Using Logic Programming Languages For Optical Music Recognition[1]

Bertrand Coüasnon
IRISA / INSA-Département Informatique
20, Avenue des buttes de Coësmes
F-35043 Rennes Cedex, France
couasnon@irisa.fr

Pascal Brisset, Igor Stéphan
IRISA
Campus Universitaire de Beaulieu
F-35042 Rennes Cedex, France
brisset,stephan@irisa.fr

Abstract

Optical Music Recognition is a particular form of document analysis in which there is much knowledge about document structure. Indeed there exists an important set of rules for musical notation, but current systems do not fully use them. We propose a new solution using a grammar to guide the segmentation of the graphical objects and their recognition. The grammar is essentially a description of the relations (relative position and size, adjacency, etc) between the graphical objects.

Inspired by *Definite Clause Grammar* techniques, the grammar can be directly implemented in λProlog, a higher-order dialect of Prolog. Moreover, the translation from the grammar into λProlog code can be done automatically.

Our approach is justified by the first encouraging results obtained with a prototype for music score recognition.

Keywords: Document analysis, Optical Music Recognition, DCG, Grammar Translation

1 Introduction

In structured document analysis, one open problem is to separate knowledge from the matching mechanism. Another difficulty is to avoid the paradox that if you want to recognize correctly, you have first to decompose[2] correctly, but in order to decompose correctly you should have already recognized.

In Optical Music Recognition decomposition problems start with staff lines which connect all the musical objects. Moreover, complex scores usually have an important number of notes and are polyphonic (different voices on a single staff). This introduces a high density of information, which creates real problems of decomposition, for example when graphical objects touch when they should not. The existing systems for recognition cannot overcome those kind of difficulties [2].

Indeed the syntactic advantages of the musical notation, i.e. the strong syntax and knowledge about structure of written music, are generally not fully used by

[1]In Proc. of the Third International Conference on the Practical Application of Prolog, Paris, France, April 1995.

[2]To avoid confusion between *segment* the geometrical term, and *segmentation* the image processing term (to partition an image), in the following we use *decomposition* for *segmentation*.

the current systems. Our aim is to develop a recognition method which takes into account these strong constraints on music writing.

We present a method to use the musical knowledge context to control all the recognition process and particularly the decomposition and labeling[3] of objects. As Fujinaga [11] states, the music notation can be formalized using a grammar. We propose to use such a grammar to control the recognition process. The context introduced by the grammar enables us to test a decomposition suited to the object the current rule tries to match.

In recognition systems, primitives are usually labeled during extraction, using only very local information, and the labels never change during the recognition. This creates a risk of generating errors without any means of correcting them. Our method enables us to definitively label a primitive only when a rule can be applied and therefore introduces high level context in labeling.

This method can be applied to different domains with also a strong syntax such as mathematical expressions, technical drawings, etc. The grammar on musical notation we propose is general enough to deal with full scores and has been validated by a musician.

Since its very beginning logic programming is involved with syntactic analysis [7]. It is well known that context free grammar rules can be straightforwardly translated into the formalism of Horn clauses programs [5]. Because of the complex space shape of information, a high-level language of description is required. Our grammar is implemented in λProlog [18] with semantics attributes connected to C libraries for pattern recognition and decomposition. Moreover the translation from grammar to λProlog can be automatically done by a *Definite Clause Grammar* parser [6, 5, 19]. We propose to use a two-dimensional extension of DCG which we choose to implement in λProlog. This higher-order extension of Prolog has been shown in [13] to be well-suited to write DCG parsers.

The system is already able to recognize full scores with up to two voices per staff, and to correct some decomposition errors. It can also detect (and sometimes correct) recognition errors (on note duration) by checking the number of beats in a bar and the vertical alignment of notes in full scores.

This paper is organized as follows. We first present some related work on optical music recognition. The musical context and the grammar are then described. The implementation of the grammar with λProlog is presented. Afterwards, the translation from the grammar to λProlog code by the λProlog DCG parser is shown. The prototype architecture and some encouraging results are described.

2 Recognizing Music Scores

2.1 Related Work

Recognizing music scores requires the detection of the staff lines (otherwise objects which are disconnected logically would seem to be one single object). First to make an initial decomposition of objects after erasing the staff lines (a technique used by most authors), and second to calculate the pitches of notes in later phases of the processing.

[3]To give a name to a graphical object

This detection is done by different treatments always considering the staff line as a real line: horizontal projections [11, 14]; searching of the longest chord [16, 22]. Only Carter [3] proposes a line adjacency graph based method which deals with non-perfectly rectilinear staff lines (a relatively frequent case).

The objects, decomposed by erasing the staff lines, are classified by different techniques (some authors use more than one): first classification on the bounding box size [21, 4]; horizontal and vertical projections [11]; extraction of primitives such as notehead, stem, etc. by using erosion or thinning or others techniques. Generally the label of these primitives is not changed during the recognition process except in [14].

The principal open problems are [2, 3]:

- The scaling up of a prototype. Indeed, most of the developed systems are prototypes adapted to simple scores. The scaling up to complex score systems is difficult (the same techniques cannot be used). For instance, the problem of scaling up from a monophonic system to a polyphonic one.
- The reconstruction of objects broken because of noise.
- The objects touching when they should not.

The lack of use of knowledge is one of the reasons why these problems are still open. Although all the existing optical music recognition systems use some musical rules. But those rules are usually selectively incorporated in the recognition process, and are not really formalized. Some solutions use some rules formalized by a grammar but mostly for limited purposes (verification, error correction or final pitch calculation). They do not embed graphical rules and are limited to simple scores. Andronico [1] is the only one to propose a formalization, including the graphical level. He uses a "natural" grammar and a set of "bidimensional" grammars. However, these grammars were only adapted to simple and monophonic scores. As previously mentioned, Fujinaga [10] states that music notation grammar is context-free and LL(k). It enables musicians (corresponding to a top-down parser) to read this notation in an efficient way. Hence, it is possible to define a grammar to formalize the music writing rules which can be found in complex scores. It is possible but difficult as all the proposed grammars are not able to deal with full scores.

