



Haskore

Musikbearbeitung mit Haskell

Paul Hudak, Yale University, USA

<http://haskell.org/haskore>



Der Datentyp Music

```
type Pitch      = (PitchClass, Octave) -- (A,4) = A440 (440Hz)
data PitchClass = Cf | C | Cs | Df | D | Ds | Ef | E | Es | Ff | F
               | Fs | Gf | G | Gs | Af | A | As | Bf | B | Bs
               deriving (Eq,Show)

type Octave     = Int
```

```
data Music =    Note Pitch Dur [NoteAttribute] | Rest Dur
              | Music :+: Music                | Music :=: Music
              | Tempo (Ratio Int) Music
              | Trans Int Music
              | Instr IName Music | Player PName Music
              | Phrase [PhraseAttribute] Music
              deriving (Show, Eq)
```

```
type Dur = Ratio Int -- (%) :: (Integral a) => a -> a -> Ratio a
```



Instrumente des MIDI Standards

- Acoustic Grand Piano | Bright Acoustic Piano | Electric Grand Piano
- | Honky Tonk Piano | Rhodes Piano | Chorused Piano | Harpsichord
- | Clavinet | Celesta | Glockenspiel | Music Box | Vibraphone | Marimba
- | Xylophone | Tubular Bells | Dulcimer | Hammond Organ
- | Percussive Organ | Rock Organ | Church Organ | Reed Organ
- | Accordion | Harmonica | Tango Accordion
- | Acoustic Guitar (nylon) | Acoustic Guitar (steel) | Electric Guitar (jazz)
- | Electric Guitar (clean) | Electric Guitar (muted) | Overdriven Guitar
- | Distortion Guitar | Guitar Harmonics | Acoustic Bass
- | Electric Bass (fingered) | Electric Bass (picked) | Fretless Bass
- | Slap Bass 1 | Slap Bass 2 | Synth Bass 1 | Synth Bass 2
- | Violin | Viola | Cello | Contrabass | Tremolo Strings
- | Pizzicato Strings | Orchestral Harp | Timpani | String Ensemble 1
- | String Ensemble 2 | Synth Strings 1 | Synth Strings 2 | Choir Aahs



... 129 an der Zahl

| Voice Oohs | SynthVoice | OrchestraHit | Trumpet | Trombone | Tuba
| Muted Trumpet | French Horn | Brass Section | Synth Brass 1 | Synth Brass 2
| Soprano Sax | Alto Sax | Tenor Sax | Baritone Sax | Oboe | Bassoon
| English Horn | Clarinet | Piccolo | Flute | Recorder | Pan Flute | Blown Bottle
| Shakuhachi | Whistle | Ocarina | Lead 1 (square) | Lead 2 (sawtooth)
| Lead 3 (calliope) | Lead 4 (chiff) | Lead 5 (charang) | Lead 6 (voice)
| Lead 7 (fifths) | Lead 8 (bass+lead) | Pad 1 (new age) | Pad 2 (warm)
| Pad 3 (polysynth) | Pad 4 (choir) | Pad 5 (bowed) | Pad 6 (metallic)
| Pad 7 (halo) | Pad 8 (sweep) | FX1 (train) | FX2 (soundtrack) | FX3 (crystal)
| FX4 (atmosphere) | FX5 (brightness) | FX6 (goblins) | FX7 (echoes) | FX8 (sci-fi)
| Sitar | Banjo | Shamisen | Koto | Kalimba | Bagpipe | Fiddle | Shanai | TinkleBell
| Agogo | Steel Drums | Woodblock | Taiko Drum | Melodic Drum | Synth Drum
| Reverse Cymbal | Guitar Fret Noise | [Breath Noise](#) | Seashore | Bird Tweet
| [TelephoneRing](#) | [Helicopter](#) | [Applause](#) | [Gunshot](#) | Percussion
deriving (Show,Eq,Ord,Enum)



Transponieren

```
type AbsPitch = Int
```

```
absPitch :: Pitch -> AbsPitch
```

```
absPitch (pc,oct) = 12*oct + pcToInt pc
```

```
pitch    :: AbsPitch -> Pitch
```

```
pitch ap = ( [C,Cs,D,Ds,E,F,Fs,G,Gs,A,As,B]  
             !! mod ap 12, quot ap 12 )
```

```
trans    :: Int -> Pitch -> Pitch
```

```
trans i p = pitch (absPitch p + i)
```

```
pcToInt :: PitchClass -> Int  
pcToInt pc = case pc of
```

```
Cf -> -1  -- should Cf be 11?  
C  -> 0  
Cs -> 1  
Df -> 1  
D  -> 2  
Ds -> 3  
Ef -> 3  
E  -> 4  
Es -> 5  
Ff -> 4  
F  -> 5  
Fs -> 6  
Gf -> 6  
G  -> 7  
Gs -> 8  
Af -> 8  
A  -> 9  
As -> 10  
Bf -> 10  
B  -> 11  
Bs -> 12 -- should Bs be 0?
```



Noten und Pausenbezeichnungen

cf, c, cs, df, d, ds, ef, e, es, ff, f, fs, gf, g, gs, af, a, as, bf, b, bs
:: Octave -> Dur -> [NoteAttribute] -> Music

cf o = Note (Cf,o); c o = Note (C,o); cs o = Note (Cs,o)
df o = Note (Df,o); d o = Note (D,o); ds o = Note (Ds,o)
...

wn, hn, qn, en, sn, tn, dhn, dqn, den, dsn **:: Dur**
wnr, hnr, qnr, enr, snr, tnr, dhnr, dqnr, denr, dsnr **:: Music**

wn = 1 ; **wnr = Rest wn** -- whole
hn = 1%2 ; **hnr = Rest hn** -- half
qn = 1%4 ; **qnr = Rest qn** -- quarter

...
dhn = 3%4 ; **dhnr = Rest dhn** -- dotted half
dqn = 3%8 ; **dqnr = Rest dqn** -- dotted quarter
...



