Computermusik
Vorlesung
WS 18, 20-23

Johannes Waldmann

14. Oktober 2024

Einleitung

Definition Computermusik

- *Computermusik* fassen wir auf als:

  Analyse und Synthese von Musik
  mithilfe der Informatik (Algorithmen, Software)

  (A: hören, verstehen; S: komponieren, aufführen)

- beruht auf Modellen aus der Musiktheorie, z.B. für
  - Erzeugung von Klängen in physikalischen Systemen,
  - das Tonmaterial:
    Tonhöhe, Konsonanz und Dissonanz, Akkorde, Skalen
  - die zeitliche Anordnung des Materials:
    Rhythmen, Melodien, Kadenzen, Kontrapunkt

Definition Musik

- die Kunst der zeitlichen Anordnung von Klängen.

  (Edgar Varese 1883–1965: *I call it organised sound*)
- „Kunst“ bedeutet: der Autor (Komponist, Interpret) will im Hörer Empfindungen hervorrufen
- das geht sowohl sehr direkt, Beispiele:
  - Tonreihe aufsteigend: Frohsinn, absteigend: Trübsal
  - Dissonanz $\Rightarrow$ Spannung, Unruhe;
    Konsonanz $\Rightarrow$ Auflösung, Ruhe

  als auch indirekt, Beispiele:
  - Zitat (Parodie) von Elementen andere Musikwerke:
    Anerkennung (Daft Punk $\Rightarrow$ Giorgio Moroder),
    Aneignung (F.S.K.), Ablehnung (Punk $\Rightarrow$ Prog Rock).

Definition Pop(uläre) Musik

- die mechanische (Aufnahme und) Vervielfältigung von Audiosignalen (seit ca. 1920, Grammophon)
  trennt die *Aufführung* vom ihrem *Resultat* (dem Klang)

  (Elijah Wald, An Alternative History of American Popular Music, Oxford Univ. Press, 2009)

- dadurch entsteht Popmusik, das ist etwas Neuartiges
  - statt Komposition (Klassik) oder Improvisation (Jazz): *Produktion* des Klangs in einem Studio
  - rezipiert wird nicht nur der Klang, sondern unzählige *Nebenprodukte*, insb. Bilder (z.B. Schallplattenhüllen)
    die Bedeutung wird daraus vom *Fan* konstruiert

  (Diederich Diederichsen, Über Popmusik, Kiepenheuer, 2014)

Hörbeispiele

- Daft Punk (Guy-Manuel de Homem-Christo und Thomas Bangalter): *Giorgio by Moroder* (LP Random Access Memories, 2013)
- Donna Summer: *I Feel Love* (Single, 1977) Produzent: G. Moroder
- Kraftwerk (Ralf Hütter und Florian Schneider): *Autobahn* (LP 1974), aufgenommen im Studio Conny Plank
- Neu! (Michael Rother und Klaus Dinger): *Hallogallo* (1972), Produzent: Conny Plank
- Stereolab (Tim Gaine, Laetitia Sadier u.a.): *Jenny Ondioline* (1993)
- Grandmaster Flash (Joseph Sadler) *The Message*(1982)
- Big Black (Steve Albini u.a.): *Kerosene* (1986)

Plan unserer Vorlesung (I)

- KW 43: Klang-Erzeugung (Physik der Musikinstrumente)
- KW 44: Klang-Analyse (Spektren) und Klangveränderung
- KW 45: Analog-Synthesizer (und ihre Simulation)
- KW 46: Algebraische Beschreibung von Klängen (csound-expression)
- KW 47: Töne, Skalen, Konsonanzen, Akkorde, Kadenzen
- KW 48: Algebra Of Music (haskore), Notensatz (lilypond)
- KW 49: A Pattern Language (tidal-cycles, supercollider)

Plan unserer Vorlesung (II)

ab hier Themen und Reihenfolge noch offen

- Rhythmus (breakbeat science)
- Mathematische Musiktheorie
- Musikgeschichte
- Zwischenstand Projekte
- Digital Audio Workstations (ardour, sooperlooper)
- algorithmische Mechanik
- Zusammenfassung, Abschluß Projekte

Organisatorisches

Die LV insgesamt

- jede Woche eine Vorlesung, eine Übung
- Prüfungszulassung: regelmäßiges und erfolgreiches Bearbeiten von Übungsaufgaben (teilw. autotool)
- Prüfung:
  - (gemeinsames) Abschluß-Konzert
  - (individuelle) Dokumentation (*welche* kreative Idee wurde *wie* realisiert)

Übungen

- Sie benutzen die Rechner im Pool (Z423/430) mit dort installierter Software. — *Kopfhörer mitbringen!*

  Es ist zu empfehlen, die gleiche Software auch auf Ihren privaten Rechnern zu installieren, damit Sie selbst experimentieren und Hausaufgaben erledigen können.

- wir verwenden ausschließlich *freie* Software (Definition: siehe `https://www.gnu.org/philosophy/free-software-intro.html`) (Debian-Pakete oder selbst kompiliert). Alles andere wäre unwissenschaftlich — weil man es eben nicht analysieren und ändern kann.

GNU/Linux-Audio

- . . . kompliziert, weil
  - alles sehr modular funktionieren soll,
  - korrekt (bei Musik: mit geringer Latenz)
  - für einen großen Bereich von Hardware (neu bis alt, teuer bis billig)
- Hardware (Soundkarte, intern/extern), Treiber
- ALSA `https://alsa-project.org/` Advanced Linux Sound Architecture
- Jack `https://wiki.archlinux.org/title/JACK_Audio_Connection_Kit`
- Pipewire, Pulseaudio (kämpfen mit Jack um Zugriff auf Hardware bzw. Alsa-Treiber)

Übungs/Haus-Aufgaben

Das sind Beispiele für Tätigkeiten, die in dieser LV (und in allen anderen) immer wieder vorkommen: nicht nur Software bedienen und Knöpfchen drehen, sondern auch: Analysieren, Rechnen, Recherchieren, historisch einordnen, Programmieren (Synthetisieren).

1. (bereits in Ü) ausprobieren: Hydrogen (Drum-Sequencer) → Rakarrack (Effekt-Prozessor)

   Audio-Routing mit qjackctl

2. Finden Sie die von Hydrogen benutzte Audio-Datei für *TR808 Emulation Kit, Kick Long*

   anhören mit `vlc`,

   konvertieren Sie mit `sox` in `wav`-Format, (Hinweis:

`man sox`),

betrachten Sie Dateiinhalt (Amplituden-Verlauf) mit

```
gnuplot -persist -e "plot 'kick.wav' binary
```

Bestimmen Sie mittels dieses Bildes die Grundfrequenz der Schwingung. Welche weitere Information ist dazu nötig, woher bekommen Sie diese?

3. betrachten Sie Dateiinhalt mit

```
od -cx kick.wav | less
```

Wo endet der Header (wo steht das erste Datenbyte)?

Suchen Sie die offizielle WAV-Spezifikation, bestimmen Sie deren bibliografische Daten (Autor/Gremium, Ort, Jahr)

Erzeugen Sie durch ein selbstgeschriebenes Programm (Sprache beliebig) eine wav-Datei, die einen (kurzen) Sinus-, Dreieck-, oder Rechteckton enthält,

ansehen mit `gnuplot`, abspielen mit `vlc`,

verwenden Sie das als Sample in Hydrogen.

4. Wie sah diese Maschine (TR808) aus?

   Welche Band führt diese Maschine im Namen? (Hinweis: `http://www.vintagesynth.com/`)

   Kann Hydrogen alle dort angegebenen Eigenschaften des Originals simulieren?

   beschreiben Sie Struktur und (einige) Elemente von *Ritchie Hawtin: Minus Orange 1*, *Aphex Twin: Flaphead* o.ä., simulieren Sie mit Hydrogen und Rakarrack.

Geräusch und Klang

Begriffe

- Geräusch:
  - erzeugt durch Schwingungen eines physikalischen Systems (z.B. Musikinstrument)
  - übertragen durch Druckschwankungen in einem Medium (z.B. Luft), durch Ohr wahrnehmbar
- Klang: . . . durch *periodische* Schwingungen . . .
- virtuelle (elektronische) Instrumente
  - simulieren den physikalischen Vorgang
  - oder speichern nur dessen Verlauf
- Unterschied zu automatischem Spiel reeller Instrumente

Modell einer periodischen Schwingung

- Modell:
  - ein Körper mit Masse m und Ruhelage 0
    bewegt sich auf einer Geraden g,
    d.h., hat zum Zeitpunkt t die Koordinate $y(t)$
  - die Rückstellkraft (bei Pendel: durch Schwerkraft, bei schwingender Saite: durch Elastizität) ist $F = -k \cdot y$.
    Notation: das ist eine Gl. zw. Funktionen (der Zeit)!
- mathematische Beschreibung
  - Geschwindigkeit $v = y'$, Beschleunigung $a = v' = y''$
  - nach Ansatz ist $a = F/m = -(k/m) \cdot y$
  - y ist Lsg. der Differentialgleichung $-(k/m)y = y''$

Numerische Näherungslösung der Dgl.

- gegeben k, m, bestimme Funktion y mit $-(k/m)y = y''$
- numerische Näherungslösung durch Simulation:

  ersetze Differentialgleichung durch Differenzengleichung

  wähle y_0 (initiale Auslenkung), $\Delta > 0$ (Zeitschritt),

  bestimme Folgen $y_0, y_1, \ldots, v_0 = 0, v_1, \ldots, a_0, a_1, \ldots$

  mit $a_i = -(k/m)y_i$, $v_{i+1} = v_i + \Delta a_i$, $y_{i+1} = y_i + \Delta v_i$
- genaueres in VL Numerik,

  z.B.: *Stabilität* besser, wenn $y_{i+1} = y_i + \Delta v_{i+1}$

Implementierung der numerischen Sim.

- Zustandstyp: $\mathbb{R} \times \mathbb{R}$ (Ort $\times$ Geschwindigkeit), Zustandsfolge mit `iterate :: (a -> a) -> a -> [a]`

```
let { d = 0.1 } in iterate
  (\ (y,v) ->
     let { a = negate y
         ; vn = v + d * a ; yn = y + d * vn
     } in (yn,vn))
  (1,0)
```

- anzeigen:

  `https://hackage.haskell.org/package/gnuplot` (Henning Thielemann), WAV ausgeben:

  `https://hackage.haskell.org/package/WAVE` (Bart Massey)

Exakte Lösung der Dgl.

- gegeben k, m, bestimme Funktion y mit $-(k/m)y = y''$
- genaueres siehe VL Analysis, z.B. Ansatz von y als
  - Potenzreihe $y = \sum_{k \in \mathbb{N}} c_k x^k$, Koeffizientenvergleich von linker und rechter Seite der Dgl.
  - Linearkombination von anderen Basisfunktionen (anstatt Potenzen)
- wenn man Glück hat, oder die numerische Lösung gesehen hat:

  Ansatz $y(t) = \cos(f \cdot t)$
- wir erhalten die *reine harmonische Schwingung*

Schwingung einer Saite

- . . . aus vielen Massepunkten, u : Ort $\times$ Zeit $\rightarrow$ Auslenkung
- Elastizität des Materials wirkt in jedem Punkt

  als Kraft in Richtung beider Nachbarn
- Differenzengl., diskret: $y_k'' = (y_{k-1} - y_k) + (y_{k+1} - y_k)$

  math. Modell, kontinuierlich: $d^2u/(dt)^2 = c \cdot d^2u/(dx)^2$,

  Randbedingungen $u(0,t) = u(1,t) = 0$, $u(x,0) = 0$
- Ansatz $u(x,t) = f(x) \cdot g(t)$, es gibt mehrere Lösungen
- Dgl. ist linear: jede Summe von Lösungen ist Lösung
- Hermann Helmholtz: Vorl. über die mathematischen Prinzpien der Akustik, Leipzig 1898 `https://archive.org/details/vorlesungenber03helmuoft`

Anpassung und Anwendung

- diese Modell ist Energie-erhaltend

  tatsächlich wird aber Energie abgegeben (1.über das Medium an den Sensor, 2. durch Reibung im schwingenden Körper als Wärme an die Umgebung)

- Modellierung der *Dämpfung* z.B. durch Reibungskraft proportional zu Geschwindigkeit $F_R = r \cdot v = r \cdot y'$

  Aufstellen und Simulation der Dgl. in Übung.

- mit diesem Modell können wir beschreiben:
  - Klang einer Saite (Gitarre, Klavier, Cembalo) (nicht Geige)
  - Klang eines Trommelfells (Fußtrommel, nicht Snare)

Beispiel: Mbira (Daumenklavier)

- Zungen aus Holz oder Metall auf Resonanzkörper
- Hörbeispiele: Stella Chiweshi: *Chigamba*,

  Konono No. 1: *Konono Wa Wa*

Beispiel: schwingende Metallstäbe

- Spielzeug-„klavier“
- Fender-Rhodes-Piano (1965–1984) `https://www.fenderrhodes.com/org/manual/toc.html`

  Hörbeispiele: Miles Davis: *Spanish Key* 1969,

  Steely Dan: *Babylon Sisters* 1980
- Vibraphon (Metallstäbe, Resonanzröhren mit beweglicher Abdeckung)

  Hörbeispiele: Tortoise: *Ry Cooder* 1994,

  Claudia Quintet: *September 20 Soterious Lakshmi* 2013

Weitere period. Schwingungen f. Instrumente

- Wirkung der Dämpfung kann durch regelmäßige Energiezufuhr ausgeschaltet werden ($\Rightarrow$ angeregte Schwingung) z.B. das Anstoßen einer Schaukel
- Geige:

  Bewegung des Bogens führt der Saite Energie zu

  regelmäßige Unterbrechung durch Kontaktverlust Bogen–Saite bei zu starker Auslenkung
- Blasinstrumente:

  Anblasen führt der schwingenden Luftmenge Energie zu

  regelmäßige Unterbrechung durch Blatt (Oboe, Saxofon), Lippen (Trompete) oder Luftsäule selbst (Orgel, Flöte)

Beispiele

- Querflöte

  Bobbi Humphrey *Harlem River Drive* 1973
- Saxophon

  John Coltrane, *A Love Supreme*, 1965
- Posaune, Mundharmonika (?)

  Lee Perry *Heavy Rainford* 2019 (Prod. Adrian Sherwood)
- Melodica (angeblasene Metallzungen, vgl. Triola)

  Augustus Pablo *King Tubbys Meets The Rockers Uptown* 1974,

  vgl. David Katz: *A beginner's guide to Augustus Pablo* Fact Magazine, 2015

  `https://www.rockersinternational.com/`

Geräusch-Instrumente

- nichtperiodisches Verhalten kann erzeugt werden durch
  - Überlagerung (fast gleichzeitiger Ablauf) sehr vieler unterschiedlicher periodischer Schwingungen
    für zahlreiche (Rhythmus)-Instrumente benutzt, z.B.
    - Maracas (Rumba-Kugel): enthalten viele kleine harte Klangkörper, die aneinanderstoßen
    - Snare (kleine Trommel): mehrere Federn, die gegen Fell der Unterseite schlagen (schnarren)
- nichtperiodische Schwingung eines phys. Systems

  z.B. Doppel-Pendel, Mehr-Körper-System

  keine direkte Anwendung als Instrument bekannt,

  Simulation evtl. für virtuelle Instrumente nützlich

Chaotische Schwingungen

- wenn man das wirklich nur simulieren möchte (nicht physikalisch realisieren)
- dann kann man auch mathematische Modelle *ohne* physikalisches Äquivalent betrachten
- Bsp: Iteration von $f : [0, 1] \to [0, 1] : x \mapsto 4 \cdot (x - 1/2)^2$ zeigt aperiodisches (chaotisches) Verhalten
- Bsp: bitweise Manipulation (der Zeit)

```
t * ((t>>12|t>>8)&63&t>>4)
```

- ergibt (im Allgemeinen) nur ein Rauschen, Grundlage für Simulation andere Klänge (mit Filtern)
- aber im Speziellen: interessante Klänge möglich

`https://wurstcaptures.untergrund.net/music/`

Hausaufgaben

1. Wie wird Musikgeschichte zitiert (im Klang und) im Text von: DJ Hell: *Electronic Germany* (2009)

   Wer singt auf *U Can Dance* des gleichen Albums? War früher (viel früher) in welcher Band? Wer hat dort anfangs elektronische Instrumente gespielt? Danach welchen Musikstil erfunden?

   weitere Beispiele für Musikzitate suchen, genau beschreiben, was zitiert wird, wie groß der Abstand ist (zeitlich, inhaltlich) und diskutieren, warum.

2. harmonischen bzw. gekoppelten Oszillator modifizieren: Schwingungen simulieren, Resultate ansehen,

   - periodische

- gedämpfte (durch Zusatz-Term in harmonischem)
- chaotische (durch Nichtlinearität in der Kopplung)

anhören

- einzeln
- als Drumkit in Hydrogen

3. die Simulation der Saite verändern:

   das Beispiel aus Helmholtz § 39 Fig. 7 realisieren (Zupfen der Saite nicht in der Mitte), Resultat mit Fig. 11 vergleichen

   § 42 realisieren (belastete Saite: ein Punkt hat andere Masse)

4. kleine Bit-Musikstücke (Beispiel: `t << (t>>10)`) vollständig analysieren, dann modifizieren.

Klang-Analyse (Grundlagen)

Definition, Motivation

- jede periodische Schwingung kann als gewichtete Summe harmonischer Schwingungen dargestellt werden (Jean Fourier, 180?, `http://www-history.mcs.st-and.ac.uk/Biographies/Fourier.html`)
- die Folge dieser Gewichte der Obertöne ist das *Spektrum*, das charaktetisiert die Klang*farbe*
- Änderung des *Amplituden*verlaufs

  linear (z.B. Filter), nichtlinear (z.B. Verzerrer)

  kann beschrieben werden als Änderung des *Spektrums*, diese ist ggf. leichter zu berechnen

Periodische Funktionen

- für $\Omega = [-\pi, \pi]$ betrachte P $= \{f \mid f : \Omega \to \mathbb{R}\}$.
- P ist *Vektorraum* (Addition, Skalierung) und Hilbert-Raum
- *Skalarprodukt* $\langle f, g \rangle := \int_\Omega f(x) \cdot g(x)\, dx$, Norm $|f| = \sqrt{\langle f, f \rangle}$
- $b_1 = 1, b_2 = \sin(x), b_3 = \cos(x), b_4 = \sin(2x), b_5 = \cos(2x), \ldots$ bilden eine *orthogonale Basis* für P
  nach geeigneter Skalierung sogar ortho*normal*
- jedes $f \in$ P eindeutig darstellbar als Linearkombination von Basisvektoren $f = \sum_i \langle f, b_i \rangle / |b_i|^2 \cdot b_i$
- weitere Voraussetzungen sind nötig (damit Integrale und Summen existieren), siehe VL Analysis
- numerisch: approximiere Integral durch Summe

Beispiel: Rechteck-Schwingung

- $\Omega \to \mathbb{R} : t \mapsto$ if $t < 0$ then -1 else if $t = 0$ then 0 else 1

  das ist die Signum- (Vorzeichen)-Funktion sign

- $\mathrm{sign}(x) = (4/\pi) \sum_{k \text{ ungerade}} \frac{\sin(kx)}{k}$

  (nur ungerade Oberwellen)

Nebenrechnungen:

- $\cos(kx)$ ist gerade Funktion, $\mathrm{sign}(x)$ ungerade,

  deswegen $\langle \mathrm{sign}(x), \cos(kx) \rangle = 0$

- $\langle \mathrm{sign}(x), \sin(kx) \rangle = 2 \cdot \int_{[0,\pi]} \sin(kx) dx = [-1/k \cdot \cos(kx)]_0^\pi =$

  if $2|k$ then 0 else $4/k$

Beispiel: Sägezahn-Schwingung

- $f : \Omega \to \mathbb{R} : x \mapsto x$
- numerische Bestimmung der Fourier-Koeffizienten

```
let k = 4 ; d = 0.01
in sum $ map (\x -> d * x * sin (k*x))
          [ negate pi, negate pi + d .. pi ]
==> -1.5707233265855 21
```

- Vermutung $f = -\frac{2}{\pi} \sum_{k \geqslant 1} \frac{(-1)^k}{k} \sin(kx)$ (alle Oberwellen)

Spektren von Audiosignalen

- Spektrum eines Signals f kann so bestimmt werden:
- teile Signal in Zeit-Intervalle (z.B. $\Delta = 1/10$ s),

  $f_i : [-\Delta, \Delta] \to \mathbb{R} : t \mapsto f(i\Delta + t)$
- wähle Frequenz-Werte $k_1, k_2, \ldots$
- bestimme Koeffizienten der Freq k_j zur Zeit $i\Delta$ als $\langle f_i, k_j \rangle$
- Anzeige z.B. in `vlc`: Audio → Visualisations → Spectrum
- es gibt schnellere Algorithmen

  (diskrete Fourier-Transformation)
- das Ohr bestimmt die Fourier-Koeffizienten durch Resonanz in der Schnecke (Cochlea), Frequenz-Auflösung ist ca. 3 Hz bei 1 kHz

Programme zur Spektral-Analyse

- Chris Cannam, Christian Landone, and Mark Sandler: *Sonic Visualiser: An Open Source Application for Viewing, Analysing, and Annotating Music Audio Files,* in Proceedings of the ACM Multimedia 2010 International Conference.

