Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

МУЗИКА НА HASKELL

Текстова частина до курсової роботи

за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи

доцент Проценко В. С.

_____

(підпис)

“____” _____ 2020 р.

Виконав студент

Бабельська М. А.

“____” _____ 2020 р.

Київ 2020

ЗМІСТ

АНОТАЦІЯ 3

ВСТУП 4

РОЗДІЛ 1. HCODECS 5

1.1. Загальні відомості 5
1.2. Музичний приклад 6
1.3. Ефективне представлення музичної нотації 10
1.4. Висновок 11

РОЗДІЛ 2. EUTERPEA 12

2.1. Загальні відомості 12
2.2. Бібліотека Euterpea - HSoM API 13
2.2.1. Нотний рівень 13
2.2.2. Допоміжні функції 18
2.2.3. Інструменти в бібліотеці Euterpea 21
2.2.4. Музичний приклад 22
2.3. Висновок 24

РОЗДІЛ 3. MEZZO 26

3.1. Загальні відомості 26
3.2. Примітивні(базові) типи 27
3.3. Композиції 29
3.4. Створення мелодії 31
3.5. Створення Midi файлу 32
3.6. Набори правил 34
3.7. Приклад 35
3.8. Висновок 36

ВИСНОВКИ 37

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 39

АНОТАЦІЯ

- HCodecs

У цьому розділі описується створення музичного секвенсера за допомогою можливостей стандартної Haskell бібліотеки HCodecs, описуються усі типи, синоніми та функції, необхідні для подальшого компонування музики і її експорту в формат Midi файлу.

- Euterpea

У цьому розділі розглядаються основні можливості бібліотеки Euterpea, на основі матеріалу, доступного за підручником «Haskell School of Music» та відкритої документації Euterpea. На основі вивчених можливостей розглядається приклад практичної частини, а саме створення власної композиції засобами бібліотеки Euterpea та її відтворення.

- Mezzo

У цьому розділі розглядаються засоби та можливості відкритої бібліотеки для створення музики Mezzo. Під час розбору основних функцій та типі проводиться її порівняння з основною в даній курсовій бібліотекою Euterpea, проводяться паралелі та відмінності, переваги даної бібліотеки над Euterpea.

ВСТУП

Музика, як і комп’ютери, стала нероздільною частиною нашого повсякденного життя. На перший погляд може здатися, що ці дві речі абсолютно не пов’язані між собою. Але це не зовсім так. Музика, звісно, є результатом творчої діяльності і, так би мовити, походить від серця та душі, але не можна заперечувати, що вона має і строго визначені «правила».

Технології уже давно прийшли до того, щоб дозволити створювати та відігравати музику за допомогою комп’ютерних програм та власне комп’ютерних пристроїв. І саме ці «правила» та наперед відомі «конструкції», такі як ноти, паузи, акорди, допомогли це зробити.

Сьогодні не обов’язково бути музикантом чи мати якусь музичну освіту для того, щоб створювати власну музику і записувати композиції. Більше того, не обов’язково навіть мати «фізичний» музичний інструмент. Той, хто хоче спробувати себе в ролі музиканта чи композитора, може мати звичайний комп’ютер та відповідну музичну програму. Користуючись інструкціями та підказками на просторах інтернету, можна з нуля створити щось своє, нехай воно навіть не буде «ідеальним» одразу. А ті, хто бажає серйозніше заглибитися, можуть використати можливості доступних мов програмування та додаткових бібліотек, які надають необхідні засоби та інструменти для створення власної музики.

Для таких цілей чудово підійде мова програмування Haskell – стандартизована, винятково функціональна мова з нестрогою семантикою. Придумувати свій «велосипед» для представлення музики «з нуля» непотрібно – мова Haskell уже має багато доступних бібліотек, які реалізовують необхідний функціонал: потрібні типи, конструкції та функції. Від користувача вимагатиметься лише трішки розуміння функціонального програмування, вміння знайти необхідну інформацію в підручниках чи документаціях та бажання творити.

РОЗДІЛ 1. HCODECS

1.1.Загальні відомості

Бібліотека HCodecs надає у використання необхідні функції для читання, запису та маніпулювання файлами форматів MIDI, WAVE та SoundFont2. HCodecs повністю написана на Haskell і вміє ефективно розбирати та будувати бінарні дані, які зберігаються в ByteStrings. Бібліотека надає у використання декілька модулів та типів.

- Codec.ByteString.Builder - дозволяє ефективно будувати ліниві ByteStrings.
- Codec.ByteString.Parser – монада для побудови структур із закодованих ByteStrings.
- Codec.Midi – модуль для роботи з MIDI файлами.
- Codec.SoundFont - модуль для роботи з SoundFont2 файлами.
- Codec.Wav - модуль для роботи з WAVE файлами.
- Data.Audio – загальний тип для представлення аудіо даних.

Секвенсер – це пристрій або програма для запису, редагування та відтворення послідовностей MIDI даних. У цьому розділі ми розберемо спрощену реалізацію музичного секвенсера, написаного за допомогою бібліотеки HCodecs. Приклад базується на музичному прикладі з підручника Холом'єва «Підручник з Haskell»[1]. Для запису даних ми будемо використовувати формат MIDI файлу.

Формат MIDI файлу дозволяє копіювати MIDI дані з одного додатку на інший. Такі файли містять інформацію про розташування нот на нотному стані, висоту звуку, швидкість відтворення та певні контрольні сигнали для гучності, вступу, синхронізації, які синхронізують музичний темп на декількох пристроях. Також формат MIDI дозволяє вказувати інструменти, які будуть використовуватися. Такі файли використовуються пристроями, які підтримують MIDI, а також програмами для відтворення музики.

Загалом, формат MIDI розрахований на управління синтезатором в реальному часі. Цей формат представляє музику в низькорівневому нотному записі. Кожен рух виконавця кодується певною подією.

Так як формат MIDI – це формат, який можна розширювати та доповнювати, до нього можна додавати свої особливі налаштування. При цьому, якщо даний відтворювач не розуміє певного повідомлення(події), то воно просто ігнорується.

Отож, за допомогою стандартної Haskell бібліотеки HCodecs ми можемо створювати музику і перетворювати нотний запис у формат MIDI файлу. Опис музики нагадує опис самого MIDI-файлу. Щоб піднятися на рівень вище, потрібно ввести поняття ноти та їх композицію та описувати музику з їх допомогою.

1.2.Музичний приклад

Розглянутий секвенсер розуміє події натискання та відпускання клавіші, а також надає у використання декілька інструментів. Так як ми використовуємо формат MIDI, нам потрібен модуль Codec.Midi. В першу чергу, нам потрібні два конструктори:

- NoteOn – визначає подію натискання на клавішу.
- NoteOff – відповідно, визначає подію відпускання клавіші.

Обидва конструктори мають такі параметри: channel – канал, key – висота та velocity – рівень гучності. Ми можемо керувати одразу декількома генераторами тону. Для їх розподілення і потрібна властивість channel. Висота та гучність кодуються цілими числами в діапазоні від 0 до 127.

Грати ми будемо декількома інструментами, які в форматі MIDI називаються програмами. Відповідний інструмент на каналі можна встановити за допомогою конструктора ProgramChange. Це конструктор має два параметри:

channel – відповідно, визначає канал, та preset – число, яке вказує код інструмента.

