Projet Oz

Jean-Louis Peffer: 72232300
Isaac Yamdjieu Tahadie: 07152201

Table des matières

1 Limitations et problèmes connus

1.1 PartitionToTimedList

Nous n'avons pas rencontrer de limitations ou de problèmes spécifiques lors de la phase de test de cette partie.

1.2 Mix

Lorsque nous avons testé le cas où `<music> ::= [partition([g])]` et que nous écrivons les `<samples> = {Mix P2T [partition([g])]} = Mixed1` dans un fichier `File1 = "wave/partition.wav"` avec `{Project2025.writeFile File1 Mixed1}`, et que nous comparons les résultats des samples retournés avec `{Project2025.readFile File1}`, on remarque qu'il y a une erreur d'assert, probablement due aux erreurs de comparaison sur les flottants (même en appelant `{Normalize}`).
C'est pour cela que nous avons écrit les tests pour `{TestPartition P2T Mixarg}` avec l'appel à `{ECHSPartition Partition P2T}` de Mix.oz en l'exportant et en l'appelant de la manière suivante : `{Mix.echsPartition [g] P2T}`

1.3 Test

Pour les tests nous avons rajouté une procedure pour tester mix, cette procédure `{TestMixChaining P2T Mix}` est similaire à `{TestP2TChaining}` mais le fait pour la fonction Mix, nous n'avons pas rajouter le cas où `<part> ::= partition(<partition>)` due à l'erreur sur les flottants mentionnes ci-dessus.

2 Justification des constructions non-déclarative

2.1 PartitionToTimedList

Pour cette partie nous avons utilisé des éléments non-déclaratifs pour plusieurs fonctions :

- `{ExtendedChordTime Pi}` une fonction helper qui sert a déterminer si les notes étendues d'un accord étendue ont la même durée (retourne true si elle n'ont pas les mêmes durée et false sinon)
- `{IsChord Pi}` une fonction helper qui sert à déterminer si Pi est un accord non étendu (true si oui, false sinon)
- `{IsExtendedChord Pi}` une fonction helper qui fais la même chose que celle ci-dessus mais pour un accord étendu
- `{ChordToExtended Chord}` une fonction qui renvoie le format étendu d'un accord simple
- `{Stretch Factor Partition}` une fonction qui traite le cas ou ⟨`partition item`⟩ `::=` ⟨`stretch(factor:` ⟨`factor`⟩ ⟨`partition`⟩`)`⟩

Pour `{ExtendedChordTime Pi}`, `{IsChord Pi}`, `{IsExtendedChord Pi}` la raison est la même. Les cellules `A = {NewCell false}` et `B = {NewCell true}` permettent de boucler sur les notes de l'accord et de vérifier si elles sont valide (si elle ne le sont pas alors B est

mis à false ou sinon A est mis a true), lorsque nous sortons de la boucle on vérifie si `B == false` et nous retournons false. Vous pouvez remarquer que nous ne vérifions pas que `A == true`, la raison de cette cellule est pour avoir une déclaration si la note est valide.
On pourrais s'en passer si nous avions compris comment retourner directement lorsqu'on est dans une boucle for, mais nous avions perdu du temps à essayer de comprendre comment cela fonctionne en Oz (on ne comprenais pas la syntaxe `break: B` de la documentation.) donc nous avons décidé d'utiliser des cellules.

Pour `{ChordToExtended Chord}` par souci de simplicité nous avons décidé de rajouter une cellule pour stocker le résultat intermédiaire lorsque nous bouclons sur les notes de l'accord. On n'aurait pu utiliser une fonction récursive pour faire cela car en utilisant une cellule il faut rajouter une étape en plus pour que la liste de note étendue ne soit pas à l'envers.

De plus dans `{Stretch}` nous avons fait appel à deux cellules `Accumulator = {NewCell nil}` qui jouaient simplement un rôle d'accumulateur, au début il contient simplement la tête de la liste et à chaque itération nous mettions à jour le contenu de la liste. Et grâce à l'appel de `{List.reverse ...}` nous obtenions une liste dans l'ordre inverse. Nous avions aussi l'appel à une deuxième cellule `NewChordAccumulator = {NewCell nil}` qui est un second accumulateur qui nous a permis de construire une liste des notes de l'accord. On aurait pu s'en passer en créant 2 fonctions récursives différentes qui utilisent des accumulateurs en arguments pour les cas où on a une note ou un accord. Mais l'utilisation de cellule nous permet de directement créer ces listes dans la fonction stretch.

2.2 Mix

Pour cette partie la fonction qui utilise une cellule est `{FindBL ListofList}`, une fonction qui permet de trouver la liste de plus grande taille parmi une liste de liste et qui retourne la taille de la plus grande liste. La raison de cette cellule est de pouvoir faire une fonction récursive sans accumulateur (plus simple a tester) dans la fonction `{FindBL ListofList}`. On n'aurait pu s'en passer en rajoutant la taille de la plus grande liste a présent en argument de la fonction récursive `{FindBLRec LofL}` et de regarder si la taille de la liste courante est plus grande que celle-ci, et si elle est plus grande on la remplace.

3 Choix d'implémentations surprenante

La première chose qui peut être surprenante est la quantité de fonctions helpers, on a fait cela pour permettre de diviser le travaille d'une fonction de transformations/filtre en plusieurs sous taches indépendantes et de tester si chaque fonctions helpers fonctionnent correctement (test local avec des browse et declare).

