Pregroup Grammars for Chords

Richard Terrat

PREGROUP GRAMMARS FOR CHORDS

Richard G. TERRAT

LIRMM/CNRS
161, rue ADA
34000 Montpellier
France
terrat@lirmm.fr

&

IRCAM
1,Place Igor Stravinsky
75004 Paris
France
terrat@ircam.fr

ABSTRACT

Pregroups had been conceived as an algebraic tool to recognize grammatically well-formed sentences in natural languages [3].

Here we wish to use pregroups to recognize well-formed chords of pitches, for a given definition of those chords. We show how a judicious choice of basic and simple types allows a context-free grammatical description.

Then we use the robustness property to extend the set of well-formed chords in a simple way.

Finally we argue in favor of an utilization of pregroups grammars for the recognition and classification of chord sequences.

1. INTRODUCTION

Following the seminal work of Noam CHOMSKY [5] attempting to provide a formal description of the syntax of natural languages, many researchers have provided various such formalisms in the musical field.

Formal descriptions are proved to be useful tools either for a better comprehension of musical structures (with recognition grammars), or with the aim of automatic composition or improvisations (with generative grammars).

Nevertheless either these grammars are easy to implement (e.g. context-free) but insufficient to account for complex musical structures, or heavily dependent of the context and thus difficult to describe and process.

Thanks to recent works we can hope to get out of this dilemma. For recognition grammars, pregroup grammars play an important role in that work.

In this paper we will first summarize in §2 the main definitions and properties of pregroups, then §3 will present the concept of type, §4 the one of typing used for the syntactic units of the language while §5 specifies the method of recognition ; §6 will show how pregroups can be used for simple examples of chord grammars and §7 concludes with current prospects.

2. PREGROUPS

2.1. Definition

A pregroup is a partially ordered monoid in which each element a has both a left adjoint a^l and a right adjoint a^r satisfying :

Contraction

$$a^l \cdot a \rightarrow 1 \qquad a \cdot a^r \rightarrow 1 \qquad (1)$$

Expansion
(Generally not needed for syntax verification)

$$1 \rightarrow a \cdot a^l \qquad 1 \rightarrow a^r \cdot a \qquad (2)$$

where "→" denotes the partial order relation, "." the multiplication and "1" the unit of the monoid.
(The multiplication sign will be omitted in the sequel).

2.2. Properties

$a\ 1 = a = 1\ a$ (3)
1 is the unit of the monoid

$(a\ b)\ c = a\ (b\ c)$ (4)
multiplication is associative

$a \rightarrow b$ and $c \rightarrow d$ implies $a\ c \rightarrow b\ d$ (5)
order is compatible with multiplication

$a\ b \rightarrow 1 \rightarrow b\ a$ implies $a = b^l$ and $b = a^r$ (6)
adjoints are unique

$(a\ b)^l = b^l\ a^l \qquad (a\ b)^r = b^r\ a^r$ (7)
adjunction is quasi-distributive

$a \rightarrow b$ implies $b^l \rightarrow a^l$ and $b^r \rightarrow a^r$ (8)
adjunction reverses the order

$a^{lr} = a = a^{rl}$ (9)
no mixed adjoints

Properties (3) to (5) are part of the definition of a monoid ; properties (6) to (9) can be derived from definitions and their proofs can be found in [3].

3. TYPES

3.1. Basic types

As linguists, we will work with the free pregroup generated by a partially ordered set of so called *basic types* which are elements of an enumerable set.
Bold face symbols : ***a, b, c,*** ... will range over basic types.

The order between basic types is declared as such, and is stored in a table.

