MMIM Modèles mathématiques en informatique musicale

Marc Chemillier

Master Atiam (Ircam), 2009-2010

Implémentation Lisp des modèles pour l'improvisation (d'après Gérard Assayag, Partie I)

- Les classes Oracle, Pythie et Improvizer
- Improviser une séquence MIDI
    - Pattern matching avec une grille harmonique
    - Navigation dans l'oracle
    - Transposition des données MIDI
    - Collecte des données MIDI

Présentation des fonctions Lisp pour l'improvisation gérant la pulsation et l'harmonie selon le mode appelé « Beat » basé sur un prototype antérieur à l'actuel OMax[1]. Le modèle fonctionne de manière off-line comme une librairie Open Music sans interaction temps réel avec Max/MSP.

Le code des exemples est disponible dans Open Music (fichier ImprotekTutorial.lisp), et les résultats peuvent être écoutés sur le Web :
http://ehess.modelisationsavoir.fr/atiam/improtek/index.html

1. La classe Oracle

C'est la classe générale implémentant l'oracle. Elle donne naissance à la classe Pythie par ajout d'un comparateur qui est la fonction de comparaison des étiquettes constituant la séquence. La classe Improvizer dérive de Pythie qui dérive de Oracle.

1.1 Exemple d'utilisation de la classe Oracle

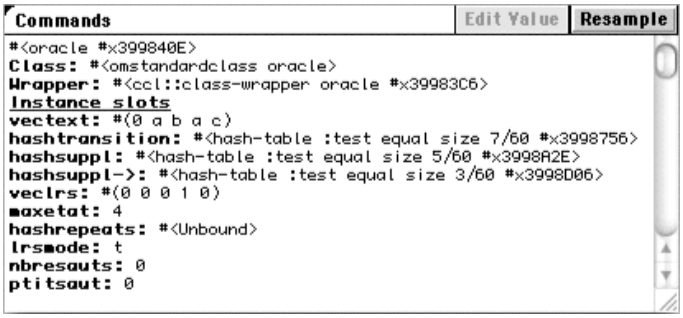
Exemple 1 : Un oracle construit sur la séquence *abac*.

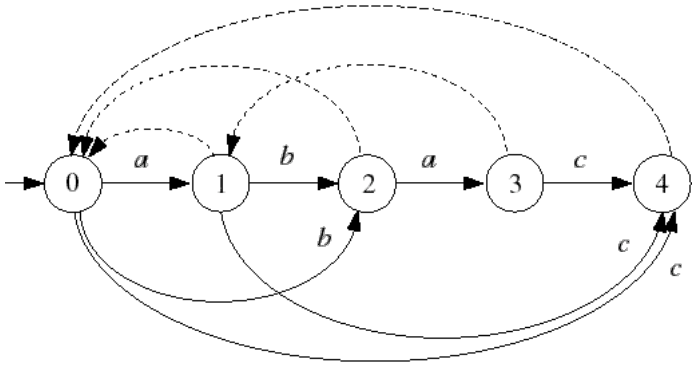
```
(setf o (make-instance 'Oracle :text '(a b a c) :lrsMode t))
```

[1] Une version ancienne OMax 2.0 datant de 2004 et tournant sous Common Lisp dans Open Music 4.9. Elle faisait suite à un premier prototype de 2001 appelé DJHarm dont l'interface temps réel avait été développée par Carlos Agon directement en Lisp dans Open Music. Les difficultés engendrées par le « garbage collecting » de Lisp ont conduit à concevoir OMax basé sur la communication de Max/MSP et Open Music.

Pour visualiser la structure de l'oracle et la valeur de ses champs :

```
(inspect o)
```





Affichage des transitions représentées par la table de hachage `(hashtransition o)` :

```
? (loop for x being the hash-key using (hash-value q) of (hashtransition o)
        do (format t "~a ---> ~a~%" x q))
(1 c) ---> 4
(3 c) ---> 4
(0 b) ---> 2
(2 a) ---> 3
(1 b) ---> 2
(0 a) ---> 1
(0 c) ---> 4
```

Affichage des liens suffixiels représentés par la table de hachage `(hashsuppl o)` :

```
? (loop for x being the hash-key using (hash-value q) of (hashsuppl o)
        do (format t "~a ===> ~a~%" x q))
1 ===> 0
2 ===> 0
3 ===> 1
4 ===> 0
0 ===> -1
```

