Generating equivalent rhythmic notations based on rhythm tree languages

Florent Jacquemard, Adrien Ycart, Masahiko Sakai

GENERATING EQUIVALENT RHYTHMIC NOTATIONS BASED ON RHYTHM TREE LANGUAGES

Florent Jacquemard
INRIA Sorbonne Universités
STMS (IRCAM-CNRS-UPMC)
IRCAM, Paris, France
florent.jacquemard@inria.fr

Adrien Ycart
School of Electronic Engineering
and Computer Science
Queen Mary Univ. of London, UK
a.ycart@qmul.ac.uk

Masahiko Sakai
Graduate School of Information Science
Nagoya University, Japan
sakai@is.nagoya-u.ac.jp

ABSTRACT

We propose a compact presentation of languages of preferred rhythms notation as formal grammars. It is based on a standard structure of rhythm trees capturing a wide range of rhythms in Western notation.

As an application, we then describe a dynamic programming algorithm for the lazy enumeration of equivalent rhythm notations (i.e. notations defining the same durations), from the simplest to the most complex. This procedure, based on the notion of rhythm grammars has been implemented and may be useful in the context of automated music transcription and computer-assistance to composition.

1. INTRODUCTION

Music notation is for music very much like writing for language. It serves as a support to convey ideas, to keep track of them in time, and as a working support for musical expression. In natural languages, there can be several synonyms to designate the same entity (such as "Rome", "Roma", "capital of Italy", "city of the seven hills", "eternal city"...) and one might prefer using one word over the others depending on the context, a special connotation, the linguistic register *etc*. Similarly, in common Western music notation, there are often many different ways to write a given sequence of durations, and the choice of one writing over another is up to the composer, and can be driven by many reasons, among which are the musical context in which it is written, or a particular interpretation or phrasing the composer wants to imply.

This is an important problem in applications related to score generation, in particular automated music transcription [1], score editors, or composer assistance environments [2]. In this context, it is interesting to assist users as much as possible in choosing appropriate notations for what they want to express.

A first question that arises is the definition of the domain of rhythms notations that one want to consider: which divisions (tuplets) are allowed? at which level? how many levels of division can we nest? In other words, we need some formalism to describe *languages of rhythms*. This corresponds for instance to the *codebooks* in the transcription procedure of [3], which are defined by *subdivision schemas* (the sequence of authorized successive regular subdivisions of a time interval), or similar notions already found in former work on transcription [4, 5, 6]. Another example is the choices of quantization parameters in user preferences of music editors such as Finale.

A second question is the design of efficient algorithms for exploring the set rhythm notations in a given language that satisfy some given properties. such that, for instance, the set of rhythms defining to the same effective given sequence of durations.

In this work, we follow a language theoretic approach to these problems of rhythm notation. We propose a notion of rhythm languages defined by formal context-free (CF) grammars, following [7] (Section 3). It generalizes the formalisms cited above, as well as a related formalism that we have used so far in a new tool for rhythm transcription [8]. The rhythms defined by such a grammar are, roughly, the derivation trees of the grammar, seen as rhythm trees (RT). The latter are a tree structure for the representation of timed sequences of events [9, 10], where the events are stored in the leaves of the tree and the duration are encoded in the tree structure, every branching defining a uniform division of a time interval (Section 2). They have been supported since many years as a native data structure for the representation of rhythms in visual environment for composition assistance based on functional programming languages such as Open Music or PWGL [2, 11].

Using standard formalisms such as CF grammars for the definition of rhythm languages allows to exploit efficient construction and decision procedures from the literature.

Here, we use an algorithm for the enumeration of RTs defined by a given acyclic grammar [12] (Section 3.4). The enumeration follows a rank assigned to RTs by weights added to the grammar's rules. These weights can be seen as a measure for rhythm complexity. The main advantages of this algorithm is that it does not need to compute all the RTs of the language in order to rank them, but instead, thanks to a monotony property of the weight functions, build the best trees from the best subtrees in a lazy way.

CF grammars are a concise and readable formalism to define RT languages, but they are also expressive. The languages they define are regular tree languages [13], and as such can be composed by Boolean operations. This en-

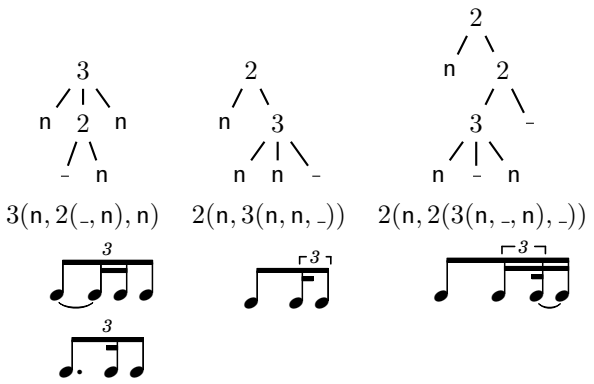

Figure 1. Rhythm Trees and corresponding notations.

ables incremental constructions of complex languages by composition of elementary languages, in a user-friendly fashion. In particular, it is possible to use Cartesian product in order to construct the intersection of two tree languages. We use this principle (Section 4) to construct a grammar defining the set of RTs from a given language L which correspond to a given sequence σ of duration values. Using the above-cited algorithm, we can then enumerate by ascending weight the RTs of L with duration value sequence σ. We then apply the same principle to provide a way to find all rhythms that denote the merging of two voices into one (Section 5).

2. RHYTHM TREES

A rhythm tree (RT) is a hierarchical representation of a sequence of timed events, where the events are in the leaves, and the inner nodes define the durations by successive subdivisions of an initial time interval.

2.1 Syntax

In our settings, a RT is either:

- a symbol n representing the beginning of an event, or
- a symbol _ representing the continuation of an event, or
- a tree of the form $t = p(t_1, \ldots, t_p)$ where p is a natural number (smaller than a given bound) and $t_1, \ldots, t_p$ is an ordered sequence of RTs.

The node labeled with p is called the *root* of t, and the sub-RTs $t_1, \ldots, t_p$ are called the *children* of t.

