TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Margus Pollmann

Haskelli teegi Euterpea uurimine

Bakalaureusetöö (9 EAP)

Juhendaja(d): Kalmer Apinis

Tartu 2017

Haskelli teegi Euterpea uurimine

Lühikokkuvõte:

Käesoleva töö eesmärgiks oli uurida ja anda ülevaade Haskelli teegist Euterpea. Töös sisalduvat materjali saab kasutada lisaõppematerjalina Haskelli konstruktorite ja tüüpide õpetamisel. Euterpea on Haskelli teek, mis võimaldab muusikateoste koostamist, helisignaalide loomist ja nende töötlemist. Töös on välja toodud, kuidas luua heliteoseid ja neid esitada, luua helisignaale ja neid töödelda ning kuidas tekitada uusi instrumendi helisid. Lisaks on näiteprogrammide abil lahti seletatud teegi funktsioonid. Töö lõpuosas on ülevaade Euterpea kohta valmivast õpikust „Haskell School of Music".

Võtmesõnad:

Haskell, Euterpea, heliteoste loomine, helisignaali töötlus

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

An Investigation of Haskell library Euterpea

Abstract:

The purpose of this Bachelor thesis is to give an overview of Haskell library Euterpea. It can be used as an additional teaching material for constructors and types in Haskell. Euterpea is a Haskell library that can be used to make musical compositions or create and process audio signals. This paper describes Euterepa library's possibilities how to make musical compositions, process signals and create them and how to generate new sounds for instruments. Also its functions are described through example programs. Finally, this paper gives a small overview of the textbook about Euterpea.

Keywords:

Haskell, Euterpea, creating compositions, audio signal processing

CERCS: P170 Computer science, numerical analysis, systems, control

Sisukord

Sissejuhatus

Käesolev bakalaureusetöö eesmärgiks on uurida ja anda ülevaade Haskelli teegist Euterpea. Valminud lõputööd on võimalik kasutada lisamaterjalina Haskelli õpetamisel. Sarnaselt kilpkonnagraafikale [11], mille abil õpetati algoritmi mõisteid, saab Euterpea teegi abil luua näidiseid, mis aitavad õpetada konstruktoreid ja teisendusi. Euterpea'ga saab luua ka ülesandeid, mis on loovama sisuga ning muudaks funktsionaalses programmeerimiskeeles loodavaid programme huvitavamaks.

Euterpea on juba kasutuses Yale'i Ülikooli Arvutiteaduste ja kunsti eriala muusika suunal (*Computing and the Arts major - Music Track*). Muusika suunal õpetatakse Euterpea ja Haskelli programmeerimiskeele abil arvutimuusika (*computer music*) teemasid [13]. Yale'i Ülikoolis on loodud ka Euterpea stuudio, kus üliõpilased kui ka ülikooli lõpetanud saavad teha eksperimentaalseid arvutimuusika uurimustöid [13].

Töö esimene peatükk annab lühiülevaate Euterpea teegist. Teises peatükis kirjeldatakse kuidas luua muusikateoseid ja neid esitada. Peatükis on näidisprogrammide abil lahti seletatud Euterpea funktsioonid. Kolmandas peatükis käsitletakse signaalide töötlemist ja nende loomist. Näidisprogrammide abil on välja toodud instrumendi loomine ja signaali töötlemine. Viimases peatükis antakse õpiku „Haskell School of Music" ülevaade, mis õpetab Euterpea teeki ja muusika loomist Haskellis.

1. Euterpea

Euterpea on Yale'i Haskell Groupi poolt loodud erialaspetsiifiline keel muusika loomiseks ja arendamiseks, mis on integreeritud Haskelli programmeerimiskeelega [4]. Euterpea nimi on tulnud Euterpe nimest, kes oli Kreeka mütoloogias lüürika muusa [15].

Euterpea on Haskelli teekide Haskore'i ja HasSound'i järglane. Haskore on Haskelli moodul muusika loomiseks, analüüsimiseks ja töötlemiseks. Haskore'is pole aga võimalust muuta muusikat kuuldavaks ning selleks on vaja veel eraldi seisvat moodulit, näiteks HasSound'i. HasSound on Haskellis loodud moodul, mis muudab loodud koodi heli kuuldavaks [4].

Euterpea teegist on väljas versioon 2. Arenduses on Euterpea jõudnud stabiilsesse faasi ehk enam suuri muudatusi koodis ei tehta. Teeki hooldatakse ja arendatakse edasi, et see ühilduks ka kõige uuemate GHC versioonidega. Teegi põhifunktsioonid ja -omadused jäävad samaks ega muudeta [4].

Euterpea on mõeldud nii hariduslikel eesmärkidel kui ka keerulisemate muusika rakenduste loomiseks. Teek on sobilik kõrgetasemelise muusika kompositsioonide loomiseks, esitamiseks ja analüüsiks, kesktaseme kontseptsioonideks nagu MIDI (*Musical Instrument Digital Interface*) ning madal-taseme heli töötluseks, helisünteesiks ja muusikainstrumentide loomiseks.

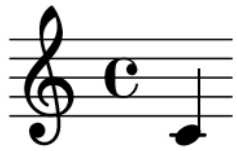
Teek on jaotatud kaheks suuremaks osaks: noodi tase ja signaali tase. Noodi tasemel on võimalik luua muusikateoseid. Loodavad teosed saavad olla alates lihtsamatest klaveripaladest kuni keerulisemate sümfooniateni. Signaali tasemel saab luua uusi instrumente ja töödelda olemasolevat heli.

2. Noodid

Noodi tasemel võimaldab Euterpea luua heliteoseid.

2.1 Nootide loomine

Muusikateoorias kirjutatakse noote noodijoonestikus, kus igal märgil on kindel tähendus.



Joonis 1. Noot noodijoonestikus

Joonisel 1 on toodud näide noodi kirjutamisest noodijoonestikus. Selle alguses on viiulivõti, mis määrab kindlaks esimese oktavi G noodi, sellele järgneb takt ning siis noot. Joonisel 1 olev noot tähistab muusikateoorias C nooti esimeses oktavis ning selle vältus ehk kestus on veerandnooti.

