Symbolsk komponering

Anders Vinjar

Oslo, 5. september 1995

Innhold

Kapittel 1

Innledning

For å starte Common Music må du gjøre et av følgende:

- **SGI:** skriv `cm` i et unix-skall
- **NeXT:** skriv `cm` i et unix-skall
- **Macintosh:** EPLE-meny→Musikk Quadra→Common Music

Høstens kurs er det første av flere kurs. Disse vil fokusere på komposisjons som strukturering av abstrakt materiale. Kursene vil ta utgangspunkt i "normal" musikk og tildels tradisjonelle komposisjons-teknikker. Vi skal gå igjennom bruk av formalismer i komposisjon, og trene på utleding og implementering av disse—samt å arbeide med musikk som et sett av symboler som kan manipuleres på ulike måter.

Kurset vil gå gjennom teknikker og problemstillinger knyttet til algoritmisk generering og editering av musikk. Vi skal gå igjennom kjernen i Common Music og Stella—som sammen danner et lisp-basert system for interaktivt musikkarbeide. Common Music er et avansert komposisjonsverktøy, og er kjernen i computermusikk virksomheten bl.a. ved CCRMA og ZKM. Stella er et tekstlig og grafisk grensesnitt til dette systemet. Common Music er installert i alle NoTAMs maskiner — SGIer, NeXTer og Macintosher, og kjører på de fleste datamaskintyper.

Common Music har store muligheter for, men er ikke begrenset til, algoritmisk spesifisering av musikk. Systemet kan generere score-filer i en lang rekke formater — Csound, Mix, MIDI, CLM, CMN, CMusic, CMix, RT og MusicKit — og det er relativt lett å legge til nye formater når det skulle trenges. Man kan også *importere* materiale fra f.eks MIDI-filer, og arbeide videre med dette på lik linje med de data systemet generer "fra scratch".

Dette kursheftet er delt opp i kapitler som hver for seg tar opp noen aspekter knyttet til symbolsk komponering. Det er eksempler og en hel del programkode knyttet sammen med teksten. For å få godt utbytte av kurset bør man gå igjennom eksemplene og programkoden på egenhånd, samt å utarbeide interessante oppgaver som man søker å løse. Prøv å jobb flere sammen, og hvis noe er uklart så bruk hverandres hjelp. Spørsmål kan selvfølgelig også rettes til Anders, enten pr. ansikt eller pr. (`andersvi@notam.uio.no`).

I tillegg til kursheftet er det smart å kikke på Rick Taube's tutorial til "Stella". Den fins i bokhylla på NoTAM, og on-line ved

`http://www.notam.uio.no/~andersvi/doc-html/stella.html`

På de samme stedene fins også en del annen dokumentasjon, bl.a. et helt nødvendig oppslagsverk til Common Music, en gjennomgang av `item-streams` som er et sentralt element i arbeide med Common Music og Stella, flere eksempler samt dokumentasjon knyttet til andre programmer vi kommer til å streife bortom på kurset, som CLM og CMN.

Kapittel 2

Datamaskin-assistert komposisjon

> Symbolsk komponering—komposisjon med symboler?—hva er nå det for noe? Hmm. . . (lang pause). . . runde, svarte prikker. Noen krøller på et ark—mest sidelengs—havner nede i høyre hjørne etterhvert—noen rare ord på et fremmed språk—noe ligner på engelsk, ispedd noen latinske/italienske gloser. Tja. . . symboler?. . . magi, ritualer, symboler, tryllerim, meditative regler — noe grusomt begynner snart å ta form!—PROGRAMMERINGXXPRÅX??. . . hjelp!. . . "aarrghh. . . så skulde det da altsaa meg og at ramme"[1]

Denne beskrivelsen passer kanskje best på noteskrift. Noter og Lisp har det til felles at de begynner øverst til venstre og tilsynelatende slutter nede og til høyre, samt at de på en eller annen måte hopper litt rundt omkring noen ganger.

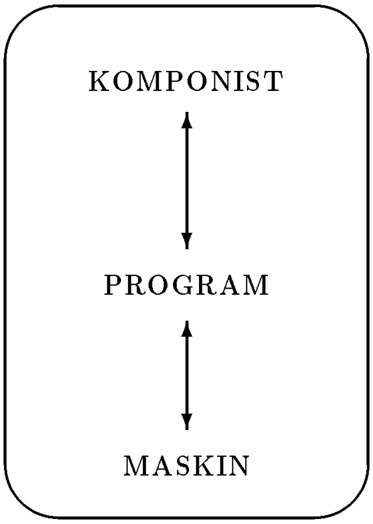
Men her dreier det seg altså om musikk og datamaskiner. Musikk og programmering. Alternative titler for kurset kunne være "Algoritmisk komposisjon", "Datamaskin-assistert komposisjon" eller "Innføring i programmering" osv. Vi *skal* arbeide med algoritmer og programmering *også*! Begrepet *Symbolsk Komponering* er mer knyttet til komponistens arbeidssituasjonen enn til ulike realisasjoner av det ferdige verket. Om musikken tas vare på i form av noter eller som ulike uttrykk nedfelt i et annet språk—norsk, Lisp—spiller jo *overhodet* ingen rolle for den skapende kunstner, ei heller for den nøytrale analytiker som ser gjennom krøllene og inn til den **egentlige** musikken!

---

[1] fritt etter Ibsen.

2.1 Datamaskiner og vi

Datamaskiner er fantastisk fleksible verktøy. Den samme maskinen kan brukes som kalkulator og til å behandle tekst. De kan brukes til å tegne med og som arkivar. Vi lager disse ulike maskinene ved å legge inn program. Vi kan forestille oss at vi arbeider med sånne program på 3 ulike nivåer. Øverst er det oss mennesker. Vi har idéer om ting som kan behandles maskinelt — algoritmer. Sånne algoritmer kan realiseres ved at vi lager program for algoritmen i et programmeringsspråk. Dette programmet blir så igjen utført av maskinen.



De programmene vi går igjennom her på kurset kan kjøres på alle vanlige maskintyper — SGI (Unix), Macintosh, PC, NeXT. Under programmeringen trenger vi ikke ta stilling til hvilken maskin vi arbeider på. Programmet vi bruker tar vare på den viktige oversettelsen fra program–kode til maskin–kode. Dette gjøres helt skjult for oss, og man må jobbe *veldig* hardt hvis man vil påvirke måten interaksjonen mellom program og maskin foregår på.

2.2 Programmer for komposisjonsarbeid

Selv om den vesentligste innsatsen med datamaskiner i musikkarbeid har dreid seg om å bruke datamaskin-prosessoren som *lydbølgekompilator* (Csound, CLM, Ceres osv.) har også folk interessert seg for mulighetene for å bygge seg komposisjonsomgivelser ved hjelp av programmer. Det er laget flere språk for dette, alle mer eller mindre spesialisert til ulike formål. F.eks. har Csound systemet innebygget et lite komposisjonsspråk — `Scot` — som kan ta imot spesielle uttrykk for å generere en Csound note-liste. På NoTAM er det også laget et fint lite språk som heter `inter`, som kan brukes til å spesifisere regler og sammenhenger til ulike formål.

2.2.1 3 nivåer

Vi kan tenke oss 3 nivåer i arbeide med komponering av musikk. Øverst er det komponistene som arbeider med mer eller mindre klare **tanker** om en komposisjon. Musikken/idéene blir formulert i en mer eller mindre tjenlig **representasjon**, som så blir **realisert** med vekslende hell. Dette er riktignok en tenkt situasjon. De ulike nivåene går over i hverandre, men det kan være klargjørende å skille dem fra hverandre et øyeblikk.

Kapittel 3

En kort gjennomgang av noen musikalske formalismer

3.1 Innledning

> "**formalisme** (av lat.), ensidig iakttagelse av den ytre form; overholdelse av reglers formelle bud og krav, selv om det krenker rimelig fornuft og menneskelighet."

