Improvising Jazz Chord Sequences by Means of Formal Grammars

Marc Chemillier
marc@info.unicaen.fr
Draft (February 20th, 2001)

1. Introduction

This paper describes a software implementing a chord generator for DJ mix system, based on some principles of jazz harmony. The system produces midi output by means of 1) a grammar-based chord sequence generator, and 2) a dictionary of precomposed measures, mapping chord symbols to midi/music samples. The harmonic aspect of a jazz accompaniment relies on a tonal chord sequence of *n* bars in length which is repeated as a loop with variations, as many times as necessary. The pianist improvises a statement of the chord sequence, varying the voicings of the chords, and introducing new chords by applying substitutions to the original sequence. The main subject of this paper is the implementation of this voicing and substitution technique.

2. Steedman's Grammar

In 1984, Steedman proposed a grammar in order to describe the harmonic substitutions played by jazz musicians. His grammar was expressed in terms of six rewriting rules taking as their input the basic blues progression, and generating an elaboration of it such as the variant played by modern jazz musicians [Steedman 1984]. He deduced the rules from the analysis of a corpus of eight blues progressions taken from a book by [Coker 1964]. Following his article, other authors have brought new developments to his original idea [Johnson-Laird 1991], [Pachet 1999], [Chemillier 2001].

In the example below, the first chord progression is the basic twelve-bar blues in *C*. The second one is a variant generated by applying two rewriting rules to the former one.

C	C	C	C7
F	F	C	C
G	G7	C	C

C	C	C	Gm7 C7
F	F	C	C
Dm7	G7	C	C

The first rule divides the fourth bar into two parts of equal length, and can be stated as:

$$C7 \rightarrow Gm7\ C7$$

The second one, which is applied to the ninth and tenth bars, can be stated as:

$$G\ G7 \rightarrow Dm7\ G7$$

The grammar consists in six rules of this type, each one stated with a variable *x* which can take any degre *C, C#, D, ...* as value. In other words, each rule in Steedman's grammar corresponds to twelve different rules. Every chord is both a terminal and a non-terminal symbol of the grammar. An additional rule is introduced in which the axiom of the grammar on the left side is replaced on the right side by a chord sequence which is the basic blues progression.

3. Automatic Parsing Using Lex

More recently, Steedman published another paper devoted to the same subject, focusing on some difficulties encountered when one tries to implement a parser based on his grammar, due to the fact that the grammar is context-sensitive [Steedman 1996]. In fact, as one can see above, some rules have more than one non-terminal symbol on the left side (this is the case for the rule *G G7* → *Dm7 G7* in the preceding example), and therefore are not of the regular type, nor the context-free type, in Chomsky's classical hierarchy. As a consequence, it seems not possible to implement this grammar as a parser using tools such as Lex or Yacc.

A solution to the problem consists in formalizing aspects of the grammar which are not explicit in the rules described above, concerning the duration of the chords. In the second rule of the preceding example, the chords on both sides of the rule have same duration. But this is not the case in the first rule, in which each chord on the right side has half the duration of the chord on the left side. The action of such rules must be restricted, since the duration of a chord in a jazz progression is generally not lesser than a quarter of a bar. In order to take into account this restriction, we have introduced a modification in the form of Steedman's rules, by adding a new symbol to the grammar representing the barline (denoted /). The new rules adapted from the original ones are then able to prevent the duration of the chords from being too short.

It can easily be shown that this new form of the grammar is equivalent to a finite automaton. The duration of a chord being restricted to values greater than a quarter of a bar, and the number of bars being fixed, it is obvious that the number of possible chord sequences generated by the grammar is finite. Then it can be generated by a finite automaton, and we have designed an algorithm to compute the corresponding automaton.

Using Lex, we have implemented this automaton as a parser, which takes a chord sequence as input, and print Recognized if the sequence can be generated by the grammar, and Not recognized otherwise. To do this, Lex requires a description of the automaton by means of regular expressions, and produces a program written in C implementing the parser. The resulting parser takes as input a sequence of chords separated by barlines (/), as in the following example, where the input sequence given to the parser is the well-known *Blues for Alice* composed by Charlie Parker:

```
$ blues
> F / Em7 A7 / Dm7 G7 / Cm7 F7 / Bb7 / Bbm7 Eb7 / Am7 / Abm7 Db7 / Gm7
/ C7 / F7 / Gm7 C7
Recognized
```

4. Chord Formal Languages and Sampled Voicings

Our software generating jazz chord sequences is a combination of two modules. The first one randomly applies substitution rules to the chord progression, thus generating a formal language of chord sequences of the regular type. The second module chooses voicings for the successive chords of the resulting chord sequence, by selecting sampled voicings in a database. The whole program has been written in Lisp using OpenMusic, which is a visual language for music composition designed at Ircam. Examples of midifiles generated by the program are available on the Web (www.info.unicaen.fr/~marc/publi/grammaires).

The voicing module maps the chords to midi data in order to produce a midi sequence which is played by a synthetizer. It relies on a table of association between chords and precomposed midi data called midi samples. This table is not restricted to one chord entries, but associates midi samples to sequences of *n* consecutive chords. This means that the module reads the input sequence through a shifting window of length *n*. Thus the formal model for this module is a finite state transduction.

