# 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

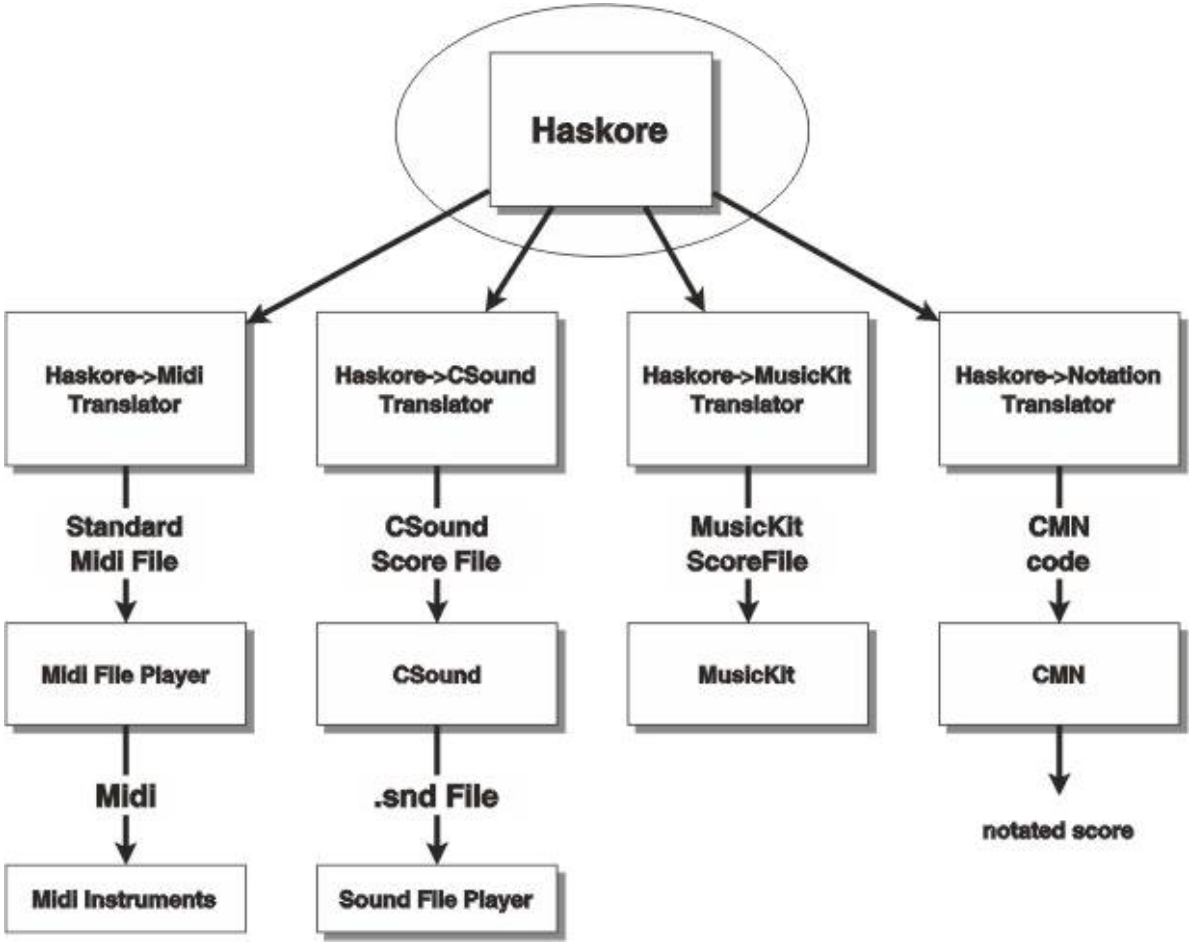
# The basics of Haskore

## 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 $m$:

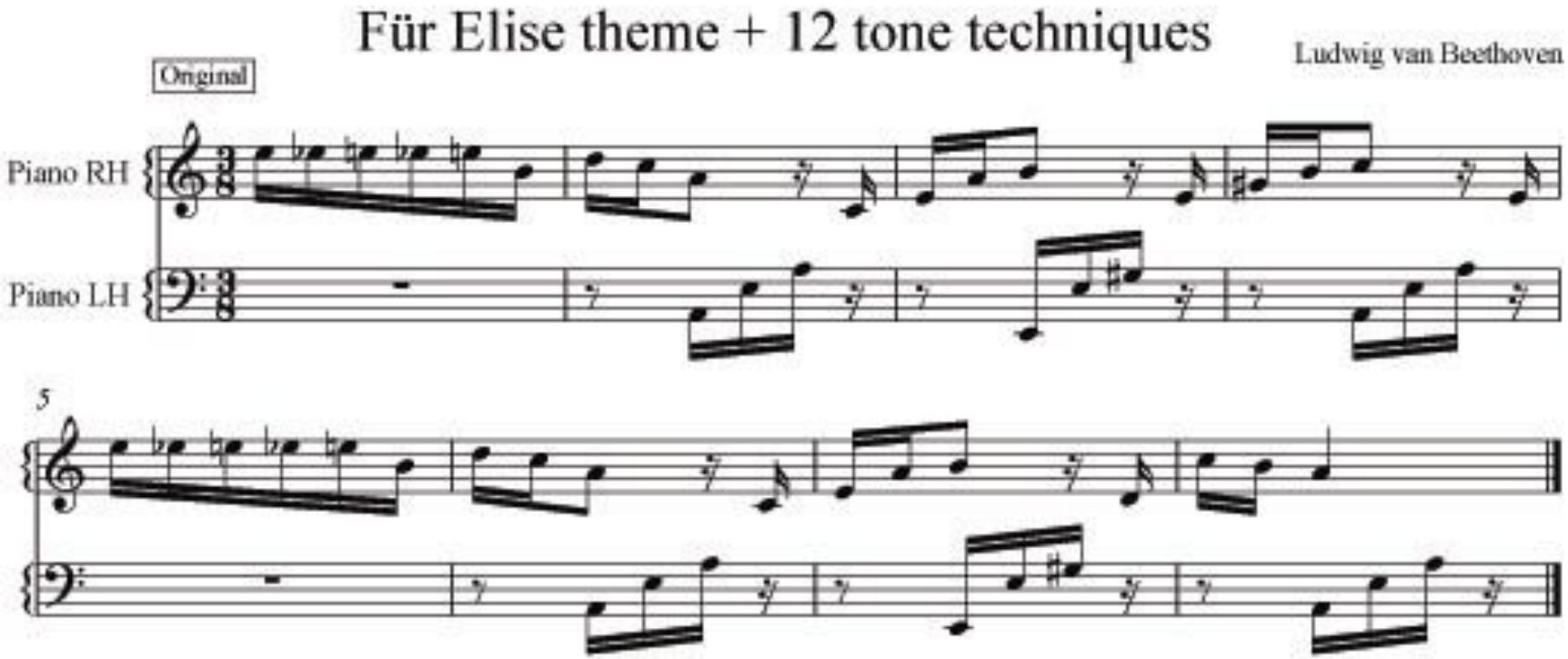
$$\text{Tempo } r_1\ r_2\ (\text{Tempo } r_3\ r_4\ m) = \text{Tempo } (r_1 * r_3)\ (r_2 * r_4)\ m$$

