Ircam
documentation

- Research reports
- Musical works
- Software

PatchWork

RepMus Library

First Edition, April 1996

IRCAM Centre Georges Pompidou

This manual was written by Gérard Assayag and Claudy Malherbe, and was produced under the editorial responsibility of Marc Battier, Marketing Office, Ircam.

PatchWork was conceived and programmed by
Mikael Laurson, Camilo Rueda, and Jacques Duthen.

The RepMus library was conceived by Gérard Assayage and Claudy Malherbe and programmed by Gérard Assayag, with additional musical expertise by Joshua Fineberg (AS->PW), François Nicolas (Feuilleté) and André Riotte (LC).

First edition of the documentation, April 1996.
This documentation corresponds to version 1.0 of the library, and to version 2.5.1 or higher of PatchWork.

Ircam
1, place Igor-Stravinsky
F-75004 Paris
Tel. (33) (1) 44 78 49 62
Fax (33) (1) 42 77 29 47
E-mail ircam-doc@ircam.fr

IRCAM Users' group

The use of this software and its documentation is restricted to members of the Ircam software users' group. For any supplementary information, contact:

Département de la Valorisation
Ircam
Place Stravinsky, F-75004 Paris

Tel. (1) 44 78 49 62
Fax (1) 42 77 29 47
E-mail: bousac@ircam.fr

Send comments or suggestions to the editor:
E-mail: bam@ircam.fr
Mail: Marc Battier,
Ircam, Département de la Valorisation
Place Stravinsky, F-75004 Paris

To see the table of contents of this manual, click on the Bookmark Button located in the Viewing section of the Adobe Acrobat Reader toolbar.

Contents

The Chords etc. Menu

make-graph

Syntax

lrepmusl::**make-graph** coll &optional pred
[function]

parameters

coll	a list of list of midics (or any number) or a list of chord-objects or a chord-line object
pred	(optional) must be the output of a **mk-pred** box.

output

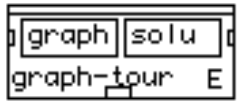
a graph object. Generally the ouput of **make-graph** is connected to the graph input of a graph-tour box.

Description

Builds a relation graph between chords in a chord set.

The default relation is the amount of common notes between chords. The *pred* input can be used to change the relation. **make-graph** may also be used to relate any kind of data that you can code into lists of numbers.

graph-tour



Syntax

lrepmusl::graph-tour graph solu &optional link order trav stat
[function]

parameters

graph	the output of a **make-graph** box
solu	positive integer. Choose a solution between 0 and n-1 (n is the number of chords)
link	(optional, menu) if 'yes' adds a low common note when there is no common notes between 2 chords.
order	(optional, menu) if '>=' maximize (default). If '<=' minimize (i.e. get path of maximum contrast).
trav	(optional, menu) if 'short' (default) short path without repetitions. If 'long' long path with repetitions.
stat	(optional, menu) if 'norm' (default) outputs the solu(nth) solution. If 'stat', prints all the solutions with an optimality factor.

output

Depends on the kind objects that have been put into the graph (see **make-graph**) :

If the graph was built with a list of lists of integers, output is a list of lists of integers.

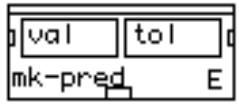
If the graph was built with a list of chord-objects or a chord-line object, output is list of chord objects.

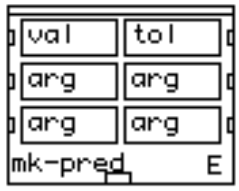
The output is generally connected to the chords input of a chordseq box.

Description

Builds a (quasi-) optimal path between chords that have been organized into a graph with the box **make-graph**. If the relation used in **make-graph** is the amount of common notes, graph-tour delivers a sequence of chords where the amount of common notes between successive chords has been maximized (or minimized). There are as many different-solutions as there are nodes (i.e. chords) in the graph.

mk-pred





Syntax

lrepmusl::**mk-pred** val tol &rest v
[function]

parameters

val	integer, value to be compared with the difference between notes of chords.
tol	integer, allowed deviation in the former comparison.
arg	(optional, integer) additional value to be used like <val>

output

a predicate function object to be connected to the pred input of a **make-graph** box.

Description

This box is used in conjunction with the **make-graph** box. It defines a predicate used to compare elements in the objects (e.g. chords) put into the graph. Each element x (e.g. note) of each object (e.g. chord) is compared to each element y of every other object. Then (y-x) is compared for equality to the parameter *val*, with the tolerance *tol*. Thus, for *val* = 0 and <tol> = 0, strict equality (e.g. common notes relation) is seek.

For *val* = 100, hal-tone upward step relation is seek. If *tol* = 25, then a quarter tone tolerance is allowed. If you build a graph using **make-graph** with these values, then find an optimal path using graph-tour, what you get is a chord sequence where there is a maximum number of half-tone steps between 2 consecutive chords, with a quarter tone tolerance.

If you add optional arguments (as many as you like), these values will be used to complexify the relation.

For instance, with <val> = 300, <opt-arg1> = 400, <opt-arg2> = 700,the optimisation will be : 'find a sequence where consecutive chords have the max amount of minor 3rd, major 3rd and perfect 5th upward steps.'

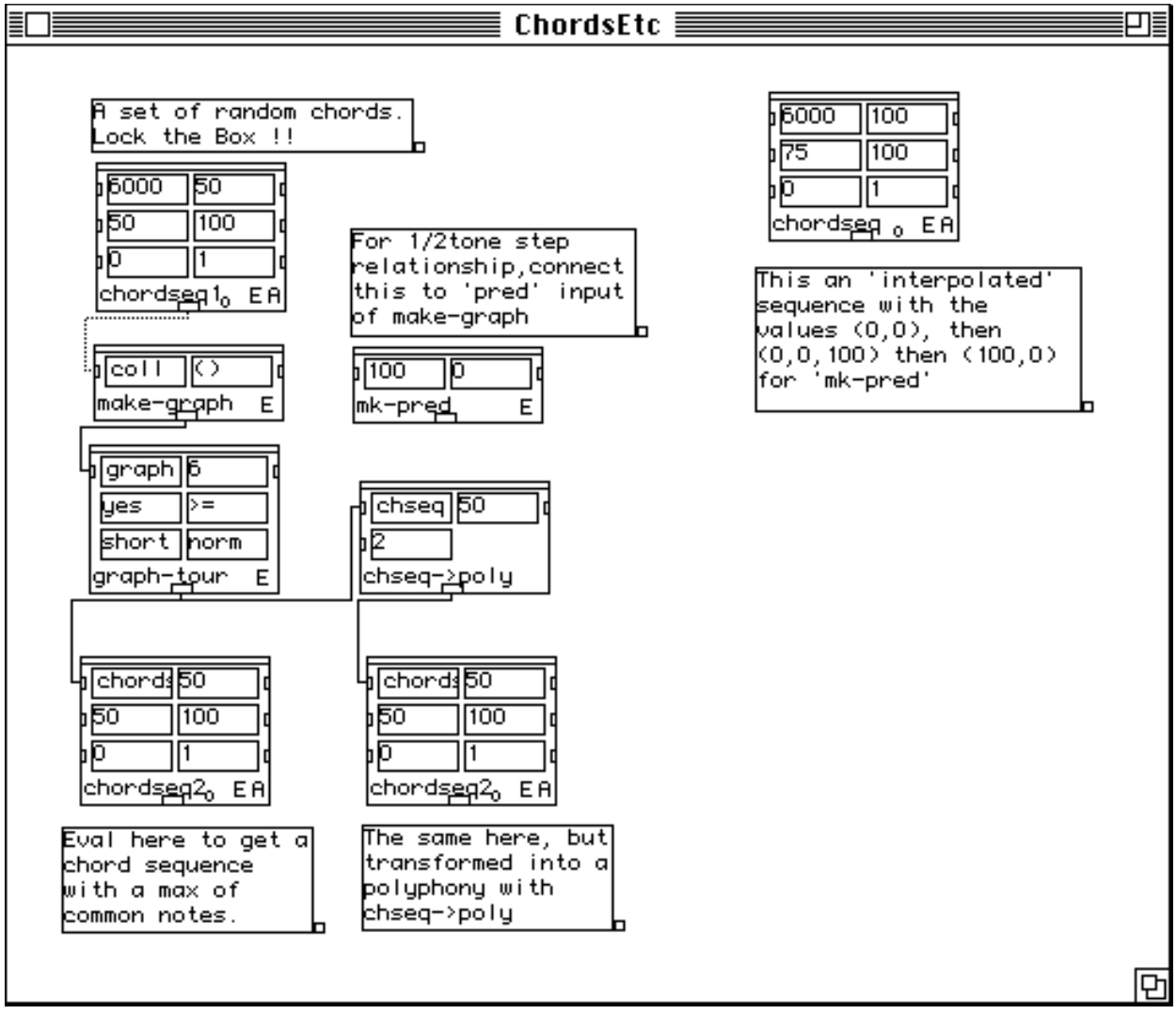


FIGURE 1 The tutorial window for boxes **make-graph**, **graph-tour** and **mk-pred**

