IRCAM
Paris
Equipe *Repmus*

Encadrants :
Jean BRESSON
Florent JACQUEMARD

Transcription rythmique dans OpenMusic

Pierre DONAT-BOUILLUD

Du 13 Mai au 19 Juillet 2013

Résumé : Il est présenté une nouvelle approche pour la transcription rythmique, en ne prenant en compte que l'étape de quantification, à l'aide de transformations d'arbre de rythme, dans l'environnement de composition assisté par ordinateur *OpenMusic*. Les arbres de rythme, puis une notation appelée *arbre de rythme symbolique* sont décrits. Ensuite, il est expliqué comment générer des arbres puis les transformer à l'aide de règles de réécriture, à partir des données temporelles d'un morceau de musique. Enfin, l'approche par arbre de rythme et l'approche par sélection de grilles d'*OpenMusic* sont comparées.

Mots clés : *Musique, Arbres, Règles de réécriture*

Je remercie mes encadrants de stage, Jean BRESSON et Florent JACQUEMARD pour leurs conseils et leur implication dans la direction de mon stage et la relecture du rapport, Adrien MAIRE pour les idées et les critiques, mes parents, pour m'avoir hébergé le temps du stage, les stagiaires, les chercheurs et les compositeurs de l'IRCAM avec qui j'ai pu avoir de passionnantes discussions.

Table des matières

Introduction

Depuis Pythagore, la musique est liée aux mathématiques ; elle fait partie des quatre sciences mathématiques à l'Antiquité, le *quadrivium* de la théorie antique, avec l'arithmétique, la géométrie, et l'astronomie. C'est le début de la formalisation. Plus tard, pendant la période classique, Rameau théorise l'harmonie [1], et des règles combinatoires régissent le contrepoint [2].

Au XX^{e} siècle, de nouvelles théories et de nouveaux outils apparaissent, que l'informatique va soutenir et amplifier : musique stochastique, musique algorithmique ... L'ordinateur permet d'explorer des idées musicales, de générer des sons, des notes, des rythmes à partir de processus de calcul, d'analyser la musique. Il permet aussi de noter la musique.

Pierre Boulez, compositeur contemporain tenant du sérialisme généralisé, né en 1 925, fonde l'Ircam, Institut de recherche et coordination en acoustique/musique, en 1 969, dans le cadre du centre Pompidou. Cet institut singulier met en relation des compositeurs, des chercheurs en informatique, en acoustique, en psychologie cognitive, et organise des concerts de musique contemporaine.

Entre autres projets de recherche, l'Ircam développe *OpenMusic* [1], un environnement de composition assisté par ordinateur.

Le travail effectué dans le cadre de mon stage porte sur la transcription rythmique, dans *OpenMusic* : il s'agit à partir d'une séquence de notes représentées par des durées de générer une partition qui comporte des rythmes en notation musicale traditionnelle, en explorant une nouvelle approche, où la transcription s'opère par des transformations sur des arbres de rythme.

Pour simplifier, on suppose que le morceau a déjà été segmenté en régions de même tempo, et qu'on connaît déjà le tempo de chacune des régions. Il reste à effectuer la quantification des durées, sur chacune des régions, pour les transformer en valeurs rythmiques.

1. Agencement vertical de la musique (domaine des hauteurs).
2. Agencement horizontal de la musique (domaine temporel).

1. Contexte

1.1. OpenMusic

Le travail a eu lieu dans l'environnement de composition assisté par ordinateur *OpenMusic*. *OpenMusic* consiste en un langage de programmation visuel basé sur *Common Lisp* et des bibliothèques spécialisées dans le traitement de la musique, pour faire de la composition par contraintes, de la composition aléatoire, de la synthèse sonore, de la musique spectrale, par exemple. C'est un langage fonctionnel, dans lequel les informations musicales sont traitées hiérarchiquement. De nouvelles composantes du langage graphique peuvent être créées en *Common Lisp* ou directement dans le langage graphique.

Il a été développé par l'Ircam depuis 1998, et est utilisé principalement par des compositeurs contemporains [2] et des musicologues, même s'il peut servir comme environnement générique de programmation graphique [3].

Les œuvres (ou les programmes) se construisent dans un *patch* (voir figure 1.1), dans lequel des boites qui possèdent des entrées et des sorties sont connectées entre elles.