Her følger noen tradisjonelle/historiske musikalske formelle metoder. Jeg skisserer problemstillingen (meget kort), viser en mulig formulering av denne i Common Music, og viser resultatet i form av noter. La disse kun være utgangspunkt. Arbeid med dem selv — bland dem sammen, utvid dem, lag musikk!

3.2 Sideblikk til *innholdet* i det vi driver med her

> "Hva utnytter hva til hva?"[1]

Det som ble skrevet om datamaskiner på side 6 gjelder utvilsomt for komponister også. De kan brukes til så mangt, og lar seg også utnytte uten å bringe altfor sterke hemninger på banen... Spørsmålet ovenfor er interessant å stille innen det feltet vi arbeider med.

---

[1] Dette er ikke svensk!

Musikk er helt avhengig av proposjoner og systematisering. Vi har alle hørt om et betydelige samsvar mellom aspekter ved matematikk og musikk. Cassiodorus (ca. 485 – 575) framla disse 4 grenene av matematikk: aritmetikk, musikk, geometri, og astronomi. Poenget for ham var at de alle befattet seg med abstrakte størrelser. Filosofen Susanne Langer fastholder at musikk er et symbol-system, hvor det *betegnede* — denotata om man vil — utgjør en "følelsenes morfologi." Hun har skrevet, "music conveys *general forms* of feelings, related to specific ones as algebraic expressions are related to arithmetic [expressions]"[2]. Musikk er for henne en slags algebra for å utrykke arketyper for følelser.

Vel, poenget her var altså at vi ikke må miste blikket for det musikalske innholdet i det vi forsøker å radere inn i en formalisme.

---

[2] Susan Langer, *Philosophy in a new key;* New York 1948

Kapittel 4

Liste over noen formalismer

4.1 Guidos metode

Vi starter for 1000 år siden med en Benedictiner munk ved navn Guido D'Arezzo. Han la fram en praktisk metode i en lærebok for sangere, *Micrologus*, som kunne frambringe en melodilinje til en hvilken som helst tekst. Han bruker tabelloppslag hvor vokalene i teksten puttes inn — og vips[1] — ut kommer en note i skalaen.

Guido brukte et system som han kalte *solmisasjon* for å holde styr på tonenes beliggenhet i skalaen. Han gikk ut fra et heksakord som omfattet den diatonske grunnskalaens seks første toner, og navnga tonene *ut re mi fa sol la* etter begynnelsesstavelsene i de seks første linjene i Johanneshymnen "*Ut* queant laxos *re*sonare fibris…". Disse trinnene var relative tonetrinn, intervaller om man vil, og han og etterkommerne utviklet et sinnrikt system for transposisjon og modulasjon. Først seinere (rundt 1600-tallet) ble disse skalatrinnene knyttet til faste tonenavn — *ut* ble *c* osv. — og *si* ble lagt til.

Γ	A	B	C	D	E	F	G	a	b	c	d	e	f	g	a
a	e	i	o	u	a	e	i	o	u	a	e	i	o	u	a

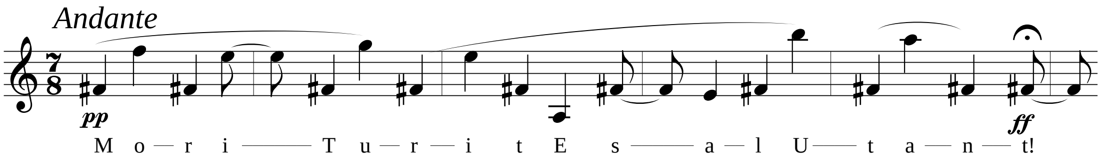
Figur 4.1: Guidos tabell

---

[1] (jada — det blir færre av disse etterhvert)

i common music kan dette f.eks. formuleres sånn:

```
(generator guido midi-note (rhythm 1.0 duration 1.0 amp 1.0)
   (setf vokal (item (items m o rr i tt u rr i t e s a l u t a n t)))
   (setf note (case vokal
                 (a (item (degrees g3 e4 c5 a5 in random)))
                 (e (item (degrees a3 f4 d5 in random)))
                 (i  (item (degrees b3 g4 e5 in random)))
                 (o (item (degrees c4 a f5 in random)))
                 (u (item (degrees d b g5 in random)))
                 (t (degree fs4)))))
```



Figur 4.2: guidos metode

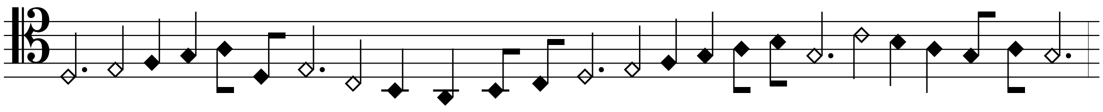
denne metoden gir 3 mulige notevalg for hver vokal (4 for *a*). den gir altså mulighet for flere melodier til en gitt tekst. guido fremholder at en del av metoden er å vurdere de ulike mulige melodiene som framkommer, og velge den som følger regler for god komposisjon. er metoden en algoritme?

4.2 isorytmikk

Isorytmikk er en annen antikk formaliseringsteknikk. Den begynte som en særegen organiserings-teknikk for rytme i motettene til Philipp de Vitry rundt 1300-tallet. Komposisjons-teknikken vokste fram av de modale prinsippene på 1200-tallet, den såkalte *moduslæren*. Prinsippet i isorytmikk er at en melodilinje — *ténor* — og et kortere rytmisk skjema — *talea* — kombineres. De blir hver for seg gjentatt et visst antall ganger. Dette sørger hele tiden for nye kombinasjoner av note- og rytme-verdi.

```
(defvar banbull 35)                 ;paven lyste hele greia lukt til...

(generator isorythm instrument-note (end banbull)
   (setf note (item (notes d3 e f g a
                           d e c b2 a b c3
                           d e f g a
                           b g c4 b3 a g
                           a g)))
   (setf rhythm (item (rhythms h. h q q e e)))
   (setf duration rhythm))
```



Figur 4.3: Isorytmikk

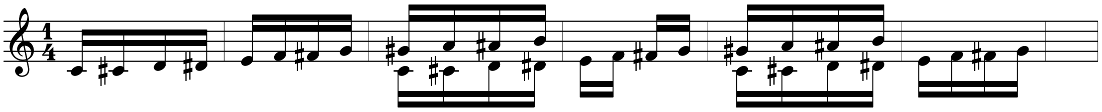
Herr Vitry skrev skriftet *Ars Nova* som ble grunnleggende for den polyfone musikken i Europa fra 1300-tallet og videre. Han og hans samtidige var på jakt etter måter å organisere flerstemthet på i formale termer. Resultatene de oppnådde ble kraftig reagert mot av øvrigheta (kirken), tilsynelatende uten å hemme utviklingen av polyfonien.

4.3 Kanon

En annen metode for å etablere et formalt anker for en komposisjon er kanon, hvorved komponisten genererer en eller flere tillegsstemmer ved å innføre en forsinket versjon av en melodilinje. Kanon eksisterte i folkelige versjoner under navn som *rondellus, rota* og også *fuge* og *caccia*. Den ble dyrket og fullendt i *Ars Nova*-tiden og fram til 1500-tallet.