## 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




```
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

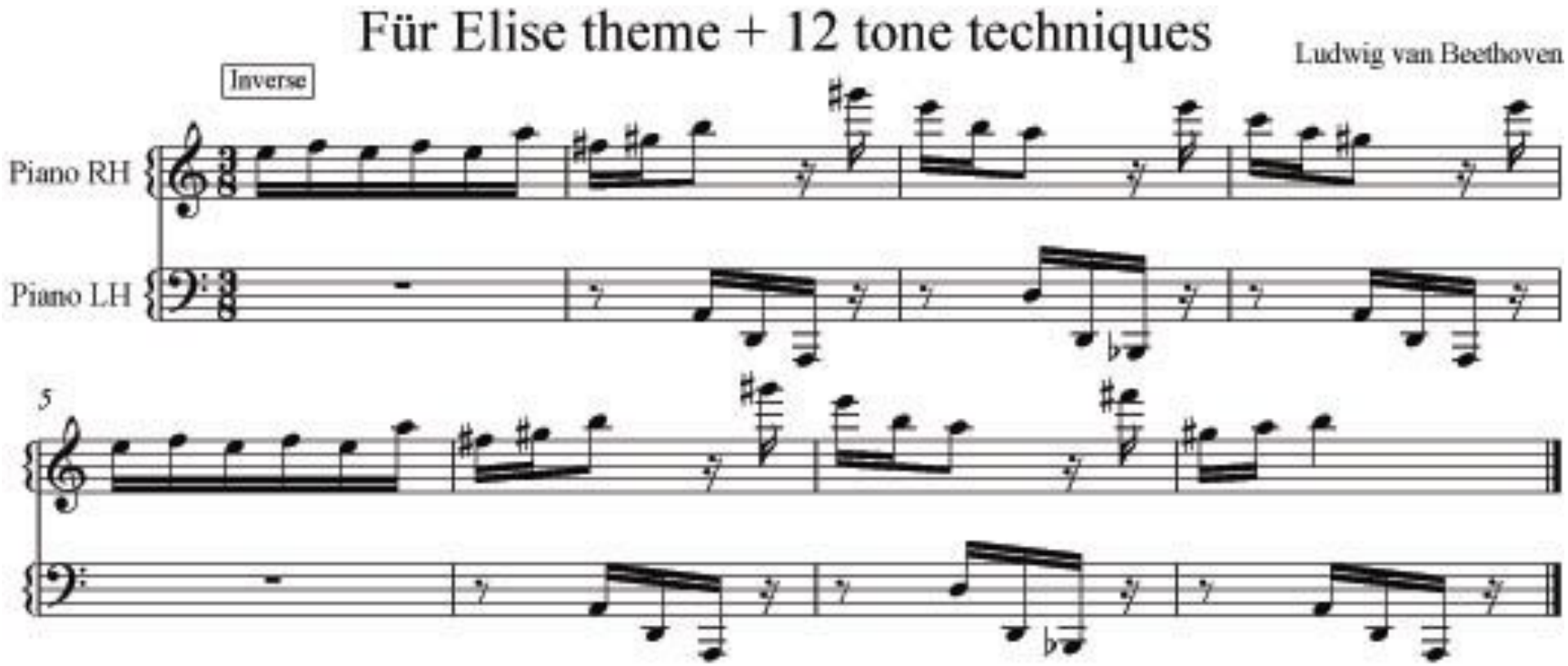
## Beethoven's Für Elise - Parallel transposition

### Transpose piece, delay & play in parallel with original

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

# Haskore examples

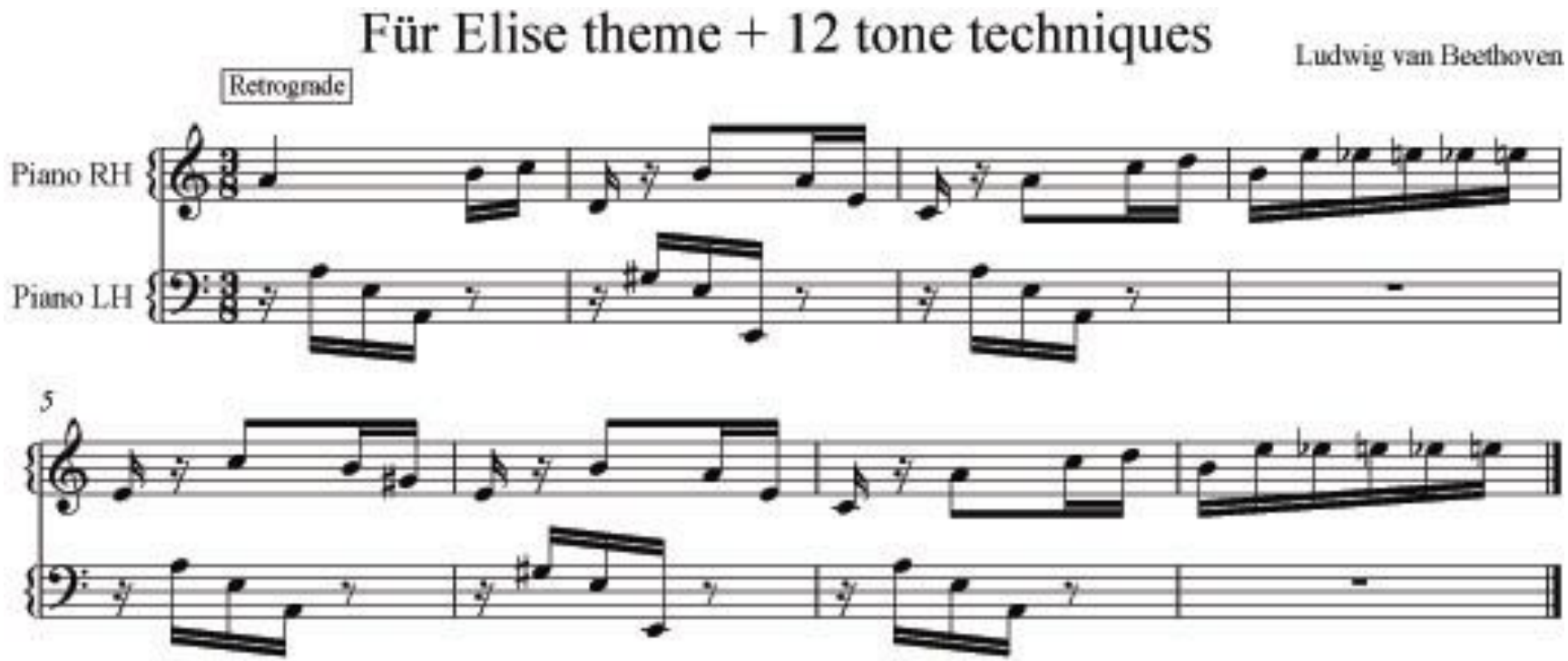
## Beethoven's Für Elise - Inverse




