REWRITE(i)
      test := 1
      end
end

Procedure ANALYZE
start
repeat:
      start
      i := number of the last rule test
      := 0
      repeat:
            start
            SATURATE(i
            )
            i := i - 1
            end
      until (i=0) or (test=1) end
until (test=0) end
```

This faster algorithm can be used if the (sufficient) dual condition is satisfied: all left and right contexts of the rules use only symbols from the external alphabet, and the patterns of the right-hand sides do not overlap (see 7). The chunk rule must also always be observed.

8. Example of a qai'da: structure and vocabulary

We will demonstrate, using a real-world example, how to formulate production rules that are as general as possible, and then how to modify the grammar to account for the stress constraints revealed by experimentation. The qai'da in question is interpreted over a 16-beat measure
16 beats, each beat being divided into 6 equal parts. The examples of variations given in the appendix are tabulated at a rate of 4 beats (24 bowls) per line. Example [1] is the prototype of the composition, which can be formulated as follows using the notation conventions defined in paragraph (2):

```
(= KP) CB * (: KP) CK
```

from which we derive the rules for generating the fixed patterns KP, CB, and CK:

```
KP   <->   dhin--dhagenadha--dhagenadhatigegenakadhinedhinaghina
CK   <->   tagetirakitadhin--dhagenadhatigegenakadhinedhinaghina
etc. see grammar no. 5 in the appendix
```

The same grammar defines variants of CB and CK to which lower weights are assigned: <10> instead of <100>.

Variations of the S1V type replace KP with a chain of E24 motifs that must be repeated in mirror image in the second part. S2V variations span two full measures. The structures of these variations are defined by rules 3 and 4 of grammar no. 2:

```
S2V   <-> (= KP CB) (= E24) CB * (: KP CB) (: E24) CK
S1V   <-> (= E24) CB * (: E24) CK
```

The other rules of Grammar No. 2 (see appendix) are used to define repetitions or mirrorings employed by certain variants of S1V and S2V.

For the intermediate alphabet, we generally use the following convention: An, Bn, etc., represent variables whose instantiation is a string of terminal symbols of length n. The words (or movable patterns) were initially listed as follows:

```
A3    <-> dhin--
A3    <-> dha--
A3    <-> dhagena
A4    <-> tirakita
A6    <-> dhagenadhin--
A6    <-> dhagenadha--
A6    <-> dha-dha-dha-
A6    <-> da-ta-da-
A6    <-> dhinedhinedhine
A6    <-> dhinedha-dhine
A6    <-> tagetirakita
A6    <-> dhinedhinaghina
A6    <-> tinetinakina
A6    <-> dhatigegenaka
A12   <-> dhatigegenakatinetinakina
```

9. Example (continued): permutations

Grammar No. 3 must transform the variables B24, B12, or B6 (terminal symbols of Grammar No. 2) into sequences of the variables A3, A4, A6, and A12 (start symbols of Grammar No. 4). It is interesting to trace the process that led to its formalization. The first grammar considered was as follows:

```
LIN
B6 <-> A A A A A A
B12 <-> A A A A A A A A A A A A
B24 <-> A A A A A A A A A A A A A A A A A A A A A A A A
A A A A <-> A3
A A A A <-> A4
A A A A A A <-> A6
A A A A A A A A A A A A <-> A12
```

This grammar is correct in the sense that any sequence of symbols A3, A4, and A6 is a derivation of B24 if the sum of the indices equals 24. However, some derivations of B24 lead to a dead end, such as:

```
A3 A6 A3 A4 A3 A4 A A
```

It is easy to see that there are never more than two isolated A's remaining. To eliminate them, the analyst creates a "bridge" in the derivation tree by connecting all dead ends to a valid path. To do this, simply insert a new grammar between #3 and #4:

