A GRAMMAR FOR PLAINE AND EASIE CODE

David Rizo, José M. Iñesta

Dept. Lenguajes y Sistemas Informáticos
Universidad de Alicante, E-03071 Alicante, Spain
drizo@dlsi.ua.es

ABSTRACT

Plaine and Easie Code is being used as the format for encoding musical content in the RISM catalogue. In this paper we propose an attributed grammar that formally specifies it. The grammar specification is written using a EBNF notation and is accompanied by syntax diagrams and some examples that show how it parses a valid input. With the formalization of the *Plaine and Easie Code* , it is easier to create translators to other formats, analyze possible extensions, and avoid syntactic misunderstandings in the encoding process. The proposed grammar has been successfully used to decode more than one million incipits from the RISM catalogue in an automatic way. In the near future, we are planning to propose some extensions to the PAEC code and specify translation semantic actions associated to the grammar productions in order to convert from PAEC to the MEI format.

1. INTRODUCTION

The *Plaine and Easie Code* [2] (PAEC) is one of the earliest codes to describe music in the information age. It was originally designed to encode music using just typewriter characters for being used with or without computers.

Its main application since its release has been the description of incipits for the Répertoire International des Sources Musicales (RISM) database [1]. This corpus contains music incipits mainly from the XVII to XIX centuries, with notations that range from mensural to western modern notation.

PAEC is thoroughly described in the International Association of Music Libraries, Archives and Documentation Centres [1] using natural language with examples enough to make it understandable. However, there are situations in which some doubts about the encoding may arise to the non trained reader, such as whether it is correct or not to place a meter change in the middle of a bar. The same way a XSD or DTD restricts the valid files for XML based notations, we propose a formal grammar [4] to complement the current specification. This way, the possible format syntactic ambiguities are avoided.

Some extensions have been verbally proposed in the RISM community to accommodate other notations, like some Spanish scores from the XVII and XVIII centuries that currently are being transcribed to western modern notation to be introduced in the RISM catalogue. However, it is difficult to analyze the impact in the addition of new symbols due to the lack of formalism in the PAEC specification. The formal grammar we propose is the right tool to perform this analysis that allows a secure extension of the format.

There is a third way a formal grammar, and specifically an attributed grammar, can be useful to: the easy and systematic translation to other formats like MEI [3]. Some ad-hoc software to decode PAEC and render a score can be found in the Bononcini Project [2]. Utilities to translate PAEC to other well-known formats are publicly available in Internet: *pae2kern* for converting to Kern format, *pae2ly* getting a Lilypond score, or *pae2xml* translating to MusicXML. None of them use systematic software language translation techniques to perform the task.

2. FORMAL GRAMMARS

Formal languages are a useful tool for defining strings of symbols constrained by rules. A formal language is supported by a formal grammar that describes how to form strings of symbols that are valid for the language. It provides the language's syntax as a set of rules for generating strings starting form a particular *start symbol*. Those rules are named *productions* and express how the left side of a rule can be substituted by the right side of it. This rules can be applied in a series from the start symbol until a given valid symbol string is produced.

In addition, this grammar application as a language producer can be used the opposite way, as the basis for a "recognizer" that determines whether a given string belongs to the language or is grammatically incorrect. For that, formal language theory uses formalisms known as automata that are able to recognize the lexical elements of the language. These elements are the basic elements for a parser based on the grammar to make his recognition task.

Parsing is the process of recognizing an input string analyzing it against the grammar of the language. The first step to it is to break it down in its lexical elements and look at its analyzed form, known as its parse tree. The tree is able to depict the analysis process from the root, where the start symbol appears, until the branches, where the lexical elements are found. Every subtree in

[1] http://www.iaml.info/en/activities/projects/plain_and_easy_code

[2] http://www.bononcini.org/index.php?Itemid=9

this structure denotes the application of a production rule from the grammar in order to achieve the input correct analysis.

The computer implementation of a string parser can be done by using different techniques like top-down or bottom-up analyzers, whose suitability depends on the kind of language defined by the grammar. In the particular case of the grammar proposed in this paper, a top-down algorithm will be utilized.

The grammar expresses the structure of the language strings, not their meaning or what can be done with them. For that, an extension of the formal grammar has to be designed, known as attribute grammar that allows the production rules to be augmented with semantic attributes and operations, that are useful both for retrieving the information that the symbols may carry and for constructing practical language translation tools.