  `https://sonicvisualiser.org/`

- Anwendungsbeispiel:

  Aphex Twin, $\Delta M_i^{-1} = -\alpha \Sigma D_i[\eta] F j_i[\eta - 1] + F\mathsf{ext}_i[\eta^- 1]$,

  Album: Windowlicker, 1999.

  hergestellt mit Metasynth (Eric Wenger, Edward Spiegel, 1999) `http://www.uisoftware.com/MetaSynth/`,

Zeit-Dehnung

- wenn man ein Audio-Signal $f :\rightarrow \mathbb{R}$ zeitlich dehnt (Bsp: $g(x) = f(s \cdot x)$ mit $s = 1/2$), dann ändert man damit die Frequenzen.
- Zeit-Dehnung *ohne* Frequenz-Änderung:
  $\text{Signal}_1 \overset{\text{Spektral-Analyse}}{\longrightarrow} \text{Spektrogramm}_1 \overset{\text{Zeit-Dehnung}}{\longrightarrow} \text{Spektrogramm}_2 \overset{\text{Spektral-Synthese}}{\longrightarrow} \text{Signal}_2$
- Audio-Kompressoren MP3, AAC haben bereits solche Signalkette, mit *Kompression* (Bitbreiten-Reduktion) statt *Zeit-Dehnung*, diese kann leicht hinzugefügt werden
- Bsp: 7038634357 (Neo Gibson): *Barry White Stretched Out And Reworked* 2022 `https://www.nts.live/shows/guests/episodes/7038634357-19th-october-2022`

Spektren von Klängen/Instrumenten

- harmonische Schwingung: keine Oberwellen,

  kommt in der Natur selten vor und ist für Musikinstrumente auch gar nicht erwünscht:

  Oberwellen ergeben interessantere Klänge,

  die auch variiert werden können

- Bsp: Gitarre: Anschlagen nahe dem Steg: viele Oberwellen, zur Saitenmitte: weniger.
- Bsp. Schlagzeug (Trommel, Tom): Anschlag Mitte/Rand
- Bsp: Orgel: offene und gedackte Pfeifen, siehe dazu Kalähne 1913 (Hausaufgabe)

Aufgaben

1. In *Autobahn* (Kraftwerk) fährt bei ca. 1:49 ein Auto am Hörer vorbei. Wie schnell?

   (Hinweis: Frequenzen mit sonic-visualier bestimmen, Doppler-Effekt verwenden)

2. wie unterscheiden sich Spektren der Luftschwingungen in offenen von einseitig geschlossenen Röhren? nach: Alfred Kalähne: *Grundzuge der mathematisch-physikalischen Akustik*, Leipzig 1913,

   `https://archive.org/details/grundzgedermath01kalgoog`

3. Fourier-Koeffizienten einer Rechteck-, Sägezahn-, Dreiecks-Schwingung bestimmen:

- Skalarprodukte symbolisch oder numerisch bestimmen
- Amplitudenverlauf in WAVE-Datei schreiben und Spektrum analysieren (`sonic-visualiser`)

4. Bestimmen Sie für das Signal *Rechteck* + *2 mal Sägezahn*

   - den Amplitudenverlauf
   - die Fourier-Koeffzienten (unter Verwendung der im Skript angegebenen Koeffz. der einzelnen Signale)

Elektrische Oszillatoren und Filter

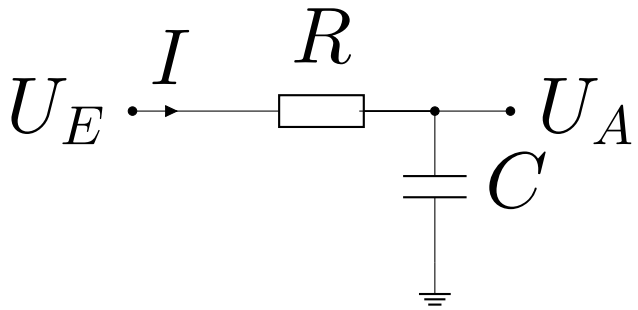
Plan

- bisher: mechanische Schwingungen
  - Bsp: Massepunkt/Feder,
  - Anwendung: akustische Musikinstrumente
    Bsp: Saiten, Membrane, Luftsäulen
- jetzt: elektrische Schwingungen (und Filter)
  - Bsp: Oszillator (LC), Tiefpaß (RC)
  - Anwendungen: diese VL: Filter, nachfolgende:
    - Analog-Synthesizer (Robert Moog 64, Don Buchla 63)
    - Simulation von A.-S. (csound, Barry Vercoe, 1985)
  - Ziele: 1. möglichst exakte Nachbildung (des Akustischen, des Analogen), 2. völlig neuartige Klänge

Elektrische Schaltungen

- Schaltung: gerichteter Graph,
  - Kanten sind Bauelemente
    - ohne Zustand: Widerstände, Verstärker (Transistor)
    - mit Zustand: Kondensator: elektrisches Feld,
      Spule: magnetisches Feld
  - durch jede Kante fließt Strom,
    jeder Knoten hat Potential
  - besondere Knoten: Masse (0), Eingabe, Ausgabe
- Zustandsänderung nach Gesetzen der Physik (Elektrik)
- vergleiche: Massepunkte, Trägheits-, Federkräfte
- Schaltung realisiert Operator F von Eingabesignal
  $g : \Omega \to \mathbb{R}$ zu Ausgabesignal $F(g) : \Omega \to \mathbb{R}$

Schaltung – Beispiel Tiefpaß



- Schaltung:
- Widerstand: $U_E - U_A = R \cdot I$

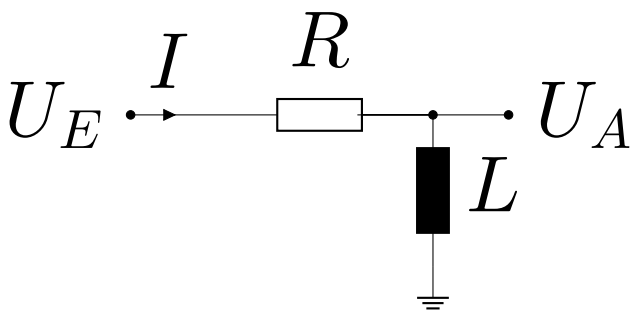
  (siehe auch Kraftwerk: *Ohm Sweet Ohm*, 1975)
- Kondensator: $I = C \cdot \dfrac{dU_A}{dt} = C \cdot U'_A$
- Bsp: $U_E(t) = 1\mathsf{V}$, $U_A(0) = 0$ (Kondensator leer)

  $C \cdot U'_A = I = (1 - U_A)/R$, Simulation, exakte Lösung
- Bsp: $U_E(t) = \sin(2\pi f t)$, $U_A(t) = ?$
- wirkt als Tiefpaß-Filter: Schwächung hoher Frequenzen

Bemerkung zur Methodik

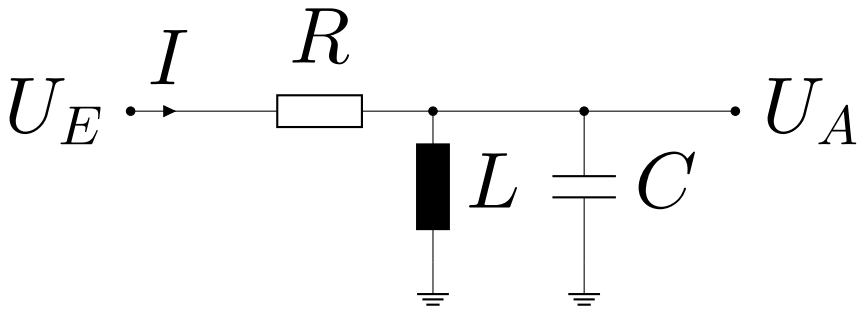
- (analoge) Schaltungstechnik, seit $\geqslant$ 100 Jahren alles wohlbekannt,
- Umformung zur praktischen Berechnung: 1. Analyse harmonischer Schwingungen, 2. Linearkombination.
- Harmonische Schwingungen fester Frequenz sind bestimmt durch Betrag r und Phase ϕ, dargestellt als *eine* komplexe Zahl $z = r\exp(i\phi) \in \mathbb{C}$
- verwende Kirchhoffsche Regeln f. komplexe Größen
  Bsp: Kondensator mit Kapazität C hat bei Kreisfrequenz ω den komplexen Widerstand (Impedanz) $1/(i\omega C)$,
  Spule mit Induktivität L hat Impedanz $i\omega L$.
- funktioniert, solange alle Bauelemente linear sind (Widerstand hängt nur von Frequenz ab)

Weitere Filter: Hochpaß, Bandpaß

- 

, Spule: $U_A = L \cdot \frac{dI}{dt} = L \cdot I'$

  wirkt als *Hochpaß* (tiefe Frequenzen werden geschwächt)

- 

, wirkt als *Bandpaß*

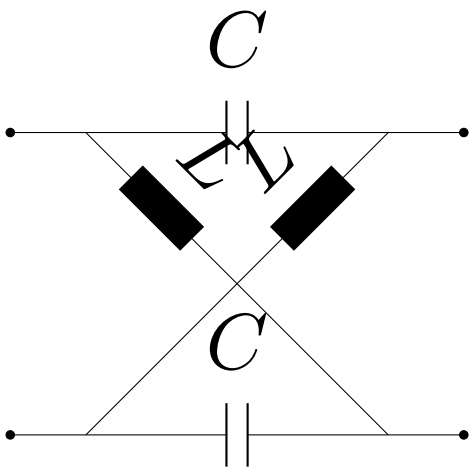
  (hohe und tiefe f geschwächt, in der Nähe der Resonanzfrequenz weniger)

- Bandpaß mit Rückführung und Verstärkung:

  wirkt als *Oszillator* (schwingt auf Resonanzfrequenz)

Weitere Filter: Allpaß

- lattice filter (d: Gitter- oder Leiter-Filter)



- vgl. Julius O. Smith, Physical Audio Signal Processing, W3K Publishing, `https://ccrma.stanford.edu/~jos/pasp/Allpass_Filters.html`
- angewendet im *Phaser*

Klangveränderung durch Filter

- ein Filter ist ein Operator von $(\Omega \to \mathbb{R})$ nach $(\Omega \to \mathbb{R})$

  (eine Funktion der Zeit auf eine Funktion der Zeit,

  d.h., Filter ist Funktion zweiter Ordnung)
- Bsp: der Operator scale$_s$: $g \mapsto (x \mapsto s \cdot g(x))$
- Bsp: der Operator shift$_t$: $g \mapsto (x \mapsto g(x - t))$

  akustisch ist das ein *Echo*. Mehrere Echos ergeben *Hall*.

  Realisierungen:
  - Tonband-Schleife
  - Federhallstrecke

  typisch für: Gitarrenklang in Surf-Musik (Bsp: Dick Dale), Gesamtklang im (Dub) Reggae (Bsp: Lee Perry)

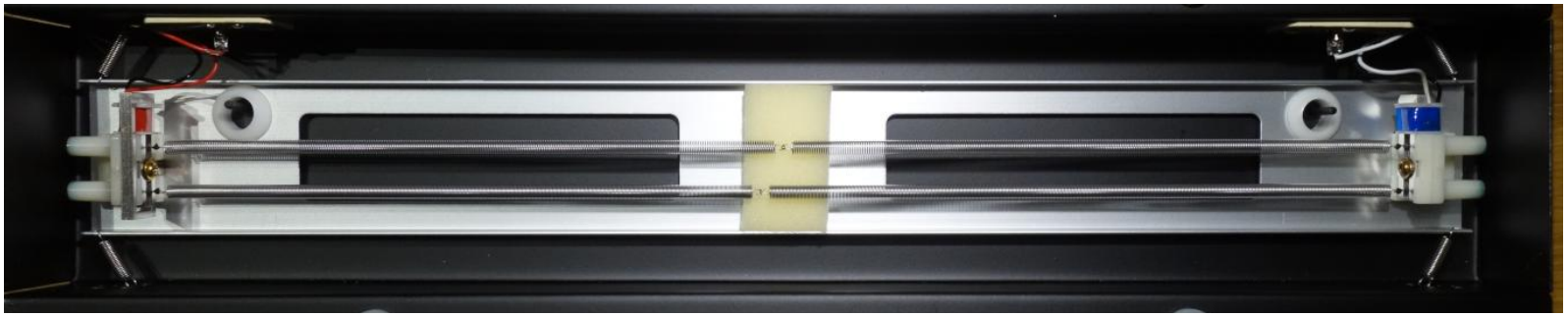
Klangveränderung durch Filter

- Operator F ist *linear* (L), wenn
  $\forall a, b \in \mathbb{R}, g, h \in (\Omega \to \mathbb{R}) : F(a \cdot g + b \cdot h) = a \cdot F(g) + b \cdot F(h)$
- F ist *zeit-invariant* (TI), wenn $\forall t \in \mathbb{R}$: $\text{shift}_t \circ F = F \circ \text{shift}_t$
- Satz: jeder LTI-Filter kann als (Limes einer unendl.) Summe von shift und scale dargestellt werden
- Satz: jeder lineare Filter operiert auch linear auf den Fourier-Koeffizienten.
- Folgerung: Obertöne werden geschwächt oder verstärkt, aber niemals „aus dem Nichts“ erzeugt.

  Das begründet den Wunsch nach nichtlinearen Filtern (Verzerrern).

Filter in der Musik-Praxis (Fender Amp)

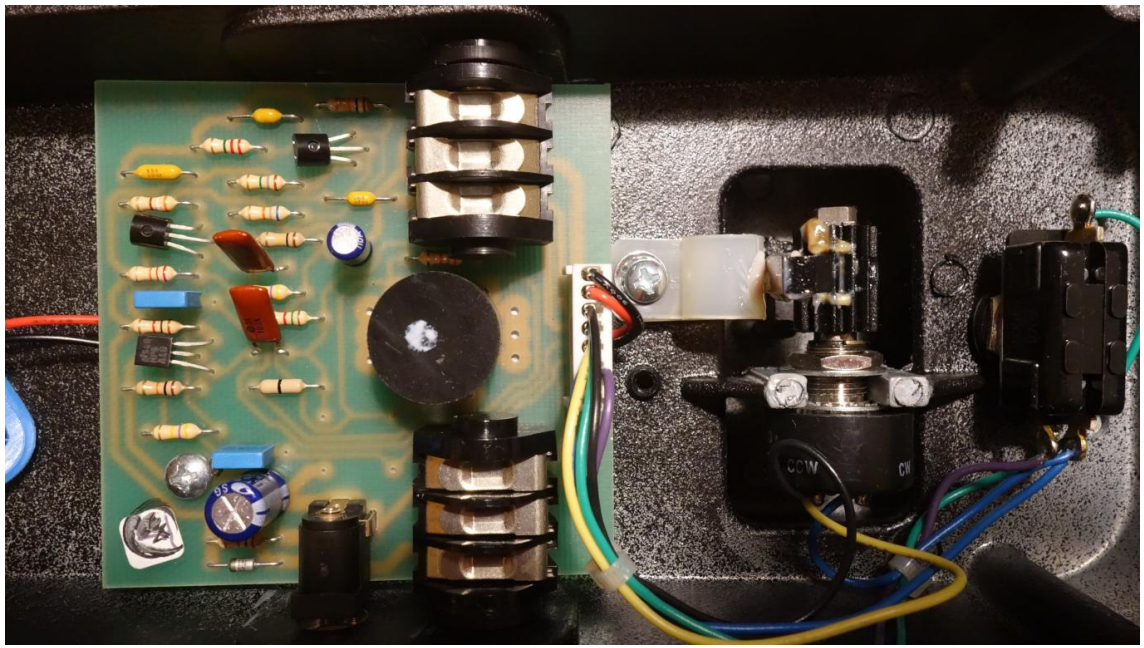
- Lautstärke (Volume): scale_s
- period. Lautstärkeänderung (Tremolo) (Speed, Intensity) ist linear, aber nicht zeit-invariant
- Federhall (Reverb): $\sum_{d \in D} \text{shift}_d$, linear, zeit-invariant



- Tiefpaß (Bass), Hochpaß (Treble)

Filter in der Musik-Praxis (Wah)

- Dunlop Crybaby GCB95

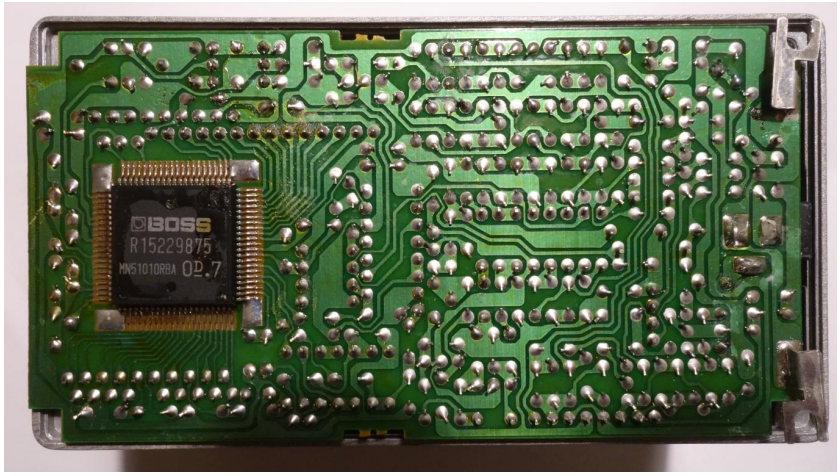


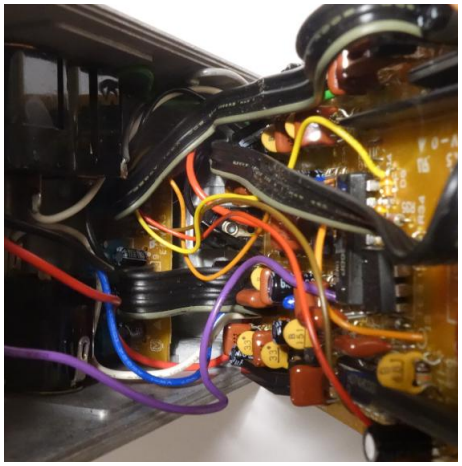
`https://www.electrosmash.com/crybaby-gcb-95`

- Bandpaß mit einstellbarer Resonanz-Frequenz
- Fußwippe (Pedal) → Zahnstange → Dreh-Potentiometer
- vgl. später: spannungsgesteuerte Filter (VCF)

Filter in der Musik-Praxis (Echo)

- Boss Digital Delay (DD 3, ab 1986)



