Categorial Grammar and Harmonic Analysis

Flávio S. Corrêa da Silva   Fábio Kon

Instituto de Matemática e Estatística da Universidade de São Paulo
Cx. Postal 66281 – 05389-970
São Paulo (SP) – Brazil – email: {fcs, kon}@ime.usp.br

Abstract

It is rather commonplace in everyday conversation to refer to the *"Language of Music"*. However, we believe the whole apparatus already built for the analysis of natural language has not been yet as thoroughly used for the analysis of musical phenomena as it could have been. In this article we present some initial ideas towards extending the application of this apparatus for the better understanding of *"Music as Language"*.

In this paper, we apply some techniques from *Categorial Grammar* to represent a simple problem of music theory, which we believe nevertheless to be of widespread interest: functional harmonic analysis. We propose an encoding of the harmonic functions of chords as syntactic categories, and show how the generation of proofs of "harmonic well-formedness" of cadences can be implemented and used as a tool to verify and to display the functional harmonic structuring of cadences.

Keywords: music analysis, harmonic analysis, categorial grammar, syntactic calculus, substructural logics.

1 Introduction

It is commonplace in everyday conversation to refer to the *"Language of Music"*. Indeed, the study of musical phenomena as linguistic objects has been developed by many authors (see e.g. [BCe84, Hol80, LJ83, Sch89]). In this article we present some initial ideas towards extending the application of this apparatus for the better understanding of *"Music as Language"*. More specifically, we employ techniques from *Categorial Grammar* to represent a rather specific and simple problem of music theory, which we believe nevertheless to be of widespread interest: functional harmonic analysis [Bri79].

The aim of Categorial Grammar [Ben87, Ben90, Ben91, Lam58, Lam89] is the analysis of syntactic well-formedness of sentences. The fundamental concept underlying Categorial Grammar is that of *syntactic categories*, which are classes to which words in a sentence must belong. Syntactic categories can be organised as formulae of some substructural logic – e.g. the so-called *Lambek Calculus* [Lam58] – in such way that syntactic well-formedness can be checked via an appropriate *proof theory* related to the logic.

In this paper we propose an encoding of the harmonic functions of chords as syntactic categories and show how the generation of proofs of "harmonic well-foundedness" of cadences can be implemented and used as a tool to verify and to display the functional harmonic structuring of cadences.

In section 2 we briefly review the concepts of Lambek Calculus that we need to use in the rest of the paper. In section 3 we introduce our encoding of harmonic functions of chords as syntactic categories, and show how they can be used to check and to display the functional harmonic structuring of cadences. In section 4 we present a simple PROLOG implementation for checking the harmonic well-foundedness of cadences and displaying functions of chords.

Finally, in section 5 we discuss these results, present some conclusions and propose some future work.

2 The Lambek Calculus

J. Lambek introduced the Syntactic Calculus – most usually called Lambek Calculus nowadays – in [Lam58], as a tool to encode the English grammar, such that well-formedness of sentences could be tested deductively.

Essentially, the Lambek Calculus corresponds to classical propositional logic devoid of any structural rule, in which implication is factorised in two non-commutative connectives. Here, we consider only the implicative fragment of that Calculus, which is sufficient for what we intend to present.

A Gentzen-style presentation of implicative Lambek Calculus can be given by the following rules, in which x, y, z are syntactic categories generated by the members of a set $\mathcal{S}$ of basic syntactic categories and $\Gamma, \overline{\Gamma}, \Delta$ are sequences of syntactic categories. The sequences $\overline{\Gamma}$ are assumed to be non-empty.

axiom: $x \vdash x$.

right-inclusion:	left-inclusion:
$\dfrac{\overline{\Gamma}, y \vdash x}{\overline{\Gamma} \vdash x/y}$.	$\dfrac{\overline{\Gamma} \vdash y; \Gamma, x, \Delta \vdash z}{\Gamma, x/y, \overline{\Gamma}, \Delta \vdash z}$.
$\dfrac{y, \overline{\Gamma} \vdash x}{\overline{\Gamma} \vdash y \backslash x}$.	$\dfrac{\overline{\Gamma} \vdash y; \Gamma, x, \Delta \vdash z}{\Gamma, \overline{\Gamma}, y \backslash x, \Delta \vdash z}$.

For example, Lambek assumes that $\mathcal{S} = \{n, s\}$, in which n stands for "noun" and s stands for "sentence". The words of the English language are attached as labels to formulae, in such way that they can only occur in specific sequences from which the category "s[entence]" can be derived.