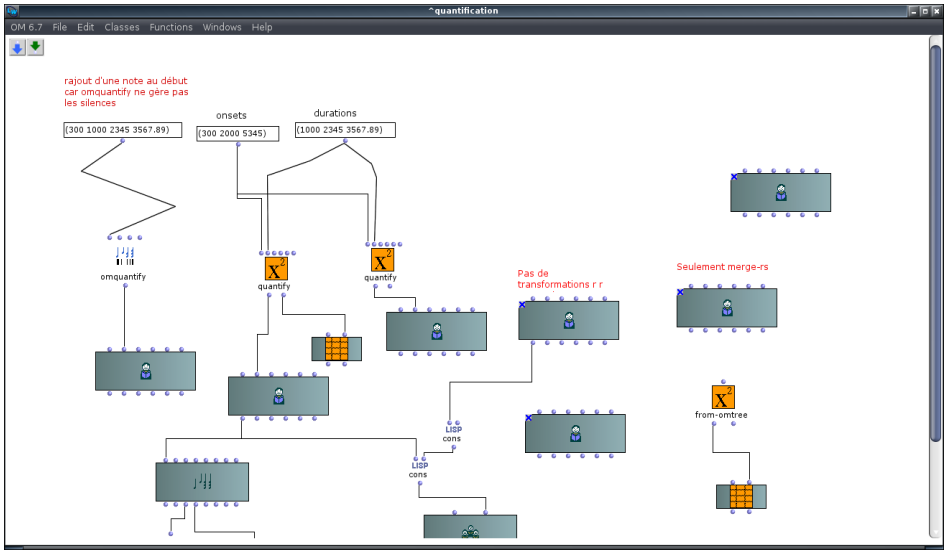

FIGURE 1.1. — Un patch, unité de base d'un programme dans *OpenMusic*

1.2. La transcription rythmique

1.2.1. Le rythme

En musique, le rythme est la perception d'une structure dans un ensemble d'événements sonores, habituellement, des notes. Il opère une division de la durée, et ces divisions interviennent à une certaine fréquence qu'on appelle tempo. Les divisions forment, en les rassemblant, des valeurs rythmiques ♩ ♪ ♪ ♩ ... Ces trois paramètres constituent ce qui est appelé en musique le rythme [4].

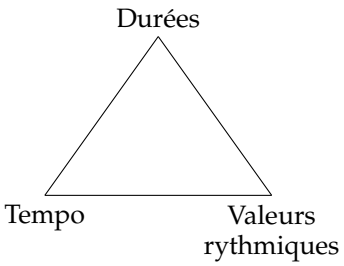

FIGURE 1.2. — Relation triadique du rythme selon [4]

Les événements sonores sont d'abord décrits par des durées, qu'on notera sous forme de deux n-uplets d'entiers (millisecondes), l'un correspondant aux attaques [1], l'autre, à leurs durées [2]. La suite de notes représentée en figure 1.3 pourrait ainsi être notée :

Attaques : $(0, 1403, 2611, 3778)$

Durées : $(1200, 700, 1000, 1000)$

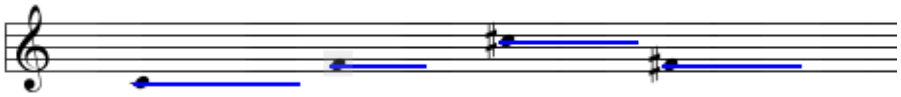

FIGURE 1.3. — Vue en durée d'une séquence de notes dans *OpenMusic*

1. *onsets* des notes en anglais.
2. Il n'y a besoin que des durées, ou que des attaques, pour définir la suite d'événements, si les chevauchements de note ne sont pas rajoutés, c'est-à-dire pas de polyphonie, et que la convention qu'une durée négative représente un silence est rajoutée, par exemple. Pour plus de clarté, les deux seront donnés.

1.2.2. Arbres de rythme

Le rythme est une structure hiérarchique : un morceau est divisé en mesures, qui comportent par exemple quatre noires, elles-même divisées chacune en deux croches ... Il est donc pertinent de représenter un rythme par un arbre.

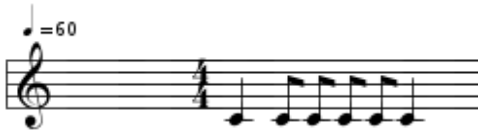

FIGURE 1.4. — Un rythme simple noté sur portée.

La figure 1.4 pourrait ainsi correspondre à l'arbre de la figure 1.5.

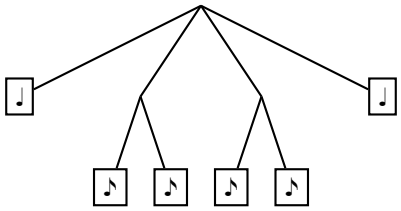

FIGURE 1.5. — Le rythme, une structure arborescente.

Dans *OpenMusic*, les arbres de rythmes sont notés sous forme de listes, qui contiennent des rapports de durées et des subdivisions[5]. Le chiffrage de la mesure, c'est-à-dire l'unité de base des divisions pour le rythme, et le nombre de ces divisions par mesure, sont intégrés à la notation. La figure 1.6 présente un arbre de rythme d'*OpenMusic* et le rythme correspondant en notation sur portée.

Dans cette notation, les notes sont représentées par des nombres positifs, entiers ou rationnels, les silences, par des nombres entiers ou rationnels négatifs ; les notes liées au symbole précédent sont des flottants.

Formalisation Dans la suite, on ne prendra pas en compte l'unité de chiffrage de la mesure, et ainsi, une mesure à $\frac{n}{4}$ sera représentée comme une mesure à $\frac{n}{2}$.

Notations 1. *On note f_k^a où $k \in \mathbb{N}$ et $a \in \{-1, 1\}$ une note ou un silence et $\mathcal{F}$ l'ensemble des notes ou silences.*

On note s_k^a où $k \in \mathbb{N}$ et $a \in \{-1, 1\}$ une liaison et $\mathcal{S}$ l'ensemble des liaisons.