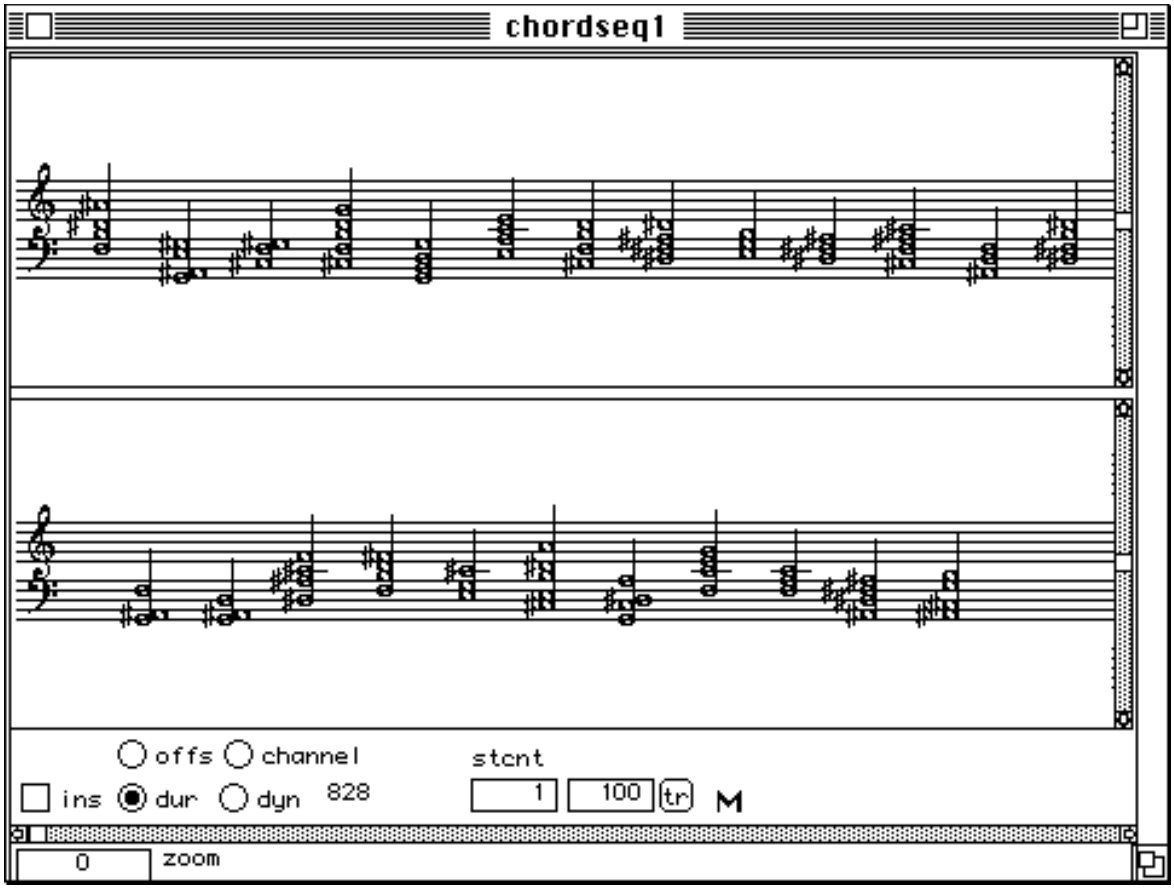


FIGURE 2 The box **chordseq1** opened

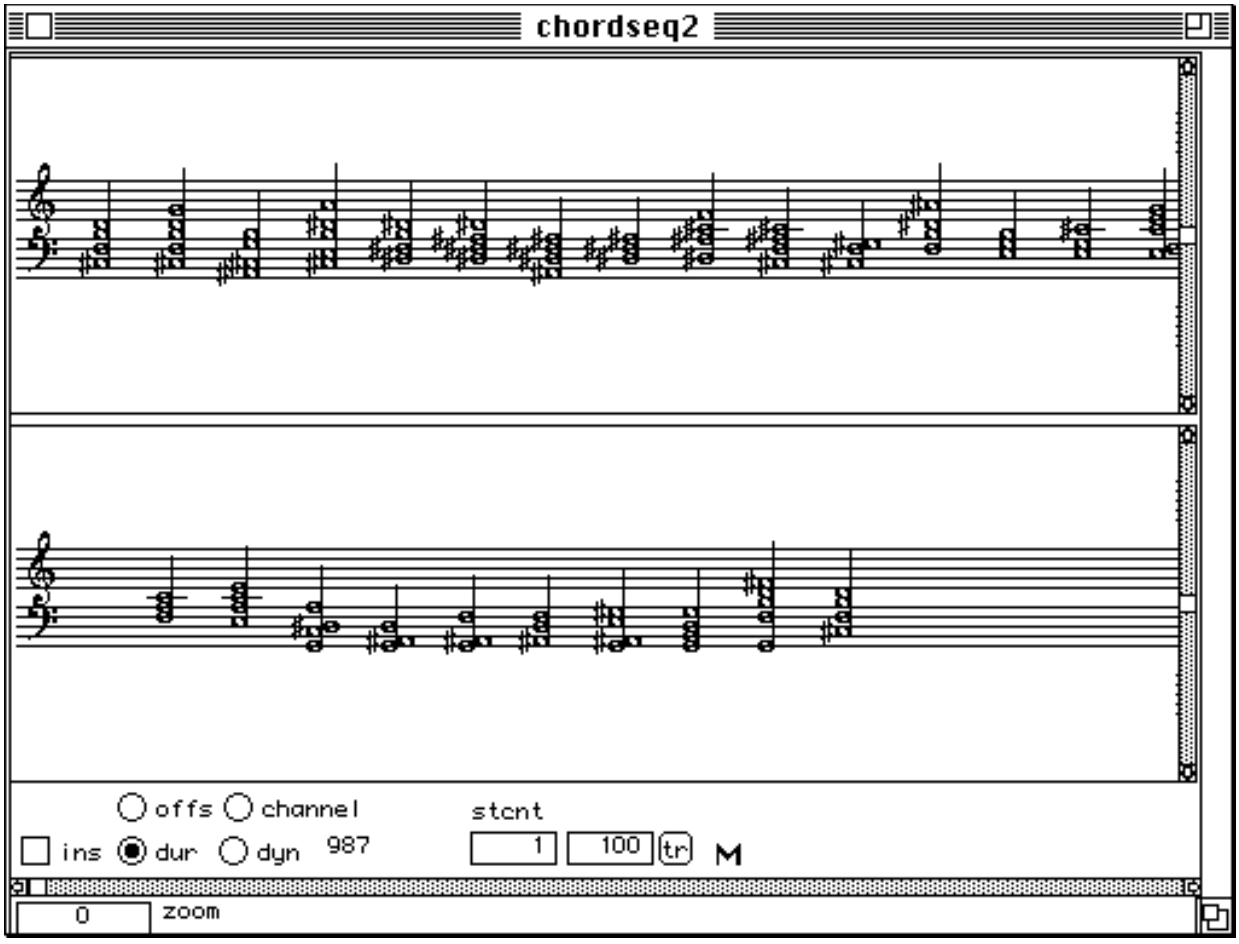


FIGURE 3 The resulting box **chordseq2** opened

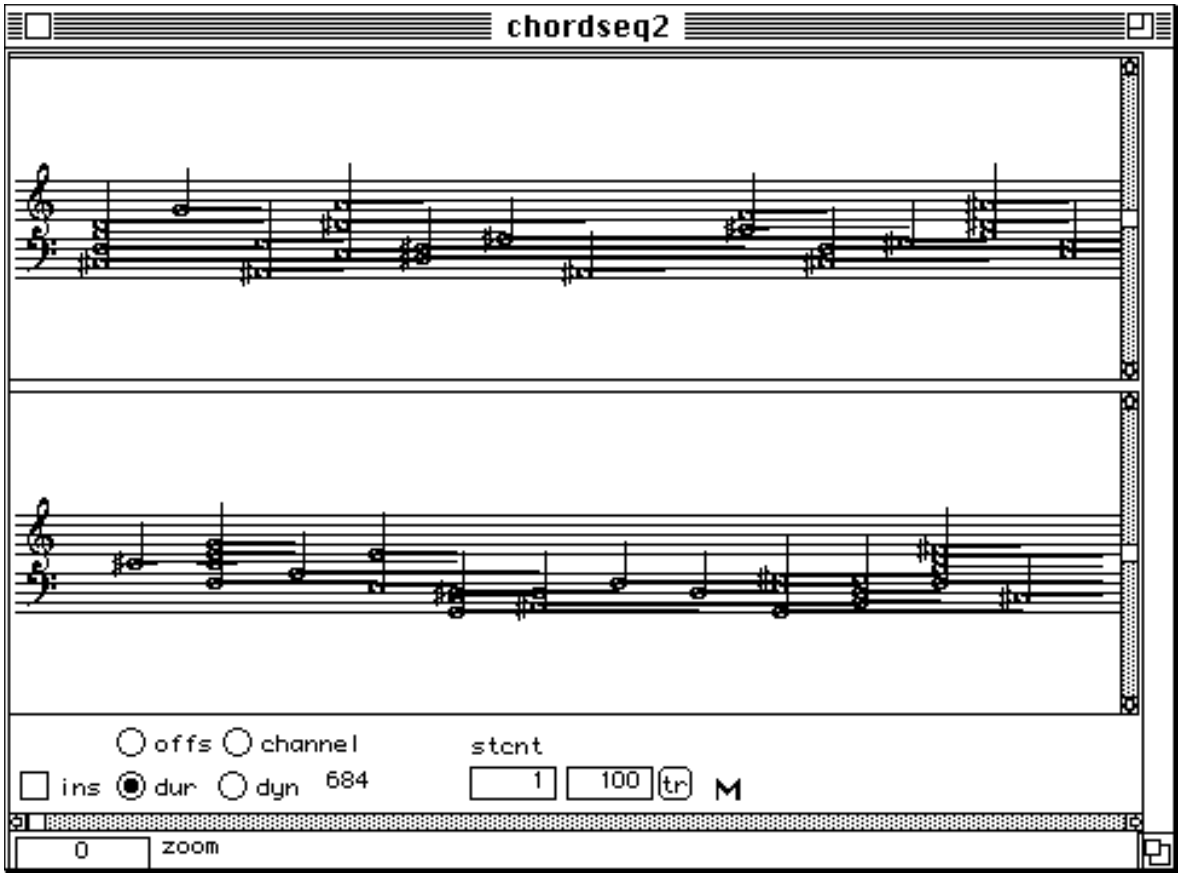


FIGURE 4 The result as transformed by **chseq->poly** box.

map-chords

Syntax

lrepmusl::**map-chords** chs1 chs2 cf ca cr cn approx penal
[function]

parameters

chs1	a list of chord-objects or a chord-line. This is the model.
chs2	a list of chord-objects or a chord-line. This is the reservoir.
cf	integer, coefficient for common notes criteria
ca	integer, coefficient for ambitus criteria
cr	integer, coefficient for register criteria
cn	integer, coefficient for number of notes criteria
approx	an integer between 1 and 16. Microtone approximation used in comparisons. 2 = 1/2tone.
penal	an integer >=0, penalty value for chord repetition

output

a list of chord-objects.

Description

map-chords takes a sequence of chords as a model, and another set of chords as a reservoir. Then it picks chords in the reservoir and it builds up a new sequence from them, trying to make that sequence look as much as possible like the model.

map-chords uses a euclidian distance measure between chords in the reservoir and chords in the model. Dimensions used are : the number of common notes, the ambitus (dist from the bottom to the to of the chord), the register (the gravity center of the chord), the difference in the number of notes. The user has the ability to give a weighting coefficient for any of these criteria thus influing on the resolution. If O the criterium is totally ignored. Typical values are between 0 and 10.

There is also a penalty parameter for chord repetition: if this value is high, a chord cannot be repeated in the sequence except if its first occurence is very far behind. Values typically between 0 (no penalty) and 10.

