



Von Mozart zur Informatik

Wie programmiert man Musik?

Wie komponiert man Programme?

Michael Hanus

Christian-Albrechts-Universität zu Kiel

Vielfältige Berührungspunkte:

- Computer-generierte Musik
- Informatikmethoden zur Speicherung von Musik (MP3)
- Suche nach Musik (Music Information Retrieval)

Konzeptuelle Gemeinsamkeiten:

- Dynamische Abläufe: Aufführung ↔ Berechnung
- Techniken zur Beherrschung dieser Abläufe

Musikalische Prinzipien aus der Schule bekannt:
(Informatik leider immer noch nicht!)

Erläuterung von Prinzipien der Informatik anhand von Musik

guter Informatiker = guter Komponist?



Analyse der mathematischen Verhältnisse alte Idee:

Pythagoras (ca. 500 v. Chr.): Monochord

- Zahlenverhältnisse der Musikintervalle
- Unterteilung einer Saite 1:2 : Abstand einer Oktave
- erste Tonleiter der Weltgeschichte

Marin Mersenne (17. Jahrhundert):

- physikalische Erklärung der Zahlenverhältnisse
- eine Oktave \approx doppelte Tonfrequenz

Beschreibung von Musik durch Frequenzanalyse?

- gut zur physikalischen Speicherung
- ungeeignet zum Aufschreiben durch Komponisten



ABSTRAKTION, MODELLIERUNG, NOTATION

Wichtige Prinzipien zur Beherrschung komplexer Systeme:

Abstraktion: ignoriere Unwichtiges (z.B. Obertöne), Beschränkung auf wesentliche Prinzipien

Modellierung: bilde ein geeignetes Modell der abstrahierten Prinzipien (z.B. zwölf Töne einer Oktave)

Notation: finde geeignete kompakte Notation (für Experten verständlich)

Musik ohne diese Prinzipien: Musik von Mozart heute noch erfahrbar?

Diese Begriffe sind auch zentral für Informatik-Systeme!



Notationen sind Ausdruck von Ideen:

Musik: Notensystem

Informatik: Programme, *ER-Diagramme*, *UML*

Bedeutung von Notationen:

nichtssagend für Laien

Bedeutung muss erlernt werden

Notation müssen adäquat sein:

Ideen möglichst einfach und präzise ausdrücken, z.B.

→ Spielweise von Instrumenten

→ Lösung/Berechnung von Problemen

Gute Notationen in der Musik, aber Informatik?



Abstraktion: Ton eines Musikinstrumentes:

- Reduktion auf Grundton, Instrumentenart und -spielweise
- ignoriere Obertöne, Klangverlauf,...

Modellierung: unterteile Frequenzspektrum in Töne:

- Oktaven, bestehend aus zwölf Tönen
- Tonarten
- Notenwerte, Tempo

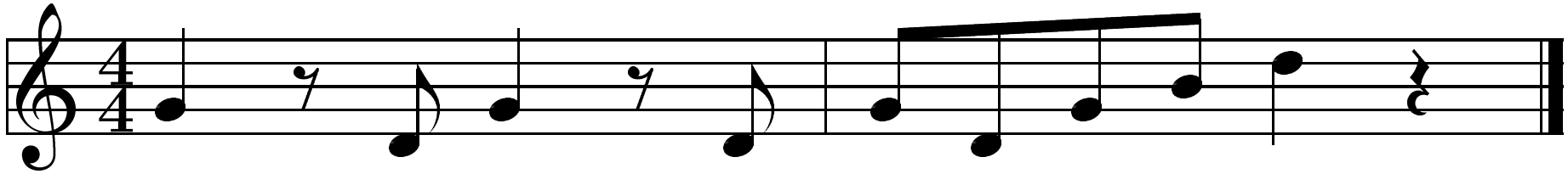
Notation: Notensystem

- Liniensystem
- Notenschlüssel
- Vorzeichen

kompakte und für Experten verständliche Notation



LASST MOZART BEGINNEN...



Programmierung von Musik:

Übersetzung in eine für den Computer verständliche Notation

- kein „Scannen“ der Notenschrift
- äquivalente *textuelle Darstellung*

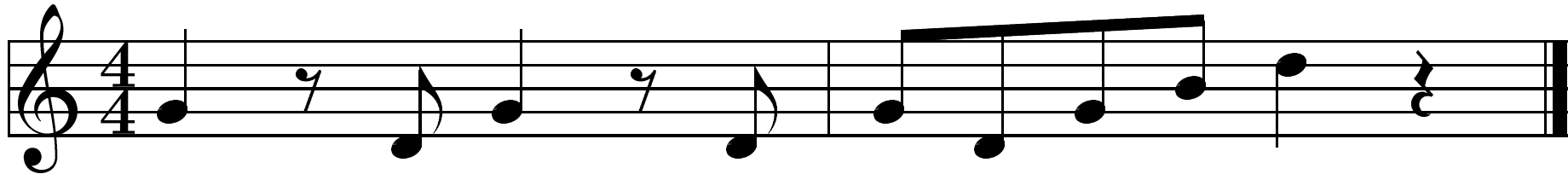
1. Ansatz: Darstellung von Noten durch entsprechende Zeichen:

(~> Buchstabentonschrift mit Oktavbereichen)

g' d' g' d' g' d' g' h' d''



LASST MOZART BEGINNEN... (2. VERSUCH)



Modellierung unzureichend:

Darstellung durch Tonhöhen reicht nicht aus, es fehlen

→ Pausen

→ Notenwerte

g' (1/4) Pause (1/8) d' (1/8) g' (1/4) Pause (1/8) d' (1/8)
g' (1/8) d' (1/8) g' (1/8) h' (1/8) d'' (1/4) Pause (1/4)



PROGRAMMIERUNG VON MUSIK: NOTEN UND PAUSEN

Bisherige Darstellung:

g' (1/4) Pause (1/8) d' (1/8) g' (1/4) Pause (1/8) d' (1/8)
g' (1/8) d' (1/8) g' (1/8) h' (1/8) d'' (1/4) Pause (1/4)

Mehrfaches Vorkommen identischer Elemente:

- Notenwerte: (1/4) (1/8) ...
- Pausen: Pause (1/8) ...

Informatiker sind bequem (sind ja auch nur Menschen...):

Definiere Abkürzungen für immer wiederkommende Elemente!



ABKÜRZUNGEN

Abkürzung:

- gebe komplexen Objekt einen Namen (z.B. Kfz = Kraftfahrzeug)
- schreibe diesen Namen anstelle des komplexen Objekts
- Vorteil: Verbesserung der Lesbarkeit (falls Abkürzung einprägsam)
(z.B. *Bafög* statt *Bundesausbildungsförderungsgesetz*)

Abkürzungen für Notenwerte

gN = (1/1) -- ganze Note
hN = (1/2) -- halbe Note
vN = (1/4) -- viertel Note
aN = (1/8) -- achtel Note
sN = (1/16) -- sechzehntel Note

...

und Pausenwerte

gP = Pause gN -- ganze Pause
hP = Pause hN -- halbe Pause
vP = Pause vN -- viertel Pause
aP = Pause aN -- achtel Pause
sP = Pause sN -- sechzehntel Pause



PROGRAMMIERUNG VON MUSIK MIT ABKÜRZUNGEN



Bisherige Darstellung:

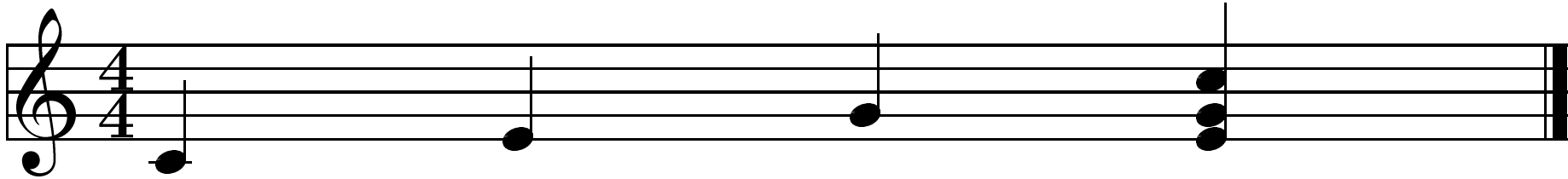
g' (1/4) Pause (1/8) d' (1/8) g' (1/4) Pause (1/8) d' (1/8)
g' (1/8) d' (1/8) g' (1/8) h' (1/8) d'' (1/4) Pause (1/4)

mit obigen Abkürzungen:

g' vN aP d' aN g' vN aP d' aN
g' aN d' aN g' aN h' aN d'' vN vP



ZUSAMMENSETZEN VON NOTEN



Bisherige Darstellung:

c' vN e' vN g' vN e' vN g' vN c'' vN

unzureichend, da kein Unterschied zwischen

→ Noten **nacheinander** spielen (c' vN e' vN g' vN)

→ Noten **gleichzeitig** spielen (e' vN g' vN c'' vN)