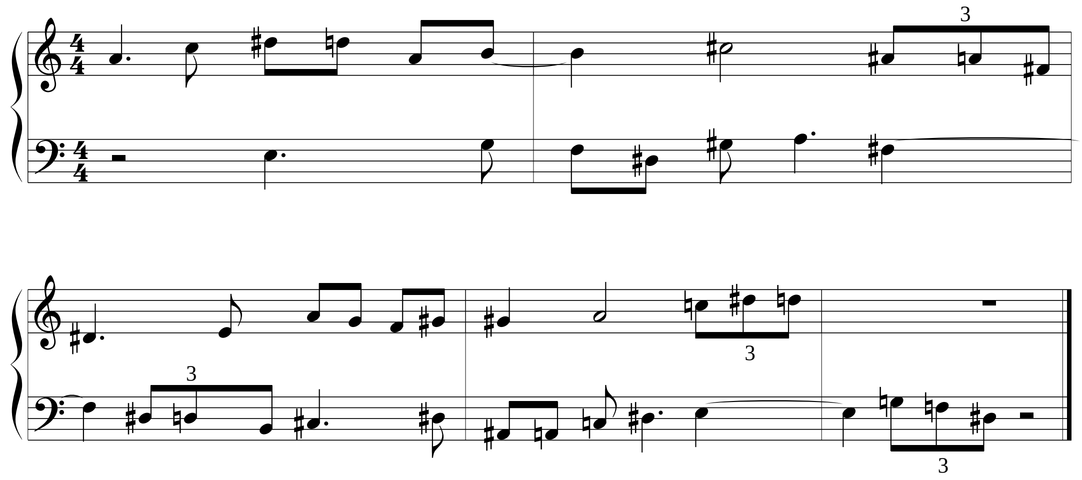
Den enkleste kanontypen, som vi finner igjen i barnesanger og 40-års-lag, er kanon i prim eller oktav. Det er da vanligvis en sirkel- eller uendelighetskanon (stemmene begynner forfra igjen når de kommer til slutten). Mer komplisert er kanon i andre intervaller, kanon i motbevegelse (imitasjons-stemmen går i en melodisk omvending av den opprinnelige) og krepskanon, med evt. augmentasjon eller diminusjon av noteverdier. Andre viktige typer er dobbeltkanon (to stykker samtidig) og cantus-firmus kanon, hvor en cantus-firmus blir akkompagnert i kanonform. Dette blir etterhvert uhyggelig komplisert å komponere når man skal ta harmoniske forhold med i betraktningen. Penderecki bruker kanonteknikker i *Threnody*, Babbitt i *Semi-Simple Variations* osv. osv.

```
(merge kanon-A ()
   (loop for begynn from 0 to 6 by 2
         for kanonnavn in '(kanon1-1 kanon1-2 kanon1-3)
         do
         (generator (name kanonnavn) instrument-note (start begynn length 12)
            (setf note (item (steps 1 from 'c4 for 12 returning note)))
            (setf rhythm (rhythm 's))
            (setf duration rhythm))))
```



Figur 4.4: Kanon-A

En kanon som tar i bruk noen av de avanserte mulighetene:

```
(merge kanon-B ()
   (generator kanon1 instrument-note ()
      (setf rhythm (item (rhythms (items 1.5 0.5 0.5 in palindrome for 6)
                                  h te te te) :kill 2))
      (setf note (item (series 0 3 6 5 0 2 4 1
                                  from (notes a4)
                                  returning note
                                  forming (items p i))))
      (setf duration rhythm))
   (generator kanon2 instrument-note (start 2.0)
      (setf rhythm (item (rhythms (items 1.5 0.5 0.5 in palindrome for 6)
                                  h te te te) :kill 2))
      (setf note (item (series 0 3 6 5 0 2 4 1
                                  from (notes ds3)
                                  returning note
                                  forming (items r ri))))
      (setf duration rhythm)))
```

Figur 4.5: Kanon-B

4.3.1 Davies: *Vesalii icones*, utdrag fra Nr. 1

Dette verket, som består av 14 danser, er et i en rekke kammer-teatralske-verk som Mr. Davies skreiv på 60-tallet. Verket inkorporerer noen religiøse elementer, og benytter seg av imitasjon og parodisering for å realisere flere samtidige, gjensidig avhengige nivåer av musikalsk og dramatisk mening. Hver av de 14 dansene er koblet til tegninger fra en anatomibok fra 1500-tallet, laget av Andreas *Vesalius*, samt 14 scener fra noe bibel-tekster om kristi kors. Davies prater om et 3-lags sett av dramatiske super-posisjoneringer: Vesalius illustrasjonene, scenene fra teksten, og danserns kropp. Tilsvarende bruker han 3 musikalske super-posisjoneringer: pop-musikk, renessanse-polyfoni, og hans egen musikk[2].

Han benytter seg bl.a. av teknikker vi har sett på her. Et lite stykke ut i første dansen (takt 8→20) tar solo-celloen en pause, og fløyte, bassett klarinett, xylofon og piano spiller en slags akkompagnerende dobbelt-kanon. Fløyte spiller melodien rett fram, klarinetten spiller en transponert, retrograd versjon av den samme melodien. Xylofonen og piano gjør det samme, men starter seinere og diminuerer rytmeverdiene for å nå igjen de andre. (Diminuering av rytmeverdier var vanlig i Ars Nova også.)

La oss forsøke å formulere noe sånt i Lisp.

Først noterekka og rytme rekka som henta fra celloen. Siden disse rekkene skal brukes av flere instrumenter er det greit å lage en funksjon som kan levere dem der og når de trengs.

---

[2] Det er ikke helt enkelt å holde de 3 fra hverandre i Davies' vedkommende. (red. anm)

```
(defun lagnoter ()
  (notes bf4 fs c5 e gs4 r c e5 bf fs4 d5 r e bf f6 g f5 ef6 r))
(defun lagrytmer ()
  (rhythms q. e e q h e q e e q q e q. e. e. h e q. e))
```

Så var det de 4 stemmene. De skal spille “parallellt” eller samtidig, så vi lager en mix, *merge* på Common Musicsk, å ha dem i. Så er det bare å gi verdier til note og rytme, evt. med justeringer i form av snuing (retrograd), transposisjon eller diminuering.

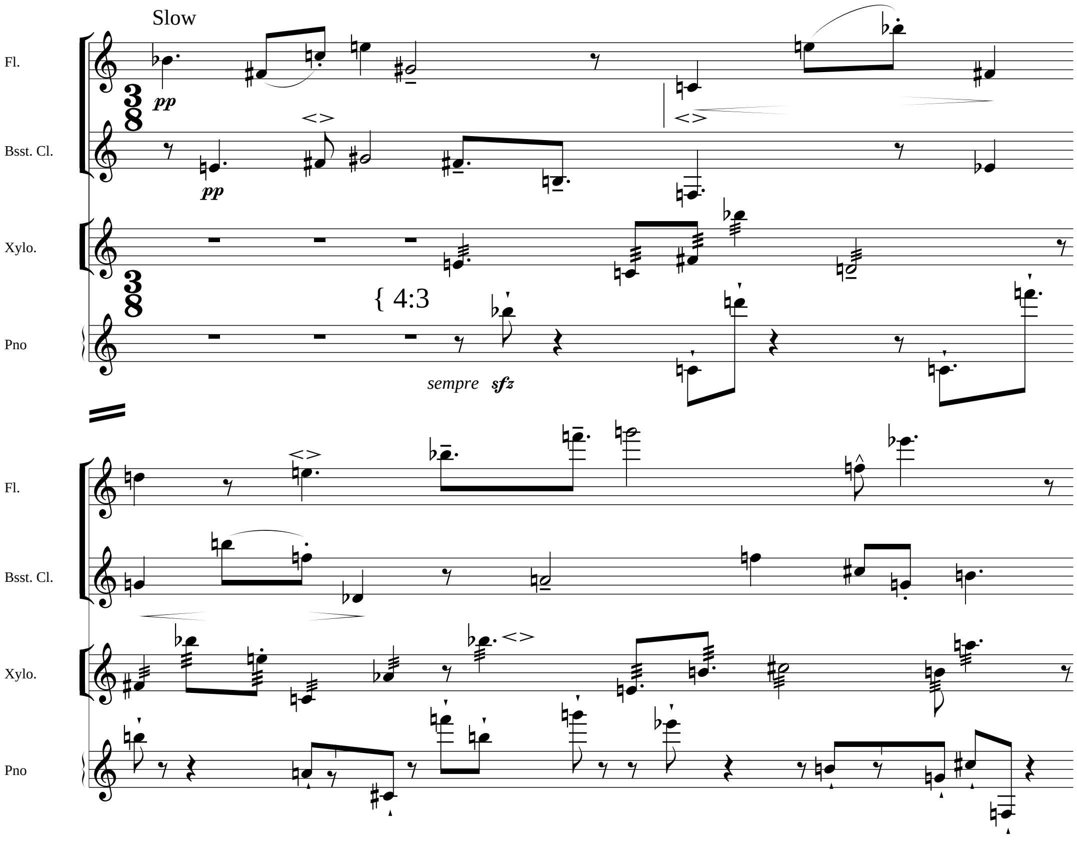
```
(merge maxwell ()
  (generator Flute instrument-note ()
    (vars (noter (lagnoter)) (rytmer (lagrytmer)))
    (setf note (item noter :kill t))
    (setf rhythm (item rytmer)))
  (generator BsstCl instrument-note ()
    (vars (noter (lagnoter)) (rytmer (lagrytmer)))
    (setf note (transpose (item (retrograde noter) :kill t) 1))
    (setf rhythm (item (retrograde rytmer))))
  (generator Xylo instrument-note (start 4.5)
    (vars (noter (lagnoter)) (rytmer (lagrytmer)))
    (setf note (transpose (item noter :kill t) -6))
    (setf rhythm (* 6/8 (item rytmer))))        ; starter seint - slutter likt
  (generator piano instrument-note (start 4.5)
    (vars (noter (lagnoter)) (rytmer (lagrytmer)))
    (setf note (transpose (item (retrograde noter) :kill t) -5))
    (setf rhythm (* 6/8 (item (retrograde rytmer))))))
```