Note that every inner node of a RT is labelled with the arity (outer degree) of the node. We adopt this redundant notation only for readability purpose. Note also that we consider only one symbol n for the representation of every kind of events. This is also for the sake of simplification, but of course, n can be duplicated into as many symbols as needed for representing notes of different pitches, chords, rests... (see Section 2.3).

Example 1 *We present in Figure 1 three examples of RTs and the corresponding rhythm notations (with or without dots), according to the duration semantics defined in Section 2.2.*

The first RT on the left is a triplet. Its second child is further divided in 2 parts, and the first part is a continuation. This means that the corresponding note (second note of the RT) is tied to the previous note (first note). In the second RT, we have a division by 2 and then by 3, and a tie between the two last notes. The last RT presents three levels of division, and two ties. Note that the depth in the RTs reflects the beaming level in the notation. For readability purposes, some tied notes have been merged, for instance in the second example, the last two tied sixteenth notes have been merged into one single eighth note.

2.2 Semantics

To every RT t, we associate a sequence of positive rational numbers called rhythmic value of t. This numbers correspond to the inter-onset intervals (IOIs) between the onset of events described in the RT. In the following, when considering the leaves of a tree, we will interchangeably use the term IOI instead of positive rational number.

In order to define the rhythmic value of the RT t, we associate to every node ν of t a positive rational number called *duration value*, and denoted by $dur(t, \nu)$.

Intuitively, the duration of a node is divided uniformly in the duration of its children, and _ is used to sum duration of leaves. Formally,

If ν is the root of t, then $dur(t, \nu) = 1$.

Otherwise, $dur(t, \nu) = \frac{dur(t,\nu_0)}{t(\nu_0)} + cdur(t, \nu)$, where

- ν_0 is the parent of ν in t,
- $t(\nu_0)$ is the label of ν_0 in t (*i.e.* its arity),
- $cdur(t, \nu) = dur(t, \nu')$ if ν is a leaf and it has a next leaf ν' labelled with _ ,
- $cdur(t, \nu) = 0$ otherwise.

The choice of duration 1 for the root node is arbitrary. It means that the fractions refer to divisions of one beat but with different root durations, they could as well refer to other orders of magnitude (bar *etc*). Note that the durations are expressed in beats (time units relative to a tempo), which makes the definition of a tempo useless here.

The events represented by a RT are stored in its leaves. Hence, the event's durations, *i.e.* the actual rhythm corresponding to a RT, denoted $val(t)$, is defined from the duration values of its leaves. More precisely, let $\nu_1 \ldots, \nu_k$ be the enumeration, in depth-first ordering, of the leaves of t labelled with n. The *rhythmic value* of t is the sequence

$$val(t) = dur(t, \nu_1), \ldots, dur(t, \nu_k).$$

It is the empty sequence when $k = 0$. Intuitively, if a rhythm tree t represents the notation of a rhythm, its rhythmic value represents the way this rhythm sounds.

Two RT t_1, t_2 are called *equivalent*, denoted $t_1 \equiv t_2$ iff $val(t_1) = val(t_2)$.

Example 2 *The three RT of Figure 1 have a rhythmic value of* $[\frac{1}{2}, \frac{1}{6}, \frac{1}{3}]$ *and are therefore equivalent.*

Deciding the equivalence of two given RT t_1 and t_2 can be done simply by evaluating their respective rhythmic values (in linear time for each tree) and comparing the values.

2.3 Pitches, Grace notes, Rests...

Note that in the above RT representation, we consider only one generic symbol n to represent any kind of event.

Straightforwardly, we could introduce new symbols labeling the leaves of RTs in order to encode any kind of finite information such as pitches, chords, rests *etc*.

Alternatively, we could maintain along with a RT t a list of event's information of same length as the rhythm value of t. This list can be used later to recover this information (*e.g.* for rendering). These details are left out of this paper.

Grace notes are events of duration 0. The approach presented in this paper can deal with grace notes. Indeed, in a RT, one note preceded by one or several grace notes can be encoded in a single symbol, labelling a leaf. For instance, the symbol n_k could describe one note preceded by k grace notes. The definition of the rhythmic value of a RT should then be extended accordingly, by adding to the sequence k times the value 0 at the appropriate place.

2.4 Related Rhythm Tree Formalisms

The definition of RTs used in this paper differs slightly from the earlier definitions of rhythm trees in Patchwork [9] and Open Music [10]. In the latter rhythm trees, inner nodes are labelled by integer values, and denote the relative length durations of sibling nodes. For example, if a node has 2 sons labelled 1 and 2, the last son is twice as long as the first (hence this node corresponds to a triplet). This is exactly equivalent as if they were labelled 2 and 4, only the ratios matter. Note that these numbers have nothing to do with the numbers used in the above RT encodings, which are just the arity of the inner nodes and carry no information. We preferred here an all-symbolic encoding (without integral values) in order to fit with the formal language framework that we use to define rhythm languages.

3. RHYTHM LANGUAGES

In this section, we propose a general finite representations for sets of RTs called rhythm grammars. Their purpose is to fix the kind of rhythm notations that we want to consider, using declarative rules.

For instance, one rule may express that at some level, division by three is allowed but division by five is not. One other rule can express that at some level we can have a leaf labeled with a note (n) or labeled with a continuation ($\_$).

3.1 Rhythm Grammars

A *rhythm grammar* (RG) is a context-free grammar $\mathcal{G} = (Q, q_0, R)$ where Q is a finite set of non-terminals, q_0 is the initial non-terminal and R is a finite set of production rules of one of the forms

$$q \to q_1, \ldots, q_p \text{ with } q, q_1, \ldots, q_p \in Q,$$

$$q \to a \text{ with } q \in Q \text{ and } a \in \{\mathsf{n}, \_\}.$$

Intuitively, these rules are applied from top to bottom to generate RTs by replacement of non-terminals by subtrees. A rule of the first kind (called *inner rule*) generate an inner node of a RT, expressing a division by p of the duration of this node. A rule of the second kind (called *terminal rule*) generate a leaf of a RT, and expresses that the label a is allowed at this leaf.