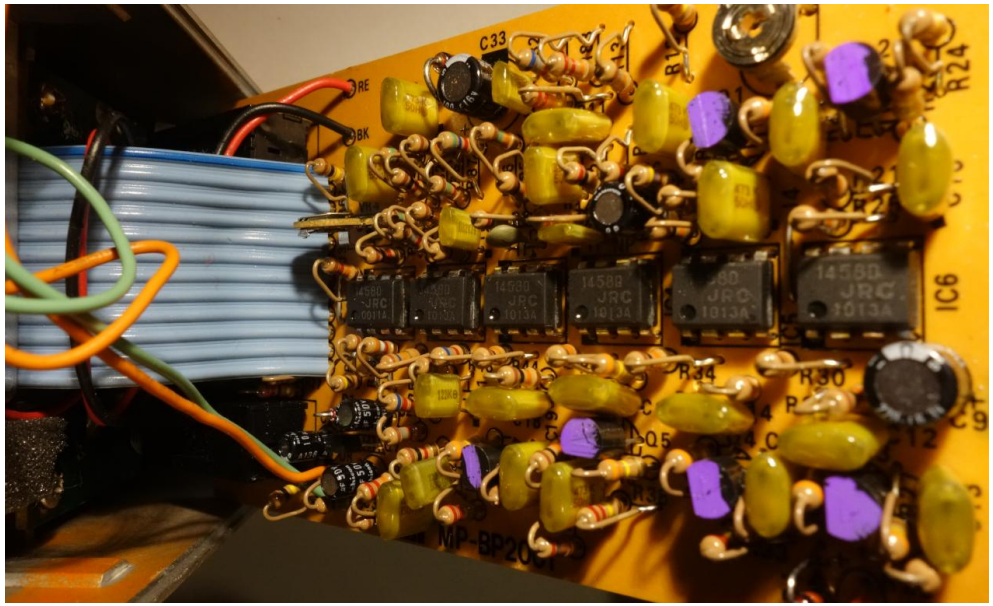


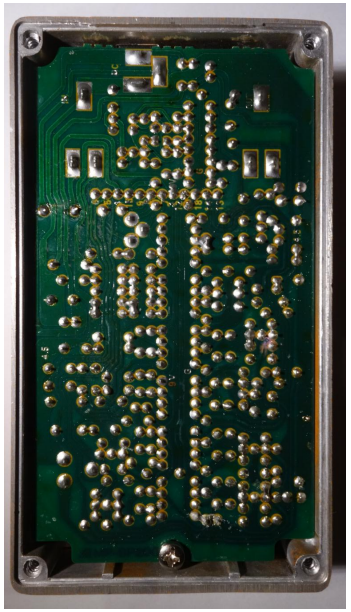
- erstes Delay-Fußpedal DD 2, 1983 `https://www.hobby-hour.com/electronics/s/dd2-delay.php`
- Echo-Zeit max. 800 ms. Samplebreite 12 bit. Ü: Taktbreite 12.5 μs . . . 50 μs. Wieviel Bit werden gespeichert?
- vgl. später: Chorus (= spannungsgest. Delay), Flanger

Chorus, Flanger

- Flanger: $f + \text{shift}_d(f)$,
  ursprünglich realisiert durch zwei Tonbandmaschinen
  für f, für $\text{shift}_d(f)$, dabei d durch Bremsen des Bandes
- Chorus: $f \mapsto f + \sum_k \text{shift}_{d_k}(f)$ mit $d_k = \epsilon \cdot \sin(\omega_k t)$, kleine ω_k
  (mehrere leicht unterschiedliche Stimmen in einem Chor)
- bei elektronischer Realisierung:
  auch mit Rückführung des Ausgangssignales
- Ibanez Swell Flanger `https://mirosol.kapsi.fi/2015/05/ibanez-sf10-swell-flanger/` mit analoger Speicherkette MN3207 `https://zeptobars.com/en/read/MN3207-1024-stage-analog-delay-line-CCD`

Phaser

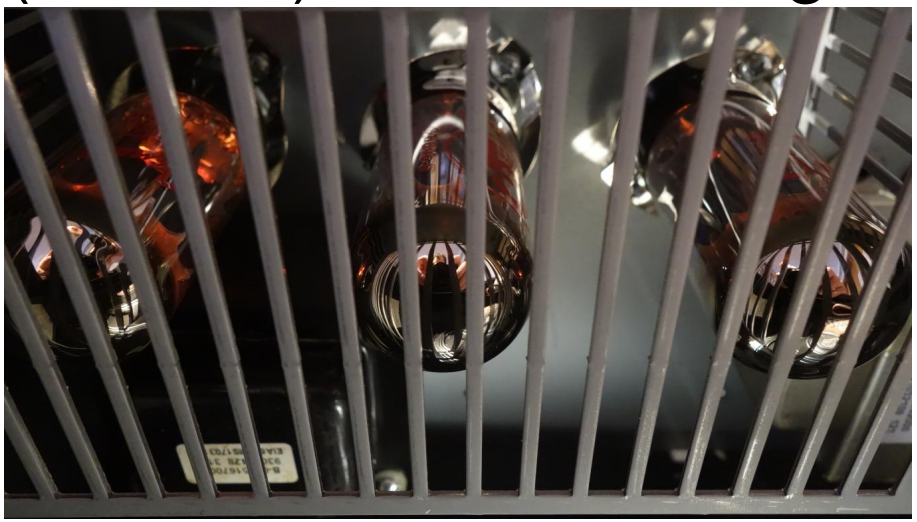


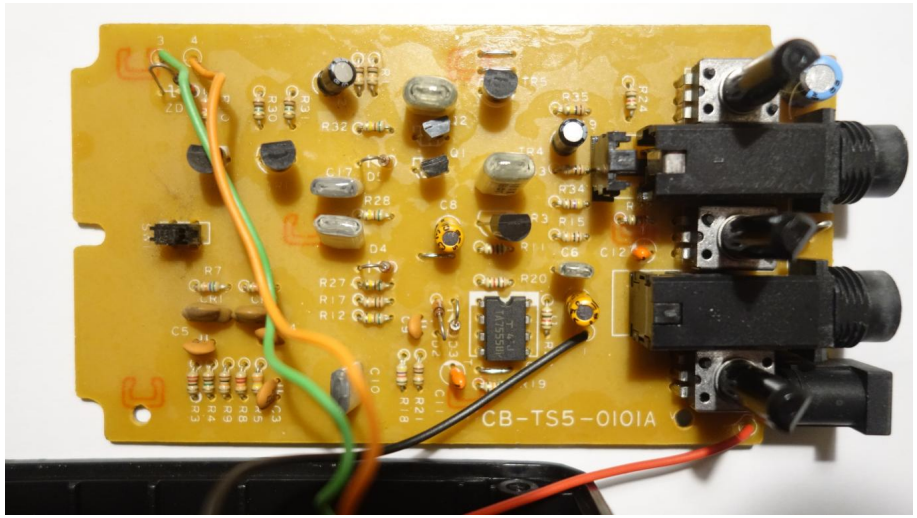


- Ibanez PH 10 (ca. 1990), `https://mirosol.kapsi.fi/2014/02/ibanez-ph10-bi-mode-phaser/`
- Phaser: $f \mapsto f + \text{Allpass}^k(f)$, Allpass$(f)$: erhält Amplituden, verschiebt Phasen (frequenz-abhängig)
- R. G. Keen: *The technology of Phase Shifters and Flangers* 1999, `http://www.geofex.com/Article_Folders/phasers/phase.html`

Nichtlineare Filter (Verzerrer)

- (Röhren)Verstärker: eigentlich (laut, aber trotzdem) linear





- Betrieb außerhalb des linearen Bereiches: technisch möglich und musikalisch interessant (Jimi Hendrix, 1966)
- damit auch Bedarf nach extremen und einstellbaren nichtlinearen Bauteilen (Vorverstärker, Verzerrer)
- sowie Simulation durch Transistoren (preiswert, robust) (aber: ist anderes physikalisches Prinzip, klingt anders) Abb. rechts: Ibanez TS5 Tubescreamer, ca. 1992

Übungsaufgaben

- Experimente mit `https://gitlab.imn.htwk-leipzig.de/waldmann/circuit`, siehe auch autotool-Aufgabe.

  Simulieren Sie einen Allpaß (lattice filter), auch Kette von solchen ($=$ Phaser)

- mit Hydrogen und Rakarrack (oder Guitarix) Aspekte des Schlagzeugs (Rhythmus, Sound) nachbauen:

  Vivien Goldman (und New Age Steppers): *Private Armies Dub* (1981)

  (Produzent: Adrian Sherwood, vgl. *Bugaloo* (2003, video)

- Phaser: für eine Sägezahnschwingung f: bestimmen Sie

Auslenkung und Spektrum des Signals $f + \text{scale}_{-1}(\text{shift}_d(f))$ abhängig von Parameter $d \in [0, \pi]$.

- Echo, Hall, Flanger selbst implementieren:

  WAVE-Datei lesen, bearbeiten (verzögern und ggf. rückkoppeln), schreiben

  anwenden auf: Sinus, Rechteck, Rauschen

  Ansatz: `https://gitlab.imn.htwk-leipzig.de/waldmann/cm-ws18/tree/master/kw43`

- voriges mit dieser Implementierung vergleichen (Steve Harris) `https://github.com/swh/ladspa/blob/master/phasers_1217.xml` (oder andere Open-Source)
- Echo, Hall, Flanger mit `sox`:

– weißes Rauschen erzeugen

```
sox -n noise.wav synth 2 noise
```

– zu zeitversetzem Signal (um 300 Samples) addieren

```
sox -M noise.wav noise.wav out.wav delay 3
```

– mit sonic-visualizer Spektrum betrachten und erklären

Spannungs-gesteuerte Osz. und Filter

Vorläufer: das Theremin (Lev Termen, 1922)

- durch Handbewegung wird Kapazität eines Kondensators in einem HF-Schwingkreis (170 kHz) (!) geändert, dadurch die Frequenz f_1 der Schwingung s_1
- Tonhöhe: $s =$ Tiefpaß$(\max(0, s_1 + s_2))$
  für s_2 aus zweitem (nicht verstimmten) Schwingkreises, s enthält (hörbare) Differenz-Frequenz $|f_1 - f_2|$
- Lautstärke: ähnlich: s_1' durch HF-Bandpaß, $s_2' = 0$ erzeugt Steuerspannung für VCA
- Hörbeispiel: Captain Beefheart: *Electricity*, 1967
- weitere frühere elektronische Instrumente: siehe *120 Years of Electronic Music* `http://120years.net/`

Spannungsgesteuerte Schaltungen

- Steuerung von System-Eigenschaften (z.B. Resonanzfrequenz, Filter-Steilheit) durch
  - Ausgabe-Spannung anderer Teilsysteme
  - Bedienerschnittstelle (Regler, Klaviatur — ebenfalls als Spannungsquellen realisiert)
- $\Rightarrow$ modularer Aufbau eines Synthesizers, Verbindung der Komponenten ($=$ Programmierung) durch Kabel/Stecker
- Robert Moog: *Voltage Controlled Electronic Music Modules*, J. Audio Engineering Soc. Volume 13 Issue 3 pp. 200-206; July 1965,

  `https://www.aes.org/e-lib/browse.cfm?elib=1204`

Spannungsgesteuerte Komponenten

- Verstärker (VCA, voltage controlled amplifier)

  eigentlich Multiplizierer: $U_A(t) = U_C(t) \cdot U_E(t)$
- Oszillator (VCO), Steuerspannung $\sim$ Frequenz

  technische Realisierung: VCA in einer Rückkopplung
- Filter (VCF): Steuerspannung $\sim$ Resonanzfrequenz

  Bsp: Moog Ladder Filter (1965),

  vgl. Tim Stinchcombe: `https://www.timstinchcombe.co.uk/synth/Moog_ladder_tf.pdf`
- periodische U_C mit kleiner Frequenz erzeugt durch

  LFO (low frequency oscillator)

Steuerspannungen aus Benutzeraktionen

- einfachste Möglichkeit: Taste drücken/loslassen

  Impulslänge je nach Eingabe, Impulshöhe konstant
- mehr Ausdruck: Stärke des Tastendrucks bestimmt Impulshöhe (konstant über gesamte Länge)
- (Luxus: *gewichtete* Tastatur, simuliert Trägheit der Klavier-Mechanik)
- (Hüll)kurvenparameter für nicht-konstante Impulse:
  - Attack (Anstiegszeit auf maximale Höhe)
  - Decay (Abfallzeit bei noch gedrückter Taste)
  - Sustain (Impulshöhe nach Decay)
  - Release (Abfallzeit nach Loslassen der Taste)

Erste Synthesizer in populärer Musik

- spannungsgesteuerte modulare Synthesizer produziert ab 1963 (Robert Moog, Don Buchla)
  (Bsp: Buchla: *In The Beginning Etude II*, 1983?)
- erste Anwendungen (auf publizierten Aufnahmen)
  - für exotische Klänge als Verzierung in Standard-Popmusik (Byrds: *Space Odyssey*, 1967)
  - als Solo-Instrument
    - als Ersatz klassischer Instrumente, für klassische Musik (Wendy Carlos: *Switched on Bach*, 1968)
      `https://www.wendycarlos.com/photos.html#studios`
    - für neuartige, eigens komponierte Musik (Morton Subotnick: *Silver Apples of the Moon*, 1967)

Baukastensysteme (alt und neu)

- Erfinder/Hersteller: Robert Moog (1964), Don Buchla (1965) Serge Tcherepnin (1968), Peter Zinovieff (EMS Electronic Music Studios 1965), Alan Robert Pearlman (ARP, 1969) Ikutaro Kakihashi (Roland, 1976),
- zum Selbstbauen: *Elektor Formant* Baupläne von C. Chapman, 1976-1978

  `https://archive.org/details/elektor-1976-12-v-072/`

  `https://www.synthmuseum.com/elektor/eleform01.html`
- Dieter Doepfer (1995), Eurorack-Standard
- Quelle: Kim Bjorn, Chris Meyer: Path and Tweak, 2018

  `https://bjooks.com/products/`

  `patch-tweak-exploring-modular-synthesis`

Simulation mit grafischer Programmierung

- ALSA modular synthesizer
- Komponenten (LFO, VCO, VCF, . . .) auf Arbeitsfläche,
- Verbindung durch Kabel (Ausgangsgrad beliebig, Eingangsgrad 1)
- Verbindungen zur Außenwelt (Bsp.)
  - In: MCV (MIDI control voltage) Steuerspannung ist Tonhöhe der Taste eines virtuellen Keyboards (z.B. `vkeybd`) oder externen Keyboards (z.B. USB-MIDI) qjackctl (Alsa): `virtual keybd` output — `ams` input
  - Out: PCM; qjackctl: `ams` out — `system` in
- Spektral-Analyse in Echtzeit: `jaaa -J`
  (jack audio analyser, Fons Adriaensen, 2004–2010)

Übungen

- Experimente mit *ALSA modular synthesizer (AMS)*.

  Beispiel: File → Demo → `example_ams_demo_bode.ams`

  Parameter einzelner Bausteine ändern (Bsp: VCF)

  Anschlüsse ändern.

  Eine Rückkopplung einfügen (siehe unten *feedback loop*)

  Raten sie die Bedeutung der Einträge im Datei-Inhalt der Synthesizer-Beschreibung.

  – Ausprobieren (einzeln) LFO, VCO, VCF (Moog filter), ADSR (Env), MCV (MIDI control value)

- die zeitliche Umkehrung einer ADSR-Kurve ist im allgemeinen nicht ADSR – sondern nur für welche Parameter?
- Nachbilden bestimmter Klänge
  * base drum,
  * snare drum,
  * der Grashüpfer in „Biene Maja“ (deutsche Tonspur der japanischen Verfilmung von 1975)
  * Becken (hihat, crash, ride)
  * Metallophon,
  * Flexaphon (Hörbeispiel: ca. bei 0:55 in Can: *Sing Swan Song*, 1972)
  * Xylophon,
  * Orgelpfeife, (Pan)Flöte

    * einzelner Wassertropfen, Regen, Wasserfall, Meer

- Steuerung durch interne Quelle (LFO) oder externe: (USB-)MIDI-Keyboard, virtuelles MIDI-Keyboard (`vkeybd`), Sequencer, der MIDI-Signale ausgibt, z.B. `http://www.filter24.org/seq24/`
- zu Video `electric/elektor-formant-2022`: den aufgenommenen Patch in AMS nachbauen
- Schaltpläne nachbauen (Krell patch, drums and percussion, feedback loop)

  die Herausforderung ist: kleine Schaltung (wenige, einfache Bauteile) mit überraschendem Klang.

Programme für Klänge

Motivation

elektrische Schaltungen zur Klangerzeugung . . .

- real bauen (Analog-Synthesizer, Moog, Buchla, . . .)
- oder simulieren. Bedienung/Beschreibung
  - grafisch (alsa modular synthesizer)
  - textuell (durch eine DSL)
    - separate DSL, Bsp: Csound
      `https://csound.com/`, Barry Vercoe 1985
    - eingebettete DSL (in Haskell): csound-expression
      Anton Kholomiov

```
hall 0.5 ( usqr 6 * (sqr (400 * usaw 2.1)))
```

Eine eDSL für (Audio-)Signale

- eDSL = eingebettete domainspezifische Sprache
  benutzt Syntax, Semantik (Typen, Namen), Bibliotheken, Werkzeuge der Gastsprache
  anwendungsspezifische Typen und Funktionen
- Signal ist Funktion von Zeit ($\mathbb{R}$) nach Amplitude ($\mathbb{R}$)

```
type R = Double; newtype Signal = S (R -> R)
constant :: R -> Signal; constant c = S $ \t -> c
osc :: R -> Signal; osc f = S $ \t->sin (2*pi*f*t)
plus, times :: Signal -> Signal -> Signal
plus  (S f) (S g) = S $ \ t -> f t + g t
```

- Bsp. `times (constant 0.3) (osc 300)`
- Source: `https://gitlab.imn.htwk-leipzig.de/waldmann/computer-mu/-/tree/master/effect`

Nützliche Eigenschaften der Gastsprache

- anstatt `times (constant 0.3) (osc 300)`

  schreibe `0.3 * osc 300`

- ```
  instance Num Signal where
    fromInteger i = constant $ fromInteger i
    (+) = plus ; (*) = times
  instance Fractional Signal where
    fromRational r = constant $ fromRational r
  ```

- polymorphe numerische Literale: Compiler ersetzt

  `0.3` $\Rightarrow$ `fromRational 0.3`,

  `300` $\Rightarrow$ `fromInteger 300`.
  ```

Operatoren für Signale (Bsp: shift)

- die Wahrheit: $\text{shift}_d(g) = (t \mapsto g(t-d))$, Implementierung:

```
shift :: R -> Signal -> Signal
shift d (S g) = S $ \ t -> g (t - d)
```

- Erweiterung: Parameter (d) zeitabhängig (ist ein Signal)

```
shift :: Signal -> Signal -> Signal
shift (S f) (S g) = S $ \ t -> g (t - f t)
```

Bsp: `shift (osc 1) s`

- Aufgabe: (spannungs) gesteuerter Oszillator, Ansatz

```
osc :: Signal -> Signal
osc (S f) = S $ \ t -> sin (2 * pi * f t * t)
```

Teste `osc (300 + 30 * osc 1)`, diskutiere.

csound-expression

- eDSL für Signale, Bsp.