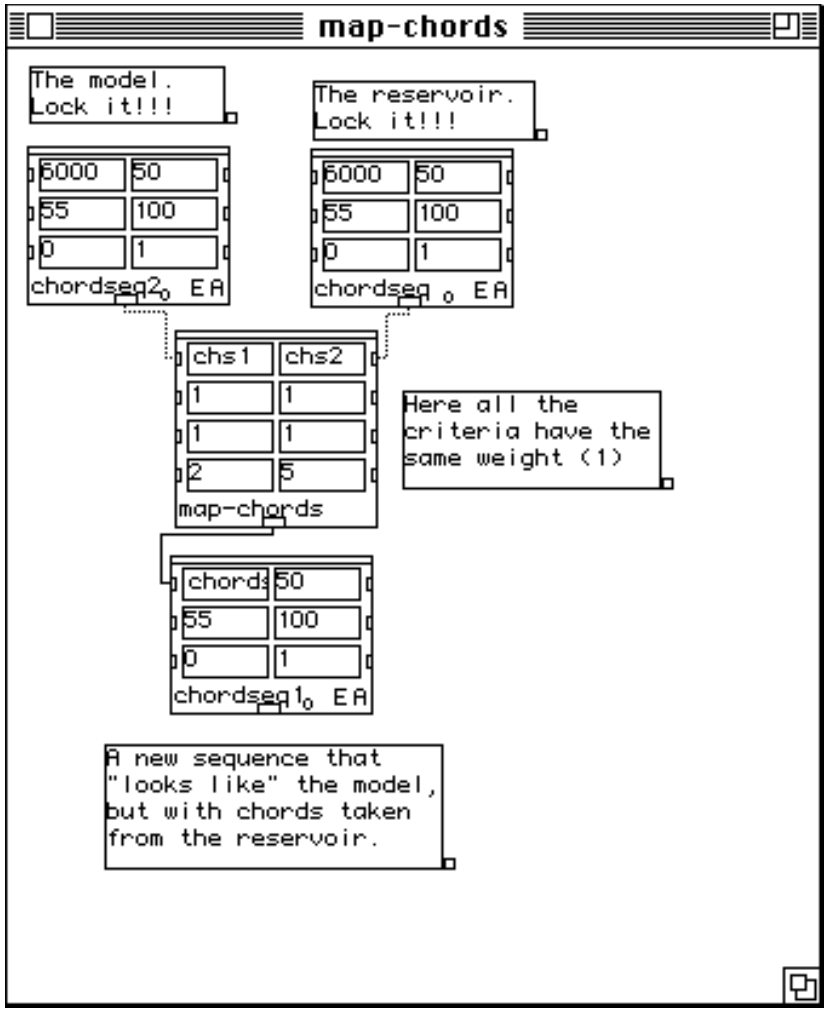


FIGURE 5 The tutorial for **map-chords** module

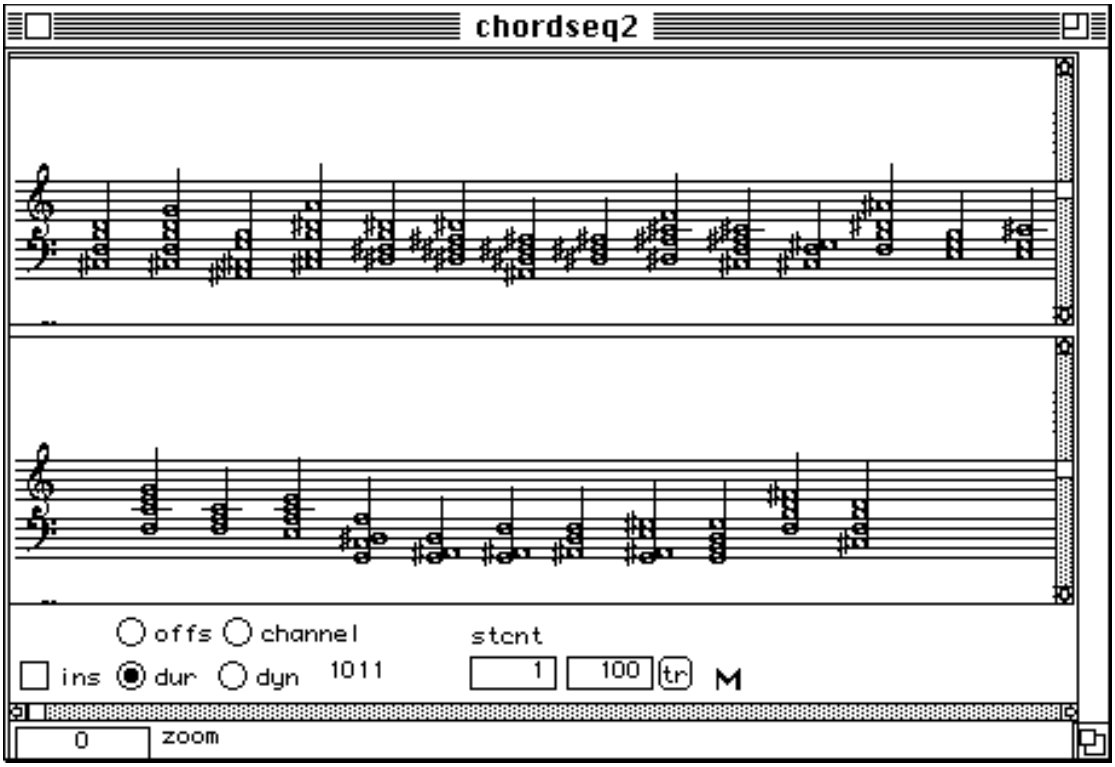


FIGURE 6 The model opened

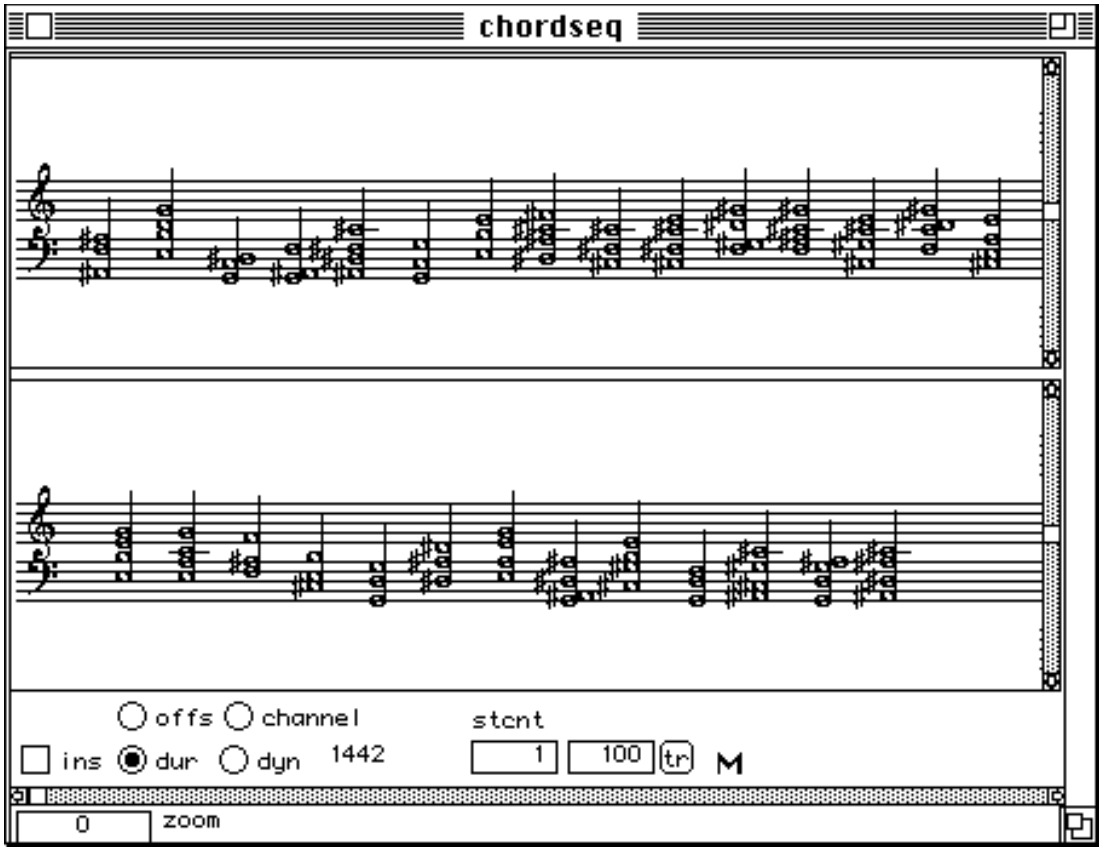


FIGURE 7 The reservoir opened

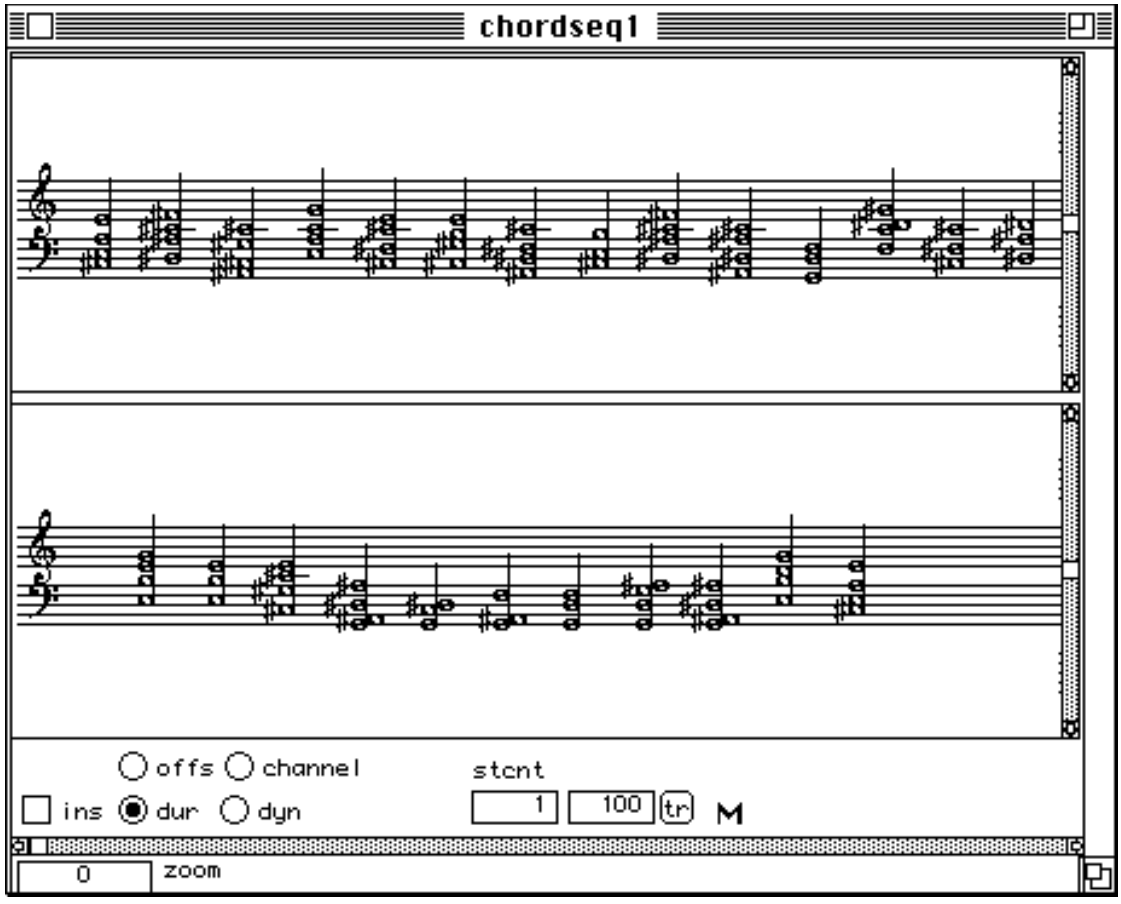
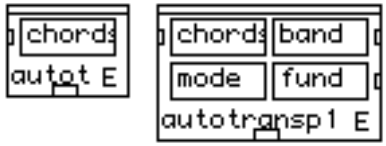


FIGURE 8 The result opened.

autotransp



Syntax

lrepmusl::**autotransp** chords &optional band mode fund
[function]

parameters

chords	a list of midics, or a chord-object, or a list of these, or a chord-line
band	(optional) a list of 2 midics, to limit the pitches down and upwards
mode	(optional, menu) if 'chrom' normal transposition, if 'spec' spectral transposition
fund	(optional, midic) gives a fundamental if in 'spec' mode.

output

a list of lists of midics

Description

Takes a chord or a series of chords and builds the auto-transposition of these chords. The auto-transposition of a chord is a set of chords resulting from transpositions of that chord, such that any note of the resulting chord is made equal to any note of the original chord. There is also a 'spectral' mode where all the notes in the transpositions are approximed to a harmonic partial of a fundamental that is specified.

If you specify a series of chords, **autotransp** will build the transposition set for every chords and put all the results in sequence.

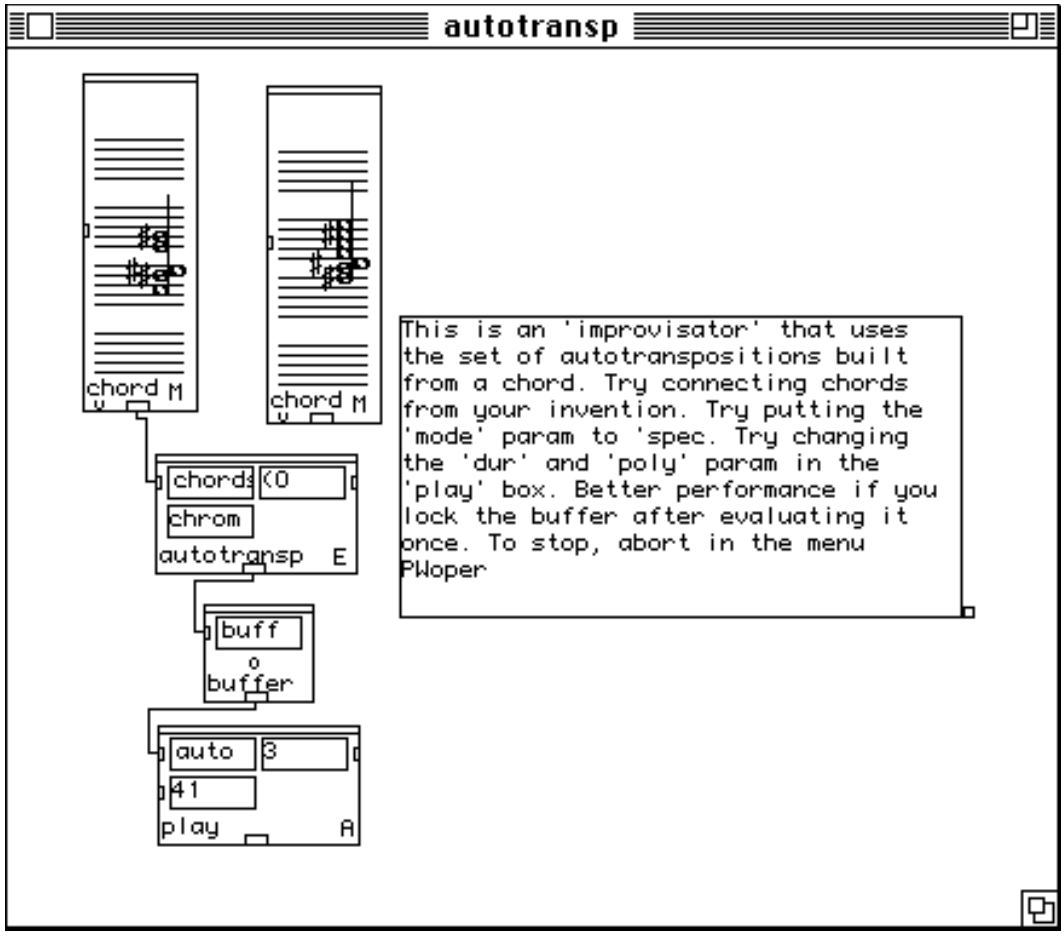
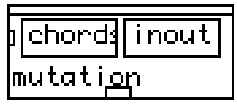


FIGURE 9 The tutorial window for **autotransp**.

mutation



Syntax

lrepmusl::**mutation** chords inout
[function]

parameters

chords a list of list of midics, or a list of **chord-object** or a **chord-line** object.
inout controls the order in which notes are added and removed.

output

a series of chord in the form of a list of lists of midics.

Description

Computes a transition sequence between two or more chords.

mutation works differently from an interpolator it generates a series of small moves - take off a note here, add a note there, move a note here etc. - that changes the first chord into the second. It does not introduce any note other than the ones that are present in the chords. If given more than two chords it generates a sequence with the transitional chords stuffed between the original chords.