On pose $\mathcal{P}_0 = \mathcal{F} \cup \mathcal{S}$. Il s'agit des symboles d'arité 0.

On note $\mathcal{X}$ l'ensemble des variables.

On note $p_{k,n}$ un symbole d'arité n, où $n \geq 0$, qui représente la durée k, et $\mathcal{P}_n$ l'ensemble de ces symboles d'arité n.

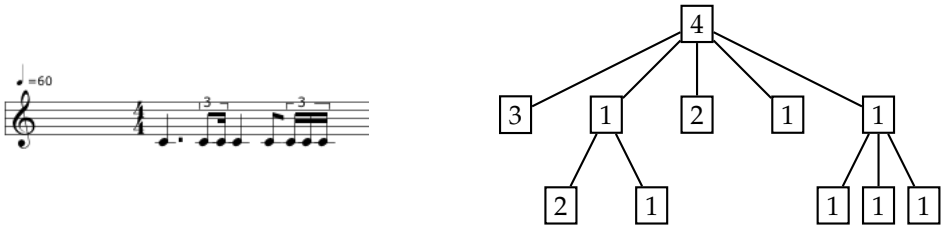

FIGURE 1.6. — L'arbre de rythme `(? (((4 4) ( 3 (1 ( 2 1)) 2 1 (1 ( 1 1 1)) ))))`

L'arbre de rythme de la figure 1.6 se note alors :

$$p_{4,5}\left[f_3^1,\ p_{1,2}\left(f_2^1, f_1^1\right),\ f_2^1,\ f_1^1,\ p_{1,1}\left(f_1^1,\ f_1^1,\ f_1^1\right)\right]$$

Quand la distinction entre notes et silences, et liaisons n'est pas nécessaire, la notation sera simplifiée en utilisant des $p_{k,0}$.

Définition 1 (Arbres de rythmes d'*OpenMusic*)**.** *L'ensemble des arbres de rythme d'*OpenMusic $\mathcal{L}_{\mathcal{OM}}$ *est le langage d'arbre défini par la donnée des termes construits sur* $\{\mathcal{P}_n\}_{n\in\mathbb{N}}$.

Pour rappel :
*Le langage des arbres de rythme d'*OpenMusic $\mathcal{L}_{\mathcal{OM}}$ *est alors défini comme le plus petit langage tel que :*
- $\mathcal{P}_0 \subset \mathcal{L}_{\mathcal{OM}}$
- *Si* $n \geq 1$, *si* $p \in \mathcal{P}_n$, *et si* $(t_1, \cdots, t_n) \in (\mathcal{L}_{\mathcal{OM}})^n$ *alors* $p(t_1, \cdots, t_n) \in \mathcal{L}_{\mathcal{OM}}$.

Forme normale La notation des arbres de rythme d'*OpenMusic* est ambiguë.
Par exemple, `(? (((4 4) ( 3 (1 ( 2 1)) 2 1 (1 ( 1 1 1)) ))))` et `(? (((4 4) ( 3 (1 ( 6 3)) 2 1 (1 ( 2 2 2)) ))))` dénotent les mêmes rythmes : la multiplication par une constante de rapports frères ne change pas les rapports.

Définition 2 (Relation d'équivalence sur les arbres de rythme)**.** *Soit* $(k, n) \in \mathbb{N}^2$. *Soit* $r \in [\![1, n]\!]$. *Soit* $(i_1, \cdots, i_n) \in [\![1, r]\!]$ *et* $(j_1, \cdots, j_n) \in [\![1, r]\!]$.
Soit $(x_{m,q})_{(m,n)\in[\![1,r]\!]\times[\![1,j_m]\!]}$
On obtient alors un nœud d'un arbre de rythme :

$$p_{k,n}\left[p_{i_1,j_1}(x_{1,1}, \cdots, x_{1,j_1}), \cdots, p_{i_r,j_r}(x_{r,1}, \cdots, x_{r,j_r})\right]$$

On définit de même :

$$p'_{k',n'}\left[p'_{i'_1,j'_1}(x'_{1,1}, \cdots, x'_{1,j'_1}), \cdots, p'_{i'_{r'},j'_{r'}}(x'_{r',1}, \cdots, x'_{r',j'_{r'}})\right]$$

Ces deux nœuds sont équivalents (noté $\sim$ *) si et seulement si :*

– $\forall (m,q) \in [\![1,r]\!] \times [\![1,j_m]\!]$, $x_{m,q} \sim x'_{m,q}$
– $n = n\prime$
– $\forall m \in [\![1,r]\!]$, $j_m \mid j'_m$ *ou* $j'_m \mid j_m$

Exemple :
`(? (((4 4) ( 3 (1 ( 2 1)) 2 1 (1 ( 1 1 1)) ))))` et
`(? (((4 4) ( 3 (1 ( 6 3)) 2 1 (1 ( 2 2 2)) ))))` notent le même rythme.

Un représentant canonique d'un arbre de rythme donné peut alors être obtenu en divisant les rapports de durée à un même niveau par leur plus grand diviseur commun.