2.2 Aims

We want to design a system able to:

- Recognize full scores (orchestra scores) where each instrument has its own staff, and the staves are vertically connected.
- Deal with complex scores. These are usually polyphonic (different voices on a single staff), and so the recognized notes have to be correctly distributed in each voice.
- Have as few recognition errors as possible and good enough results to avoid human verification. This is realistic because musical writing has enough rules and redundancy. For example number of beats in a bar and vertical alignment

of notes in full scores. We also allow the system to indicate the places it could not produce satisfactory results (an operator can then correct these areas).

- Be extended to handwritten scores (if they are not too badly written).
- Extend the vocabulary relatively easily.
- Recognize only the "classical" music notation. Indeed the notation used by some composers in the 20th century can be really different from one work to another. Usually there is a glossary at the beginning of the score.

2.3 Difficulties

Touching Objects We want to recognize complex and full scores. It means dealing with high densities of symbols. High density causes connections between music objects that syntactically do not touch each other. It creates some decomposition problems (Fig. 1).

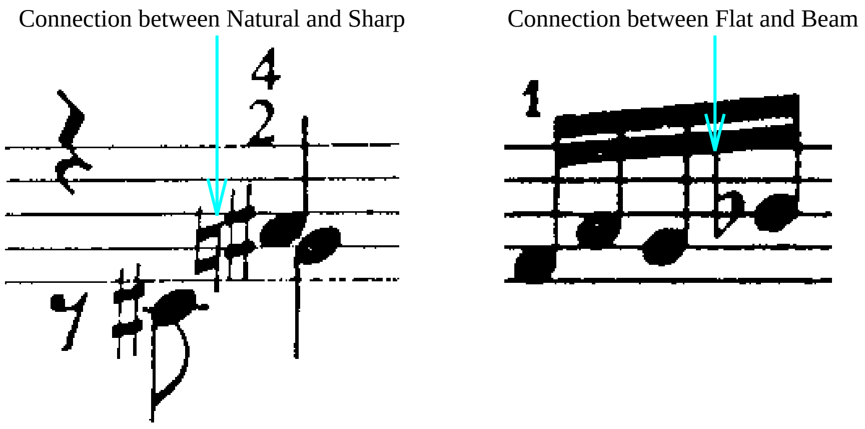
Connection between Natural and Sharp
Connection between Flat and Beam

Figure 1: Examples of objects that should not touch.

Broken Objects Scanning produces some noises and the possible bad quality of initial documents gives broken objects. Usually, the affected objects are small (*e.g.* tuplets's figures). Removing staff lines is necessary for the recognition process but could also produce some broken objects. It is the most frequent case.

These decomposition problems are unavoidable. As the classical decomposition techniques at the image level cannot solve the problems, we must use the most of the music context.

2.4 Primitives: Segments and Symbols

2.4.1 Information found on a Score

In order to define the grammar, we now study the structure of music information. We can, as Martin [16] did, divide the objects on a score into two classes:

- The constructed: called "iconics" by Martin (*e.g.* simple stemed notes, beamed notes) composed of segments (as stems, beams) and noteheads plus a set of assembly rules on those segments and noteheads.

- The symbolics: for instance clefs, accidentals, etc. can be considered as characters. Optical character recognition (OCR) techniques can then be used to recognize them. A notehead becomes a symbol when its associated stem is recognized.

As the constructed objects have rules to describe their shape, the grammar has to integrate those rules. Hence, the grammar terminals have to match first the segments (*e.g.* stems, beams) which are elements of the constructed objects, and then the symbols (*e.g.* clefs, accidentals)

If we are interested in recognition and not interpretation, we can also observe on a score an independence between the way the notes are beamed and the musical semantics. Indeed, we can consider beamed notes syntactically equivalent to a sequence of simple eighth notes. Although the beaming information is important for easier recognition, it complicates the treatment of simultaneous notes on a full score. Consequently, we have to distinguish between two levels of information on a score:

- Physical level: which corresponds to the way notes and their attributes (accidentals, accents...) are formed and adjusted on the score.

- Logical level: which corresponds to the syntactic way of using notes in writing music. Therefore, these notes are independent of the beaming, and have in attribute all the note information: pitch, value, accidental...

Indeed, at the logical level, it is not necessary to know, for example, where a sharp is graphically positioned relative to a notehead, but simply that the note is sharped. That concentration of the musical information on the note makes it possible for a musician to check the simultaneity of notes among all the staves of a full score. This separation in two levels of the musical information requires the definition of a grammar to model it.

We note that, because a full score is a list of vertically synchronized staves, the grammar for a full score is almost directly derived from the grammar for a single staff.

2.4.2 Extraction of Primitives

We have seen that terminals of the grammar are segments and symbols. Therefore it is necessary to extract from the image the primitives needed to represent those terminals.

We have used a technique for extracting segments [20] which is really robust considering the problems of quality of the processed images (scale, curvature, bias, different noises). Moreover, its vectorization easily permits staff line identification, even if they are not rigorously rectilinear.

The usual representation of symbols in OCR is the pixel array. Therefore, we can use the connected components calculated after the staff lines are removed. These connected components can be clustered or decomposed during the analysis to obtain the right pixel array representing a symbol.

2.5 Presentation of the grammar

We have shown that the difficulties due to the goals require the use of the context in the recognition process. The musical context can be introduced by a formalization of the musical knowledge in a grammar. The grammar, because of the kind of objects on a score, parses segments and connected components.

