A meta-level architecture applied to the analysis of Jazz chord sequences

François Pachet
LAFORIA, Institut Blaise Pascal,
Université Paris VI, 4, place Jussieu, 75252Paris Cedex 05

We describe here a knowledge-based system that analyses Jazz Chord Sequences, i.e. that find underlying tonalities for each chord of a given sequence. Rather than relying on formal properties of Tonal Music to perform the analysis, which yields good results only for a limited subset of well-formed chord sequences, we decided to mimic the human behavior, by representing the actual knowledge associated with this activity.
We show that the expertise involved in analysing chord sequences consists both in domain knowledge (related to the theory of Standard Western Harmony), and meta-level knowledge, related to the control of reasoning. The system is based on a forward chaining, first order, Object-oriented environment, and performs the analysis by first looking for well-known patterns of chords (such as two-fives) for which the analysis is straightforward, and then by filling the gaps between analysed patterns, using common sense knowledge, such as "prefer the analysis that minimize the number of underlying tonalities". The system is thus able to handle chords that would, in theory, be considered as incorrect (such as a Major seventh chord in first position in a blues).

Introduction. The problem

The problem of jazz chord sequence analysis consists in finding underlying **tonalities** for each of the **chord** of a given **sequence**. Chords are specified by they **root** (out of A, A#, B .. G#), and their **structure** (such as minor, major, major seventh, flat 5, 13). To simplify the presentation, tonalities are defined as **scales** (sequences of notes), and are indicated by their root and the **type** of the scale (out of major, harmonic minor, melodic minor). Analysing a chord consists thus in finding the underlying tonality, and the **degree** of the chord in this tonality. An isolated chord may belong to various tonalities. For instance the chord [C Major] may be analysed as (first degree of the C major tonality) or (fifth degree of the F major tonality). Only the relative positions of the chords in the sequence will allow deciding in which tonality the chord has to be analysed.

A natural vs formal solution.

Two kinds of solutions exist for the problem. Formal solutions, such as grammar-based solutions [Steedman] consist in applying backward-chaining transformation rules to the sequence until a root sequence is found (such as the basic 12-bar blues sequence). Thoses methods rely heavily on the natural formal properties of chords, but are limited because they analyse only well-formed sequences that may be derived from a root sequence, which constrains heavily the number of sequences that may be analysed. On the other hand, knowledge-based solutions consists in representing the knowledge used by musicians in the process of analysing, and mimicking it in a appropriate environment. We explored the second solution, using the NéOpus Knowledge Representation environment [Pachet 1,3], in which knowledge is represented in terms of objects, forward-chaining rules, and meta-rules that control the reasoning.

A natural and general method

A common method for analysing chord sequence is the following :

1- For each chord of the sequence, identify all the possible underlying tonalities, as given by the standard theory.
2- Identify well know small patterns of chords, whose analysis is straightforward, such as two-fives, "anatoles".
3- Once those patterns are found, fill the gaps between patterns, by using preference rules that maximize the extent of the underlying tonalitites (or minimize the number of tonalities).

4- Use rules of thumb, that are not taken into account by the standard theory, such as "a seventh chord may be a first degree, if it is on a strong beat of the sequence", or "if there is an anatole at the end of the sequence, then the underlying tonality of the anatole is certainly the general tonality of the sequence".

In this scheme, several kinds of Knowledge have to be represented. We sketch out here how our system handle them.

An implementation with the NéOpus system

The NéOpus system is an Object-oriented, forward-chaining first order inference engine, inserted in the Smalltalk-80 environment [Pachet 1,3]. The Smalltalk-80 environment is well suited to represent domain objects, by their structure, and behavior. NéOpus adds a forward-chaining rule facility, that allow to represent knowledge in terms of relations between objects. Finally, a declarative control architecture allows to specify the control of knowledge bases by meta-rules.
In this sytem, the various kinds of knowledge are naturally represented : objects, rules and meta-rules.