Generally in the literature, one considers the (context-free) language of words generated by CF grammars – in our case, they would be words over $\{\mathsf{n}, \_\}$. However, recall that RTs encode events in the leaves but also durations of these events in the inner nodes. Since we need these two informations for rhythm encoding, we need both kind of nodes, and therefore we will be more interested in the set of RTs generated by a RG (which is a regular tree language), than in the (context-free) language of the words found in a traversal of the leaves of these RT.

Formally, the language $\mathcal{L}_q(\mathcal{G})$ of a RG $\mathcal{G} = (Q, q_0, R)$ in non-terminal $q \in Q$ is defined recursively by

$$\mathcal{L}_q(\mathcal{G}) := \{a \text{ if } q \to a \in R\} \cup \bigcup_{q \to q_1,\ldots,q_p \in R} \{p(t_1, \ldots t_p) \mid t_1 \in \mathcal{L}_{q_1}(\mathcal{G}), \ldots, t_p \in \mathcal{L}_{q_p}(\mathcal{G})\}.$$

The language of $\mathcal{G}$ is $\mathcal{L}(\mathcal{G}) = \mathcal{L}_{q_0}(\mathcal{G})$.

Example 3 *Let us consider the following RT $\mathcal{G}$, with initial non-terminal q_0.*

$q_0 \to \mathsf{n}$	$q_2 \to \_$	$q_3 \to \_$
$q_0 \to q_2, q_2$	$q_2 \to \mathsf{n}$	$q_3 \to \mathsf{n}$
$q_0 \to q_3, q_3, q_3$	$q_2 \to q_4, q_4$	$q_3 \to q_5, q_5$
	$q_2 \to q_5, q_5, q_5$	
$q_4 \to \_$	$q_5 \to \_$	
$q_4 \to \mathsf{n}$	$q_5 \to \mathsf{n}$	
$q_4 \to q_5, q_5, q_5$		

The three rules with left-hand side q_0 express that the root of RT in $\mathcal{G}$'s language can be either a single event n, or a division by 2, with children in q_2, or a division by 3, with children in q_3.

At the next level, starting with q_2, we can have leaves labeled with $\_$ or n, or division by 2 (with children in q_4), or division by 3 (with children in q_5). Starting with q_3, we can have only leaves or division by 2 (with children in q_5). At the next level, q_5 will generate only leaves ($\_$ or n) whereas at q_4 we can have a last division by 3.

Note that rules of $\mathcal{G}$ are acyclic, and hence define a finite RT language.

All the three RTs of Figure 1 are in the language of this RG. The precise computations to obtain these RT are described in Example 5 below.

3.2 Weighted Rhythm Grammars

We extend RG into weighted rhythm grammars (WRG) by adding a weight (real value) to each production rule, with the notations $q \xrightarrow{w} q_1, \ldots, q_p$ and $q \xrightarrow{w'} a$ respectively for weighted inner and terminal rules with weights w and w'.

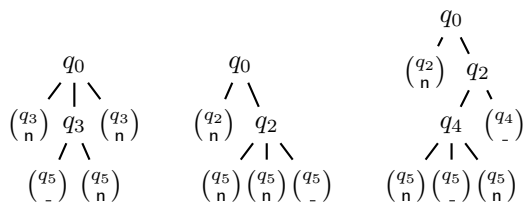

Figure 2. Runs on the RTs of Fig. 1 of the WRG of Ex. 4.

Example 4 *We describe below a weighted extension of the RG of Example 3.*

$$
\begin{array}{lll}
q_0 \xrightarrow{0.1} \mathsf{n} & q_2 \xrightarrow{0.2} \_ & q_3 \xrightarrow{0.2} \_ \\
q_0 \xrightarrow{0.35} q_2, q_2 & q_2 \xrightarrow{0.1} \mathsf{n} & q_3 \xrightarrow{0.1} \mathsf{n} \\
q_0 \xrightarrow{0.45} q_3, q_3, q_3 & q_2 \xrightarrow{0.5} q_4, q_4 & q_3 \xrightarrow{0.5} q_5, q_5 \\
 & q_2 \xrightarrow{0.6} q_5, q_5, q_5 & \\
q_4 \xrightarrow{0.2} \_ & & \\
q_4 \xrightarrow{0.1} \mathsf{n} & q_5 \xrightarrow{0.2} \_ & \\
q_4 \xrightarrow{0.75} q_5, q_5, q_5 & q_5 \xrightarrow{0.1} \mathsf{n} &
\end{array}
$$

The weights can be thought as penalties for symbols or divisions. For instance, every note n *induces a minimal penalty of* 0.1 *whereas a continuation* _ *induces a penalty of* 0.2*, in order to penalize ties. At level 0 (root), a duplet induces a penalty of* 0.35 *whereas for triplet it is a bit bigger (*0.45*). The situation is similar at lower levels. See Section 3.3 for lengthier discussion on choosing weights.*

The weights in WRG are used to rank the RTs (by ascending weight). In order to associate to every RT a unique weight by a WRG, we use the following notion of run. Intuitively, a run represents the sequence of application of the grammar's production rules in order to obtain a RT. Formally, a *run* of a WRG $\mathcal{G} = (Q, q_0, R)$ on a RT t is a relabelling of the nodes of t with production rules of R, such that for every inner node ν labeled with $q \xrightarrow{w} q_1, \ldots, q_p$, for every $1 \leq i \leq p$, the i^{th} children node of ν is labeled by a rule of R of left-hand side q_i.

Example 5 *In Figure 2, we display runs of the WRG of Example 4 on the three RTs of Figure 1. For the sake of readability, we only display the left-hand side of rules on inner nodes and the non-terminal and symbol on leaves (the whole run can be easily recovered from this).*

Note that there can be several runs for one RT of the language (although it is not the case in the above example).

The *weight* of a run is the sum of weights of the rules at all nodes of this run. The weight associated to a RT t by a WRG $\mathcal{G}$ is either undefined if there exists no run of $\mathcal{G}$ on t, or the minimum of the weights of the runs of $\mathcal{G}$ on t.