```
hall 0.5 ( usqr 6 * (sqr (400 * usaw 2.1)))
```

- Signal wird *symbolisch* repräsentiert
  (als abstrakter Syntaxbaum)
- zur Ausgaben (*rendering*):
  - wird in Csound-Ausdruck kompiliert,
    ansehen mit
    ```
    ( renderCsd $ hall ...) >>= putStrLn
    ```
  - dieser wird vom Csound-Backend interpretiert
    (berechnet Amplitudenverlauf)
    Ausgabe auf Audio-Schnittstelle oder in Datei

CE-Beispiel: Additive Synthese

- Fourier-Darstellung der Rechteck-Schwingung

```
let f = 300
in sum $ map (\k -> (osc $ k * f) / k )
       $ map fromIntegral [1, 3 .. 9]
```

- das funktioniert, weil...
  - `k` und `f` den Typ `Sig` haben (nicht Zahl!)
  - für `Sig` die Addition definiert ist (`instance Num Sig`)
- Klang vergleichen mit `sqr f`, obere Grenze (9) variieren
- Übung: desgl. für Sägezahn-Schwingung,

  für Summe vieler harmonischer S. mit zufälliger Frequenz

weitere Csound/CE-Beispiele und -Quellen

- Wind, Glocke `https://hackage.haskell.org/package/csound-catalog-0.7.2/docs/Csound-Catalog-Wave.html#v:mildWind`
- Schlagzeuge (Hans Mikelson) `https://hackage.haskell.org/package/csound-catalog-0.7.2/docs/Csound-Catalog-Drum-Hm.html`
- Anton Kholomiov: *Speed up you Csound workflow with Haskell*, Csound Journal 23 (2017)
  `http://csoundjournal.com/issue23/Csound_expression_paper.html`
- J. W.: Types in Csound-Expression: `https://www.imn.htwk-leipzig.de/~waldmann/etc/untutorial/ce/`

Schnittstellen für Live-Spiel: MIDI

- MIDI-Signalquelle
  - reelles Keyboard: `https://github.com/spell-music/csound-expression/issues/51`
  - virtuelles Keyboard:

```
vdac $ midi $ \ m -> return $ osc $ sig $ cpsmidi
```

- Signaturen:

```
midi :: Sigs a => (Msg -> SE a) -> SE a
cpsmidi :: Msg -> D
sig :: D -> Sig
osc :: Sig -> Sig
vdac :: RenderCsd a => a -> IO ()
return :: Monad m => a -> m a
```

Schnittstelle für Live-Spiel: GUI

- Verwendung von GUI-Elementen aus Csound:

```
dac $ do
 (g,f) <- slider "f" (expSpan 20 2e4) 440
 panel g ; return $ osc f
```

- Signaturen:

```
slider :: String -> ValSpan -> Double -> Source Sig
expSpan :: Double -> Double -> ValSpan
type Source a = SE (Gui, Input a); type Input a = a
panel :: Gui -> SE () ; osc :: Sig -> Sig
dac :: RenderCsd a => a -> IO ()
```

- mehrere Element kombinieren durch

```
hor,ver :: [Gui] -> Gui
```

Ring-Modulation

- ringmod$(f, g)(t) := f(t) \cdot g(t)$, die Multiplikation der Signale,

```
dac $ (osc 300 + osc 400) * osc 60
```

  ist in analoger Hardware aber *nicht* einfach.
- Warum ist der RM so interessant für die Musik?

  $2 \sin \alpha \cdot \sin \beta = \cos(\alpha - \beta) - \cos(\alpha + \beta)$

  f : 2.5 kHz $+$ 4.5 kHz, g : 500 Hz, RM(f, g): 2,3,4,5 kHz
- Anwendungen in Populärmusik/Film:
  - Mahavishnu Orchestra (John McLaughlin) *On the Way home to Earth* 1975
  - BBC Radiophonic Workshop: die Stimme der Daleks in *Dr. Who* 1963–

Übungen

- csound-expression:
  - die Aufgaben für alsa-modular-synthesizer (Maultrommel usw.)
  - Tonerzeugung: Beispiele anhören `https://gitlab.imn.htwk-leipzig.de/waldmann/cm-ws18/tree/master/kw46/data` und den Csound-Expression-Ausdruck raten. Benutzt wurden nur: osc, usaw, usqr, white, mul, at, (+), (-), (*), hall, fvdelay
  - Erzeugen Sie selbst solche Beispiele! Hochladen und die anderen raten lassen.
  - Automatisierung dieses Aufgabentypes (bewerten, generieren): Masterarbeit (abgeschlossen 2022)

  - benutzen Sie `fvdelay` mit veränderlicher Zeitverschiebung.
    Vergleichen Sie mit dem `shift`-Operator aus
    `https://gitlab.imn.htwk-leipzig.de/waldmann/computer-mu/-/blob/master/effect/Main.hs#L12`
  - GUI benutzen `https://github.com/spell-music/csound-expression/blob/master/tutorial/chapters/FxFamily.md#ui-stompboxes`
  - zum Vergleich — Hörbeispiel: *Autechre*: Perlence (Album: Quaristice, 2008)
- zu Autechre: `https://www.factmag.com/2017/02/25/autechre-gear-synths-samplers-drum-machines-effects`
  lustige Kritik an Max/MSP:
  `https://news.ycombinator.com/item?id=22346556`

- Spektrum eines ring-modulierten Signals betrachten

```
writeSnd "ring.wav" $ setDur 10 $ sqr 300 * saw (10
:! sonic-visualizer ring.wav
```

- Dalek-Stimme nachbauen: Stereo-Signal erzeugen

```
sox .local/share/SuperCollider/downloaded-quarks/Di
```

und modulieren:

```
import Csound.Base; import Csound.Sam
dac $ mul (osc (( usaw 0.5) * 50 )) $ ( loop  $ seg
```

Harmonielehre

Motivation, Plan

- bisher: Geräusche,

  Töne (Grundfrequenz, Obertöne, Spektren)

- heute: welche Töne klingen gut

  – zusammen (in Akkorden),
  – nacheinander (in Melodien)?

- später:

  – Folgen von Akkorden (Kadenzen),
  – Führung mehrerer Stimmen (Kontrapunkt)

  und Notation dafür (algebraisch, grafisch – Partituren)

Klassische Literatur

- Hermann von Helmholtz: *Die Lehre von den Tonempfindungen als physiologische Grundlage für die Theorie der Musik*, Vieweg, Braunschweig 1863.

  https://reader.digitale-sammlungen.de/de/fs1/object/display/bsb10598685_00366.html
- (von H.H. zitiert) Leonhard Euler: *Tentamen novae theoriae Musicae*, Petropoli, 1739. https://scholarlycommons.pacific.edu/euler-works/33/

  vgl. Patrice Bailhache: Music translated into Mathematics,

  https://web.archive.org/web/20050313140417/http://sonic-arts.org/monzo/euler/euler-en.htm
- Hugo Riemann: *Katechismus der Harmonielehre*, Leipzig, 1890 https://archive.org/details/katechismusderh00riemgoog/

Die Naturtonreihe

- bei schwingender Saite, schwingender Luftsäule kommen neben Grundton f ganzzahlige Obertöne vor, bilden die Naturtonreihe $f, 2f, 3f, 4f, 5f, \ldots$
- einzelne Obertöne lassen sich durch passende Spielweise betonen (isolieren) (z.B. Flageolett)

  Bsp: Canned Heat: *On the Road Again*, 196?. (Flageolett-Töne im Intro)
- die Naturtonreihe bis $15f$ reduziert (durch Halbieren)

  1, 9/8, 5/4, 11/8, 3/2, 13/8, 7/4, 15/8, 2

  c d e $\approx$ f g $\approx$ a♭ $\approx$ b♭, $\approx$ b c'

  11/8: das Alphorn-Fa
- wie stimmt man Instrumente mit mehreren Saiten?

Konsonanz

- wie stimmt man Instrumente mit mehreren Saiten $f, g, \ldots$

  g *nicht* als Oberton von f,

  sondern wir wollen neue Töne. Welche?

- Töne wie z.B. 300 Hz, 315 Hz

  – klingen nicht gut zusammen (sondern rauh, dissonant)
  – das Ohr nimmt die Schwebung (mit 15 Hz) wahr

- Schwebungen zw. 10 Hz und 40 Hz sind unangenehm

  (nach Helmholtz: 33 Hz ist am schlimmsten)

  konsonante Töne haben keine solchen Schwebungen

(Vermeiden von) Schwebungen

- f, g konsonant $:=_{\text{def}}$ keine Schwebung geringer Frequenz zwischen Obertönen von f und Obertönen von g.
- $S(f,g) := \min\{|a \cdot f - b \cdot g| : a, b \in \mathbb{N}, af \neq bg\}$

  Bsp: $S(300, 315), S(270, 375), S(270, 360) \ldots$
- Satz: $S(f,g)$ ist der . . . von f und g.

  (Begriff und Berechnung bekannt aus 1. Semester)

Konsonanz

- f, g konsonant, wenn $\gcd(f, g)$ groß ...
  - absolut: $\gcd(f, g) > 40$Hz
  - relativ: $\gcd(f, g)/\max(f, g) \to$ groß
- Hör-Eindruck: $(80, 120)$ gegenüber $(480, 520)$

  spricht für die relative Definition.
- Def: $R(f, g) := \gcd(f, g)/\max(f, g)$

  hier $\gcd(f, g)$ auch für $f, g \in \mathbb{Q}$ definiert als $\min_{a,b}\{af - bg\}$
- Aufg: Bestimme $1 = f_0 < f_1 \ldots < f_k = 2$

  mit $W(f) = \min\{R(f_i, f_j) \mid 0 \leqslant i < j \leqslant k\}$ maximal

  Bsp: $k = 3$. Für $1, 4/3, 5/3, 2$ ist $W(f) = \ldots$, geht besser?

Die Töne nach Pythagoras

- nach Pythagoras (ca. 500 v.Chr.)
- konstruiere Tonmenge
  - beginne mit $f_0 = 1$ (Grundfrequenz)
  - multipliziere mit 3/2
    (die einfachste nichtriviale Harmonie)
  - falls $\geqslant 2$: multipliziere mit 1/2 (die triviale Harmonie)

$$f_0 = 1, \quad f_1 = \frac{3}{2}, \quad \frac{9}{4} \to \frac{9}{8}, \quad \frac{27}{16}, \quad \frac{81}{32} \to \frac{81}{64}, \quad \frac{243}{128}, \quad \frac{729}{256} \to \frac{729}{512}, \quad \dots$$

- alle Werte sind paarweise verschieden
- endliche Tonmengen bis zu Werten nahe bei 1 (bzw. 2)
  - $f_5 = 243/128 \approx 2 \cdot 0.95$, pentatonische Skala: $f_0, \dots, f_4$
  - $f_7 = 2187/2048 \approx 1.07$, diatonische Skala: $f_0, \dots, f_6$
  - $f_{12} = 3^{12}/2^{19} \approx 1.01$, chromatische Skala: $f_0, \dots, f_{11}$

Herleitung der Pentatonik

- $\begin{array}{cccccccc} f_0 & f_1 & f_2 & f_3 & f_4 & f_5 & f_6 & f_7 \\ 1, & \frac{3}{2}, & \frac{9}{4} \to \frac{9}{8}, & \frac{27}{16}, & \frac{81}{32} \to \frac{81}{64}, & \frac{243}{128}, & \frac{729}{256} \to \frac{729}{512}, & \frac{2187}{1024} \to \frac{2187}{2048} \\ 1 & 1.5 & 1.12 & 1.69 & 1.27 & 1.90 & 1.42 & 1.07 \end{array}$
- *pentatonische* Skala: $f_0, \ldots, f_4$,

  nach Frequenzen geordnet: $f_0 < f_2 < f_4 < f_1 < f_3 (< 2f_0)$
- Abstände (Verhältnisse) benachbarter Töne:

  $f_2/f_0 = f_4/f_2 = f_3/f_1 = 9/8 = 1.125$,
  $f_1/f_4 = f_0/f_3 = 32/27 \approx 1.185$
- Bsp: The Monochrome Set: *Iceman*, 2015.

  Intro verwendet Töne $\{d, e, f\sharp, a, b\}$,

  das ist Pentatonik mit Grundton d.

Herleitung der Diatonik

- $\begin{array}{ccccccccccccc} f_0 & & f_1 & & f_2 & & f_3 & & f_4 & & f_5 & & f_6 & & f_7 \\ 1, & & \frac{3}{2}, & & \frac{9}{4} \to \frac{9}{8}, & & \frac{27}{16}, & & \frac{81}{32} \to \frac{81}{64}, & & \frac{243}{128}, & & \frac{729}{256} \to \frac{729}{512}, & & \frac{2187}{1024} \to \frac{2187}{2048} \\ 1 & & 1.5 & & 1.12 & & 1.69 & & 1.27 & & 1.90 & & 1.42 & & 1.07 \end{array}$
- *diatonische* Skala: $f_0, \ldots, f_4, \underline{f_5}, \underline{f_6}$,
  geordnet: $f_0 < f_2 < f_4 < \underline{f_6} < f_1 < f_3 < \underline{f_5} (< 2f_0)$
- Abstände (Verhältnisse) benachbarter Töne:
  $f_2/f_0 = \cdots = 9/8 = 1.125$, $f_1/f_6 = 2f_0/f_5 = 2^8/3^5 \approx 1.05$.
- Anwendung: weiße Tasten (ganze Töne) auf dem Klavier
  $f_0 = F, f_2 = G, f_4 = A, f_6 = B, f_1 = C, f_3 = D, f_5 = E$
- Benennung: alphabetisch, im Deutschen: H statt B
  siehe auch `https://krebszuchtaufamrum.bandcamp.com/track/es-ist-ein-b-du-arsch`

Herleitung der Chromatik

- $f_0 = 1, f_1 = 3/2, f_2 = 9/8, \ldots, f_{12}/f_0 = 3^{12}/2^{19} \approx 1.01$
- *chromatische Skala*: $f_0, \ldots, f_{11}$, geordnet:

  $f_0 < \underline{f_7} < f_2 < \underline{f_9} < f_4 < \underline{f_{11}} < f_6 < f_1 < \underline{f_8} < f_3 < \underline{f_{10}} < f_5$

  unterstrichen sind die neuen (nicht diatonischen) Töne

  $\approx$ die schwarzen Tasten (Halbtöne) auf dem Klavier
- Bezeichnungen: nach den ganzen Tönen,

  $f_1 = C$, $f_8 = C\sharp =$ Cis, $f_3 = D$
- Abstände sind $f_7/f_0 = f_9/f_2 = \cdots = 3^7/2^{11} \approx 1.07$,
  $f_2/f_7 = f_4/f_9 = \cdots = 2^8/3^5 \approx 1.05$

  d.h., in dieser Stimmung gibt es zwei verschiedene Halbton-Abstände

Eigenschaften, weitere Stimmungen

- die pythagoreische Reihe:
  - schließt nicht $1 \neq 3^{12}/2^{19}$ (pythagoreisches Komma)
  - Konsonanzen aus der Naturtonreihe fehlen, z.B. 4:5:6. angenähert durch $c : e : g = f_1 : f_5 : f_2 = 1 : \frac{81}{64} : \frac{3}{2}$ der Fehler $\frac{81}{64}/\frac{5}{4}$ ist das diatonische Komma
  - unterschiedlich große Halbtöne $\Rightarrow$ Transposition (Verschiebung) ändert Frequenz-Verhältnisse
- *reine Stimmung*: 4:5:6 für spezielle Akkorde:

  Tonika c,e,g, Subdominante f,a,c, Dominante g,b,d.
- *gleichtemperierte S.*: pythagoreisches K. wird geschlossen: jeder Halbton ist $2^{1/12} \approx 1.06$, dann $g : c = 2^{7/12} \approx 1.4983 \neq 3/2$, invariant unter Transposition

Die diatonische Skala

- T: Ganzton, S: Halbton: $c \overset{T}{-} d \overset{T}{-} e \overset{S}{-} f \overset{T}{-} g \overset{T}{-} a \overset{T}{-} b \overset{S}{-} c'$
- hiervon sind die Intervallbezeichnungen abgeleitet:

  Sekunde, Terz, Quarte, Quinte, Sexte, Septime, Oktave.
- es gibt zwei Terzen:

  große Terz ($2T = 4S$) $c - e$, kleine Terz ($T + S = 3S$): $e - g$

  zwei Septimen: kleine ($10S$) $d - c'$, große ($11S$): $c - b$
- der *Modus* beschreibt eine zyklische Verschiebung:
  - ionisch (Dur): $c, d, \ldots,$
  - äolisch (Moll): $a, b, \ldots.$

Akkorde (Dreiklänge)

- Grundformen (konsonant):
  - Dur (große Terz, kleine Terz) $C = \{c, e, g\}$
    in C-Dur-Skala enthalten: C, F, G
  - Moll (kleine Terz, große Terz) $C^- = \{c, e\flat, g\}$
    in C-Dur-Skala enthalten: D^-, E^-, A^-
- Modifikationen (dissonant):
  - vermindert: (kleine, kleine) $C^0 = \{c, e\flat, g\flat\}$
    in C-Dur-Skala enthalten: B^0
  - vergrößert: (große, große) $C^+ = \{c, e, g\sharp\}$
    nicht in C-Dur-Skala enthalten.

Akkorde (Vierklänge)

- konsonanter Dreiklang plus Septime (kleine oder große)
- Bsp: $C^{7} = \{c, e, g, b\flat\}, C^{\text{maj7}} = \{c, e, g, b\}$
- skalen-eigene Vierklänge:
  $C^{\text{maj7}} = \{c, e, g, b\}, D^{-7} = \{d, f, a, c\},$
  $E^{-7}, F^{\text{maj7}}, G^{7}, A^{-7}, B^{-7(\flat 5)}$
- simple Realisierung in `electribe 2`:
  - Dur-Skala, 4 Noten pro Akkord (Grundton, $+2, +4, +6$), da kann überhaupt nichts schief gehen, . . .
  - auch bei „frei improvisierter“ Melodie nicht (XY-Pad: X ist Tonhöhe (aus Skala), Y ist Arpeggio)
  - das klingt aber doch beliebig, woher kommt die musikalische Spannung?

Aufgaben

1. bestimmen Sie die Frequenzverhältnisse für C-Dur, d-Moll und e-Moll in der C-Dur-Skala bei Stimmung
   - diatonisch
   - rein
   - gleich temperiert

   und vergleiche Sie akustisch (csound-expression)
2. Konstruktion der chromatischen Töne nach Paul Hindemith (Unterweisung im Tonsatz, 1937):

(a) zu jedem Ton aus der Obertonreihe des Grundtons (c) werden mögliche Grundtöne bestimmt. Bsp: $5 \cdot c = 4 \cdot ?$. Dabei Multiplikation mit $1 \ldots 6$, Division durch $1, (2), 3, (4), 5$, mit Identifikation von Oktaven.

Welche Töne entstehen aus c?

(b) Dieser Vorgang wird für jeden der entstandenen Töne wiederholt.
Welche neuen Töne enstehen? Sind die Abstände gleichmäßig (oder fehlen noch Töne)? Vergleich mit pythagoreischer Skala.

3. was hat H. Helmholtz auf S. 291f. gerechnet/gezeichnet? Rekonstruieren Sie die „einfachste mathematische Formel“, erzeugen Sie daraus die Diagramme, vergleichen Sie mit denen im Buch
4. was hat L. Euler gerechnet? (Helmholtz S. 349, Fußnote) Überführen Sie die dort zitierte rekursive Definition der Stufenzahl in eine explizite Formel.

Bestimmen Sie die Stufenzahl der Akkorde aus der 1.

Aufgabe.

Wo steht die Definition im Originaltext von Euler?

5. Welches sind (nach 5, 7, 12) die nächsten interessanten Längen von pythagoreischen Tonfolgen?

   Betrachten Sie dazu die Verhältnisse benachbarter Töne (Pentatonik: 32/27 und 9/8, Diatonik: 9/8 und $2^8/3^5$, Chromatik: $2^8/3^5$ und $3^7/2^{11}$). Wann verschwindet (durch hinzukommende Zwischentöne) das größere der beiden chromatischen Verhältnisse? Welche anderen Verhältnisse gibt es dann? Wie geht das weiter?

   Gibt es solche Skalen in der (historischen) Musikpraxis? (Aktuell vgl.
   `https://oddsound.com/usingmtsesp.php`,

Oddsound und Richard D. James (= Aphex Twin), 2021)

6. mit alsa-modular-synthesizer (Module: CV: Random, Quantizer — werden auch in einigen Demos verwendet) oder csound-expression
   - Akkorde (Dreiklänge, Vierklänge) erzeugen.
   - Akkorde aus einer Skala zufällig aneinanderreihen,
   - dazu eine zufällige Melodie aus dieser Skala

   Spezifikation von Tonfolgen in CSE vgl.