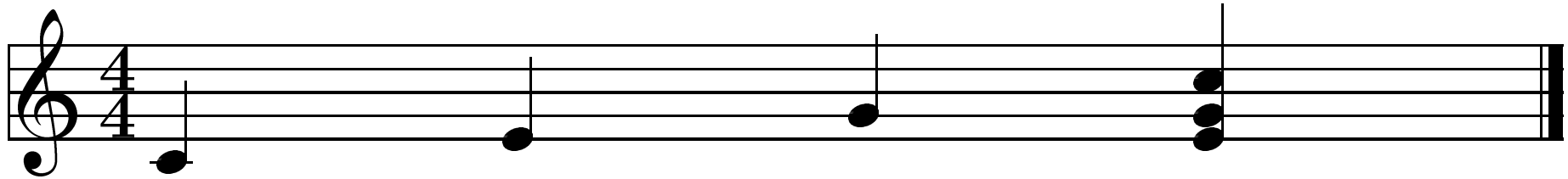
Modelliere die Art der Zusammensetzung durch **Kompositionsoperatoren**:

$m1$ **:+:** $m2$: spiele Musik $m1$ und $m2$ **nacheinander**

$m1$ **:=:** $m2$: spiele Musik $m1$ und $m2$ **gleichzeitig**



KOMPOSITIONSOPERATOREN FÜR MUSIK



Darstellung mit Kompositionsoperatoren:

$c' \ vN \ :+ : e' \ vN \ :+ : g' \ vN \ :+ : (e' \ vN \ := : g' \ vN \ := : c'' \ vN)$

Kompositionsoperatoren:

- Ausdrucksmittel, um größere Strukturen zu bilden
- Beispiel Mathematik: $1 * (3 + (4 * 5))$
 - “+”: bilde die Summe zweier Ausdrücke
 - “*”: bilde das Produkt zweier Ausdrücke
- Beispiel Musik:
 - hintereinander notieren: nacheinander spielen
 - übereinander notieren: gleichzeitig spielen (Noten, Begleitung, . . .)



Musik ist zusammengesetzt aus

- ① Noten einer bestimmten Tonhöhe und Dauer
- ② Pausen einer bestimmten Dauer
- ③ nacheinander gespielte Musik (*sequenzielle Komposition*)
- ④ gleichzeitig gespielte Musik (*parallele Komposition*)

Musikdaten: (lese “|” als Alternative)

```
data Musik = Note GanzeZahl Bruch
           | Pause Bruch
           | Musik :+: Musik
           | Musik :=: Musik
```

Anmerkung: Tonhöhe nicht als Frequenz, sondern als Halbtonwerte:

c' hN = Note 0 (1/2)

e' vN = Note 4 (1/4)

c'' aN = Note 12 (1/8)



NAMEN FÜR HALBTONWERTE

Nützlich: Namen (ce, cis, de, dis,...) statt Zahlen für Halbtonwerte

~> Programme lesbarer für den Menschen

„Programme müssen geschrieben werden, damit Menschen sie lesen, und nur nebenbei, damit Maschinen sie ausführen.“

(Abelson/Sussman: Struktur und Interpretation von Computerprogrammen, 2001)

ce = 0 cis = 1 des = 1

de = 2 dis = 3 es = 3

...

Hiermit:

c' hN = Note ce (1/2)

d' vN = Note de (1/4)

c'' aN = Note (ce+12) (1/8)



... UND WO BLEIBT DIE INFORMATIK?

Bisher:

- Übersetzung: 2-dimensionales Notensystem → Zeichenfolge
- Zeichenfolge durch Computer verarbeitbar (z.B. Klangsynthese)

Informatik:

- Auffinden regulärer Strukturen in Anwendungen
- Modellierung und Implementierung dieser Strukturen
- dadurch: Bereitstellung von Lösungen für *Klassen* von Problemen
- Parametrisierung der Strukturen
- konkrete Instanz durch Festlegung der Parameter

Nächster Schritt: finde reguläre Strukturen in Musikkompositionen



AKKORDE

Akkord: Zusammenklang von mehr als zwei Tönen

Beispiel: C-Durdreiklang: $c' \ vN ::= e' \ vN ::= g' \ vN$

in Musikdaten: $\text{Note } 0 \ (1/4) ::= \text{Note } 4 \ (1/4) ::= \text{Note } 7 \ (1/4)$

Allgemeiner Aufbau von Durdreiklängen:

Grundton $::= \text{Grundton}+4 ::= \text{Grundton}+7$

Formal in Musikdaten (t : Grundtonhöhe, d : Dauer):

$\text{durDreiklang } t \ d = \text{Note } t \ d ::= \text{Note } (t+4) \ d ::= \text{Note } (t+7) \ d$

\leadsto **Abkürzung mit Parametern, Funktionen**

Beispiele: C-Durdreiklang: $\text{durDreiklang } ce \ (1/4)$

F-Durdreiklang: $\text{durDreiklang } ef \ (1/4)$



BEDEUTUNG VON ABKÜRZUNGEN

`durDreiklang t d = Note t d ::= Note (t+4) d ::= Note (t+7) d`

Bedeutung der Abkürzungen/Funktionen: Vorschriften zum **Ausrechnen**

- ① Parameter bekannt (Werte für `t` und `d`) \rightsquigarrow setze diese ein
- ② ersetze Abkürzung durch Definition (rechte Seite)
- ③ rechne rechte Seite weiter aus

Analog zur **Anwendung von Formeln** (Mathematik, Physik, ...)