Litt salting må gjøres med piano-stemmen når det gjelder varigheten til notene. Det er to måter å gjøre dette på. Den enkleste er kanskje å sette inn uttrykket `(setf duration (* 6/8 (rhythm 'e)))` inne i generatoren. Eventuelt kan du skrive i editoren:

```
set piano duration (* 6/8 (rhythm 'e))
```

eller noe lignende.

Noen kommentarer til koden ovenfor er kanskje på sin plass. Egentlig holder vi på med uhyggelig kompliserte ting her. Funksjonene `retrograde`, `transpose` er



Figur 4.6: DAVIES: *Vesalii icones* — takt 8–13

greie, det samme er å gange med en faktor for å forkorte verdien som kommer fra rytmestrømmene. Når en sånn `generator` eller `algorithm` blir kalt for å lage noter, blir kroppen utført i en løkke som gir nye verdier til de ulike parametrene. For hver ny note blir løkka utført en gang, og alle lisp-uttrykkene som står der blir evaluert for å returnere verdier. (`vars...`) -uttrykket i starten av generatoren sørger for at de variablene som får verdi der (noter, rytmer) kun får verdi den første gangen uttrykket blir utført. Dette er mye bedre forklart i "Common Music Dictionary" og i "Stella Tutorial", samt i de eksemplene som følger med kildekoden.

I komposisjoner fra rundt 1500-tallet finner man i komposisjoner ofte ordspill i form av gåter knytta til kanon, som må løses før verket kan spilles. Løsningen på disse gåtene ga en metode for hvordan verket skulle framføres. I satsen *Agnus Dei III* av Guillaume Dufays *Missa L'Homme armé* fins ordene *Cancer et plenus et redeat medius* ("La krabben fortsette fullt og returner halvt"). Man har tolket dette til å indikere at Cantus Firmus synges to ganger, først i fulle note-verdier, så i dobbelt tempo og retrograd bevegelse (indikert ved "redeat"). Imidlertid, siden krabber vanligvis krabber baklengs/sidelengs må jo forover egentlig bli retrograd, og returen derfor i vanlig retning...

4.4 Musikalsk Akrostikk

I renessansen ble det benyttet et metode som kalles *sogetto cavato* — "utskjært ting" — som ligner på Guidos metode, hvorved man genererte en musikalsk frase ved å ta ut viktige stavelser fra en tekst og bruke solmisasjon for å finne fram til tonene. Seinere har komponister gjort noe tilsvarende når de, som J. S. Bach bygger, et tema fra sitt eget navn. I Bachs tilfelle ble temaet brukt i hans siste verk, *Kunst der Fuge*, i den siste, uferdige, fugen. (Han planla den som en kvadruppel-fuge bestående av fire temaer. I Bachs tilfelle blir hvert tema introdusert i sin egen fuge for deretter å forenes. For mystikere er det interessant å vite at han døde straks etter han hadde introdusert temaet som er nevnt ovenfor.) Temaet B–A–C–H er brukt av mange andre komponiser seinere, bl.a. Beethoven, Schumann og Liszt[3]. Schumann brukte denne typen akrostikk flere ganger i pianostykkene sine, f.eks. i *Abegg Variasjoner*.

Prinsippet i seg selv er neppe brukt til annet enn å leke med temaer i komposisjoner som baserer sin struktur på helt andre ting. Det de tjener som eksempel på er fundamentale kunstverk som ikke desto mindre løser problemer knyttet til de formalismer vi her snakker om.

4.4.1 Eksempel på musikalsk akrostikk

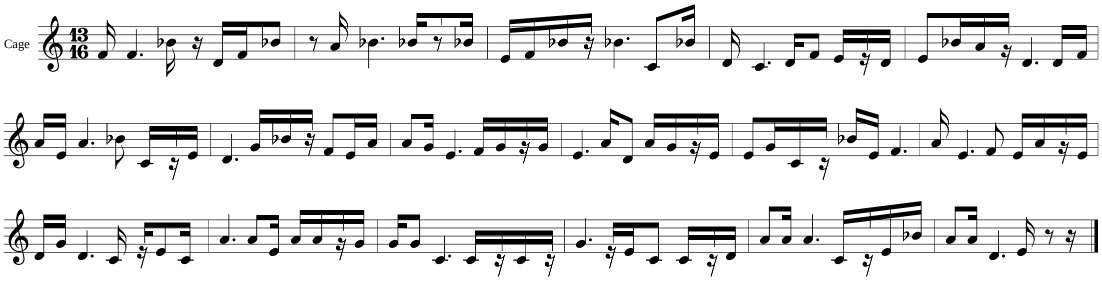
Dette lille eksempelet er egnet til å illustrere 3 viktige ting:

1. form og materiale henger sammen
2. relevant behandling av temaet
3. musikalsk akrostikk

I tillegg viser eksempelet en metode for å *veie* tilfeldige valg.

---

[3] ...de døde alle sammen...

```
(generator cage instrument-note (length 100)
  (vars
    (prob (expr (interpl count 0 0 100 8))))
  (setf note
       (item (notes (g4 weight prob)
                    (a weight prob)
                    bf
                    (c weight prob)
                    d
                    (e weight prob)
                    f
                    (r max 1)              ; ta en pause en gang i mellom
                    in random for 1)))     ; rekalk. av vekt pr. note
  (setf rhythm
       (item (rhythms s e te te te q. in random tempo 40)))
  (setf duration rhythm))
```

Figur 4.7: Akrostikk med varierende sannsynlighet

Algoritmen utsetter sannsynlighetsberegningen for notene C,A,G,E til "seinere". Når algoritmen spilles, blir disse notenes relative vektlegging forandret som en funksjon av verdien på algoritmens "*count*"-variabel som går fra 0 til $length - 1$ for algoritmen. Variabelen "*prob*" er det utsatte uttrykket (en "*expr*") som beregner vektene til C,A,G,E–notene. Mens algoritmen utføres vil "expr" -en automatisk bli regnet ut hver gang den blir støtt på i random-strømmen. Siden random-item-streams rekalkulerer sannsynligheten for elementene sine en gang pr. strømmens *periode*, har vi satt perioden til 1 her for å få sannsynligheten rekalkulert på en note pr. note basis. Sannsynlighetene er uttrykt som en interpolasjons-funksjon, så du kan eksperimentere med koordinatene for å oppnå ulike effekter[4].