```
notes = fmap temp $ fmap (220 * ) [1, 5/4,
q = mel [mel notes, har notes]
dac $ mix $ sco oscInstr q
```

```
https://github.com/spell-music/csound-expression/
blob/master/tutorial/chapters/ScoresTutorial.md#
```

7. zur Stimmung der Gitarre:

- man kann die unteren (tiefen) Saiten so stimmen: Saite mit Flagoelett bei 1/4 $=$ nächst-höhere Saite mit Flageolett bei 1/3.
  Welches Intervall ist das? Wenn man bei tiefem E beginnt und alle Saitenpaaare so stimmt, welcher Ton ist dann auf der 6. (höchsten) Saite?
  Das Intervall zwischen 4. und 5. Saite wird bei üblicher Stimmung um einen halben Ton verringert.
- Es werden gern auch abweichende Stimmungen verwendet, vgl. `http://www.sonicyouth.com/mustang/tab/tunings.html`
  Warum?

Bsp: Sonic Youth: *Hyperstation*, Album: Daydream Nation (1988) — Das Bild auf der Hülle ist `https://www.gerhard-richter.com/en/art/paintings/photo-paintings/candles-6/candle-5195/`

- Stimmungen für Hawaii-Gitarre: John Ely (2008) `http://www.hawaiiansteel.com/tunings/my_tunings.php`
  Bei C6th und A6th: welche benachbarten Saiten ergeben Dur- und Moll-Dreiklänge?
  Rechnen Sie die Frequenzen für die angegebenen Verstimmungen nach (z.B. C6th: G plus 6 cent). Welche Frequenzverhältnisse werden dadurch für die Akkorde erreicht?

8. Bestimmen Sie die Abweichung des Alphorn-Fa vom nächsten chromatischen Ton in Cent.

Algebraische Komposition

Einleitung

- klassisch: Musikstück repräsentiert durch Partitur,
  - Ton repräsentiert d. Note, bezeichnet Tonhöhe, -dauer
  - Tempo, Klangfarbe, Lautstärke
    - durch weiter Annotationen spezifiziert
    - oder nicht, d.h., dem Interpreten überlassen
  - Komposition:
    - Noten nebeneinander bedeutet Töne nacheinander
    - Noten (Zeilen) übereinander: Töne (Stimmen) gleichzeitig
- jetzt: Musikstück repräsent. d. (abstrakten Syntax-)Baum

Literatur, Software

- Paul Hudak , Tom Makucevich , Syam Gadde , Bo Whong: *Haskore Music Notation - An Algebra of Music*, JFP 1995, `https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.8687`

  Partitur wird kompiliert zu MIDI-Strom, der von Hard- oder Software-Synthesizer interpretiert wird

  Donya Quick et al.: `https://euterpea.com/`

- Anton Kholomiov: Csound-Expression Tutorial – Scores, `https://github.com/spell-music/csound-expression/blob/master/tutorial/chapters/ScoresTutorial.md`

  Partitur wird durch Csound-Instrumente interpretiert, P. kann Csound-spezifische Elemente enthalten

Partituren als Abstrakte Syntaxbäume

- ```
  data Music a
    = Prim (Primitive a)
    | (Music a) :+: (Music a)   -- sequentielle K.
    | (Music a) :=: (Music a)   -- parallele K.
    | Modify Control (Music a)
  data Primitive a = Note Dur a | Rest Dur
  data Control = Tempo Rational
    | Transpose AbsPitch
    | Instrument InstrumentName | ...
  ```
- Beispiele

```
Prim (Note 1 60) :: Music AbsPitch
Prim (Note 1 (C,4)) :: Music Pitch
Modify (Transpose 4)
    (Prim (Note 1 (C,4)) :=: Prim (Note 1 (G,4)))
```
```

Konstruktion von Partituren

```
tr = transpose
minor x = chord [x, tr 3 x, tr 7 x]
bass x = tr (-12) $ line
  [ scaleDurations (1/2) x, tr 7 x, tr (-5) x ]
pat x = chord
  [ instrument AcousticGrandPiano
    $ line [ Rest qn, minor x, Rest qn, minor x ]
  , instrument AcousticBass $ bass x ]
theme = line $ map (\x -> pat $ x 2 qn) [ a,e,d,a ]
```

- benutzerdefinierte Namen (`tr`), Funktionen (`minor`), Standard-Funktionen (`map`)
- beschreibt diesen AST:

```
(Modify (Instrument AcousticGrandPiano) (Prim (Rest (1 % 4)) :+: ((Prim (Note
(1 % 4) (A,2)) :=: (Modify (Transpose 3) (Prim (Note (1 % 4) (A,2))) :=: (Modify
(Transpose 7) (Prim (Note (1 % 4) (A,2))) :=: Prim (Rest (0 % 1))))) :+: (Prim
(Rest (1 % 4)) :+: ((Prim (Note (1 % 4) (A,2)) :=: (Modify (Transpose 3) (Prim
(Note (1 % 4) (A,2))) :=: (Modify (Transpose 7) (Prim (Note (1 % 4) (A,2))) ...
```

Von Partitur zu Interpretation

- ```
  data MEvent = MEvent
    {  eTime    :: PTime, -- onset time
       eInst    :: InstrumentName, -- instrument
       ePitch   :: AbsPitch, -- pitch number
       eDur     :: DurT, -- note duration
       eVol     :: Volume, -- volume
       eParams  :: [Double] } -- optional other parame
  type Performance = [ MEvent ] -- aufsteigende onset
  ```

- ```
  musicToMEvents
     ::  MContext -> Music1 -> (Performance, DurT)
  musicToMEvents
        c@MContext{mcTime=t, mcDur=dt} (m1 :+: m2) =
   let (evs1,d1) = musicToMEvents c m1
       (evs2,d2) = musicToMEvents c{mcTime = t+d1} m2
   in  (evs1 ++ evs2, d1+d2)
  ```

Partitur und Interpretation

- die Konstruktoren der Partitur definieren die Signatur Σ einer Algebra
- die Partituren sind Terme über dieser Signatur
- Performances bilden die Trägermenge D einer Σ-Algebra
- die Aufführung (`perform :: Music a -> Performance`) ist die Interpretation von Term nach Algebra:

  dabei wird jedes Symbol aus Σ durch eine Funktion über D ersetzt, Bsp:

  – `:+:` durch `++` (Verkettung),
  – `:=:` durch `merge` (Zusammenfügen)

Eigenschaften der Operationen

- für die zweistelligen Kompositionen

```
seq2, par2 :: Score -> Score -> Score
seq2 = (:+:), par2 = (:=:)
```

- `seq2` ist semantisch assoziativ: für alle p, x, y, z

```
perform p (seq2 (seq2 x y) z)
  = perform p (seq2 x (seq2 y z))
```

  neutrales Element? kommutativ? Desgl. für `par2`

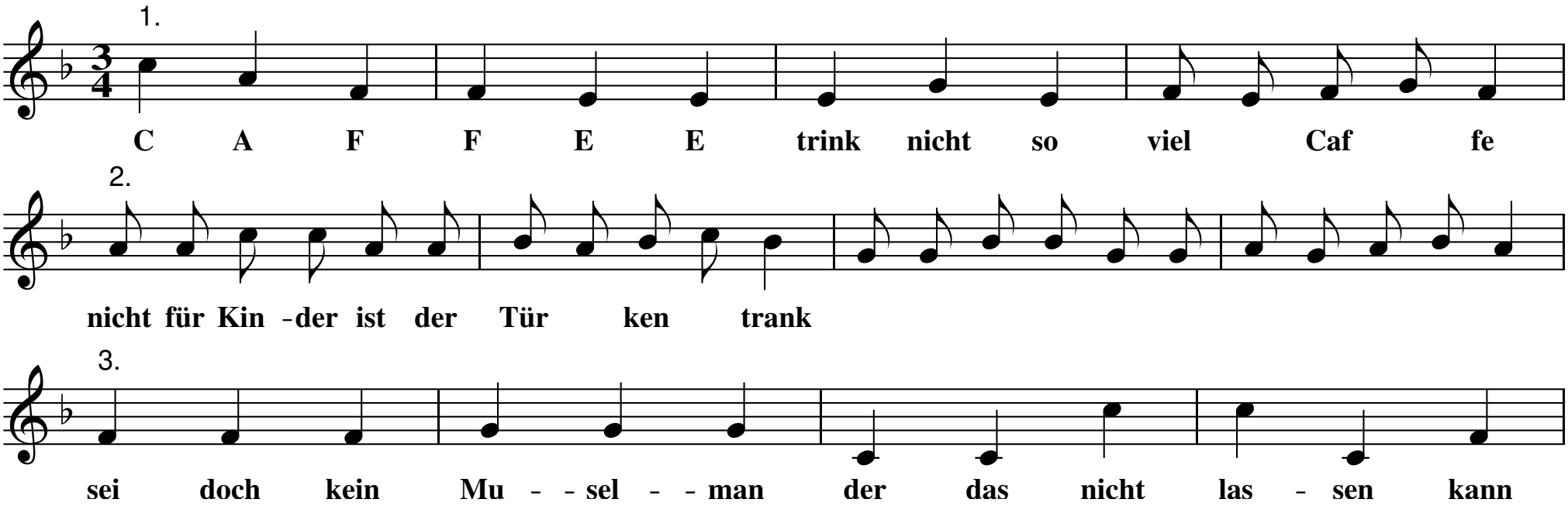
- gelten Distributiv-Gesetze? (Nein).
- Ü: wann sind `par2 (seq2 a b) (seq2 c d)` und `seq2 (par2 a c) (par2 b d)` semantisch gleich?

Historische Formen der Mehrstimmigkeit

- Cantus Firmus (feststehende Melodie, die anderen Stimmen sind Verzierung)
- Kontrapunkt (Note gegen Note): Vorschriften zur Konstruktion der Begleit-Stimmen, u.a.
  - Konsonanzen zu bestimmten (schweren) Zeitpunkten
  - keine Parallelen (gleichmäßiges Auf- oder Absteigen)
- Fuge (Flucht, die Stimmen fliehen voreinander) alle Stimmen sind aus *einem* Thema konstruiert durch
  - zeitlichen Versatz (Kanon), zeitliche Spiegelung
  - Versatz der Tonhöhe, Skalierung des Tempos, ...
- Kadenzen (Akkordfolgen) mit untergeordneten Stimmen

Kanon

- eine Stimme wird mehrfach zeitlich versetzt
- Beispiel: Karl Gottlieb Hering (1766-1853): *C A F F E E*



- Übung: 1. Harmonien bestimmen, 2. programmieren

Fuge

- eine anspruchsvolle Form des Kontrapunktes. alle Stimmen sind aus *einem* Thema konstruiert durch
  - zeitlichen Versatz (wie im Kanon)
  - Versatz der Tonhöhe
  - zeitliche Spiegelung
  - Tonhöhen-Spiegelung
  - Skalierung des Tempos
- Johann Sebastian Bach (1685–1750): *Die Kunst der Fuge*, Contrapunctus XV - Canon per Augmentationem in Contrario Motu, (Solist: Pierre-Laurent Aimard, 2008)

  `https://www.mutopiaproject.org/ftp/BachJS/BWV1080/contrapunctusXV/`
- Ü: Operatoren in Partitur erkennen, implementieren

Akkorde (Ton-Inhalt)

- Grundformen: Dur und Moll

```
tr = transpose
major x = chord [ x, tr 4 x, tr 7 x ]
minor x = chord [ x, tr 3 x, tr 7 x ]
```

- mit Septime (kleiner, großer)

```
major7 x = chord [ x, tr 4 x, tr 7 x, tr 10 x ]
major7maj x = chord [ x, tr 4 x, tr 7 x, tr 11 x ]
```

- weitere Varianten durch Umstellen (anderer Grundton); Hinzufügen, Ändern, Weglassen von Tönen

Die Kadenz

- lateinisch cadere $=$ fallen
- die Voll-Kadenz (Quinten abwärts) in C-Dur:

3kl.	C	F	B^0	E^-	A^-	D^-	G	C
4kl.	C^{maj7}	F^{maj7}	$B^{-7(\flat 5)}$	E^{-7}	A^{-7}	D^{-7}	G^7	C^7
Ton	I	IV	VII	III	VI	II	V	I
	T	S		Dp	Tp	Sp	D	T

- verkürze, ersetze $B^0 = \{b, d, f(, a)\}$ durch $G = \{g, b, d(, f)\}$,

  ergibt die Kadenz: $C, F, G, C = T, S, D, T$
- Tonika (1,3,5), Dominante (5,7,9),Subdominante (4,6,8),
- Dur: $T = (c, e, g)$, $S = (f, a, c)$, $D = (g, b, d)$,
- Moll: $t = (c, e\flat, g)$, $s = (f, a\flat, c)$, $d = (g, b\flat, d)$

Diatonik in Euterpea

- Euterpea verwendet chromatische Tonleiter (wg. MIDI)
- anderen Skalen dorthin umrechnen, z.B.

```
dia x =
  let scale = [0,2,4,5,7,9,11]
      (d,m) = divMod x (length scale)
  in note qn $ 60 + d*12 + scale !! m
```

- dann Tetrachord (vgl. Electribe Chord-Modus)

```
tetra x = chord $ map dia [x,x+2,x+4,x+6]
```

Funktions-Harmonik

- Betrachtung der Akkorde nach ihrer (vermuteten, häufigen) Funktion in musikalischer Phrase.

  nach Hugo Riemann (1849–1919):

  T These, S Antithese, D Synthese.
- in einer Kadenz können Akkorde durch Parallelen vertreten werden
- die Parallelen (mit gleicher großer Terz)

  $Tp = (-1, 1, 3) = (a, c, e)$, $tP = (3, 5, 7) = (e\flat, g, b\flat)$,

  entsprechend Dp, dP, Sp, sP
- The Beatles: *Penny Lane*: T, Tp, Sp, D (C, Am, Dm, G)
- harmonische Analyse einer Stelle aus Bach: BWV 268

Vermischte Dokumente zur Harmonielehre

- Über Hugo Riemann, von dessen Sohn Robert:
  `http://www.hugo-riemann.de/`
- Kritik an Riemann durch Heinrich Schenker (1868–1935)
  `https://web.archive.org/web/20120403032916/http://www.schenkerdocumentsonline.org:80/profiles/person/entity-000712.html`
- Kritik an einer Kritik an Schenkers Theorie der *Urlinie*
  `https://web.archive.org/web/20160731145955/http://schenkerdocumentsonline.org/documents/other/OJ-21-24_1.html`

Kadenzen in der Popmusik

- Kadenz T S (T) D T
- Beispiele: tausende, u.a. Beach Boys: *Little Honda*, 1964 (Version von Yo La Tengo, 1997)

  Strophe: $DDDD|GGDD|AADA$
- The Jesus and Mary Chain: *Upside Down*, 1984 (auf Creation Records). Strophe: $4 \cdot (GGGC)\ 4 \cdot C\ 4 \cdot G$
- Beatles: *Tomorrow Never Knows*, 1966.
- Lou Reed: „One chord is fine. Two chords is pushing it. Three chords and you're into jazz."
- Thelonius Monk: *Round Midnight*, 1944

Übungen

1. zu Folie „Partitur und Interpretation“:

   (a) Warum „schwach monoton“, nicht stark?
   (b) Welche Rechnung muß im Zweig `Par2 x y ->` stattfinden? Wie werden die Teilresultate verknüpft?
   (c) Welches ist der abstrakte Datentyp für `[Event]` (welche Operationen gehören zur API)? Welche effiziente Implementierungen dafür kennen Sie?

2. Fragen von Folie „Eigenschaften der Operationen“

3. zu Bach: Contrapunktus XV (canon per augmentationem in contrariu motu)

   (a) Bestimmen Sie die globale zeitliche Struktur der

Komposition.
Der 1.Takt der 1. Stimme erscheint (gedehnt und gespiegelt) in Takt 5 und 6 der 2. Stimme. Wo noch? Was zeigt der Trennstrich nach Takt 52 an?

(b) Bestimmen Sie die Tonhöhen-Abbildung (Spiegelung) von erster zu zweiter Stimme.
Lesehilfe: Der Violin-Schlüssel bezeichnet das G (der Kringel, zweite Notenlinie von unten), der Baß-Schlüssel bezeichnet das F (der Doppelpunkt, zweite Notenlinie von oben)

4. Programmieren Sie das Thema von Jean-Michel Jarre: Oxygen Pt. 2 als eine Verschmelzung von zwei einfachen Melodien

5. Programmieren Sie den CAFFEE-Kanon (3 Stimmen,

jede mit eigenem Instrument).

(a) Ergänzen Sie `https://gitlab.imn.htwk-leipzig.de/waldmann/cm-ws18/blob/master/kw47/Caffee.hs`
Beschreibung der Bibliotheks-Funktionen:
`https://github.com/spell-music/csound-expression/blob/master/tutorial/chapters/ScoresTutorial.md`
oder in Euterpea

(b) Benutzen Sie eine Darstellung (d.h., Unterprogramme), die die lokale Strukur ausnutzt, z.B.: zweite Häfte der 2. Zeile ist Transposition der ersten Hälfte.
Wir verschieben nicht chromatisch (2 Halbtöne), sondern diatonisch (1 Ton in der F-Dur-Skala).

6. Realisieren Sie auf ähnliche Weise eine Voll-Kadenz

   (a) effizient programmieren unter Benutzung der Skalen-Numerierung
   (b) eine dazu passende Melodie programmieren
   Hinweis: jede Melodie (aus Skalentönen) paßt

7. verschiedene Software-Synthesizer ausprobieren zum Abspielen von mit Euterpea erzeugten MIDI-Strömen:

   - (in der VL gezeigt) fluidsynth/qsynth
     `fluidsynth -s -a jack -j -r 48000`
   - Alsa Modular Synthesizer (Verwenden Sie Demos, die das MIDI-Input-Element `MCV` enthalten)
   - Csound-Expression
     `dacBy ( def {csdFlags = def { rtmidi = Jus`

```
  $ midi $ \ m -> return  $ osc $ sig $ cp
```

https://github.com/spell-music/csound-expression/issues/51#issuecomment-437162344

8. Euterpea: von Partitur zu MIDI und zurück

```
song = line ...
writeMidi "foo.midi" song
Right m <- importFile "foo.midi"
song' = fromMidi m
```

Dann `song` und `song'` vergleichen

Performing with Patterns of Time

Überblick

- Quelle: Thor Magnusson und Alex McLean: P.w.P.o.T, Kap. 14 in: Oxford Handbook of Algorithmic Music, OUP 2018, `https://slab.org/publications/`

  Software: `https://tidalcycles.org/`
- algebraische Beschreibung von periodischen Verläufen (Parameter für Klänge), eingebettete (in Haskell) DSL
- Back-end: `https://supercollider.github.io/` James McCartney, 1996–
- Tidal benutzt SC zum Abspielen von Samples
- Tidal ist System für live-coding (durch ghci-Kommandos)

Tidal - Beispiel

- Sound-Server (supercollider) starten

```
sclang dirt_startup.scd
```

- Ghci starten

```
ghci
:script BootTidal.hs
```

- Klänge ausgeben

```
d1 $ s "bd [sn sn]"
d2 $ s "[jvbass*2]*2" |*| n "0 1 2 3 4"
hush
```

Grundlagen Tidal (Modell)

- ein Muster `m :: Pattern a` beschreibt eine periodische Abbildung von Zeit nach `a`
- elementares Muster: `pure x` mit Periode 1
- `c :: ControlMap` Parameter zum Sample-Abspielen

  s: Verzeichnis, n: Datei-Nummer, begin, end, speed ...
- Funktionen zum Abspielen:

```
d1, d2, ... :: Pattern ControlMap -> IO ()
```

- ```
  let p = pure $ M.fromList [("s", VS "bd")]
  d1 p
  queryArc p (Arc 0 3)
    [(0>1)|s: "bd",(1>2)|s: "bd",(2>3)|s: "bd"]
  ```
  ```

Transformation von Mustern

- Bsp. verwenden `p = run 3`, mit `queryArc p (Arc 0 1)`

```
==> [(0>1/3)|0, (1/3>2/3)|1, (2/3>1)|2]
```

- Transformation der Werte `fmap (\ x -> x+1) p`

```
==> [(0>1/3)|1, (1/3>2/3)|2, (2/3>1)|3]
```

- zeitliche Verschiebung `(1/3) <~  run 3`

```
==> [(0>1/3)|1, (1/3>2/3)|2, (2/3>1)|0]
```

- zeitliche Streckung `slow 2 p`

```
[(0>2/3)|0, (2/3>1)-4/3|1, 2/3-(1>4/3)|1, (4/3>2)|2
```

Parallele Komposition

- parallele Komposition (Vereinigung der Ereignismengen)

```
stack :: [Pattern a] -> Pattern a

fast 3 $ stack [ slow 2 (s "sn"), slow 3 (s "bd") ]
```

- parallele Komposition mit Kombination der Werte