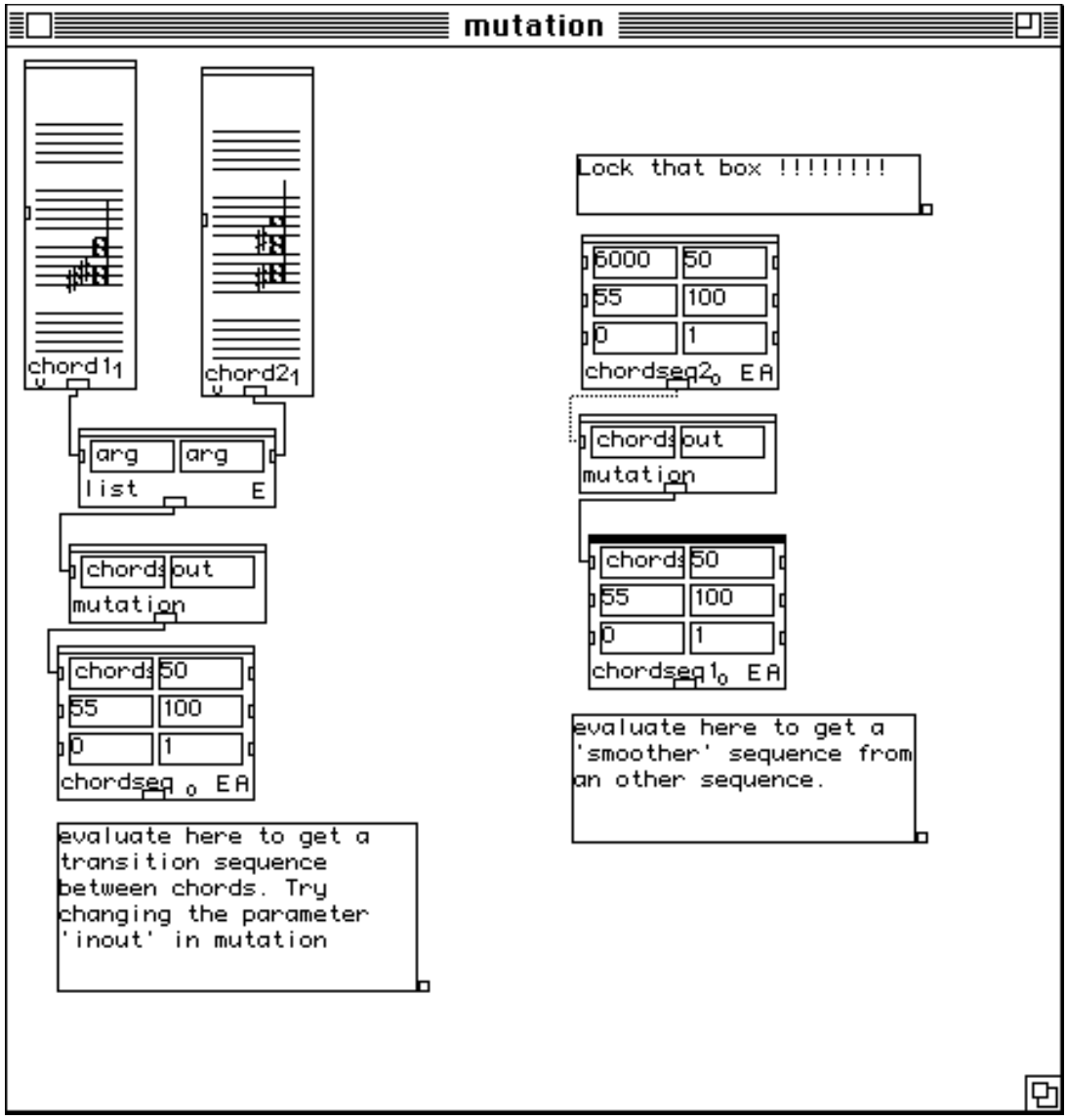


FIGURE 10 The tutorial window for **mutation**

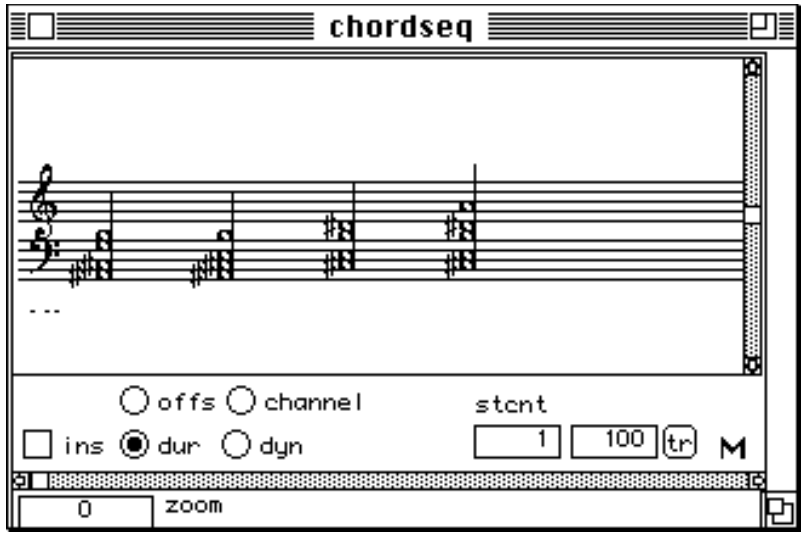


FIGURE 11 The mutation of two chords.

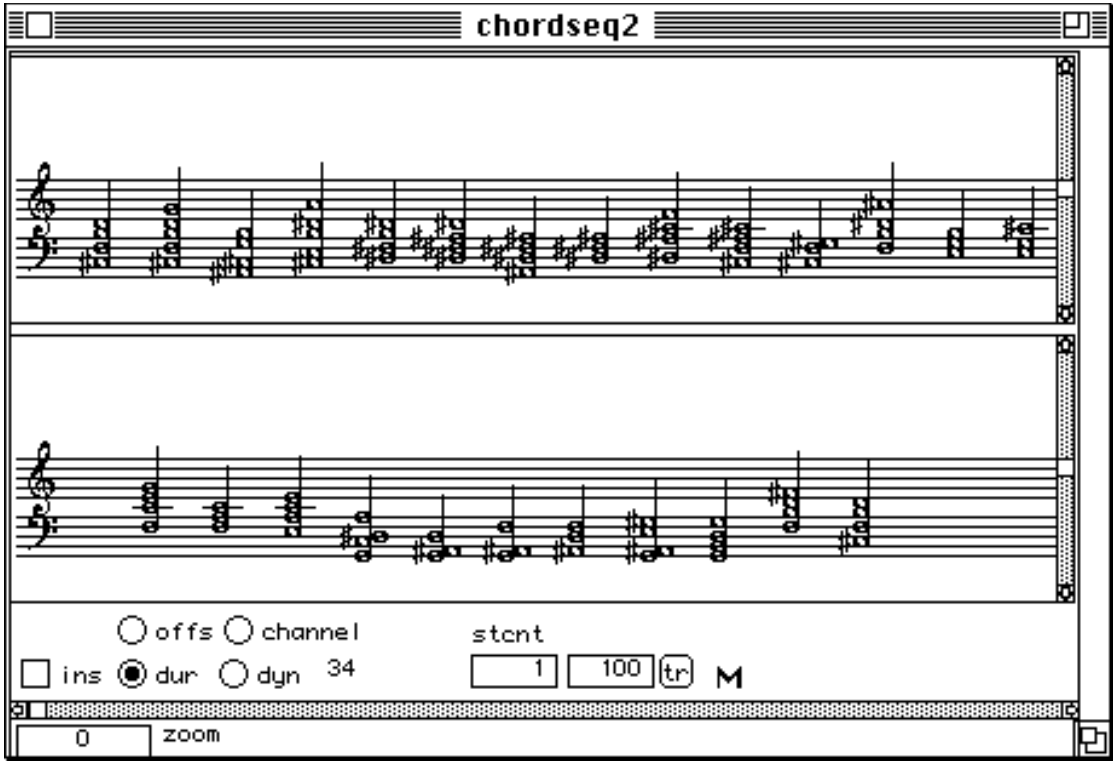


FIGURE 12 Inputting a chord sequence to **mutation**.

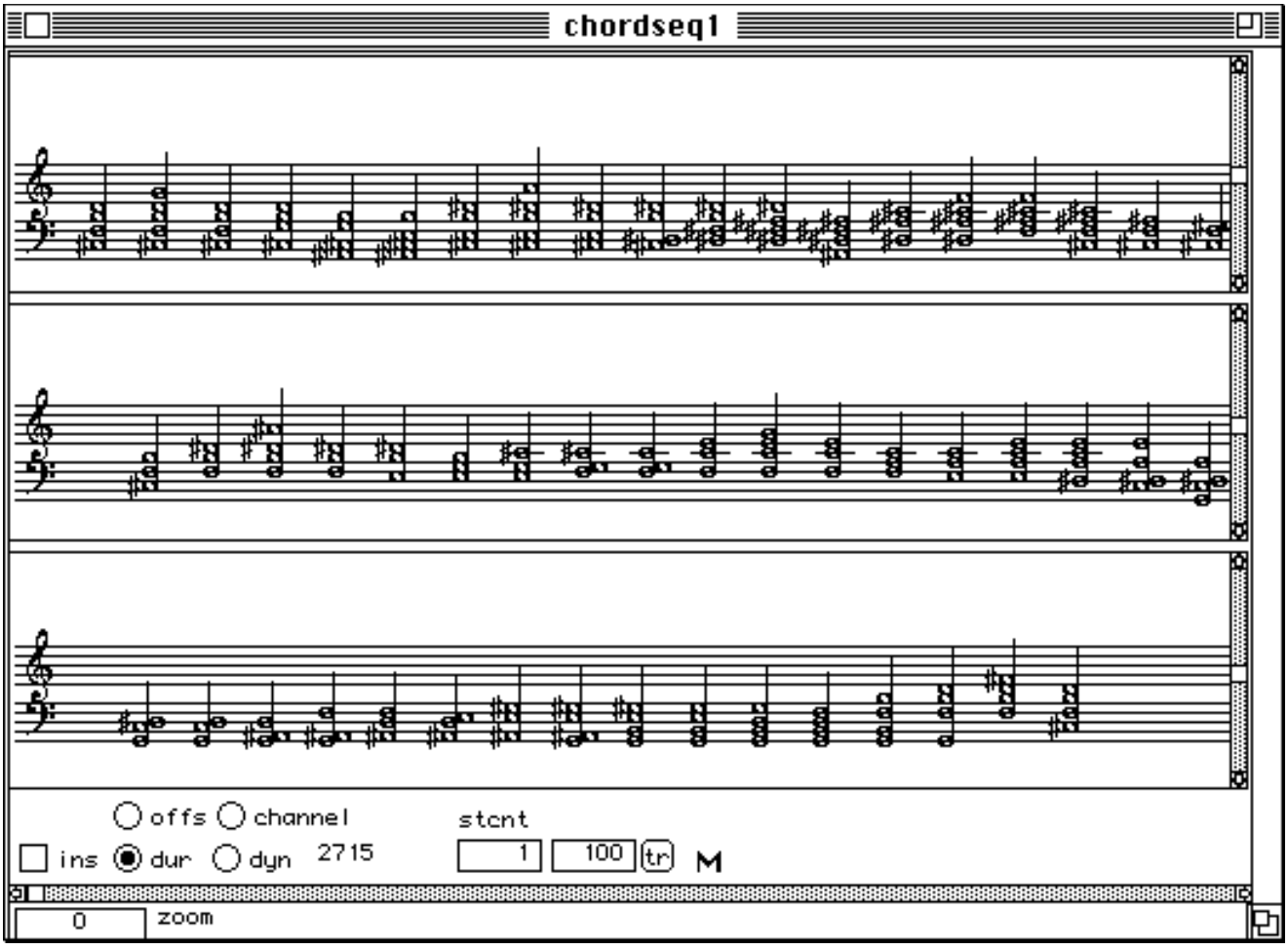
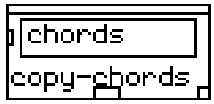


FIGURE 13 The result of mutation on a chord sequence

copy-chords



Syntax

lrepmusl::**copy-chords** chords
[function]

parameters

chords a chord (in midics or object form) or a list of same, or a **chord-line** object.

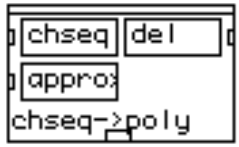
output

same type as input.

Description

Deep copies a chord or chord list or chord sequence. Very useful to overcome some of PatchWork board-effects on chords (i.e. editing a chord inside some editor causes a change in an other editor...)

chseq->poly



Syntax

lrepmusl::**chseq->poly** chseq del approx
[function]

parameters

chseq	a list of list of midics, or a list of **chord-objects** or a **chord-line** object.
del	positive integer, defines the time interval between two chords.
approx	integer (1, 2, 4, 8) tells the approximation used for finding common notes.

output

A list of chord objects suitable for input to a chordseq module.

Description

Changes a sequence of chords in a polyphony where common notes between two chords are changed into a single sustained note (harmonic link).