La création de l'oracle `(make-instance 'Oracle :text '(a b a c) :lrsMode t))` effectue automatiquement l'apprentissage de la séquence `(a b a c)`. Elle parcourt tous les objets de la séquence en appelant pour chacun `(ajouter-objet myoracle objet)` qui est la fonction implémentant l'algorithme de Crochemore :

- elle crée un nouvel état d'arrivée pour lire l'objet,
- puis elle suit les liens suffixiels d'état en état :

- si `(transition myoracle etat objet)` ne donne pas déjà une transition avec cet objet comme étiquette, on l'ajoute jusqu'au nouvel état, puis :
  - si on est dans l'état 0, on s'arrête et on ajoute un lien suffixiel vers 0,
  - sinon on continue à suivre les liens suffixiels,
- si une transition existe déjà, on s'arrête et on ajoute un lien suffixiel du nouvel état vers l'état d'arrivée de cette transition.

1.2 Les champs de la classe Oracle

Champ `vectext` (longueur par défaut = 1000).
Vecteur des étiquettes de la séquence d'origine dans l'ordre où elles sont apprises, numérotées à partir de 1 avec un état initial supplémentaire égal à 0. Ce sont les étiquettes des transitions de l'automate dit « écorché » de la séquence. Accessible par `(text myoracle t)` pour le vecteur complet ou `(text myoracle etat)` pour l'étiquette correspondant à un état.

Champ `hashtransition` accessible par `(hashtransition myoracle)`.
Dans la classe Oracle, la table de transition contient des couples (key val) où key est un couple (état objet) et val l'état d'arrivée de la flèche qui part de cet état avec l'étiquette objet.
État d'arrivée de la transition partant d'un état avec l'étiquette objet :
`(transition myoracle etat objet)`

Champ `hashsuppl` accessible par `(hashsuppl myoracle)`.
La table des liens suffixiels contient des couples (key val) où key est le numéro d'un état et val l'état d'arrivée du lien suffixiel.

Champ `hashsuppl->` accessible par `(hashsuppl-> myoracle)`.
Ce sont les liens suffixiels inversés, c'est-à-dire les mêmes flèches orientées vers l'avant, mais parmi les liens suffixiels arrivant à un état, on ne garde que celui provenant de l'état le plus éloigné (le plus grand).

Champ `vectlrs`.
(lrs = longest repeated suffix)
longueur par défaut = 1000
Donne les longueurs des plus longs suffixes répétés pour chaque état.

Exemple 1 (suite) : Oracle construit sur *abac*. On peut afficher les liens suffixiels « inverses » (vers l'avant) représentés par la table de hachage `(hashsuppl-> o)` :

```
? (loop for x being the hash-key using (hash-value q) of (hashsuppl-> o)
        do (format t "~a ===> ~a~%" x q))
1 ===> 3
-1 ===> 0
0 ===> 4
```

Dans les liens inversés, les états 2, 3, 4 qui ne sont l'arrivée d'aucun lien suffixiel sont absents. Pour 0 et 1, on ne garde que le lien venant de l'état le plus grand, respectivement 4 et 3.

L'appel `(veclrs o)` donne les longueurs des suffixes répétés pour la séquence *abac* :

```
#(0 0 0 1 0)
```
Le seul suffixe répété est *a* de longueur 1, dans le mot *aba* qui correspond à l'état 3.

2. Les classes Pythie et Improvizer

2.1 La classe Pythie

Dans l'algorithme de Crochemore, lorsqu'on suit les liens suffixiels en se demandant : « *est-ce qu'une transition existe déjà depuis cet état avec la même étiquette ?* », on fait la comparaison des étiquettes. La classe Pythie est définie à partir de Oracle avec des fonctions de comparaison autres que `equal`.

Les transitions de la classe Pythie sont implémentées de manière différente. Il s'agit de couples (key val) où key est un état et val la liste des états d'arrivée de toutes les flèches qui partent de cet état. Les étiquettes des flèches sont sous-entendues, car dans l'oracle des facteurs, toutes les transitions arrivant sur un état ont la même étiquette, donc elles sont déterminées par les états d'arrivée.

Exemple 1 (fin) : Pythie construite sur la même séquence que précédemment *abac*.

```
(setf o
  (make-instance 'Pythie :text '(a b a c) :comparateur 'equal :lrsMode t))
```

Affichage des transitions de la classe Pythie :

```
? (loop for x being the hash-key using (hash-value q) of (hashtransition o)
        do (format t "~a ---> ~a~%" x q))
1 ---> (4 2)
2 ---> (3)
3 ---> (4)
0 ---> (4 2 1)
```

L'affichage des liens suffixiels directs ou inverses est le même que pour l'oracle.