The two levels of information on a score imply a particular structure to the grammar. This structure is composed of two levels of parsing: a graphical one corresponding to the physical level, and a syntactic one corresponding to the logical level. As the grammar describes an image, it is necessary to introduce some particular operators to define the relative position of the elements.

For the presentation of the grammar, we will take some simple rules and describe graphically the grammar structure (Fig. 3)(Fig. 4). We can't present here all the complete grammar as it takes 5 pages of description. We use the Backus notation (BNF) where, for legibility reasons, the grammatical variables are not bracketed by $< >$ and the terminals are written in italic, position operators in bold. It is also necessary to define a factorization notation. These two notations have the following meaning:

- Position Connector:

  $$A\ \mathcal{P}\ B$$

  means A and, at the position $\mathcal{P}$ in relation to A, we find B.

- Factorization notation:

  $$A(B\!:\!C)$$

  means $A\ B$ and $A\ C$

  The notation is only used in association with the position operators, so we have:

  $$A(\mathcal{P}_1\ B\!:\!\mathcal{P}_2\ C)$$

  means $A\mathcal{P}_1B$ and $A\mathcal{P}_2C$

To extend a grammar rule describing a single staff to a full score, i.e. a list of staves, we define a "map_staff" operator which applies the rule to each staff of the system. We use the following notation:

$$\mathit{map\_staff}(R)$$

to define the extension of a single staff rule R to a system of staves rule.

2.5.1 Graphical Part

In order to illustrate the graphical level (Fig. 3) of the grammar, we present (Fig. 2) a set of simplified rules which defines a beamed eighth note. There is only one beam that links the notes. The chords, dots, accents and slurs are not defined in this example:

The grammar variables represent:

ConsRest eighth rest, sixteenth rest...(Constructed rests)

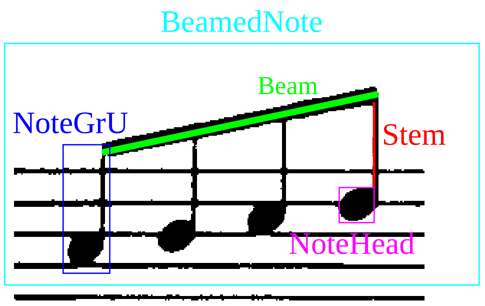

Figure 2: Graphical representation of the rule BeamedNote

NoteGr single graphical note (half, quarter, eighth…)

NoteGrU graphical note with an up oriented stem

NoteGrD graphical note with a down oriented stem

NoteHead notehead with all its attributes

Head can be a black head or a white head

Accidental can be natural, flat, sharp, double-flat, double-sharp

The position operators have the following meaning:

- `left_tip` : at the left tip
- `closev` : vertically close
- `right_tip` : at the right tip
- `closel_downtip` : close on the left of the down tip
- `closel_same_line` : close on the left at the same pitch

The terminals *Beam* and *Stem* cause a labeling of the corresponding segments.

```
BeamedNote ::= Beam (left_tip NoteGr :
                     closev [NoteGr | ConsRest]* :
                     right_tip NoteGr)
NoteGr     ::= NoteGrU | NoteGrD
NoteGrU    ::= Stem closel_downtip NoteHead
NoteHead   ::= Head closel_same_line [Accidental]?
```

Consequently, we can interpret the rule NoteGrU as follows:

There is a NoteGrU if there is a stem, and close at the left tip of the stem, there is a NoteHead

The NoteGrD (graphical note with a down oriented stem) is interpreted similarly.

The factorization notation (:) is illustrated by the rule BeamedNote. This rule means that a beamed note is made of a beam and at each tip of this beam there is a graphical note and in between there is some graphical notes. Each position operator `left_tip`, `closev` and `right_tip` refers to the terminal *Beam*. This is useful when different objects are positioned in relation to a single other one.

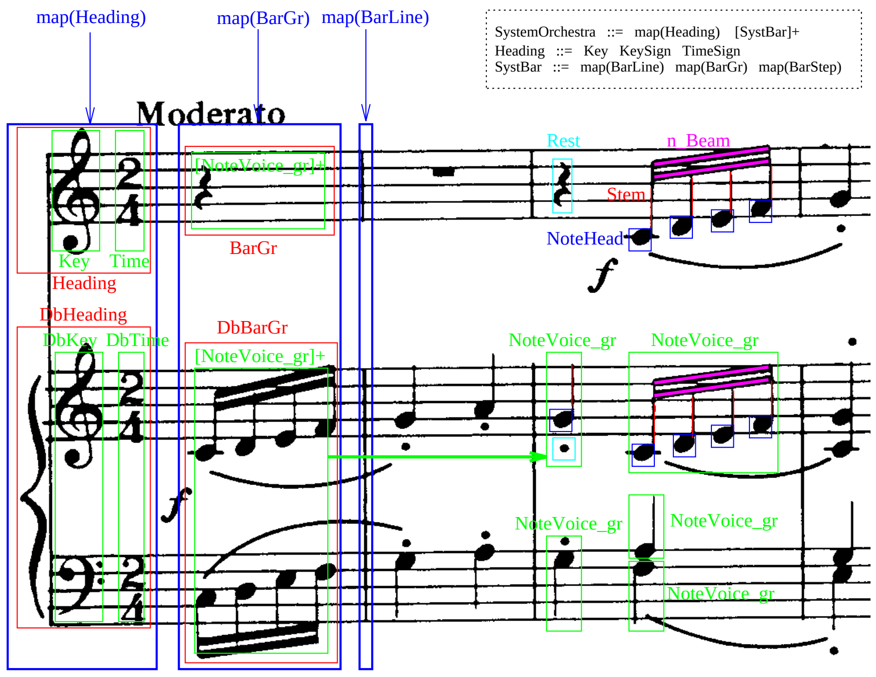

Figure 3: Beginning of the grammar and map_staff(BarGr) (graphical level)

Polyphony is introduced at the graphical level (Fig. 3) with a notion of notes starting at the same time (NoteSimulGr). Actually, we consider a bar at the graphical level (BarGr) as a succession of notes starting at the same time (NoteSimulGr). For each of these moments there is only one graphical note by voice.

