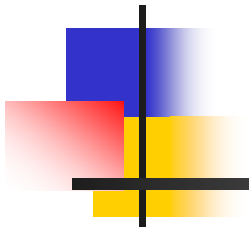
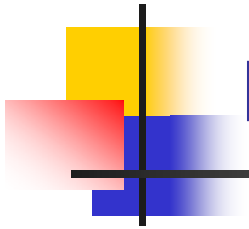


Haskore
Musikbearbeitung mit Haskell

Paul Hudak, Yale University, USA

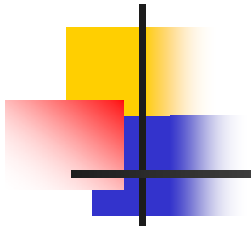
http://haskell.org/haskore



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          | Rest Dur
                  | Music :+: Music       | Music :=: Music
                  | Tempo  (Ratio Int) Music
                  | Trans  Int  Music
                  | Instr  IName Music
                  deriving (Show, Eq)
type Dur   = Ratio Int   -- (%) :: (Integral a) => a -> a -> Ratio a
```

Musical Instrument Digital Interface

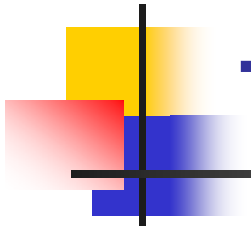


Instrumente des MIDI Standards

```
data IName = AcousticGrandPiano  | BrightAcousticPiano | ElectricGrandPiano
    | HonkyTonkPiano      | RhodesPiano         | ChorusedPiano  | Harpsichord
    | Clavinet        | Celesta | Glockenspiel  | MusicBox  | Vibraphone | Marimba
    | Xylophone     | TubularBells   | Dulcimer     | HammondOrgan
    | PercussiveOrgan | RockOrgan | ChurchOrgan         | ReedOrgan
    | Accordion            | Harmonica          | TangoAccordion
    | AcousticGuitarNylon   | AcousticGuitarSteel | ElectricGuitarJazz
    | ElectricGuitarClean   | ElectricGuitarMuted | OverdrivenGuitar
    | DistortionGuitar      | GuitarHarmonics     | AcousticBass
    | ElectricBassFingered  | ElectricBassPicked  | FretlessBass
    | SlapBass1            | SlapBass2           | SynthBass1 | SynthBass2
    | Violin        | Viola | Cello  | Contrabass | TremoloStrings
    | PizzicatoStrings      | OrchestralHarp      | Timpani  | StringEnsemble1  .....
```

... 129 an der Zahl

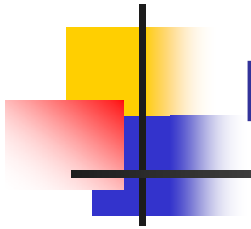
```
| StringEnsemble2     | SynthStrings1       | SynthStrings2         | ChoirAahs
| VoiceOohs | SynthVoice   | OrchestraHit   | Trumpet   | Trombone  | Tuba
| MutedTrumpet          | FrenchHorn          | BrassSection | SynthBrass1
| SynthBrass2     | SopranoSax  | AltoSax | TenorSax  BaritoneSax | Oboe
| Bassoon  | EnglishHorn  | Clarinet  | Piccolo  | Flute    | Recorder | PanFlute
| BlownBottle | Shakuhachi   | Whistle  | Ocarina  | Lead1Square
| Lead2Sawtooth  | Lead3Calliope  | Lead4Chiff  | Lead5Charang | Lead6Voice
| Lead7Fifths   | Lead8BassLead  | Pad1NewAge | Pad2Warm  | Pad3Polysynth
| Pad4Choir  | Pad5Bowed | Pad6Metallic | Pad7Halo | Pad8Sweep  | FX1Train
| FX2Soundtrack | FX3Crystal | FX4Atmosphere | FX5Brightness  FX6Goblins
| FX7Echoes | FX8SciFi | Sitar | Banjo  | Shamisen  | Koto | Kalimba  | Bagpipe
| Fiddle | Shanai  | TinkleBell  | Agogo | SteelDrums | Woodblock | TaikoDrum
| MelodicDrum | SynthDrum | ReverseCymbal | GuitarFretNoise  | BreathNoise
| Seashore | BirdTweet  | TelephoneRing  | Helicopter | Applause | Gunshot
| Percussion                          deriving (Show,Eq,Ord,Enum)
```



Transponieren

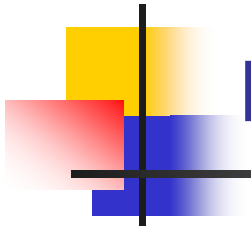
```
type AbsPitch = Int