Dans la classe Pythie, le champ `hashtransition` a une structure différente de celui de Oracle. La méthode `(flink etat mypythie)` donne la liste des états d'arrivée possibles à partir d'un état ce qui revient à faire `(gethash etat (hashtransition mypythie))`. La méthode `(transition mypythie etat objet)` a une définition différente pour Pythie de celle de la classe Oracle. Elle « compare » l'objet avec les étiquettes de la séquence d'origine correspondant aux états d'arrivée possibles, et donne l'état d'arrivée pour lequel la comparaison réussit.

2.2 La classe Improvizer

La classe Improvizer est définie à partir de Pythie en particularisant pour des applications musicales les objets de la séquence apprise lors de la construction de l'oracle. Ces objets sont

des événements qui peuvent être de plusieurs types. L'un d'eux est défini par la classe Beat. Il s'agit de « tranches » d'événements MIDI dans un contexte de tempo régulier correspondant à la durée d'*une pulsation*, ou éventuellement d'*un nombre entier de pulsations*, avec un label harmonique associé.

Attention : Pour les événements de la classe Beat, la fonction `CompareEvents` ne prend en compte que les labels harmoniques dans la détection des « répétitions » pour construire l'oracle.

Exemple 2 : On construit un Improvizer sur la séquence de notes montante et descendante C D E F E D C avec des durées égales de 500 ms. Les labels ne sont pas à proprement parler « harmoniques », mais désignent seulement les notes `(c)`, `(d)` etc.

```
(setf beat-list '(((c) ((60 0 500 80 1))) ((d) ((62 0 500 80 1)) ((e) ((64
0 500 80 1))) ((f) ((65 0 500 80 1))) ((e) ((64 0 500 80 1))) ((d) ((62 0
500 80 1))) ((c) ((60 0 500 80 1)))))
(setf beats (make-beat-list beat-list))
(setf myoracle
  (make-instance 'Improvizer :comparateur 'CompareEvents :lrsMode t))
(loop for i from 0 to (1- (length beats))
      do (learn-event myoracle (nth i beats)))
```

Affichage de la table de transition (en réordonnant les états) :

```
0 ---> (1 2 3 4)
1 ---> (2)
2 ---> (3 7)
3 ---> (4 6)
4 ---> (5)
5 ---> (6)
6 ---> (7)
```

3. Improviser une séquence MIDI

3.1 Pattern matching avec une grille harmonique

Improviser en suivant une grille revient à chercher dans la séquence d'origine des successions de labels identiques à ceux de la grille d'entrée, c'est-à-dire à faire du *pattern matching*.

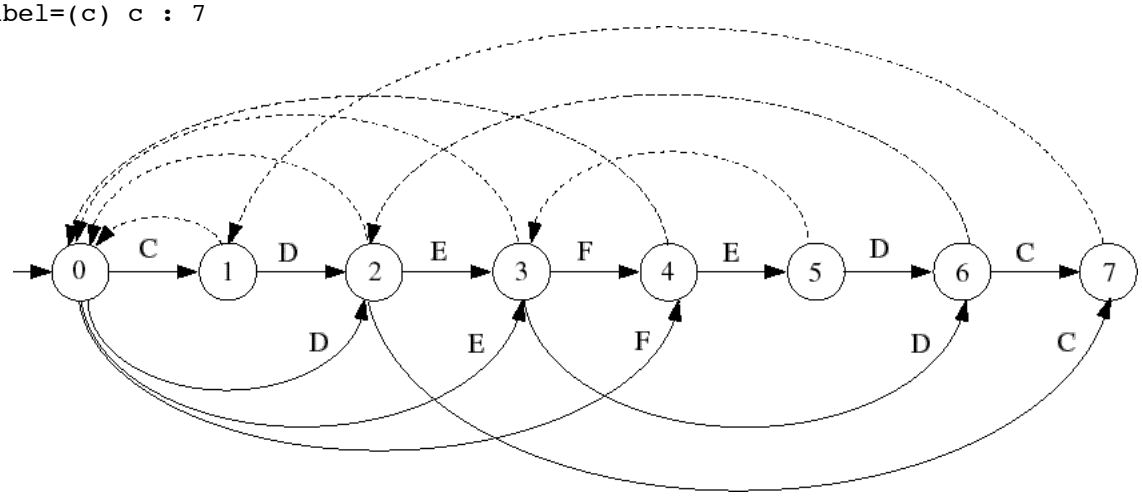
Avec la structure de l'oracle, le pattern matching consiste à suivre un chemin dans l'automate avec des transitions étiquetées par les labels de la grille d'entrée si c'est possible. Quand ça ne l'est pas, on peut suivre un lien suffixiel permettant d'aller à un autre état où l'on peut éventuellement lire l'étiquette courante (les liens suffixiels garantissent que dans la séquence d'origine, il existe une partie commune entre les deux fragments concaténés), ou bien on cherche cette étiquette dans un état quelconque de l'automate indépendamment du chemin parcouru.