Тепер розберемо, що ж таке MIDI файл. Він описується типом Midi і містить три параметри:

- fileType – тип файлу; може набувати наступних значень: SingleTrack – файл з одним треком, MultiTrack – файл з декількома треками, MultiPattern – файл, який містить групи треків(також називають - візерунки).
- timeDiv – тип, який кодує швидкість запису повідомлення.
- tracks – список подій з часовими відліками; у форматі MIDI час відраховується з моменту останньої події.

Ноти представляються як послідовність подій. Для цього у прикладі визначено власний тип Event, який містить такі параметри: eventStart – час початку події, eventDur – тривалість події та eventContent – зміст події.

```
data Event t a = Event {
        eventStart :: t,
        eventDur :: t,
        eventContent :: a
} deriving (Show, Eq)
```

Також у прикладі введено тип Track – тип, для позначення послідовності подій(trackEvents), які тривають деякий час(trackDur).

```
data Track t a = Track {
        trackDur :: t,
        trackEvents :: [Event t a]}
```

Наші події прив’язані до часу, отже дуже корисно мати операції, які будуть міняти розташування подій у часі. Для цього маємо визначені функції delayEvent – функція для затримки появи події, добавляє деяке число до часу появи самої події, та stretchEvent – функція для розтягування або пришвидшення тривалості події.

Наш приклад описує музику в MIDI, а отже ноти містять містити чотири основні параметри – висота, гучність, номер інструмента та додатковий параметр, який визначатиме, чи була нота зіграна на барабані(так як у MIDI ноти, зіграні ударними, опрацьовуються особливим способом). Тривалість ноти кодується в самій події.

```
data Note = Note {
        noteInstr :: Instr,
        noteVolume :: Volume,
        notePitch :: Pitch,
        isDrum :: Bool }
```

Для зручнішого відображення нот введено зручний синонім. Ця функція будує трек, який міститиме одну ноту.

```
note :: Int -> Score
note n = Track 1 [Event 0 1 (Note 0 64 (60+n) False)]
```

Крім того, існують визначені функції, які зміщують групу нот в іншу октаву. Як приклад, функція higher – приймає число октав, на які необхідно змістити вверх висоту у всіх нотах треку.

```
higher :: Int -> Score -> Score
higher n = fmap (\a -> a{ notePitch = 12*n + notePitch a })
```

Поки що наші ноти тривають одну одиницю часу. Але приклад описує та використовує й інші тривалості. Це реалізовано за допомогою функції stretch. Наприклад, bn = stretch 2 – тривалість ноти в дві одиниці часу.

Тепер нам потрібні операції, за допомогою яких ми можемо змінювати інструмент. Така можливість досить легко реалізована у прикладі за допомогою класу Functor. Наприклад,

```
instr :: Int -> Score -> Score
instr n = fmap $ \a -> a{ noteInstr = n, isDrum = False }
```

Як уже зазначалося вище, в форматі MIDI ударні інструменти опрацьовуються особливим способом. В цьому випадку інструмент кодується висотою звуку. Тому нам необхідно міняти параметр notePitch. Досить зручно мати визначений синоніми для створення нот, які граються на барабанах.

```
bam :: Int -> Score
bam n = Track 1 [Event 0 1 (Note 0 n 35 True)]
```

Використовуючи вищеописані функції та типи, ми можемо описати музику і складати мелодії. Проте ми ще не можемо цю мелодію інтерпретувати. Для цього нам необхідна функція, яка буде переводити події в формат MIDI. Ми реалізовуємо найпростіший приклад. Будемо вважати, що у нас всього 15 інструментів, а всі інші інструменти – це ударні. Також в нашій реалізації на одному каналі можна відтворювати лише один інструмент.

```
render :: Score -> Midi
render s = M.Midi M.SingleTrack (M.TicksPerBeat divisions) [toTrack s]
```

Крім цього, нам потрібно ще декілька допоміжних функцій.

Функція groupInstr розподілятиме нотний запис по інструментах. Вона буде повертати пару, де перший елемент міститиме список списків нот для неударних інструментів, а другий – список нот для ударних інструментів. Таку пару в набір MIDI-повідомлень ми будемо перетворювати за допомогою функції mergeInstr. Також нам потрібно мати функцію tfmTime. Ця функція необхідна для відсортовування подій у часі, щоб від абсолютних відліків у часі до відносних. Тепер ми можемо визначити допоміжну функцію toTrack.

```
toTrack :: Score -> M.Track M.Ticks
toTrack = addEndMsg . tfmTime . mergeInstr . groupInstr
```

Тепер ми можемо перевірити, що у нас вийшло. Для цього у нас буде функція out. Вона буде переводити нотний запис у значення типу Midi та записувати результат у файл. Також можемо вказати, щоб функція одразу запускала створений файл за допомогою програвача TiMidity.

```
out = (>> system "timidity res.mid") . exportFile "res.mid" . render
```

Як приклад, можемо запустити на відтворення ноту C(до) або ж мелодію.

```
out c

let x = line [c, hn e, hn e, low b, c]

out x
```

1.3.Ефективне представлення музичної нотації

Наша спрощена реалізація, яку ми розглянули, насправді не є дуже ефективною. Підручник[2] пропонує для цього покращене рішення. Для ефективнішого представлення тип Track визначається по-іншому.

```
data Track t a = Track {

        trackDur :: t,

        trackEvents :: TList t a }
```

Тип TList дозволяє швидко здійснювати об’єднання списків.

```
data TList t a = Empty | Single a | Append (TList t a) (TList t a) | TFun (Tfm t) (TList t a)
```

Така зміна типу Track необхідна для того, щоб ефективніше конвертувати події у часі. За це відповідає конструктор TFun. Він здійснює лінійну конвертацію списку у часі. Така конвертація кодується двома числами – масштаб та зміщення. Значення Tfm k d визначає лінійну функцію f(x) = kx + d.

1.4.Висновок

Використовуючи стандартну Haskell бібліотеку HCodecs розглянутий музичний приклад реалізовує досить простий секвенсер, який дозволяє записувати музику, відігравати її та створювати файли формату MIDI. Із цієї бібліотеки використано примітивні конструкції, такі як події та конвертацію подій у часі(зсуви, масштабування). З їх допомогою реалізовано власні функції для кращого представлення нот і роботи з ними.

Бібліотека HCodecs дозволяє представляти музику лише на низькорівневому нотному записі. Для того, щоб оперувати музикою на вищих рівнях, можна використовувати такі бібліотеки, як Haskore і Euterpea.

РОЗДІЛ 2. EUTERPEA

2.1.Загальні відомості

Haskell, як високорівнева мова програмування, дозволяє реалізовувати музику низько- та високорівневому записі. У бібліотеці Euterpea, щоб розрізнити ці два терміни, використовується нотний та сигнальний рівні.

На нотному рівні, найменшою одиницею, що використовується, є нота. Усе інше будується з використанням саме цього типу. Завдяки абстракції даних, функцій вищого порядку та декларативній семантиці, Haskell спрощує програмування на цьому рівні. Також, крім звичайного представлення музики, можна використовувати такі аспекти, як алгоритмічна композиція, використання фракталів, граматичних систем, стохастичних процесів.

На сигнальному рівні, усе фактично зводиться до створення звуку в комп’ютерній програмі. У цьому випадку найменшою одиницею є сигнал. Звук представляється як дискретна вибірка безперервного аудіо сигналу з досить високою швидкістю, який людське вухо фактично не може відрізнити від безперервного. Частота дискретизації зазвичай становить 44100 сигналів на секунду – стандартна частота дискретизації, яка використовується для компакт-дисків, mp3 файлів тощо. У бібліотеці Euterpea ці деталі приховані: сигнали трактуються абстрактно, як суцільні величини. На сигнальному рівні можна вивчати методи синтезу звуку – імітацію музичних інструментів або навіть створення власного «штучного» інструменту, обробку звуку та використання спеціальних ефектів, наприклад – реверберація, спотворення тощо.