Flächeninhalt eines Quadrats: `qflaeche r = r*r`

Konkretes Quadrat: `r=5`:

`qflaeche 5 = 5*5` (*Einsetzen der Definition*)
`= 25` (*Ausrechnen der rechten Seite*)



AUSRECHNEN VON MUSIKFORMELN

$\text{durDreiklang } t \ d = \text{Note } t \ d ::= \text{Note } (t+4) \ d ::= \text{Note } (t+7) \ d$

Bedeutung der Abkürzungen/Funktionen: Vorschriften zum **Ausrechnen**

- ① Parameter bekannt (Werte für t und d) \rightsquigarrow setze diese ein
- ② ersetze Abkürzung durch Definition (rechte Seite)
- ③ rechne rechte Seite weiter aus

D-Durdreiklang: $\text{durDreiklang } de \ (1/4)$

Ausrechnen: $t = de$, $d = (1/4)$:

$\text{durDreiklang } de \ (1/4)$

$= \text{Note } de \ (1/4) ::= \text{Note } (de+4) \ (1/4) ::= \text{Note } (de+7) \ (1/4)$

$= \text{Note } 2 \ (1/4) ::= \text{Note } (2+4) \ (1/4) ::= \text{Note } (2+7) \ (1/4) \quad (de = 2)$

$= \text{Note } 2 \ (1/4) ::= \text{Note } 6 \ (1/4) ::= \text{Note } 9 \ (1/4)$



WEITERE MUSIKALISCHE ABKÜRZUNGEN

Molldreiklänge: kleine Terz + große Terz

mollDreiklang $t d = \text{Note } t d ::= \text{Note } (t+3) d ::= \text{Note } (t+7) d$

mollDreiklang $ce (1/2)$

$= \text{Note } 0 (1/2) ::= \text{Note } 3 (1/2) ::= \text{Note } 7 (1/2)$

Quintenintervall (leere Quinte) zum Grundton t :

quinte $t d = \text{Note } t d ::= \text{Note } (t+7) d$

Verlängerung einer Dauer d um die Hälfte ihres Wertes:

punktiert $d = (3/2) * d$

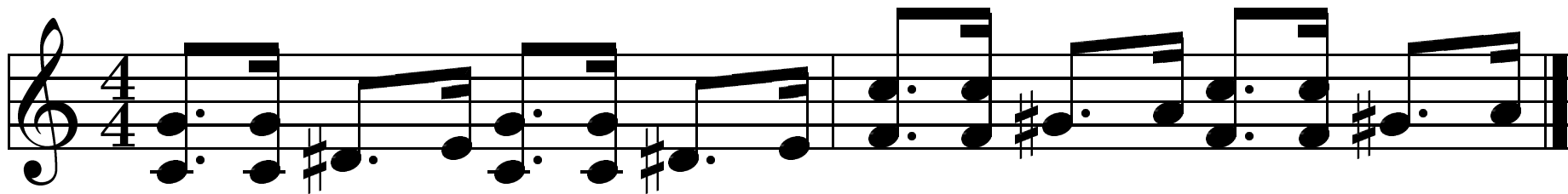
punktiert $vN = (3/8)$

Einfache Wiederholung von Musik m :

wdh $m = m :+: m$



BEISPIEL: BLUES



Programmierung:

- Noten nicht direkt abschreiben
- ähnliche Strukturen finden und mit Funktionen umsetzen

Hier: Halbtakte identisch, Takte identisch bis auf Grundton

```
phrase t = quinte t (punktiert aN) :+: quinte t sN :+:  
          Note (t+3) (punktiert aN) :+: Note (t+4) sN
```

Hiermit Blues programmieren:

```
blues t = wdh (phrase t) :+: wdh (phrase (t+5)) :+: wdh (phrase t)  
        :+: phrase (t+7) :+: phrase (t+5)
```

```
meinBlues = wdh (blues be) :+: quinte be hN
```



VORTEILE DES PROGRAMMIERUNG

```
phrase t = quinte t (punktiert aN) :+: quinte t sN :+:  
          Note (t+3) (punktiert aN) :+: Note (t+4) sN
```

```
blues t = wdh (phrase t) :+: wdh (phrase (t+5)) :+: wdh (phrase t)  
          :+: phrase (t+7) :+: phrase (t+5)
```

```
meinBlues = wdh (blues be) :+: quinte be hN
```