Exemple 2 (suite) : On reprend l'Improvizer sur la séquence C D E F E D C et on calcule une improvisation que l'on peut jouer avec `play` :

```
(setf labels '((e) (d) (c) (f) (e) (d) (c)))
(setf impro (ImprovizeOnHarmGrid myoracle 7 labels))
(play (beats->chseq impro 500 0))
```

L'affichage des étapes du calcul montre que la fonction suit les enchaînements appris (ici les mouvements conjoints descendants), c'est-à-dire qu'elle parcourt les transitions de l'oracle ce qui se traduit par l'affichage `c :` (pour continuité) suivi du numéro d'état :

```
((e) (d) (c) (f) (e) (d) (c))
label=(e) Starting point : 3
label=(d) c : 6
label=(c) c : 7
label=(f) nothing, new : 4
label=(e) c : 5
label=(d) c : 6
label=(c) c : 7
```



Si un enchaînement ne figure pas dans la séquence d'origine, `Improvize` cherche les labels isolément. Dans cet exemple, la répétition de la note C est absente de la séquence apprise, donc les labels répétés `(c)` sont choisis de façon autonome ce qui se traduit par l'affichage `new`.

```
((c) (c) (c) (d) (e) (d) (c))
label=(c) Starting point : 1
label=(c) nothing, new : 1
label=(c) nothing, new : 1
label=(d) c : 2
label=(e) c : 3
label=(d) c : 6
label=(c) c : 7
```

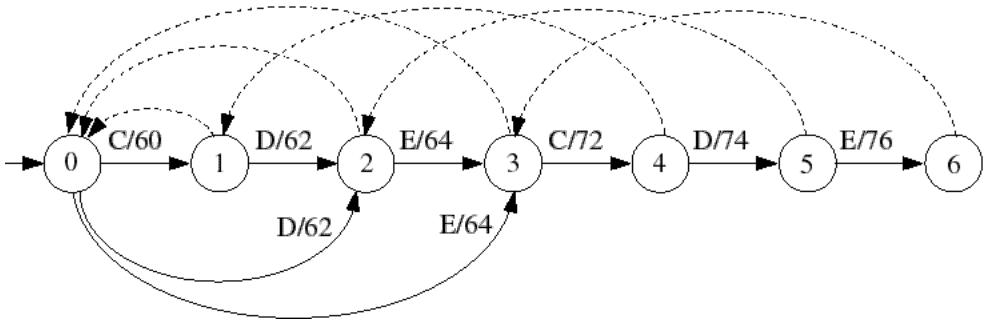
Exemple 3 : Autre exemple d'Improvizer construit sur une séquence comportant deux fois la suite de labels C D E associée aux notes 60 62 64, puis à l'octave supérieure 72 74 76.

```
(setf beat-list '(((c) ((60 0 500 80 1))) ((d) ((62 0 500 80 1))) ((e) ((64
0 500 80 1))) ((c) ((72 0 500 80 1))) ((d) ((74 0 500 80 1))) ((e) ((76 0
500 80 1)))))
```

Les transitions sont les suivantes :

```
0 ---> (1 2 3)
1 ---> (2)
2 ---> (3)
3 ---> (4)
4 ---> (5)
5 ---> (6)
```