Euterpea's on võimalik luua noote andmetüübiga *Primitive*. *Primitive* saab olla noot (`Note :: Dur -> a -> Primitive a`), millele on juurde antud kestus *Dur* ja tüübimuutuja (*type variable*), näiteks *Pitch*, mis näitab tähtnime ja tema oktavi.

Alljärgnev materjal on refereeritud H. Nestra raamatust [9]. Haskellis on kasutusel tüübisignatuur kujul:

$x_1, ..., x_n$ `::` t,

kus $x_1 ... x_n$ on muutujad ja t on tüübiavaldis. Tüübisignatuur näitab defineeritavate muutujate tüüpi ehk muutuja x_1 kuni x_n tüübiks on t. Muutujate nimed kirjutatakse väikese algustähega.

Noodi loomiseks tuleb kõigepealt luua tüübisignatuur, mille tüübiks on *Primitive Pitch*. Joonisel 1 oleva noodi kirjutamine Euterpea's näeb välja järgmiselt:

```
naide1 :: Primitive Pitch
naide1 = Note (1/4) (C, 4)
```

Tüübikonstruktsioone, mis võtavad argumendiks tüüpe, nagu näiteks *Primitive*, kutsutakse tüübiperedeks [9].

Funktsioon võtab argumente ühest kindlast tüübist ja saadav tulemus on ka kindlat tüüpi [9]. Kui *a* ja *b* on suvalised tüübid, siis `a -> b` on funktsioonitüüp, kui ta koosneb funktsioonidest. Funktsioonitüübi argumenditüübiks on *a* ja väärtusetüübiks on *b*. Näiteks noodi loomise funktsioon `Note :: Dur -> a -> Primitive a` võtab *Dur* tüüpi argumendi ja *a* tüüpi argumendi ning tagastab väärtuse, mille tüübiks on *Primitive a*. Funktsionaalses programmeerimiskeeles võivad argumendi ja väärtusetüübid olla omakorda funktsioonitüübid. Funktsioonitüüp on parem-assotsiatiivne ehk paremalt võib sulud ära jätta:

`a -> (b -> c) == a -> b -> c.`

Pitch on Euterpea's helikõrguse jaoks loodud tüüp (`type Pitch = (PitchClass, Octave)`), kus esimeseks paari muutujaks on helikõrguse tähtnimi ja teiseks tema oktav. Loodud tüüp vastab klaverinootidele, kus (A, 0) on kõige madalam ning (C, 8) on kõige kõrgem noot. Eelpool toodud näide vastab klaveril keskmisele C noodile ning Euterpea's tähistatakse seda (C, 4).

Euterpea's on muusika mõistete jaoks loodud vastavad tüübisünonüümid, et muuta arusaadavamaks funktsioonidele antavad väärtused. Seda võimaldab Haskelli tüübisünonüümide deklaratsioon, mis annab tüübiavaldisele nime, mis on algse tüübiavaldisega samaväärne. Tüübisünonüümi defineerimiseks tuleb deklaratsiooni ette lisada võtmesõna *type* [9]. Lisaks helikõrgusele *Pitch* ja kestusele *Dur* (`type Dur = Rational`) on oktavi jaoks loodud tüüp *Octave* (`type Octave = Int`) ning absoluutse helikõrguse jaoks *AbsPitch* (`type AbsPitch = Int`).

Algebraliste andmetüüpide loomiseks on olemas võtmesõna *data*. See sobib uute tüüpide ja nendesse kuuluvate andmete sissetoomiseks. Tüübi deklaratsioon näeb välja järgmiselt:

`data t =` a_1 `| ... |` a_n,

kus *t* on uus tüübiavaldis ja a_1, …, a_n on konstruktorid, mis eraldatud üksteisest püstkriipsudega [9].

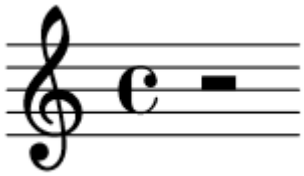
Helikõrguste jaoks on loodud vastav loenditüüp *PitchClass* (`data PitchClass = Cff | Cf | … | Bs | Bss`), kus on kirjas tähtnimed alates C-st kuni B-ni (kasutusel on inglisekeelne tähtnimetus). Lisaks tavalistele tähtnimetustele on olemas ka alusheli teisendused ehk alteratsioonid [3], vastavalt kas heli on kõrgendatud (*s – sharp* ehk diees, *ss* - duubeldiees) või madaldatud (*f – flat* ehk bemoll, *ff* - duubelbemoll).

Muusikas väljendatakse mitmeid väärtusi ratsionaalarvudena, näiteks takte ja helivältust. Heli kestuse *Dur* jaoks kasutatakse tüüpi *Rational*, sest need numbrid on täpsemad võrreldes ujukomaarvudega. Lihtsustamaks ratsionaalarvude kasutamist on Euterpea's loodud heli kestuse sünonüümid, et murdude asemel kirjutada heli kestus lühenditega: *wn – whole note* – tervenoot = 1, *hn – half note* – poolnoot = ½, qn – *quarter note* – veerandnoot = 1/4, jne.

Seega on võimalik kirjutada eeltoodud näide järgmiselt:

```
naide1 :: Primitive Pitch
naide1 = Note qn (C, 4)
```

Vaikus muusikateoses on väga kõnekas [11]. Et seda oleks võimalik esitada on muusikas olemas pausid. Euterpea's on pause võimalik luua samas andmetüübis *Primitive*, millega loodi ka noote. Pausi loomiseks tuleb kirjutada *Rest* ning anda sellele juurde kestus (`Rest :: Dur -> Primitive a`). Näiteks poolpausi kirjutamine:



Joonis 2. Paus noodijoonestikus

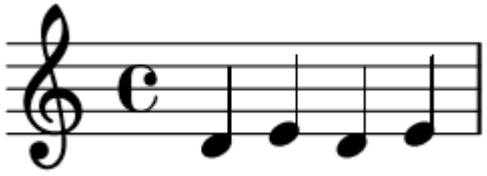
```
naide2 :: Primitive Pitch
naide2 = Rest (1/2) või naide2 = Rest hn
```