3. PAEC GRAMMAR

The proposed grammar is detailed in both tables for the lexical rules (Table 1) and syntactic productions (Table 2). The rules have been named in order to facilitate its understanding. Only those rules that are not self-explanatory are described using a semantic action (typewriter text between curly braces). Some simple syntactic rules are not directly implemented using a lexical rule because it would make impossible the parsing. The equivalent syntax diagrams for all rules can be found below in section 4.

In order to translate from a given PAEC input to any other format like MEI, just semantic actions must be added after each recognized token with the required content translations like figures or pitches literals. A complete implementation using the framework ANTLR v4 can be downloaded from *http://grfia.dlsi.ua.es/cm/*.

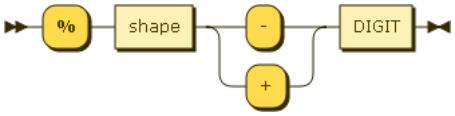
4. SYNTAX DIAGRAMS AND SAMPLE PARSING TREES

In this section all non unit rules in Table 2 are described using equivalent syntax diagrams [3]. They have been accompanied by sample input strings taken from the IAML PAEC specification website and drawn represented using parsing trees over the proposed grammar. See figures from 1 to 11.

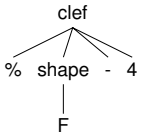
5. CONCLUSION

In this paper a formal attributed grammar in EBNF notation has been introduced that can help in avoiding syntactic ambiguities, translating to other formats, and be used as tool for analyzing possible extensions of the code. The proposed grammar has been successfully used to decode more than one million incipits from the RISM catalogue in an automatic way. In the near future, we are planning

[3]The online tool `http://bottlecaps.de/convert/` has been used to generate them.

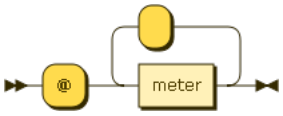


(a) Grammar rule

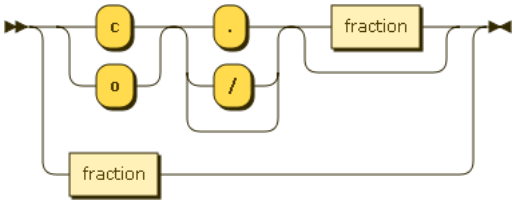


(b) Parsing tree for: `%F-4`

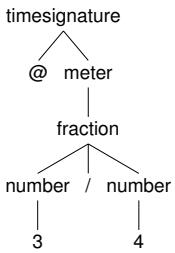
Figure 1: clef → **percent** shape (**minus** | **plus**) **digit**



(a) Grammar rule

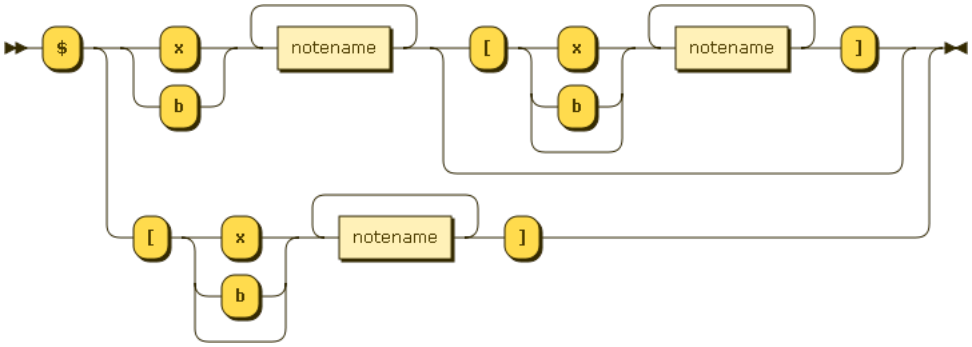


(b) Grammar rule for meter

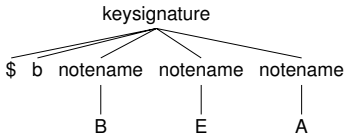


(c) Parsing tree for:
`@3/4 4/4`

Figure 2: timesignature →**at** meter (**space** meter)*

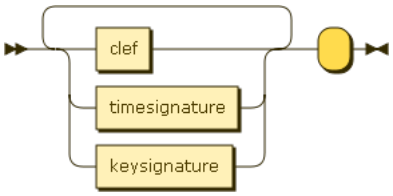


(a) Grammar rule

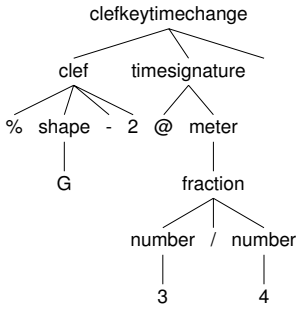


(b) Parsing tree for: `$bBEA`

Figure 3: **dollar** (**sharp** | **flat**) **notename+** (**leftbracket** (**sharp** | **flat**)? notename+ **rightbracket**)? | (**leftbracket** (**sharp** | **flat**)? notename+ **rightbracket**)?



