MPIL(LI332)
Année 2015/2016
Mini-projet de MPIL : partie 1

Objectifs

1. Structure d'arbre
2. Lecture et écriture de fichiers MIDI
3. Module graphique
4. Stratégies d'écriture

Le chant *polyphonique* (*i.e.* à plusieurs voix) apparaît vers le IX^e siècle dans la musique religieuse (chant grégoriens). L'une des pratiques les plus anciennes consiste à improviser une deuxième voix, le *contre-chant*, par dessus une ligne mélodique simple et dépourvue de rythme appelée *cantus firmus* (figure 1).

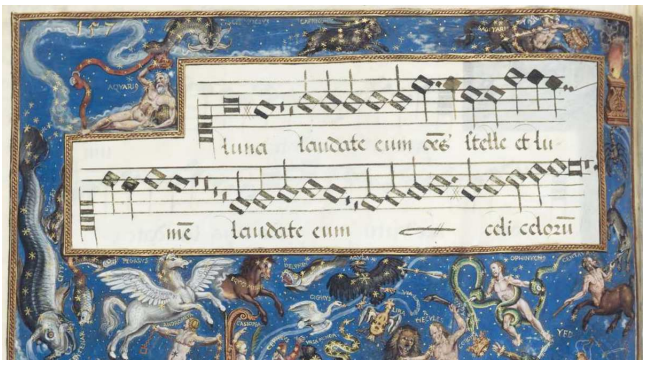

FIGURE 1 – Exemple de *cantus firmus*. La carte du ciel étoilé peinte par Hans Mielich en 1570 pour le duc Albert V de Bavière en illustration du motet Laudate Dominum d'Orlando de Lassus.

Dans le cadre de ce projet, nous souhaitons créer un algorithme de génération automatique de contre-chant. Pour un cantus firmus donné, l'ensemble des contre-chants possibles peut être représenté dans une structure d'arbre de listes proche de celle d'un arbre lexical. Il suffit ensuite de parcourir l'une de ces branches pour générer une mélodie, de la même manière que l'on parcoure les branches d'un arbre lexical pour former des mots.

Cette première partie ne porte que sur l'implémentation de la structure d'arbre représentant l'ensemble des contre-chants possibles et l'import de données depuis un fichier *csv*. L'ensemble du projet comportera également un module de lecture et écriture de fichiers au format MIDI et d'édition de partition. Nous proposons une implémentation constituée d'un module de lecture de fichiers, d'un module de structure d'arbre et d'un foncteur prenant ces deux modules en argument pour construire un module principal (voir figure 2).

Exercice 1 : Structure d'arbre

Une voix est une succession de notes que l'on représente par une liste d'entiers. Chaque entier est associé à une hauteur de note, le Ré central étant fixé à 0 et les altérations ignorées. Ainsi, la liste $[|0; 1; 2|]$ représente la mélodie Ré, Mi, Fa.

Q.1.1 Créer un module de lecture de fichiers qui comporte une exception en cas de fichier invalide et une fonction de lecture qui prend en entrée un nom de fichier et retourne une liste d'entier. On écrira d'abord une signature *READSCORE* suffisamment général, puis une implémentation adaptée à la lecture de fichiers *CSV* (cf fichier *cantus.csv*).

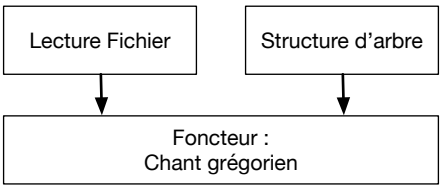

FIGURE 2 – Schéma de fonctionnement des différents modules du système. Le module principale est généré par un foncteur prenant en argument le module de lecture d'un fichier et le module de structure d'arbre

Solution :

Les règles strictes qui régissent l'écriture du contre-chant permettent d'aborder le problème sans connaissances musicales. On note $f(1), ..., f(n), ...f(N)$ la liste d'entiers représentant le cantus firmus. On souhaite construire à partir de cette liste l'ensemble des liste $c(1), ..., c(n), ...c(N)$ représentant les contre-chants possibles.

contrainte tessiture les notes du contre chant $c(n)$ ne peuvent prendre des valeurs comprises qu'entre 4 et 11.

contrainte harmonique la différence $|f(n) - c(n)|$ doit être comprise $[0; 2; 4; 5; 7]$

contrainte mélodique la différence $|c(n-1) - c(n)|$ doit être comprise $[0; 1; 2; 3; 4; 5; 7]$

Le module de structure d'arbre comporte la signature suivante

On se propose dans les questions suivantes d'écrire une implémentation de la signature *TREESTRUCTURE*. Vous pourrez vous aider de l'exemple d'arbre donné dans le fichier *exemple.ml* pour tester vos fonctions, et du schéma représentant la construction d'un tel arbre (voir figure 3).