The substitution module operates by randomly choosing a position in the chord progression where a rule may be applied. The basic procedure takes as argument a grid and an integer *k*, and returns a new grid resulting in *k* applications of the substitution process to the input grid. Here are two examples, with respectively one and ten substitutions applied to the basic blues progression stored in the variable `*basic-grid*`. In these examples, chords generated during the substitution process are marked with a star.

```
? (rewriting *basic-grid* 1)
((c) / (c) / ((g *) 7) / (c 7) / (f) / (f) / (c) / (c) / (g) / (g 7) /
(c) / (c))

? (rewriting *basic-grid* 10)
((c) / (c) / (c) / ((c# *) 7) (c 7) / (f) / ((f# *) 7) / ((f *) 7) /
((bb *) 7) / ((eb *)) ((eb *) 7) / ((d *) 7) ((c# *)) / (c) / (c))
```

In the present state of the software, the user may interact in real time with it, thus truly improvising. This interaction can be done in three different ways.

First, one can modify the depth of the substitution process, which is the number k of rules applications introduced in the examples above. When a grid is being played, the user has the possibility to modify the value of k for the computation of the next grid. Thus, the harmonic sequence can evolve between a basic state of the grid with few substitutions and a more complicated state with many substitutions.

The second possibility to interact with the system is to orientate manualy the choice of the chords. This possibility is closely related to the property we have mentioned in the previous section concerning the regular type of the chord sequence language generated by Steedman's grammar (restricted to durations which are not lesser than a beat). The chord X_{i+1} which can follow a given chord X_i may result in the activation of any substitution rule applied to the current chord sequence, but furthermore, it can result in the activation of k successive rules applied in a cumulative way. Although k might take any integer value, it is possible to compute all the X_{i+1} that can follow X_i, thanks to the fact that Steedman's grammar is equivalent to a finite state automaton. As usual, this automaton may be represented as a graph with a finite number of states, and arrows representing transitions between states. When the program is running, it is possible to display the exact state of the system, and then to give the user the possibility to choose which transition will be activated from the current state.

A third way to interact with the system is to choose new midi samples for the voicing module. The database containing all the available midi samples is organized into different tables corresponding to different moods, effects, and so on. Currently, the voicings stored in the database are arrangements in the style of popular electronic music such as house, ambient, etc (including bass, percussion and various effects). A very simple table of piano voicings in the boogie-woogie style is available on the Web at the address indicated above.

4. DJing with a Chord Generator

This way of interacting with the system by choosing midi samples corresponding to different types of voicings is close to the DJ technique of selecting samples from records and mixing them together. The DJing task consists in selecting extracts of precomposed music (usually stored on vinyl records), and combining them to produce a new composition (by means of two turntables connected to a mixer). There exist many softwares simulating the DJing task, by mixing in real time different type of audio files (wav, aiff, mp3). Our system shares several aspects in common with this type of software. The main difference lies in the fact that midi samples are not chosen manualy, but with an automatic procedure implementing the substitution grammar described above. Thus our chord generator can be viewed as an extension of traditionnal DJ software systems.

As a further develoment of the system, we plan to give the user the possibility to update the voicing table in real time by playing new voicings on a midi keyboard connected to the computer. An extension of this updating process will consists in applying the statistical learning scheme developped by [Assayag 1999]. This scheme embodies an incremental parsing algorithm based on the Lempel-Ziv compression technique. While the system is running, the user plays voicings corresponding to the chords generated by the system. Then stylistic characteristics (textures, transitions, etc.) of these voicing are learned by the system and added to the voicing database for further selection in the improvisation process.

5. References

Assayag G., Dubnov S., Delerue O., Guessing the Composer's Mind: Applying Universal Prediction to Musical Style, *Proc. of the ICMC 99*, Beijing, China (I.C.M.A., San-Francisco, 1999).

Chemillier M., Grammaires, automates et musique, F. Pachet (éd.), *Informatique musicale* (Hermès, Paris, to appear 2001).

Coker J., *Improvising Jazz*, 1964, reprint Fireside, 1987.

Johnson-Laird P., Jazz Improvisation: A Theory at the Computational Level, in: Cross I., Howell P., West R. (eds.), *Representing Musical Structure* (Academic Press, 1991) 291-326.

Pachet F., Surprising harmonies, *JIM 99, 6èmes journées d'informatique musicale* (CNET-CEMAMu, Issy-les-Moulineaux, 1999) 187-206.

Pachet F., Computer Analysis of Jazz Chord Sequences. Is Solar a Blues ?, in: Miranda E. (ed.), *Readings in Music and Artificial Intelligence* (Harwood Academic Publishers, 2000).

Steedman M.J., A Generative Grammar for Jazz Chord Sequences, *Music Perception* **2** (1) (1984) 52-77.

Steedman M.J., The Blues and the Abstract Truth: Music and Mentals Models, in: A. Garnham, J. Oakhill (eds.), *Mentals Models in Cognitive Science* (Erlbaum, Mahwah, NJ, 1996).

Ulrich J.W., The analysis and synthesis of jazz by computer, *Fifth International Joint Conference o Artificial Intelligence* (1977) 865-872.