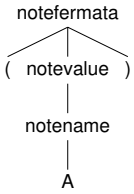
(a) Grammar rule



(b) Parsing tree for: `%G-2$bB@3/4␣`

Figure 4: clefkeytimechange →(clef | timesignature | keysignature)+ **space**

(a) Grammar rule



(b) Parsing tree for: `(A)`

Figure 7: notefermata →**leftpar** notevalue **rightpar**

to propose some extensions to the PAEC code and specify translation semantic actions associated to the grammar productions in order to convert from PAEC to the MEI format.

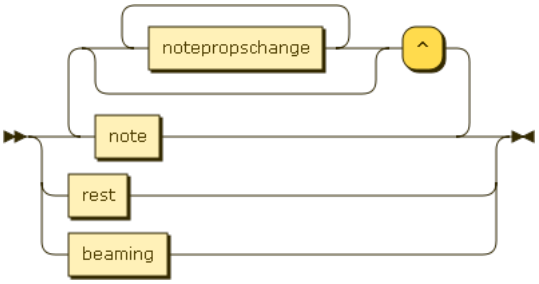
6. ACKNOWLEDGEMENTS

This work was funded by the University of Alicante's programme UA-GRE-12-34, and the Spanish project TIN2009-14247-C02-02. Special thanks to the CSIC researchers Antonio Ezquerro and Luis Antonio González, and Klaus Keill and Stephan Hirsch from RISM.

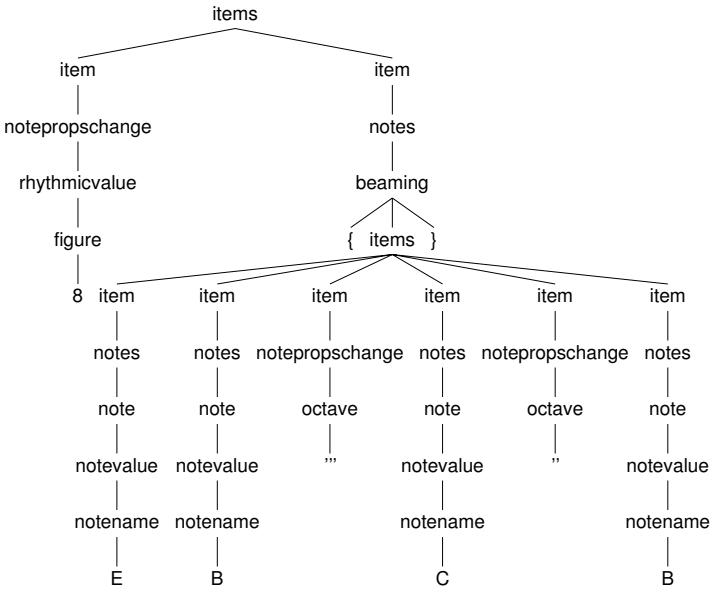
7. REFERENCES

[1] "Répertoire International des Sources Musicales (RISM) Series A/II: Music manuscripts after 1600 on CD-ROM," Tech. Rep., 2005.

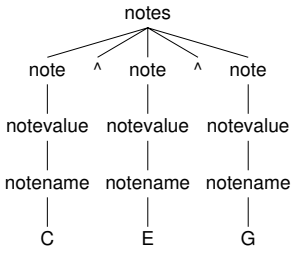
[2] B. Brook, "The Simplified 'Plaine and Easie Code System' for Notating Music: A Proposal