```
ORD
A A          --> A2
A3 A         --> A4
#A3 A        --> A #A3   [Negative context A3]
A6 A2        --> A4 A4
#A6 A2       --> A2 #A6 [Negative context A6]
```

?These rules allow isolated A's to be moved to the left and then absorbed. In this case, elimination is always possible. The rules above are not used in analysis (single arrow "-->").

10. Accentuation

Experiments show that a large number of variations synthesized by the previous grammar are not acceptable. This is the case with example [5] cited in the appendix. In fact, most words cannot be permuted arbitrarily: since this composition is interpreted at a very fast tempo, the stress must fall on a beat or a half-beat. (Each beat is divided into six units.) New variables must be chosen to designate the words: each of them now represents a sequence of events that begins on a strong beat. In the specific case of dhagenadhin--, the stress can fall on dha as well as on dhin. This word is therefore represented by "A3 A3" and no longer by A6. Certain six-unit motifs, on the other hand, are subject to specific contextual constraints. The corresponding words are designated by the variable D6.

Grammar No. 4 becomes: RND

```
A3    <-> dhin--
A3    <-> dha--
A3    <-> dhagena
A3 A3 <-> dhagenadhin--
A3 A3 <-> dhagenadha--
A4    <-> tirakita
D6    <-> dha-dha-dha-
D6    <-> dha-ta-dha-
D6    <-> dhinedha-dhine
A6    <-> dhinedhinedhine
A6    <-> tage...
etc... Other rules remain unchanged
```

Grammar No. 3 still needs to be rewritten to account for accentuation. This is a left-linear grammar that allows us to express this constraint. First, we write down all the production rules that allow B24 to be broken down into a sequence of A3, A4, A6, and D6. We then eliminate those that produce incorrect placements:

```
LIN
B24   <-> A3 B21
B24   <-> A4 B20
B24   <-> A6 B18
B24   <-> D6 B18
B21   <-> A3 B18
B21   <-> A4 B17 removed:     emphasis    on the 8th symbol
B21   <-> A6 B15
B21   <-> D6 B15 removed:     emphasis    incorrect for D6
etc...
```

A new problem arises in synthesis: in many cases, the Bol Processor produces sequences "A4 A4 A4 A4 A4 A4" that are deemed unacceptable due to their repetitiveness. This problem is solved by introducing the negative context #A4 into rule 27:

```
[27] #A4 B12 <-> #A4 A4 B8
```

so that no more than three consecutive A4s can be produced (or accepted).

?11. Templates

A phrase is analyzable only if it is represented using parentheses, pointers, and gender markers (structural symbols, see 3). It is possible, however, to insert these symbols by placing the unstructured phrase (as entered by the musicologist) onto a template. The dots represent the slots for terminal symbols:

```
(= (= ......) * (: ......) .............) ........................
etc.
```

The Bol Processor is equipped with a feature that generates all the templates for a given grammar (see "Grammar" No. 6 in the appendix). To parse a sentence, simply enter it without any structural symbols. The inference engine then runs the analysis on all compatible templates.

12. Rule Validation and Generalization

Rule validation can be summarized as follows: the weights of all rules in the transformational grammars are reset to zero. Then a representative sample of (correct) sentences is subjected to the membership test, ensuring that each time a rule is used, its weight is incremented. It is worth reconsidering the rules whose weights remain zero after analyzing numerous variations, as they may reveal possibilities unexplored by the musician. To determine their validity, one can return to the synthesis and assign them a

.

This method, which allows, starting from a grammar assumed to contain the entire language, the elimination of derivations leading to incorrect sentences, presupposes a certain independence of the rules: in the example above, all motifs recognized by grammar no. 4 (see appendix) should correspond to different variable names. Grammar No. 3 would contain the rules producing all possible permutations of these variables, and validation would then allow us to eliminate those that are invalid.

Whether one proceeds by creation (learning and inductive reasoning) or by rule validation (deductive reasoning), it is necessary, after each modification, to unify the variables that lead to the same derivations, and then to remove the rules that the unification has rendered redundant [Bel 87b]. These two techniques also converge on the principle that any generalization or particularization of the grammar must be open to question if its effects are contradicted by experience. The modeling of these analytical methods, which allow the musicologist to systematically manage their knowledge of the language and the hypotheses used to develop it, currently constitutes the bulk of the theoretical work on this project.

Bibliography

[Bel 87a] B. Bel: The Grammars and the Inference Engine of the Bol Processor; Representation and Knowledge Processing Group, CNRS (Marseille), document 237, 1987

[Bel 87b] B. Bel: Grammars of rhythmic languages; Thesis in Mathematics and Computer Science; Artificial Intelligence Group, Faculty of Sciences, Luminy (Marseille), 1987

[Hart 76] J.M. Hart: Right and Left Parses in Phrase-Structure Grammars; Information & Control 32, pp. 242-262, 1976

[Hart 80] J.M. Hart: Derivation Structures for Strictly Context-Sensitive Grammars; Information & Control 45, pp. 68-89, 1980

[Kain 81] R. Kain: Automata Theory: Machines and Languages; Krieger (Malabar), 1981

[Kippen 86] J.R. Kippen: An Ethnomusicological Approach to the Analysis of Musical Cognition; Music Perception, University of California (San Diego), forthcoming

[Révész 85] G.E. Révész: Introduction to Formal Languages; McGraw-Hill (Singapore), 1985

[Salomaa 73] A. Salomaa: Formal Languages; Academic Press (New York), 1973

Appendix: Example of a qa'ida (Lucknow style)

The terminal alphabet of grammar No. 5 and the structural symbols of grammar No. 2 are explained in paragraph (2). The numbers in square brackets <...> represent the weights of the rules.

GRAM#1 [1] RND [Variation families]