Example 6 *The weights associated by the WRG of Example 4 to the RTs of Figure 1 are respectively:*

$$
\begin{array}{rl}
3(\mathsf{n}, 2(\_, \mathsf{n}), \mathsf{n}): & 1.45 \\
2(\mathsf{n}, 3(\mathsf{n}, \mathsf{n}, \_)): & 1.45 \\
2(\mathsf{n}, 2(3(\mathsf{n}, \_, \mathsf{n}), \_)): & 2.3
\end{array}
$$

3.3 Choice of the Weight Values

The choice of the weights in production rules is crucial, as it is what will determine how the various RTs will be ranked. Generally speaking, we want RTs to be ranked according to their complexity. However, rhythmic complexity is difficult to define because it is highly subjective, and depends strongly of the context in which the rhythm is embedded. Some studies have tackled the perception of rhythmic complexity (see [14] for various definitions), but to the best of our knowledge, none have treated of the complexity of rhythm notation.

One naive definition would be to consider basic measures such as the size and the depth of the RT: the more nodes there are and the deeper the tree is, the more "complex" the rhythm is likely to be. The resulting weight function would be of the form : $q \xrightarrow{d+s*p} q_1, \ldots, q_p$ where d and s are two positive coefficients penalizing the depth and the size of the tree, respectively.

This definition shows its limits when comparing for example a quintolet and a sextolet. The quintolet is quite unusual, and will often be seen as complex, while a sextolet is much more common. Size alone cannot account for the complexity of a notation. Same with depth:  is generally considered less complex than a quintolet, even though its tree is deeper.

To go around this problem, a measure for notation complexity that tries to take into account these musical considerations was proposed in [8]. It is based on a ranking of divisions complexity proposed in [15], in which arities are ranked as follows, from less complex to more complex : 1, 2, 4, 3, 6, 8, 5, 7,... We can thus define a function $\beta(p)$ representing the penalty associated to an arity p, and the weight function becomes $q \xrightarrow{\beta(p)} q_1, \ldots, q_p$

Still, in this measure, the function $\beta(p)$ is arbitrarily determined, even though its contour is determined based on musical considerations. To determine it more relevantly, we could perform large-scale analysis of music score corpora, and choose $\beta(p)$ according to the frequency of each arity (the higher the frequency, the lower the weight). We could go even further in this corpus-based estimation of weights by assigning to each production rule of our RG a weight inversely proportional to its frequency in a corpus of scores. This would allow us to not only take into account the arity of the nodes in the RT, but also the depth at which they are found, and even the series of subdivisions they are in, depending on the grammar chosen.

3.4 k-best Parsing Algorithm

The k-best Parsing Algorithm [16] is a dynamic programming algorithm that, given a weighted context-free grammar $\mathcal{G} = (Q, q_0, R)$, enumerates its k best runs (ranked by their weight), where k is a parameter given by the user. In [8], we proposed an implementation of this algorithm from which the following stems. Here, we apply this algorithm to the WRG $\mathcal{G}$, and enumerate its least-weight RTs.

The algorithm uses a table which associates to every non-terminal $q \in Q$ two ordered lists of runs of $\mathcal{G}$:

$bests[q]$, containing the runs of $\mathcal{G}$ on RTs with q at the root and of minimal weight, as well as their weights.

$cands[q]$ (candidate bests), containing runs of $\mathcal{G}$ among which the next best will be chosen.

In each of those lists, the runs are not stored in-extenso. The elements of those lists are lists of pairs of the form $\big(\langle q_1, i_1\rangle, \ldots, \langle q_p, i_p\rangle\big)$, where every $q_j \in Q$ and every i_j is an index in $bests[q_j]$, and R contains a rule $q \xrightarrow{w} q_1, \ldots, q_p$. Those lists will also be called runs in what follows.

3.4.1 Initialization of the table

For each $q \in Q$, $bests[q]$ is initialy empty, and $cands[q]$ is initialized with one run $\big(\langle q_1, 1\rangle, \ldots, \langle q_p, 1\rangle\big)$ for each rule $q \xrightarrow{w} q_1, \ldots, q_p$ in R, with an unknown weight, and one run $()$ of weight w' for each rule $q \xrightarrow{w'} a$ in R.

3.4.2 Algorithm

The algorithm evaluates the weights of candidates in the table in a lazy fashion, and transfers candidates of minimal weights in the best list. It works recursively, by computing the weight of each run from the weights of its sons. The main function, $best(k, q)$, returns the k-th best run of root q:

1. If $best[q]$ already contains k elements or more, then return the the k-th run of this list and its weight.

2. Otherwise, evaluate the weight of all runs in $cand[q]$ as follows: for a run $\big(\langle q_1, i_1\rangle, \ldots, \langle q_p, i_p\rangle\big) \in cand[q]$ whose weight is unknown, call recursively $best(i_j, q_j)$ for each $1 \leq j \leq p$, and then evaluate the weight by summing the weights of the sub-runs and the weight w of the rule $q \xrightarrow{w} q_1, \ldots, q_p$.

3. Once all the weights of the runs in $cand[q]$ have been evaluated, remove the run of smallest weight from this list, add it to $best[q]$ (together with its weight). Then add to $cand[q]$ the following next runs, with unknown weight: $\big(\langle q_1, i_1{+}1\rangle, \ldots, \langle q_p, i_p\rangle\big)$, $\big(\langle q_1, i_1\rangle, \langle q_2, i_2{+}1\rangle, \ldots, \langle q_p, i_p\rangle\big)$, $\ldots$, $\big(\langle q_1, i_1\rangle, \ldots, \langle q_p, i_p + 1\rangle\big)$. This step ensures us that the next best is in the candidate list. Repeat 2. and 3. until $best[q]$ or $cand[q]$ is empty (in the later case, $best(k, q)$ is undefined).

4. ENUMERATION OF EQUIVALENT RHYTHMS

Now we have all the elements to represent and enumerate the set of rhythms equivalent to a given rhythm. Let us first reformulate precisely, in the above settings, the problem we are interested in:

given a weighted rhythm grammar $\mathcal{G}$ and a non-empty sequence σ of positive rational numbers (IOIs),

return a weighted rhythm grammar $\mathcal{G}_\sigma$ such that $\mathcal{L}(\mathcal{G}_\sigma) = \{t \in \mathcal{L}(\mathcal{G}) \mid val(t) = \sigma\}$.