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?
```



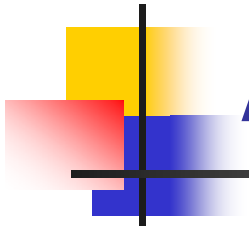
Noten und Pausenbezeichnungen

```
cf, c, cs, df, d, ds, ef, e, es, ff, f, fs, gf, g, gs, af, a, as, bf, b, bs
          :: Octave -> Dur -> 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 (perform defCon m))

defCon :: Context
defCon = Context {   cTime   = 0,
                     cInst   = AcousticGrandPiano,
                     cDur    = metro 120 qn,
                     cKey    = 0 }

testNT    m = do     test m
                     system "mplay32   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**

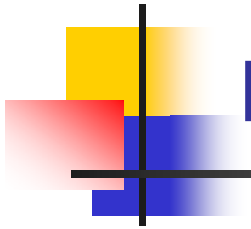
```
delay           :: Dur -> Music -> Music
delay d m       = Rest d :+: m
```

- **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 (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
```



Begleitung

- Aus dem Midi Standard:

```
data PercussionSound = AcousticBassDrum          -- MIDI Key 35
        | BassDrum1 | SideStick | AcousticSnare
        |   . . .    | MuteCuica | OpenCuica | MuteTriangle
        | OpenTriangle                           -- MIDI Key 82
        deriving (Show,Eq,Ord,Ix,Enum)
```

- Umwandlung in Musik:

```
perc :: PercussionSound -> Dur -> Music
perc ps = Note (pitch (fromEnum ps + 35))
```

- Beispiel:

```
funkGroove = let  p1 = perc LowTom qn
                  p2 = perc AcousticSnare en
          in Tempo 3 (Instr Percussion (cut 8 (repeatM
                ((p1 :+: qnr :+: p2 :+: qnr :+: p2 :+:
                        p1 :+: p1 :+: qnr :+: p2 :+: enr)
                        :=: roll en (perc ClosedHiHat 2) ) )))
```

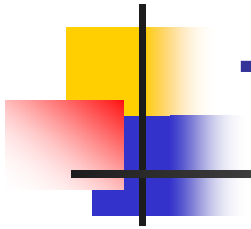
Weitere Beispiele

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))
```

