

# The Haskore Computer Music System

## An example of an Embedded Domain Specific Language in Haskell

Bogdan Dumitriu   José Pedro Magalhães

Department of Computer Science  
Utrecht University

June 13, 2005

# Outline

- 1 Introduction
  - Definition
  - Usage of Haskell features
  - Players and Performance
- 2 Input / Output
  - MIDI and CSound files
- 3 Mathematic reasoning
  - Music theory expressed in Haskell
- 4 Examples

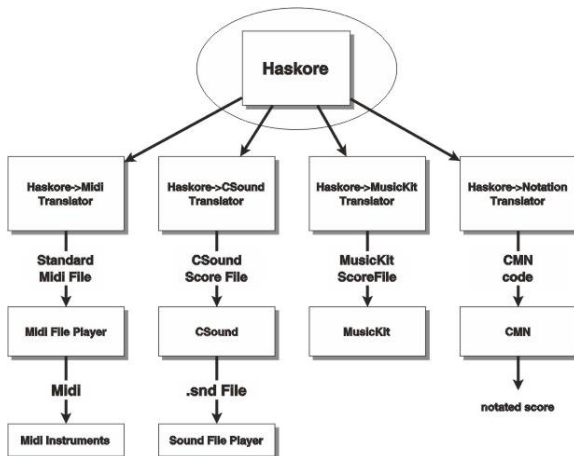
# The basics of Haskore

## What is Haskore?

- Collection of Haskell modules designed for expressing musical structures in Haskell
- Musical objects consist of:
  - Primitive notions (notes and rests)
  - Operations to transform musical objects (transposition and tempo)
  - Operations to combine musical objects (sequential/parallel compositions)

# The basics of Haskore

## Overall System Diagram



# The basics of Haskore

## Usage of Haskell features

### The Music data type:

```
data Music = Note Pitch Dur [NoteAttribute] -- a note    \ atomic
          | Rest Dur                         -- a rest    / objects
          | Music :+: Music                  -- sequential composition
          | Music :=: Music                  -- parallel composition
          | Tempo (Ratio Int) Music          -- scale the tempo
          | Trans Int Music                  -- transposition
          | Instr IName Music                -- instrument label
          | Player PName Music               -- player label
          | Phrase [PhraseAttribute] Music -- phrase attributes
          deriving (Show, Eq)
```

# The basics of Haskore

## Usage of Haskell features

### The Music data type:

```
data Music = Note Pitch Dur [NoteAttribute] -- a note    \ atomic
          | Rest Dur                        -- a rest    / objects
          | Music :+: Music                 -- sequential composition
          | Music :=: Music                 -- parallel composition
          | Tempo (Ratio Int) Music         -- scale the tempo
          | Trans Int Music                 -- transposition
          | Instr IName Music               -- instrument label
          | Player PName Music              -- player label
          | Phrase [PhraseAttribute] Music -- phrase attributes
          deriving (Show, Eq)
```

### • Higher-order functions:

```
line, chord :: [Music] -> Music
line = foldr1 (:+:)
chord = foldr1 (:=:)
```

# The basics of Haskore

## Usage of Haskell features

### The Music data type:

```
data Music = Note Pitch Dur [NoteAttribute] -- a note    \ atomic
          | Rest Dur                        -- a rest    / objects
          | Music :+: Music                 -- sequential composition
          | Music :=: Music                 -- parallel composition
          | Tempo (Ratio Int) Music        -- scale the tempo
          | Trans Int Music                 -- transposition
          | Instr IName Music               -- instrument label
          | Player PName Music              -- player label
          | Phrase [PhraseAttribute] Music -- phrase attributes
          deriving (Show, Eq)
```

- Higher-order functions:

```
line, chord :: [Music] -> Music
line = foldr1 (:+:)
chord = foldr1 (:=:)
```

- Infinite objects:

```
repeatM :: Music -> Music
repeatM m = m :+: repeatM m
```

# Players and Performance

## Players - abstraction

- `Music` abstraction independent from musical interpretation
- Haskore models real life!

```
data Player = MkPlayer { pName :: PName
                        , playNote :: NoteFun
                        , interpPhrase :: PhraseFun
                        , notatePlayer :: NotateFun
                        } deriving Show

type PMap = PName -> Player
type NoteFun = Context -> Pitch -> Dur -> [NoteAttribute] -> Performance
type PhraseFun =
    PMap -> Context -> [PhraseAttribute] -> Music -> (Performance, DurT)
type NotateFun = () -- not implemented
```