2.2 Heliteose loomine

Andmetüüp *Primitive* võimaldab luua noote ja pause, aga see ei võimalda neid esitada. Teose esitamine ja muutmine helifailiks on kirjeldatud peatükis 2.5. Nootide ja pauside esitamiseks ning kompositsioonide ehk heliteoste loomiseks on Euterpea's rekursiivne andmetüüp *Music*, mis defineerib noodi tasemel muusika struktuuri (`Prim :: Primitive a -> Music a`). Näiteks ühe noodiga kompositsioon:

```
naide3 :: Music Pitch
naide3 = Prim (Note qn (C, 4))
```

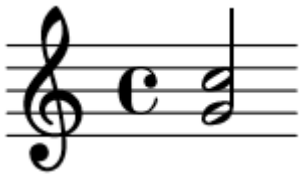
Teineteisele järgnevad helid on noodijoonestikus kirjutatud üksteise järele. Samal põhimõttel on ka Euterpea's võimalik kirjutada järjest esitatavaid helisid. Järjest *Music* väärtuste, näiteks nootide, esitamiseks on loodud andmetüübis *Music* vastav konstruktor (`:+:`)(`(:+:) :: Music a -> Music a -> Music a`). Konstruktor lisatakse üksteise järele kirjutatud *Music* väärtuste vahele.



Joonis 3. Teineteisele järgnevad helid

```
naide4 :: Music Pitch
naide4 = Prim (Note (1/4) (C, 4)) :+:
     Prim (Note (1/4) (D, 4)) :+:
     Prim (Note qn (C, 4)) :+:
     Prim (Note qn (D, 4))
```

Samaaegselt kõlavad helid on noodistikus kirjutatud üksteise alla. Euterpea teegis lisatakse *Music* väärtuste vahele konstruktor (`:=:`)(`(:=:) :: Music a -> Music a -> Music a`).



Joonis 4. Samaaegselt kõlavad helid

```
naide5 :: Music Pitch
naide5 = Prim (Note (1/2) (G, 4)) :=: Prim (Note hn (C, 5))
```

Kui nootide kestus ei ole sama, siis mängitakse noodid nii, et lühem ja pikem noot hakkavad kõlama samaaegselt. Peale lühema noodi kõlamist jääb kõlama ainult pikem noot. Funktsiooni *offset* (`offset :: Dur -> Music a -> Music a`) abil on võimalik alustada pikema noodiga ja siis teatud hetkel sisse tuua lühem noot. Funktsioon *offset* lisab antud noodile ette kasutaja poolt määratud pikkusega pausi.

```
naide6 :: Music Pitch

naide6 = Prim (Note hn (E, 3)) :=:

     offset qn (Prim (Note qn (F, 3)))
```

Peale nootide on vaja kompositsiooni lisada ka lisainformatsiooni, mis näitaks näiteks tempot või määraks teose helistiku. Selleks on võimalik andmetüübist *Music* valida konstruktor *Modify* (`Modify :: Control -> Music a -> Music a`).

Modify'le antav argument *Control* on andmetüüp, mis võimaldab tempo muutmist (*scale the tempo),* transponeerimist ehk heliteost ühest helistikust üle kanda teise helistikku [14], muuta instrumenti, anda fraasile omadusi ja määrata helistikku. Lisaks on kasutajal võimalus luua ka uusi funktsioone, mis muudavad heliteost.

Euterpea's erineb tempo määramine muusikateooria tempo määramisest. Kui teoorias määratakse tempot metronoomi järgi, siis Euterpea's antakse tempole (`Tempo :: Rational -> Control`) juurde vastav kordaja mis kiirusega seda esitada. Tempo kordaja „1" tähistab algset kiirust. Ühest suurem arv on kiirem ja ühest väiksem arv on aeglasem tempo. Tempo kordaja ei tohi olla null.

```
naide7 :: Music Pitch

naide7 = Modify (Tempo 2) (Prim (Note 1 (D, 5)))
```

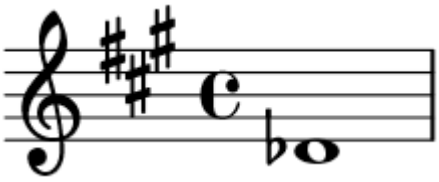
Heliteose esitamisel erinevatel instrumentidel võib selguda, et helistik ei ole mängimiseks sobiv. Sellisel juhul tuleb heliteos üle viia sobivasse helistikku ehk tuleb heliteost transponeerida [14]. Transponeerimisel antakse konstruktorile *Transpose* (`Transpose :: AbsPitch -> Control`) väärtus, mis vastab sellele mitme oktavi võrra muudetakse helikõrgust. Nullist suurem arv tõstab kõrgemaks, nullist madalam vähendab helikõrgust.

```
naide8 :: Music Pitch

naide8 = Modify (Transpose 3) (Prim (Note (1/4) (C, 4)))
```

Euterpea's on vaikimisi valitud instrumendiks klaver (*AcousticGrandPiano*). Et oleks võimalik muuta instrumenti või luua teoseid mitmele instrumendile on teegis eeldefineeritud üle 100 erineva instrumendi heli. Olemasolevate helide nimekirja on võimalik näha lähtekoodis [8] või konsoolis kirjutades `:i InstrumentName`. Lisaks on võimalik luua ka ise uue instrumendi heli. Uue instrumendi heli loomine on kirjeldatud peatükis 3.2.

```
naide9 :: Music Pitch

nadie9 = Modify Violin (Prim (Note hn (E,5)))
```

Helistik on kindlaks määratud algusnoodiga helilaad [14]. Heliteose helistikku näitab võtmemärkide arv, mis asuvad noodirea alguses noodivõtme järel. Euterpea's on võimalik helistik paika määrata konstruktoriga *KeySig* (`KeySig :: PitchClass -> Mode -> Control`). Esimesene argument *PitchClass* on noodi tähtnimetus ja teine *Mode* on vastavalt helilaadile kas duur (*Major*) või moll (*Minor*).