(a) Grammar rule

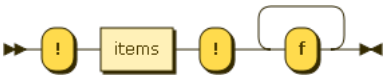


(b) Parsing tree for: `8{EB'''C''B}`

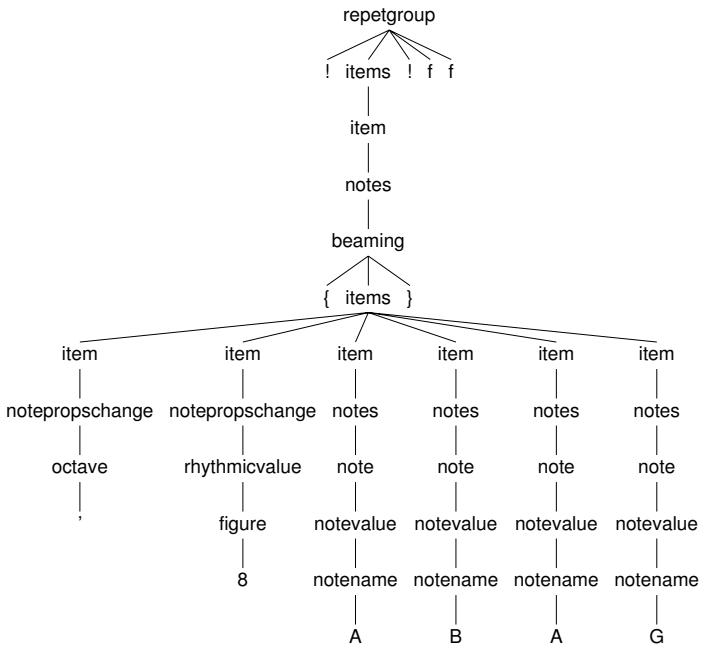


(c) Parsing tree for a chord: `C^E^G`

Figure 5: notes →(note (**chord** note)*) | rest | beaming

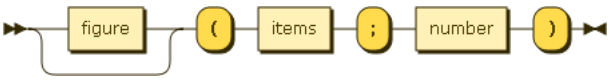


(a) Grammar rule

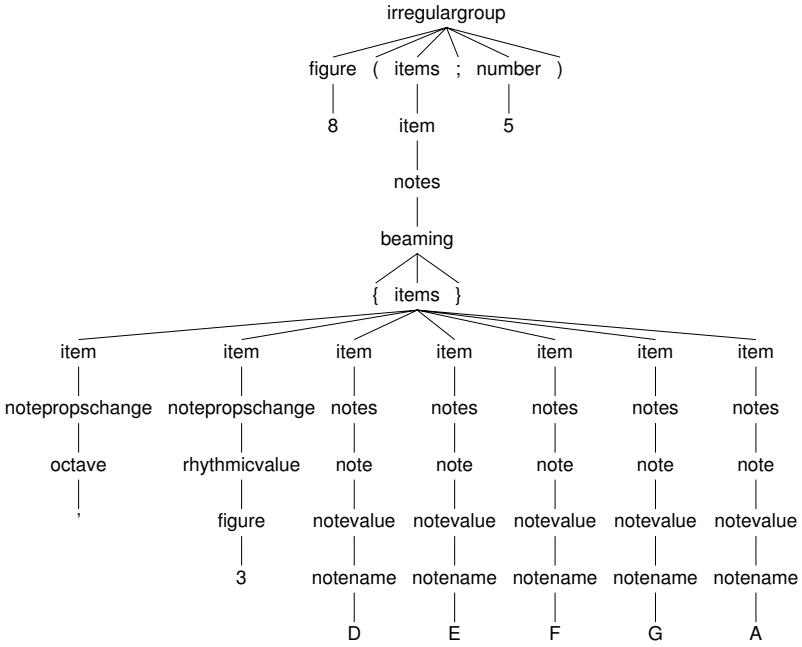


(b) Parsing tree for: `!'8ABAG!ff`

Figure 6: repetgroup →**repetgrpdelim** items **repetgrpdelim repetmark+**

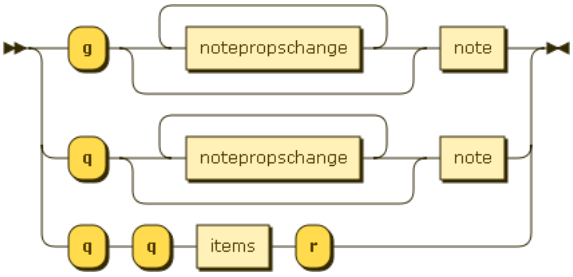


(a) Grammar rule

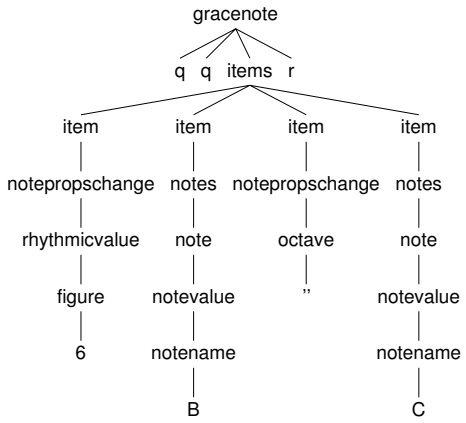


(b) Parsing tree for: `8({'3DEFGA};5)`

Figure 9: irregulargroup →figure? **leftpar** items **semicolon** number **rightpar**