---

[4] eksempelet er lånt fra kildekoden til Common Music

4.5 Serielle teknikker

Her fins både rekke-ekstremister og folk med Gynt'ske holdninger til rekkene sine. Hellig eller ikke, metoden til Schönberg og gjengen hans ble utviklet som en logisk videreføring av en utvikling som gikk mot et totalt absurd tonalitetsprinsipp i kunstmusikken. 12-tone musikken ble brukt på en måte som slo beina under ethvert forsøk på å feste den resulterende musikken til tonale sentra, og i stedet fikk vi en fokus på rekkas struktur og valget av mulige avarter av denne.

N.B. Dette er nok et eksempel på at den filosofiske og estetiske holdningen til en gruppe komponister gis uttrykk gjennom en formalisme.

Det var vel få komponister i modernismen som ikke hadde et aktivt forhold til serialismen. Alle de ulike manifestasjoner det har fått gjør ordet "serialisme" til et generelt og litt betydningsløst begrep. Folk ble pasienter, venner slutta med det, og noen seiret vel syntes de. Grunnen til dette er kanskje alle de mulighetene som ligger i prinsippet. Hvis man ordner flere enn en musikalsk parameter etter rekke-prinsipper, blir det fort så mange mulige valg mellom ulike permutasjoner at komponisten raskt må inn på banen med subjektive valg igjen. F.eks. hvis man bare ordner note-klasse og rytme etter hver sin rekke og anvender de normale variasjonene på disse, for så å kombinere note-rekkene og rytme-rekkene, har man plutselig (expt (* 12 4) 2) mulige kombinasjoner av rekker å velge utfra. I følge kombinatorisk analyse blir antall mulige permutasjoner av *en* gruppe med 12 elementer visstnok ca. 480 millioner! I praksis blir man stående å velge blant dette enorme antall muligheter for å avgjøre organiseringen av et musikkverk.

Cage påsto at dette valget burde avgjøres av tilfeldigheter, mens folk som Boulez ofte har framhevet komponistens ansvar i dette tilfellet. Andre, som Milton Babbitt, som tilsynelatende er en ultra-rekkist, bruker dem hemningsløst, men har kanskje ikke helt klart å forholde seg til det at man faktisk foretar avgjørende estetiske valg når man velger en bestemt del-mengde fra alle mulige rekker. (Fyr løs)[5]

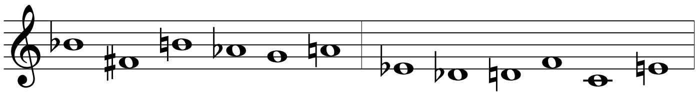
4.5.1 Eksempel på seriell metode

Rene serielle komposisjoner i Babbitt-land er enkelt i Common Music, f.eks. kan item-stream-generatoren *series* brukes til å generere rekker og deres varianter direkte. Disse rekkene kan selvsagt anvendes på hvilke som helst parametre. Babbitts komposisjon *Semi-Simple Variations* er et eksempel på en, for Babbitt, typisk måte å organisere et musikk-stykke på. Verket baserer seg på en 12-tone

---

[5] ikke på meg!

rekke som blir presentert øverst i høyre hånd i starten av stykket.



Figur 4.8: *Semi-Simple Variations* — utgangsrekke

```
(read-items (series 10 6 11 8 7 9 3 1 2 5 0 4 from 'c4 returning note))
```

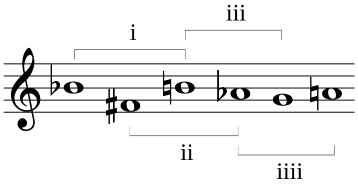
```
(AS4 FS4 B4 GS4 G4 A4 DS4 CS4 D4 F4 C4 E4)
```

Hvis vi kikker litt nærmere på rekka ser vi at den siste heksakorden er en transponert, retrograd versjon av den første heksakorden.

```
(read-items (series 4 0 5 2 1 3
                    from (notes fs4 c4)
                    forming (items p r)
                    returning note)
  12)
```

```
(AS4 FS4 B4 GS4 G4 A4 DS4 CS4 D4 F4 C4 E4)
```

I tillegg utleder han noen rekker som er basert på manipulasjoner av 4 “trikorder” henta fra denne heksakorden:

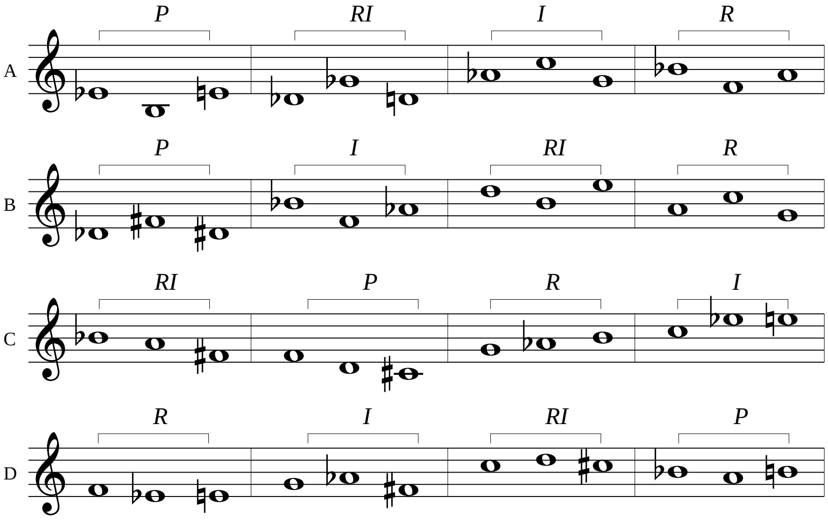


Figur 4.9: *Semi-Simple Variations* — 4 trikorder

Hver av disse genererer et derivert sett:

```
(series 4 0 5 from (notes b3 gf4 c5 f) forming (items p ri i r))
(series 0 5 2 from (notes df4 bf e5 g) forming (items p i ri r))
(series 5 2 1 from (notes b c gf f5) forming (items ri p r i))
(series 2 1 3 from (notes d a ef5 af) forming (items r i ri p))
```

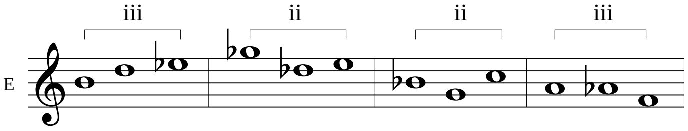
En femte rekke er derivert fra to av trikordene ovenfor (ii og iii):



Figur 4.10: *Semi-Simple Variations* — deriverte rekker

Han har tatt de 3 siste og de 3 første elementene fra C og transponert ned et trinn, samt de 6 midterste elementene fra B og transponert de første 3 opp 9 trinn og de neste 3 ned 3 trinn. Dette kan like greit gjøres i editoren:

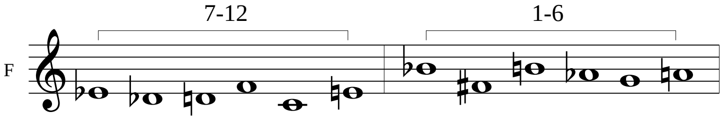
```
copy c[10:12],b[4:9],c[1:3]
paste top-level e
transpose e note -1
transpose e[4:6] note 9
transpose e[7:9] note -3
```



Figur 4.11: *Semi-Simple Variations* — 5. rekke

Til slutt lager han enda en rekke ved å bytte om på rekkefølgen av de to opprinnelige heksakordene:

```
copy babbitt-orig
paste top-level f
move f[1:6] f
```



Figur 4.12: *Semi-Simple Variations* — 6. rekke

Han har nå en haug materiale som han benytter i det videre arbeidet.