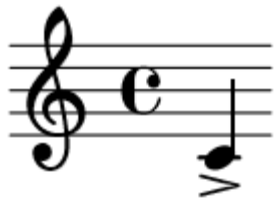


Joonis 5. Helistiku määramine

```
naide10 :: Music Pitch

naide10 = Modify (KeySig A Major) (Prim (Note 1 (Df, 4)))
```

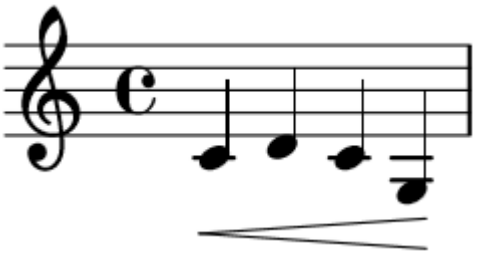
Muusikateoorias [14] nimetatakse helitöö ülesehitust muusikateose vormiks. Vorm koosneb omakorda väiksematest osadest. Vormi kõige väiksem muusikaline mõte on motiiv [1]. Kaks motiivi moodustavad omakorda fraasi. Fraas on noodijoonestikus välja toodud tavaliselt fraasikaare või märgiga väike *v*. Euterpea's loodud konstruktor *Phrase* (`Phrase :: [PhraseAttribute] -> Control`) sarnaneb muusikateoorias olevale fraasile, kus üht nootide hulka vaadeldakse koos. Konstruktori abil on võimalik tervele teosele või selle teatud osale anda erinevaid omadusi (*PhraseAttribute*). Euterpea teegis on fraasile võimalik lisada dünaamikat, tempot, artikulatsiooni või ornamentikat.

Dünaamika tähendab heliteose esitamise tugevust ja selle muutumist [12]. Euterpea's on dünaamikas võimalik lisada aktsent (*accent*) ehk heli rõhutamine [3], muuta heliteose esitamise tugevust ja selle muutumist ning järk-järgulist kõlatugevuse muutumist. Dünaamika lisamiseks on *Phrase*'le vaja anda *PhraseAttribute*'ks *Dyn* (`Dyn :: Dynamic -> PhraseAttribute`).



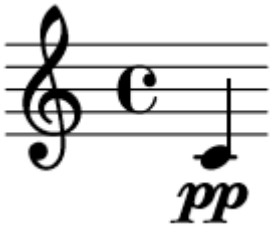
Joonis 6. Aktsendi märk

```
naide11 :: Music Pitch

naide11 = Modify (Phrase [Dyn (Accent (1/4))])

     (Prim (Note qn (C, 4)))
```



Joonis 7. Järk-järguline kõlatugevuse suurendamine (*crescendo*)

```
naide12 :: Music Pitch

naide12 = Modify (Phrase [Dyn (Crescendo 1.5)])

     (Prim (Note qn (C, 4))) :+:

     Prim (Note qn (D, 4))) :+:

     Prim (Note qn (C, 4))) :+:

     Prim (Note qn (G, 3))))
```



Joonis 8. Dünaamika noodijoonestikus (*pianissimo*)

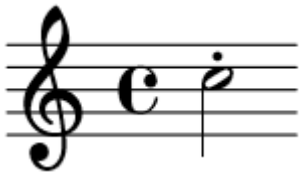
```
naide13 :: Music Pitch
```

```
naide13 = Modify (Phrase [Dyn (StdLoudness PP)])
     (Prim (Note qn (C, 4)))
```

Tempo ei pruugi muusikapala vältel olla alati sama, vaid võib muutuda aeglasemaks või kiiremaks. Tempo muutumiseks on *PhraseAttribute*'is konstruktor *Tmp* (`Tmp :: Tempo -> PhraseAttribute`). Konstruktorile antakse juurde tempomärge ja kordaja. Tempomärgiks võib olla kas *Accelerando* (kiirenev tempo) või *Ritardando* (aeglustuv tempo).

```
naide14 :: Music Pitch
naide14 = Modify (Phrase [Tmp (Accelerando 1.5)])
     (Prim (Note (1/8) (A, 4)))
```

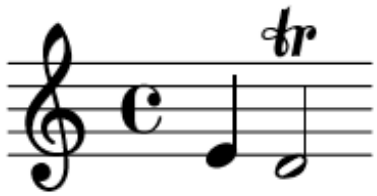
Phrase konstruktoris on võimalik ka artikulatsioon ehk üksikute helide ühendamine või esiletoomine [3]. Artikulatsiooni lisamiseks tuleb *Phrase*'le anda konstruktor *Art* (`Art :: Articulation -> PhraseAttribute`). Artikulatsioonivõtted, näiteks *Staccato,* on Euterpea's juba eeldefineeritud.



Joonis 9. *Staccato* noodijoonestikus

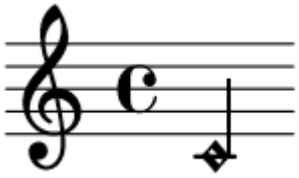
```
naide15 :: Music Pitch
naide15 = Modify (Phrase [Art (Staccato (1/2))])
     (Prim (Note (1/2) (C, 5)))
```

Viimaseks fraasi omaduseks on võimalik lisada eellööke ehk ornamentikat (*ornament*). Kasutada saab juba eeldefineeritud lööke, näiteks *trill*, luua ise uus löök (`Instruction String`), lisada noodi pikkus muutes noodipead (`Head NoteHead`) või transponeerida heli diatoonilisel skaalal (`DiatonicTras Int`). Ornamentika lisamiseks on konstruktor *Orn* (`Orn :: Ornament -> PhraseAttribute`).



Joonis 10. Ornamentika noodijoonestikus (*trill*)

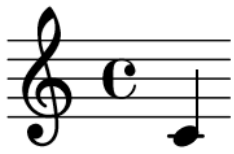
```
naide16 :: Music Pitch
naide16 = Modify (Phrase [Orn Trill])
     (Prim (Note hn (D, 4)))
```



Joonis 11. Noodipea

```
naide17 :: Music Pitch
naide17 = Modify (Phrase [Orn (Head DiamondHead)])
     (Prim (Note hn (C, 4))
```

Selline pikalt nootide ja konstruktorite kirjutamine võib muuta koodi pikaks ja raskesti loetavaks. Et seda ei juhtuks on teegis loodud funktsioonid, mis aitavad seda lihtsustada. Nootide kirjutamiseks on olemas kaks funktsiooni. Esimese funktsiooni nimeks on *note*, mille argumentideks on kõigepealt kestvus ja siis tema helikõrgus. Teine funktsioon vastab noodi tähtnimele ning saab argumentiks kaasa tema oktavi ja kestvuse. Näiteks `Prim (Note qn (C,4))` on võimalik kirjutada järgmiselt:



Joonis 12. Esimese oktavi C noot noodijoonestikus

```
Prim (Note qn (C, 4)) == note qn (C, 4) == c 4 qn == c 4 (1/4)
```

Pauside jaoks on loodud eraldi funktsioonid. Funktsioonile *rest* antakse ette selle kestus, kas siis murdarvuna või lühendiga. Teine võimalus pausi lisamiseks on teegis loodud funktsioonid, mis vastavad kestuse lühendile ja mille lõppu on lisatud *r*.

```
Prim (Rest (1/2)) == rest (1/2) == rest hn == hnr
```

2.3 Näide - Euterpea funktsioonid

Teegi kasutamiseks tuleb esmalt importida Euterpea, kasutades võtmesõna *import*.

```
import Euterpea
```

Kõigepealt tuleb deklareerida loodav teos. Funktsioonide näitamiseks on loodud read, mis on omakorda lahutatud väiksemateks osadeks, et lihtsustada koodi lugemist.

```
rida1, rida1_1, rida1_2, rida1_3, rida2, rida2_1, rida2_2,
rida2_3, rida3, rida3_1, rida3_2, rida4, rida5, teos,

teosHelistik :: Music Pitch
```

Samaaegselt mängitavaid noote on võimalik kirjutada mitmel erineval viisil, näiteks rida1_1 ja rida1_2 puhul on kasutusel konstruktor (`:=:`). Sama teeb ka funktsioon *chord* (`chord :: [Music a] -> Music a`), mis võtab listi nootidest ja muudab nad samaaegselt esitatavaks.

```
rida1_1 = note qn (C, 4) :=: note qn (E, 4) :=:
     note qn (A, 4)
rida1_2 = c 4 qn :=: f 4 qn :=: a 4 qn
rida1_3 = chord [c 4 qn, f 4 qn, g 4 qn]
```

Nootide järjest esitamiseks on funktsioon *line* (`line :: [Music a] -> Music a`), mis võtab listi nootidest ja lisab nende vahele konstruktori (`:+:`), mis mängib noote järjest.

```
rida2_1 = line [d 2 1, e 2 qn, g 2 qn]
```

Funktsioon *invert* (`invert :: Music a -> Music a`) on inversioon ehk peegel, mille puhul toimub helikõrguse peegeldus horisontaaltelje suhtes [3]. Funktsioon peegel-

dab esimese noodi ümber järgmisi noote. Teatud helikõrguse ümber inversiooni tegemiseks on funktsioon *invertAt* (`invertAt :: Pitch -> Music Pitch -> Music Pitch`).

```
rida2_2 = rida2_1 :+: rest hn :+: invert rida2_1
```

Erinevate omaduste lisamiseks on funktsioon *pharse* (`phrase :: [PhraseAttribute] -> Music a -> Music a`). Näites muudetakse rida2_2 heli tugevust astmele *ff – fortissimo*.

```
rida2_3 = phrase [Dyn (StdLoudness FF)] rida2_2
```

Funktsiooni *shiftPitches* (`shiftPitches :: AbsPitch -> Music Pitch -> Music Pitch`) abil on võimalik loodud teose helikõrgust tõsta või madaldada etteantud väärtuse võrra.

```
rida3_1 = shiftPitches 4 rida1
```

Tempo muutmiseks on funktsioon *tempo* (`tempo :: Dur -> Music a -> Music a`).

```
rida3_3 = tempo 0.8 rida3_1
```

Sama noodi või teatud teose osa kordamiseks on funktsioon *times* (`times :: Int -> Music a -> Music a`). Lõpmata arv kordi kordamiseks on funktsioon *forever* (`forever :: Music a -> Music a`).

```
rida3_2 = line [times 3 (d 3 en), times 2 (c 3 en), g 4 1,
     a 4 1]
rida4 = times 3 (chord [rida1, rida2])
```

Instrumendi lisamiseks *instrument* (`instrument :: InstrumentName -> Music a -> Music a`).

```
rida1 = instrument Pad1NewAge (line [times 8 rida1_1,
     times 3 rida1_2, times 5 rida1_3])
rida2 = instrument SynthBass1 rida2_3
rida3 = instrument Pad2Warm chord ([times 2 rida3_1, rida3_3])
```

Juba loodud teose osa tagurpidi mängimiseks ei tule seda uuesti kirjutada. Selle jaoks on olemas funktsioon *retro* (`retro :: Music a -> Music a`). Rida2 esitatakse tagurpidi ja selle kõlatugevus kahaneb järk-järgult.

```
rida5 = phrase [Dyn (Diminuendo 1.3)] (retro rida2)
```

Lõpliku teose saab kokku panna kasutades funktsiooni *line* või konstruktorit (`:+:`).

```
teos = rida1 :+: rida4 :+: (rida3 :=: rida2) :+: rida5
```

Helistikku on võimalik määrata funktsiooni *keysig* (`keysig :: PitchClass -> Mode -> Music a -> Music a`) abil. Funktsioonile antakse ette helikõrguse tähtnimetus, vastavalt kas duur (*Major*) või moll (*Minor*).

```
teosHelistik = keysig C Minor teos
```

2.4 Näide - Löökpillid ja funktsioonide jätk

Muusikainstrumentide valikus on ka löökpillid (*Percussion*), näiteks suur trumm (*bass drum*). Nende lisamine konstruktori *Instrument* abil võib osutada keeruliseks, sest siis on vaja teada, milline löökpilli heli vastab kindlale noodile. Löökpilli heli lihtsamaks lisamiseks on funktsioon *perc* (`perc :: PercussionSound -> Dur -> Music Pitch`). Funktsioon võtab löökpilli nime ja selle kestuse ning loob neist *Music* väärtuse. Löökpillide nimed on võimalik näha konsoolis kirjutades `:i PercussionSound` või lähtekoodist [5].

```
lookpill1, lookpill2, lookpill3 :: Music Pitch

lookpill1 = instrument Percussion (c 2 1)

lookpill2 = perc LowTom wn

lookpill3 = perc ElectricSnare sn
```