1.2.3. La quantification

Les attaques et les durées, données au départ, sont une donnée continue, qu'il faut discrétiser pour les placer sur la grille des subdivisions.

Définition 3 (Quantification)**.** *La quantification est le procédé qui permet d'approcher un signal continu (ou à valeurs dans un ensemble discret de grande taille) par les valeurs d'un ensemble discret d'assez petite taille.*

Le cas de la quantification rythmique :

Valeurs dans un ensemble discret de grande taille début des notes et durée des notes (dans $\mathbb{R}$) ;

Ensemble discret d'assez petite taille les rythmes, division rationnelle du temps, ♩, ♪, 𝅗𝅥, ♫, 𝅝 … organisés sous la forme d'un arbre de rythme.

Il faudra aussi déterminer le tempo du morceau pour effectuer la quantification [3].

Applications La quantification est utilisée pour caler sur des subdivisions une interprétation à l'aide d'un clavier MIDI [4], ou bien pour obtenir une partition à partir d'un enregistrement, avec un microphone, et dans ce dernier cas, il faudra aussi une quantification des hauteurs mélodiques. Elle peut aussi être outil de composition. Certains compositeurs de l'Ircam [2] génèrent ainsi une suite de durées à partir d'équations, comme les équations de Navier-Stokes [5], puis quantifient pour obtenir des rythmes plus simples ; d'autres enregistrent le rebond d'une bille et quantifient l'enregistrement. L'objectif est d'obtenir une partition en notation musicale traditionnelle, qui soit lisible et jouable par un instrumentiste humain.

3. Voir la section 1.2.1.
4. Musical Instrument Digital Interface : protocole de communication et de commande pour instruments et dispositifs de commande électronique dédiés à la musique, dont les ordinateurs.
5. Les équations qui régissent la mécanique des fluide.

1.2.4. Revue des approches existantes pour la quantification rythmique

Quantification naïve sur grille Une première façon de quantifier est de générer une grille de subdivisions régulières, puis de caler les attaques et les durées sur la grille.

Cette approche atteint très vite ses limites face à des rythmes complexes, ou même dès que la subdivision principale choisie ne correspond pas à la subdivision principale du morceau, par exemple, si une subdivision binaire[6] est choisie alors que le morceau est ternaire[7].

Approche heuristique Le principe consiste à tester un certain nombre de grilles, pas forcément régulières, à leur attribuer une note, avec une mesure d'erreurs, et à choisir la grille qui a la meilleure note. Dans le cas du quantificateur *Kant* [6] utilisé il y a quelques années dans *OpenMusic*, trois mesures, des distances entre les attaques données et leur alignement sur la grille, sont utilisées pour choisir entre les grilles :

- somme des rapports cubiques (attaque donnée - attaque sur la grille)
- mesure de simplicité d'une division (du plus simple au plus compliqué) ; 1,2,4,3,6,8,5,7, etc
- distance euclidienne entre attaque donnée et attaque sur la grille

La distance euclidienne permet d'arbitrer entre les deux premières distances.

Approche bayésienne L'approche bayésienne [7] apprend le style d'un compositeur ou d'une époque pour quantifier, et donc explicite les présupposés stylistiques des méthodes précédentes.

Elle se fonde sur la formule de Bayes :

$$p(B|A) = \frac{p(A|B)p(B)}{\sum_i (A|B = bi)p(B = bi)} \tag{1.1}$$

p étant une probabilité.

La résolution de la quantification fait alors apparaître une mesure de la complexité de la partition fondée sur la longueur de son code de Shannon, une pénalisation des tempos rapides, et une mesure d'erreur grâce à la distance de Mahalonabis.

Cette approche permet d'obtenir de bons résultats si le style de la source est connu, par exemple, de la musique classique. Cependant, dans le cas de l'aide à la composition, le style évolue au cours de l'acte créateur du compositeur, et cette approche a surtout été développée pour des entrées sur clavier MIDI et devient trop rigide.

En pratique, dans *OpenMusic* Le quantificateur *Kant* est amélioré par Benoît Meudic sous la forme d'une bibliothèque *OpenMusic*, *OMKant* [8] : des fonctionnalités de détection du tempo, de détection de régions, sont ajoutées, ainsi qu'un mode interactif.

6. Les temps sont divisés en deux.
7. Les temps sont divisés en trois.

Le quantificateur actuel, `omquantify` est basé sur le quantificateur de *Patchwork*, l'ancêtre d'*OpenMusic*, et utilise le même principe de sélection entre plusieurs grilles. Il ne détecte pas le tempo qui doit être rentré en paramètre, et ne gère pas bien les silences.

2. Transcription par réécriture

Une nouvelle approche pour la transcription rythmique est envisagée : effectuer des transformations sur les arbres de rythme.

Dans *OMKant* ou dans `omquantify`, le résultat d'une quantification est ultimement converti en arbres de rythme, qui constituent une structure de donnée de base d'*OpenMusic*. L'idée ici est de faire intervenir les arbres de rythmes plus tôt dans le calcul, et de les transformer par réécriture.