Hence, given a RT t, the WRG $\mathcal{G}_{val(t)}$ will represent the set of WRT of $\mathcal{G}$ equivalent to t. Moreover, using the algorithm of Section 3.4, we can enumerate this set.

4.1 Grammar Product Construction

Let $\mathcal{G} = (Q, q_0, R)$. The construction of $\mathcal{G}_\sigma$ works as a Cartesian product, following the similar construction for tree automata [13]. The non-terminals of $\mathcal{G}_\sigma$ are pairs of the form $\langle \tau, q\rangle$ where $q \in Q$ and τ is a part of σ, in a sense explained below. Its initial non-terminal is $\langle \sigma, q_0\rangle$, and every production rule of $\mathcal{G}_\sigma$ is either of the form: $\langle \tau, q\rangle \xrightarrow{w} a$ such that $q \xrightarrow{w} a \in R$ and τ is a singleton sequence, or $\langle \tau, q\rangle \xrightarrow{w} \langle \tau_1, q_1\rangle, \ldots, \langle \tau_p, q_p\rangle$ such that $q \xrightarrow{w} q_1, \ldots, q_p \in R$ and $\tau_1, \ldots, \tau_p$ is a partition of τ in p parts of equal length, where the length of a sequence of positive rational numbers is the sum of its elements.

The only tricky point for partitioning τ in p parts is that it may require to split some rational number r in two parts r_1 and r_2, such that $r = r_1 + r_2$, where r_1 will be the last element of some τ_i and r_2 will be the first element of τ_{i+1} (necessarily a continuation). For instance, the partition of $[\frac{1}{2}, \frac{1}{6}, \frac{1}{3}]$ in two parts is $[\frac{1}{2}]$, $[\frac{1}{6}, \frac{1}{3}]$, but the partition of $[\frac{1}{6}, \frac{1}{3}]$ in two parts of equal length is $[\frac{1}{6}, \frac{1}{12}]$, $[\frac{1}{4}]$, and in this partition, $\frac{1}{4}$ has to be a continuation since the duration $\frac{1}{3}$ has been cut into $\frac{1}{12} + \frac{1}{4}$.

Let us state this precisely. We consider non-empty sequences of positive rational numbers with a sign: $-\tau$ means that the first element of the sequence τ is a continuation and $+\tau$ means that it is not (the sign $+$ may be omitted). Now we define the concatenation $\odot$ of signed sequences of positive rational numbers by:

$$\delta[r_1, \ldots, r_n] \odot +[s_1, \ldots, s_m] = \delta[r_1, \ldots, r_n, s_1, \ldots, s_m]$$
$$\delta[r_1, \ldots, r_n] \odot -[s_1, \ldots, s_m] = \delta[r_1, \ldots, r_n + s_1, s_2, \ldots, s_m]$$

where δ is $+$ or $-$ and $n, m \geq 1$.

The *length* of a signed sequence $\sigma = \delta[r_1, \ldots, r_n]$ of rational numbers is $\sum_{i=1}^{n} r_i$, denoted $\|\sigma\|$. A p-partition of a signed sequence τ (for $p > 0$) is a sequence $\tau_1, \ldots, \tau_p$ of signed sequences such that $\tau_1 \odot \ldots \odot \tau_p = \tau$ and $\|\tau_1\| = \ldots = \|\tau_p\| = \frac{\|\tau\|}{p}$. Note that it is unique for a given couple (τ, p).

Example 7 $+[\frac{1}{2}, \frac{1}{6}, \frac{1}{3}] = +[\frac{1}{2}] \odot +[\frac{1}{6}, \frac{1}{3}]$ *and* $+[\frac{1}{6}, \frac{1}{3}] = +[\frac{1}{6}, \frac{1}{12}] \odot -[\frac{1}{4}]$.
These two concatenation are 2*-partitions.*

Now we can describe precisely the construction of $\mathcal{G}_\sigma$ from the RG $\mathcal{G}$ and the sequence σ. Every non-terminal of $\mathcal{G}_\sigma$ is a pair made of a signed sequence τ of positive rational numbers and a non-terminal q of $\mathcal{G}$, denoted by $\langle \tau, q\rangle$.

1. $\mathcal{G}_\sigma$ contains the non-terminal $\langle +\sigma, q_0\rangle$ (initial).

2. For every non-terminal $\langle \tau, q\rangle$ of $\mathcal{G}_\sigma$, and every production rule $q \xrightarrow{w} q_1, \ldots, q_p$ of $\mathcal{G}$, $\mathcal{G}_\sigma$ contains the production rule $\langle \tau, q\rangle \xrightarrow{w} \langle \tau_1, q_1\rangle, \ldots, \langle \tau_p, q_p\rangle$ such that $\tau_1, \ldots, \tau_p$ is the p-partition of τ, and $\mathcal{G}_\sigma$ contains the non-terminals $\langle \tau_1, q_1\rangle, \ldots, \langle \tau_p, q_p\rangle$.

3. For every singleton non-terminal $\langle +[r], q\rangle$ of $\mathcal{G}_\sigma$ ($r \in \mathbb{Q}_+$) and every production rule $q \xrightarrow{w} \mathsf{n}$ of $\mathcal{G}$, $\mathcal{G}_\sigma$ contains the production rule $\langle +[r], q\rangle \xrightarrow{w} \mathsf{n}$.

4. For every singleton non-terminal $\langle -[r], q\rangle$ of $\mathcal{G}_\sigma$ and every production rule $q \xrightarrow{w} \_$ of $\mathcal{G}$, $\mathcal{G}_\sigma$ contains the production rule $\langle -[r], q\rangle \xrightarrow{w} \_$.

The size of $\mathcal{G}_\sigma$ is at most the size of $\mathcal{G}$ times the size of σ. The correctness of construction follows from the property that: $\mathcal{L}_{\langle\tau,q\rangle} = \{t \in \mathcal{L}_q \mid val(t) = |\tau|\}$, where $|\tau|$ denotes the absolute value of τ, *i.e.* the sequence τ without its sign.

4.2 Examples

Example 8 *Let us consider the application of the above procedure to the WRG $\mathcal{G}$ of Example 4 and the rhythmic value $\sigma = [\frac{1}{2}, \frac{1}{6}, \frac{1}{3}]$.*

The initial non-terminal of $\mathcal{G}_\sigma$ is $\langle\sigma, q_0\rangle$ (we omit the $+$ sign).