# Players and Performance

## Performance

- `Performance` = Haskore's internal representation of a music's interpretation
- `Music`  $\rightarrow$  `Performance` allows for different interpretations for the same music

```
type Performance = [Event]
data Event = Event {eTime    :: Time
                   ,eInst    :: IName
                   ,ePitch   :: AbsPitch
                   ,eDur     :: DurT
                   ,eVol     :: Volume
                   ,pFields  :: [Float]
                   } deriving (Eq,Ord,Show)

perform :: PMap -> Context -> Music -> Performance
```

# Input / Output

## MIDI and CSound files

### MIDI

Haskore has also built-in support for writing and reading MIDI files. MIDI ("Musical Instrument Digital Interface") is a standard protocol adopted by most, if not all, manufacturers of electronic instruments. In this way Haskore ensures it will be able to communicate with virtually any other decent music system.

# Input / Output

## MIDI and CSound files

### MIDI

Haskore has also built-in support for writing and reading MIDI files. MIDI ("Musical Instrument Digital Interface") is a standard protocol adopted by most, if not all, manufacturers of electronic instruments. In this way Haskore ensures it will be able to communicate with virtually any other decent music system.

### CSound

Furthermore, Haskore can also output to CSound files. CSound is a software synthesizer that allows its user to create a virtually unlimited number of sounds and instruments. By supporting CSound output, Haskore gives its users access to all the powerful features of a software sound synthesizer.

# Mathematic reasoning for music

- Haskore also allows for mathematical reasoning, due to
  - Notion of literal interpretation
  - Haskell being a pure language
- Example:

## Axiom 1

$\forall r_1, r_2, r_3, r_4$  and  $m$ :

$$\text{Tempo } r_1 \ r_2 \ (\text{Tempo } r_3 \ r_4 \ m) = \text{Tempo } (r_1 * r_3) \ (r_2 * r_4) \ m$$

## Proof:

```
perform dt (Tempo r1 r2 (Tempo r3 r4 m))
= perform (r2*dt/r1) (Tempo r3 r4 m)    -- unfolding perform
= perform (r4*(r2*dt/r1)/r3) m          -- unfolding perform
= perform ((r2*r4)*dt/(r1*r3)) m        -- simple arithmetic
= perform dt (Tempo (r1*r3) (r2*r4) m)  -- folding perform
```

# Haskore examples

## Beethoven's Für Elise - Original theme

### Für Elise theme + 12 tone techniques

Ludwig van Beethoven

Original

Piano RH

Piano LH

5



furElise

# Haskore examples

## Beethoven's Für Elise - Music

```
furElise :: Music
furElise =
  let up1 = (Note (E,5) sn []) :+: (Note (Ef,5) sn []) :+: (Note (E,5) sn [])
           :+: (Note (Ef,5) sn []) :+: (Note (E,5) sn []) :+: (Note (B,4) sn [])
           :+: (Note (D,5) sn []) :+: (Note (C,5) sn []) :+: (Note (A,4) en [])
           :+: (Rest sn) :+: (Note (C,4) sn []) :+: (Note (E,4) sn [])
           :+: (Note (A,4) sn []) :+: (Note (B,4) en []) :+: (Rest sn)
  up2 = (Note (E,4) sn []) :+: (Note (Gs,4) sn []) :+: (Note (B,4) sn [])
       :+: (Note (C,5) en []) :+: (Rest sn) :+: (Note (E,4) sn [])
  up3 = (Note (D,4) sn []) :+: (Note (C,5) sn []) :+: (Note (B,4) sn [])
       :+: (Note (A,4) qn [])
  up  = up1 :+: up2 :+: up1 :+: up3

  dw1 = (Rest hn) :+: (Note (A,2) sn []) :+: (Note (E,3) sn [])
       :+: (Note (A,3) sn []) :+: (Rest sn) :+: (Rest en)
       :+: (Note (E,1) sn []) :+: (Note (E,2) sn []) :+: (Note (Gs,3) sn [])
       :+: (Rest sn) :+: (Rest en) :+: (Note (A,2) sn [])
       :+: (Note (E,3) sn []) :+: (Note (A,3) sn []) :+: (Rest sn)
  dw  = dw1 :+: dw1

  in (Instr "Acoustic Grand Piano" (dw == up))
```