Giving a more specific example, if we assume the words `John` and `milk` to be of category "n[oun]", the word `fresh` to be of category "n/n" (a qualifier – must precede the noun it is qualifying to produce a qualified noun) and the word `likes` to be of category $n\backslash s/n$ (a transitive verb – forms a sentence if preceded by a noun – the subject – and followed by another noun – the object of the sentence), we can prove the well-formedness of `John likes fresh milk` with the deduction tree presented in figure 1 (we abbreviate `John, likes, fresh,` and `milk` by their initials `J, l, f,` and `m`).

$$\dfrac{\dfrac{\mathtt{m}\colon n \vdash \mathtt{m}\colon n \qquad \dfrac{\mathtt{fm}\colon n \vdash \mathtt{fm}\colon n \quad \mathtt{Jlfm}\colon s \vdash \mathtt{Jlfm}\colon s}{\mathtt{Jl}\colon s/n,\ \mathtt{fm}\colon n \vdash \mathtt{Jlfm}\colon s}}{\mathtt{Jl}\colon s/n,\ \mathtt{f}\colon n/n,\ \mathtt{m}\colon n \vdash \mathtt{Jlfm}\colon s} \qquad \mathtt{J}\colon n \vdash \mathtt{J}\colon n}{\mathtt{J}\colon n,\ \mathtt{l}\colon n\backslash s/n,\ \mathtt{f}\colon n/n,\ \mathtt{m}\colon n \vdash \mathtt{Jlfm}\colon s}$$

Figure 1: Deduction for "John likes fresh milk"

This deduction proves that from the sequence `J`: n, `l`: $n\backslash s/n$, `f`: n/n, `m`: n we can derive the well-formed sentence `Jlfm`: s.

We have employed this Calculus to encode the functional grammar of tonal chords, as detailed in the following sections.

3 Tonal Chords and Syntactic Categories

The set of basic syntactic categories for functional harmony must be large enough to permit the characterisation of all different functions each chord may have in a cadence. We have employed a set of three basic syntactic categories $\mathcal{S} = \{a, b, c\}$, a and c loosely corresponding to the concepts of *tonic* and *cadence*, related to Lambek's *noun* and *sentence* functions, and b

corresponding to an intermediate concept leading to the idea of *subdominant*. Intuitively, we have a as tonic, $a \setminus b$ as subdominant (fulfilled when preceded by something of category a) and $b \setminus c/a$ as a full cadence (fulfilled when preceded by some chain of chords of category b and followed by something of category a). In order to present our proposed encoding of chords as representatives of syntactic categories, we must introduce some notation.

We have adopted the (first twelve) MIDI codes for pitch values, and hence the notes **C, C♯, D** ... are denoted respectively as $0, 1, 2, \ldots$. The syntactic categories of the functions of chords can then be encoded in a dictionary like the one presented in table 1. In this dictionary, $i = 0, 1, \ldots, 11$, and these numbers are operated modulo 12, i.e. $6 + 5 = 11, 6 + 6 = 0, 6 + 7 = 1$ etc. (and, of course, in table 1 we have only a small fragment of one such dictionary).

Major Mode

entry	chord	tonality						
		i	i+1	i+3	i+5	i+7	i+8	i+10
i^1	i,i+4,i+7	a			$b\backslash c/a$	$a\backslash b$		
i^2	i,i+3,i+7, i+10							$a\backslash b$
i^d3	i,i+4,i+7, i+10				$b\backslash c/a$			
i^4	i,i+4,i+7, i+11	a				$a\backslash b$		
i^5	i,i+4,i+7, i+11,i+2	a						
i^7	i,i+3,i+8		$b\backslash c/a$	$a\backslash b$			a	

Minor Mode

entry	chord	tonality					
		i	i+1	i+4	i+5	i+7	i+9
i^1	i,i+4,i+7				$b\backslash c/a$		
i^3	i,i+4,i+7, i+10				$b\backslash c/a$		
i^6	i,i+3,i+7	a				$a\backslash b$	
i^7	i,i+3,i+8		$b\backslash c/a$				
i^8	i,i+4,i+9			$a\backslash b$			a

Table 1: Dictionary of Syntactic Categories of Chords ($i = 1, \ldots, 11$ is the root of the chord)

It should be observed that syntactic categories refer to specific tonalities and modes. We avoid referring explicitly to tonalities and to modes in our deduction trees to preserve our notation as simple as possible. Now, using the notation of table 1, if we attach the perfect major triads $0^1, 5^1, 7^1$ as labels to the categories $a, a\backslash b$ and $b\backslash c/a$, we can derive the well-formedness of the perfect cadence $\{0^1, 5^1, 7^1, 0^1\}$ (figure 2).