2.5.2 Syntactic Part

Fig. 4 shows the syntactic level of the same piece of music as Fig. 3.

The notion of Step creates a vertical cutting of the whole score. A Step corresponds to the smallest duration in a column of vertically aligned notes (in a system). This notion of step solves some of the problems due to polyphony and full scores. Indeed, it can manage the simultaneity (i.e. the verticality) of notes on a same staff (case of polyphony), and on the different staves of one system (full scores).

Information such as dynamic markings, phrasing slurs, etc. which are associated to the staff and not to a note, are introduced in the grammar at the step level.

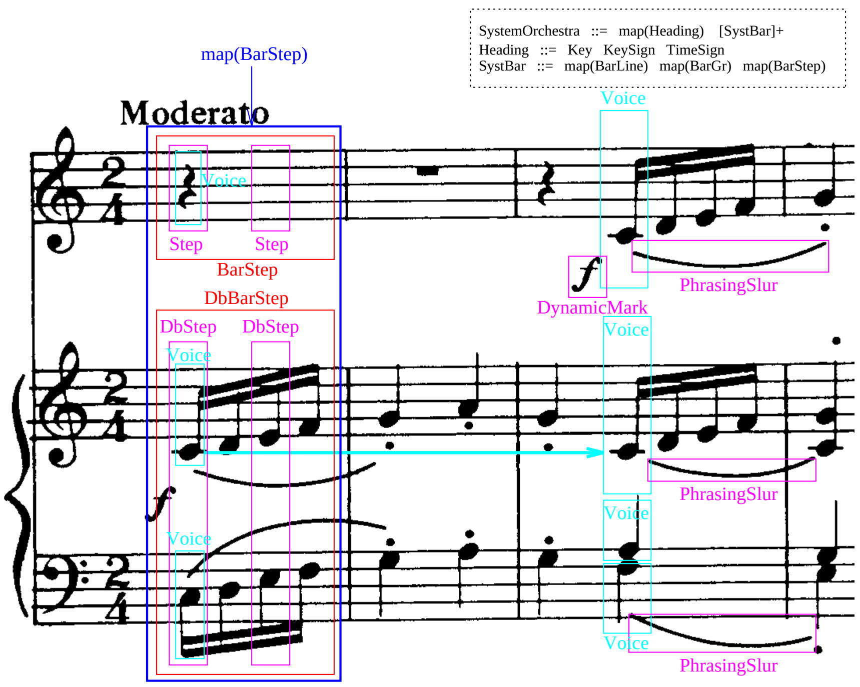
SystemOrchestra ::= map(Heading) [SystBar]+
Heading ::= Key KeySign TimeSign
SystBar ::= map(BarLine) map(BarGr) map(BarStep)
map(BarStep)
Moderato
Voice
Step
Step
BarStep
DbBarStep
DbStep
DbStep
DynamicMark
PhrasingSlur
PhrasingSlur
PhrasingSlur

Figure 4: map_staff(BarStep) (syntactic level)

2.5.3 Staves System

We can illustrate the use of the map_staff operator by seeing the rule BarSystem:

```
BarSystem ::= map_staff(Heading)
              map_staff(BarGr)
              map_staff(BarStep)
```

For example, if the system has four staves, the operator map_staff will apply four times the rule Heading, then the second map_staff will apply four times the rule BarGr to detect all the notes and four times the rule BarStep to make the syntactical checking. All the attributes are automatically given to the right staff.

2.6 Presentation of the Parser

The input parsed structure is built at the image processing level. It is organized as follows:

- Score: List of SystemStaff
- SystemStaff: List of SystemBar (of possible SystemBar)
- SystemBar: List of Bar (of possible Bar)
- Bar:

  - Bar Line (possible)
  - List of connected components above the Staff
  - List of connected components on the Staff
  - List of connected components below the Staff
  - List of horizontal segments
  - List of vertical segments

As the image (and the grammar) has two dimensions, we introduce in the parsing process a cursor with the following components:

- the current position in the image;
- the element to parse (a segment or a connected component);
- a research zone, modified by the position operators.

In fact, this cursor corresponds to the head of the parsing list in classical parsers. The next element to parse is not necessary the next one in the structure. It depends on the current position and the research zone. If the research zone is defined then the next element (if it exists) has to be in this area, if it is not defined then it is simply the next element (a segment or a connected component) considering the X axis. There is a set of tools called in a grammar rule to access the structure.

The parsed structure is composed of segments (non-labeled), and of connected components which do not necessarily represent a symbol (unlike classical parsers, the input structure is not composed of the same element type). So the parser has first to make a labeling (for segments, and connected components representing a symbol) and second to detect errors of decomposition (when a connected component does not represent a symbol) and to change the parsed structure to correct these errors.

For instance, a segment with the features of a stem could also be an element of a sharp. The parser is able to label definitively the segment (as a Stem) only with the context embedded in the grammar.

In the same way, a connected component is not always decomposed correctly to represent a symbol. It is once again with the context that the component is decomposed in order to find a symbol in it.

The parser is also able to cluster some non-labeled connected components, with the help of the context to rebuild a symbol.

3 Implementation of the Grammar with λProlog

We present in this section how the grammar is implemented with λProlog, a dialect of Prolog. Hence, the obtained λProlog program is a parser. The implementation of the grammar is directly derived from works concerning DCG (*Definite Clause Grammars*) [6, 5, 19].

3.1 λProlog

λProlog [17] is a higher-order extension of Prolog where first-order terms are replaced by typed λterms. The first comment about this extension is to say that it is *not* a functional one; the λterms will not be used to *evaluate* but only to *encode*.