Однією з найбільших різниць в програмуванні музики на нотному та сигнальному рівнях є кількість пам’яті, що використовується. Припустимо, ми граємо музику, використовуючи метроном, який виставлений на 96 ударів за хвилину. Це означає, що один біт триває 0.625 секунди. При частоті дискретизації 44100 ми отримуємо 55125 сигналів на секунду(2 * 0.625 * 44100).

Кожен сигнал зазвичай займає декілька байтів пам’яті. Це стандартний мінімум, необхідний для обчислень на сигнальному рівні. На противагу цьому, на нотному рівні нам необхідний тільки оператор або структура даних, яка відповідає за відтворення ноти і вимагає лише невеликої кількості байтів.

2.2.Бібліотека Euterpea - HSoM API

2.2.1. Нотний рівень

Книга «Haskell School of Music» досліджує основи комп'ютерної музики та функціонального програмування через мову програмування Haskell. Ця книга дозволяє програмістам взаємодіяти з музикою, використовуючи знайомий їм носій. Читачі дізнаються, як використовувати бібліотеку Euterpea для Haskell[4] для представлення та створення власної музики з кодом, не потребуючи іншого музичного програмного забезпечення. Автором є Пол Реймонд Худак - американський професор інформатики в Єльському університеті, відомий своєю участю у розробці мови програмування Haskell, та хороший джазовий музикант.

В музичній теорії, нота – це поєднання висоти та тривалості. Тривалість вимірюється в бітах. В бібліотеці Euterpea тривалість має тип Dur. Один біт позначає тривалість цілої ноти(whole note), відповідно – тривалість ½ позначає половину ноти(half note) і т.д. Нота є найменшою сутністю в бібліотеці Euterpea, яка є відтворюваною і має тип Music Pitch. Паралельно з нею базовою сутністю вважається пауза – rest – фактично, нота без висоти. Ці дві сутності визначаються так:

note :: Dur → Pitch → Music Pitch

rest :: Dur → Music Pitch

Таким чином, note d p – це нота з тривалістю d і вистою p, a rest d – це пауза з тривалістю d. Наприклад, note (1/4) (A, 4) – це нота A в четвертій октаві з тривалістю ¼. Результуючий тип, який матимуть сутності note та rest – це Music Pitch.

Дуже зручно використовувати стандартні типи даних у Haskell для створення власних синонімів, щоб представити потрібну концепцію. Так можна використати тип Int для представлення октави та Rational для представлення тривалості ноти. В бібліотеці Euterpea за умовою октава 4 відповідає октаві, що на піаніно містить середню ноту C(до). Таким чином самі концепції октави та тривалості можна виразити як:

```
type Octave = Int
type Dur = Rational
```

Варто пам’ятати, що синонім типу не створює новий тип даних. Він лише дає нове ім’я для уже існуючого типу, за яким нам буде зручніше звертатися і позначати необхідну концепцію. Синоніми можна визначати не лише для примітивів, таких як Int, але й для складніших структур, наприклад, пар. Так як в музичній теорії висота визначається як пара, що складається з тону та октави, синонім Pitch (висота) дуже зручно представити як:

```
type Pitch = (PitchClass, Octave)
```

Для того, щоб повністю зрозуміти, що собою являє сутність note, варто показати, як представляється тип PitchClass. Використовувати цілі числа для цього - не найбільш елегантний варіант. Найкраще представити тони так, щоб вони нагадували звичайні назви класів висоти – C, C#, D і т.д. Haskell дозволяє зробити це, використавши алгебраїчні типи даних. Таким чином ми отримаємо:

```
data PitchClass = Cff | Cf | C | …
```

Крім цього, ми також можемо визначити сутності note та rest, об’єднавши їх в один тип. Ноту і паузу можна розглядати як примітивні типи в музиці, тому в бібліотеці Euterpea вони описані наступним чином:

```
data Primitive = Note Dur Pitch | Rest Dur
```

Незважаючи на те, що Note і Rest є конструкторами даних, вони все ще є функціями, а отже, мають тип. А оскільки вони мають змінні типу, вони є прикладами поліморфних функцій.

Тепер, коли ми маємо примітивні типи та необхідні синоніми, нам потрібно представити тип, який дозволить нам поєднувати ці примітиви один з одним і створювати більші композиції. Для цього бібліотека Euterpea визначає поліморфний тип даних Music, який визначає фундаментальну структуру музичної сутності на рівні нот:

```
data Music a = Prim (Primitive a) | Music a :+: Music a | Music a :=: Music a
| Modify Control (Music a)
```

Оголошення типу Music фактично означає, що цей тип може мати одну з чотирьох наведених форм:

1. Prim p, де p – це деяке примітивне значення типу Primitive. Наприклад, Prim (Note qn (A, 4)) – нота A в четвертій октаві з тривалістю qn.

2. m1 :+: m2. Оператор :+: відповідає за послідовне відтворення. У випадку note (1/4) (A, 4) :+: note (1/2) (C, 4) ми отримаємо послідовне виконання ноти A і C.

3. m1 :=: m2. Оператор :=: відповідає за паралельне відтворення. Таким чином, запис note (1/4) (A, 4) :=: note (1/2) (C, 4) дозволить паралельно(одночасно) відтворити ноти A і C. Результуюча тривалість композиції двох нот – це довша із тривалостей m1 та m2.

4. Modify cntrl m – модифікатор для типу Music. Як саме буде модифіковано композицію, визначає параметр cntrl, який має тип Control.

Конструктор Modify використовує тип даних Control, щоб змінювати значення Music. Наприклад, ми можемо модифікувати гучність, тривалість та змінювати інструмент, яким буде відтворено наші композиції. Наведемо декілька прикладів.

Припустимо, нам потрібно відтворити деяку мелодію так, щоб її гучність поступово зменшувалася, тим самим створюючи ефект «затухання». З допомогою Modify це можна зробити, використавши один із конструкторів типу

Control, а саме Phrase. Цей конструктор приймає деякий набір атрибутів і модифікує мелодію відповідно до вказаних значень. Наприклад:

melody1 = Modify (Phrase [Dyn $ Diminuendo 0.8]) $ addVolume 100 melody

За допомогою функції addVolume ми вказуємо початкове значення 100 для нашої композиції melody. Конструктор Phrase приймає атрибути Dyn та Diminuendo. Атрибут Dyn відповідає за регуляцію гучності композиції, а Diminuendo – за її зменшення. В музичній теорії термін diminuendo буквально означає «зменшується» і є ознакою поступового зменшення гучності музики. Таким чином, ми отримаємо модифіковане звучання нашої композиції melody, гучність якої буде поступово зменшуватися під час відтворення. Зворотній ефект, тобто поступове підвищення гучності, можна отримавши, замінивши атрибут Diminuendo на Crescendo.

melody2 = Modify (Phrase [Dyn $ Crescendo 2.0]) $ addVolume 50 melody

Ми також можемо «на ходу» змінювати і сам темп музики, прискорюючи чи сповільнюючи мелодію. Найпростіший варіант, це використати конструктор Tempo типу Control - Modify (Tempo 2) melody. Таким чином, ми отримаємо удвічі швидше виконання нашої композиції melody. Але інколи нам необхідно використовувати більш елегантні модифікації темпу з певними «ефектами».