Bluesprogramm:

- kompakte Notation
- Ähnlichkeiten explizit \rightsquigarrow verbessertes Strukturverständnis
- ohne Programmierung: aufwändiges Aufschreiben



VORTEILE DES PROGRAMMIERUNG

meinBlues =

```
(((((Note 10 (3/16) :=: Note 17 (3/16)) :+: (Note 10 (1/16) :=: Note 17
(1/16)) :+: Note 13 (3/16) :+: Note 14 (1/16)) :+: (Note 10 (3/16) :=: Note 17
(3/16)) :+: (Note 10 (1/16) :=: Note 17 (1/16)) :+: Note 13 (3/16) :+: Note 14
(1/16)) :+: ((Note 15 (3/16) :=: Note 22 (3/16)) :+: (Note 15 (1/16) :=: Note
22 (1/16)) :+: Note 18 (3/16) :+: Note 19 (1/16)) :+: (Note 15 (3/16) :=: Note
22 (3/16)) :+: (Note 15 (1/16) :=: Note 22 (1/16)) :+: Note 18 (3/16) :+: Note
19 (1/16)) :+: ((Note 10 (3/16) :=: Note 17 (3/16)) :+: (Note 10 (1/16) :=:
Note 17 (1/16)) :+: Note 13 (3/16) :+: Note 14 (1/16)) :+: (Note 10 (3/16) :=:
Note 17 (3/16)) :+: (Note 10 (1/16) :=: Note 17 (1/16)) :+: Note 13 (3/16) :+:
Note 14 (1/16)) :+: ((Note 17 (3/16) :=: Note 24 (3/16)) :+: (Note 17 (1/16)
:=: Note 24 (1/16)) :+: Note 20 (3/16) :+: Note 21 (1/16)) :+: (Note 15 (3/16)
:=: Note 22 (3/16)) :+: (Note 15 (1/16) :=: Note 22 (1/16)) :+: Note 18 (3/16)
:+: Note 19 (1/16)) :+: ((Note 10 (3/16) :=: Note 17 (3/16)) :+: (Note 10
(1/16) :=: Note 17 (1/16)) :+: Note 13 (3/16) :+: Note 14 (1/16)) :+: (Note 10
(3/16) :=: Note 17 (3/16)) :+: (Note 10 (1/16) :=: Note 17 (1/16)) :+: Note 13
(3/16) :+: Note 14 (1/16)) :+: ((Note 15 (3/16) :=: Note 22 (3/16)) :+: (Note
15 (1/16) :=: Note 22 (1/16)) :+: Note 18 (3/16) :+: Note 19 (1/16)) :+: (Note
15 (3/16) :=: Note 22 (3/16)) :+: (Note 15 (1/16) :=: Note 22 (1/16)) :+: Note
18 (3/16) :+: Note 19 (1/16)) :+: ((Note 10 (3/16) :=: Note 17 (3/16)) :+:
(Note 10 (1/16) :=: Note 17 (1/16)) :+: Note 13 (3/16) :+: Note 14 (1/16)) :+:
(Note 10 (3/16) :=: Note 17 (3/16)) :+: (Note 10 (1/16) :=: Note 17 (1/16))
:+: Note 13 (3/16) :+: Note 14 (1/16)) :+: ((Note 17 (3/16) :=: Note 24
(3/16)) :+: (Note 17 (1/16) :=: Note 24 (1/16)) :+: Note 20 (3/16) :+: Note 21
(1/16)) :+: (Note 15 (3/16) :=: Note 22 (3/16)) :+: (Note 15 (1/16) :=: Note
22 (1/16)) :+: Note 18 (3/16) :+: Note 19 (1/16)) :+: Note 10 (1/2) :=: Note
17 (1/2)
```



MEHR MUSIK!

Einfache Wiederholung von Musik:

```
wdh m = m :+: m
```

Musik m dreimal spielen:

```
wdh3 m = m :+: wd3 m
```

Musik m viermal spielen:

```
wdh4 m = m :+: wd3 m
```

...

Musik m n -mal spielen:

```
mehrfach n m = if n>1 then m :+: mehrfach (n-1) m  
              else m
```

Fallunterscheidung!