```
invert furElise
```

# Haskore examples

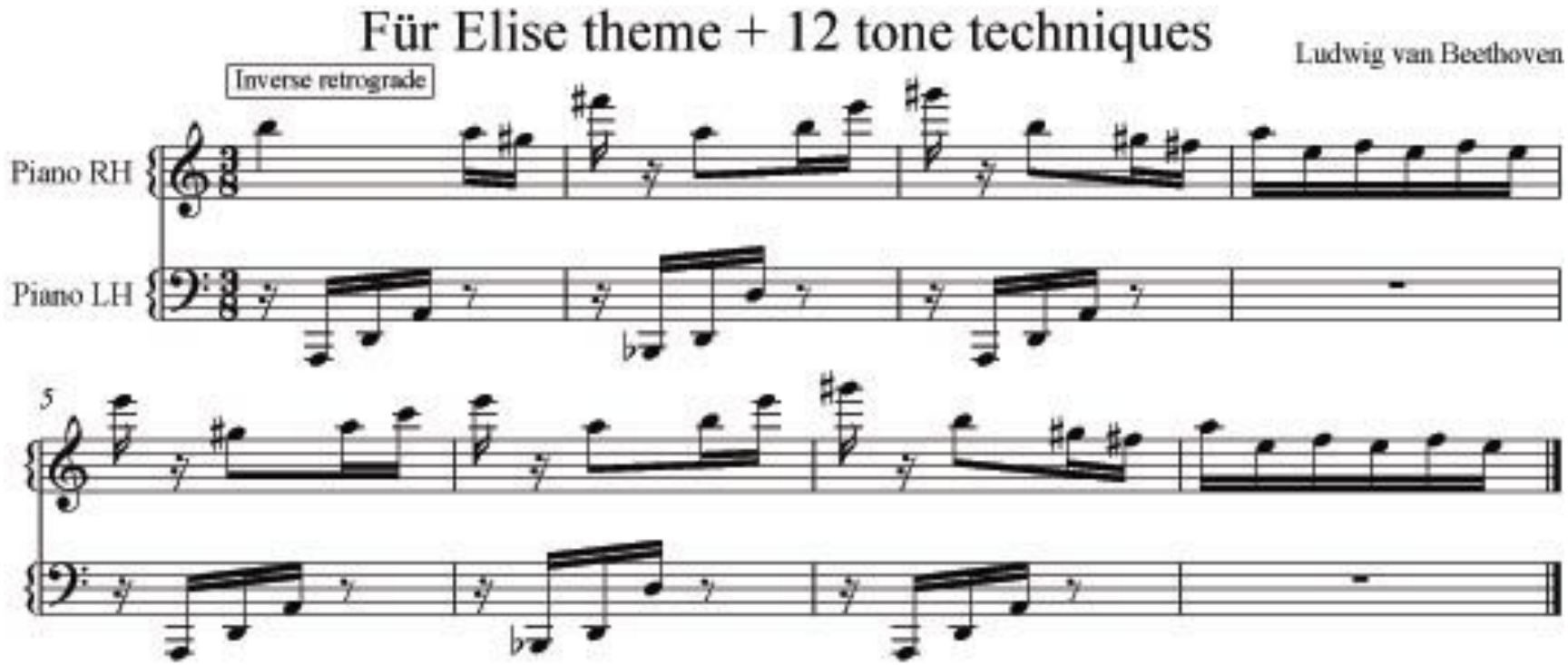
## Beethoven's Für Elise - Retrograde




```
retrograde furElise
```

# Haskore examples

## Beethoven's Für Elise - Inverse retrograde




```
(invert . retrograde) furElise
```