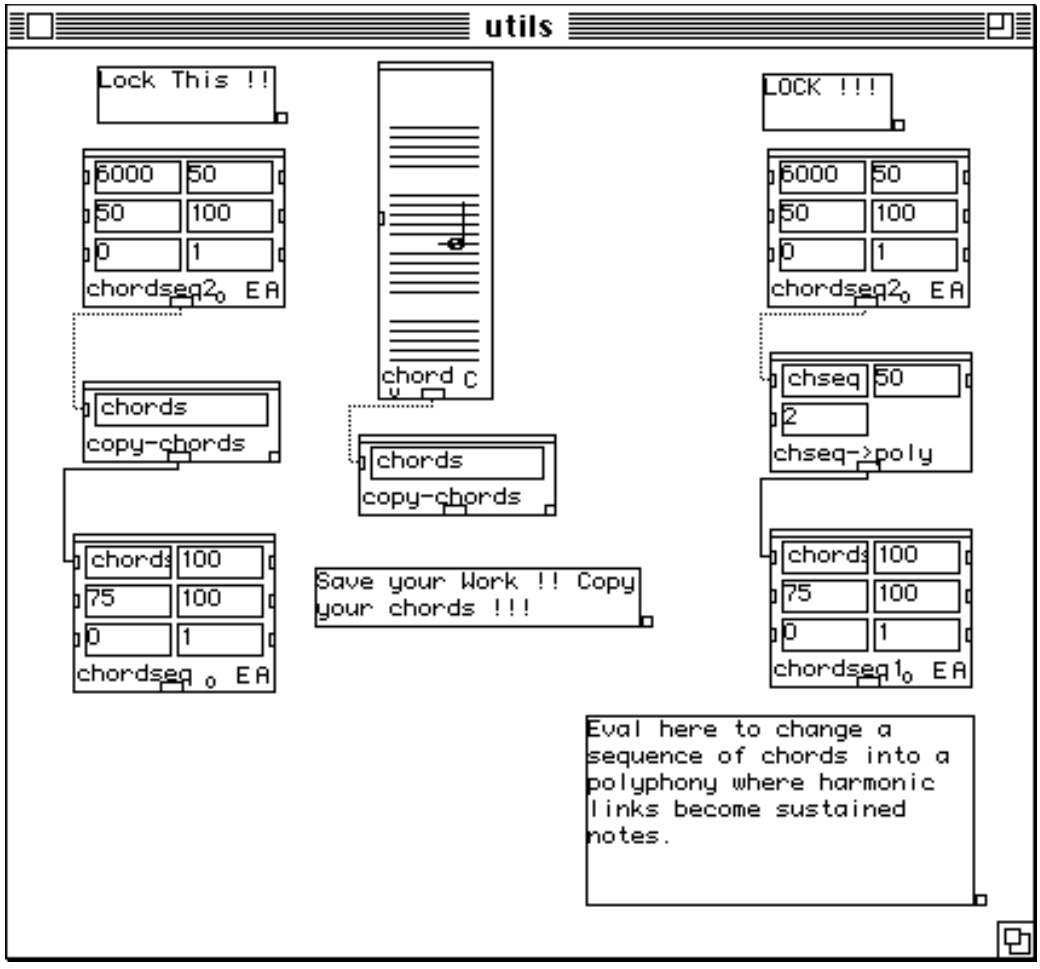


FIGURE 14 The tutorial window for **copy-chords** and **chseq->poly**.

The Metrics Modulation Menu

feuillete

Syntax

screamer::**feuillete** imp timp puls tpuls mes tmes vit npuls

[function]

parameters

imp	integer or ratio, impulsion (1/16 = sixteenth note, 1/12 a triplet unit etc.)
timp	integer, impulsion tempo (120 means 120 impulses in a mn)
puls	integer or ratio, pulsation (1/4 = quarter note, 1/8 = eighth note etc.)
tpuls	integer, pulsation tempo (60 means 60 pulsation in a mn)
mes	integer or ratio, measure signature (3/4 means 3 quarter notes)
tmes	integer, measure tempo (20 means 20 measures in a mn)
vit	integer or ratio, number of subdivision of the pulsation (3 : triplet, 2/3: triplets with notes linked 2 by 2)
npuls	integer, number of pulsation in a measure, an alternative to *mes* parameter

output

a c-measure-line object to be connected to a rtm box. All the solutions to the constraint system are put one after the other.

Description

Builds a series of measures that obey to some constraints on metrics structure.

The metrics structure is defined with 3 levels : the measure (a group of pulsations), the pulsation (the unit denotated by the measure signature's denominator), the impulsion (the subdivision of the pulsation, i.e. triplets inside quarter notes in a 4/4 measure).

All the parameters can take a value of -1 which means : UNDEFINED. Generally you specify only some parameters, put -1 in the others. This defines a constraint system that is solved for you by **feuillete**.

All the parameter can take a list instead of a single value. A list (v1 v2 ... vn) means that the considered parameter can take any value among v1,v2,...,vn.

All the parameters can take a list of the form (b v1 v2). This means the considered parameter can take all the values BETWEEN v1 and v2.

You can specify strange values like 5/16 for the pulsation. This means that there is a first level of WRITTEN pulsation which is the quarter note (1/4), subdivided into 4 smaller unit (sixteenth notes). The smaller units are linked 5 by 5 (5/16) which lets you hear another pulsation. This is combinable with any impulsion speed (i.e. you can put triplets in that perceived pulsation).

This kind of manipulation can be very complex but you still have a precise control over what you are building. It is very easy to generate for instance metrics modulation à la Carter. This module is inspired by Francois Nicolas paper : "Le feuillete du tempo" thus the name. This module uses the Constraint Solver 'screamer' by J.F. Siskind and D.A. McAllester from Univ. of Pennsylvania and MIT.

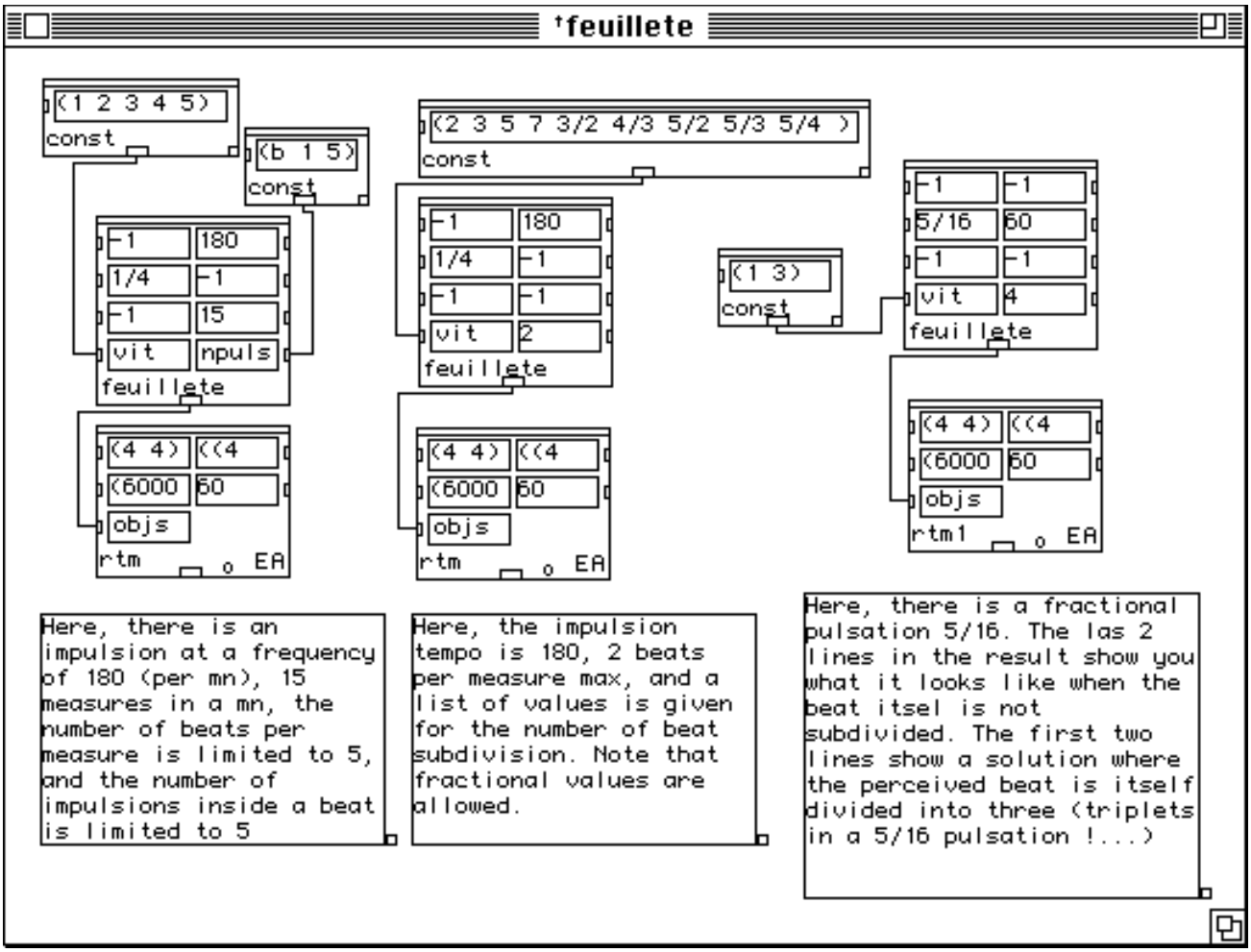


FIGURE 15 The tutorial window for **feuillete**.

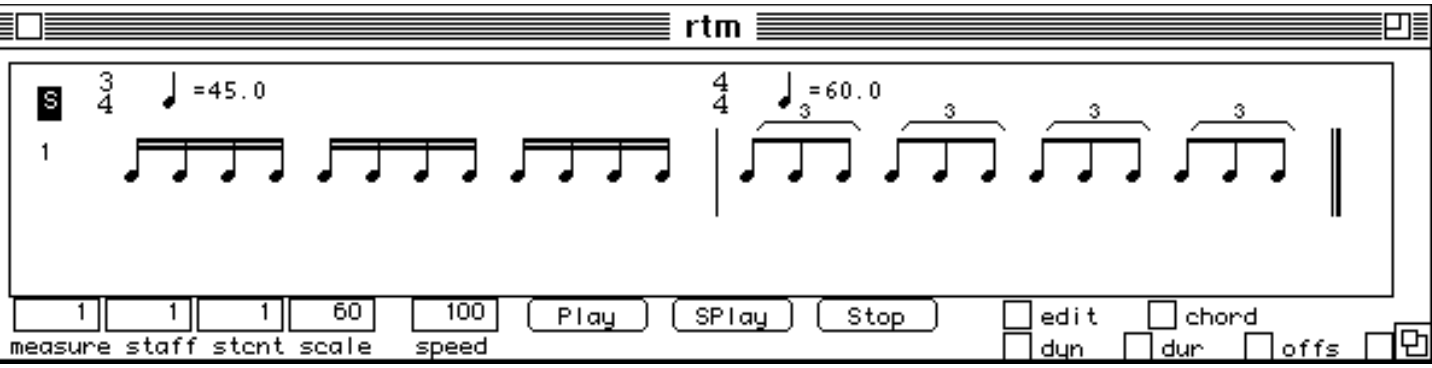


FIGURE 16 The first result (from left to right)

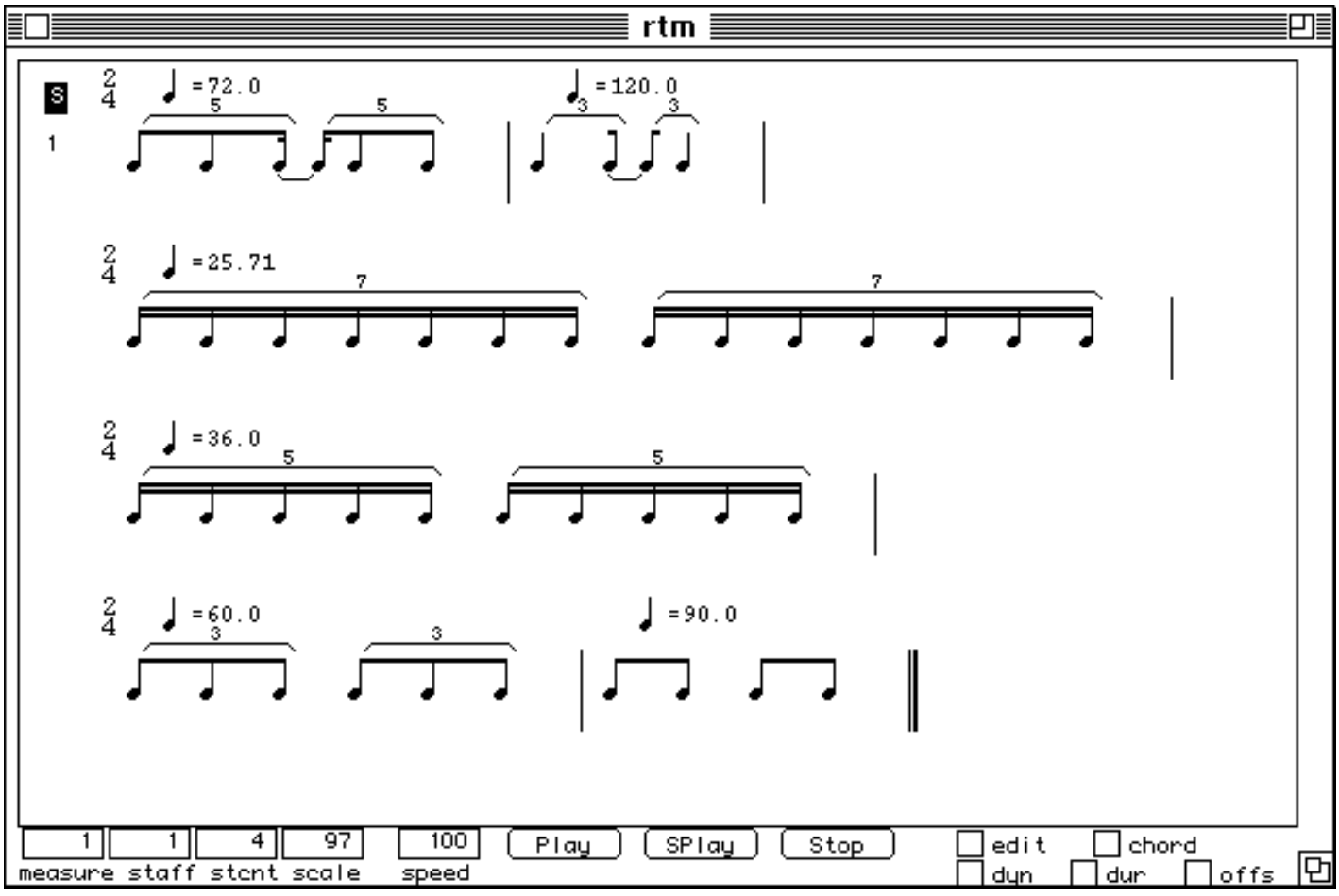


FIGURE 17 The second result (from left to right)