```
GRAM#1 [2] <100>  LEFT  S <-> S2V   [double]
GRAM#1 [3] <100>  LEFT  S <-> S1V   [single]
-----------------------------------------------------------------------------------------------------------
GRAM#2 [1]              RND         [Structures]
GRAM#2 [2] <1>          LEFT E24 <-> KP
GRAM#2 [3] <100>  LEFT  S2V <-> (= KP CB ) (= E24 ) CB *(: KP CB ) (: E24 ) CK
GRAM#2 [4] <100>  LEFT  S1V <-> (= E24 ) CB *(: E24 ) CK
GRAM#2 [5] <100>  LEFT  E24 <-> B24
GRAM#2 [6] <33>   LEFT  E24 <-> E12 E12
GRAM#2 [7] <33>   LEFT  E24 <-> (= E12 ) (: E12 )
GRAM#2 [8] <33>   LEFT E24 <-> (= E12 ) *(: E12 )
GRAM#2 [9] <100> LEFT E12 <-> B12
GRAM#2 [10] <50> LEFT E12 <-> (= B6 ) *(: B6 )
-----------------------------------------------------------------------------------------------------------
GRAM#3 [1]  LIN         [Permutations]
GRAM#3 [2] <100>        B6 <-> A6
GRAM#3 [3] <100>        B24 <-> A6 B18
GRAM#3 [4] <2>          B24 <-> D6 B18
GRAM#3 [5] <20>         B24 <-> A4 B20
GRAM#3 [6] <100>        B24 <-> A3 B21
GRAM#3 [7] <100>        B21 <-> A6 B15
GRAM#3 [8] <100>        B21 <-> A3 B18
GRAM#3 [9] <100>        B20 <-> A4 B16
GRAM#3 [10] <100>       B20 <-> A6 B14
GRAM#3 [11] <100>       B18 <-> A6 B12
GRAM#3 [12] <5>         B18 <-> D6 B12
GRAM#3 [13] <100>       B18 <-> A4 B14
GRAM#3 [14] <100>       B18 <-> A3 B15
GRAM#3 [15] <100>       B16 <-> A4 B12
GRAM#3 [16] <100>       B16 <-> A6 B10
GRAM#3 [17] <100>       B15 <-> A6 B9
GRAM#3 [18] <100>       B15 <-> A3 B12
GRAM#3 [19] <100>       B14 <-> A4 B10
GRAM#3 [20] <100>       B14 <-> A6 B8
GRAM#3 [21] <100>       B12 <-> A12
GRAM#3 [22] <100>       B12 <-> A6 B6
GRAM#3 [23] <5>         B12 <-> D6 B6
GRAM#3 [24] <5>         B12 <-> A6 D6
GRAM#3 [25] <100>       B12 <-> A3 B9
GRAM#3 [26] <100>       B10 <-> A6 A4
GRAM#3 [27] <100>       #A4 B12 <-> #A4 A4 B8   [Negative context A4]
GRAM#3 [28] <100>       ) B12 <-> ) A4 B8
GRAM#3 [29] <5>         B10 <-> A4 D6
GRAM#3 [30] <100>       B9 <-> A3 B6
GRAM#3 [31] <5>         B9 <-> A3 D6
GRAM#3 [32] <100>       B9 <-> A6 A3
GRAM#3 [33] <100>       B8 <-> A4 A4
GRAM#3 [34] <100>       B6 <-> A3 A3
-----------------------------------------------------------------------------------------------------------
GRAM#4 [1] RND          [Movable words or patterns]
GRAM#4 [2] <30>   A3 <-> dhin--
GRAM#4 [3] <30>   A3 <-> dha--
GRAM#4 [4] <30>   A3 <-> dhagena
GRAM#4 [5] <100> A3 A3 <-> dhagenadhin--
```