$$\dfrac{\dfrac{0^1 : a \vdash 0^1 : a \qquad \dfrac{0^15^1 : b \vdash 0^15^1 : b \quad 0^15^17^10^1 : c \vdash 0^15^17^10^1 : c}{0^15^1 : b, 7^10^1 : b\backslash c \vdash 0^15^17^10^1 : c}}{0^1 : a, 5^1 : a\backslash b, 7^10^1 : b\backslash c \vdash 0^15^17^10^1 : c} \qquad 0^1 : a \vdash 0^1 : a}{0^1 : a, 5^1 : a\backslash b, 7^1 : b\backslash c/a, 0^1 : a \vdash 0^15^17^10^1 : c}$$

Figure 2: Deduction for the perfect cadence

```
| ?- harmon([[c,e,g], [d,f,a], [g,b,d], [c,e,g]], X).

X = [[c,maj]] ?

yes

| ?-
```

Figure 3: Using the Theorem Prover

A theorem prover for this Calculus can be implemented in PROLOG, and in the following section we present a very simple implementation for it.

4 Functional Harmony in PROLOG

The PROLOG code for a simple implementation of a theorem prover for the Calculus presented above is introduced in the appendix following this paper. This program works as follows: given a sequence of chords $[C_1, ..., C_n]$, the procedure `genseq` converts this sequence into a sequence of sets of harmonic functions $\mathcal{F}_i$ that each chord C_i can have. From these, the procedure `cadence` selects the functions $f_i \in \mathcal{F}_i$ such that from $f_i : i = 1, ..., n$ we can derive the function c of any tone and mode. These functions are then presented as solutions, with the corresponding tone and mode of the derived cadence.

For example, if we want to check the well-formedness of the sequence of chords in figure 2, we obtain the following (figure 3). This output indicates that, for the fragment of tonal functional harmony encoded above, the only syntactic category of type "c" that can be derived from the given sequence of chords is that of **C** major.

5 Conclusions and Future Work

In this paper we presented an encoding of the harmonic functions of chords as syntactic categories, and showed how the generation of proofs of "harmonic well-foundedness" of cadences could be used as a tool to verify and to display the harmonic functional structuring of cadences. We have also presented an implementation of a theorem prover for automating this verification.

Clearly, there is still much to be done on turning Categorial Grammar applied to functional harmonic analysis a more friendly tool for musicians. Nonetheless, our initial experiments suggest that this can be a useful tool, not only for analysis but also for generation of cadences upon certain constraints, e.g. when building accompaniments for given melodies.

Immediate future work shall include the study of applicability of this tool in practical situations of interest for musicians and for students of music, and the extension of our "dictionary" to encompass richer harmonic cadences. It shall also be interesting to further analyse the mathematical properties of tonal harmony under the viewpoint of Lambek Calculus, and to study what the (musical) consequences could be of altering some of these mathematical properties (e.g. by adding some structural rules or different connectives to the Calculus).

The program presented here is also available by ftp at
`ftp.ime.usp.br:/pub/music/lambek`, or directly from the authors.

Acknowledgements: this work has been partially supported by FAPESP grant 93/0603-01, and CNPq grant 300041/93-4.

References

[BCe84] M. Baroni and L. Callegari (eds.). *Musical Grammars and Computer Analysis.* Univ. degli Studi di Bologna e Modena, 1984.

[Ben87] J. Benthem. A Linguistic Turn: New Directions in Logic. In *Logic, Methodology and Philosophy of Science VII*, pages 205–240. Elsevier, 1987.

[Ben90] J. Benthem. Categorial Grammar and Type Theory. *Journal of Philosophical Logic*, 19:115–168, 1990.

[Ben91] J. Benthem. Language in Action. *Journal of Philosophical Logic*, 20:225–263, 1991.

[Bri79] C. M. Brisolla. *Princípios de Harmonia Funcional.* Novas Metas, 1979.

[Hol80] S. R. Holtzman. A Generative Grammar Definition Language for Music. *Interface*, 9:1–48, 1980.

[Lam58] J. Lambek. The Mathematics of Sentence Structure. *American Mathematics Monthly*, 65:154–169, 1958.

[Lam89] J. Lambek. Multicategories Revisited. *Contemporary Mathematics*, 92:217–239, 1989.

[LJ83] F. Lerdahl and R. Jackendoff. *A Generative Theory of Tonal Music.* MIT Press, 1983.

[Sch89] E. F. Schurmann. *A Música como Linguagem.* Brasiliense / CNPq, 1989.