Prolog is a good candidate to handle formal statements but is not quite appropriate to encode terms containing variables. For example, there is not a *good* representation of a function with a Prolog term which handles classic operations (application, composition, ...). λProlog can favorably replace Prolog for this kind of task.

Most of the novelty of λProlog relies on the λterms. They are handled via higher-order unification which works modulo $\alpha\beta\eta$equivalence. Compared to Prolog, the language of goals is also extended: universal and existential quantifications and implication are allowed in the body of a clause.

We give here an example of a λProlog program in order to show both the syntax and the features of the language. Suppose you want to write a program to derive functions composed of arithmetic and trigonometric operators (say `+`, `*` and `sin`).

First, to handle its higher-order features, λProlog uses a curried notation (like LISP). The term $3 + sin(2 * 3.14)$ may be written

```
3 + (sin (2 * 3.14))
```

As announced, a function will be coded with a λterm. So the function $x \rightarrow 3 + sin(2 * x)$ may be written

```
x\ (3 + (sin (2 * x)))
```

As we can see, the λabstraction is denoted with an infix backslash (\\).

With this encoding, it is straightforward to write a `derive` predicate[4]:

```
derive x\ x  x\ 1 . % identity
derive x\ C  x\ 0 . % constant function
derive x\ ((F1 x) + (F2 x))  x\ ((F1_ x) + (F2_ x)) :- % sum
  derive F1 F1_,
  derive F2 F2_.

% product
derive x\ ((F1 x)*(F2 x))  x\ ((F1_ x)*(F2 x) + (F1 x)*(F2_ x)) :-
  derive F1 F1_,
  derive F2 F2_.

derive x\ (sin (F x))  x\ ((F_ x)*(cos (F x))) :- % sinus
  derive F F_.
```

Note that the `x` can be renamed (by αconversion) in any expression.

Hence the following goal can be deduced from the program[5]:

```
derive X\ (sin X+2*X) X\ ((1+(0*X+2*1))*(cos X+2*X)).
```

[4]There is a better way to write this predicate using the universal quantification (`pi`) but this solution is simpler for a beginner.

[5]Of course, it would be necessary to apply a simplification procedure to the result of the derivation.

One important feature of λProlog used in our application is the existential quantification of variables in the body of a clause (noted `sigma`). All logic variables of a Prolog clause are universally quantified at the clause level. But if a variable occurs only in the body of the clause, it is logically correct to quantify it existentially in the body. For example, the classic clause

```
ancestor X Y :- father X Z, ancestor Z Y.
```

is better written

```
ancestor X Y :- sigma Z\ (father X Z, ancestor Z Y).
```

where the scope of the variable Z is more precisely defined.

3.2 One Simple Rule: BeamedNote

λProlog is really appropriate to implement a recursive-descent analyzer for the language generated by a grammar. We want to transform a grammar rule into a λProlog program.

As an example, we present the translation of the BeamedNote rule of the grammar:

BeamedNote ::= *Beam* (`left_tip` NoteGr :
`closev` [NoteGr | ConsRest]$*$:
`right_tip` NoteGr)

To make this syntax computable, we transform it using the following operators:

- `'::-'` The constructor of a rule.
- `'&&'` The connector of conjunction (the concatenation in the original grammar).
- `'##'` The operator of factorization (:).
- `'?$'` The position operator ($\mathcal{P}$).

So we get

```
beamed_note ::-
  beam BeamSeg &&
    ( ?$(left_tip BeamSeg) && note_gr BeamSeg ##
      rest_beamed_note BeamSeg ##
      ?$(right_tip BeamSeg) && note_gr BeamSeg ).
```

where the component `closeV` [NoteGr | ConsRest]$*$ is hidden in the operator `rest_beamed_note`

Then, the rule is translated into the following λProlog clause:

```
beamed_note Bar_In Bar_Out Curs_In Curs_Out :-
  beam BeamSeg Bar_In Bar_I1 Curs_In Curs_I1,
  left_tip BeamSeg Curs_I1 Curs_I2,
  note_gr BeamSeg Bar_I1 Bar_I2 Curs_I2 Curs_I3,
  rest_beamed_note BeamSeg Bar_I2 Bar_I3 Curs_I3 Curs_I4,
  right_tip BeamSeg Curs_I4 Curs_I5,
  note_gr BeamSeg Bar_I3 Bar_Out Curs_I5 Curs_Out.
```

This translation consists of adding the circulation of the parsed structure and the cursor. In fact, most of the time we have to add four parameters (two for input and two for output) to each predicate. The way the parameters are transmitted between two predicates is defined by the operators.

This translation is an extension of the DCG one. There are four added arguments instead of two but the principles are the same. As for DCGs, it is possible to automatically translate the grammar down to λProlog code. This translator will be described in the next section.

3.3 Touching Objects: Segment - Symbol

To decompose correctly a symbol touching a segment (*e.g.* connection between flat and beam in Fig. 1) we delete all the segments recognized as "stem" and "beam". But to keep the structural definition of the grammar we introduce an operator `delay` to modify the operational part of the parser.

For example the NoteGrU rule

NoteGrU ::= *Stem* `closel_downtip` NoteHead

will be finally coded with:

```
note_gr_U BeamSeg ::-
  stem BeamSeg StemSeg &&
  ?$(closel_downtip StemSeg) && checkHead &&
  delay noteHead.
```

The operator `delay` will stop the clauses `note_gr` at the level of the predicate `noteHead`, until the end of the parsing of the bar is achieved. At this moment, all the segments labeled "stem" and "beam" are deleted from the image (making a natural decomposition between a segment and a touching symbol). Then the process handles the connected components and the parsing restarts at the level where each clause `note_gr` was stopped, with a new input structure.