Objects

Smalltalk classes represent the domain objects suchs as chords, tonalities, chord sequences. Thanks to the inheritance mechanism, common structure and behavior are shared. For instance, a root class called `ElementInASequence` is defined by having a `position` and a `lenght` in the sequence. Class `Chord` is defined as a subclass of `ElementInASequence`, that adds a `root`, a `structure`, and handles the tonalities. The chord sequence itself is represented by class `ChordSequence`, and is defined as having a `listOfChords` (instances of class `Chord`), and a `generalTonality`, as well as methods to handle access to its chords.
Class `Chord` implements methods to compute intervals, to perform transpositions, to compute the lists of pitches from the chord definition. Similarily, the computation of all possible tonalities for a given chord is handled by Smalltalk methods, implementing the corresponding calculus. For instance, getting all the possible tonalities of a chord, say (A sharp, flat fifth, 11th) corresponds to the following Smalltalk message transmission :

```
(Chord !A# 5b 11) possibleTonalities.
```

The result will be a list (an `OrderedCollection`) of instances of class `Tonality`.

rules

NéOpus rules are used to represent the knowledge associated to patterns of chords. The syntax is the following sequence : a rule name, followed by a variable declaration part between vertical bars, in which variables are declared by their class name; a list of premises (which may be any Smalltalk expression) and an action part (also any Smalltalk expression) (see exemples). Rules are used to represent knowledge associated with chord patterns, and chord analysis.

rules for defining patterns

For instance, the notion of "Two-five" is well represented in this manner. A "Two-five" is a sequence of two chords, such as (D min - G 7), so that : the root of the second is the root of the first transposed by a major fourth, and the first one is minor, and the second one major. However, a lot of variations on this basic definition exist. Once again, those variations cannot always be expressed in terms of transformation rules. For instance, the sequence (D 7 9+ - G 7) may be, in some contexts, interpreted as a two-five (if the surrounding chords have all been analysed in the corresponding tonality for instance).

Here are for instance, two rules that define two-fives respectively in a major and a minor tonality :

```
majorTwoFive
   | Chord c1 c2 |
c1 isMinor.
c2 <- c1 followingChord.
(c1 includes: c1 flatFifth) not.
c2 isSeventh.
c2 root = c1 root fifth.
actions
|aTwoFive|
aTwoFive <-TwoFive new root: (c2 root
fourth) index: c1 index; tonality:
#major.
aTwoFive go
```

```
minorTwoFIve
   | Chord c1 c2 |
c1 isMinor.
c2 <- c1 followingChord.
c1 includes: c1 flatFifth.
c2 isSeventh.
c2 root = c1 root fifth.
actions
|aTwoFive|
aTwoFive <-TwoFive new root: (c2 root
fourth) index: c1 index; tonality:
#minor.
aTwoFive go
```

rules for performing the analysis

Rules are also used to represent the knowledge associated with the analysis itself. Those rules will find underlying tonalities for those chords who do not belong to analysed patterns. General rules indicates that two adjacent, non analysed chords that share only one common possible tonality have to be analysed in this common tonality. Other rules handle specific cases, such as the following rule: "when harmonic minor and melodic minor are possible for an isolated non analysed chord, then chose harmonic".
Here are two rules that handle such cases. The first one says that the general tonality of a sequence is determined if there is an anatole at the end. The second one choses between harmonic and melodic tonalities in the case of a major isolated chord.

```
endOfSequenceAnatole
   | Anatole a . Sequence s |
s == a sequence.
s generalTonality doesNotExist.
a isAtEnd.

actions
s generalTonality: a tonality
```

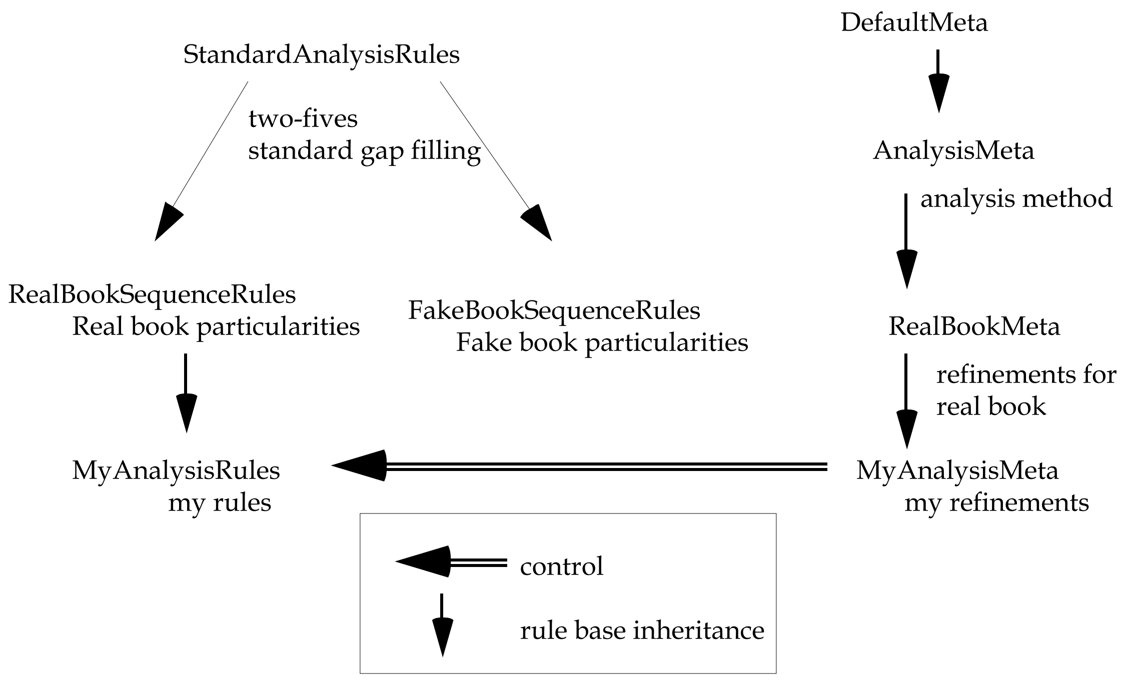
```
choiceMelHarm
   | Chord c |
c notAnalysed.
c isMajor.
c hasTonalityIn: #minorHarmonic.
c hasTonalityIn: #minorMelodic.
actions
c analyse:
    (c tonalityIn:#minorHarmonic).
```