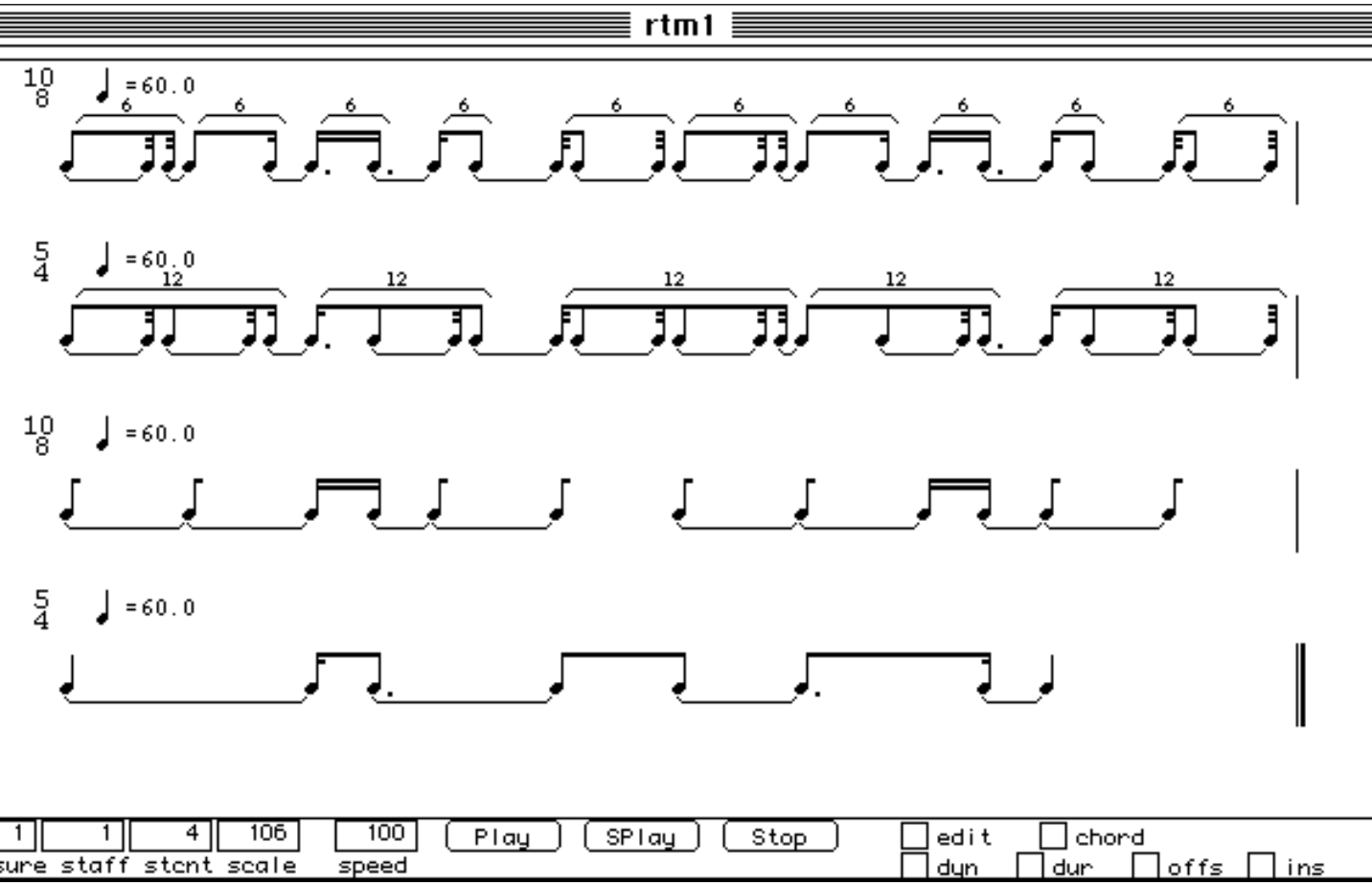
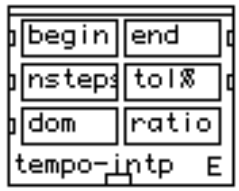


FIGURE 18 The third result (from left to right)

tempo-intp



Syntax

screamer::**tempo-intp** begin end nsteps tol% dom ratio &optional sol

[function]

parameters

begin	number, initial tempo
end	number, final tempo
nsteps	integer, number of interpolation steps
tol%	integer, allowed deviation when reaching the final tempo.
dom	integer, all ratios whose num and denum are smaller or equal to dom may be used.
ratio	menu, 'any' means any ratio will do, '=' means all ratios must be equal, '->' means ratios are increasing, '<-' means ratios are decreasing.
sol	(optional, menu) 'seq' means all solutions are concatenated, 'list' means' all solutions are put into a list.

output

A **c-measure-line** to be connected to a **rtm** box. If the sol parameter is 'list', a list of **c-measure-line** to be connected to a **poly-rtm** box. This option is also convenient to choose a solution among many, with the **posn-match** box.

Description

Builds a series of measures where the tempo changes smoothly from a starting value to an end value. At each step, a metrics modulation is performed. Typical subdivisions of the beat (impulsions) are computed to optimize the modulation. A set of ratios is used to pass from a measure to an other. You have control over these ratios : they can be always the same, or increasing, or decreasing, or in any order.

You can specify a domain for the ratio. *dom*=3 means, 1, 2, 3, 1/2, 1/3, 2/3, 3/2 are allowed. The more ratios allowed, the more solutions.

tempo-intp yields all the possible solutions in the constraint system specified by the parameter values. The solutions are concatenated in a **measure-line** or gathered into a list, depending on the parameter *sol*.

This module uses the Constraint Solver 'Screamer' by J.F. Siskind and D.A. McAllester from Univ. of Pennsylvania and MIT.

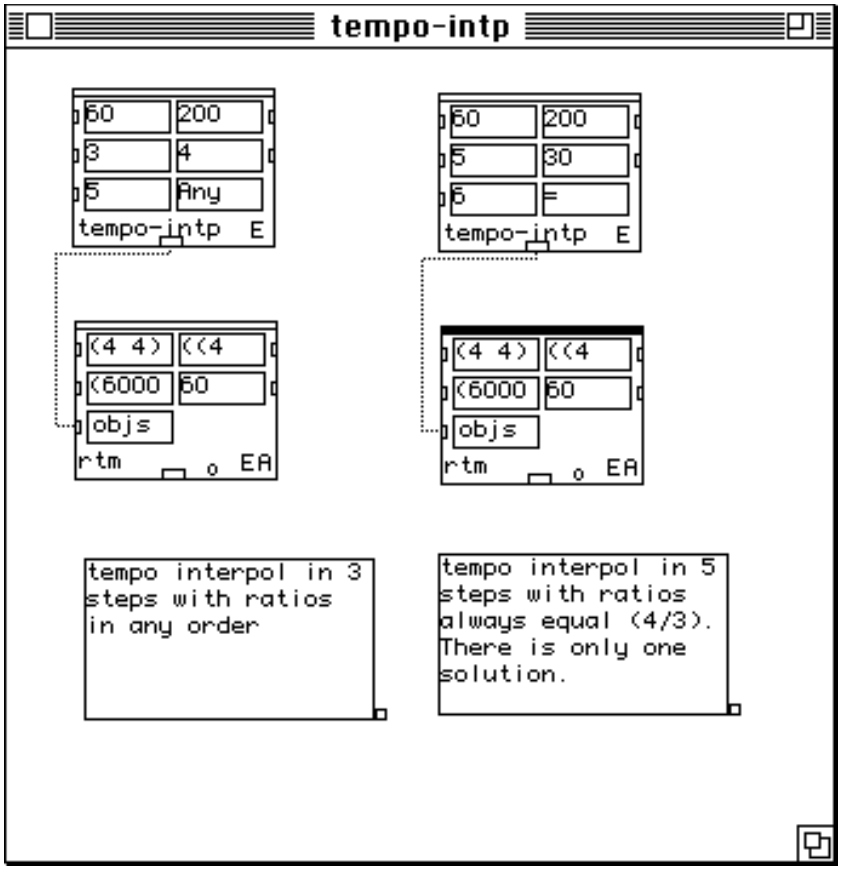


FIGURE 19 The tutorial window for **tempo-intp**

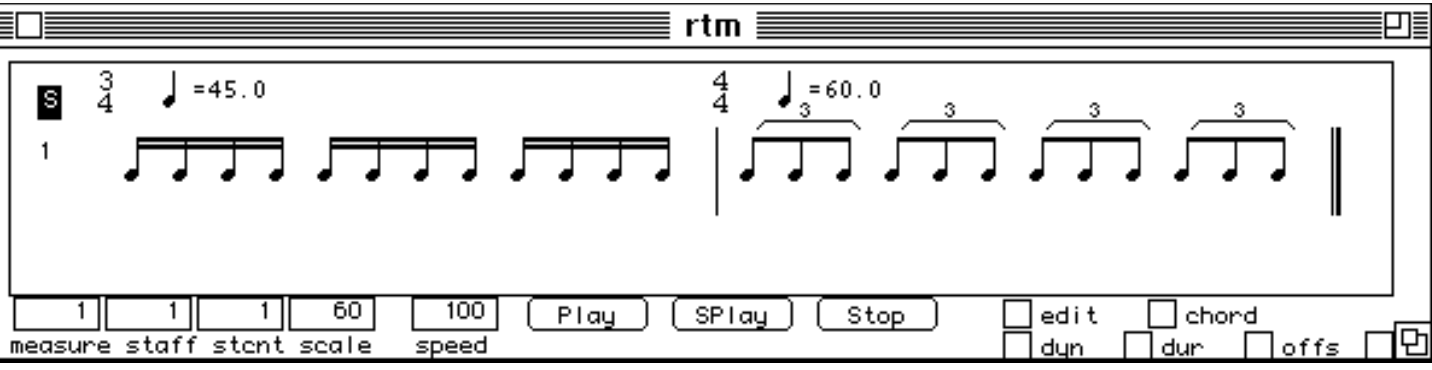


FIGURE 20 The first result (on the left)

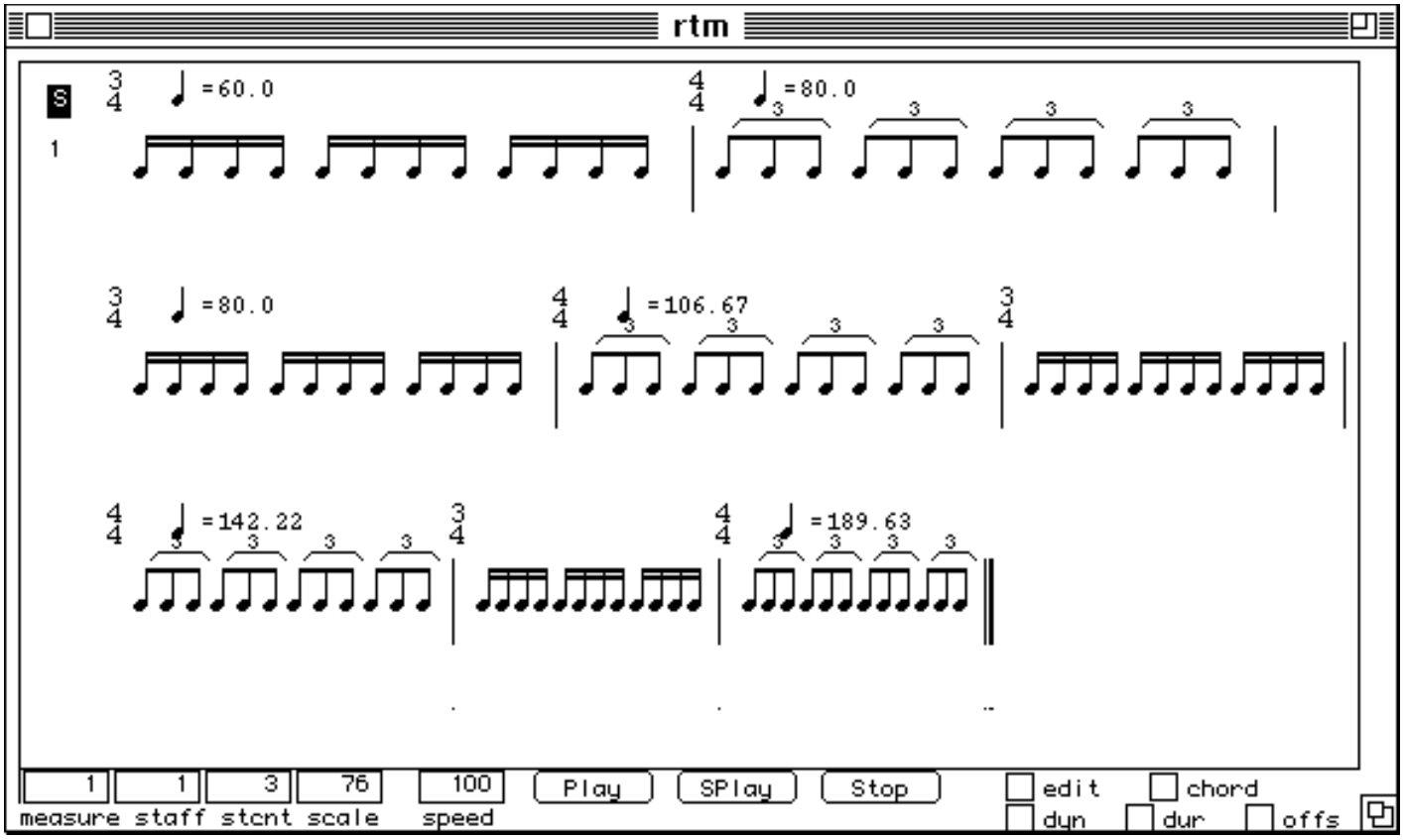


FIGURE 21 The second result (right)