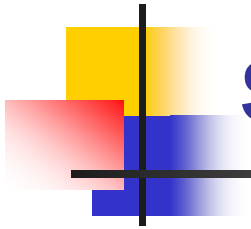
Zum Beispiel:

```
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

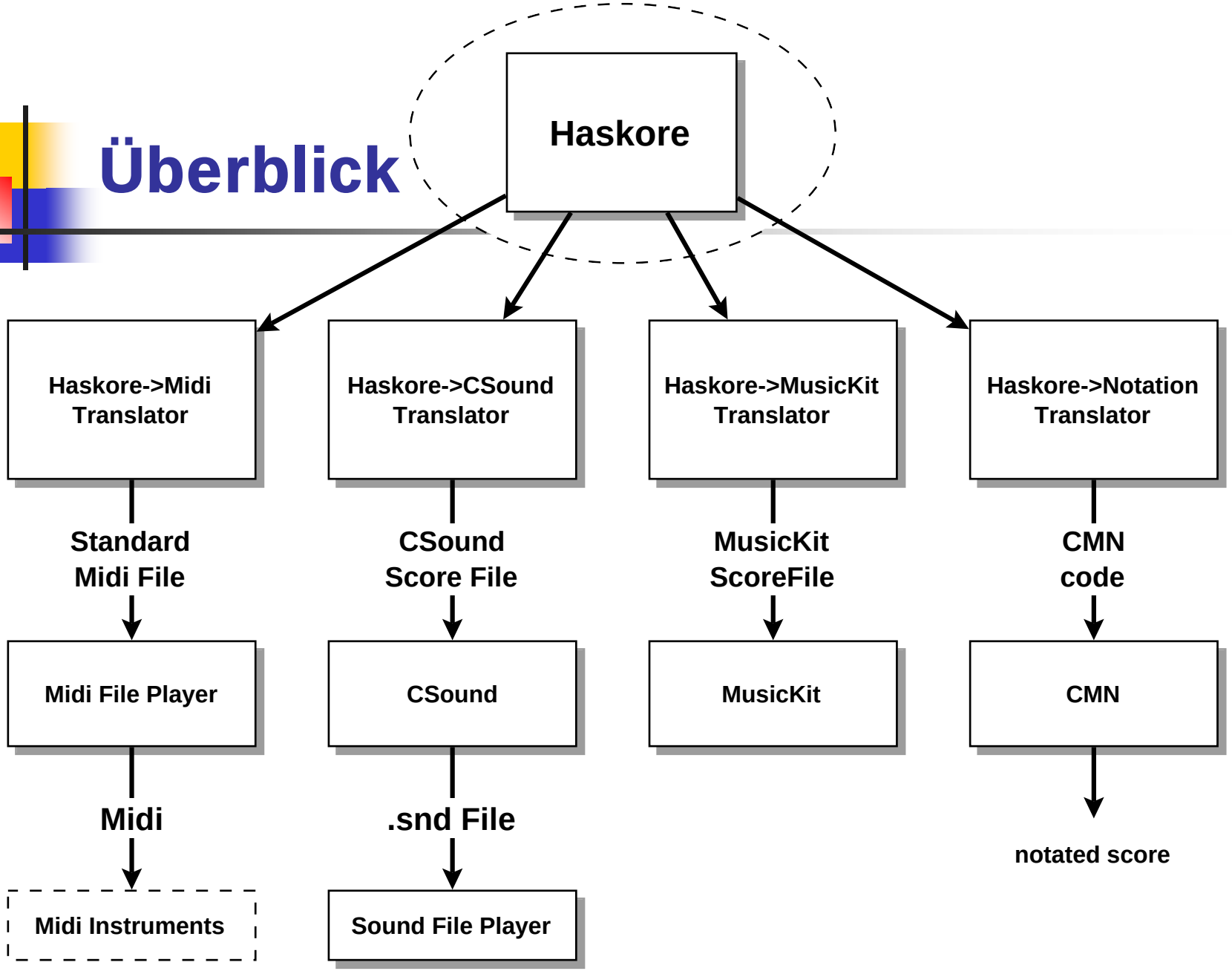
```
trill :: Int -> Dur -> Music -> Music
trill i d n@(Note p nd)
   = if d >= nd then n  else Note p d
                                  :+: trill (negate i) d
                                        (Note (trans i p) (nd-d))
trill i d (Tempo a m)   = Tempo  a (trill i (d*a) m)
trill i d (Trans a m)   = Trans  a (trill i d m)
trill i d (Instr a m)   = Instr  a (trill i d m)
trill _ _ _             = 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 = m1 :+: m2 :+: m3 :+: m4
m1 = trilln 2 5 (bf 6 en) :+:  line [ef 7 en, ef 6 en, ef 7 en]
m2 = line [bf 6 sn, c 7 sn, bf 6 sn, g 6 sn, ef 6 en, bf 5 en]
m3 = line [ef 6 sn, f 6 sn, g 6 sn, af 6 sn, 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
Haskore->Midi Translator
Haskore->CSound Translator
Haskore->MusicKit Translator
Haskore->Notation Translator
Standard Midi File
CSound Score File
MusicKit ScoreFile
CMN code
Midi File Player
CSound
MusicKit
CMN
Midi
.snd File
notated score
Midi Instruments
Sound File Player