2.1. Arbres symboliques de rythme

Une infinité d'arbres de rythme peuvent représenter le même rythme, même s'il est possible de choisir un représentant « plus simple »(voir 1.2.2). Les arbres de rythme utilisent une infinité d'étiquettes (des nombres, dans $\mathbb{N}$). Cette notation est plus adaptée à une manipulation arithmétique que symbolique.

Une nouvelle notation a été mise au point, inspirée de la notation utilisée par David Rizo dans [9].

La nouvelle notation utilise un petit nombre de symboles :

n : note

r : silence

s : liaison avec la note précédente [1]

$=$: le premier symbole différent de = dure "1+ le nombre de = précédents adjacents" [2] (voir figure 2.1).

Chacun des fils d'un nœud a une durée égale.

L'arbre de la figure 1.6 correspond alors à la figure 2.2.

Notations 2. *On note $\widehat{\mathcal{L}}$ l'ensemble $\{n, r, s, =\}$. Il s'agit des symboles d'arité 0.*
On note $\mathcal{N} = \{h_n / \ n \in \mathbb{N}^*\}$ où h_n est un symbole d'arité n.

Définition 4 (Arbres de rythme symboliques)**.** *Le langage des arbres de rythme symboliques $\mathcal{L}_{\mathcal{S}}$ est alors le langage de termes créé à l'aide des symboles de $\widehat{\mathcal{L}} \cup \mathcal{N}$.*

Théorème 1 (Equivalence des arbres symboliques de rythme et des arbres de rythme d'*OpenMusic*)**.** *Il existe un isomorphisme $\widehat{\chi}$ entre $\mathcal{L}_{\mathcal{S}}$ et $\mathcal{L}_{\mathcal{OM}/\sim}$.*

1. Dans un parcours en profondeur.
2. Cela revient à faire de la numération unaire.

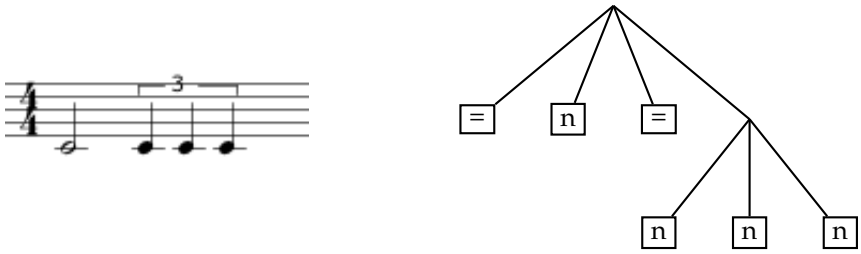

FIGURE 2.1. — Arbre symbolique de rythme `( = n = ( n n n))`

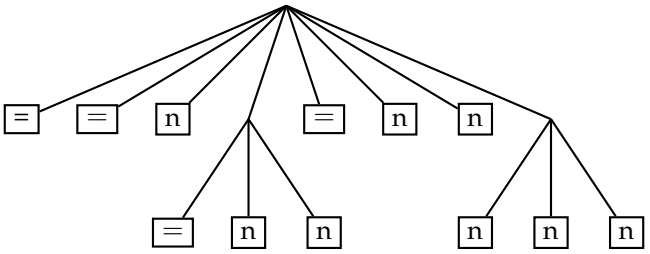

FIGURE 2.2. — L'arbre symbolique de rythme `( = = n ( = n n) = n n ( n n n ))`

Démonstration. L'idée de la démonstration est de construire deux morphismes ϕ et ψ tels que :

$$\phi(\mathcal{L}_{\mathcal{OM}/\sim}) = \mathcal{L}_{\mathcal{S}} \tag{2.1}$$

$$\psi(\mathcal{L}_{\mathcal{S}}) = \mathcal{L}_{\mathcal{OM}/\sim} \tag{2.2}$$

On aura alors l'inversibilité à gauche et à droite, d'où la bijectivité.

La démonstration étant assez pénible à lire (de nombreux cas à traiter), la définition des morphismes sous la forme du code source en *Common Lisp* qui opère la conversion entre les deux notations est reproduite en annexe A. □

2.2. Quantification par transformation d'arbres

L'algorithme de quantification procède ainsi :

1. Génération d'un arbre complet à partir des données de durée et d'attaque en entrée, et de paramètres de l'utilisateur ;
2. Application de règles de réécriture sur l'arbre pour le simplifier.

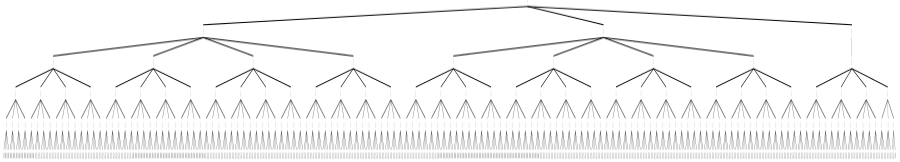

FIGURE 2.3. — Arbre de départ

2.2.1. Génération de l'arbre

La génération de l'arbre est assez brutale : après avoir choisi une profondeur, un tempo, et un nombre de fils (arité maximale de l'arbre), l'arbre complet de profondeur et d'arité choisies est créé (voir figure 2.3). Les notes les plus proches des données en entrée sont placés sur l'arbre complet, dans les feuilles. Lorsque la durée d'une note détectée excède la durée représentée par une feuille, elle est représentée par une note de la durée d'une feuille suivie du nombre approprié de liaisons.