Mitme funktsiooni korraga kasutamine on lihtsam, aga need lisavad *Music* väärtuse lõppu null kestusega pause. Need null väärtusega pausid ei mõjuta heli esitlust, aga muudavad koodi pikemaks ja keerulisemaks. Loodud löökpillidest loome lühikese teose, kasutades funktsiooni *line1* (`line1 :: [Music a] -> Music a`). *Line1* sarnaneb funktsiooniga *line*, mis võtab *Music* väärtustest koosneva listi ja loob neist järjest esitatava kompositsiooni. *Line1* aga ei lisa lõppu pausi kestusega null ega tekita see tõttu juurde lisa koodi, mida vaja ei ole.

```
lpTeos :: Music Pitch
lpTeos = tempo 4 (times 3
     (line1 [lookpill1, lookpill2, lookpill3]))
```

Juba loodud heliteose kestust on võimalik lühendada või pikendada funktsiooniga *scaleDurations* (`scaleDurations :: Rational -> Music a -> Music a`). Funktsioon jagab läbi kõik teose nootide vältused etteantud arvuga. Näiteks teose vähendamiseks poole võrra tuleb anda funktsioonile väärtuseks kaks.

```
lpTeosLyhendatud :: Music Pitch
lpTeosLyhendatud = scaleDurations 2 lpTeos
```

Olemasoleva *Music* väärtuse instrumendi vahetamiseks on funktsioon *changeInstrument* (`changeInstrument :: InstrumentName -> Music a -> Music a`). Kõigi instrumentide eemaldamiseks teoselt on *removeInstruments* (`removeInstruments :: Music a -> Music a`).

```
uusInstrument :: Music Pitch
uusInstrument = changeInstrument Bagpipe lookpill1
```

Funktsioon *offset* (`offset :: Dur -> Music a -> Music a`) lisab etteantud väärtuse ette pausi.

```
lpTeosValmis :: Music Pitch
lpTeosValmis = offset 1 lpTeos :=: (times 3 uusInstrument)
```

Euterpea's on loodud ka funktsioon *remove* (`remove :: Dur -> Music a -> Music a`), mis eemaldab antud kestuse ulatuses teose algust.

```
eemaldamine :: Music Pitch
eemaldamine = remove 2 teos
```

2.5 Teoste esitamine ja salvestamine

Loodud heliteoste esitamiseks peavad need olema deklareeritud *Music* väärtustena.

```
teos :: Music Pitch
teos = c 4 1 :+: d 4 1 :+: e 4 1 :+: f 4 1 :+:
```

```
    g 4 1 :+: a 4 1:+: b 4 1
```

Funktsioon *play* (`play :: (ToMusic1 a, NFData a) => Music a -> IO ()`) esitab loodud *Music* väärtuse vaikimisi valitud MIDI väljundseadmes (kõlaris).

```
play teos
```

Funktsiooni *playDev* abil on ise võimalik valida milline MIDI seade esitab heli. Talle antakse ette numbri kujul MIDI seade ning *Music* väärtus. Arvutis olemasolevaid MIDI seadmeid saab näha funktsiooniga *devices*.

```
playDev 1 teos
```

Funktsioon *play* toetab lõpmata pikkusega väärtusi. Kui *Music* väärtusel on kindel kestus, siis saab esitada teost funktsiooni *playS* või *playDevS.*

```
playS teos
```

```
playDevS 1 teos
```

Teose salvestamiseks on funktsioon *writeMidi* (`writeMidi :: ToMusic1 a => FilePath -> Music a -> IO ()`). Funktsioon loob MIDI-faili, mida on võimalik esitada igas arvutis millel on olemas helikaart. Funktsiooni argumendiks antav faili teekond (*FilePath*) peab lõppema faililaiendiga .mid.

```
writeMidi "teos.mid" teos
```

Olemasolevat MIDI-faili saab ka tagasi *Music* väärtuseks muuta. Selleks on vaja eelnevalt luua funktsioon `laeMidiFail`, mis laeks MIDI-faili õigesti sisse.

```
laeMidiFail fail = do
    sisu <- importFile fail
    case sisu of
        Left viga -> error viga
        Right midi  -> return midi
```

Saadud andmed ei ole veel *Music* väärtuse kujul. Õigele kujule saamiseks on vaja luua uus funktsioon:

```
midiMusic fail = do
  x <- laeMidiFail fail
  print (fromMidi x)
```

3. Signaalid

Euterpea teek võimaldab elektroonilisest signaalist heli luua ja seda töödelda. Saab luua uusi helisid, instrumente või järele teha olemasoleva instrumendi häält. Heli on võimalik filtreerida, kasutada ekvalaiserit või lisada efekte.

Signaalide jaoks on loodud klass *Clock*, mis on defineeritud kujul:

```
class Clock p where
     rate :: p → Double
```

Klassi *Clock* meetod *rate* määrab helisignaali sämplimissageduse (*sampling rate*). Teegis on kasutusel kaks juba olemasolevat *clock* tüüpi *AudRate* ja *CtrRate*. *AudRate'i* sagedus on defineeritud 44.1 kHz, mis on tavaline sämplimissagedus [16], ja *CtrRate* on 4.41 kHz. Kuna neid kahte tüüpi kasutatakse tihti, siis on loodud nende lihtsamaks kasutamiseks sünonüümid:

```
type AudSF a b = SigFun AudRate a b

type CtrSF a b = SigFun CtrRate a b
```