L'improvisation suit la grille en sélectionnant des notes MIDI qui basculent entre les deux octaves. Dans l'exemple suivant, la suite de labels C D E D E C D E D E C est jouée d'abord avec la suite de notes (72 74 76) sur les états 4, 5 et 6 de l'oracle, puis avec (62 64) sur les états 2 et 3, puis avec (72 74 76) sur les états 4, 5 et 6, et enfin avec (62 64 60) sur les états 1, 2, et 3.




```
? (play (beats->chseq (ImprovizeOnHarmGrid myoracle 11 labels) 500 0))
((c) (d) (e) (d) (e) (c) (d) (e) (d) (e) (c))
label=(c) Starting point : 4
label=(d) c : 5
label=(e) c : 6
label=(d) nothing, new : 2
label=(e) c : 3
label=(c) c : 4
label=(d) c : 5
label=(e) c : 6
label=(d) nothing, new : 2
label=(e) c : 3
label=(c) nothing, new : 1
```

3.2 La fonction de navigation dans l'oracle

Le calcul d'une improvisation avec un oracle Improvizer en suivant une grille d'accords se fait avec `(ImprovizeOnHarmGrid myoracle length labels)`. La variable length donne la longueur de l'improvisation voulue. Elle correspond au nombre d'accords de la grille labels.
Remarque : Notons qu'`ImprovizeOnHarmGrid` lance une fonction générique `Improvize` qui peut aussi calculer une improvisation sans référence à une grille avec des événements d'un autre type que la classe Beat (de ce fait il existe plusieurs fonctions `eligible-beat?` adaptées aux différents cas).

À chaque étape, on teste la « continuité » du calcul dans l'oracle, c'est-à-dire le nombre de transitions successives autorisées. Ce contrôle limite la longueur des segments **recopiés** dans la séquence d'origine.
Si la continuité est inférieure à une valeur maximale (fixée à 8), ou bien s'il n'y a pas de lien suffixiel à partir de l'état obtenu, on se met en mode « continuité » en augmentant la continuité de 1, sinon en mode « suppléance » en réinitialisant la continuité à 0.

La variable `continuity` effectue ce contrôle. Le champ `max-continuity` de la classe Improvizer mémorise la valeur maximale souhaitée.

Le calcul consiste à lire les labels successifs de la grille harmonique et à chercher des beats correspondants dans l'oracle :
• En mode « continuité » : on choisit une transition avec la bonne étiquette s'il en existe et on avance en sélectionnant le beat correspondant, sinon on passe en mode « suppléance » s'il y a des liens suffixiels ou en mode « nothing » s'il n'y en a pas.
• En mode « suppléance » : on suit un lien suffixiel s'il y en a (en choisissant le plus long suffixe répété) et on avance en sélectionnant le beat trouvé si c'est possible, sinon on affiche « nothing » en passant dans le mode correspondant.
• En mode « nothing » : on cherche le label courant indépendamment de ce qui précède et on avance si on le trouve en sélectionnant le beat correspondant, sinon on met un beat vide et on affiche « empty ».

3.3 Transposition des données MIDI suivant la grille

Au cours de la navigation dans l'oracle, il arrive qu'on cherche le label courant indépendamment de la continuité de ce qu'on a lu précédemment (mode « nothing »). Dans cette situation, on peut se permettre de faire une transposition qui augmente le nombre de possibilités de l'oracle. La recherche du label se fait non plus seulement à l'unisson, mais aussi à une distance maximale d'une tierce mineure au-dessus ou au-dessous de la grille donnée :

- Si l'accord courant de la grille est X et si l'on trouve $X + d$ dans l'oracle, il faut transposer ses données MIDI de l'intervalle $-d$ pour les adapter à la grille.

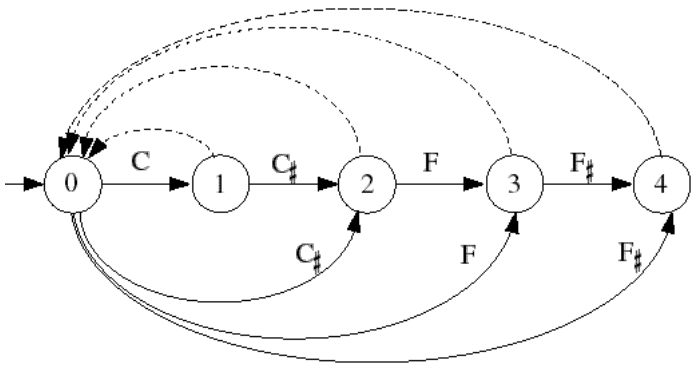
- Mais pour garder la continuité avec les accords ultérieurs, il faut chercher non pas les accords suivants Y de la grille mais les accords modifiés $Y + d$, ce qui obligera à faire également la transposition de l'intervalle $-d$ sur leurs données MIDI.