В музиці часто використовується прийом поступового збільшення/зменшення темпу мелодії. В бібліотеці Euterpea це також можна зробити з допомогою конструктора Phrase з використанням атрибутів Tmp, який відповідає за темп мелодії, та Ritardando або Accelerando – поступове зменшення та збільшення темпу відповідно. Як приклад:

melody3 = Modify (Phrase [Tmp $ Ritardando 0.5]) melody

В музиці є стандартні значення тривалості ноти, і за допомогою визначеного синоніму типу Dur в бібліотеці Euterpea визначені конкретні функції тривалості.

```
bn, wn, hn, qn … :: Dur
bnr, wnr , hnr , qnr … :: Music Pitch
bn = 2; bnr = rest bn …
wn = 1; wnr = rest wn …
```

Так само, на основі типу даних Note визначено функції, які відповідають за кожну ноту. Такий запис значно спрощує використання примітивів і робить код простішим і зрозумілішим для читання.

```
cff , cf , c … :: Octave → Dur → Music Pitch
cf o d = note d (Cf , o)
```

Інколи може бути необхідно поводитися з висотами, не як з синонімами типів, а як зі звичайними цілими числами. Для цього в бібліотеці Euterpea визначено синонім типу AbsPitch, який визначає «абсолютну висоту».

```
type AbsPitch = Int
```

Математично абсолютну висоту можна визначити так: значення октави множиться на 12 і до отриманого результату додаємо індекс тону(класу висоти). В Haskell це можна виразити наступним чином:

```
absPitch :: Pitch → AbsPitch
absPitch (pc, oct) = 12 ∗ oct + pcToInt pc,
```

де pcToInt - це функція, яка перетворює певний тон(клас висоти) в індекс.

Відповідно, ми маємо зворотну функцію, яка конвертує абсолютну висоту в висоту. Але таке перетворення має свої складнощі через енгармонічні еквіваленти. Наприклад, абсолютна висота 15 може відповідати як і висоті (Ds, 1), так і висоті (Ef, 1). В таких випадках, бібліотека Euterpea застосовує наступний підхід: завжди повертати дієз(тобто D#, або ж Ds) в таких неоднозначних випадках.

```
pitch :: AbsPitch → Pitch
```

```
pitch ap =
let (oct , n) = divMod ap 12
in ([C, Cs,D, Ds,E, F, Fs,G, Gs,A, As,B ] !! n, oct )
```

Маючи тепер функції absPitch та pitch, можна легко визначити функцію trans, яка буде зміщувати висоту на вказане значення.

```
trans :: Int → Pitch → Pitch
trans i p = pitch (absPitch p + i )
```

2.2.2. Допоміжні функції

В попередньому розділі ми розглядали такі оператори як :+: та :=:. Для спрощення запису композицій в бібліотеці Euterpea є альтернативні наведеним операторам функції. Одна з них – це функція line. Функція line має наступний тип: line :: [Music a] -> Music a. Таким чином, на вхід ми подаємо певний список нот і отримуємо одну суцільну композицію на виході. Функція line є рекурсивною, а також поліморфною, і визначається наступним чином:

```
line :: [Music a ] → Music a
line [ ] = rest 0
line (m : ms) = m :+: line ms
```

Очевидно, що функція line є аналогічною оператору :+:. Подібна ж функція існує і для оператора :=:. Це функція chord, яка, відповідно, також є рекурсивною та поліморфною.

```
chord :: [Music a ] → Music a
chord [ ] = rest 0
chord (m : ms) = m :=: chord ms
```

Така реалізація, звісно, має місце. Але врахувавши те, що ми працюємо з мовою Haskell, яка надає багато допоміжних і ефективних функцій для роботи зі списками, функції line та chord можуть бути визначенні, використовуючи стандартну функцію fold. Таким чином, ми отримаємо наступні визначення:

```
line ms = fold (:+:) (rest 0) ms

chord ms = fold (:=:) (rest 0) ms
```

Інколи необхідно затримати відтворення деякої ноти. Для таких випадків в бібліотеці Euterpea реалізовано функцію offset. Ця функція приймає на вхід деяку ноту та бажану тривалість затримки і просто «ставить» паузу(rest) перед цією нотою: offset d m = offset d :+: m.

Також корисною функцією є функція times. Вона дозволяє повторювати задану мелодію вказану кількість разів.

```
times :: Int → Music a → Music a

times 0 m = rest 0

times n m = m :+: times (n − 1) m
```

Цікавим є те, як несувора семантика Haskell дозволяє визначити нескінченні музичні значення. Наприклад, постійне відтворення деякої мелодії можна визначити, використовуючи цю просту функцію:

```
forever :: Music a → Music a

forever m = m :+: repeat m
```

Так, наприклад, нескінченний остинато - прийом, заснований на багаторазовому повторенні в музичному творі будь-якої мелодії або окремого звуку, можна виразити таким чином, а потім використовувати в різних контекстах, які автоматично витягують лише ту частину, яка фактично потрібна.

Поняття інверсії, ретрогради(зворотного відтворення), ретроградної інверсії тощо, що використовуються у музиці, також визначені в бібліотеці

Euterpea. Зазвичай, ці функції застосовуються до рядків нот, тобто до мелодії. Ретрограда – це просто зворотне відтворення ноти. Інверсія - перестановка елементів зверху вниз в інтервалі, акорді або мелодії. Інверсія рядку відбувається відносно заданої висоти(за замовчування, як правило, це перша висота), де інтервали між послідовними висотами перевернуті. Якщо абсолютна висота першої ноти дорівнює ap, то кожна висота p перетворюється в абсолютну висоту зі значенням ap - (absPitch p - ap), або ж - 2 ∗ ap - absPitch p.

Щоб зробити це в Haskell, потрібно мати функцію, яка буде перетворювати рядок, створений за допомогою line, в список.

```
lineToList :: Music a → [Music a ]

lineToList (Prim (Rest 0)) = [ ]

lineToList (n :+: ns) =n : lineToList ns
```

Маючи цю функцію, можна визначити функцію інверсії:

```
invert :: Music Pitch → Music Pitch

invert m = let l@(Prim (Note r) : ) = lineToList m

           inv (Prim (Note d p)) =

           note d (pitch (2 ∗ absPitch r − absPitch p))

           inv (Prim (Rest d)) = rest d

       in line (map inv l)
```

Тепер, з функціями lineToList та invert ми легко визначаємо решту функцій, таких як ретрограда, ретроградну інверсію і т.п. за допомогою композиції функцій – стандартного прийому в Haskell, де результат виходу однієї функції застосовується в якості входу для іншої функції.

```
retro, retroInvert , invertRetro :: Music Pitch → Music Pitch

retro = line . reverse . lineToList
```

```
retroInvert = retro . invert

invertRetro = invert . retro
```

2.2.3. Інструменти в бібліотеці Euterpea

За замовчуванням, в бібліотеці Euterpea для відтворення нот та мелодій використовується інструмент піаніно. Але, очевидно, хотілося б мати можливість імітувати й інші інструменти. Це можна робити за допомогою уже згаданого конструктора Modify, використавши один із конструкторів типу Control, а саме Instrument. Бібліотека Euterpea надає можливість імітації великої кількості інструментів, серед них рояль, віолончель, скрипка, тромбон, флейта, акордеон, губна гармоніка та багато інших. Єдине, на що варто звернутися особливу увагу, так це те, що ударні інструменти застосовуються по особливому. Тому їх будемо розглядати окремо.