From the production rules of $\mathcal{G}$ starting with q_0, and 2- and 3-partitions of σ, we obtain

$$\langle\sigma, q_0\rangle \xrightarrow{0.35} \langle[\tfrac{1}{2}], q_2\rangle, \langle[\tfrac{1}{6}, \tfrac{1}{3}], q_2\rangle$$
$$\langle\sigma, q_0\rangle \xrightarrow{0.45} \langle[\tfrac{1}{3}], q_3\rangle, \langle-[\tfrac{1}{6}, \tfrac{1}{6}], q_3\rangle, \langle[\tfrac{1}{3}], q_3\rangle$$

From the new non-terminals with singleton IOIs, we have

$$\langle[\tfrac{1}{2}], q_2\rangle \xrightarrow{0.1} \mathsf{n} \qquad \langle[\tfrac{1}{3}], q_3\rangle \xrightarrow{0.1} \mathsf{n}$$

From the new non-terminal $\langle[\frac{1}{6}, \frac{1}{3}], q_2\rangle$, we have:

$$\langle[\tfrac{1}{6}, \tfrac{1}{3}], q_2\rangle \xrightarrow{0.5} \langle[\tfrac{1}{6}, \tfrac{1}{12}], q_4\rangle, \langle-[\tfrac{1}{4}], q_4\rangle$$
$$\langle[\tfrac{1}{6}, \tfrac{1}{3}], q_2\rangle \xrightarrow{0.6} \langle[\tfrac{1}{6}], q_5\rangle, \langle[\tfrac{1}{6}], q_5\rangle, \langle-[\tfrac{1}{6}], q_5\rangle$$

and then:

$$\langle[\tfrac{1}{6}, \tfrac{1}{12}], q_4\rangle \xrightarrow{0.75} \langle[\tfrac{1}{12}], q_5\rangle, \langle-[\tfrac{1}{12}], q_5\rangle, \langle[\tfrac{1}{12}], q_5\rangle$$

From the other non-singleton non-terminal $\langle-[\frac{1}{6}, \frac{1}{6}], q_3\rangle$ we have:

$$\langle-[\tfrac{1}{6}, \tfrac{1}{6}], q_3\rangle \xrightarrow{0.5} \langle-[\tfrac{1}{6}], q_5\rangle, \langle[\tfrac{1}{6}], q_5\rangle$$

Finally, from the remaining singleton non-terminals:

$$\langle[\tfrac{1}{12}], q_5\rangle \xrightarrow{0.1} \mathsf{n} \qquad \langle-[\tfrac{1}{12}], q_5\rangle \xrightarrow{0.2} \_$$
$$\langle-[\tfrac{1}{4}], q_4\rangle \xrightarrow{0.2} \_ \qquad \langle[\tfrac{1}{6}], q_5\rangle \xrightarrow{0.1} \mathsf{n}$$
$$\langle-[\tfrac{1}{6}], q_5\rangle \xrightarrow{0.2} \_$$

The language of this grammar contains the three RTs displayed in Figure 1, with the weights listed in Example 6.

Example 9 *With a more elaborate WRG $\mathcal{G}$ (10 non-terminals and 55 production rules) and the same σ as in Example 8, we obtain a WRG $\mathcal{G}_\sigma$ with 73 non-terminals and 79 production rules, whose language contains the 7 RTs displayed Figure 3 (with or without dots).*

One can notice in the examples that some of the notations proposed do not seem to be generated by the grammars described. For instance, in the middle example in Figure 1, the notation displayed does not match exactly the structure of the RT above: the matching notation should indeed be:

For the sake of readability, we grouped the ties between equal note values. For instance, in the previous example, we grouped the last two linked sixteenth notes into one eighth note. This allows us to display simpler, more idiomatic and more compact solutions. This simplification step is performed on the Lilypond translation of RTs generated by the grammars described, as a post-processing step, by using simple rewriting rules.

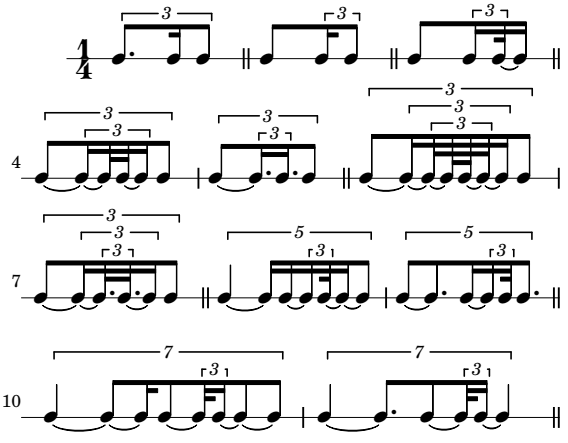

Figure 3. Enumeration for $\sigma = [\frac{1}{2}, \frac{1}{6}, \frac{1}{3}]$ and a more elaborate WRG than in Example 4. The RTs are separated by double bars. Single bar separate two version with or without dots of the same RT.

4.3 RT Rewrite Rules

In [17, 18] we have proposed systems of rules for rewriting RTs into equivalent RTs. This includes rules such as $2(\_, \_) \to \_$ or $2(\mathsf{n}, \_) \to \mathsf{n}$ or $3\big(2(x_1, x_2), 2(x_3, x_4), 2(x_5, x_6)\big) \to 2\big(3(x_1, x_2, x_3), 3(x_4, x_5, x_6)\big)$.
We have considered using RT rewriting for the exploration of equivalent rhythm notations. However, we gave up due to the complexity of this syntactic approach. Rewrite rules as above can indeed be applied at any node of a RT, as long as the subtree at this node matches the left-hand-side of the rule. This gives generally, for a given non trivial RT, many choices of rewrite positions and induces a high divergence in the application of rewrite rules. To be applicable, some restrictive rewrite strategy must be considered and this strategies has to be proven complete.

In comparison, we found the above semantic approach, based on rhythmic values, dramatically less complex.