Q.1.2 Les solutions possibles pour le cantus firmus sont représentées dans une structure d'arbre lexical. Écrire le type *holy_tree* qui représente une telle structure. On fera en sorte que chaque feuille de l'arbre contienne un couple d'entier composé du contre-chant et du cantus firmus, ceci afin de pouvoir choisir un chemin lors du parcours de l'arbre sur des critères de distance entre contre-chant et cantus firmus.

Contrairement à l'arbre lexical, il n'y a pas de booléen indiquant la fin d'un mot et toute séquence partant de la racine et mesurant la même longueur que le cantus firmus constitue un contre-chant possible.

Solution :

Q.1.3 Écrire une fonction *noeud_possibles* qui prend en entrée une note du cantus firmus $f(n)$ et une note du contre chant $c(n-1)$ et qui donne en sortie une *holy_tree list* contenant les notes possibles pour le contre chant. On tient compte pour cela des 3 règles, mélodique, harmonique et absolue, définies précédemment.

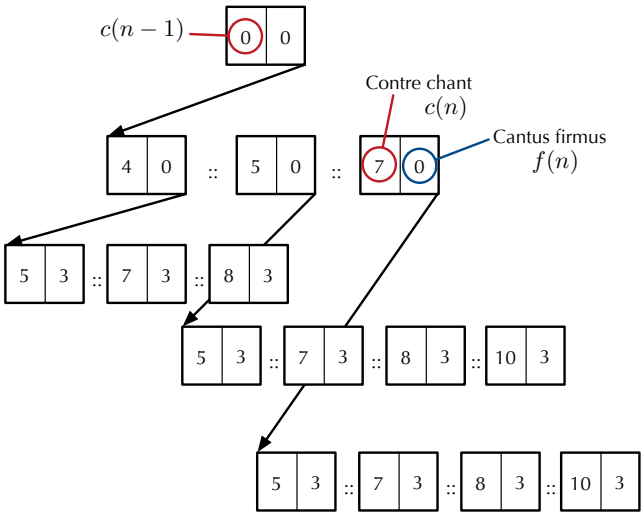

FIGURE 3 – Structure représentant les contre chants possibles pour le cantus firmus (0,3). Chaque noeud contient un couple d'entiers et une liste de noeuds. La liste de noeuds correspond aux potentielles notes suivantes.

Solution :

Q.1.4 A l'aide de la fonction *noeud possible* précédemment définie, écrire une fonction *construire arbre* qui prend en argument une liste d'entiers correspondant au cantus firmus et retourne le *holy tree* des contre chants possibles. Un exemple de construction d'arbre est donné dans *exemple.ml*.

Solution :

Q.1.5 Un chemin partant de la racine de l'arbre est allant jusqu'à une feuille définit un contre-chant potentiel. Il est possible que lors de la construction de l'arbre, certains chemins aboutissent à une impasse (aucune contre-chant ne permet de satisfaire les contraintes à l'étape suivante). On souhaite, avant la phase de parcours de l'arbre, supprimer les chemins impasse. Écrire une fonction *nettoyer_arbre* qui prend en entrée un *holy_tree* de profondeur n (chemin le plus long depuis la racine jusqu'à une feuille) et retourne un *holy_tree* dont les chemins de longueur strictement inférieure à n ont été supprimés.

Solution :

Q.1.6 La dernière étape est d'écrire une fonction qui prend en entier un *holy_tree* auquel la fonction *nettoyer_arbre* a été appliquée, effectue de manière aléatoire un parcours dans cet arbre depuis la racine jusqu'à une feuille, et retourne ce parcours sous la forme d'une liste d'entiers qui représente alors un contre-chant possible au cantus firmus de l'arbre.

Solution :