$\mathbf{B} = \{ \boldsymbol{a, b, c, \ldots.} \}$

3.2. Simple types

Simple types are formed from basic types and their adjoints. Thus they form an infinite enumerable set :

$$\sum = \{ \ldots \boldsymbol{a}^{ll}, \boldsymbol{a}^{l}, \boldsymbol{a}, \boldsymbol{a}^{r}, \boldsymbol{a}^{rr} , \ldots \boldsymbol{b}^{ll}, \boldsymbol{b}^{l}, \boldsymbol{b}, \boldsymbol{b}^{r}, \boldsymbol{b}^{rr} , \ldots \}$$

Simple types inherit the order from the basic types as follows :

if $\boldsymbol{a} \rightarrow \boldsymbol{b}$ then $\boldsymbol{b}^{l} \rightarrow \boldsymbol{a}^{l}$, $\boldsymbol{b}^{r} \rightarrow \boldsymbol{a}^{r}$, $\boldsymbol{a}^{ll} \rightarrow \boldsymbol{b}^{ll}$, $\boldsymbol{a}^{rr} \rightarrow \boldsymbol{b}^{rr}$,

Hence to check whether $a \rightarrow b$, it is sufficient to check whether a and b have the same exponent, i.e. $a = \boldsymbol{a}^{s}$, $b = \boldsymbol{b}^{s}$ where $\boldsymbol{a}$ and $\boldsymbol{b}$ are basic types and s consists of a finite number n of repetitions of the same suffix, either ^{l} or ^{r}.

If a and b have the same exponent, then $a \rightarrow b$ if either n is even and $\boldsymbol{a} \rightarrow \boldsymbol{b}$, or n is odd and $\boldsymbol{b} \rightarrow \boldsymbol{a}$. Otherwise, neither a reduces to b nor b to a.

Contractions of simple types can be understood as rules such as :

$\boldsymbol{a}^{ll}\, \boldsymbol{a}^{l} \rightarrow 1$, $\boldsymbol{a}^{l}\, \boldsymbol{a} \rightarrow 1$, $\boldsymbol{a}\, \boldsymbol{a}^{r} \rightarrow 1$, $\boldsymbol{a}^{r}\, \boldsymbol{a}^{rr} \rightarrow 1$

3.3. Types

Strings of simple types will be called compounded types or more simply : types.

The order $\rightarrow$ can be read as "reduces to".

4. TYPING

4.1. Assignment

As a first step, the language to be analyzed has to be described in some grammatical terms. The smallest syntactic units (words) are supposed to be in a dictionary.

The next step is to define the set of basic types and its ordering. Then, to every word, one, or more generally several types have to be assigned.

The typing assignment must be done in such a way that the sequence of words constitutes a well-formed construct if and only if the corresponding string of its types reduces to the basic type corresponding to the construct. If more than one type is assigned to a word, it is sufficient that one of these types yields a string reducing to the final type.

Every typing respecting this condition is said to be *correct*, i.e. it allows to recognize only well-formed constructs and *complete*, i.e. it recognizes all well-formed constructs.

4.2. Extension and robustness

An important property of typing is the possibility to extend the constructs in a monotonic way, i.e. without changing the properties of the previous string : this is called the *robustness*.

These extensions can be done in two ways :

4.2.1. Assigning new types

If a new type y is assigned to a word w, without changing the previous assigned types, and x is one of the previous assigned types such that $y \rightarrow x$ then every string of words recognized as well-formed using the type assignment x for w is also well-formed using y.

4.2.2. Extension by new basic types

Let **B** be a given set of basic types. It is possible to extend this set by declaring new basic types and their order relations, obtaining thus a larger set of basic types $\mathbf{B'} \supset \mathbf{B}$. Then the free pregroup **P'** generated by **B'** includes the free pregroup **P** generated by **B**.

If a and b belong to **P** and $a \rightarrow b$ can be derived in **P**, it can also be derived in **P'**.

4.3. Conservativity

If $a \rightarrow b$ can be derived in some pregroup **P'** and both a and b belong to a smaller pregroup $\mathbf{P} \subset \mathbf{P'}$, then the whole reduction can be done in **P**.

The proof of this non trivial property can be found in [3].

5. LANGUAGE ANALYSIS

5.1. Step by step

The robustness and conservativity properties allow to proceed step by step to analyze a language. One can take a subset of the language and show its correctness and completeness. Then one can extend the fragments either by assigning new types to words or by adding new basic types and verifying that the typing involving the new types is also correct and complete for the new construct.