Det er all mulig grunn til å være kritisk når man vurderer et sånt arbeide. Noen framhever serialismen, og særlig den integrerte serialismen, som en mulighet til unngå begrensninger av egne subjektive føringer, og da særlig i startfasen av et arbeid. Man lager seg ganske enkelt et program som genererer verdier for alle parametre i stykket, men foretar allikevel et valg blant alle de millioner muligheter som er tilgjengelig.

Og, kanskje viktigere, man velger noen spesielle "håndtak" til å snu og vende på tingene. De samme tonerekkene fra Babbitt som ble formulert ved hjelp av `(series...` funksjonen i CM, kunne vi nådd fram til ad en haug andre veier. Tildels også lettere i den enkelte rekkes tilfelle. Poenget er imidlertid, at med den spesielle måten vi formulerte strømmene på her fikk vi lett kontroll over versjonen av rekka (`p`, `i`, `r`, `ri`) og offset (`from (notes b3 gf4 c5 f)`). Det gjør at den representasjonen vi brukte kan sies både å gå i hop med det estetiske innholdet i musikken, samt at de ulike rekkene alle er utledet ved hjelp av det samme formelle apparatet. Som datamusikere bør vi være opptatt av om den måten vi arbeider på henger sammen med det vi syns resultatet av arbeidet står for.

4.6 Sjanse — Valg

Noen har vel litt problemer når de har møtt på innledningsforedraget for svigertryner, og stolt forteller at "Jo! Jegdriver med musikk...hm...og datamaskiner..." Ikke så rart kanskje. Tanken om mekanisk komposisjon oppsto i opplysningstida, og trivdes voldsomt inntil noen kom drassende med sjeler og impresjoner og forviste idéen til ultræ futurum ultimæ et sted. Terningkastene til Mozart og hans samtidige er ikke bare vrøvl, det ble konstruert musikk-skapende spill, eller leker, for komposisjon av menuetter og tilsvarende. Musikalsk skikk og bruk i klassisismen var svært regulert, og gjorde det mulig å la utfallet av f.eks. et terningkast få bestemme et valg fra en tabell med musikalske figurer, for så å fortsette å kaste terninger inntil menuetten var ferdig.

4.6.1 Generering — Utvelging

For å vurdere denne tradisjonen som formalisme, er det nyttig å sammenligne med Guidos metode. Begge metodene kan sies å bestå av to faser: den første hvor mulighetene blir lagt ut, den andre hvor valget gjøres. Disse kan kalles *genererings*fasen og *utvelgings*fasen. I Guidos tilfelle er genereringsfasen overføringen fra en vokal til et sett av mulige notehøyder. I den siste metoden består fasen i det å bestemme et sett av mulige fortsettelsesfigurer (besto gjerne av flere takter). Utvelgingsfasen hos Guido består i å foreta et søk etter best egnede fortsettelsesnote — basert på regler for god melodiføring[6]. I tilfellet med Mozart & Co. er utvelgingen overført til en helt uforutsigbar tilfeldighetsoperasjon.

Komponister tenkte helt likt da som nå, nemlig at en “utregnet” komposisjon, basert på tilfeldighet eller ikke, kunne tjene til å stimulere komponistens fantasi. Det å regne ut komposisjoner på den måten ble gjort helt fra barokken og gjennom hele klassisismen. Teknikken ble kalt *ars inveniendi*. Det var antagelig en del diskusjon rundt anstendigheten av automatisk kompilering av musikk — ihvertfall ble det i London i 1775 utgitt en bok som het “*A Tabular System Whereby Any Person without the Least Knowledge of Musick May Compose Ten Thousand Different Minuets in the Most Pleasing and Correct Manner*”.

Tanken om at man kan/skal gjøre musikken “kalkulerbar” er altså ikke hverken ny eller bare noe de gamle monokordistene dreiv med. Moderne sannsynlighetsregning begynte vel omtrent med Pascal, og teorier om tilfeldigheters nytte eller “sannhet” har vi også syslet med en stund. Musikk har alltid og overalt blitt forsøkt underlagt regelmessighet.

4.6.2 Stokastiske prosesser

Interessant å vite at Xenakis utarbeidet sine metoder med utgangspunkt nettopp i Darmstadt. Han var elev bl.a. av Messiaen, og hans vei[7] ut av smørja den integrerte serialismen etterlot seg var å anvende sannsynlighetsregning på noen musikalske parametre, for derved å oppnå en mer direkte, intim kontroll over andre musikalske parametre. Stokastisk musikk er nettopp dette, hvor komponisten kontrollerer en del globale parametrene, og lar spredte torden og ettermiddagsbyger få herje litt fritt blandt arter på lavere nivå. *Fritt* er vel ikke helt riktig, ihvertfall når det gjelder Xenakis. Andre lar typisk musikerne få velge selv innenfor visse begrensninger, eller blant noen få muligheter.

---

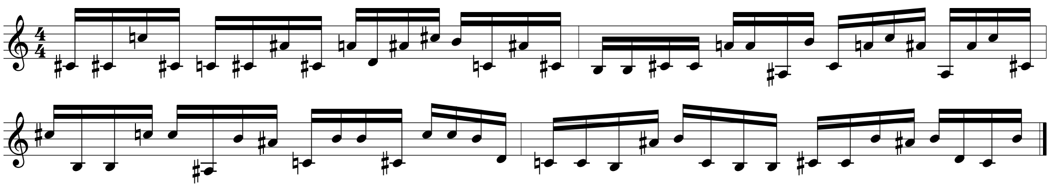
[6] Guido-eksempelet som vi implementerte tidligere er derfor ikke en algoritme, metoden baserer seg på skjønn “utenfra”

[7] en av dem

En metode som er mye brukt består i å bestemme seg for et visst makroskikt i musikken, hvorpå man lager seg et grensesnitt inn i dette globale skiktet. Snittet består typisk av funksjoner over tid, som setter ulike grenser for lavnivå parametre i stykket. Alle mulige funksjoner er anvendbare, men ofte er det enkelt og greit interpolasjons-funksjoner hvor man tegner eller mater inn x og y verdier direkte.

Dette eksempelet bruker en interpolasjons-funksjon hvor y-verdien representerer sannsynligheten for at den tilhørende x-verdien skal inntreffe. Denne distribusjonen blir så skalert og justert så toppene i distribusjonen faller nogenlunde sammen med 'c4 og 'c5.

```
(generator iannis instrument-note (length 62)
  (vars (distribusjon (pfunc '(0 0 0.15 1 0.3 0 0.7 0 0.85 1 1 0))))
  (let ((oppslag (interp (random 1.0) distribusjon)))
    (setf note (note (floor (rescale oppslag 0 1 58 74)))))  ; ≈ 'c4 og 'c5
  (setf rhythm 0.25)
  (setf duration rhythm))
```



Figur 4.13: *iannis* — sannsynlighets distribusjon

Interpolasjons-funksjoner kan brukes i alle mulige sammenhenger til å gi ulike parametre verdi. Vi kan også lage interpolasjons-funksjoner som interpolerer mellom oppslag i andre interpolasjons-funksjoner osv. Øyvind Hammer har laget en grafisk editor for interpolasjons-funksjoner på SGI'ene som heter "bredit", og på NeXT'ene fins det et fint program som heter "EnvelopeEd". Så vidt jeg vet er det en egen grafisk funksjons-editor innebygd i det grafiske grensesnittet til Common Music — *Cappella*.[8]

4.7 Rutiner fra signal prosessering

Den innfallsvinkelen til komposisjon vi snakker om her er ikke helt ulik den vi har når vi arbeider med tradisjonell signalprosessering eller lydbehandling. Random generatorer, oscillatorer, filtre, envelopes osv. — det meste er overførbart til

---

[8] Foreløpig under utvikling, og foreløpig kun på Macintosh.