3.1 Signaali loomine

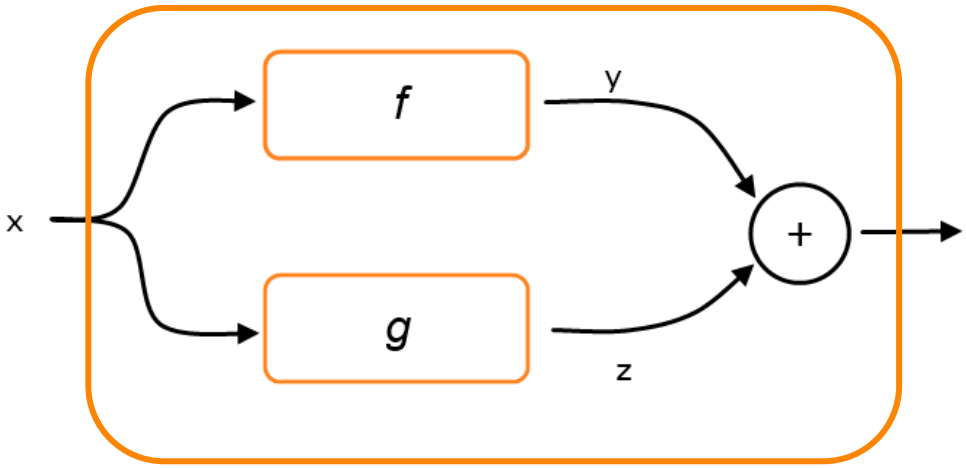
Euterpea's ei ole väärtusi, mis esitavad signaale, vaid on olemas signaali funktsioonid. Signaali funktsioonide kasutamiseks on vaja luua uus funktsioon, mis määrab sämplimissageduse ja sisendid ning väljastab uue loodud signaali heli.

Joonis 13. Digitaalne signaalitöötlus

Uut funktsiooni võib vaadelda kui signaalitöötluse diagrammi, kus jooned tähistavad signaale ja kastid tähistavad signaalide funktsioone, mis teisendavad saadud sisendit [2].

3.1.1 Noolesüntaks

Teek kasutab noolesüntaksit (ingl Arrows) [10], mis jätkab sarnast noolte ja kastide vormi.



Joonis 14. Noolesüntaks

Joonisel 14 on kujutatud noolesüntaksit, kus sisend *x* antakse arvutustele *f* ja *g*. Saadud uuetele väärtustele antakse uued nimed *y* ja *z*, mis omavahel liidetakse ja antakse väljundiks. Haskelli koodis näeks see välja järgnevalt:

```
proc x -> do
     y <- f -< x
     z <- g -< x
     returnA -< y + z
```

Süntaksis on *proc* (*arrow abstraction*) lambda avaldis, mis konstrueerib noole ja seob sisendi mustriga [10]. Sisend *x* antakse funktsioonidele kasutades noole rakendust (*arrow application*) −<. Süntaksile antakse üks sisend, mida saab süntaksi sees kasutada ja edasi anda kasutades noole rakendusi. Väljundeid on süntaksil sama moodi ainult üks.

3.1.2 Signaali funktsioon

Signaali loomise funktsiooni näidismudel on järgnev:

```
signaaliNimi :: AudSF InType OutType
signaaliNimi = proc sisend -> do
    väljund <- signaaliFunktsioon -< sisend
    returnA -< väljund
```

Esmalt deklareeritakse signaali funktsiooni tüüp. Argumentideks on sämplimissagedus, sisend, mis võib olla ka tühi, ja väljundi tüüp. Funktsiooni sees kasutatakse noolesüntaksit, mis annab edasi sisendeid ja väljundeid signaali funktsioonidelt. Signaali funktsioonid on Euterpea's juba eeldefineeritud. Võimalik on sinusoide luua, müra genereerida, signaali filtreerida ja viivitusi lisada. Igal funktsioonil on sisendid ja parameetrid erinevad [6].

Loodud signaali tuleb salvestada funktsiooni *outFile* (`outFile :: (AudioSample a, Clock c) ⇒ String → Double → SigFun c () a → IO ()`) abil. Esimene argument on WAV-faili nimi, kuhu loodud signaal salvestada. Teine argument on kestus sekundites ja viimane on signaali funktsioon. On loodud ka funktsioon *outFileNorm* (`outFileNorm :: (AudioSample a, Clock p) => String -> Double -> Euterpea.IO.Audio.IO.Signal p () a -> IO ()`), mis normaliseerib väljundi, kui signaali amplituud läheb üle ühe.

3.1.3 Näide - siinuse graafik määratud sagedusel 250 Hz

```
{-# LANGUAGE Arrows #-}
import Euterpea

naide250 :: AudSF () Double
naide250 = proc _ -> do
     valjund <- oscFixed 250 -< ()
     returnA -< valjund
heli = outFile „naide.wav" 5 naide250
```

Näide loob viie sekundilise heli sagedusel 250 Hz. Noolesüntaksi kasutamiseks on vaja lisada `{-# LANGUAGE Arrows #-}`, mis võimaldab süntaksi kasutamist. Importima peab ka Euterpea. Signaali funktsioon naide250 võtab sisendi, mis siin kohal ei ole oluline ja väljastab *Double* tüüpi väärtuse. Funktsiooni sees olev *oscFixed* loob etteantud sagedusele vastava signaali. Viie sekundiline helifail salvestatakse funktsiooni *outFile* abil.

3.2 Instrumendi loomine

Euterpea võimaldab kasutajal luua uusi instrumendi helisid. Järgnev näiteprogramm selgitab uue heli loomist Euterpea's:

Euterpea kasutab Haskelli Arrows süntaksi. Selleks peab faili alguses olema {-# LANGUAGE Arrows #-}. Vaja on importida ka Euterpea teek.

```
{-# LANGUAGE Arrows #-}
import Euterpea
```

Luua tabel, mis esitab signaali lainekuju (*wave table*). Teegis on funktsioonid, mis loovad selle tabeli andes neile vastavad argumendid (igal funktsioonil on erinevad). Antud näites luuakse sinusoid 4096 kirjega ja põhivõnkesagedus amplituudiga 0.5.

```
testTabel :: Table
testTabel = tableSinesN 4096 [0.5]
```

Instrumendi heli saab olla kas stereo- või monoheli. Monoheliga instrumendi jaoks tuleb deklareerida `Instr (Mono AudRate)` ja stereoheli jaoks `Instr (Stereo AudRate)`. Tüübile *Instr* antakse ette neli parameetrit – kestus (*dur*), helikõrgus *AbsPitch* väärtusena (*pch*), heli tugevus (*vol*) ja parameetrid listi kujul (*params*). Antud helikõrguse saab muuta sageduseks funktsiooniga *apToHz*. Kasutatakse noolesüntaksit, mis signaali funktsioonile *osc* annab ette, millisel sagedusel peab väljundsignaal olema.