(alles programmierbar mit Funktionen+Fallunterscheidung)

Kaufhausmusik:

```
endlos m = m :+: endlos m
```



MEHRSTIMMIGKEIT

Bru- der Ja- kob, Bru- der Ja- kob,

Bru- der Ja- kob, Bru- der Ja- kob,

Bru- der Ja- kob, Bru- der Ja- kob,

phrase1 = c' vN :+: d' vN :+: e' vN :+: c' vN

phrase2 = e' vN :+: f' vN :+: g' hN

phrase3 = g' aN :+: a' aN :+: g' aN :+: f' aN :+: e' vN :+: c' vN

phrase4 = c' vN :+: g vN :+: c' hN



MEHRSTIMMIGKEIT

Vollständige **Strophe**: Wiederholung der einzelnen Phrasen

strophe = wdh phrase1 :+: wdh phrase2 :+: wdh phrase3 :+: wdh phrase4

Viele **Strophen**:

strophen = ~~endlos strophe~~ wdh strophe

Kanon: gleiche Strophen, zeitlich versetzt:

einsatz d m = (Pause d) :+: m

4-stimmiger Kanon:

bruderJakob = strophen ::=

 einsatz (2/1) strophen ::=

 einsatz (4/1) strophen ::=

 einsatz (6/1) strophen



Musikkomposition ist exakte Struktur:

```
data Musik = Note GanzeZahl Bruch
           | Pause Bruch
           | Musik :+: Musik
           | Musik :=: Musik
```

- Rechnen wie mit anderen mathematischen Objekten möglich
- Definition der Berechnungen über den Aufbau der Struktur (analog zu Bruchrechnung, Rechnen mit komplexen Zahlen,...)

Beispiel: **Bruchrechnung**

$$\frac{a}{b} + \frac{c}{d} = \frac{a*d+b*c}{b*d}$$
$$\frac{a}{b} * \frac{c}{d} = \frac{a*c}{b*d}$$

Ausrechnen: wie bisher (Definition anwenden und weiter ausrechnen)

$$\frac{2}{3} + \frac{3}{4} = \frac{2 * 4 + 3 * 3}{3 * 4} = \frac{17}{12}$$



RECHNEN MIT MUSIK: TRANSPOSITION

Transponiere Musik: `transponiere i m`

erhöhe alle Noten in Musik `m` um Anzahl `i` von Halbtönen

`transponiere i (Note t d) = (Note (t+i) d)`

`transponiere i (Pause d) = (Pause d)`

`transponiere i (m1 :+: m2) = (transponiere i m1) :+: (transponiere i m2)`

`transponiere i (m1 :=: m2) = (transponiere i m1) :=: (transponiere i m2)`

Beispiel: **Kanon mit transponierten Stimmen**

`bruderJakobT = strophen :=:`

`einsatz (2/1) (transponiere 12 strophen) :=:`

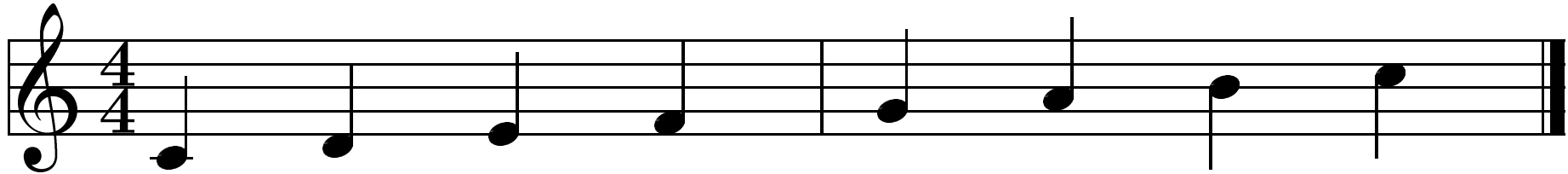
`einsatz (4/1) strophen :=:`

`einsatz (6/1) (transponiere 12 strophen)`



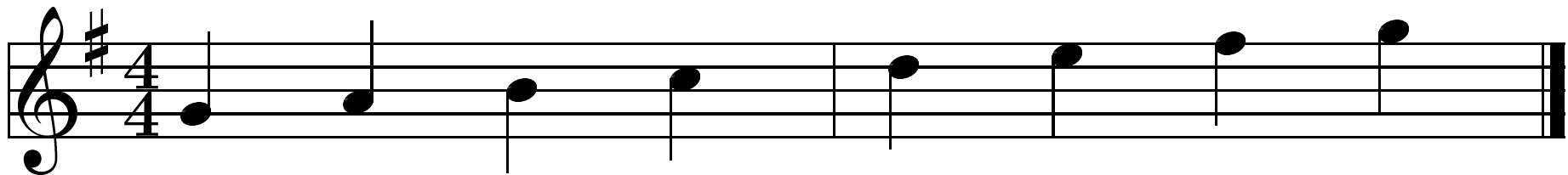
RECHNEN MIT MUSIK: TRANSPOSITION

Beispiel: C-Dur-Tonleiter



cDurNoten = c' vN :+: d' vN :+: e' vN :+: f' vN :+:
g' vN :+: a' vN :+: h' vN :+: c'' vN