strukturering av musikk på et “høyere” nivå. Alle vanlige algoritmer for lydsyntese og lydbearbeiding kan anvendes til å gi verdier til musikalske parametre i et komposisjonsarbeid. Ring-modulasjon, FM, ulineær kurveforming, kan brukes uten videre. I tillegg kan man bruke analyse-data fra ulike analyse-teknikker og la dataene styre musikalske forløp. Alt Øyvind Hammer har skrevet om dette i heftet “Digital lydbehandling…” er derfor morsomt stoff å kikke nærmere på. Litt justeringer må til: f.eks. må sampling-rate vanligvis reduseres til i størrelsesorden 1/10000 av det man vanligvis opererer med i lydarbeid. Utfordringene blir her, som med alle andre teknikker, å gjøre noe meningsfullt. La oss gjøre noen forsøk:

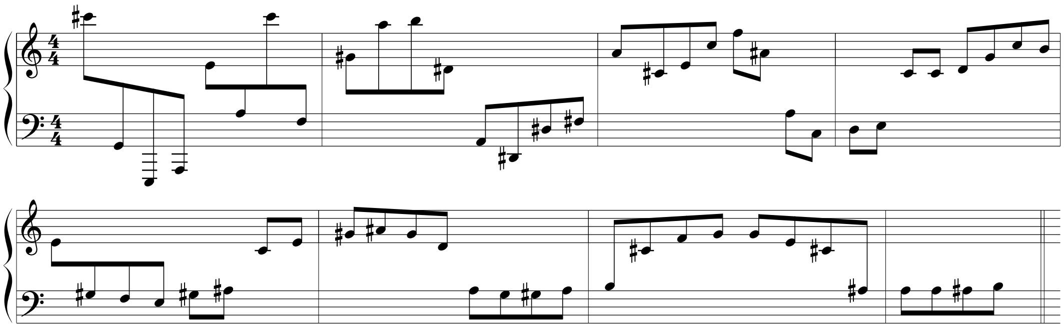
4.7.1 Karplus-Strong-algoritmen

Karplus-Strong-algoritmen blir brukt til å syntetisere lyden fra oppspente strenger som blir slått an. Kort skissert virker algoritmen som følger:

- En tabell med en viss lengde fylles med helt tilfeldige tall.
- En peker føres bortover tabellen og henter ut et og et tall som den leverer til avspilling.
- For hvert tall den plukker ut av tabellen putter den inn igjen en lavpass-filtrert versjon av det samme tallet.

Vi trenger en *tabell* for å ta vare på et visst antall noter, og et *lavpass-filter* til å modifisere nota etter at den er spilt og før den blir puttet tilbake i tabellen igjen. Vi tar det ikke så grusomt nøye med filteret her. Hvis vi for hver tone vi tar ut av tabellen og sender til avspilling tar *gjennomsnittet* av *denne* tonen og *forrige* (allerede filtrerte) tone og putter tilbake i tabellen, er det godt nok. Dette vil virke som et lavpass-filter, og etterhvert jevne ut forskjellen mellom tonene.

```
(defun fyll-tabell (lengde)     ; tabellen fylles med tilfeldige vedier
  (loop repeat lengde
        collect (+ 20 (random 70))))

(generator pluck instrument-note (length 60)
  ;; vi vil ha nye tabell-verdier hver gang generatoren kalles
  (vars (tbl-lgd 10)
        (tabell (make-array tbl-lgd :initial-contents (fyll-tabell tbl-lgd))))
  (let* ((denne (aref tabell (mod count tbl-lgd)))
         (forrige (aref tabell (mod (1- count) tbl-lgd)))
         (inn-i-tabell (/ (+ tabell-oppslag forrige-oppslag) 2)))
    (setf note (note (floor tabell-oppslag)))
    (setf (aref tabell (mod count tbl-lgd)) inn-i-tabell)
    (setf rhythm 0.5)))
```

Figur 4.14: *Pluck* — forløpet blir lavpass-filtrert

Selvfølgelig kan formelle beskrivelser fra et hvilket som helst felt brukes som materiale for en musikalsk komposisjon. Astronomi, biologi, geologi osv.

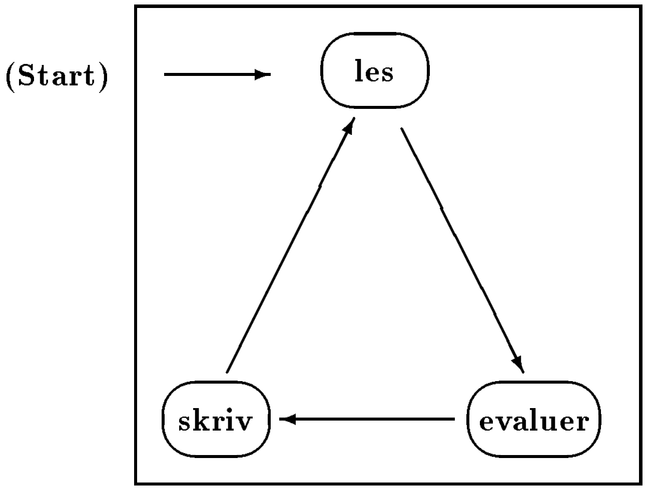
Kapittel 5

Litt om Lisp

5.1 Lisp

LISP står for **LIS**t **P**rocessing. Det er symbolsk programmeringsspråk. Språket gir programmereren mulighet til å kombinere små rekursive funksjoner for å skape desto kraftigere funksjoner på et høyere nivå. Dette kapittelet gir en innfoøring til LISP og dets celler og nervetråder.

5.2 "Les–evaluer–skriv" løkke



Når vi arbeider med Lisp, leser den noe som vi typisk skriver på tastaturet, eller velger fra en meny. Dette blir så evaluert av Lisp og resultatet av denne behandlingen blir skrevet ut til skjermen. Etter det er Lisp klar til å ta imot nye uttrykk fra brukeren, som igjen blir evaluert og resultatet skrevet ut. Denne løkka kalles *les–evaluer–skriv* løkka, og den er aktiv helt til du skrur av Lispen, eller det oppstår en feil.

5.2.1 Promptet

Vi kan se at Lisp er klar til å ta imot nye uttrykk ved at den skriver ut et *prompt* til skjermen, og plasserer markøren straks etter promptet. Hvordan dette promptet ser ut kan variere litt med forskjellige Lisp versjoner, men typisk består den av et ord etterfulgt av et kolon eller en vinkelparantes. F.eks. kan promptet i ACL (Allegro Common Lisp) se sånn ut:

```
USER(1):
```

eller:

```
<cl>
```

I MCL (Macintosh Common Lisp) er promptet et enkelt spørsmålstegn:

```
?
```

Prøv å skriv følgende til promptet og observer hva som skjer:

```
USER(2): 123.45
USER(3): -981723478923487
USER(4): "hei"
USER(5): 1/4
```

Lisp svarer med sin tolkning av det du skriver. Verdien er helt enkelt tallet selv. Foreløpig ser dette greit ut, men Lisp kan brukes til mer enn ekko-maskin. Vi kommer snart til kraftigere saker.