The Cribles Menu

lc

Syntax

lrepmusl::**lc** prog-lc
[function]

binary operators

+	union
-	intersection
*	sieve composition
/	set difference
//	set symetrical difference

unary operators

c (x)	complementary sieve of the sieve 'x'
d(i1 i2 .. in)	defines an arbitrary sieve (i1 i2 ... in) with i1,i2... increasing integers
a(s b e)	defines a random sieve with step close to 'a', between values 'b' and 'e'
e <lisp form>	evaluates <lisp form>

examples : c = e (append (c1) (reverse (c1)) computes a palindrome from the sieve c1 and puts it into c. If you use sieve-symbols in <lisp form> put them between parentheses (e.g. (c1)).

p(s c1 c2 ... cn) where 's' is a symbol, 'c1'...'cn' are previously defined sieves. Computes a set partition of the set c1 U c2 U ... cn. Then the subsets are put in symbols built from 's'.

Example : after evaluating p(x c1 c2 c3), the symbol x1 (resp. x2, x3) is set to contain the element of c1 (resp. c2 c3) that are not elements of the 2 other sets. The symbol x12 contains elements common to c1 and c2 but not members of c3. x13 and x23 follow the same model. x123 is the intersection of the three sets.

parameters

prog-lc	the output of a **text-win** box

output

nil

Description

Computes a set of sieves (cribles) from a set of sieve expressions contained in a **text-win** box connected to it.

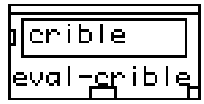
A sieve is a list of increasing positive integers. See the tutorial for examples of the language (**lc**) used for writing sieve expressions. Once evaluated, all the symbols that appear on the left side of the '=' operator (e.g. c1 in the expression 'c1 = c2 + c3') inside the **text-win** are defined and can be used in the eval-crible, crible-list and crible-rtm modules, in the *crible* parameter.

simple sieve : (step offset begin end)

example : c = (2 0 0 8) defines a sieve with a period 2 between 0 and 8: (0 2 4 6 8)

c = (2 1 4 10) defines (5 7 9).

eval-crible



Syntax

lrepmusl::**eval**-crible crible
[function]

parameters

crible a symbol or a list of symbols

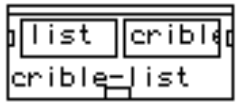
output

a sieve (a list of increasing integers) or a list of sieve

Description

Evaluates a symbol or a list of symbols defined with the **lc** box.

crible-list



Syntax

lrepmusl::**crible-list** list crible
[function]

parameters

crible a symbol or a list of symbols defined with a **lc** box
list a list

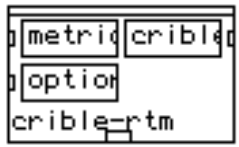
output

a list.

Description

Apply a sieve defined with the **lc** box to any list.

crible-rtm



Syntax

lrepmusl::**crible-rtm** metrique crible option

[function]

parameters

metrique	a **c-measure-line** (output from a **rtm** box)
crible	a symbol or a list of symbols defined with a **lc** box
option	menu, 'silence' means impulsions ignored by the sieve are made silent, 'liaison' means a selected impulsion is linked to following until next selected impulsion

output

a **c-measure-line** or a list of **c-measure line**, depending on the *crible* parameter. Connect to a **rtm** or **poly-rtm** depending on the type of output.

Description

Apply a sieve defined with the **lc** box to a metric/rhythmic structure.

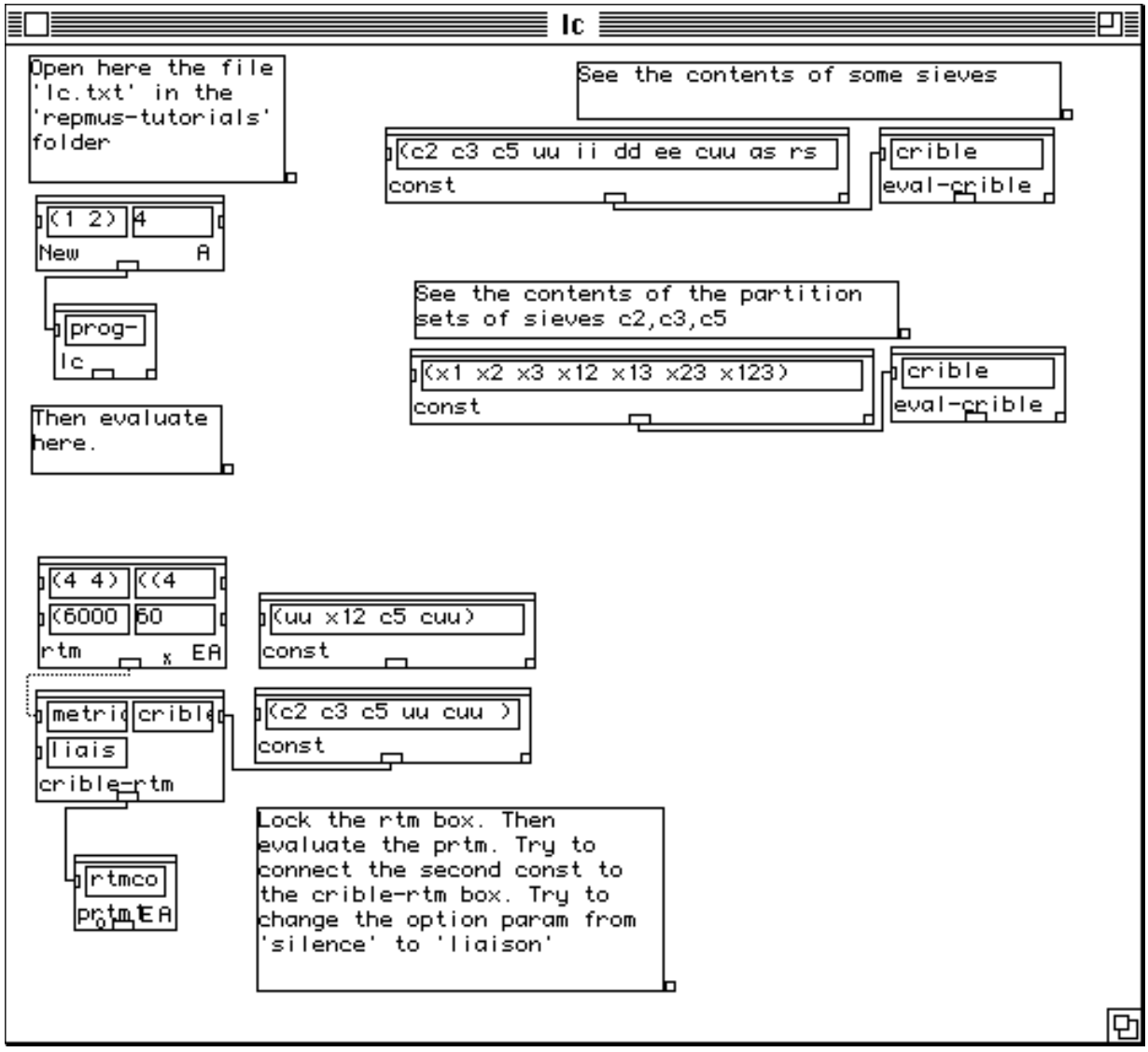


FIGURE 22 The tutorial box for **lc**, **crible-rtm**, **eval-crible**

The lc language

```
;;; the lc language
;;; define and manipulate sieves
;;; then use them on pitches or rythm
;;; always begin a line with the symbol '%'
;;; define simple sieves with period 2, 3, 5 between 0 and 100
% c2 = (2 0 0 30)
% c3 = (3 0 0 30)
% c5 = (5 0 0 30)
;;; define simple sieve with period 7, offset 2, between 2 and 16
% c7 = (7 2 2 16)
;;; define the union of c2, c3, c5
% uu = c2 + c3 + c5
;;; define the intersection of c7 and uu
% ii = c7 - uu
;;; take from c3 elements in the composition of c2 by c3
% dd = c3 / (c2 * c3)
;;; a set containing elements of c2 not in c3 and elements of c3 not in c2
% ee = c2 // c3
;;; the complementary sieve of uu
% cuu = c ( uu )
;;; an arbitrary sieve
% as = d(0 2 7 12 13 25 26 91)
;;; a random sieve with period 'close' to 4 between 10 and 100
% rs = a(4 10 100)
;;; evaluate a lisp form : take off the last element of uu
;; note that uu MUST be inside parentheses
% Lf = e (reverse (rest (reverse (uu))))
;;; compute a partition of a set
;;; defines the sets x1,x2,x3,x12,x13,x23,x123 which are all the non-intersecting subsets
;;; that can be made out of 3 sets.
% pp = p (x c2 c3 c5)
```

The text file used in the lc example

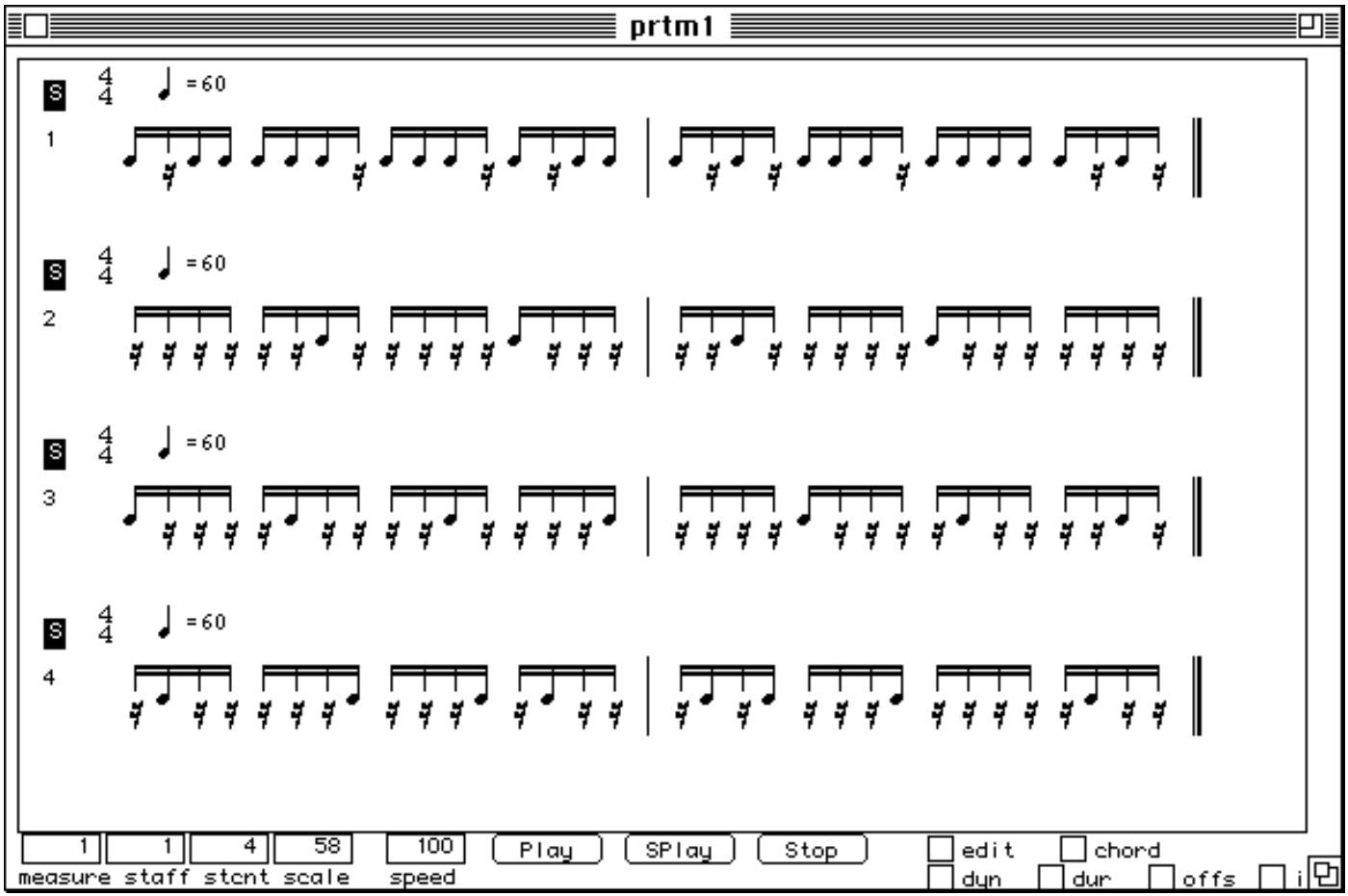


FIGURE 23 The result of **crible-rtm** on a stream of sixteenth notes.