```
(<*>) :: Pattern (a -> b) -> Pattern a -> Pattern b
```

  Struktur von: beiden Seiten (`<*>`), links (`<*`), rechts (`*>`)

  - `p |+| q = (pure (+) <*> p) <*> q`
  - `(#) = (|>)` = Struktur von links, Werte von rechts
    vgl. `p # gain 0.7 # gain 0.3` mit
    `p # gain 0.7 |* gain 0.3`

Sequentielle Komposition? (Verschränkung)

- Nicht wie in z.B. `:+:` in Euterpea,

  weil ein Tidal-Muster kein Ende hat!

- Verschränkung von Mustern:

  ```
  cat :: [Pattern a] -> Pattern a
  ```

  für $p = \texttt{cat}[p_0, .., p_{k-1}]$ bedeutet
  $p[0 \ldots 1] = p_0[0 \ldots 1], p[1 \ldots 2] = p_1[0 \ldots 1], p[2 \ldots 3] = p_2[0 \ldots 1], p[3 \ldots 4] = p_0[1 \ldots 2], p[4 \ldots 5] = p_1[1 \ldots 2], \ldots$

- Denkmodell: jedes Muster ist Tonbandmaschine (Spur), bei $\texttt{cat}[p_0, .., p_{k-1}]$ spielt der Reihe nach jede Maschine p_i für 1 Einheit

Die Muster-DSL von Tidal

- zusätzlich zur bisher beschriebenen eDSL:

  konkrete Syntax für Operationen, Transformationen:

  Bsp: `"[[bd sn?]*2, [hc?*2 ho]*4]"`

  – Hintereinander: in `<>`: `cat`, in `[]`: `fastcat`,
  – Komma in (beiden) Klammern: `stack` (außen)
  – `x*3` bedeutet: `fast 3 x`, `x/2` bedeutet: `slow 2 x`
  – `x?`: `degrade x` (einige Ereignisse weglassen)
- `s $ fromString "[bd sn]"` ist äquivalent zu

  ```
  s $ fastcat [ pure "bd", pure "sn" ]
  ```

  mit `:set -XOverloadedStrings` : `s "[bd sn]"`
- damit kürzere Notation, aber Vorteile der eDSL (statische Typisierung, Funktionen, HO) werden aufgegeben

Audio-Effekte in Tidal

- Ausdrucksmittel sind hier (z.B. ggü. csound-expression) absichtlich beschränkt, Schwerpunkt von Tidal ist die Kombination von (zeitlichen) Mustern, nicht von Effekten
- ein globaler Effekt-Weg, Parameter in `ControlMap`

```
d1 $ sound "sn"
  # delay 0.7 # delaytime (2/3) # delayfeedback 0.7
  # room 0.7  # size 0.9
  # cutoff 400 # resonance 0.7
```

- Parameter sind auch Muster, z.B., `size "0.5 0.9"`
- durch `orbit <string>` unabhängige Effektstrecken

```
stack [ .. # orbit "0", .. # orbit "1" ]
```

- (seit 1.7): *control bus*: ändert Parameter für aktive Effekte

Live Coding

- der eigentliche Erfindungsgrund und Anwendungsfall von Tidalcycles ist „live coding“: sichtbare Arbeit am Quelltext (wird projiziert) während der Aufführung
- ghci sendet Ereignisse an Backend (supercollider), falls Eingabe typkorrekt. falls nicht: bleibt bisheriges Muster.
- *from scratch live coding*: Editor ist anfangs leer (und der gesamte Text bleibt auf einer Bildschirmseite)
- *kollaboratives Live-Coding*. Bsp.: Damian Silvani: *Web-based P2P collaborative editor for live coding music and graphics* `https://munshkr.github.io/flok/`
- aktuell (2023) 21./22. Dezember: 36 Stunden live coding `https://solstice.toplap.org/`

Live Coding mit Tidalcycles: Künstler

- Alex McLean, *Making music with text[ure]*, `https://slab.org/publications/`
- Mike Hodnick, *I program computers and music.* `https://www.kindohm.com/`

  Kindohm at International Conference on Live Coding, October 15th 2016, at The Spice Factory, Hamilton, Ontario, Canada.

  `https://www.youtube.com/watch?v=smQOiFt8e4Q`

  `https://github.com/kindohm/365tidalpatterns`
- vgl.

  `https://web.archive.org/web/20170609193157/http://iclc.livecodenetwork.org/2015/papers.html`

Übungen

1. Tidal installieren und starten

(a) jack richtig configurieren, siehe `https://gitlab.imn.htwk-leipzig.de/waldmann/cm-ws18#hinweise-zur-richtigen-konfiguration-von-a`

(b) SuperDirt installieren

(c) dann SC-Server starten mit

```
sclang superdirt_startup.scd
```

(d) Tidal-Cycles installieren

```
cabal install --lib tidal
```

(e) Tidal-Cycles starten

```
ghci -ghci-script $(ghc-pkg field -f $HOM
```

```
d1 $ s "bd sn"
hush
```

(f) Ü: warum ist das Boot-File kein reguläres Haskell-Modul?

2. Tidal verstehen:

   für den Typ `Pattern a`: Welche Eigenschaften gelten für die Konstruktoren (`pure`,`silence`) und Operatoren (`fmap`,`stack`,`(<*>)`,`(<*)`,`(*>)`,`cat`)

(a) z.B.: Welche sind assoziativ, kommutativ, haben neutrale Elemente; welche Distributivgesetze gelten?
(b) Überprüfen Sie die Axiome von `Functor` und `Applicative`
(c) Eigenschaften von `fast`? (in Beziehung zu anderen) 1.

wenn das erste Argument konstant ist, 2. wenn es ein Muster ist.

vgl. Types in Tidal-Cycles:
`https://www.imn.htwk-leipzig.de/~waldmann/etc/untutorial/tc/`

3. Tidal hören:

(a) Kindohm (Mike Hodnick) `https://github.com/kindohm/365tidalpatterns`

(b) `https://gitlab.imn.htwk-leipzig.de/waldmann/computer-mu/-/tree/master/tidal/code`

4. Tidal benutzen

(a) für einige Audio-Files

(`https://gitlab.imn.htwk-leipzig.de/waldmann/cm-ws18/tree/master/kw49/data`)
den Tidal-Quelltext erraten. Hinweis: benutzt wurden
`s "casio:1"`, `fast`, `speed`, `rev`, `every`, `room`

(b) Steve Reich: *Piano Phase* nachbauen.
Hinweis: chromatische Tonfolgen so möglich:
`s "sine" |+| speed (fmap (\i -> 2**(i/12))`

(c) Antonio Carlos Jobim, Newton Mendonca: *One Note Samba*, Rec. Stan Getz, Charlie Byrd, 1962, LP *Jazz Samba*. Der Stil wurde als *Bossa Nova* bekannt.

i. Welche Rolle spielt der festgehaltene Ton (f) im jeweiligen Akkord? ($D^{-7}D^{\flat 7}C^{-7}B^{7\flat 5}$)

ii. Programmieren Sie den Rhythmus (Stick ab 1:28 min)

Planung der Abschluß-Projekte

- Ziel: Methoden aus der Vorlesung benutzen, um eine musikalische Wirkung zu gestalten. Diese *live* vorführen.
- Ideen für Projekte:

1. (Standard: jedes Vorlesungsthema, siehe auch Übungsaufgaben)
2. Verknüpfung von zwei verschiedenen Themen
3. „Hacks“, d.h., Verwendung einer Methode/eines Werkzeugs zu einem nicht bestimmungsgemäßen Zweck
4. Verknüpfung mit Themen aus anderer Vorlesung, z.B. Robotik
5. Bezug zu Leipzig, z.B.

  - J. S. Bach, H. Riemann, Jutta Hipp `https://www.bluenote.com/artist/jutta-hipp/`
  - Lipsi `https://de.wikipedia.org/wiki/Lipsi_(Tanz)`,
  - Musikautomaten `https://mfm.uni-leipzig.de/dt/dasmuseum/Publik_6onlinepub.php`
- Projekt besteht aus Bericht und Vorführung. Je 2 Personen sollen zusammenarbeiten. Bis KW 51 Gruppen/Themen nennen, bis KW 54 Abstract und Gliederung vorlegen. Zu Vorlesungsende abzugeben sind Bericht (PDF, ca. 10 Seiten) sowie Arbeitsversionen der Quelltexte und Audiodateien. Können bis Vorführung noch überarbeitet werden. Im Bericht sind individuelle Beiträge zu markieren, sonst gemeinsame Note.

- Bewertet werden
  1. Plan: was soll stattfinden, wie soll es wirken?
  2. Inhalt: Bezug zu Themen, Methoden, Werkzeugen aus der Vorlesung, ggf. durch eigene Recherchen ergänzt
  3. Form: wissenschaftliches Schreiben, vgl. Simon Peyton Jones: `https://www.microsoft.com/en-us/research/academic-program/write-great-research-paper/`
  4. Technik/Vorführung: stimmt mit Beschreibung überein, wurde geübt, ohne Verzögerungen präsentiert, Präsentation ist nachvollziehbar (Quelltexte, Befehle, Eingaben/Ausgaben sind live sichtbar)

Vermischtes zu Tidalcycles

Praktisches Live-Coding: Editor, Transitionen

- tidal-Modus für Emacs, andere Editoren ähnlich `https://github.com/tidalcycles/Tidal/blob/main/tidal.el`
  - Ctrl-S: startet ghci, lädt BootTidal.hs (Ctrl-Q: stoppt)
  - Ctrl-Enter: Textblock (enthält Kursorposition, durch Leerzeilen begrenzt) ausführen
- Transitionen

  `https://tidalcycles.org/docs/reference/transitions`

```
d1       $ s "bd sn bd sn*2"
jump 1   $ s "[hc]*3 hc"
xfade 1 $ s "[ho ho*2]*4"
```

Tidalcycles und FRP

- FRP: functional reactive programming

  Conal Elliot, Paul Hudak: *Functional Reactive Animation*, ICFP 1997, `http://conal.net/papers/frp.html`

  Verhalten (Signal): Funktion von Zeitpunkt nach Wert

  Ereignis: Folge von Zeitpunkt $\times$ Wert

- Tidalcycles: *Muster* ist Funktion von Zeit-Intervall nach Tidal-Ereignis-Folge,

  Tidal-*Ereignis* ist Zeit-*Intervall* und Wert

  (tatsächlich zwei Intervalle, *part* und *whole*)

Direkte u. indirekte Konstruktion von Mustern

- direkt: Muster als konkreter Datentyp, Konstruktor

```
Pattern :: (State -> [Event a]) -> Pattern a

pure v = Pattern $ \(State a _) ->
 map (\a'->Event (Context [])(Just a')(sect a a') v
   $ cycleArcsInArc a
```

- indirekt: (Muster als abstrakter D.), API-Operationen

  Bsp: `zoom::(Time,Time)->Pattern a->Pattern a`)
  Implementierung hier auch über API

```
zoomArc (Arc s e) p = let d = e - s in splitQueries
  $ withResultArc (mapCycle ((/d) . subtract s))
  $ withQueryArc (mapCycle ((+s) . (*d))) p
```

- `chacha p = fastcat[ zoom(0,1/4)p, zoom(1/4,2/4)p`
  `, zoom(2/4,3/4)p, fast 2 $ zoom (3/4,1) p ]`

Noten und Skalen

- Instrumente (Software-Synthesizer):

  Parameter `n` ist Index in chromatischer Skale

```
s "superpiano" >| n ("<0 5 7 0>" + "[0,4,7]")
```

- andere Skalen: Umrechnung nach chromatisch, Bsp:

```
  >| n (scale "major" $ "<0 1 7 0>" + "[0,2,4,6]")
```

- Samples: (`n` ist Sample-Nummer), Tonhöhe durch `speed`

  Umrechnung von chromatischem Index i durch $(2^{i/12})$

```
s "trump" >| speed ((2**(1/12))**("<0 5 7 0>" - 12)
```

Pseudo-Zufall

- `irand n` liefert ganze Zahl in $\{0, 1 \ldots, n-1\}$
- `rand` liefert rationale Zahl in Intervall $[0, 1]$
- beides sind stetige Muster, Diskretisierung:

```
.... (scale "major" $ segment 2 $ irand 7)
```

- dieser Pseudozufall ist deterministisch,
  `irand 7 - irand 7` liefert immer 0.

  Entfernen der Korrelation durch

```
irand 7 - slow 1.1 ( irand 7 )
```

Muster von Mustern

- (vorige VL) Kombination von Mustern

  `Pattern (a -> b) -> Pattern a -> Pattern b`

  mit Struktur von: links `<*`, rechts `*>`, beiden `<*>`
- (diese VL) Muster von Mustern

  `join :: Pattern (Pattern a) -> Pattern a`

  Anwendung: `bind p f = join (fmap f p)`

  `bind :: Pattern e -> (e -> Pattern a) -> Pattern a)`
- für Verhalten: $\texttt{join}(f) = t \mapsto f(t)(t)$, (die Diagonale)

  für Muster: unwrap, innerJoin, outerJoin
- squeezeJoin (
- J.W. 2020: `https://www.imn.htwk-leipzig.de/~waldmann/etc/untutorial/tc/monad/`

Ereignisse und Steuerdaten

- Ereignis (Event) enthält Zeit-Intervall und Wert,
- werden intern benutzt (Konstruktion, Kombination, Transformation von Pattern): polymorph im Wert

  und extern (Senden zu Supercollider):

  Wert vom Typ `ControlMap = M.Map Key Value`
- Key ist Bezeichner im SC-Programm für Synthesizer/Sample/Effekt-Parameter,

  ControlMap simuliert Record mit Klang-Steurdaten.
- dabei Klang-Steuerung nur bei -Erzeugung

```
s "sp" + gain "[1 0.8]"
(0>½)|gain: 1.0f, s: "sp" (½>1)|gain: 0.8f, s: "sp"
```

  erzeugt zwei Töne

Steuerbusse (seit Tidal 1.7)

- für jeden Parameter (Bsp: `lpf`) kann man Busse anlegen (muß man selbst numerieren, Bsp: `5`)
- jeder der Bus enthält ein Muster, das Werte des Parameters bestimmt.
-
```
d1 $ (slow 4 $ s "gtr:2" )
  # lpfbus 5 (segment 30 $ range 200 4000 sine)
  # lpq 0.2
```
- die Bus-Ereignisse starten keinen neuen Ton:

```
(0>1/3)-4|^cutoff: 5, cutoff: 400.0f,
          n: 2.0f, resonance: 0.2f, s: "gtr"
```

(ich verstehe nicht, wie das implementiert ist)

Sonifikation von Algorithmen

- Earth to Abigail: (aufgeführt bei *Execute 2021*)

  `https://www.earthtoabigail.com/blog/`

  `audio-representation-bubble-sort-with-ruby-sonicpi`

  zufällige Folge (von Skalentönen) wird geordnet, jeder Element-Vergleich ergibt 1 oder 2 Töne

- ```
  for r in n-2 .. 0 ; for i in 0 .. r
      play arr[i] # Play current value
      if arr[i] > arr[i+1]
        arr[i], arr[i+1] = arr[i+1], arr[i]
        sample :elec_blip2 # Play a sound when a swap
  ```

- Ü: Nach-Implementierung lesen, variieren:

  `https://git.imn.htwk-leipzig.de/waldmann/`

  `computer-mu/-/blob/master/tidal/code/bubble.tidal`
  ```

Aufgaben

1. Suchen Sie (z.B. in Solstice 2021/22)
   Tidalcycles-Programme, die über die einfache Form

```
d1 $ sometimes (jux rev)          -- Modifikat
   $ sometimesBy 0.2 (fast 2)
   $ note "<c d e a b>*3"         -- Noten
   # s "[gtr:1|superpiano]"       -- Instrumen
   # bpf (rangex 40 1000 rand) -- Effekte
```

   hinausgehen durch Verwendung anderer Strukturen (siehe *most functional program*) und weiterer Operatoren (`squeezeJoin` u.ä.)

   Bsp: `https://gitlab.imn.htwk-leipzig.de/`

`waldmann/computer-mu/-/blob/master/tidal/code/dub.tidal`

2. die besondere Rolle der Null in `rangex`: muß das so sein (dann schreiben Sie eine kurze Begründung für die Dokumentation) oder gibt es eine bessere Implementierung?
   `https://club.tidalcycles.org/t/logarithmic-midi-control-in-tidal/4486/4`

Rhythmus, Breaks, Sampling

Motivation, Plan

- technisch (Tidalcycles):
  - Wiederholung: Konstruktion von zeitlicher Struktur parallel (stack), verschränkt (cat)
  - neu: Modifikation von Struktur (every, rotate)
- Ziel, Anwendung: Analyse und Synthese von Rhythmen in der Musik, besonders: Tanzmusik

  was klingt gut (Beispiele) und warum (Gründe)
- was gut klingt, wird zitiert (gesamplet)

Ungerade Rhythmen

- welche Rhythmen kommen in der (europ.) Praxis vor?

  4/4 (Polka, Rock'n'Roll), 3/4 (Walzer), aber nicht nur!

- Zwiefacher

  `https://www.br.de/mediathek/video/woher-kommt-der-zwiefache-verzwickter-tanz-av:584f862a3b467900119cdb27` (6 min: ein Zwiefacher über den Zwiefacher)

- Tanz $=$ rhythmische Bewegung, aber . . .

  `https://www.br.de/mediathek/video/alpha-retro-gestatten-sie-la-bamba-rheinlaender-wig5e848ef5eb0248001cd940da` (24 min 24 s)

Ungerade Rhythmen – Hörbeispiele

- Norma Tanega: *You're Dead*, 1966
- Paul Desmond (rec. Dave Brubeck): *Take Five*, 1959

```
stack [ s "superpiano"
        >| n "[[~ e4'min7] ~ e4'min7 ~ b3'min7 ]"
  , s "gretsch:4" >| gain "1*10"
  , s "sn:4"      >| gain "0 1 1 0 1"
  , s "clubkick"  >| gain "1 0 0 1 0"   ]
```

- gern in Krimiserien:
  - Lalo Shifrin: *Mission Impossible* (Theme) 1966
  - Billy Goldenberg: *Kojak* (Theme) 1973
  - Erich Ferstl: *Alpha Alpha* (Thema) 1972
  - Hartmut Behrsing: *Polizeiruf 110* (Thema) 1972
- im (Post)Rock: Blind Idiot God: *Slackjaw* 1992

Weitere zeitliche Operatoren in Tidal

- schneller (fast), langsamer (slow),

```
fast,slow :: Pattern Time -> Pattern a -> Pattern a
```

  $\mathsf{fast}(f)(s) = \lambda t \mapsto s(f \cdot t)$, $\mathsf{slow}(f)(s) = \mathsf{fast}(1/f)(s)$

- Spiegelung jedes einzelnen Zyklus
  $\mathsf{rev}(s) = \lambda t \mapsto \mathsf{let}\ a = \lfloor t \rfloor; d = t - a; d' = 1 - d\ \mathsf{in}\ s(a + d')$
- verschieben: später `~>`, früher `<~`, Ü: Definitionen?

  Bsp: `"0 0 0 0.1 0" ~> "1 0 0 1 0"`
- Anwendung: `(swingBy 0.2 5 $ gain "1*10")`

```
swingBy x n =
   inside n (withinArc (Arc 0.5 1) (x ~>))
inside n f p = fast n $ f $ slow n p
```

Operatoren zweiter Ordnung

- nur anw., wenn Zyklus-Nummer die Bedingung erfüllt

```
when :: (Int -> Bool)
     -> (Pattern a -> Pattern a)
     ->  Pattern a -> Pattern a
```

- implementiere damit `every  :: Pattern Int`

`->(Pattern a->Pattern a)->Pattern a->Pattern a`

- nur auf den rechten Kanal anwenden

```
jux  :: (Pattern ControlMap -> Pattern ControlMap)
     -> Pattern ControlMap -> Pattern ControlMap
```

- diskutiere: ganz außen `jux rev` — welche Patterns überleben das? (mit 1. ganz einfacher, 2. ganz beliebiger, also gar keiner Struktur)

Synkopen

- Def: unerwartete Verschiebung (der Betonung) von Ereignissen in einem Muster

```
gain "[1 0.7]*4"
gain "[1 0.7 0.7 1 0.7 0.7 1 0.7]"
```

- Konstruktion von synkopischen Mustern:
  - mit den gezeigten Operatoren (deswegen wurden sie definiert)
  - nach anderen mathematischen Prinzipien,
  - (später) systematisch oder zufällig kombiniert (algorithmische Komposition, Aleatorik)
- empirische Analyse von Rhythmen/Synkopen

Clave

- clave son: $[1,0,0,1,\ 0,0,1,0\ ,0,0,1,0,\ 1,0,0,0]$

  erste Hälfte ist `euclid 3 8`,

  Kombination mit zweiter Hälfte durch

```
cat [ "1(3,8)" , "0 1 1 0" ]
```

- rumba clave: $[1,0,0,1,\ 0,0,0,1\ ,0,0,1,0,\ 1,0,0,0]$

  aus clave son durch Verschiebung (`~>`)

  des letzten Viertels der ersten Hälfte (um $1/8$)

```
cat [ "0 0 0 0.125" ~> "1(3,8)" , "0 1 1 0" ]
```

- Frank Manabe, Bob Weiner: *Afro-Cuban Rhythms for Drumset*, 1990.