Appendix: A Theorem Prover for Functions of Chords

```
/****************************************************************/
/* harmon(L, A) - the collection of harmonic justifications for  */
/*               sequence of chords L is A                       */
/****************************************************************/

harmon(L, A) :- miditransl(L, Ln), genseq(Ln, F),
    setof([X,Y], cadence(F, X, Y), An), midiback(An, A).

miditransl([H|T], [Hn|Tn]) :- transchord(H, Hn), miditransl(T, Tn).
miditransl([], []).

transchord([H|T], [Hn|Tn]) :- transnote(H, Hn), transchord(T, Tn).
transchord([], []).

transnote(c, 0). transnote(c_sharp, 1). transnote(d_flat, 1).
transnote(d, 2). transnote(d_sharp, 3). transnote(e_flat, 3).
transnote(e, 4).
transnote(f, 5). transnote(f_sharp, 6). transnote(g_flat, 6).
transnote(g, 7). transnote(g_sharp, 8). transnote(a_flat, 8).
transnote(a, 9). transnote(a_sharp, 10).transnote(b_flat, 10).
transnote(b, 11).

midiback([H|T], [Hn|Tn]) :- noteback(H, Hn), midiback(T, Tn).
midiback([], []).

noteback([0, M], [c, M]). noteback([1, M], [c_sharp_d_flat, M]).
noteback([2, M], [d, M]). noteback([3, M], [d_sharp_e_flat, M]).
noteback([4, M], [e, M]).
noteback([5, M], [f, M]). noteback([6, M], [f_sharp_g_flat, M]).
noteback([7, M], [g, M]). noteback([8, M], [g_sharp_a_flat, M]).
noteback([9, M], [a, M]). noteback([10, M], [a_sharp_b_flat, M]).
noteback([11, M], [b, M]).

/****************************************************************/
/* genseq(S, L) - the collection of candidate sequences of       */
/*               harmonic functions for S is L                   */
/****************************************************************/

genseq(S, L) :- genfunct(S, F), remap(F, L).

genfunct([H|T], L) :- genfunct(T, T2), setof(F, function(H, F), S),
    append([S], T2, L).
genfunct([], []).

function([H|T], [Y, Fun, Mod]) :- funct(Lf, [Z, Fun, Mod]),
    match(H, [H|T], Lf), Y is ((Z + H) mod 12).
```

```
/***************************************************************/
/* funct(H0, F0) - dictionary of harmonic functions            */
/***************************************************************/

funct([0,4,7], [0, [a], maj]). funct([0,4,7], [5, [b,e,c,d,a], maj]).
funct([0,4,7], [7, [a,e,b], maj]).
funct([0,3,7,10], [10, [a,e,b], maj]).
funct([0,4,7,10], [5, [b,e,c,d,a], maj]).
funct([0,4,7,11], [0, [a], maj]). funct([0,4,7,11], [7, [a,e,b], maj]).
funct([0,4,7,11,2], [0, [a], maj]).
funct([0,3,8], [1, [b,e,c,d,a], maj]).
funct([0,3,8], [3, [a,e,b], maj]). funct([0,3,8], [8, [a], maj]).
funct([0,4,7], [5, [b,e,c,d,a], min]).
funct([0,4,7,10], [5, [b,e,c,d,a], min]).
funct([0,3,7], [0, [a], min]). funct([0,3,7], [7, [a,e,b], min]).
funct([0,3,8], [1, [b,e,c,d,a], min]).
funct([0,4,9], [4, [a,e,b], min]). funct([0,4,9], [9, [a], min]).

/***************************************************************/

match(X, [H1|T1], [H2|T2]) :- H1 is ((X + H2) mod 12), match(X, T1, T2).
match(_, [], []).

remap([H|T], L) :- remap(T, T2), combine(H, T2, L).
remap([T], L) :- combine(T, [[]], L).

combine([H|T], L1, L2) :- combine(T, L1, T2), comb(H, L1, H2),
    append(H2, T2, L2).
combine([], _, []).

comb(A, [H1|T1], [H2|T2]) :- comb(A, T1, T2), append([A], H1, H2).
comb(_, [], []).

append([H|T], L1, [H|T2]) :- append(T, L1, T2).
append([], L, L).
```

```
/****************************************************************/
/* cadence(L, Ton, Mod) - L forms a cadence of Ton - Mod        */
/****************************************************************/

cadence([H|_], X, Y) :- theor(H, [X, [c], Y]).
cadence([_|T], X, Y) :- cadence(T, X, Y).

theor([H|T], [X, F, Y]) :- theor(T, [X, L, Y]),
    prove(H, [X, L, Y], [X, F, Y]).
theor([[X, F, Y]], [X, F, Y]).

prove([X, L1, Y], [X, [F|T2], Y], [X, L2, Y]) :-
    invert(L1, [F, d|T1]), invert(T1, T1i), append(T1i, T2, L2).
prove([X, L1, Y], [X, [F, e|T2], Y], [X, L2, Y]) :-
    invert(L1, [F|T1]), invert(T1, T1i), append(T1i, T2, L2).

invert([H|T], L) :- invert(T, Ti), append(Ti, [H], L).
invert([], []).

/****************************************************************/
```

Figure 4: A Theorem Prover for Functions of Chords