The previous clause `note_gr_U` succeeds if `checkHead` finds a `notehead` at the position defined by the position operator. `delay` then simply adds the predicate `noteHead` (with some of its arguments) to the list of all the already stopped note-Head. At the end of the bar, the segments labeled "stem" and "beam" are deleted from the bar structure. Then the connected components are calculated in a new bar structure. Each delayed predicate is awoken with that new bar structure in which the symbols touching segments are correctly decomposed.

Note that this `delay` mechanism is not the same that the `freeze` of Prolog II. Here, the delay is explicitly handled by the interpreter of the parser and does not depend on variable bindings. However, because goals are manipulated, this mechanism can be seen as a higher-order one.

3.4 Full Score

To recognize full scores from rules for single staves, we use the map_staff operator. We present here the translation of the BarSystem rule of the grammar:

```
BarSystem ::= map_staff(Heading)
              map_staff(BarGr)
              map_staff(BarStep)
```

As for a single staff rule (like BeamedNote), to make this syntax computable we use the following operators:

- `'::-/'` The constructor of a rule for full scores.
- `'&&/'` The connector of conjunction between two full scores rules.

So we get

```
bar_system ::-/ map_staff heading &&/
                map_staff bar_gr &&/
                map_staff bar_step &&/
```

The difference with the operators used in the BeamedNote rule is in the added parameters. The operators `'::-/'` and `'&&/'` add parameters which are lists of the parameters added by `'::-'`, `'&&'`, etc.

To translate the operator `map_staff`, we use the predicate `mappred` which map the parameter of `map_staff` to each staff of the system. So we have the following clauses in λProlog:

```
mappred Pred [] [].
mappred Pred [X1 | L1] [X2 | L2] :-
  Pred X1 X2,
  mappred Pred L1 L2.
```

and

```
bar_system List_Bar_In List_Bar_Out ... :-
  mappred heading List_Bar_In List_Bar_1 ... ,
  mappred bar_gr List_Bar_1 List_Bar_2 ... ,
  mappred bar_step List_Bar_2 List_Bar_Out ...
```

the `'...'` represents the attributes which are not shown here for legibility reasons.

3.5 Future Processing of Other Difficulties

The system has to be able to decide if a connected component is correctly decomposed in order to continue the decomposition if necessary. The symbols are represented by the connected components, therefore, it is these components (or their decomposition) which are presented to a classifier (in an array of pixels form). The classifier has to reject an entry which does not correspond to a learned class in order to determine if the symbol is correctly decomposed or not. According to the work done on character recognition, it seems that the classifiers based on Radial Basis Function (RBF)[12] have this characteristic. We will have to add such a kind of classifier to the grammar. With this classifier, the system would manage to solve the problems according to the aims like touching objects (Symbol - Symbol), broken objects, polyphony, syntactic checking and expansion to handwritten scores [9].

4 Compilation of the Grammar with λProlog

We have described a grammar used to specify the music score we want to analyze (see 2.6). We have shown how a rule of this grammar can be implemented as a λProlog clause in order to directly get a parser. We show in this section how the translation can be done automatically using λProlog.

4.1 Writing the Translator in λProlog

Writing a translator from DCG grammar to Prolog code is not an easy exercise ([5] page 221). Even if it is conceptually simple, Prolog is not so good to handle variables in data-structure. It has been shown in [13] that the task is more simple using λProlog.

4.1.1 Basic Translation

A translator for DCG basically has to add two new arguments to every literal of a clause. Because terms are curried in λProlog, adding arguments to a term just means applying the term to the arguments. For example, if we want to add the arguments `In` and `Out` to `Lit`, we write `(Lit In Out)`. Hence, the clause to translate a literal may be written as follows:

```
expand Lit In\Out\ (Lit In Out) :- literal Lit.
```

In Prolog, this operation needs the extra-logical `=..` (see the `tag` predicate of [5]).

A DCG translator also has to make the appropriate links between these new arguments. One of the difficulties is to introduce new intermediate variables. A conjunction `(a, b)` of a DCG rule applied to `In` and `Out` is translated in the conjunction of goals `a In X, b X Out`. In order to avoid name clashes and because the intermediate variable has a local known scope, the existential quantification (`sigma`) is appropriate to define the translation:

```
expand (Lit1, Lit2) In\Out\ (sigma X\ (Lit1_ In X, Lit2_ X Out)):-
  expand Lit1 Lit1_,
  expand Lit2 Lit2_.
```

The complete translator for standard DCG is given in [13].

This translation scheme can be applied to the extended DCG presented in section 2.6. For example the rule for translating the `&&` connector is written similarly:

```
expand (Left && Right)
       Si\So\Ci\Co\ (sigma LinkS\ (sigma LinkC\
         (Left_ Si LinkS Ci LinkC, Right_ LinkS So LinkC Co))) :-
  expand Left Left_,
  expand Right Right_.
```

Thanks to the expressive power of the language, the translator needs only about hundred lines of commented code (for compiling the twelve connectives of the grammar-language). This conciseness of the code ensures clarity and extensibility.

4.1.2 Compiling Mappings

One of the original parts of the grammar is the handling of full scores using a higher-order *map* connector. As presented in section 3.4, this connector can be translated into λProlog with a `mappred` predicate. The next step is to *compile* the `mappred` calls using the standard technique of partial evaluation: for each instance of a `mappred` call, an ad hoc predicate is generated.

For example, the *map_staff(Heading)* goal in the grammar is translated into the λProlog goal `(map_heading L1 L2 ...)` with the predicate `map_heading` defined as a specialization of `mappred` on `heading`:

```
map_heading [] [] ...
map_heading [X1 | L1] [X2 | L2] ... :-
  heading X1 X2,
  map_heading L1 L2 ...
```

This translation optimization leads to first-order code instead of the expensive higher-order code. We get efficient λProlog code and so, we have a higher-order grammar feature for free.