2.2.2. Règles de réécriture

Les règles de réécriture sur les arbres rythmiques symboliques agissent sur une feuille ou sur un nœud interne et peuvent préserver [3] ou modifier les rythmes.

La figure 2.4 présente une règle qui préserve les rythmes, la figure 2.5, une règle qui modifie le rythme.

Les règles disponibles sont visibles en annexe B.

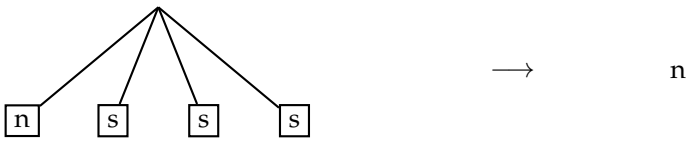

FIGURE 2.4. — Règle de réécriture de fusion de notes liées

Définition 5 (Monotonie)**.** *Pour tout symbole de fonction f d'arité n, pour tous termes $s_1, \dots s_n$ et t, si $s_i > t$, alors $f(s_1, \dots, s_i, \dots, s_n) > f(s_1, \dots t, \dots, s_n)$.*

Définition 6 (Stabilité par substitution)**.** *Pour toute substitution σ, pour tous termes s et s, si $s < t$, alors $\sigma s < \sigma t$.*

Théorème 2 (Terminaison)**.** *Le système de réécriture proposé vérifie la propriété de terminaison.*

3. Les durées des notes ne changent pas.

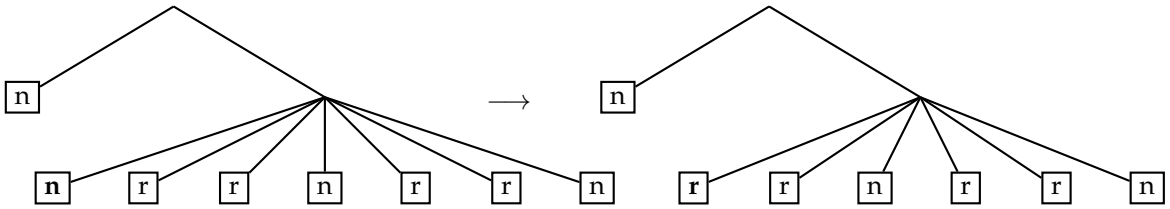

FIGURE 2.5. — Règle de réécriture d'élagage

Démonstration. Pour toutes les transformations sauf *gen*-=, la taille du terme après réécriture est *strictement* inférieure à celle du terme avant réécriture. Or il n'existe pas de suites strictement décroissantes dans $\mathbb{N}$. L'ordre « taille des termes », noté $<_t$, est *bien fondé* sur cet ensemble de règles de réécriture.

Cet ordre est aussi *monotone* et manifestement *stable par substitution*.

Par ailleurs, la transformation *gen*-= ne peut pas être appliquée à nouveau une fois qu'elle a été appliquée.

L'ordre $<_t$ peut donc être complété par composition lexicographique sur le système de réécriture entier en $<_s$ qui est alors *bien fondé*, reste *monotone*, et *stable par substitution*.

$<_s$ est donc un ordre de réduction.

Le système de réécriture vérifie ainsi la propriété de terminaison (voir [10]).

□

3. Résultats

3.1. La librairie *OMRewrite*

Le quantificateur a été implémenté en *Common Lisp* sous forme d'une bibliothèque *OpenMusic*, *OMRewrite*.

Elle permet d'utiliser le quantificateur, mais aussi de créer de nouvelles transformations, et de les appliquer sur des arbres. Des outils graphiques et interactifs ont été mis au point : la figure 3.1 montre par exemple une interface qui permet de définir une règle de réécriture simple entre termes et d'afficher la représentation arborescente des termes. La partie gauche représente le terme de départ, et la partie droite, le terme après réécriture. Il est possible de choisir une profondeur minimale et une profondeur maximale.

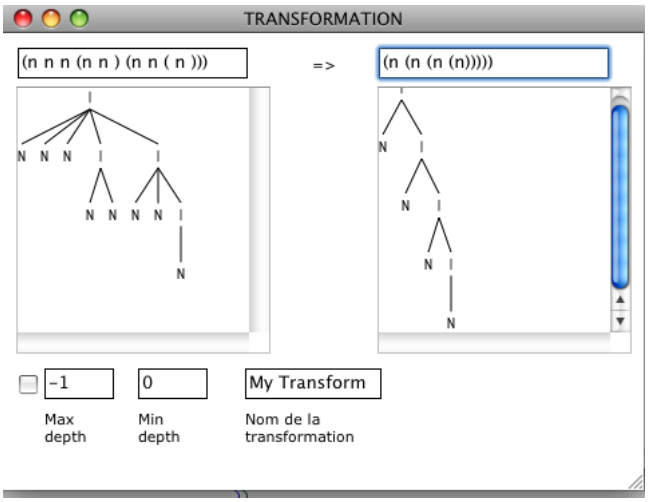

FIGURE 3.1. — Créer une règle de réécriture simple sans coder en Lisp

Des fonctions pour effectuer de la séparation de morceaux polyphoniques pour ensuite quantifier chacune des voix ont été aussi ajoutées.