Наведемо приклад модифікації мелодії, для того щоб відтворити її, припустимо, за допомогою скрипки. Це робиться дуже просто: Modify (Instrument Violin) melody.

Ударні інструменти – це досить складне поняття для абстрактного представлення. З одного боку, ударний інструмент – це просто інший інструмент, чому ж тоді його використання відрізняється від звичайних? А з іншого боку, навіть звичайна практика позначення трактує його у спеціальний спосіб, не зважаючи на те, що таке позначення має багато спільного з неударним позначенням. Формат Midi так само неоднозначно поводиться з ударними інструментами: з одного боку, ударний звук вибирається шляхом визначення октави та висоти, як і будь-який інший інструмент; але з іншого боку, ці висоти не мають жодного тонального значення, вони є лише зручним способом для вибору з великої кількості звуків ударних.

Дійсно, частина Загального стандарту Midi(General MIDI Standard) - це набір імен для часто використовуваних звуків ударних. Отже, все, що залишається зробити - це спосіб перетворити ці звуки ударних у значення мелодії, тобто в ноти:

```
perc :: PercussionSound → Dur → Music Pitch

perc ps dur = note dur (pitch (fromEnum ps + 35))
```

Наведемо приклад використання ударних. Визначимо дві ноти – b та s, які будуть відповідати за відтворення стандартних в Euterpea ударних інструментів BassDrum1 та AcousticSnare відповідно з тривалістю en, або ж 1/8. Тоді ми застосуємо функцію instrument до списку нот, яка приймає значення інструмента і за допомогою конструктора Modify модифікує значення інструмента, який буде відтворювати задану мелодію. Результат буде мати тип [Music Pitch], тобто новий список нот.

```
kicks = let b = perc BassDrum1 en
            s = perc AcousticSnare en
        in  map (instrument Percussion) [b, rest en, s, rest en, rest en, b, s, rest en]
```

2.2.4. Музичний приклад

Розглянемо приклад коду з практичної частини, присвяченої бібліотеці Euterpea. У нас є музична композиція, розрахована на три інструменти – скрипка, віолончель та акустичне фортепіано. Крім цього, в композиції ми маємо два акустичних фортепіано – одне відіграє свою мелодію в скрипковому ключі (the treble clef), інше – в басовому ключі (the bass clef). Таким чином, можна вважати, що ми маємо справу з чотирма інструментами.

Кожний ряд ми будемо відігравати паралельно, відповідно до нотного запису.

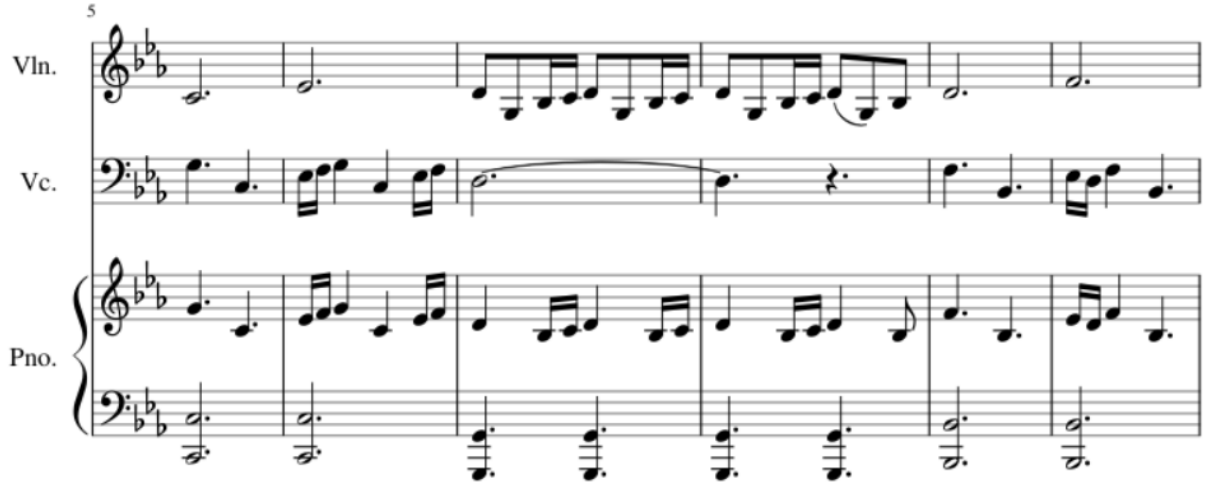

Рисунок 1.1

Наведемо приклад запису вищенаведеної композиції(рис. 1.1) засобами бібліотеки Euterpea. Функції vSound, cSound, ptcSound та pbcSound приймають номер ряду, який потрібно відтворити, і повертають його. Для спрощення, наведемо приклад цих функцій, які просто повертають один ряд.

Ряд для скрипки:

vSound n = line [c 3 dhn, ef 3 dhn] :+: times 3 (line [d 3 en, g 2 en, bf 3 sn, c 3 sn]) :+: line [d 3 en, g 2 en, bf 3 en, d 3 dhn, f 3 dhn]

Ряд для віолончелі:

cSound n = line [g 3 dqn, c 3 dqn, ef 3 sn, f 3 sn, g 3 qn, c 3 qn, ef 3 sn, f 3 sn, d 3 dhn, d 3 dqn, dqnr, f 3 dqn, bf 3 dqn, e 3 sn, d 3 sn, f 3 qn, bf 3 dqn]

Ряд для акустичного піаніно (скрипковий ключ):

ptcSound n = line [g 4 dqn, c 4 dqn, ef 4 sn, f 4 sn, g 4 qn, c 4 qn, ef 4 sn, f 4 sn, d 4 qn, bf 4 sn, c 4 sn, d 4 qn, bf 4 sn, c 4 sn, d 4 qn, bf 4 sn, c 4 sn, d 4 qn, bf 4 en, f 4 dqn, bf 4 dqn, ef 4 sn, d 4 sn, f 4 qn, bf 4 dqn]

Ряд для акустичного піаніно (басовий ключ):

pbcSound n = times 2 (chord [c 3 dhn, a 4 dhn]) :+: times 4 (chord [g 3 dqn, g 4 dqn]) :+: times 2 (chord [bf 3 dhn, bf 4 dhn])

Усі чотири ряди ми повинні виконати паралельно. Врахувавши те, що й усі наступні ряди повинні бути відтворені наступним чином, ми створимо додаткові функції: playInstr - буде відтворювати мелодію відповідним інструментом, playLine - буде відтворювати уже модифіковані рядки паралельно.

```
playInstr instr = Modify (Instrument instr)

playLine n = playInstr Violin (changeLoudness 80 (vSound n)) :=:
        playInstr Viola (changeLoudness 70 (cSound n)) :=:
        playInstr AcousticGrandPiano (ptcSound n) :=:
        playInstr AcousticGrandPiano (pbcSound n)
```

Мелодії в бібліотеці Euterpea мають встановлену стандартну гучність. Ми хочемо модифікувати гучність для інструментів скрипки та віолончелі, щоб вони звучали тихіше на фоні фортепіано. Це ми робимо за допомогою додаткової функції changeLoudness, яка приймає відповідну гучність і мелодію та модифікує її за допомогою розглянутого у попередніх розділах конструктора Phrase.

```
changeLoudness l s = (Modify (Phrase [Dyn (Loudness l)]) s)
```

Тепер, щоб запустити відтворення нашої мелодії, створимо функції part та main. Перша створює паралельну композицію наших рядків, а друга, за допомогою команди play, відтворює їх. Нам залишиться лише запустити наш модуль у терміналі та ввести команду main.

```
part = playLine 2

main = play $ part
```

2.3.Висновок

Бібліотека Euterpea - це кросплатформенна доменна мова для програмування комп’ютерних музичних програм, вбудована в мову

програмування Haskell. Euterpea - мова широкого спектра, підходить для представлення музики високого рівня, алгоритмічної композиції, аналізу музики, роботи з MIDI, низькорівневої обробки звуку, синтезу звуку та дизайну віртуальних інструментів. Вона надає у використання усі необхідні інструменти для зручної роботи з музикою і є легкою в освоєнні.