5 A Prototype

We give in this section a description of the architecture of the complete system and some encouraging results obtained with the first prototype.

5.1 Architecture

One of the advantages of the system is the compilation of the grammatical modules down to λProlog (**gram2pm** in Fig. 5) which makes it adaptable to other recognition problems by simply changing the grammar. The system is made of two parts: one in C code and one in λProlog code. The graphical interface is the entry point of the program and starts λProlog which then control all the process by calling C modules for image processing (segment extraction), classification... or λProlog modules for the parsing (Fig. 5).

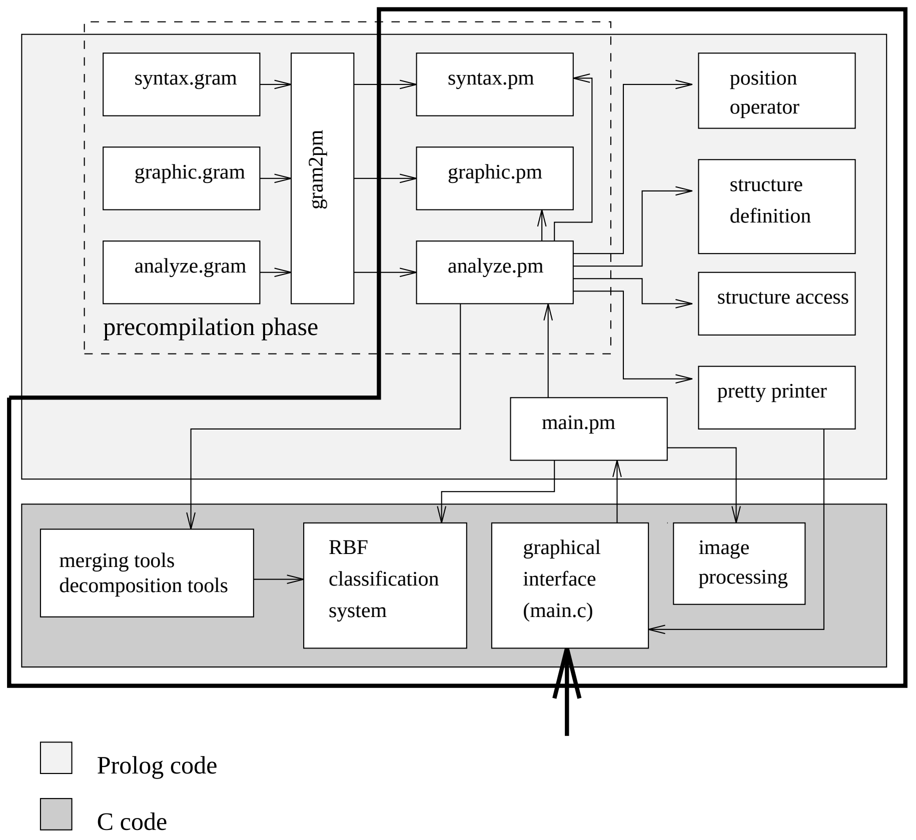

Figure 5: Architecture of the system

5.2 First Results

The current grammar is completely defined for full scores, with different voices on a single staff, chords on a voice, accents, tuplets (*e.g.* triplet), pause, octave, dynamic markings, phrasing slurs, rhythmic slurs and appoggiatura. Abbreviations, ornaments (except appoggiatura) and lyrics are not yet included in the grammar. This work was validated by a professional musician.

Most of the tools needed by the parser and some of the position operators are already written. Because of the independence between the grammar and the parser, we can test the parser with a small grammar without lost of generality. We have defined a subset of the initial grammar to recognize polyphonic full scores with clefs, key and time signature, half, quarter, single eighth notes and beamed notes, accidentals, dots after a note, rests, bar lines, with the `delay` operator and the processing of the pitches. As the system is able to recognize full scores, it is also able to make two kinds of control: first the number of beats in a bar (per voice) according to the time signature and second the vertical alignment of notes in a system. These two controls allows to detect most of recognition errors and sometimes to propose a correction of those errors.

To compute an image of 1900x900 pixels, a 3 staves full score with 78 recognized elements (on 282 parsed elements) on a Sun Sparc 10, it takes 6s for the image processing (extraction of segments, detection and staves removal, building of the structure) and 5s for the parsing (using λProlog). For the moment, the prototype represents 600 lines of commented grammar code, 5 000 of commented λProlog code and 13 000 lines of commented C code for the image processing module.

Due to time constraints, we have not yet added the RBF classification system [12], and the decomposition and merging of connected components. To test the implementation of the grammar, the current classification system is simply reduced to a file with the class of each recognizable component. These elements are tools for the parser, and should not be a problem for scaling the system. Indeed the parser is the kernel of the system and is independent of the grammar and the classification, decomposition and merging tools. Once the parser works, the system could be scaled up only by changing the grammar given to the parser while the tools will be written independently.

By contrast with most other systems, ours does not label at the primitive extraction, but it is the grammar by controlling the whole recognition process, which labels segments by using the context, when a rule can be applied to several elements. At the end of the parsing, this system can also point out the unrecognized objects or the bars which do not match the music syntax (susceptible to being badly recognized).

6 Conclusion

6.1 Music score recognition

We defined a recognition system for music scores fully controlled by a grammar. The current grammar is able to deal with full scores, with different voices on a single staff, with chords on a voice, to recognize accents, phrasing slurs, dynamic markings, etc.

As opposed to the other systems, our approach formalizes all the rules used for recognition and enables a maximum integration of the context. That formalization also produces an homogeneous system.

In most other systems, the syntax is used to check a labeling. Here, the syntax is used to control the whole recognition process and helps having a more reliable labeling. Usually, the grammatical methods work only at a high level, whereas this system can get back to the image level to produce an accurate decomposition and thus a good recognition.