5. POLYRHYTHMS

We present an application of our approach to obtain good notations for polyrhythms, following studies in [19]. Polyrhythms occurs when two or more rhythmic figures that are not perceived as deriving directly one from the other, or as simple manifestations of the same meter are played simultaneously [20]. In our context, it can be seen as having simultaneously two RTs that do not share the same root-level arities.

Merging two RTs that have different arities is not a trivial task when considering only the trees. However, merging two rhythms is trivial when considering only their onsets: we only have to merge the two onset lists into one list, and order onsets from the smaller to the greater.

To notate polyrhythms, we thus go around the problem of merging trees by going through the time domain. We first convert the two RTs into their rhythmic values, from which we obtain the corresponding onset lists. We then merge the two onset lists into one onset list, from which we get a

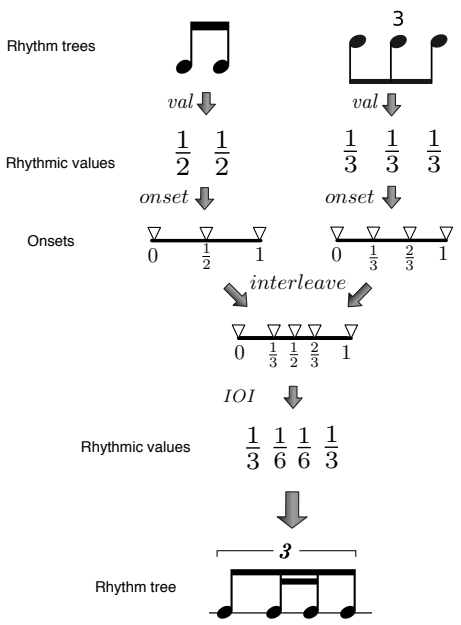

Figure 4. Workflow to obtain notations corresponding to a polyrhythm.

merged rhythmic value (as an IOI sequence). We then enumerate the equivalent notations of this merged rhythmic value to obtain RTs corresponding to the merged polyrhythm. A schema of this workflow can be found in Figure 4.

The rhythmic value obtained by merging two rhythmic values σ_1 and σ_2 will be denoted as $\sigma_1 \parallel \sigma_2$. Intuitively, $\sigma_1 \parallel \sigma_2$ corresponds to the sequence of durations obtained by playing both σ_1 and σ_2 at the same time.
Let us reformulate the problem of interest :

given a weighted rhythm grammar $\mathcal{G}$ and two non-empty sequences of positive rational numbers σ_1 and σ_2,

return a weighted rhythm grammar $\mathcal{G}_\sigma$ such that $\mathcal{L}(\mathcal{G}_\sigma) = \{t \in \mathcal{L}(\mathcal{G}) \mid val(t) = \sigma_1 \parallel \sigma_2\}$.

Therefore, this problem is very similar to the one we addressed in Section 4 and can be solved with the same procedure.

Example 10 *Figure 5 presents a list of rhythms obtained for* $[\frac{1}{2}, \frac{1}{2}] \parallel [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$*, using the complex WRG mentionned in Example 9.*

All we need to do now is to define the $\parallel$ operator. To a rhythmic value $\sigma = [d_1, \ldots, d_p]$ we associate a sequence of onsets $onset(\sigma) = [o_1, \ldots, o_{p+1}]$ defined by $o_i = \sum_{j=1}^{i-1} d_j$ for all $0 < i \leq p + 1$. Note that $o_1 = 0$.

Example 11 *If* $\sigma = [\frac{1}{2}, \frac{1}{6}, \frac{1}{3}]$*, then* $onset(\sigma) = [0, \frac{1}{2}, \frac{2}{3}, 1]$.

The inverse transformation, called IOI, associates to a sequence of onsets $\ell = [o_1, \ldots, o_{p+1}]$ the rhythmic value $\sigma = [d_1, \ldots, d_p]$ defined by $d_i = o_{i+1} - o_i$ for all $1 \leq i \leq p$. Finally,

$$\sigma_1 \parallel \sigma_2 = IOI\big(onset(\sigma_1) \cup onset(\sigma_2)\big).$$

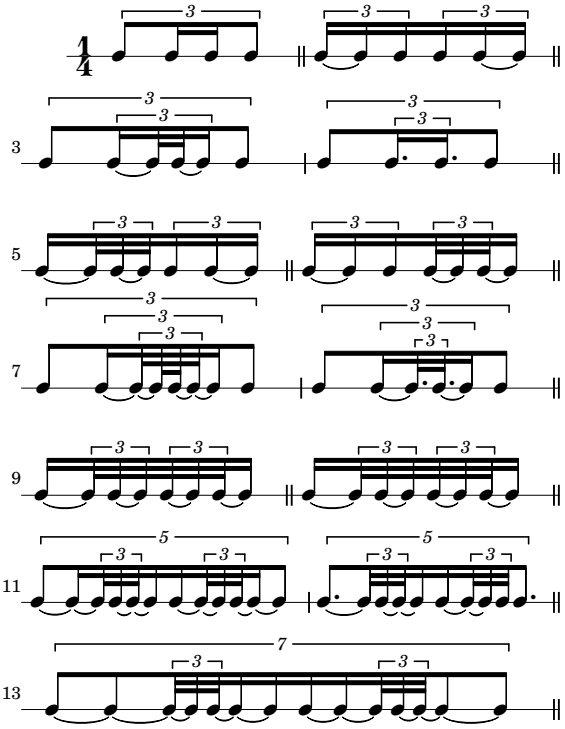

Figure 5. RTs for the merge of $[\frac{1}{2}, \frac{1}{2}]$ and $[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$.

Here, the operator $\cup$ denotes the interleaving of sequences of onsets, performed as expected to respect the ordering of onset values.

During the above processing, a special care must be given to the handling of the events associated to the initial rhythmic values σ_1 and σ_2, in order to reassign them properly in $\sigma_1 \parallel \sigma_2$ (see Section 2.3). We leave these details out of this paper.

Example 12 *Figure 6 shows examples of rhythms obtained for* $\sigma_1 \parallel \sigma_2$ *with* $\sigma_1 = [\frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}, \frac{1}{5}]$ *and* $\sigma_2 = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$ *corresponding respectively to the left and right hand of a part of bar of Chopin Nocturne No3 op.9.*

Note that in principle, we could merge two rhythmic values that are not the same length. Nevertheless, here, a rhythmic value is always normalized to have a total duration of 1.