Ми розглянули основні можливості бібліотеки Euterpea за допомогою підручника «Haskell School of Music» та відкритої документації Euterpea, а також розібрали розроблений на основі отриманих знань приклад створення музики з використанням різних інструментів та модифікацій мелодії.

РОЗДІЛ 3. MEZZO

3.1.Загальні відомості

Mezzo - це бібліотека, написана на Haskell, і вбудована доменна мова для опису музики. Особливість цієї бібліотеки полягає у тому, що вона може виконувати різні правила музичної композиції статично, тобто під час компіляції. А це означає, що якщо ви напишете «погану» музику, тобто таку, що не підпадає під якесь музичне правило, то ваша композиція не буде компілюватися. Це можна вважати за дуже сувору перевірку музичної орфографії. Крім того, Mezzo є бібліотекою лише з кількома залежностями. Її можна легко встановити за допомогою Cabal – системи побудови та управління пакетами бібліотек та програм в Haskell.

Мови опису музики - це текстові зображення музичних творів, які використовуються для введення нот або транскрипції. Більшість таких мов надають різні способи введення нот і пауз та способів їх поєднання в мелодії та акорди. Бібліотека Mezzo додатково дозволяє користувачам вводити акорди в їх символічному позначенні, так само як і послідовності акордів використовуючи схематичний опис функціональної гармонії.

Бібліотека Mezzo надає два способи створення музичних значень.

- Літеральний стиль. Використовуючи літеральний стиль ми можемо створювати музичні значення за допомогою явних конструкторів літеральних значень. Наприклад, нота C в четвертій октаві записується як pitch _c _na _o4, і такий запис можна скоротити до _cn. Чверть ноти (висота з тривалістю) в такому випадку може бути записана як noteP (pitch _c _na _o4) _qu або noteP _cn _qu. Літеральні значення мають префікс із підкресленням і їх можна комбінувати за допомогою конструкторів pitch та noteP.

- Стиль «будівельника» (Builder style). Цей стиль можна вважати кращим за літеральний стиль, тому що він більш стислий, гнучкий і читабельний. За

допомогою цього стилю музичні значення створюються шляхом послідовного запису їх атрибутів зліва направо. Як приклад, запис ноти чверті ноти C буде виглядати як c qn. Якщо ж необхідно записати шістнадцяткову ноту F# з крапкою у 5-й октаві, то ми зможемо зробити це наступним чином - fs sn' або f sharp sn'. Подібно до цього, запис мажор акорду ноти C буде виглядати як c maj qc. Такий запис досить подібний до нотного запису в бібліотеці Euterpea, яку ми розглядали в попередньому розділі.

У наступному розділі ми будемо розглядати основні можливості бібліотеки Mezzo використовуючи саме стиль «будівельника» (Builder style).

3.2.Примітивні(базові) типи

В бібліотеці Mezzo до базових типів відноситься нота, пауза та акорд. Подібно як і в бібліотеці Euterpea, нота визначається певною висотою та тривалістю. В стилі «будівельника» (Builder style) висота має явне значення, яке складається з трьох частин:

- Клас висоти(тон). Один із c, d, e, f, g, a або b. Визначає позицію ноти в октаві, або ж, якщо провести паралель, білу клавішу на піаніно.

- Альтерація. Це суфікс для класу висоти(тону) і може бути або f - дієз(#), наприклад, bf qn, або s - бемоль(♭), наприклад, fs qn. Позначення для натуральних альтерацій не визначені, тому c означає натуральну ноту C. Альтерації також можна записувати як окремі атрибути, наприклад, c sharp qn, і навіть повторювати, наприклад, подвійний дієз можна записати як c sharp sharp qn або cs sharp qn.

- Октава. Остання частина значення ноти. За замовчування використовується 4 октава і тому вона не позначається. Нижчі октави позначаються наступним чином: _ (одне нижнє підкреслення - 1 октава), __ (подвійне нижнє підкреслення – 2 октава), _3 (3 октава), _4 (4 октава) і _5 (5

октава). Для позначення вищих октав використовується ' (апостроф): ', '', '3 та '4. Так, наприклад, нота C в 2-й октаві записується як c__ qn, а нота B бемоль в 7-й октаві як bf'3 qn.

Тривалість ноти записується після висоти. Для них значенням тривалості є перша літера назви тривалості(з англ. eighth, quarter і т.д.) за якою йде n, наприклад, чверть ноти позначається як qn. Також є позначення тривалостей і для нот з крапкою. В музичній теорії, нота з крапкою – це нота, яка подовжується на половину вказаної тривалості. Позначається така тривалість додаванням ' (апостроф) до стандартного позначення тривалості, наприклад, qn'.

Позначення пауз подібне до позначень для нот. Але замість висоти записується r, а у випадку тривалості – n заміняється на r. Наприклад, чверть паузи буде позначатися як r qr, а ціла пауза з крапкою як r wr'.

Третій базовий тип – акорд – визначається наступними параметрами: корінь (висота), тип, інверсія та тривалість. Наприклад, мажорна тріада чверті ноти C в 5-й інверсії запишеться як c maj inv qc.

- Корінь. Визначається так само, як і в ноті, наприклад, c, af'' і т.п.

- Тип. Означає один з трьох класів акордів: діада – гармонічний інтервал, який складається з двох нот, буває 5-ти типів, тріада – це три звуки, розміщені по різних терціях, та тетрада – чотири звуки, розміщені по різних терціях.

- Інверсія. Усі позначення типів акордів можуть супроводжуватися символом ' і окремим атрибутом i0, i1, i2 або i3. Такий запис визначатиме нульову, першу, другу або ж третю інверсії, наприклад, c maj' i2 qc. Крім того, атрибут інверсії може бути доданий до акорду один або ж декілька разів (відповідно, щоб застосувати декілька інверсій), наприклад, c maj inv qc.

Аналогічно до нотного запису та запису паузи, тривалість акорду закінчується з c(з англ. chord) і, відповідно, акорд теж може бути з крапкою: c min7 qc, f sharp hdim inv wc', де тривалості – це qc та wc'.

Крім цього, можна використовувати прийом дублювання. Так, наприклад, діади та тріади можуть бути подвоєні, для того щоб виразити ними тріади та тетради відповідно. Це можна зробити, повторивши нижню ноту на одну октаву вище. Дублювання позначається додаванням D до кінця типу акорду. Наприклад, fifthD, majD або augD.

3.3.Композиції

Подібно до бібліотеки Euterpea, в бібліотеці Mezzo є два способи для створення композиції музики – послідовна (мелодійна) та паралельна (гармонічна) композиції. Крім цього, Mezzo надає досить зручний спосіб запису мелодій.