3.1 PartitionToTimedList

La plus part des helpers sont des fonctions utilisés pour déterminer si nous avons une note, note étendue, accord ou accord étendu qui sont utile pour faire une partition correcte. Ensuite le reste sont la pour les transformations. Celle qui nous ont le plus surpris pour les transformations sont les fonctions `{MapNote Note Sharp}` qui retournent la valeur entière équivalente d'une note (comprise entre 0 et 1100), et leurs inverses `{MapintPos Int}` et

`{MapintNeg Int}` qui respectivement transforment la valeur positive/négative en note équivalente (un entier negatif ne correspond pas à la note équivalente en valeur absolue) donc il était nécessaire de faire 2 fonctions différentes. Ces fonctions sont utiles pour transposer une note efficacement. On a aussi fait des fonctions pour déterminer le nombre d'octave à augmenter ou descendre (`{HowManyOUp Transposednote}` et `{HowManyODown Note Semi}`) car l'arithmétique pour déterminer le changement d'octave vers le bas ou le haut ne fonctionne pas de la même manière.

3.2 Mix

Pour cette partie le choix de copier-coller certaines fonctions helpers de PartitionTo-TimedList a été fait pour pouvoir déterminer la hauteur d'une note (celle-ci a été faite en utilisant `{TransposeNote Nint Octave Semi Duration Instrument}` qui transpose la note à déterminer l'hauteur de 1 ou -1 semiton jusqu'à ce qu'on trouve la note de référence, tout en gardant compte de combien de fois nous avons transposé la note. On aurait pu exporter les fonctions voulues du fichier PartitionTotimedList et les appelées dans Mix, mais cela prenait plus de temps à tester, car en copiant ces fonctions on pouvait tester localement. Un autre choix d'implémentations qui peux être surprenant est l'utilisation de threads dans les fonctions pour l'échantillonnage et pour certaines transformations, cela a été fait car si on utilisait Mix pour échantillonner une partition on aurait eu des listes très longue (de $\pm$ 44100 éléments ou plus) et l'utilisation de threads permet de déterminer les samples plus rapidement sans avoir de blocage.

Nous avons aussi utilisé la fonction `{Build N F}` qui a permis de construire une liste de longueur N. La question ici est de se demander pourquoi n'avons nous pas utilisé la fonction `{MakeList}` qui est dans la documentation Mozart. `{MakeList N}` créait une liste de longueur N remplie de variables fresh (non-liée). Alors que `{Build N F}` est beaucoup plus générique c'est à dire que à chaque indice différent, il peut produire une valeur différente. Par exemple dans la fonction `{Fade Start Finish Music P2T}` elle a permis de générer une suite par exemple `{0.0,1/D,...}` pour le fadeIn et `{1.0,(D-1)/D,...}` pour le fadeOut. Ce qu'il y a de surprenant pour cette fonction c'est qu'elle nous a permis d'embarquer n'importe quelle logique dans la génération d'éléments. Elle était utile pour le `{Fade}` car cette dernière se combine très bien avec le `{Build ..}` pour tout type de séquence indexée. Nous aurions pu utiliser un appel de la fonction de la documentation Mozart sur une liste d'indice, mais dans le cas de la fonction filtre `{Fade ...}`, `{Build ...}` évite de d'abord construire explicitement une liste par exemple comme 0|1|2|...|D-1|nil. De plus nous avons utilisé plusieurs autres fonctions comme :

— `{Zero N}` Cette fonction qui a été appelée dans plusieurs autres fonctions comme `{Cut Start Finish Music P2T}` et dans `{Echo Decay Delay Repeat Music P2T}` qui permettait de créer une séquence d'échantillon silencieux, elle a été fondamentale pour pouvoir décaler un signal dans le temps dans le cas par exemple de `{Echo}` et de compléter un segment trop court dans le cas de `{Cut}`

— `{TakeSamples N L}` cette fonction va prélever exactement N échantillons au début de la liste, et rejeter le reste. On peut voir son utilité dans `{Cut}`.

— `{DropSamples N L}` Cette fonction ignore les N premiers échantillons d'une liste, et va renvoyer tout ce qui suit. Dans `{Cut}` elle va décaler la lecture de la musique jusqu'à l'instant de départ Start.

— `{MultList L1 L2}` Qui nous a permis d'appliquer point par point un facteur à une série d'échantillons.

4 Extensions

Nous avons décidé de rajouter l'extension de la créativité en transcrivant l'instrumentale de Careless Whisper par George Michael, elle reprend surtout la partie classique ou le saxophone joue (mais comme nous n'avons pas implémenté l'extension pour les instruments, elle joue donc avec une onde sinusoïdale) et ensuite la partie où les accords sont joués. Vous pouvez remarquer que nous avons utilisé un merge pour la partie des accords, cela a été fait pour éviter un son qui est trop fort comparé au début. De plus vous pouvez remarquer beaucoup de "click", pour éviter cela on aurait pu rajouter des fades à chaque note, mais à cause de la vitesse à laquelle les notes sont jouées on allait perdre le rythme de la partition. Le fichier qui contient la partition est contenue dans example.dj.oz, nous avons utiliser stretch pour adapter le ryhtme de la musique a un bpm équivalent de 100 beats par minute (tandis que la musique original est a 77bpm) et ensuite repeat pour répéter la partie iconique de la musique.