meta-rules

Rules are not to be triggered in any order. The process of analysing consists in various steps that are not reflected in the rules themselves. In order to represent this, we use the control architecture of NéOpus, to specify the control of those rules. Rules are organized in protocols (groups of rules). In order to stick to the natural method presented above, packs of rules have to be executed in a particular sequence.
Our architecture separate the domain rules, from the control rules. Rules are organized in rule bases, which may be activated by other rule bases, called meta-bases. In our case, all the domain rules (pattern identification, gaps filling) are included in rule-base `Analysis`. A meta-base `AnalysisMeta` contains all the rules necessary to represent the chosen analysis method. `AnalysisMeta` controls the activation of `Analysis` by an association handled by the system. Other meta-bases are implemented, that define other analysing procedures, and may be associated the the `Analysis` rule base.

rule base inheritance

Rule bases may interact with each other, using a rule base inheritance mechanism, translated from the class-based language inheritance mechanism, with a different interpretation: a rule base A inheriting from rule base B (defined as a sub-base of rule base B), will be considered more specific than B, and all its super bases. Rules of the most specific rule bases will be prefered at execution time, by the rule interpreter. A rule base tree is thus constructed, in which rule bases implementing more specific knowledge will inherit from more general rule bases. In our example, two different branches are built, one for domain rules, from root base `StandardAnalysisRules`; the other for control strategies, from root base `DefautMeta`. More specific rule bases are defined, that handle the variations necessary th analyse sequence from the Real book, or the Fake book. Then user's rule bases are defined as sub bases of such existing rule bases.



the rule base inheritance tree for chord sequences analysis

Discussion
Our system is to be compared with the backward-chaining apporach described in [Steedman], that defines a set of transformation rules with which an initial chord sequence is analysed and reduced to a root sequence (such as the 12-bar blues). We are mostly interested by those chord sequences that would not be considered syntacticaly correct by such grammars, but for which there are acceptable solutions (the evidence is that musicians do analyse such chord sequences). For instance, tunes like C. Mingus's "Good bye pork pie hat", or even the standard D. Gillespie "Night in Tunisia" may not be analysed correctly with those techniques, because they are only locally syntactically correct, and do not derive from a root sequence. Our system gives plausible analyses for those tunes. However, unlike backard-chaining systems, it is not able to give multiple solutions in cases of ambiguous chords. Moreover, our system is interesting because it gives the opportunity of experiencing various control strategies on the same knowledge base.
This system is an application of a general architecture for representing deductive knowledge in an Object-Oriented environments [Pachet], which has also been applied for representing Knowledge used by accompagnying Jazz Musicians.

References
[Pachet F. 1] NéOpus User's Manual. Rapport LAFORIA, n° 91/14, July 91.
[Pachet F. 2] Towards a expert system that follows human improvisation. Rapport de DEA. Ircam, Paris 1988.
[Pachet F. 3] Reasoning with objects : the NéOpus environment, East EurOOpe, Bratislava, Sept. 91.
[Steedman Mark J.] A Generative Grammar for Jazz Chord Sequences. Music Perception, 1984, Vol. 2, N° 1, pp 52-77.