5.2. Type checking algorithms

A type checking algorithm is an algorithm deciding whether $a \rightarrow b$ for arbitrary types a and b. In fact, it is sufficient to prove that $b^1\ a \rightarrow 1$ since the following property :

$a \rightarrow b$ if and only if $b^1\ a \rightarrow 1$ (10)

It has been shown that such an algorithm exists and that its complexity is at most $O(n^3)$

6. A SIMPLE CHORD GRAMMAR

6.1. Rules for "non dissonant" chords

Let us call "non dissonant" a chord which has no harmonics in "critical bands" [4]. A non dissonant chord may be "non consonant" since to be consonant the pitches of the chord must have several common harmonics or at least have close harmonics that produces only vibratos. Thus a chord may be neither dissonant nor consonant. So are most chords in the equal temperament.

Many harmony handbooks give rules for generation or recognition of these chords [2]. These rules are always expressed in a natural language.

On the other hand, classical grammars [1] are used for chord sequences descriptions [6] especially in Jazz music. Unfortunately these grammars are context sensitive and some of these rules must be completed by natural language sentences.

More recently, categorical grammars have been used to cope with these difficulties [7] [8].

6.2. A first simple approach

6.2.1. Definition

Let us call chord a non dissonant chord. Let us give the following simple definition :

a chord is a sequence of at least 3 pitches such that the distance between two successive pitches of the chord is at least 3 semi-tones.

This is a simple way to prevent close harmonics from appearing in critical bands.

This musical definition can be turned into an arithmetic one :

0 is assigned to the first pitch of the chord called the root

the other pitches of the chord are made of an increasing sequence of at least 2 integers ≥3 such as the difference between 2 consecutive elements of the sequence is ≥ 3.

Each of these integers represents the chromatic distance of a pitch to the root.

Let us now define a grammar where words are the pitches and correct sentences are the chords.

6.2.2. Basic types

We define the following set of basic types

Ch	for the chord
P (i)	for the last pitch
P (i,j)	for the other pitches

i is related to the chromatic distance of a pitch to its root ; j is related to the rank of a pitch

and the following order :

P (i,j) → P (k,j) iff k > i (11)

P (i,j) → P (i) (12)

6.2.3. Typing

Each pitch of the chord is represented by an integer measuring its chromatic distance to the root : for example a m7b9 chord is represented by the following sequence : 3 7 10 13

To each of these integers, say n, we assign one of the 3 following types :

P (n-3)^r Ch $n \geq 6$
for the pitch in the last position

P (n,1) $n \geq 3$
for the pitch in the first position

P (n-3,k)^r P(n,k+1) $n \geq 3(k+1)$
for the pitches in an intermediate position k

For the above example, the typing is :

3	P (3,1)
7	P (4,1)^r P(7,2)
10	P (7,2)^r P (10,3)
13	P (10)^r Ch

leading to the following type of the whole sequence :
P (3,1) P (4,1)^r P (7,2) P (7,2)^r P (10,3) P (10)^r Ch
As P (3,1) → P (4,1) according to (11)
and P(10,3) → P (10) according to (12)
the previous type reduces to Ch proving the correctness of the chord

Let us now take an other example, such as a m6 chord, represented by the following sequence : 3 7 9
and its type : P (3,1) P (4,1)^r P (7,2) P (6)^r Ch
that reduces to : P (7,2) P(6)^r Ch
which is irreducible.
Such a chord is, according to our definition, incorrect.

6.3. An extension

The above definition for a chord appears to be too restrictive. According to the property of robustness, we can extend the assignment of types without changing the previous language.