```
GRAM#4 [6] <100>  A3 A3 <-> dhagenadha--
GRAM#4 [7] <100>  A4 <-> tirakita
GRAM#4 [8] <50>   D6 <-> dha-dha-dha-
GRAM#4 [9] <50>   D6 <-> dha-ta-dha-
GRAM#4 [10] <10>  A6 <-> dhinedhinedhine
GRAM#4 [11] <100>       D6 <-> dhinedha-dhine
GRAM#4 [12] <100>       A6 <-> tagetirakita
GRAM#4 [13] <100>       A6 <-> dhinedhinaghina
GRAM#4 [14] <10>  A6 <-> tinetinakina
GRAM#4 [15] <100>       A6 <-> dhatigegenaka
GRAM#4 [16] <100>       A12 <-> dhatigegenakatinetinakina
```

```
GRAM#5 [1]  RND         [Fixed Patterns]
GRAM#5 [2] <100>  LEFT  KP <-> dhin--dhagenadha-
-dhagenadhatigegenakadhinedhinaghina
GRAM#5 [3] <100> LEFT CK <-> tagetirakitadhin-
-dhagenadhatigegenakadhinedhinaghina
GRAM#5 [4] <10>   LEFT CK <-> tagetirakitaghina-
dhagenadhatigegenakadhinedhinaghina
GRAM#5 [5] <100> LEFT CB <-> tagetirakitadhin-
-dhagenadhatigegenakatinetinakina
GRAM#5 [6] <10>   LEFT CB <-> tagetirakitaghina-
dhagenadhatigegenakatinetinakina
```

The templates generated by this grammar:

```
GRAM#6 [1]              TEM         [templates]
GRAM#6 [2] (=.................................................)
(=.........................) .........................
*(.................................................)
(:.........................) .........................
GRAM#6 [3] (=.................................................) (=.............
(=.......) *(:.......) ) .........................
*(:.................................................) (:.............. (=.......)
*(:.......) ).........................
GRAM#6 [4] (=.................................................) (= (=......)
*(:.......) ..............) .........................
*(:.................................................) (: (=......) *(.......)
..............).........................
GRAM#6 [5] (=.................................................) (= (=......)
*(:.......) (=......) *(:.......)).........................
*(:.................................................) (: (=......) *(.......)
(=......) *(:......) ) .........................
GRAM#6 [6] (=.................................................) (=
(=.............) (:.............) ) .........................
*(:.................................................) (: (= .............)
(:.............) ) .........................
GRAM#6 [7] (=.................................................) (= (= (=......)
*(:.......) ) (: (=......) *(:.......) ) ) .........................
*(:.................................................) (: (= (=......)
*(:.......) ) (: (=......) *(:.......) ) ) .........................
GRAM#6 [8] (=.................................................) (=
(=.............) *(:.............) ) .........................
```