- La situation peut se reproduire avec un accord ultérieur de la grille X (qui est en fait un $Y + d$) pour lequel on trouve dans l'oracle $X + d' = Y + (d + d')$ avec d' dans les bornes $[-3 - d, 3 - d]$. La nouvelle transposition sera le cumul $d + d'$ qui reste dans les bornes $[-3, 3]$ et on continue avec cette nouvelle valeur appliquée aux accords de la grille $Y + (d + d')$ en transposant leurs données MIDI de $-(d + d')$.

La fonction qui cherche le label courant à une transposition près est :
```
(find-beat-label-match myoracle label)
```

Exemple 4 : Improvizer construit sur les enchaînements C C# F F#.

```
(setf beat-list '(((c) ((60 0 500 80 1))) ((c#) ((61 0 500 80 1))) ((f)
((65 0 500 80 1))) ((f#) ((66 0 500 80 1)))))
```



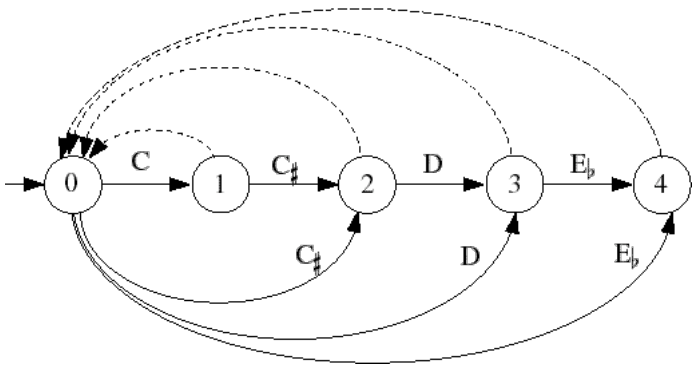
Voici un exemple de grille factorisée en segments parcourus avec des transpositions différentes :
(C C# C) (G G#) (C#) (C C) (G# G)
correspondant aux étapes suivantes :

	Transpositions					
Labels	0	–2	–1	0	–3	–2
C	C					
C#	C#					
C	C					
G	[G]	F				
G#		F#				
C#		[B]	C			
C			[B]	C		
C				C		
G#				[G#]	F	
G					[E]	F

```
((c) (c#) (c) (g) (g#) (c#) (c) (c) (g#) (g))
label=(c) Starting point : 1
label=(c#) c : 2
label=(c) new : 1 transpo=0
label=(g) new : 3 transpo=-2
label=(f#) c : 4
label=(b) new : 1 transpo=-1
label=(b) new : 1 transpo=0
label=(c) new : 1 transpo=0
label=(g#) new : 3 transpo=-3
label=(e) new : 3 transpo=-2
```

Exemple 4 (suite) : Autre Improvizer construit sur les enchaînements C C# D Eb.

```
(setf beat-list '(((c) ((60 0 500 80 1))) ((c#) ((61 0 500 80 1))) ((d)
((62 0 500 80 1))) ((eb) ((63 0 500 80 1)))))
```



Voici un exemple de grille factorisée selon des transpositions différentes avec des segments plus longs du fait de l'existence dans l'oracle de continuités plus longues :
(C C# D Eb) (E F F#) (A Bb B C)

	Transpositions		
Labels	0	–3	3
C	C		
C#	C#		
D	D		
Eb	Eb		
E	[E]	C#	
F		D	
F#		Eb	
A		[F#]	C
Bb			C#
B			D
C			Eb

```
((c) (c#) (d) (eb) (e) (f) (f#) (a) (bb) (b) (c))
label=(c) Starting point : 1
label=(c#) c : 2
label=(d) c : 3
label=(eb) c : 4
label=(e) new : 2 transpo=-3
label=(d) c : 3
label=(eb) c : 4
label=(f#) new : 1 transpo=3
label=(c#) c : 2
label=(d) c : 3
label=(eb) c : 4
```

3.4 Collecte des données MIDI de la séquence générée

On peut jouer une improvisation directement dans Open Music avec la fonction `play` (sans utiliser Max). Pour cela, il faut collecter les données MIDI contenues dans les Beats successifs.