G-Dur-Tonleiter durch Transposition um 5 Halbtöne:



gDurNoten = transponiere 5 cDurNoten



RECHNEN MIT MUSIK: TEMPOTRANSFORMATION

Transformiere Tempo: tempo t m

spiele alle Noten und Pausen in Musik m t-mal so schnell

z.B. m doppelt so schnell: tempo 2 m

tempo t (Note n d) = Note n (d/t)

tempo t (Pause d) = Pause (d/t)

tempo t (m1 :+: m2) = (tempo t m1) :+: (tempo t m2)

tempo t (m1 ::= m2) = (tempo t m1) ::= (tempo t m2)

Definition verschiedener **Tempi**: adagio = 7/6

andante = 9/6

allegro = 2

presto = 3

Beispiel:

cDurNoten ::= wdh (tempo allegro (transponiere 12 cDurNoten))



RECHNEN MIT MUSIK: RÜCKWÄRTS SPIELEN

rueckwaerts m : spiele Musik m rückwärts

rueckwaerts (Note n d) = Note n d

rueckwaerts (Pause d) = Pause d

rueckwaerts (m1 :+: m2) = (rueckwaerts m2) :+: (rueckwaerts m1)

rueckwaerts (m1 :=: m2) = ~~(rueckwaerts m1) xxxxxx (rueckwaerts m2)~~

let d1 = dauer m1

d2 = dauer m2 in

if d1>d2 then rueckwaerts m1 :=: (Pause (d1-d2) :+: rueckwaerts m2)

else (Pause (d2-d1) :+: rueckwaerts m1) :=: rueckwaerts m2

Hierzu definieren wir die **Dauer** von Musik:

dauer (Note t d) = d

dauer (Pause d) = d

dauer (m1 :+: m2) = (dauer m1) + (dauer m2)

dauer (m1 :=: m2) = max (dauer m1) (dauer m2)

Beispiel: cDurNoten :+: rueckwaerts cDurNoten



RECHNEN MIT MUSIK: ZEITBESCHRÄNKUNG

Beschränke Länge der Musik: schnitt laenge m

spiele Musik m bis zur Gesamtdauer laenge

```
schnitt laenge m | laenge <= 0 = Pause 0 -- Definition mit Bedingung
schnitt laenge (Note n d) = Note n (min d laenge)
schnitt laenge (Pause d) = Pause (min d laenge)
schnitt laenge (m1 :=: m2) = schnitt laenge m1 :=: schnitt laenge m2
schnitt laenge (m1 :+: m2) = let sm1 = schnitt laenge m1
                             sm2 = schnitt (laenge - dauer sm1) m2
                             in sm1 :+: sm2
```

Kanon mit Endlosstrophen:

stropfen = endlos strophe

Abspielen: (schnitt 16 bruderJakob)



Musik m n -mal hintereinander spielen:

```
mehrfach n m = if n>1 then m :+: mehrfach (n-1) m
                else m
```

Musik m n -mal spielen und jedesmal **um i Halbtöne transponieren**:

```
mfTrans i n m = if n>1 then m :+: mfTrans i (n-1) (transponiere i m)
                 else m
```

\rightsquigarrow mfTrans 1 12 (c' aN)

\rightsquigarrow mfTrans 1 12 (c' aN :=: e' aN)

\rightsquigarrow mfTrans 1 12 (durDreiklang ce aN)



HÖHERES RECHNEN MIT MUSIK

Musik m n -mal spielen und **Richtung jedesmal umdrehen**:

```
mfRueck n m = if n>1 then m :+: mfRueck (n-1) (rueckwaerts m)
              else m
```

\rightsquigarrow mfRueck 6 (c' aN :+: e' aN :+: g' aN :+: c'' aN)

Musik m n -mal spielen und jedesmal **Tempo um t ändern**:

```
mfTempo t n m = if n>1 then m :+: mfTempo t (n-1) (tempo t m)
                 else m
```

\rightsquigarrow mfTempo 2 6 (c' gN :+: e' gN :+: g' gN :+: c'' gN)



HÖHERES RECHNEN MIT MUSIK

Betrachte **Unterschiede** in der Definition obiger Funktionen:

```
mfTrans i n m = if n>1 then m :+: mfTrans i (n-1) (transponiere i m)
                else m
```

```
mfRueck   n m = if n>1 then m :+: mfRueck   (n-1) (rueckwaerts m)
                else m
```

```
mfTempo t n m = if n>1 then m :+: mfTempo t (n-1) (tempo t m)
                else m
```

~> ein generelles Schema **mit Musikfunktion f als Parameter**

```
mehrfach f n m = if n>1 then m :+: mehrfach f (n-1) (f m)
                else m
```

~> mehrfach **rueckwaerts** 6 (c' aN :+: e' aN :+: g' aN :+: c'' aN)