```
*(:.....) (: (= .....)
*(:.....) ).....
GRAM#6 [9] (=.....) (= (= (=......)
*(:......) ) *(: (=......) *(:......) ) ) .....
*(:.....) (: (= (=......)
*(:......) ) *(: (=......) *(:......) ) ) .....
GRAM#6 [10] (=.....) .....
*(:.....) .....
GRAM#6 [11] (=..... (=......) *(:......) ) .....
*(:..... (=......) *(:......)) .....
GRAM#6 [12] (= (=......) *(:......) .....).....*(:
(=......) *(:......) .....) .....
GRAM#6 [13] (= (=......) *(:......) (=......) *(:......) )
..... *(: (=......) *(:......)(=......) *(.......))
.....
GRAM#6 [14] (= (=.....) (:.....) ).....*(:
(=.....) (:.....) ) .....
GRAM#6 [15] (= (= (=......) *(:......) ) (: (=......) *(:......) ) )
..... *(: (= (=......)*(:......) ) (: (=......) *(.......)
) ) .....
GRAM#6 [16] (= (=.....) *(:.....) ).....*(:
(=.....) *(:.....)) .....
GRAM#6 [17] (= (= (=......) *(:......) ) *(: (=......) *(:......) ) )
..... *(: (= (=......) *(:......) ) *(: (=......)
*(:......) ) ) .....
?Examples of variations


      Prototype of the qa‘ida:

[1]   dhin--dhagena     dha--dhagena      dhatigegenaka     dhinedhinaghina
      tagetirakita      dhin--dhagena     dhatigegenaka     tinetinakina
      tin--takena ta--takena tatikekenaka tinetinakina
      tagetirakita      dhin--dhagena     dhatigegenaka     dhinedhinaghina

that is: (=dhin--dhagena                  dha--dhagena      dhatigegenaka
                        dhinedhinaghina ) tagetirakita      dhin--dhagena
                        dhatigegenaka     tinetinakina
      *(:tin--takena    ta--takena tatikekenaka       tinetinakina )
      tagetirakita      dhin--dhagena     dhatigegenaka     dhinedhinaghina

      Simple variation:

[2]   dhin--dhatige     genakadhin--      tirakitatira      kitatirakita
(=E24)
      tagetirakita      dhin--dhagena     dhatigegenaka     tinetinakina
      CB
      tin--tatike       kenakatin--       tirakitatira      kitatirakita
*(:E24)
      tagetirakita      dhin--dhagena     dhatigegenaka     dhinedhinaghina
      CK

that is: (=dhin--       genakadhin--      tirakitatira      kitatirakita)
dhatige
      tagetirakita      dhin--dhagena     dhatigegenaka     tinetinakina
      *(:tin--tatike    kenakatin--       tirakitatira      kitatirakita )
```

tagetirakita dhin--dhagena dhatigegenaka dhinedhinaghina

Double variations:

[3] dhin--dhagena dha--dhagena dhatigegenaka dhinedhinaghina
(=KP
tagetirakita dhin--dhagena dhatigegenaka tinetinakina
CB)
dhin--dhagena dha-dha-dha- dhatigegenaka tinetinakina
(=E24)
tagetirakita dhin--dhagena dhatigegenaka tinetinakina
CB
tin--takena ta--takena tatikekenaka tinetinakina *(:KP
taketirakita tin--takena tatikekenaka tinetinakina
CB)
dhin--dhagena dha-dha-dha- dhatigegenaka tinetinakina
(:E24)
tagetirakita dhin--dhagena dhatigegenaka dhinedhinaghina
CK

[4] dhin--dhagena dha--dhagena dhatigegenaka dhinedhinaghina
tagetirakita dhin--dhagena dhatigegenaka tinetinakina
dhinedhinaghina dhinedha-dhine dhatigegenaka tinetinakina
tagetirakita dhin--dhagena dhatigegenaka tinetinakina
tin--takena ta--takena tatikekenaka tinetinakina
taketirakita tin--takena tatikekenaka tinetinakina
dhinedhinaghina dhinedha-dhine dhatigegenaka tinetinakina
tagetirakita dhin--dhagena dhatigegenaka dhinedhinaghina

[5] dhin--dhagena dha--dhagena dhatigegenaka dhinedhinaghina
tagetirakita dhin--dhagena dhatigegenaka tinetinakina
dhin--tiraki tadhagenadhati gegenakatira kitatirakita
tagetirakita dhin--dhagena dhatigegenaka tinetinakina
tin--takena ta--takena tatikekenaka tinetinakina
taketirakita tin--takena tatikekenaka tinetinakina
dhin--tiraki tadhagenadhati gegenakatira kitatirakita
tagetirakita dhin--dhagena dhatigegenaka dhinedhinaghina

(Incorrect variation: the accent on "dha" is misplaced)