Syncopation . . . in Groove Music

- Witek et al.: *Syncopation, Body-Movement and Pleasure in Groove Music*, 2014 (Text S2, Figure S6) `https://doi.org/10.1371/journal.pone.0094446`

  exakt beschrieben ist das nicht, aber man kann raten:

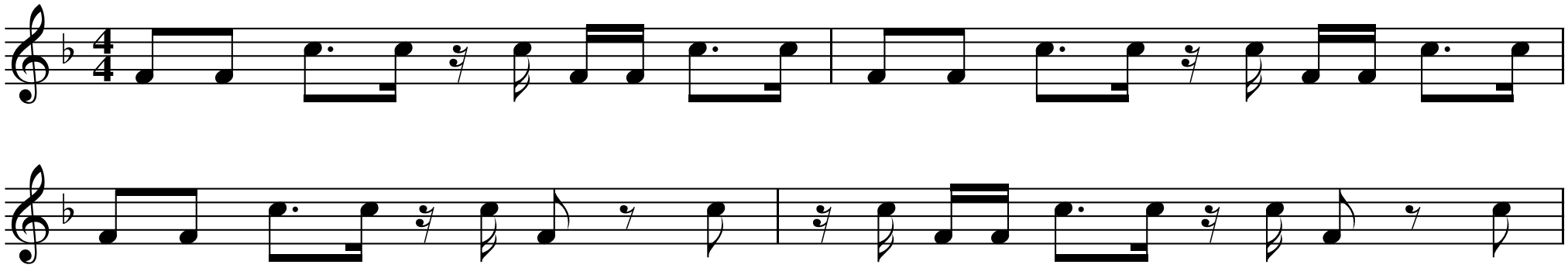
  – Zeitpunkt $e = p/2^k$ hat metrisches Gewicht $w(e) = -k$
  – Synkope $=$ benachbarte e_1 (Note) und e_2 (Pause oder Note auf anderem I.) mit $w(e_1) < w(e_2)$, warum?

- die naheliende Frage ist dann: Muster mit gegebenem Synkopations-Grad automatisch erzeugen

  1. irgendwie, 2. durch (Tidal-)Operatoren

Amen Brother

- The Winstons: *Amen Brother*, 1969.
  (Schlagzeug: Gregory Sylvester Coleman) Break: 1:25



  angeblich „most sampled track in the history of music“
- weitere Beispiele für *break beat*-Drumming
  - The Meters (dr: Ziggy Modeliste), Bsp: *Cissy Strut* 1969
  - Booker T and the MGs (dr: Al Jackson Jr), Bsp: *Melting Pot* 1971

Überblick Sampling

- Def: Sample $=$ Wellenform eines Audio-Signals
- Anwendung 1: Optimierung der Synthese
  - Klänge im Voraus synthetisieren, abspeichern,
  - wenn Software-Simulation des physikalischen Systems nicht in Echtzeit möglich ist
  - Bsp: Synclavier (1977, 32 MB Speicher) `http://www.vintagesynth.com/misc/synclav.php`
- Anwendung 2: musikalische Aussage
  Audio-Signal wird aus erkennbarer Quelle *zitiert*:
  - einzelner Klang (eines bestimmten Instrumentes)
  - *zusammenhängende* Klänge (Teil eines Musikstückes)

Samples in Tidal-Cycles

- Sample als Audio-Datei (wav), wird bei Start von Supercollider geladen, siehe auch `https://tidalcycles.org/docs/reference/sampling`
- `d1 $ s "breaks152"` spielt das Sample ab
  - in Original-Länge l und -Tempo
  - beginnend zu jeder (!) Tidal-Periode p

  d.h. auch selbst-überlappend, falls $l > p$
- ```
  stack [ fast 4 $ s "hc"
    , s "breaks152" # begin 0 # end 0.3 # speed 0.9 ]
  ```

  begin und end sind relativ zu l,

  fast und speed sind unabhängig
- Verarbeitung mit `chop`, `slice`, `splice`, `striate`
  ```

Beispiel Sample-Verarbeitung (1)

```
let frev n = rev . fast n . rev . slow n
in  d1 $ stack
  [ "<4 2 4 8>*2" >>= \ k ->
      sometimesBy 0.3 (frev (pure k))
    $ chop 16 $ sound "breaks125:0"
  ,  "<4 2 8>" >>= \ k ->
      sometimesBy 0.5 (frev (pure k))
    $ chop  8 $ sound "breaks125:1"
  , s "clubkick:1(3,8)" # gain 0.9
  , s "jungbass:7(1,8)"
  ]
```

- wie wirkt `frev n` einzeln?
- wie wirkt `(>>=)`?

Übung

1. weitere Beispiele für ungerade Takte in der Pop/Rockmusik analysieren und finden
   - Soizweger Zwoagsang *Ja wer koan Zwiefachn ka*, 2015. `https://www.br.de/mediathek/video/soizweger-zwoagsang-ja-wer-koan-zwiefachn-5a3c777c185c080018d22d08`
   - $11/4$-Takt: Go-Betweens: *Cattle and Cane*, 1983.
   - längere Perioden?
2. programmieren Sie ein Thema im Stil des *Salegy*, eines Musikstils aus Madagaskar, dessen rhythmische Betonung zwischen gerade und ungerade changiert, $12/8 = 3 \cdot (4/8) = 4 \cdot (3/8)$

Hörbeispiele: `https://www.nts.live/shows/nts-10-matt-groening/episodes/henry-kaiser-23rd-april-2021`

3. Das Bjorklund-Verfahren, siehe `https://hackage.haskell.org/package/tidal-1.6.1/docs/src/Sound.Tidal.Bjorklund.html` und Papers von G. Toussaint.

   Beispiel $E(5, 8)$.

   Begründen Sie, daß die Implementierung die Spezifikation (im Skript) erfüllt.

4. bestätigen Sie die angegebenen Vorkommen von $E(k, n)$ z.B. bei lateinamerikanischen Rhythmen.

   Hörbeispiele Bossa Nova:

- Stan Getz/Joao Gilberto 1964,
- Quincy Jones: Big Band Bossa Nova 1962;
- Senor Coconut (Uwe Schmidt, Atom TM): El Baile Aleman, 2000.

5. Arbeiten mit (eigenen) Samples in Tidal, vgl. `https://tidalcycles.org/docs/configuration/AudioSamples/audiosamples/` experimentieren Sie mit `breaks165`, `bev` (wie angegeben), `led` (schwierig)

6. implementieren Sie Ideen aus: Nick Collins: *Algorithmic Composition Methods for Breakbeat Science*, 2001 `https://composerprogrammer.com/research/acmethodsforbbsci.pdf`

7. autotool-Aufgaben zu Euklidischen Rhythmen (gleichmäßige Verteilung von Ereignissen/Zahlen in einem Raster).

   Entwickeln Sie eine Theorie für den zweidimensionalen Fall, vgl. `https://gitlab.imn.htwk-leipzig.de/autotool/all0/issues/562`

Algorithmische Komposition

Motivation

- klassische Partitur beschreibt das Musikstück *extensional* (durch Angabe der zu spielenden Töne)
- jetzt: *intensional* (durch Angabe einer Vorschrift (Algorithmus) zur Bestimmung der zu spielenden Töne)
- z.B. algebraische Ausdrücke (par, seq)
- jetzt auch: *randomisierte* Algorithmen zur Komposition
- Aufführung durch Maschinen *oder Menschen*
- Quelle (Übersicht): Gerhard Nierhaus: *Algorithmic Composition*, Springer 2009,

  `https://gerhardnierhaus.com/books-on-ac`

Geschichte der Alg. Komposition (Beispiele)

- mit Würfeln und Tabellen:
  - Johann Philipp Kirnberger: *Der allzeit fertige Menuetten- und Polonaisen-Komponist*, 1757
  - Carl Philipp Emanuel Bach: *Einfall einen doppelten Contrapunct in der Oktave von sechs Tacten zu machen ohne die Regeln davon zu wissen*, 1758
- mit Rechenmaschinen
  - Lejaren Hiller, Loenard Isaacson: *Illiac Suite*, 1955
- das Ziel ist hier immer die Nachahmung bekannter Musikstile

Geschichte der Alg. Komposition

- hier geht es um wirklich neue Musik:
- Iannis Xenakis: `https://www.iannis-xenakis.org/`

  Metastasis, 1955; Buch *Formalized Music — Thought and Mathematics in Music*, 1963,
- Gottfried Michael Koenig (1926–2021)

  `https://www.koenigproject.nl/`, *Projekt 1* 1964, *Projekt 2* 1966, *Sound Synthesis Program* 1971

  RU-Prinzip: . . . Unwiederholbarkeit von Reihenelementen („unregelmäßig“) einerseits und gruppenbildende Multiplikations-reihen („regelmäßig“) andererseits.

  Nachruf: `https://www.concertzender.nl/in-memoriam-gottfried-michael-koenig-2/` (Roland Kuit)

Komposition und Constraints

- die Kompositions-Aufgabe: bestimme eine (bestmögliche) Partitur, die diese Bedingungen (Constraints) erfüllt:
  - Randbedingungen
    (Anzahl Stimmen, Tonart, Metrum, Anzahl Takte)
  - musikalische Regeln (keine Dissonanzen, Parallelen)
  - ggf. Ähnlichkeit zu Vorlagen
    (Bsp: eine Fuge im Stil von Bach)
  - ggf. Vermeidung der Ähnlichkeit zu Vorlagen
    (Bsp: eine Fuge, aber anders als die vorige)
- maschinelle Lösung dieser Aufgabe durch
  - exakte Verfahren (vgl. VL Constraint-Programmierung)
  - statistische Näherungsverfahren (sog. maschinelles Lernen)

Modelle für musikalische Eigenschaften

- Constraint: Häufigkeiten aufeinanderfolgender Töne

  Modell: stochastischer endlicher Automat (Markov-Prozeß)

- Constraint: Häufigkeiten globaler Strukturelemente

  Modell: stochastische generative Grammatik

- Constraint: spannendes Verhältnis zwischen mehreren Stimmen

  Modell: Zweipersonenspiel

- Constraint: Regelmäßigkeit ohne Wiederholungen

  Modell: zellulärer Automat, Lindenmayer-System

Algorithmische Komposition und Kreativität?

- wenn die Kompositionsarbeit scheinbar durch einen Computer übernommen wird — welche Rolle haben: der Komponist? der Interpret? der Hörer?
- der kreative Vorgang ist: Komponist schreibt das Programm (wenigstens: wählt Programme aus und stellt die Parameter ein)
- bei *live coding* ist das ein zentraler Aspekt (Publikum sieht den Bildschirm des Komponisten)
- vgl. aber Joseph Schillinger: *The Mathematical Basis of the Arts*, 1943. (S. 17: fünf Erscheinungsformen der Künste) `https://archive.org/details/TheMathematicalBasisOfTheArtsJosephSchillin`

Pseudozufall in Tidalcycles

- Pseudozufallsgröße im Intervall $[0, 1]$

```
rand :: Fractional a => Pattern a
rand = Pattern (\(State a@(Arc s e) _) ->
 [Event (Context []) Nothing a
  (realToFrac $ (timeToRand ((e + s)/2) :: Double))
```

- Transformation auf Intervall $[l, r]$ durch `range l r rand`
- ganze Zahlen $[0, 1 \ldots m - 1]$ durch `irand m`
- Auswahl aus einer Liste (mit möglicher Implementierung)

```
choose :: [a] -> Pattern a
choose xs = (xs !!) <$> irand (length xs)
```

Stetige und diskrete Muster

- `rand`, `irand` und daraus konstruierte (`choose`) Muster sind stetig (continuous): haben für jeden (reellen) Zeitpunkt einen Wert,
- `ControlPattern` muß immer diskret sein,

  denn OSC-Nachrichten sind diskret.

  Diskretisierung durch: `segment 4 rand`
- pseudo-zufällige Melodie (links z.B. `s "superpiano"`)

```
... >| n (segment 4 $ irand 12)
... >| n (scale "major" $ segment 4 $ irand 7)
```

- pseudo-zufällige Akkordfolge (mit falscher Stimmführung)

```
... >| n (scale "major" $ segment 4 $ irand 7)
    |+ n "[0,2,4]"
```

Kampf dem Determinisimus

- der „zufällige Wert“ ist Funktionswert des Mittelpunktes des abgefragten Intervalls:

  gleiche Fragen → gleiche Antworten

```
queryArc
  (stack [segment 2 $ irand 8, segment 2 $ irand 8]
  (Arc 0 1)
[[](0>½)|5,[](½>1)|1,[](0>½)|5,[](½>1)|1]
```

- zur parallelen Komposition unabhängiger Stimmen: die Zeit für den Generator lokal verschieben

```
stack [ segment 2 $ irand 8
      , segment 2 $ slow 1.1 $ irand 8 ]
```

  Ü: `slow 1.1` ist hier besser als `rotR 1`, warum?

Beispiel-Komposition mit vielen irand

- mehrere gleichartige Stimmen
  (`m i` ist Note i in diat. Skala)

```
stack [ m ( slow 4.5 $ segment 1 $ irand 10)
          |>| ( (1/8) ~> gain "[1 0.8 ]*2" )
   , ... ]
```

- `https://git.imn.htwk-leipzig.de/waldmann/computer-mu/-/blob/master/tidal/code/piano.tidal`
- ist sehr einfache Anwendung von Koenigs Prinzipien:
  U(nregelmäßig): `irand`, R(egelmäßig): `[_ _]*2`
- ist *zu einfach*, globale Struktur ist starr (Skala, Rhythmus), Stimmen sind nicht weiter korreliert
  (weder mit eigener Vergangenheit noch untereinander)

Stochastische Sprachen

- (klassische) Sprache über Alphabet Σ ist Abbildung $L : \Sigma^* \to \{0, 1\}$ (die Zweiermenge)
- stochastische Sprache über Σ ist Abbildung $L : \Sigma^* \to [0, 1]$ (das Intervall reeller Zahlen)

  $L(w) \approx$ die Wahrscheinlichkeit, mit der $w \in L$
- Plan: stochastische Sprache

  für $\Sigma =$ Elementar-Ereignisse (z.B. Noten)

  – so definieren, daß interessante $w \in \Sigma^*$ hohe Wahrscheinlichkeit haben
  – durch endliches Objekt (Automat, Grammatik) repräsentieren

Stochastische Automaten, Markov-Prozesse

- ein endlicher stochastischer Automat A besteht aus:
  - Zustandsmenge Q
  - Initialvektor $I \in (Q \to [0,1])$,
  - Transitionsmatrix $T \in (Q \times Q \to [0,1])$.

  wobei I stochastischer Vektor ($\sum_{q \in Q} I(q) = 1$)

  und T stoch. Matrix (jede Zeile ist stoch. Vektor)
- stochastischer Prozeß erzeugt Wort $w = q_0 q_1 \ldots q_n \in Q^n$:

  wähle $q_0 \in Q$ nach Verteilung I,

  wähle $q_{k+1} \in Q$ nach Verteilung $T(q_k)$.
- Anwendungen in der Musik: $Q =$ Noten, $Q =$ Akkorde.
- dabei wird aber die globale Struktur (nach Riemann: die Funktion der Akkorde) ignoriert!

Markov-Prozesse in Tidalcycles

- ```
  markovPat :: Pattern Int -- Länge
    -> Pattern Int         -- Start
    -> [[Double]]          -- Übergangsmatrix
    -> Pattern Int
  ```
- ```
  notes = [0 .. 21]
  note_prob n = if 0 == mod n 7 then 5 else 1
  jump_prob ps x y = (ps  ++ repeat 0) !! abs (x-y)
  transitions ps = [ [ jump_prob ps x y * note_prob y
     | y <- notes] | x <- notes]
  period = 128
  mel ps = n $ scale "major" $ slow period
    $ markovPat period 0 $ transitions ps
  mel [ 1, 10, 10,  2,  2 ]
  ```
- https://git.imn.htwk-leipzig.de/waldmann/computer-mu/-/blob/master/tidal/code/markov.tidal

Markov-Prozesse mit verstecktem Zustand

- MP wie bisher:
  – Zustandsmenge Q, Initalvektor I, Transitionsmatrix T beschreibt Fkt. $Q^* \to [0,1]$ (Verteilung auf jedem Q^k)
- jetzt erweitert: Ausgabe-Alphabet Σ, Ausgabe-Matrix A: für jedes $q \in Q$ eine Verteilung $A(q)$ auf Σ
- Bsp: $Q =$ Akkorde, $\Sigma =$ Noten, $A(q)$ ist Verteilung auf Tönen des Akkordes q (wählt jeweils einen Ton aus)
- beschreibt Fkt $\Sigma^* \to [0,1]$ (Verteilung auf jedem Σ^k)
- Ausgabefolge $\in \Sigma^*$ wird beobachtet, Zustandsfolge $\in Q^*$ ist unsichtbar: *hidden Markov model*
- in Q kann (flache) globale Struktur versteckt werden,
- für baumartige Struktur sind endliche Q nicht ausreichend

Die Wiederholung des Zufalls?

- eine zufällige Tonfolge

```
note (scale "major" $ segment 8 $ irand 7)
  # s "superpiano"
```

- wie wiederholt man solche (Teil)folgen?