The AudioSculpt to PatchWork Menu

as->pw

Syntax

lrepmusl::**as->pw** analyse vmin vmax delta mmin mmax approx npoly
[function]

parameters

analyse	connect here the output of a text-win module where you have read the analysis text file.
vmin, *vmax*	integers, amplitudes will be scaled between *vmin* and *vmax* velocities
delta	integer, events whose onset-time fall within a window of *delta* 1/100sec will be gathered into chords
mmin, *mmax*	midic values that define the allowed pitch range for the output.
approx	1,2,4, or 8. Micro-tonal approximation.
npoly	tries and reduce the polyphony to *npoly* notes at the same time by taking the louder partials first.

output

a list of chords to be connected to a **chordseq** module.

Description

Converts partials-analysis data, obtained within AudioSculpt by the 'Export Partials' command, in a suitable format for displaying and manipulating in PatchWork.

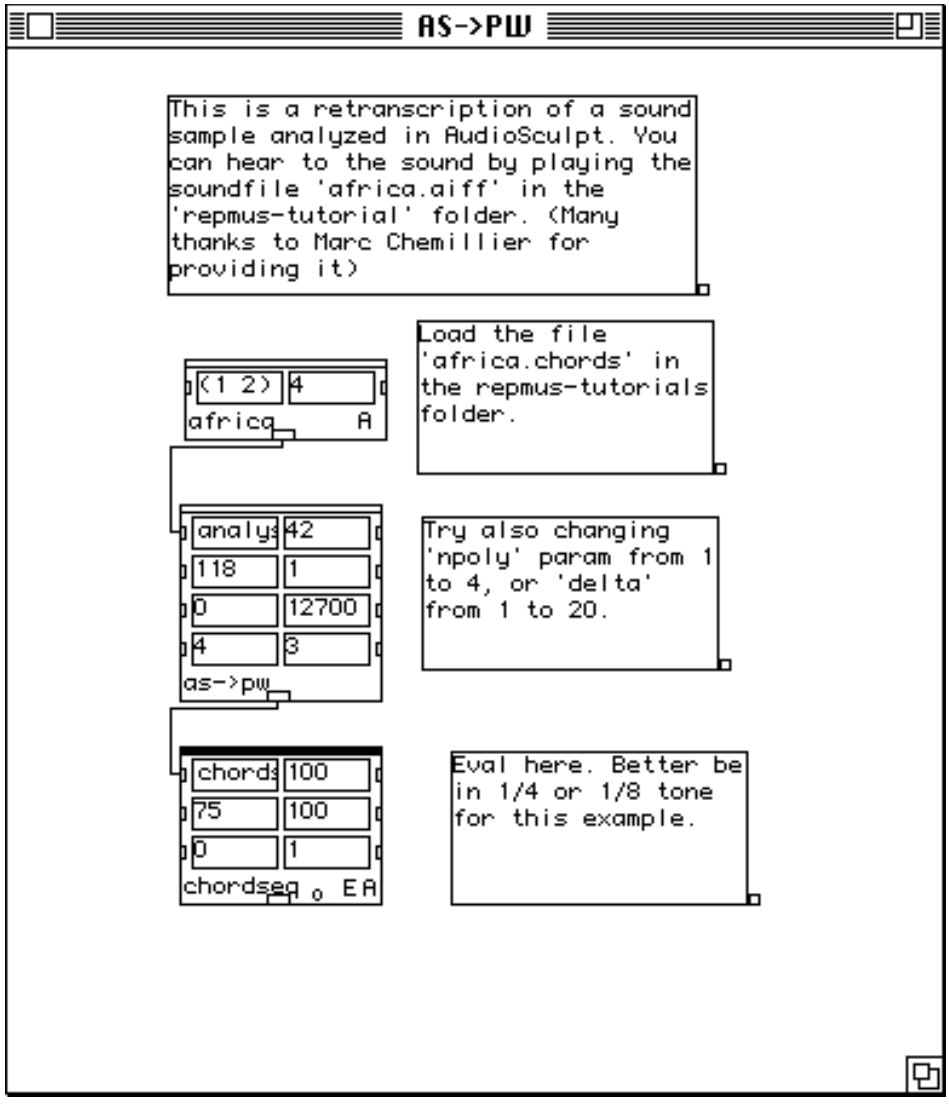


FIGURE 24 The tutorial for **as->pw** box.

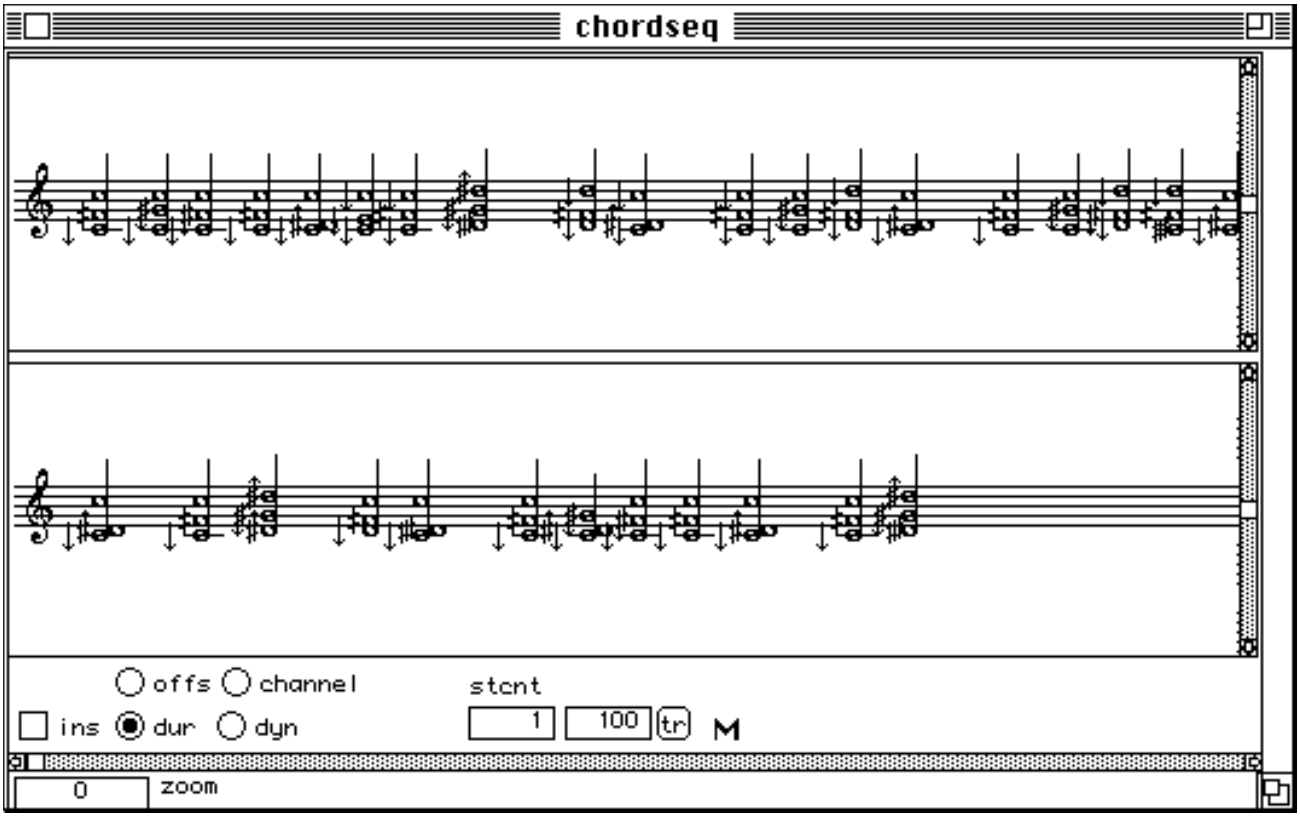


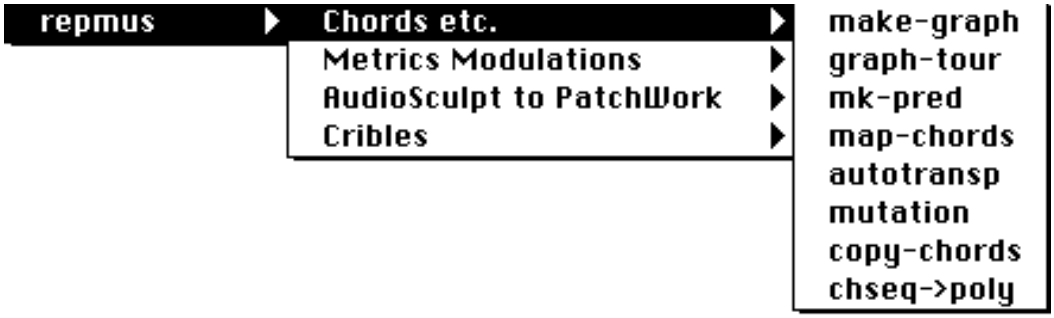
FIGURE 25 The output from **as->pw** box.

The beginning of the analysis file generated by AudioSculpt and opened in the **text-win** module :

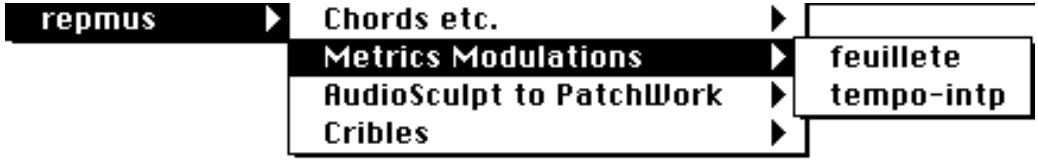
```
( PARTIALS 122
( POINTS 2
 0.016     258.236     2.711
 0.162     258.236     2.711)
( POINTS 2
 0.016     359.591     1.348
 0.162     359.591     1.348)
( POINTS 2
 0.016     520.252     2.131
 0.162     520.252     2.131)
( POINTS 2
 0.016     635.026     -16.256
 0.162     635.026     -16.256)
( POINTS 2
 0.168     259.297     -4.472
 0.273     259.297     -4.472)
( POINTS 2
 0.168     314.202     -5.420
 0.273     314.202     -5.420)
( POINTS 2
 0.168     408.783     6.391
 0.273     408.783     6.391)
```

The RepMus Menus

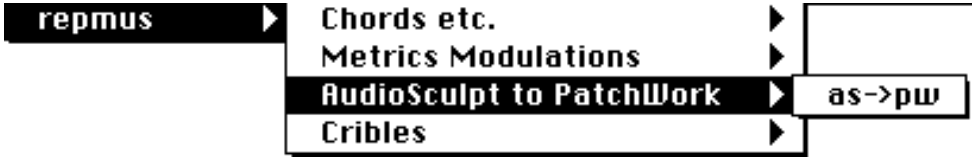
The Chords etc. Menu



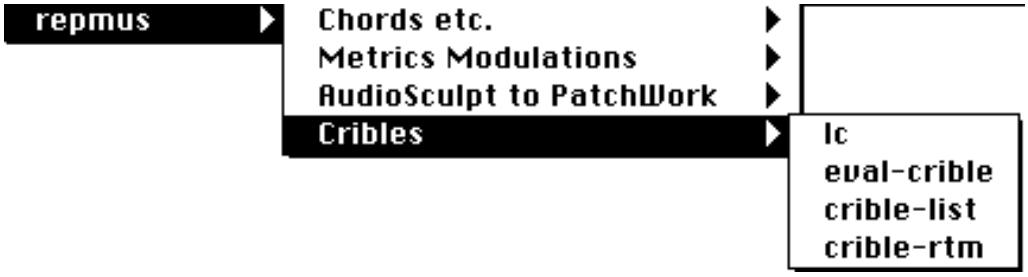
The Metrics Modulation Menu



The AudioSculpt to PatchWork Menu



The Cribles Menu



Index