One advantage of the grammar formalization is the separation between the operating part of the system and the definition of musical rules. That separation allows us to easily modify the rules or adapt the system to another kind of structured document. Indeed, one only has to define a new grammar to build a new recognition system, using the same parser.

Moreover, this system is an answer to the presently unsolved problems described by [2] and [3]: reconstruction of broken objects, decomposition of touching objects. It should also contribute in solving the scaling up problem. It is already able to deal with full scores, polyphony (for the moment only 2 voices per staff). With the help of the number of beats in a bar (according to the time signature) and the vertical alignment of notes (in full scores) the system can detect and correct recognition errors on note duration. This produces more reliable results.

6.2 λProlog use

The use of λProlog in this application is justified by the formal approach, i.e. the grammatical approach we took. Implementing a parser with λProlog is a straightforward task which can be proved correct and be done automatically.

However, λProlog is not appropriate to implement low level algorithms like pattern recognition or object classification. So, it was essential for our application to dispose of a λProlog system fully interfaced with foreign languages (C in our case).

λProlog, has also been used to compile the grammar. Thanks to the high-level features of the language, the λterms, the compiler has been quickly written and can be easily extended (for new grammar connectors).

Acknowledgments The authors would like to particularly thank Bernard Retif, musician, in charge of computer music lectures at the University of Rennes II, for the contribution of all his musical skills and the time he spent for the development of the grammar adapted to complex scores.

References

[1] A. Andronico and A. Ciampa. On automatic pattern recognition and acquistion of printed music. In *Proceedings of the International Computer Music Conference*, pages 245–278, Venice, Italy, 1982.

[2] Dorothea Blostein and Henry Baird. A critical survey of music image analysis. In Springer-Verlag, editor, *Structured Document Image Analysis*, pages 405–434, Eds. H.S. Baird, H. Bunke, K. Yamamoto, 1992.

[3] Nicholas P. Carter and Richard A. Bacon. Automatic recognition of printed music. In Springer-Verlag, editor, *Structured Document Image Analysis*, pages 456–465, Eds. H.S. Baird, H. Bunke, K. Yamamoto, 1992.

[4] N.P. Carter, R.A. Bacon, and T. Messenger. The acquisition representation and reconstruction of printed music by computer : a review. *Computers and the Humanities*, 22:117–136, 1988.

[5] W.F. Clocksin and C.S. Mellish. *Programming in Prolog.* Springer-Verlag, 1987.

[6] A. Colmerauer. *Les Grammaires de Métamorphose.* Technical Report, Groupe d'Intelligence Artificielle, Marseille, France, 1975. [8].

[7] A. Colmerauer. *Les systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur.* Publication Interne 43, Département d'Informatique, Université de Montréal, Canada, 1970.

[8] A. Colmerauer. Metamorphosis grammars. In L. Bolc, editor, *Natural Language Communication with Computers*, pages 133–187, Springer-Verlag, 1978.

[9] Bertrand Couasnon and Jean Camillerapp. Using grammars to segment and recognize music scores. In *International Association for Pattern Recognition Workshop on Document Analysis Systems*, pages 15–27, Kaiserslautern, Germany, October 1994.

[10] I. Fujinaga. *Optical Music Recognition using projections.* Master's thesis, McGill University, Faculty of Music, Montreal, Canada, 1988.

[11] Ichiro Fujinaga, Bo Alphonce, Bruce Pennycook, and Natalie Boisvert. Optical recognition of music notation by computer. *Computers in Music Research 1*, 161–164, 1989.

[12] Pascal Gentric. Constructive methods for a new classifier based on a radial-basis-function neural network. In *International Workshop on Artificial Neural Networks*, Barcelona, Spain, 1993.

[13] S. Le Huitouze, P. Louvet, and O. Ridoux. Logic grammars and λProlog. In D.S. Warren, editor, *10th Int. Conf. Logic Programming*, pages 64–79, MIT Press, 1993.

[14] Hirokazu Kato and Seiji Inokuchi. A recognition system for printed piano music using musical knowledge and constraints. In Springer-Verlag, editor, *Structured Document Image Analysis*, pages 435–455, Eds. H.S. Baird, H. Bunke, K. Yamamoto, 1992.

[15] Philippe Martin. Reconnaissance de partitions musicales et réseaux de neurones: une etude. In *RFIA, Septième congrès Reconnaissances des Formes et Intelligence Artificielle*, pages 217–226, Paris, France, 1989.

[16] Philippe Martin. *Réseaux de neurones artificiels : Application à la reconnaissance optique de partitions musicales.* PhD thesis, IMAG, Grenoble, France, 1992.

[17] D.A. Miller and G. Nadathur. Higher-order logic programming. In E. Shapiro, editor, *3rd Int. Conf. Logic Programming, LNCS 225*, pages 448–462, Springer-Verlag, 1986.

[18] D.A. Miller and G. Nadathur. A logic programming approach to manipulating formulas and programs. In S. Haridi, editor, *IEEE Symp. Logic Programming*, pages 379–388, San Francisco, CA, USA, 1987.

[19] F.C.N. Pereira and D.H.D. Warren. Definite clauses for language analysis. *Artificial Intelligence*, 13:231–278, 1980.

[20] Vincent Poulain d'Andecy, Jean Camillerapp, and Ivan Leplumey. Kalman filtering for segment detection: application to music scores analysis. In *ICPR, 12th International Conference on Pattern Recognition (IAPR)*, pages 301–305, Jerusalem, Israel, October 1994.

[21] D. S. Prerau. Do-re-mi: a program that recognizes music notation. *Computer and the Humanites*, 9(1):25–29, January 1975.

[22] J. W. Roach and J.E. Tatem. Using main knowledge in low-level visual processing to interpret handwritten music : an experiment. *Pattern Recognition*, 21(1):33–44, 1988.