Паралельна (гармонічна) композиція відіграє два фрагменти музики одночасно. За паралельне відтворення відповідає оператор :-:. Як приклад, зіграти чверть нот G, E та C паралельно можна в наступний спосіб:

```
g qn :-: e qn :-: c qn
```

Для збереження послідовності, фрагменти музики повинні бути складені від найнижчої ноти до найвищої. Саме тому вищенаведений приклад буде відтворювати мажорну тріаду C. Фрагментами композиції можуть бути будь-які музичні композиції, допоки тривалості кожного фрагменту є однаковими. Якщо ж тривалості музичних композицій не збігаються, найкоротша композиція повинна бути доповнена паузами там, де необхідно.

Використовуючи оператор :-:, гармонічні інтервали перевіряються на правильність, саме тому ця операція в першу чергу призначена для контрапунктних композицій – композицій, в яких більше, ніж одна музична лінія відіграється в один і той же час.

Для випадку, коли необхідно створити гомофонну мелодію – мелодія, де є лише одна головна лінія, а інші – це акомпанементи, краще використовувати функцію hom, яка не враховує правил гармонічного руху.

```
(g qn :|: a qn) `hom` (c qn :|: d qn)
```

Послідовна (мелодійна) композиція може бути застосована за допомогою оператора :|:. Очевидно, мелодійна композиція буде відігравати фрагменти музики один за одним:

```
c qn :|: d qn :|: e qn
```

Враховуючи сувору перевірку на синтаксис в бібліотеці Mezzo, фрагменти музики в мелодійній композиції повинні не тільки бути нотами або одиничними голосами, але й повинні мати однакову кількість голосів. Як приклад, наступний запис

```
c qn :|: c maj qc
```

не вдасться скомпілювати, так як перший фрагмент – це одна нота з одним голосом, а другий фрагмент – це акорд з трьома голосами. Це можна виправити явно додавши до першого фрагменту паузи або додавши тихі голоси за допомогою функцій pad, pad2, pad3 або pad4:

```
pad2 (c qn) :|: c maj qc
```

Такий запис додає порожні голоси нижче існуючих голосів. Але у деяких випадках (наприклад, з контрапунктними композиціями), може виникнути потреба зберегти верхній та нижній голоси, а середній зробити тихим. В такому випадку, можна використати функцію restWhile для того, щоб записати голос однакової довжини як аргумент, але без будь-яких нот.

```
comp = (top :-: restWhile top :-: bottom) :|: c maj qc
```

Вищенаведений приклад також показує, як мелодійна і гармонічна композиції працюють разом. Оскільки це лише комбінування музичних значень, немає ніяких обмежень щодо порядку та вкладення операторів.

3.4.Створення мелодії

Створення довгої мелодії, яка складається з послідовності нот та пауз, може бути тривалим і повторюваним процесом. Одна з проблем – це те, що зміна тривалості нот не є дуже частим явищем, тобто ми можемо мати довгі рядки з нот з однаковою тривалістю. Не зважаючи на це, ми все ще вказуємо тривалість для кожної ноти окремо:

```
c qn :|: c en :|: d en :|: ef en :|: d en :|: c en :|: b_ en :|: c hn
```

На відміну від бібліотека Euterpea, де такий підхід є єдиним рішенням для запису послідовностей, бібліотека Mezzo надає більш стислий спосіб для створення мелодії, де лише зміни тривалості є явними:

```
start $ melody :| c :< c :| d :| ef :| d :| c :| b_ :> c :| c
```

Мелодії (melody) – це фактично перелік нот з конструкторами, які вказують на тривалість наступної ноти. Усі мелодії повинні починатися з ключового слова melody. Це ключове слово ініціалізує мелодію і встановлює тривалість «за замовчуванням» як чверть ноти. Перетворити мелодію на відтворюване значення музики можна за допомогою функції start. Для створення мелодії використовуються наступні конструктори:

- (:|) . Конструктор (:|) означає, що наступна нота матиме таку ж тривалість, як і попередня. Наприклад, melody :| c :| d :| e створить мелодію з трьох чверть нот, так як melody ініціалізує початкову тривалість в чверть ноти.

- (:<<<), (:<<), (:<), (:^), (:>) та (:>>). Вказані конструктори позначають тривалість ноти в 1/32, 1/16, 1/8, ¼, ½ та 1 відповідно.

- (:~|). Конструктор (:~|) використовується для пауз. Він позначає, що наступна пауза матиме таку ж тривалість, як і попереднє значення, незалежно від того, нота це чи пауза.

- (:~<<<), (:~<<), (:~<), (:~^), (:~>) та (:~>>). Вказані конструктори також використовуються лише для пауз і позначають паузи в 1/32, 1/16, 1/8, ¼, ½ та 1 відповідно.

- Усі конструктори, які змінюють тривалість (окрім (:<<<) та (:~<<<)) можуть супроводжуватися . (крапкою) вкінці, для того щоб позначити тривалість ноти з крапкою або паузи з крапкою. Наприклад, запис мелодії melody :^ c :^. d :> e позначає мелодію, загальної тривалості в чверть ноти, чверть ноти з крапкою та півноти.

3.5.Створення Midi файлу

Значення музики (Music) не може бути легко експортоване в Midi формат, адже є багато атрибутів, які потрібно спершу визначити (зокрема, набір правил, які використовуються для перевірки правильності композиції, без якої фрагменти не складатимуться). Це можна досягти, створивши партитуру (Score) з музичного значення, і визначивши атрибути, такі як темп, визначення часу та набір правил. Партитуру також можна використовувати для розбиття великої музичної композиції на менші фрагменти, які будуть заздалегідь перевірені. Такий підхід значно скорочує час на перевірку коректності, адже коректність кожного фрагменту перевіряється незалежно. Це також дозволяє зміну темпу або ключів, оскільки кожна секція має свій власний набір атрибутів.

Партитури створюються, використовуючи наступний синтаксис:

score [<attribute_name> <attribute_value>]* withMusic <music>

Ключове слово score починає створювати партитуру, за яким має бути ключове слово withMusic без або з декількома необов’язковими атрибутами між ними. Атрибути score це звичайний набір пар «ім’я – значення» написаних один за одним без жодних знаків пунктуації. На даний момент Mezzo підтримує наступні атрибути:

- Атрибут section. Короткий опис секції (фрагменту музики), який описує партитура. Це атрибут загалом використовується лише для документації, щоб у вихідному коді було зрозуміло, де яка секція знаходиться.

- Атрибут setKeySig. Ключове позначення партитури, наприклад c_maj, a_min, fs_maj, bf_min.

- Атрибут setTimeSig. Позначення часу партитури, може набувати одне з трьох значень: duple, triple або quadruple. Це впливає на кількість акордів, які відіграються.

- Атрибут setTempo. Визначає темп секції у бітах за хвилину. Це впливає на темп відтворення Midi файлу.

- Атрибут setRuleSet. Набір правил, призначений для перевірки коректності секції. Може набувати значень free, classical, strict або будь-яке інше визначене користувачем правило.

Ключове слово withMusic закінчує створення партитури з її аргументами. Якщо жодні атрибути не змінюються, тобто використовуються стандартні атрибути score, функція defScore m може бути використана замість score withMusic m для створення партитури зі фрагменту m.