~> mehrfach **(tempo 2)** 6 (c' aN :+: e' aN :+: g' aN :+: c'' aN)



NOCH HÖHERES RECHNEN MIT MUSIK

Musik m n -stimmig spielen und jedesmal mit einer Funktion f verändern:

```
mehrstimmig f n m = if n>1 then m :=: (mehrstimmig f (n-1) (f m))
                    else m
```

↪ mehrstimmig (transponiere 12) 4 (c vN :+: e vN :+: g vN :+: c' vN)

↪ mehrstimmig (einsatz gN) 4 (c vN :+: e vN :+: g vN :+: c' vN)

Musik m n -stimmig spielen und jedesmal mit einer Funktion f verändern
und Ergebnismusik noch einmal mit einer weiteren Funktion g verändern:

```
mehrstimmig f g n m = if n>1 then m :=: g (mehrstimmig f g (n-1) (f m))
                    else m
```

```
halbtonlauf = mehrstimmig (transponiere 1) (einsatz sN) 24 (c' sN)
```

↪ wdh (halbtonlauf :=: rueckwaerts halbtonlauf)



NOCH HÖHERES RECHNEN MIT MUSIK

```
mehrstimmig f g n m = if n>1 then m :=: g (mehrstimmig f g (n-1) (f m))  
                      else m
```

→ kompakte Notation für komplexe Strukturen:

„Kaskadenmusik“:

```
lauf      = mehrstimmig (transponiere 5) (einsatz sN) 8 (c' sN)
```

```
kaskade   = mehrstimmig (transponiere 4) (einsatz vN) 8 lauf
```

```
kaskaden  = mehrstimmig id (einsatz aN) 2 kaskade
```

```
kaskadenMusik = kaskaden :+: rueckwaerts kaskaden
```



Programmierte Musik \approx formalisierte Musik

↪ Musik zugänglich für weitere Analysen

Automatische Erzeugung von Begleitakkorden

Eingabe: Melodie (Folge von Noten)

Ausgabe: Folge von Akkorden (z.B. für Gitarrenbegleitung)

Methode:

- ① rate Akkord und berechne Dissonanzen zwischen Akkord und Melodie in einem Takt
- ② falls Dissonanzen zu groß, unterteile Takt
- ③ wähle Akkord mit kleinster Dissonanz

Programmierparadigma: **Logikprogrammierung**



BEWEISEN VON MUSIKALISCHEN EIGENSCHAFTEN

Programmierte Musik \approx formalisierte Musik:
beweisbare Eigenschaften von Musik?

Gleichheit von Musikstücken:

transponiere 2 (tempo 2 (quinte ce (1/2))) $\stackrel{?}{=}$ quinte de (1/4)

beide Seiten ausrechnen: Note 2 (1/4) $:=:$ Note 9 (1/4)

Anwendung: **Programmoptimierung**

Problem: Gleichheit nach Ausrechnen \neq „hörbare Gleichheit“

Note ce (1/4) $:=:$ Note ge (1/4) \neq Note ge (1/4) $:=:$ Note ce (1/4)

Lösung:

- „hörbare Gleichheit“ \approx identisch, wenn *aufgeführt*
- definiere Gesetze für *Gleichheit bezüglich einer Aufführung*
- benutze diese, um Gleichheit von Musik nachzuweisen



ZUSAMMENFASSUNG

Beherrschung komplexer Systeme durch

- **Abstraktion** (ignoriere Unwichtiges)
- **Modellierung** (bilde geeignetes Modell)
- **Notation** (finde kompakte und verständliche Symbole)

Spezifisch für jeweiligen Anwendungsbereich

- Musik
- Architektur
- Ingenieurwissenschaften
- ...

Besonderheit der Informatik:

- **Notationen durch Computer ausführbar** \leadsto Programme
- definiere **Abstraktionen von Notationen** (durDreiklang, mehrstimmig,...)



FAZIT

Programme sind Notationen von Ideen eines Anwendungsbereichs

Programme müssen lesbar, verständlich und „schön“ sein

Gute Programme \approx gute Kompositionen

(aufgebaut aus einfachen Einheiten und geschickten Kombinationen)

Ähnlich wie in der Architektur, aber:

leider viele Programme, die an Plattenbauten erinnern

auch bedingt durch Taylorisierung der Programmierung (“lines of code”)

Programmiersprachen können helfen, gute Kompositionen zu entwickeln:

- Beispiele in diesem Vortrag: funktionale Programme
(Sprache: Haskell, Bibliothek: Haskore [Hudak, Yale])
- ähnliche Eleganz mit traditionellen Sprachen (Fortran, C, Java) kaum erreichbar

Entwicklung adäquater Programmiersprachen: aktuelle Forschung