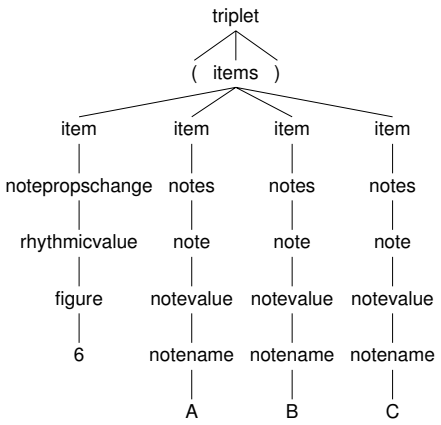
(a) Grammar rule



(b) Parsing tree for: `qq6B''Cr`

Figure 8: gracenote →(**letter_g** notepropschange* note) | (**letter_q** notepropschange* note) | (**letter_q letter_q** items **letter_r**)

(a) Grammar rule



(b) Parsing tree for: `(6ABC)`

Figure 10: triplet →**leftpar** items **rightpar**

for International Adoption,” *Fontes Artis Musicae*, vol. 12, no. 2-3, pp. 156–160, Jan. 1965.

[3] A. Hankinson, P. Roland, and I. Fujinaga, “The Music Encoding Initiative as a Document-Encoding Framework,” *International Conference on Music Information Retrieval*, 2011.

[4] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages and Computation (Addison-Wesley series in computer science)*, 1st ed. Addison-Wesley Publishing Company, Apr. 1979.

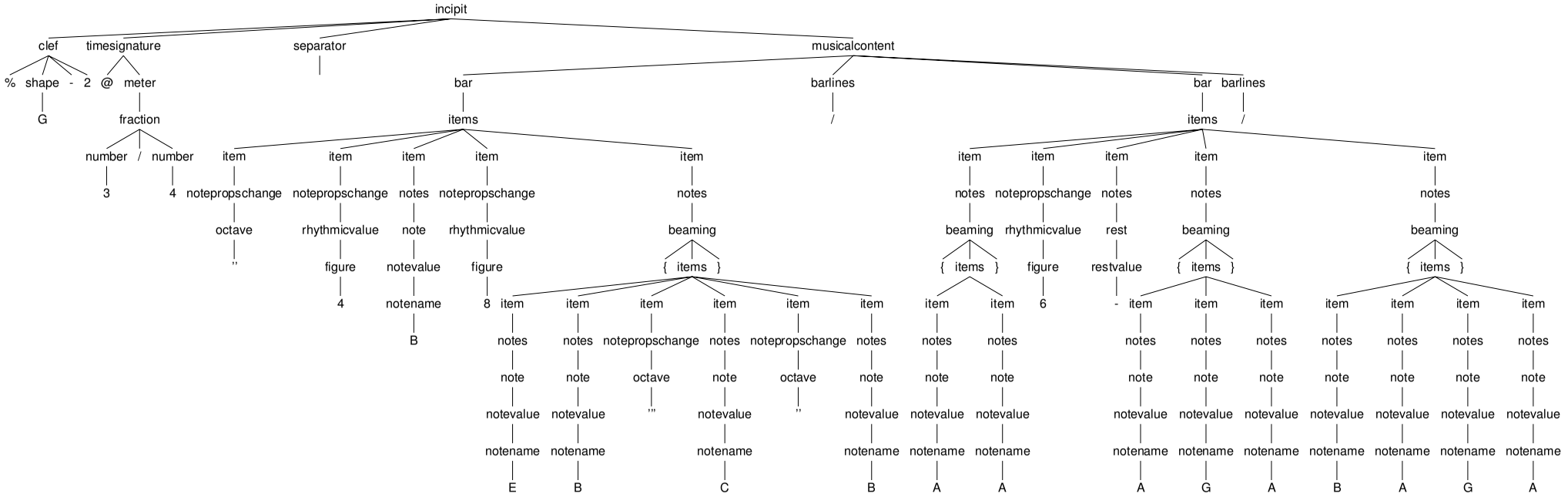
TOKEN	DEFINITION
notenames	`('A' .. 'G')`
digit	`('0'..'9')`
sharp	`'x'`
flat	`'b'`
doublesharp	`'xx'`
doubleflat	`'bb'`
natural	`'n'`
octave7	`'""'`
octave6	`'"'`
octave5	`'"'`
octave4	`''`
octave1	`',,,'`
octave2	`',,'`
octave3	`','`
chord	`'^'`
measurerest	`'='`
space	␣
tab	⌢
separator	space
percent	`'%'`
dollar	`'$'`
measurerepet	`'i'`
leftpar	`'('`
rightpar	`')'`
leftbracket	`'['`
rightbracket	`']'`
leftcurbraces	`'{'`
rightcurbraces	`'}'`
semicolon	`';'`
repetgrpdelim	`'!'`
repetmark	`'f'`
letter_g	`'g'`
letter_q	`'q'`
letter_r	`'r'`
measurerepet	`'i'`
minus	`'-'`
plus	`'+'`
common	`'c'`
perfect	`'o'`
trill	`'t'`
dot	`'.'`
colon	`':'`
questionmark	`'?'`
slash	`'/'`
at	`'@'`