Тепер, коли ми створили партитуру з музичної композиції, ми можемо експортувати її в Midi файл, використовуючи функцію renderScore. Ця функція має наступний тип:

```
renderScore :: FilePath -> Title -> Score -> IO ()
```

Ця функція приймає на вхід шлях до файлу (визначається як стрічка - String), назву (Title) Midi треку (також стрічка) і партитуру (Score) для експорту. Після успішного перетворення, ми отримаємо повідомлення «Composition rendered to <filepath>». Якщо музичний фрагмент розбитий на декілька дрібніших партитур (секцій), всі вони можуть бути з’єднані і перетворені за допомогою функції renderScores.

```
renderScores :: FilePath -> Title -> [Score] -> IO ()
```

Ця функція схожа на функцію renderScores, але замість однієї партитури (Score) вона приймає список партитур (в довільному порядку). Варто замітити, що це дозволяє повторно використовувати секції. Наприклад, рефрен – повторювана секція, має бути описаний лише раз, і партитура, створена з нього, може бути повторно використана без додаткових перевірок на коректність.

Композиції Mezzo можна відтворити наживо, у терміналі, використовуючи функцію playLive'.

```
playLive' :: Score -> IO ()
```

Крім цього, можна використовувати запис типу playLive = playLive' . defScore для того, щоб одразу відтворювати музику.

3.6.Набори правил

Різні типи жанрів вимагають різних наборів правил для перевірки коректності музичного твору. Бібліотека Mezzo надає користувачу можливість обирати різні рівні строгості (strictness), які називаються наборами правил (rule sets). Крім того, користувач може частково або навіть повністю змінювати на власний розсуд обраний набір правил. Зокрема, можна налаштувати такий набір правил, який вимкне будь-яку перевірку на коректність, тим самим дозволяючи повну свободу для творчості.

В бібліотеці Mezzo є три наперед визначених набори правил з різними рівнями строгості. Набір правил визначається атрибутом setRuleSet в партитурі (score), яку ми розглядали в попередньому розділі.

- Вільний набір правил (The free rule set). Цей набір не застосовує жодних музичних правил, тому спосіб запису музики не обмежується.

- Класичний набір правил (The classical rule set). Він застосовує загальні правила класичної музики, не надто обмежуючи її.

- Строгий набір правил (The strict rule set). Застосовує більшість правил суворих контрапунктних композицій, які часто ґрунтуються на вокальному виконанні. Цей набір правил є найбільш обмежуючим і, як правило, призводить до більш тривалого часу компіляції.

Крім цього, як уже згадувалося, бібліотека Mezzo дозволяє визначати власні користувацькі набори правил, хоча додавання складних правил вимагає глибшого розуміння Mezzo зсередини та обчислень на рівні типів. Набори правил – це колекція обмежень, одна для усіх способів створення музичного значення: гармонічної композиції, мелодійної композиції, гомофонної композиції, пауз, нот та акордів. Будь-яке з цих значень може бути використане як набір правил, адже його тип – це сутність типу класу RuleSet. Як приклад:

```
class RuleSet t where
    type HarmConstraints  t (m1 :: Partiture n1 l) (m2 :: Partiture n2 l) :: Constraint
    type NoteConstraints  t (r :: RootType)       (d :: Duration)       :: Constraint
```

Кожен із обмежень приймає на вхід аргументи відповідного музичного (Music) конструктора. Наприклад, HarmConstraints приймає два скомпонованих музичних фрагменти (типу Partiture) як аргументи і повертає вираз Constraint, який визначає, чи можуть ці фрагменти бути гармонічно скомпонованими.

3.7.Приклад

Як приклад, наведемо фрагмент коду, який дозволяє зіграти знамениту мелодію Jingle Bells засобами бібліотеки Mezzo.

```
p1 = start $ melody :< e :| e :^ e :< e :| e :^ e :< e :| g :<. c :<< d :>> e
p2 = start $ melody :< f :| f :<. f :<< f :< f :| e :<. e :<< e :< e :| d :| d :| e :^ d :| g
jb = defScore $ p1 :|: p2
```

За допомогою мелодій (melody) та її конструкторів ми записуємо два фрагменти музики у функцію p1 та p2 відповідно. Тоді ми просто поєднуємо ці два фрагменти у повну мелодію за допомогою конструктора :|:, для їх послідовного відтворення. Як ми вже знаємо, за допомогою функції defScore ми просто відтворюємо нашу мелодію. Таким чином, запустивши у терміналі функцію jb ми почуємо нашу мелодію.

3.8.Висновок

Бібліотека Mezzo є досить зручним і ефективним інструментом для створення музичних композиції та їх експорту в відповідні Midi файли. Її синтаксис та можливості досить схожі з бібліотекою Euterpea. Проте, на відміну від Euterpea, бібліотека Mezzo також пропонує досить зручний спосіб для запису довгих мелодій (за допомогою melody та її конструкторів), а також надає в користування наперед визначені набори правил, з можливістю створи користувацький набір, які дозволяють вказувати певні обмеження для створюваних композиції. Таким чином, бібліотеку Mezzo можна вважати покращеним варіантом бібліотеки Euterpea для більш професійного створення музики та композицій.

ВИСНОВКИ

В ході курсової роботи було проведено дослідження та вивчення способів та бібліотек для створення музичних композиції за допомогою мови програмування Haskell. Серед них ми описали реалізацію досить простого музичного секвенсера за прикладом із підручника «Підручник з Haskell»[3] за допомогою стандартної бібліотеки Haskell HCodecs. Ця бібліотека надає основні функції для роботи з музикою та форматом Midi. Використовуючи її можливості, створено власні надбудови для ефективнішого представлення та зручнішого запису музики і її конвертації у музичний файл.

Крім того, ми розглянули популярну бібліотеку Euterpea, використовуючи навчальні матеріали з підручника «Haskell School of Music» та відкриту документацію самої Euterpea. Ми так само розібрали основні типи даних та функції, що дозволяють представляти музику у Haskell, і написали власну музичну композицію, використовуючи доступні методи.

Нарешті, ми також розглянули ще одну бібліотеку для створення музики – Mezzo. Під час вивчення її основних можливостей та доступного функціоналу, ми провели її порівняння з бібліотекою Euterpea: визначили схожі синтаксичні конструкції, особливості представлення нот, пауз, акордів, мелодій, розглянули додаткові можливості, недоступні в бібліотеці Euterpea, такі як набори правил або лаконічний спосіб представлення мелодій.

Крім вищерозглянутих бібліотек та способів, існує багато інших бібліотек для мови Haskell, кожна з яких пропонує як і стандартний набір операцій для створення музичних композицій, так і свої власні надбудови та доповнення для більш ефективного контролю та представлення мелодій засобами Haskell та різних музичних форматів.

Таким чином, ми переконалися, що сама мова Haskell є хорошим інструментарієм для тих, хто хоче спробувати попрацювати над музикою в рамках комп’ютерної програми і перевірити свої вміння та знання в музичній галузі. Різноманіття бібліотек дозволяє знайти зручний і зрозумілий набір функцій для композиторів та музикантів з різним рівнем знань, а також і для пересічних користувачів, які просто бажають «погратися» з музичними композиціями і спробувати себе у ролі композитора.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. HCodecs: A library to read, write and manipulate MIDI, WAVE, and SoundFont2 files. [Електронний ресурс] https://hackage.haskell.org/package/HCodecs
2. Холом’єв А. Підручник з Haskell.
3. Haskell School of Music. [Електронний ресурс] http://euterpea.com/haskell-school-of-music/
4. Euterpea. [Електронний ресурс] http://euterpea.com/euterpea/
5. Euterpea: Library for computer music research and education. [Електронний ресурс] https://hackage.haskell.org/package/Euterpea
6. Mezzo. [Електронний ресурс] https://github.com/DimaSamoz/mezzo#composing-in-mezzo