Beispiel: Alle Jahre wieder

ajw :: Music

ajw = let o = 4 in

(g o dqn [] :+: a o en [] :+: g o qn [] :+: f o qn []) :+:

(e o hn [] :+: d o hn []) :+:

**(c o qn [] :+: d o en [] :+: e o en [] :+: f o qn [] :+:
e o qn []) :+:**

d o dhn [] :+: qnr :+:

e o qn [] :+: g o qn [] :+: a o qn [] :+: g o qn [] :+:

c (o+1) hn [] :+: b o qn [] :+: a o qn [] :+:

g o qn [] :+: f o en [] :+: e o en [] :+: f o qn [] :+:

g o qn [] :+: e o dhn [] :+: qnr



Abspielen von Musik

```
test :: Music -> IO ()
```

```
test m = outputMidiFile "test.mid"
```

```
    (performToMidi (testPerf m) defUpm)
```

```
testPerf :: Music -> Performance
```

```
testPerf m = perform defPMap defCon m
```

```
defCon      :: Context      -- a default Context
```

```
defPMap     :: String -> Player -- default player map
```

```
defUpm      :: UserPatchMap -- default user patch map
```

```
play m = do test m; system "wmplayer test.mid"; return ()
```




Nützliche Hilfsfunktionen

- **Notenfolgen und Akkorde:**

`line, chord` :: [Music] -> Music
`line` = foldr (:+:) (Rest 0)
`chord` = foldr (:=:) (Rest 0)

Beispiel: C major arpeggio und C Akkord:

`cMaj` = [n 4 qn [] | n <- [c, e, g]]
`cMajArp` = line cMaj
`cMajChd` = chord cMaj

- **Wiederholung (endlich oder unendlich)**

`repeatNM` :: Int -> Music -> Music
`repeatNM n m` = m :+: repeatM m (n-1)
`repeatNM 0 m` = Rest 0

- **Verzögerung**

`delay` :: Dur -> Music -> Music
`delay d m` = Rest d :+: m

- **Umkehrung**

`revM` :: Music -> Music



Musik rückwärts spielen

revM :: Music -> Music

revM n@(Note _ _ _) = n

revM r@(Rest _) = r

revM (Tempo a m) = Tempo a (revM m)

revM (Trans i m) = Trans i (revM m)

revM (Instr i m) = Instr i (revM m)

revM (Phrase pas m) = Phrase pas (revM m)

revM (m1 :+: m2) = revM m2 :+: revM m1

revM (m1 :=: m2) =

let d1 = dur m1 -- Musikdauer bestimmen

d2 = dur m2

in if d1>d2

then revM m1 :=: (Rest (d1-d2) :+: revM m2)

else (Rest (d2-d1) :+: revM m1) :=: revM m2



Weitere Beispiele

Wende n mal f iterativ auf Musik und g auf akkumulierte Sequenzen an:

rep :: (Music -> Music) -> (Music -> Music) ->
Int -> Music -> Music

rep f g 0 m = Rest 0

rep f g n m = m ::= g (rep f g ($n-1$) (f m))

Zum Beispiel:

run = **rep** (Trans 5) (delay tn) 8 (c 4 tn [])

cascade = **rep** (Trans 4) (delay en) 8 run

cascades = **rep** id (delay sn) 2 cascade

waterfall = cascades :+: revM cascades



Triller

trill :: Int -> Dur -> Music -> Music

trill i d n@(Note p nd as)

= if d >= nd then n else Note p d as

:+: trill (negate i) d

(Note (trans i p) (nd-d) as)

trill i d (Tempo a m) = Tempo a (trill i (d*a) m)

trill i d (Trans a m) = Trans a (trill i d m)

trill i d (Instr a m) = Instr a (trill i d m)

trill i d (Player a m) = Player a (trill i d m)

trill i d (Phrase a m) = Phrase a (trill i d m)

trill _ _ _ = error "Trill input must be a single note"

trilln :: Int -> Int -> Music -> Music

trilln i nTimes m = trill i (dur m / (nTimes%1)) m



Stars and Stripes forever

```
ssfMelody = line (m1 ++ m2 ++ m3 ++ m4)
```

```
m1 = [trilln 2 5 (bf 6 en []),
```

```
      Phrase [Art staccato] (line [ef 7 en [], ef 6 en [], ef 7 en []])] ]
```

```
m2 = [Phrase [Art legato]
```

```
      (line [bf 6 sn [], c 7 sn [], bf 6 sn [], g 6 sn []]),
```

```
      Phrase [Art staccato] (line [ef 6 en [], bf 5 en []])] ]
```

```
m3 = [Phrase [Art legato]
```

```
      (line [ef 6 sn [], f 6 sn [], g 6 sn [], af 6 sn []]),
```

```
      Phrase [Art staccato] (line [bf 6 en [], ef 7 en []])] ]
```

```
m4 = [trill 2 tn (bf 6 qn []), bf 6 sn [], denr]
```

```
ssf = Instr "flute" (Tempo 2 (ssfMelody))
```

Überblick

Haskore