# Haskore examples

## Beethoven's Für Elise - Different performances

### Basic performance

```
myPerform = perform (const defPlayer) c  
  
  where c = Context 0 defPlayer "" (metro 300 en) (absPitch (C, 1)) 50
```

# Haskore examples

## Beethoven's Für Elise - Different performances

### Basic performance

```
myPerform = perform (const defPlayer) c  
  where c = Context 0 defPlayer "" (metro 300 en) (absPitch (C, 1)) 50
```

### Variation 1 - 2 octaves higher, double time

```
myPerform = perform (const defPlayer) c  
  where c = Context 0 defPlayer "" (metro 150 en) (absPitch (C, 3)) 50
```



# Haskore examples

## Beethoven's Für Elise - Different performances

### Basic performance

```
myPerform = perform (const defPlayer) c  
  where c = Context 0 defPlayer "" (metro 300 en) (absPitch (C, 1)) 50
```

### Variation 1 - 2 octaves higher, double time

```
myPerform = perform (const defPlayer) c  
  where c = Context 0 defPlayer "" (metro 150 en) (absPitch (C, 3)) 50
```

### Variation 2 - 1 octave lower, 2 semitones lower, double volume

```
myPerform = perform (const defPlayer) c  
  where c = Context 0 defPlayer "" (metro 150 en) (absPitch (A, 0)) 100
```

# Haskore examples

## Beethoven's Für Elise - Parallel transposition

Transpose piece, delay & play in parallel with original

```
(delay 1 (Trans 5 m)) ::= m
```

# Haskore examples

## Beethoven's Für Elise - Inverse

Für Elise theme + 12 tone techniques Ludwig van Beethoven

Inverse

The image displays a musical score for 'Für Elise' by Ludwig van Beethoven, illustrating the application of 12-tone techniques. The score is presented in two systems, each with a Piano Right Hand (RH) and Piano Left Hand (LH) part. The first system shows the original theme in the RH and its inverse in the LH. The second system shows the original theme in the RH and its inverse in the LH, with a measure number '5' indicating the start of the second system. The RH part consists of a sequence of eighth notes, while the LH part consists of a sequence of eighth notes. The RH part is in G major, and the LH part is in E minor. The score is written in 3/8 time.

invert furElise

# Haskore examples

## Beethoven's Für Elise - Retrograde

### Für Elise theme + 12 tone techniques

Ludwig van Beethoven

Retrograde

The image displays a musical score for the retrograde of the Für Elise theme. It consists of two systems of staves. The first system is labeled 'Piano RH' (Right Hand) and 'Piano LH' (Left Hand). The second system is labeled '5' and continues the piece. The score is written in 3/8 time and features a key signature of one flat (B-flat). The retrograde is indicated by a box labeled 'Retrograde' above the first staff. The notation includes various musical symbols such as notes, rests, and accidentals, representing the original theme played backwards.

retrograde furElise

# Haskore examples

## Beethoven's Für Elise - Inverse retrograde

Für Elise theme + 12 tone techniques

Ludwig van Beethoven

Inverse retrograde

The image displays a musical score for 'Für Elise' by Ludwig van Beethoven. It consists of four staves. The top two staves are labeled 'Piano RH' (Right Hand) and 'Piano LH' (Left Hand). The bottom two staves are unlabeled but correspond to the same parts. The top staff (Piano RH) shows the original theme, starting with a treble clef and a key signature of one sharp (F#). The bottom staff (Piano LH) shows the inverse retrograde of the theme, starting with a bass clef and a key signature of one flat (Bb). The score is divided into two systems, with a measure number '5' at the beginning of the second system. The first system contains four measures, and the second system contains four measures. The first three measures of each system show the original theme, and the fourth measure shows the inverse retrograde. The first measure of the first system is marked with a box labeled 'Inverse retrograde'.

(invert . retrograde) furElise