Monoheli korral on väljundsignaale ainult üks.

```
instrumentHeli :: Instr (Stereo AudRate)
instrumentHeli dur pch vol params =
     let freq = apToHz pch
     in  proc _ -> do
         x <- osc testTabel 0 -< freq
         y <- osc testTabel 1 -< freq
         returnA -< (x, y)
```

Defineerida uue instrumendi nimi.

```
uusInstr :: InstrumentName
uusInstr = CustomInstrument "Uus"
```

Luua uus instrumentide *Map* (`InstrMap`). Kui heli on mono, siis *Stereo* asemel on *Mono.* Uude *Map*'i lisatakse paari kujul instrumendi nimi ja selle heli.

```
instrMap :: InstrMap (Stereo AudRate)
instrMap = [(uusInstr, instrumentHeli)]
```

Kui instrument on loodud, lisatakse see heliteosele funktsiooniga *instrument* (`instrument :: InstrumentName -> Music a -> Music a`).

```
meloodia = instrument uusInstr (c 4 1 :+: d 4 1)
```

Funktsiooniga *writeWav* (`writeWav :: String -> InstrMap -> Music a`) saab salvestada loodud teose, kus on kasutusel uus heli.

```
writeIt = writeWav "testInstr.wav" instrMap meloodia
```

4. Haskell School of Music

„Haskell School of Music“ on Paul Hudaki poolt alustatud, hiljem Euterpea hooldaja Donya Quicki poolt üle võetud, õpik Euterpea ja muusikaga töötamise kohta Haskellis. Õpik sisaldab õpetusi ja näiteid Euterpea teegi ja graafiliste vidinate (*widget*s) kohta, mille abil saab luua kasutajaliideseid. Raamatu avaldamise aeg on plaanitud 2017.aastal. Raamatu kohta on võimalik vaadata uuendusi koduleheküljelt [7].

Raamatuga käib kaasas ka oma teek, HSoM, mis laiendab juba olemasolevat Euterpea teeki ja lisab juurde *Musical User Interfaces* (MUI), mille abil saab luua kasutajaliideseid.

Kokkuvõte

Antud bakalaureusetöö eesmärgiks oli anda ülevaade Haskelli teegist Euterpea. Töös toodi välja teegi erinevad võimalused ja loodud näiteprogrammide abil on näidatud Euterpea funktsioonid.

Töös jagati Euterpea võimalused kaheks: nootide tase ehk heliteoste loomine ja signaalide tase. Heliteoste loomise all kirjeldati kuidas muusikateooria viia kokku Euterpea teegiga ja kuidas on võimalik loodud heliteoseid esitada. Signaalide tasemel kirjeldati signaalide töötlust ja nende loomist. Signaalide all on välja toodud ka Euterpea's kasutatav noolesüntaksi ja uue instrumendi heli loomine ja selle esitamine.

Tööd on võimalik kasutada lisamaterjalina programmeerimiskeele Haskelli õpetamisel. Näiteks Euterpea abil on võimalik luua ülesandeid, mida saab kasutada konstruktorite ja tüüpide õpetamisel.

Teeki saavad kasutada ka muusikud, et luua enda muusikateoseid kasutades programmeerimist. Võimalus on luua ka täiesti uusi ja unikaalseid helisid, mida on võimalik kasutada loodavates teostes.

Viidatud kirjandus

[1] Ainsalu E. Muusika põhiõpetus. Tallinn: Kirjastus „Valgus“. 1968.

[2] Digitaalne signaalitöötlus: https://et.wikipedia.org/wiki/Digitaalne_signaalit%C3%B6%C3%B6tlus (6.05.2017)

[3] Eelhein-Issakainen E. Muusikaleksikon. Tallinn: Eesti Entsüklopeediakirjastus. 1996.

[4] Euterpea kodulehekülg: http://www.euterpea.com/ (6.05.2017)

[5] Euterpea lähtekood Github'is: https://github.com/Euterpea/Euterpea2 (5.05.2017)

[6] Euterpea signaalide lühikokkuvõte: http://euterpea.com/wpcontent/uploads/2016/12/euterpea_signal_quick_reference.pdf (28.04.2017)

[7] Haskell School of Music kodulehekülg: http://www.euterpea.com/haskell-school-of-music/ (3.05.2017)

[8] Muusikainstrumentide loend lähtekoodis: https://github.com/Euterpea/Euterpea2/blob/master/Euterpea/Music.lhs#L41 (30.04.2017)

[9] Nestra H. Sissejuhatus funktsionaalsesse programmeerimisse. Tartu: Tartu Ülikooli Kirjastus. 2010.

[10] Paterson R. A New Notation for Arrows. *International Conference on Functional Programming*, 2001. http://www.staff.city.ac.uk/~ross/papers/notation.pdf (10.05.2017)

[11] Programmeerimisest maalähedaselt veebileht - Kilpkonnagraafika: https://courses.cs.ut.ee/2015/progmaa/spring/Main/Kilpkonn (1.05.2017)

[12] Sepp A., Garšnek I., Ojakäär J. Muusikaõpik IX klassile. AVITA 2000. lk. 126 – 144.

[13] The Yale Haskell Groupi Euterpea kodulehekülg: http://haskell.cs.yale.edu/euterpea/ (3.05.2017)

[14] Visnapuu M. Abimaterjal muusikateooriast. AS Võru Täht trükikoda. 2012.

[15] Õpik Haskell School of Music: http://haskell.cs.yale.edu/wp-content/uploads/2015/03/HSoM.pdf (5.05.2017)

[16] 44.1 kHz digitaalses helis: https://en.wikipedia.org/wiki/44,100_Hz (6.05.2017)

Lisad

I. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, **Margus Pollmann**,
(*autori nimi*)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose
Haskelli teegi Euterpea uurimine,
(*lõputöö pealkiri*)

mille juhendaja on **Kalmer Apinis**,
(*juhendaja nimi*)

1.1. reprodutseerimiseks säilitamise ja üldsusele kättesaadavaks tegemise eesmärgil, sealhulgas digitaalarhiivi DSpace-is lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
1.2. üldsusele kättesaadavaks tegemiseks Tartu Ülikooli veebikeskkonna kaudu, sealhulgas digitaalarhiivi DSpace´i kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tartus, **11.05.2017**