Table 1: Lexical rules for the grammar.

incipit	→ clef timesignature? keysignature? separator musicalcontent
clef	→ **percent** shape (**minus**{`modern notation`} | **plus**{`mensural notation`}) **digit**
keysignature	→ **dollar** (**sharp** | **flat**) **notename+** (**leftbracket** (**sharp** | **flat**)? notename+ **rightbracket**)? | (**leftbracket** (**sharp** | **flat**)? notename+ **rightbracket**)?
timesignature	→ **at** meter (**space** meter {`use permanent meter changes`})*
meter	→ ((**common** | **perfect**) (**dot**{`perfect`} | **slash**{`allabreve`}) fraction?) | fraction
fraction	→ number (**slash** number)?
gracenote	→ (**letter_g** notepropschange* note {`acciaccatura`}) | (**letter_q** notepropschange* note {`appoggiatura`}) | (**letter_q letter_q** items **letter_r** {`slide or multiple appoggiatura`})
octave	→ **octave7** | **octave6** | **octave5** | **octave4** | **octave3** | **octave2** | **octave1**
rhythmicvalue	→ figure **dot***
figure	→ **digit** {`digit.value is the rhythmic value`}
accidental	→ **sharp** | **doublesharp** | **flat** | **doubleflat** | **natural**
musicalcontent	→ **bar** (barlines bar)* barlines?
bar	→ items | **measurerepet** | **measurerest number**? {`number.value is the number of repetitions`}
clefkeytimechange	→ (clef | timesignature | keysignature)+ **space**
items	→ item+
item	→ clefkeytimechange | irregulargroup | triplet | notes | repetgroup | notepropschange
repetgroup	→ **repetgrpdelim** items {`push the items to repeat`} **repetgrpdelim** (**repetmark**{`pop items to repeat`})+
triplet	→ **leftpar** items **rightpar**
irregulargroup	→ figure? {`denotes total value of group`} **leftpar** items **semicolon** number {`number of notes of grp`} **rightpar**
notepropschange	→ octave | rhythmicvalue | accidental
notes	→ note (**chord** notepropschange* note)* | rest | beaming
beaming	→ **leftcurbraces** items **rightcurbraces**
note	→ gracenote | notevalue | notefermata
notefermata	→ **leftpar** notevalue **rightpar**
rest	→ restvalue | restfermata
restfermata	→ **leftpar** restvalue **rightpar**
notevalue	→ notename notesuffix? **dot***
restvalue	→ **minus**
notesuffix	→ **trill** slur? | slur **trill**?
slur	→ **plus**{`slur note to next`}
barlines	→ **slash** | **slash slash** | **slash slash semicolon** | **semicolon slash slash** | **semicolon slash slash semicolon**
shape	→ **letter_g** | **notenames**
notename	→ **notenames**
number	→ **digit+**
separator	→ (**space** | **questionmark** | **semicolon**)+

Table 2: PAEC Grammar

(a) Grammar rule



(b) Parsing tree for: `%G-2@3/4$bBEA ''4B8{EB'''C''B}/{AA}6-{AGA}{BAGA}/`

Figure 11: incipit →clef timesignature? keysignature? separator musicalcontent