6. CONCLUSION

We proposed a formalism based on formal grammars to define languages of weighted rhythm trees, and a procedure to lazily enumerate these trees by ascending weight. It is applied, via grammar constructions, to the problem of enumerating rhythm trees defining the same given sequence of durations (IOIs), and enumerating merges of two polyrhythmic voices into one.

This has been implemented in C++ [1] , with command line prototypes outputing the enumerations of RTs which were then translated into a graphical representation in Lilypond.

[1] `http://qparse.gforge.inria.fr`

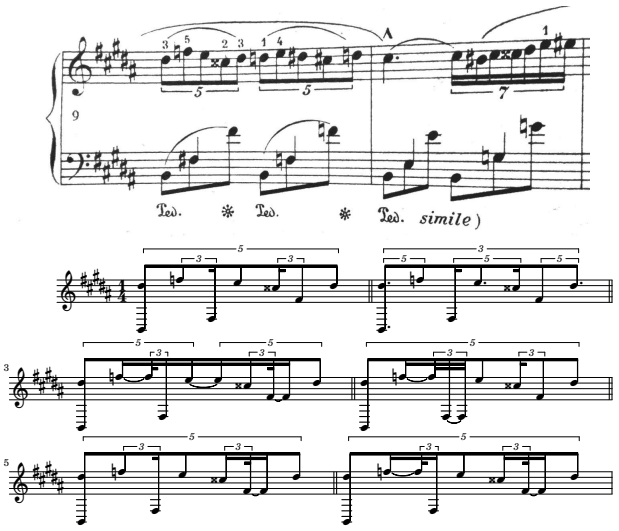

Figure 6. Chopin Nocturne No 3 op. 9 and 6 alternative notations merging both hands for 1st half of bar 9.

The Lilypond code was moreover post-processed for the improvements described at the end of Section 4.2.

After evaluation with this proof-of-concept prototype, this implementation will be integrated as a dynamic library in OpenMusic, for use in particular as a backend procedure for the transcription framework of [8].

Acknowledgments

The authors would like to thank Karim Haddad for his valuable knowledge and advices on rhythm and notation.

7. REFERENCES

[1] A. Klapuri *et al.*, "Musical meter estimation and music transcription," in *Cambridge Music Processing Colloquium*, 2003, pp. 40–45.

[2] J. Bresson, C. Agon, and G. Assayag, "OpenMusic: visual programming environment for music composition, analysis and research," in *Proc. of the 19th ACM Int. Conf. on Multimedia*. ACM, 2011, pp. 743–746.

[3] A. T. Cemgil, P. Desain, and B. Kappen, "Rhythm quantization for transcription," *Computer Music Journal*, vol. 24, no. 2, pp. 60–76, 2000.

[4] H. C. Longuet-Higgins and C. S. Lee, "The rhythmic interpretation of monophonic music," *Music Perception: An Interdisciplinary Journal*, vol. 1, no. 4, pp. 424–441, 1984.

[5] J. Pressing and P. Lawrence, "Transcribe: A comprehensive autotranscription program," in *International Computer Music Conference Proceedings (ICMC)*, 1993, pp. 343–345.

[6] C. Agon, G. Assayag, J. Fineberg, and C. Rueda, "Kant: a critique of pure quantification," in *International Computer Music Conference Proceedings (ICMC)*, 1994, pp. 52–59.

[7] C. S. Lee, "The rhythmic interpretation of simple musical sequences: Towards a perceptual model," *Musical Structure and Cognition*, vol. 3, pp. 53–69, 1985.

[8] A. Ycart, F. Jacquemard, J. Bresson, and S. Staworko, "A Supervised Approach for Rhythm Transcription Based on Tree Series Enumeration," in *International Computer Music Conference Proc. (ICMC)*, 2016.

[9] M. Laurson, "Patchwork: A visual programming language and some musical applications," Sibelius Academy, Helsinki, Tech. Rep., 1996.

[10] C. Agon, K. Haddad, and G. Assayag, "Representation and rendering of rhythm structures," in *Proceedings 2d Int. Conf. on Web Delivering of Music*, 2002.

[11] M. Laurson, M. Kuuskankare, and V. Norilo, "An overview of PWGL, a visual programming environment for music," *Computer Music Journal*, vol. 33, no. 1, pp. 19–31, Mar. 2009.

[12] L. Huang, "Advanced dynamic programming in semiring and hypergraph frameworks," in *In COLING Tutorial*, 2008.

[13] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, C. Löding, D. Lugiez, S. Tison, and M. Tommasi, *Tree Automata Techniques and Applications*. http://tata.gforge.inria.fr, 2007.

[14] I. Shmulevich and D.-J. Povel, "Complexity measures of musical rhythms," *Rhythm perception and production*, pp. 239–244, 2000.

[15] C. Agon, G. Assayag, J. Fineberg, and C. Rueda, "Kant: A critique of pure quantification," in *Proc. of ICMC*, Aarhus, Denmark, 1994, pp. 52–9.

[16] L. Huang and D. Chiang, "Better k-best parsing," in *Proc. of the 9th Int. Workshop on Parsing Technology*. Association for Comp. Linguistics, 2005, pp. 53–64.

[17] P. Donat-Bouillud, F. Jacquemard, and M. Sakai, "Towards an Equational Theory of Rhythm Notation," in *Music Encoding Conference* May 2015.

[18] F. Jacquemard, P. Donat-Bouillud, and J. Bresson, "A Structural Theory of Rhythm Notation based on Tree Representations and Term Rewriting," in *Proc. 5th Int, Conf. on Mathematics and Computation in Music (MCM)*, Springer LNAI vol. 9110, 2015.

[19] K. Haddad, "Fragments de recherche et d'expérimentation: Eléments de réflexions autour de l'écriture rythmique d'Emmanuel Nunes," *seminaire Mamux*, Ircam, 2012.

[20] D. M. Randel, *The Harvard dictionary of music*. Harvard University Press, 2003, vol. 16.