5.2.2 Evaluering

At vi taster inn noe, og at programmet skriver ut en melding på skjermen tilslutt virker forsåvidt enkelt og greit, det er den delen av løkka som har med evalueringen av det inntastede uttrykket å gjøre som krever mest oppmerksomhet. Det er en del forutsetninger som skal til for at Lisp skal kunne komme seg gjennom disse ulike fasene av arbeidet. For det første må hieroglyfene vi taster være mulig å dechiffrere for Lisp i lese delen. Det vil si at det vi skriver må være et skikkelig *Lisp-uttrykk*. Det er noen enkle, generelle regler for hva som utgjør et Lisp-uttrykk. De må være en av følgende type:

- **tall:** 1234567 987.65 -1.098765432
- **symbol:** `rotere` `pizzicato`
- **tegn:** `#\a` `#\B`
- **string:** "nu!" "tempi" "dette ER en string"
- **liste:** (1 3 "hei") `(nummer-en (tre (2)) (fire))`

5.2.3 Enkle regneuttrykk

Vi kan sette i gang å skrive ulike uttrykk. Først noen kalkulasjoner:

```
  USER(6): (+ 1 3)
⇒ 4
  USER(7): (+ 4 3 9 8)24
⇒ så
  USER(8): (* 2 3 4 5)
⇒ 120
  USER(9): (+ (* 7 8) 3 (/ 9 3))
⇒ 62
```

Denne formen å notere prosedyrer i, med funksjonen først og argumentene på rekke etter, blir kalt for Cambridge Polish Notation (CPN), eller "prefix notasjon". Den er helt grunnleggende i Lisp. En av fordelene med denne notasjonsformen er at en funksjon kan gis et vilkårlig antall argumenter. Vi ser og at argumentene til en funksjon hver for seg kan bestå av sammensatte uttrykk. Argumentene er atskilt av et mellomrom.

Et vanlig Lisp-uttrykk er vanligvis av typen:

(**f** *a* *b* *c*)

Når dette blir levert til Lisp, vil lese-delen av les–evaluer–skriv løkka gå igjennom uttrykket for å se at hvert enkelt element — på alle nivåer — hver for seg er riktige uttrykk. Når Lisp så skal forsøke å evaluere uttrykket finner den forhåpentligvis en funksjon først i lista. Deretter går den videre igjennom argumentene ett for ett, dukker ned i evt. del-uttrykk og gjør det samme med dem, og anvender så funksjonen på de resulterende argumentene. Resultatet av hele denne utregningen blir levert tilbake som *verdien av uttrykket.*

5.2.4 Når feil oppstår

Hvis du tastet feil i sted, så det Lisp leste ikke var et riktig Lisp-uttrykk eller umulig å evaluere, havnet du i "debuggeren". Vi kan se av promptet og eventuelle feilmeldinger om vi befinner oss i debuggeren eller ikke. Debuggeren hjelper til når noe går galt ved å forsøke å gi meldinger om hvor en eventuell feil oppstod. En god debugger er et veldig kraftig hjelpemiddel i utvikling av programmer, selv om den kan virke vel kryptisk til å starte med.[1] Det blir mer om debuggeren seinere, den første utfordringen blir å komme seg vekk derfra og tilbake til det normale Lisp-promptet.

```
  USER(10): (* 3 (/ 5 0))
⇒ Error:  Attempt to divide 5 by zero
  [1] USER(11):
```

Oftest kan vi få hjelp ved å skrive `H` eller `:help` el., evt. ved å velge noe fra en meny. Det vi trenger å gjøre i dette tilfellet er å foreta en *resetting* av Lispen. I ACL gjøres det enkelt og greit ved å skrive `:reset`.

```
  USER(10): (* 3 (/ 5 0))
⇒ Error:  Attempt to divide 5 by zero
  [1] USER(11): :reset
  USER(12):
```

5.3 Lisp-uttrykk

Som sagt krever Lisp at det vi leverer til den er skikkelige Lisp-uttrykk. De grunnleggende typer uttrykk i Lisp ble kort nevnt på side 29, men det er grunn til å utdype litt. Som vi har sett kan disse grunn-uttrykkene kombineres til sammensatte uttrykk.

---

[1] seinere og...

tall
symbol
tegn
string
liste

tall i Lisp er det ikke noe spesielt med. Muligens bortsett fra at de kan bestå av vilkårlig mange enkeltsifre. De kan ha et minus-tegn først (f.eks −123.45). De kan inkludere ett eneste desimaltegn — et *punktum* — med et vilkårlig antall sifre bak desimaltegnet. Noen eksempler på tillatte tall er gitt i oversikten på side 29.

symbol er det vi kan bruke som variabel, funksjon osv.. Symboler er enten *bundet* til en bestemt verdi, eller *ubundet.* Måten å binde en verdi til et symbol på avhenger litt av hva slags verdi det er. To måter vi kommer til å bruke mye er

(**defun** *symbol [verdi]*)

hvis verdien vi skal binde til symbolet er en funksjon, og

(**setf** *symbol verdi*)

for å binde andre typer verdier til et symbol.

Vi kan skrive

```
USER(1): (setf kvart 0.25)
USER(2): (setf kvint 0.2)
```

og senere bruke disse symbolene i regneuttrykk:

```
  USER(3): (+ kvart kvint)
⇒ 0.45
  USER(4): (* kvart kvint pi)
⇒ 0.15707963502015866d0
```

For å kunne bruke symbolene i regneuttrykk må de alt ha en verdi, ellers får vi en feilmelding. Mange symboler er allerede definert i Lisp, f.eks. har symbolet `pi` den "vanlige" verdien av π. To av de viktigste symbolene som alt har verdi heter `t` og `nil`, og de betyr noe sånt som "sant" og "usant".

En konstruksjon du må kunne i forbindelse med symboler er:

(**quote** *uttrykk*)

eller den forkortede skrivemåten:

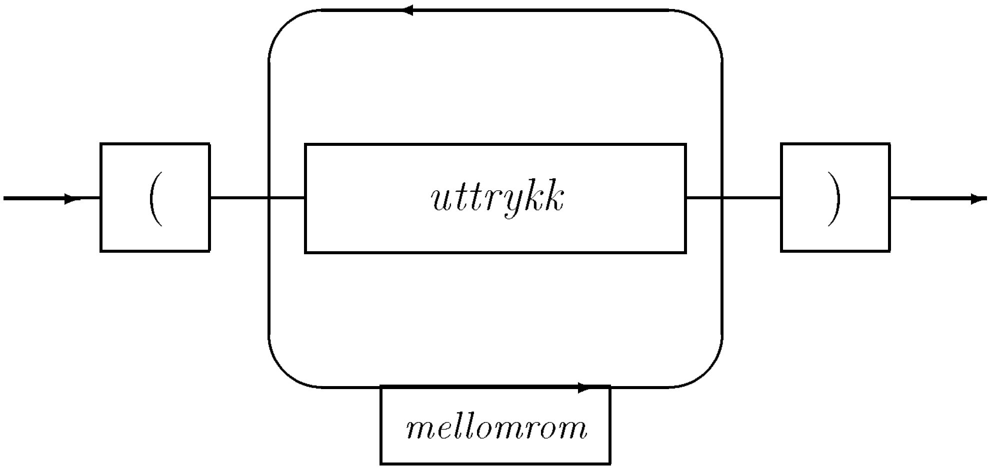
'*uttrykk*

Vi sier at quote hindrer evaluering. Verdien som Lisp returnerer er *navnet* på symbolet.

string En string består av et anførselstegn etterfulgt av null eller flere tegn, og avsluttet med et anførselstegn. Det er noen veldig få tegn som ikke får delta i stringen uten videre. (Prøv å gjett hvilke!)

tegn Hvis du absolutt må skrive enkelt-tegn må du bruke den noe sære skrivemåten *hæsj-bækkslæsj* — `#\a` — for tegnet a. Det fins endel "ikke-skrivbare-tegn" som ikke er i alfabetet eller som ikke fins på tastaturet, f.eks. fins det tegn for linje-skift, slutten av en fil osv. Fatt mot, vi kommer til å ha lite eller ingen direkte kontakt med denne formen for enkelt-tegn i resten av dette heftet.

liste i Lisp består av en parantes-*start*, etterfulgt av et eller flere Lisp-uttrykk adskilt av mellomrom, og lukket igjen med en parantes-*slutt*. Mellomrom er et eller flere space-tegn, tabulator-tegn eller linjeskift i alle kombinasjoner.



“Faren ligger ikke i at datamaskiner begynner å tenke som oss, men derimot i at vi begynner å tenke som datamaskiner…”

Register