En rajoutant certaines transformations [1], il est possible d'obtenir un système de réécriture confluent. Comme le système de réécriture (voir partie 2) termine, il est

1. Le plus simple est une transformation qui réduit la profondeur de l'arbre de rythme à 1, en ne notant plus que des proportions, en utilisant des suites de symboles =.

convergent. Il existe une unique forme normale pour chaque terme, et ainsi, l'équivalence de deux rythmes peut être décidée, en calculant leur forme normale, et en comparant les formes normales obtenues.

La bibliothèque peut être utilisée pour appliquer des règles de réécriture arbitraires, et pas seulement pour quantifier, ce qui a intéressé certains compositeurs.

3.2. Comparaison de `omquantify` et du quantificateur d'*OMRewrite*

`omquantify` reste plus précis que le quantificateur d'*OMRewrite*. En particulier, la première note semble souvent mal quantifié par *OMRewrite*. La table 3.1 compare les deux quantificateurs pour une même entrée. Le paramètre 60 correspond dans les deux cas au tempo, $(4, 4)$ au chiffrage la mesure, qui comporte alors quatre temps, et 4, dans le cas d'*OMRewrite* au nombre de subdivisions[2] ; 8 et 3 sont des paramètres qui permettent de contrôler la profondeur des subdivisions. Pour *OMRewrite*, *silence-strategy* est utilisée lors de l'étape de construction de l'arbre, le quantificateur ne prendra pas en compte les notes ou parties de notes plus petites que la plus petite subdivision.

Pour comparer la précision des résultats, les valeurs rythmiques ont été reconverties en durée.

	Attaques (ms)	Durées (ms)
En entrée	(300 2000 5345)	(1600 2345 3567.89)
OMRewrite	(500 2000 5500)	(1250 2250 3250)
`omquantify`	(286 2000 5333)	(1589 2333 3542)

TABLE 3.1. — Comparaison entre `omquantify` avec paramètres $60, (4, 4), 8$ et *OMRewrite* avec paramètres $4, 3, 60$ et *silence-strategy*

OMRewrite	0.22	0.04	0.09
`omquantify`	0.007	0.005	0.007

TABLE 3.2. — Erreur relative sur les durées pour les données de la figure 3.1

La moindre précision en durée (voir table 3.2) et en attaques (voir table 3.3) pourrait s'expliquer en remarquant que la quantification par transformation d'arbre opère localement, et hiérarchiquement. L'information temporelle est perdue. Cependant, le fait que la transformation est locale peut être un avantage pour de très longs morceaux : les erreurs ne sont pas propagées.

2. Chaque nœud aura 4 fils, lors de la phase de création de l'arbre.

OMRewrite	0.66	0	0.03
`omquantify`	0.05	0	0.002

TABLE 3.3. — Erreur relative sur les attaques pour les données de la figure 3.1

Il faut aussi noter que seule la précision en durée a été comparée. Il serait aussi pertinent, mais plus subjectif, de comparer la simplicité des notations obtenues, ainsi que la facilité d'utilisation et de paramétrage.

Conclusion

Une nouvelle notation pour les arbres de rythme a été proposée, qui a permis de définir diverses transformations sur les arbres, et d'évaluer une nouvelle approche pour la transcription rythmique.

La nouvelle approche de quantification par transformation d'arbre semble n'être pas assez précise en ce qui concerne la conservation des durées et des attaques mais elle permet de simplifier élégamment des arbres, et le principe d'utiliser des règles de réécriture sur des arbres de rythme pour l'aide à la composition s'avère intéressant pour les compositeurs : simplification de rythmes, équivalence de rythmes, définition de transformations par l'utilisateur. De nouvelles règles de réécriture pourraient aussi être ajoutées pour améliorer la précision en durée et en attaque.

Par ailleurs, la précision n'est pas le seul critère d'une bonne transcription rythmique : la lisibilité, l'adéquation à des critères esthétiques choisis par le compositeur en sont d'autres, sur lesquels *OMRewrite*, plus versatile, permet facilement d'agir..

Les travaux réalisés ont été présentés au cours d'un séminaire du groupe de travail sur le rythme à l'IRCAM et la nouvelle bibliothèque *OMRewrite* a été rendue publique sur le site [3] d'*OpenMusic* .

D'autres pistes à propos des arbres de rythme et de la quantification sont explorées à l'Ircam parallèlement, comme de l'apprentissage d'automates d'arbre, pour apprendre le style du compositeur, où la notation d'arbre symbolique de rythme pourra être réutilisée.

3. `http://repmus.ircam.fr/openmusic/`

Bibliographie

[1] Jean BRESSON, Carlos AGON et Gérard ASSAYAG : Openmusic : visual programming environment for music composition, analysis and research. *In Proceedings of the 19th ACM international conference on Multimedia*, pages 743–746. ACM, 2011.