For example, if we accept chords with a minimum chromatic distance of 2 semi-tones between its pitches, the following assignments have to be added :

P (n-2)^r Ch $n \geq 4$
for the pitch in the last position

P (n,1) $n \geq 2$

for the pitch in the first position

$P\ (n\text{-}2,k)^r\ P(n,k+1)$ $\quad n \geq 2(k+1)$

for the pitches in an intermediate position k

So, the above example of a m6 chord : 3 7 9 can be assigned to the following type :

$P\ (3,1)\ P\ (5,1)^r\ P\ (7,2)\ P\ (7)^r$ Ch

that reduces to Ch and becomes correct

6.4. Chord specifications

In the previous simple examples, one single basic type Ch has been assigned to a valid chord. It is possible to refine the definitions in order to specify more precisely the kinds of chords.

For example, if we want to distinguish major and minor chords, different specific assignments of the first pitch, i.e. the third degree, must be done. They will be associated to 2 different assignments of the chord, namely the major and minor ones.

New basic types:

m	for minor
M	for major
Chm	for a minor chord
ChM	for a major chord

New typing :

m P(3,1) for the first pitch iff its value is 3

M P(4,1) for the first pitch iff its value is 4

$P\ (n\text{-}2)^r\ m^r$ Chm $\quad n \geq 4$

$P\ (n\text{-}2)^r\ M^r$ ChM $\quad n \geq 4$

for the pitch in the last position

A simple minor chord will be represented by the sequence : 3 7

and its type by : m $P(3,1)\ P(5)^r\ m^r$ Chm

that reduces to : Chm

In the same way, we can specify 7^{th} chords, and so on. This may lead to 4 classic chord types as : ChM, Chm, Ch7, Chm7.

6.5. Chord sequences

Furthermore, we can add the relative root of the chord in its definition in order to specify its degree relative to a given tone. Thus we can distinguish, for instance, a Vm7 from a IIm7 in a chord sequence. Each of these chords will thus be assigned to a proper single type or a sequence of such types.

At this point, the analysis of a chord sequence becomes possible. A new grammar at a higher level will now consider chords as clauses and chord sequences as sentences [8]. This is a classical way of analyzing a complex language level by level.

To each basic type(s) representing a specific chord we will now assign new types.

7. CONCLUSION

We have shown that pregroup grammars can be used to describe the context-sensitive syntax of chords. Further works involving sequences of chords are under investigation. For example, the six rules of Steedman [6] concerning various chord sequences of blues are described by a context-sensitive generative grammar containing some assertions expressed in a natural language. Steedman himself is now using combinatory categorical recognition grammars [7] to cope with context sensitivity. In that direction pregroup grammars may be possible for context-free formal recognition (rather that generation).

Furthermore, as pregroup grammars are able to recognize specific sentence constructions in natural languages, they may also be able to recognize specific chord sequences allowing their classification in some particular context ; for example in jazz music : blues, rag, anatole (chord sequence of "I got rhythm" – G. Gershwin -) etc .. Such classifications would be a useful help for indexing pieces in large databases, especially on the WWW, either by extracting chord files from "real books" or possibly with the help of software extracting chord sequences from scores or even audio files [5]

8. REFERENCES

[1] CHOMSKY Noam (1979) – *Structures syntaxiques* – Editions du Seuil

[2] EBERHARD Daniel Mark (2001) - *Das Wichtigste in der Musik steht nicht in den Noten* – Skript zum (Jazz-) Improvisations Workshop

[3] LAMBEK Joachim (2000) - *An algebraic approach to English sentence* - unpublished lecture notes, McGill University, QC, Canada

[4] PIERCE John (1984) – *Le son musical* – Ed. Pour la Science

[5] SHEH Alexander, ELLIS Daniel P. W. (2003) – *Chord Segmentation and Recognition using EM-Trained Hidden Markov Models* – ISMIR 2003

[6] STEEDMAN Mark (1984) – *A Generative Grammar for Jazz Chord Sequences* – Music Perception 2, 52-77 1984

[7] STEEDMAN Mark (2003) – *Formal Grammars for Computational Music Analysis : The Blues and the Abstract Truth* – INFORMS Atlanta October 2003

[8] TERRAT Richard (2004) – *A Pregroup grammar for chord Sequences* – SMC 04 – Paris October 2004