```
cat $ replicate 4 $ segment 8 $ irand 7
```

- graduelles Umschalten zwischen solchen Folgen: `https://git.imn.htwk-leipzig.de/waldmann/computer-mu/-/blob/master/tidal/code/random-not-random.tidal`

Übungen

1. Algorithmische Komposition in Euterpea,

   Beispiel: `https://www.donyaquick.com/interesting-music-in-four-lines-of-code/`.

2. Stochastische Musik in Euterpea

   `https://git.imn.htwk-leipzig.de/waldmann/cm-ws18/blob/master/kw48/stoch.hs`

   zufällige Permutation: Implementierung vervollständigen

3. G. Koenig: Projekt 1, 1964 (Reiner Wehinger: PR1-SC Re-Implementierung in Supercollider, 2016) ausprobieren

   `https://koenigproject.nl/project-1-files-download/`

4. Funktionen für Markov-Ketten in Tidalcycles

https://github.com/tidalcycles/Tidal/blob/main/src/Sound/Tidal/UI.hs#L1126

Analysieren Sie diese (oder andere) Anwendungen, modifizieren Sie:

https://git.imn.htwk-leipzig.de/waldmann/computer-mu/-/tree/master/tidal/code markov, markov-chords

Partituren: Text und Grafik

Definition, Motivation

- Partitur: schriftliche Fixierung von Musik zum Zweck der Archivierung und späteren Aufführung
- soll enthalten und deutlich machen:
  - wesentliche Details (z.B. Melodie, Noten), nicht alle
  - globale Struktur (Gruppierung, Hierarchie von Details)
- Partitur ist Repräsentation des abstrakten Syntaxbaumes (AST), dieser wird rekonstruiert und dann interpretiert.
- wesentliche Operatoren im AST: sequentielle und parallele Komposition, Wiederholung, Verzweigung
- die Verarbeitungskapazität (Stack-Tiefe) des Menschen ist aber beschränkt $\Rightarrow$ Höhe des AST ist klein ($\leqslant 5$?)

Algebraische Notation (Wdhlg.)

- Def: textuelle Repräsentation des AST (z.B.: Knoten-Namen als Wörter, Teilbäume in Klammern)

  Bsp: der Datentyp `Music` in Euterpea, `TPat` in Tidal, Mini-Notation in Tidal

- eingebettete DSL (Euterpea, Tidal): man sieht/bearbeitet nicht den AST der Daten, sondern den AST eines Programmes der Gastsprache, das die Daten erzeugt.

- auch nicht eingebettete Musik-DSLs erfinden dann das Fahrrad neu (Namen, Blöcke/Sichtbarkeit, UP $=$ Namen mit Parametern, Typen) (*doomed to re-invent . . . poorly!*)

- maschinennahe (flache) ASTs: der Event-Stream in Tidal (OSC-Nachrichten), MIDI-Stream- und Dateiformat.

Grafische Partituren, Notensatz

- zweidimensionale Notation:
  - für jede einzelne Stimme (Notensystem)
    - Ordinate (nach oben) (Notenlinie): Tonhöhe
    - Abszisse (nach rechts): Zeit

    Zeit auch durch Form der Noten (Köpfe, Hälse)
  - für mehrere Stimmen:
    - Notensysteme übereinander bedeutet Gleichzeitigkeit
- Herstellung solcher Partitur:
  - *grafisch*: WYSIWIG
  - *programmatisch*: textuelle Repr. des AST editieren, der die Form abstrakt beschreibt, dann maschinell in konkrete Form übersetzen (kompilieren)
- auch Export und Import des AST aus anderen Sprachen

Einige Aspekte von Partituren

- Horizontalstruktur (Zeit): optische Position (im Takt) zeigt zeitliche Position, Leerplatz nach Note entspricht deren Länge, alle Takte gleich breit, Zeilenumbruch nur bei Takt-Ende, alle Zeilen gleichviele Takte
- Vertikalstruktur: gleichzeitige Ereignisse in verschiedenen Stimmen stehen übereinander (Akkordname, Melodienote, Textsilbe, Baßnote)
- der zweckmäßige und gefällige Notensatz ist eine Kunstform (genau wie der Schriftsatz)

  `https://imslp.org/wiki/Repository_of_music-notation_mistakes_(Coulon,_Jean-Pierre)`

  und dessen Automatisierung ebenso

Partitur-Publikationen

- klassische Leipziger Musikverlage:

  Edition Peters, Hofmeister
- Beck: Song Reader (2012)

Weitere Notenbezeichnungssysteme

- . . . für die Diatonik (Ganztonreihe)

  lateinische Zahlwörter (Grundton 1, Sekunde 2, Terz 3, Quarte 4, Quinte 5, Sexte 6, Septime 7, Oktave 8)

- (verschobenes) Alphabet $1 = c, d, e, f, g, a, b, 8 = c'$
- Tonsilben (Solfeggio): 1=*do, re, mi, fa, so, la, ti*

  vgl. Hawkwind: *Doremi Fasol Latido* (1972)

- klassische hindustanische Musik: Silben für:

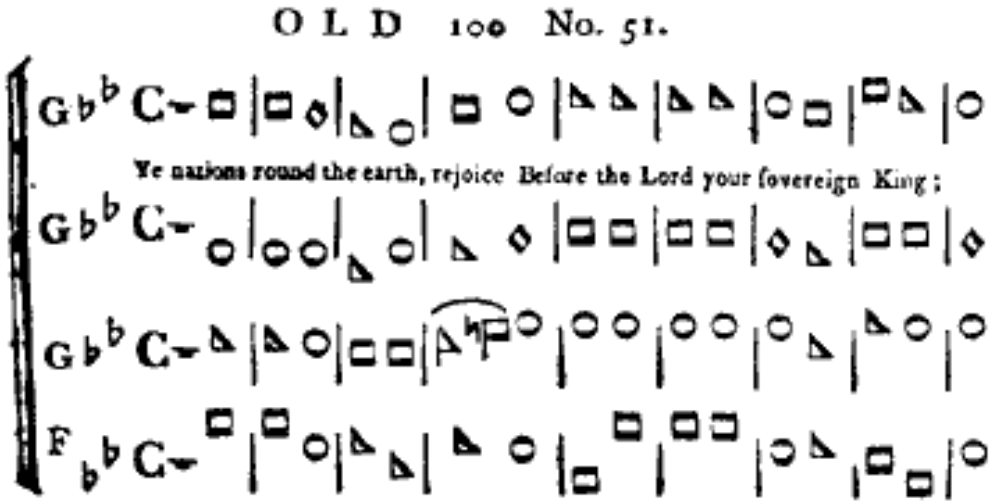
  Tonhöhen (ähnlich Solefeggio) *sa, re, ga, ma, pa, dha, ni*

  Tabla-Klänge: *dhaa, ga, ge, gi, ka, ke, dhi, dhin, tin, tun, tit, ti, te, Ta, tr, naa, ne, re, kat, taa, dhaage, tiTa, tirikiTa*

  `https://raag-hindustani.com/Rhythm.html#`

Shape Notes

- Tonhöhe = Notenkopfform (Bsp. Andrew Law, 1803)



- Anwendung: *Sacred Harp* (B.F. White, E. J. King, 1844)

  vgl. Warren Steel, 2003,

  `https://web.archive.org/web/20031206091941/http://www.mcsr.olemiss.edu/~mudws/harp.html`

- Hörbeispiel: Cordelia's Dad: *Idumena*, 1996

Notensatz mit Lilypond

- kompiliert algebraische Musik-Beschreibung (Programmtext) zu grafischer Darstellung (PDF) sowie zu MIDI-Datenstrom
- Warum? `http://lilypond.org/essay.html`
- Wie? `http://lilypond.org/doc/v2.19/Documentation/contributor-big-page.html#overview-of-lilypond-architecture`
  - Implementierung in C++
  - Anwender-definierte Erweiterungen in LISP (Scheme)
  - Backend benutzt Metafont `https://web.archive.org/web/20110927042453/http://www.tex.ac.uk/ctan/systems/knuth/dist/mf/mf.web`

Sematik von Lilypond (lokal)

- ```
  \version "2.18.2" \header { } \score {
    \relative c' { c4 d e f | g a b c }
    \layout { }  \midi { }
  }
  ```
- lokale Struktur (Folge von Noten einer Stimme)
  - Einzelnote (Bsp: `c''4`): Name, Oktave, Zeit
  - fall Zeit nicht angegeben, dann vorigen Wert benutzen
    bei Taktstrich: Zeit muß Vielfaches des Taktes sein
  - relative Notation: immer die nächstliegende Oktave

  damit muß man für Melodien *wesentlich* weniger schreiben als bei Euterpea (u.ä.)
- dafür zahlt man den Preis der Kontextabhängigkeit.

  das geht in Tidal (Haskell) nicht—aber in Mini-Notation?
  ```

Semantik von Lilypond (global)

- Zusammensetzung von Teil-Partituren:
  - sequentiell: hintereinander
  - parallel: Operator `<< ... >>`

```
\relative c' {c4 d e <<f a c>> | <<g b d>> a b c}
```

- Notensysteme (für mehrstimmigen Satz)

```
<< \new Staff{\relative c' { c4 d e f | g a b c }}
  \new Staff {\relative c''{ g  a g f | e d d c }}
>>
```

- Wiederholungen:
  - notiert (Wiederholungszeichen in Partitur):

```
repeat volta 2 {c1}
```

  - expandiert: `repeat unfold 2 {c1}`

Semantik von Lilypond: Unterprogramme

- Namen zur Bezeichnung von Teilpartituren

```
foo = \relative c' { c4 d e f | g a b c }
\score { << \new Staff { \foo }
            \new Staff { \transpose c e \foo }
         >> }
```

- vergleichbar zu Unterprogrammen, *aber*
  - Unterprogramme haben keine Argumente
  - Unterprogramme sind global (außerhalb `\score{..}`)
  - es gibt keine Bedingungen und Verzweigungen

  Das ist ein riesengroßer Rückschritt im Vergleich zu Haskore u.ä.
- Work-around: man kann LISP-Code schreiben

Semantik von Lilypond (Kontexte)

- `\new Staff`, `\new Voice`, ... legt *Kontexte* an,
- zu jedem Kontext gehören mehrere *engraver*,

  diese verarbeiten *musical expression* zu grafischen Elementen: `Clef_engraver`, `Key_engraver`, `Note_heads_engraver`, ...
- benannter Kontext kann „von außen“ benutzt werden

```
<< \new Staff { \new Voice = "foo" {} }
   \new Staff { \new Voice \relative c' {c4 d e f}
   \context Voice = "foo" \relative c'{  g a b c }
>>
```

- Anwendung: Text (Silben) unter Melodie (Noten)

Anwendung von Kontexten

- Text (Silben) unter Melodie (Noten)
- ```
  << \new Staff { \new Voice = "foo" {} }
     \new Lyrics = "here"
     \context Voice = "foo"
        \relative c' { g a b c }
     \context Lyrics = "here" {
       \lyricsto "foo" { fas- ci- na- ting }
     } >>
  ```
- spezifiziert wird:
  - wo der Text erscheint: `\new Lyrics`
  - nach welcher Stimme er ausgerichtet wird: `\lyricsto`
  ```

Weitere textuelle Partitur-Formate

- ABC Notation `https://abcwiki.org/abc:syntax`

  sehr kompakte Notation, praktisch für einfache Stücke

  Bsp: Quelltext für C-A-F-F-E-E

- `https://www.ctan.org/pkg/musixtex` (Daniel Taupin, Andreas Egler, ...) eDSL für TEX

  ideal für Notensatz, ungeeignet für Austausch

- MusicXML `https://w3c.github.io/musicxml/tutorial/structure-of-musicxml-files/` ist Austausch-Format

  XML represents data in a hierarchy, but musical scores are more like a lattice. How do we reconcile this?

Übersetzung zwischen Partitur-Formaten

- (kommerzielle Anbieter wollen nur Import, nie Export)
- Hans Hoglund: Music Suite (z.B. FARM 2014) `https://music-suite.github.io/docs/ref/`

  . . . is a language for describing music, based on Haskell

  (NB: also eine Instanz von `https://xkcd.com/927/`)
- The idea of defining a custom internal representation, but relying on standardized formats for input and output is influenced by Pandoc (`https://pandoc.org/`, John MacFarlane) (das kann z.B. LaTeX↔ Markdown ↔ HTML)
- Lilypond-Import für Music Suite? `https://github.com/jwaldmann/lilypond-parse#goals`

Übung

1. Idumena (shape notes) Aufnahme und Partitur vergleichen (beide Gesangsstimmen zuordnen)
2. Lilypond (Beispiele siehe Repo)
   (a) `lilypond basic.ly;evince basic.pdf;timidity basic.`
   (b) *Chase the Devil* analysieren, modifizieren, ergänzen,
   (c) Schlagzeug hinzufügen

   ```
   \new DrumStaff { \drummode { bd4 bd } }
   ```
3. Versionen der Partitur Telemann vergleichen (gedruckt, lilypond), Erfüllung der Kriterien aus Coulon: *Music Notation Mistakes* überprüfen.
4. (Programm)ablaufsteuerung in Partituren: Beschreiben Sie die Semantik von

(a) Wiederholung mit Variantenteil
(b) dal segno als fine

durch

(a) goto-Programm
(b) strukturiertes Programm (AST, algebraische Operatoren, für Euterpea, für Tidal)
(c) Vergleichen Sie Ihren AST-Entwurf mit der Syntax (`alternative`) in Lilypond.

5. Music Suite, andere Formate:

   Programm installieren, Importe und Exporte ausprobieren, auch round-trip.

   Lilypond-Import: Masterarbeit! Bestätigen Sie die in der o.g. zitierten Quelle genannten Probleme durch Blick in den Quelltext des Lilypond-Parsers.

Algorithmische Komposition (II)

Motivation/Wiederholung

- akustisches Signal: flach (Folge von Geräuschen/Tönen), abstrahiert als Folge v. Noten (z.B. MIDI-Ereignissen)
- ist entstanden aus strukturierter Idee der Komposition (Sonatenhaupsatzform, Kadenz, Melodie, Leitmotiv)
- diese Struktur erkennen (Analyse, Parsing), abstrahieren (komprimieren), zur Synthese benutzen
- Vorlage ist Beschreibung natürlicher Sprachen (und nachgebildet: Programmiersprachen) durch
  - Lexik (Wörter, als Buchstabenfolgen): flach
  - Syntax (Gruppierung, nach semantischer Funktion): strukturiert (Baum)

Flache Statistische Analyse und Synthese

- Markov-Automat (endl. Zustandsmenge, stoch. Ü-matrix)
- wenn Zustand $=$ Note, dann Gedächtnis der Länge 1 (Verteilung der Note n_t hängt von n_{t-1} ab)
- Verallgemeinerung: Gedächtnis (Kontext) der Länge k (Verteilung der Note n_t hängt von $n_{t-k}, \ldots, n_{t-1}$ ab)
- Analyse: Verteilungen (bedingte Wsk.n) durch Auszählen aller Teilwörter der Länge $k+1$ bestimmen
- dünne (sparse) Repräsentation der Ü-Matrix

```
type Distribution e = M.Map e Int
type Histogram e = M.Map [e] (Distribution e)
```

- Erweiterung: Ausdünnung des Kontextes, z.B. n_t hängt von $n_{t-1}, n_{t-2}, n_{t-4}, n_{t-8}$ ab (entspricht *attention heads* in *transformer* NN)

Generative Theorie der Tonalen Musik (GTTM)

- Fritz Lerdahl, Ray Jackendoff: *A Generative Theory of Tonal Music*, MIT Press, 1983
  - Ziel: formal description of musical intuition of a listener who is experienced in a musical idiom
  - Methode: Konstruktion von (Ableitungs)bäumen
  - Realisierung: well-formedness rules, preference rules f.:
    - Gruppierung, Betonung (metrische Struktur)
    - time-span reduction (Abstraktion)
    - prolongotional reduction (Spannung, Auflösung)
- vgl. Hansen: *The Legacy of GTTM*, `http://www.dym.dk/dym_pdf_files/volume_38/volume_38_033_055.pdf` 2010
- math. Modell: stochastische kontextfreie Grammatik (äquivalent: stochastischer Baum-Automat)

Stochastische Baum-Automaten

- bisher: Wort-Automat (über Alphabet Σ, z.B. Noten)
  - klassisch (DFA) beschreibt Menge v. Wörtern $\subseteq \Sigma^*$
  - stochastisch (Markov) Verteilung $\Sigma^* \to [0 \ldots 1]$
- jetzt: Baum-Automat, über Signatur Σ,

  z.B. $\{$ silence, pure _, fast _, cat, stack $\}$
  - klassisch: Transitionen $v \to f(v_1, \ldots, v_k)$ für $f \in \Sigma_k$,
    solcher Automat beschreibt Menge v. Bäumen (Termen)
    Bsp: $S \to f(S,S), S \to g(S,S), S \to h()$:
    alle Binärbäume (innere Knoten f oder g, Blätter h)
  - stochastisch: Transitionen mit Gewichten,
    Bsp: $S \to [(1/4, f(S,S)), (1/2, g(S,S)), (1/4, h())]$
    solcher Automat beschreibt Verteilung auf Bäumen,

Generierung interessanter Bäume

- nach vorgegebener (stochastischer) CFG, Bsp.
  Stück → Tonika Dom. Ton., Dom. → Subdom. Dom., . . .
- Regeln siehe z.B.: Donya Quick, Paul Hudak: *Grammar-Based Automated Music Composition in Haskell* `https://functional-art.org/2013/quick.pdf`
- die Größe der zu erzeugenden Bäume berücksichtigen!
  ist nicht trivial: bei $S \to [(3/4, f(S,T)), (1/4, g(T))]$
  Entscheidung für erste Regel nicht unbedingt mit $3/4$,
  Welche Größen für linken/rechten Teilbaum von f?
- vgl. Größen-Parameter für Quickcheck (Claessen, Hughes, ICFP 2000) `http://www.cs.tufts.edu/~nr/cs257/archive/john-hughes/quick.pdf`

Bestimmung harmonischer Strukturen

- José Pedro Magalhães, W. Bas de Haas: *Functional Modelling of Musical Harmony*, ICFP 2011

  `https://github.com/haas/harmtrace`

  „Given a sequence of chord labels, the harmonic function of a chord in its tonal context is automatically derived.“
- Spektral-Analyse → chord labels, diese jedoch fehlerbehaftet. Wie reparieren?
- benutze Baumstruktur, beschrieben durch Grammatik und fehlerkorrigierenden Parser (Swierstra 2009)

  `https://hackage.haskell.org/package/uu-parsinglib`
- `harmtrace parse -g pop -c "C:maj C:maj  F:maj G:maj`

Grammatik-Inferenz (einfachste Form)

- Larson, Moffat: *Offline dictionary-based compression*, `https://doi.org/10.1109/DCC.1999.755679`

  Algorithmus Re-Pair: ein häufigstes Paar benachbarter Buchstaben wird durch einen neuen Buchst. ersetzt.

  repair$(w) =$ eine *singleton CFG* (kontextfreie Gr. G für eine einelementige Sprache) mit Lang(G) $= \{w\}$

- Anwendungen:
  - (zitierter Artikel) zur verlustfreien Kompression
  - (?) als Grundlage für Inferenz anderer CFG
  - als Vorverarbeitung für andere stochastische Analyse-Verfahren, Bsp: Sprachmodelle: nicht auf Buchstaben, sondern auf Gruppen (Silben)

Übungen

1. Algorithmische Komposition in Euterpea,

   Beispiel: `https://www.donyaquick.com/interesting-music-in-four-lines-of-code/`.

2. Stochastische Musik in Euterpea

   `https://git.imn.htwk-leipzig.de/waldmann/cm-ws18/blob/master/kw48/stoch.hs`

   zufällige Permutation: Implementierung vervollständigen

3. G. Koenig: Projekt 1, 1964 (Reiner Wehinger: PR1-SC Re-Implementierung in Supercollider, 2016) ausprobieren

   `https://koenigproject.nl/project-1-files-download/`

4. deterministische nichtperiodische Musik

benutzen Sie https://git.imn.htwk-leipzig.de/waldmann/cm-ws18/blob/master/kw48/dnp.hs

5. Funktionen für Markov-Ketten, L-Systeme in Tidalcycles
   https://github.com/tidalcycles/Tidal/blob/main/src/Sound/Tidal/UI.hs#L1126

   Analysieren Sie diese (oder andere) Anwendungen, modifizieren Sie:
   https://git.imn.htwk-leipzig.de/waldmann/computer-mu/-/tree/master/tidal/code markov, markov-chords

6. Vuza-Kanons
   https://git.imn.htwk-leipzig.de/waldmann/computer-mu/-/blob/master/vuza/Canon.hs

Zusammenfassung, Ausblick

- Inhalt der VL, stark komprimiert:
  - algebraische Beschreibung
  - von Klängen (Bsp: csound-expression) und
  - musikalischen Strukturen (B: Euterpea, Tidalcycles)

  dazu Grundlagen (math. Modelle) und ihre Geschichte
- Projekte (Abgabe, Begutachtung, Überarbeitung, Präsentation) nach Plan `https://git.imn.htwk-leipzig.de/waldmann/cm-ws23/-/issues/27`
- ich betreue gerne Abschlußarbeiten zur Musikinformatik, insb. Dokumentation, Reparatur, Erweiterung von open-source Comptermusiksystemen (z.B. siehe Issue-Tracker von tidalcycles, csound-expression)

Plan

- KW 42 Einleitung
- KW 43 Geräusch und Klang
- KW 44 (Spektral)Analyse von Klängen
- KW 45 Elektrische Oszillatoren und Filter
- KW 46 Spannungs-gesteuerte Osz. und Filter
- KW 47 Programme für Klänge
- KW 48 Töne (Skalen), Harmonien
- KW 49 (Algebraische) Komposition
- KW 50 Performing with Patterns of Time
- KW 51 Kombination von Mustern d. Fkt. höh. Ordnung
  bis KW 51: Anmeldung der Abschlußprojekte
- KW 54 Rhythmus, Breaks
- KW 55 Algorithmische Komposition (lokal)
- KW 56 Notensatz
- KW 57 Algorithmische Komposition u. Analyse (global)