[2] Carlos AGON, Gérard ASSAYAG et Jean BRESSON : *The OM Composer's Book*. IRCAM-centre Pompidou, 2006.

[3] Jean BRESSON, Carlos AGON et Gérard ASSAYAG : Visual lisp/clos programming in Openmusic. *Higher-Order and Symbolic Computation*, 22(1):81–111, 2009.

[4] Karim HADDAD : Fragments de recherche et d'expérimentation : Eléments de réflexions autour de l'écriture rythmique d'Emmanuel Nunes. Présentation à un séminaire Mamux à l'Ircam, nov 2012.

[5] Carlos AGON, Karim HADDAD et Gérard ASSAYAG : Représentation et rendu de structures rythmiques, 2002.

[6] Carlos AGON, Gérard ASSAYAG, Joshua FINEBERG et Camilo RUEDA : Kant : a Critique of Pure Quantification. *In ICMC*, 1994.

[7] Ali Taylan CEMGIL, Peter DESAIN et Bert KAPPEN : Rhythm quantization for transcription. *Computer Music Journal*, 24(2):60–76, 2000.

[8] Benoît MEUDIC : *Détermination automatique de la pulsation, de la métrique et des motifs musicaux dans des interprétations à tempo variable d'œuvres polyphoniques*. Thèse de doctorat, Ircam, 2004.

[9] David RIZO : *Symbolic music comparison with tree data structures*. Thèse de doctorat, Universidad de Alicante, 2010.

[10] R KRZYSZTOF : *Apt, Logic programming, Handbook of theoretical computer science (vol. B) : formal models and semantics*. MIT Press, Cambridge, MA, 1991.

A. Fonctions de conversion entre les arbres de rythme

Des arbres symboliques de rythme vers les arbres de rythme d'*OpenMusic*.

```
(defun tree-to-om
       (tree &optional  (last-symbol nil last-symbol-supplied-p) (nb-pulses 0) )
  "Converts the internal measure representation into an OM rhythm tree.
  n : note
  r : rest
  s : slur
  = : merge with following pulse"

  ;Conversion

  (let ((new-children nil)
        (eq-count 0)) ;to count the number of :=
    (dolist (child tree)
    (cond ( (equalp child :n) ; note
            ( push (1+ eq-count) new-children)
            (setf eq-count 0)
            (setf last-symbol 1))

          ( (equalp child :r) ;rest
            ( push (- (1+ eq-count)) new-children)
            (setf eq-count 0)
            (setf last-symbol -1))

          ( (listp child) ;subdivision
            (multiple-value-bind (mes last) (tree-to-om child last-symbol eq-count )
                (push  mes new-children)
            (setf eq-count 0)
            (setf last-symbol last)));to remember the last symbol of the subdivision

          ( (equalp child :=) (incf eq-count)) ;a merge symbol

          ( (equalp child :s) ; a slur
            (if  (numberp last-symbol)
                 ( push (* (float last-symbol) (1+ eq-count)) new-children)
              (progn
                (print ":s at very beginning so I decided it was a note.")
                ( push  (1+ eq-count) new-children)))
            (setf eq-count 0))

          ( t (print "Error while converting")))
    )

(setf new-children (nreverse new-children))

(values (list (1+ nb-pulses) new-children) last-symbol);return the subdivision and the
     last symbol
))
```

Des arbres de rythme d'*OpenMusic* vers les arbres symboliques de rythme.

```
(defun pulse-from-om (om-measure)
  "Converts an om measure (more generally, a subdivision) into the measure
    internal representation"
  (unless (null om-measure)
    (let* ( (subdi (first om-measure))
            (symbols (typecase subdi

                       (number (expand-symb subdi))

                       (list  (expand-symb (first subdi) (pulse-from-om (
    canonize (cadr subdi)))))
                       ))
            )
      (append symbols (pulse-from-om (rest om-measure)))
      ))
)

(defun symbols-from-om (symb)
  "Converts unary numbers into their internal representation"
  (typecase symb

    (float :s)

    (integer (if (plusp symb) :n :r))
    )
  )

(defun expand-symb (symb &optional (subdivision nil))
  "When n > 1 hit, expand it. Works on an isolated n or if it is the duration
     of a subdivision
as in (2 (1 1)).
Example :
4 -> := := := n"
  (append (make-list (1- (abs (floor symb))) :initial-element :=)
          (if (null subdivision) (list (symbols-from-om symb)) (list
    subdivision)))
)
```

B. Règles de réécriture

Transformation	Arguments	Description	Profondeur min	Changement du rythme ?
merge-rs	subdivision	Fusionne les silences et les liaisons qui occupent toute une subdivision	1	Non
merge-ns	subdivision	Fusionne les n suivis de s qui occupent toute une subdivision	1	Non
gen-=	subdivision	Fusionne des silences ou des notes qui n'occupent pas toute une mesure	0	Non
move-up	subdivision	Remonte les subdivisions unitaires	1	Non
dilate	subdivision nb-neighbours-left nb-neighbours-right	Dilate une subdivision	0	Oui
metric-reduce	subdivision nb-neighbours-left nb-neighbours-right	Supprime les subdivisions comportant des silences nouvellement créés par une dilatation	0	Oui