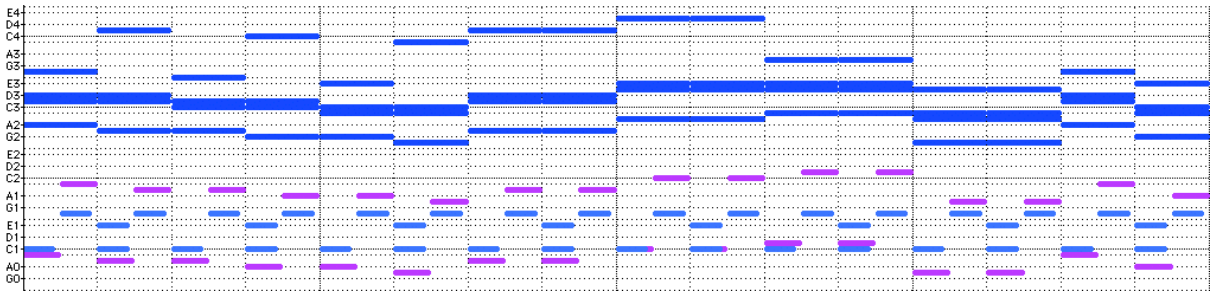
Exemple 5 : Un Improvizer avec des labels d'accords réels Bm7 et E7 et les voicings associés en données MIDI (piano sur canal 1, basse sur canal 2, le canal 14 servant à coder les accords). En naviguant dans l'oracle, chaque accord peut être transposé au maximum une tierce mineure au-dessus ou au-dessous. Ainsi les voicings de Bm7 s'appliquent pour G#m7-Dm7 et ceux de E7 pour C#7-G7, ce qui permet de jouer la séquence Bm7 E7 Bbm7 Eb7 Am7 D7 E7 E7 F#7 F#7 C#m7 C#m7 G#m7 G#m7 Bm7 Am7.

```
(setf fromrain  '(                              ;beat duration = 484 ms
((b m7) ((11 0 484 2 14) (35 0 242 97 2) (66 0 484 69 1) (62 0 484 69 1)
(61 0 484 69 1) (57 0 484 69 1) (47 242 242 97 2)))
((e 7)  ((4 0 484 3 14) (34 0 242 97 2) (73 0 484 69 1) (62 0 484 69 1) (61
0 484 69 1) (56 0 484 69 1) (46 242 242 97 2)))))

(setf oracle5
      (let ((beats (make-beat-list fromrain)) (o (NewImprovizer)))
        (loop for i from 0 to (1- (length beats))
              do (learn-event o (nth i beats))) o))


(setf 2beatpoumchi '((nolabel nolabel 2) ((36 0 207 121 10) (42 242 207 121
10) (36 484 207 121 10) (40 484 207 121 10) (42 726 207 121 10))))


(setf labels5 '((b m7) (e 7) (bb m7) (eb 7) (a m7) (d 7) (e 7) (e 7) (f# 7)
(f# 7) (c# m7) (c# m7) (g# m7) (g# m7) (b m7) (a m7)))


(progn (setf impro nil)
(pgmout 4 1) (pgmout 33 2)
(ctrlchg 7 127 1) (ctrlchg 7 127 2)
(let* ((n (length labels5))
       (drumpart (make-list (round (/ n 2))
                            :initial-element 2beatpoumchi)))
  (setf impro (merger (beats->chseq (ImprovizeOnHarmGrid oracle5 n labels5)
                                    484 0)
                      (beats->chseq (make-beat-list drumpart) 484 0)))
  (play impro)))
```

On obtient l'affichage suivant indiquant les transpositions :

On peut visualiser le piano-roll du résultat en le sauvant dans un fichier MIDI :

```
(save-as-midi impro)
```



La fonction de collectage des données MIDI contenues dans les Beats est `beats->chseq`. Elle utilise la fonction :
`(make-quanti-chords note-list deltachords)`
qui prend une liste de notes représentées par des quintuplets (pitch onset vel dur off chan), c'est-à-dire dans le format des MidiSet associés aux Beats, et fabrique un objet chordseq jouable dans Open Music.

On peut combiner deux chordseq avec la fonction `merger` pour synchroniser plusieurs improvisations ou ajouter une partie additionnelle. On peut effectuer des réglages avec `(pgmout 4 1)` pour le choix d'un son ou `(ctrlchg 7 127 1)` pour le contrôle du volume. On peut sauver une improvisation représentée par un chordseq dans un fichier MIDI avec `save-as-midi`.