PWGL LIBRARY

Jacopo Baboni Schilingi

JBS-CMI

jbs-cmi
(v 1.0)

January 18, 2010

Contents

1 Start-Here

This documentation has been written by Jacopo Baboni Schilingi and Julien Vincenot. This is a library dedicated to useful and personal functions inside PWGL software.

(1) Write-entities

(2) Pitch

(3) Matrix

(4) Special-combinations

(5) Grouping

(6) Utilities

The name of this library derives from an old project of research called 'Composizione per Modelli Interattivi' that I started at the end of my studies in Milan, then continued at Ircam and then at Tempo Reale.
This collection is not at all homogeneous. It is a personal library that collects many functions in several domains.
It seems that in the computer aided composition community, these functions have been quite used. That is why I decided to put them here. For those who are familiar with the ancient CMI, you will find almost all the functions. I removed the chapter STRUCTURE because, thanks to the developments in constraints domain done by Mikael Laurson, you can find in JBS-Constraints library a whole set of rules conceived to calculate rhythmical structures.
Many of these functions are related to the Hyper-Systemic theory. For those who might be interested, please refer to 'La musique Hyper-Systemique', Edition Mix. Paris 2007, by Jacopo Baboni Schilingi.
In future, you can freely download the always last upgraded version on my web site www.baboni-schilingi.com Please, if you find some bugs or anomalies, write directly to me at jbs@baboni-schilingi.com

2 0-Write-Music-Entities

2.1 0-Write-Entities

This part of the library is devoted to two tools for writing inside the Score-Editor using rhythm notation, pitch notation, grace note and expressions.

2.2 1-Write-Music-Entity

1.1 - WRITE-MUSIC-ENTITY
This function [1] allows you to write in a simple manner a rhythmical sequence (using rhythm tree structures) synchronized with pitches and music expressions.

In [a] you set the rhythm tree structure. In this example we have (1 1 1 1), that means that the measure will be divided in 4 equal pulses.
In [b] you set the corresponding pitches.
ATTENTION : If the number of pitches is smaller than the number of pulses, the last pitch will be repeated in order to get the same length than pulses. If the number of pulses is smaller than the number of pitches, the last pulse will be repeated to get the same length than the pitches. Please use the two PWGL-SWITCHES [2] and [3] to change either the number of pulses or the number of pitches.
In [c] you can define which music expression you want to be synchronized with pulses and pitches. Syntactically the expressions have to be in parenthesis like this (:mf) or (:accent), but also like this ((:accent :crescendo1) (:crescendo1) (:crescendo1) (:crescendo1 :sf))
(Please read the PWGL Help and the 02-ENP-Constructor tutorial in order to learn how many ENP expressions do exist.)
ATTENTION : If the number of expressions is smaller than the number of pulses or of pitches the last expression will be repeated in order to get the same length of pitches.
PLEASE evaluate the Score-Editor [6] in order to see the possible results. Use also the PWGL-SWITCH [4] and [5] in order to change the time signature (nominator and divisor).

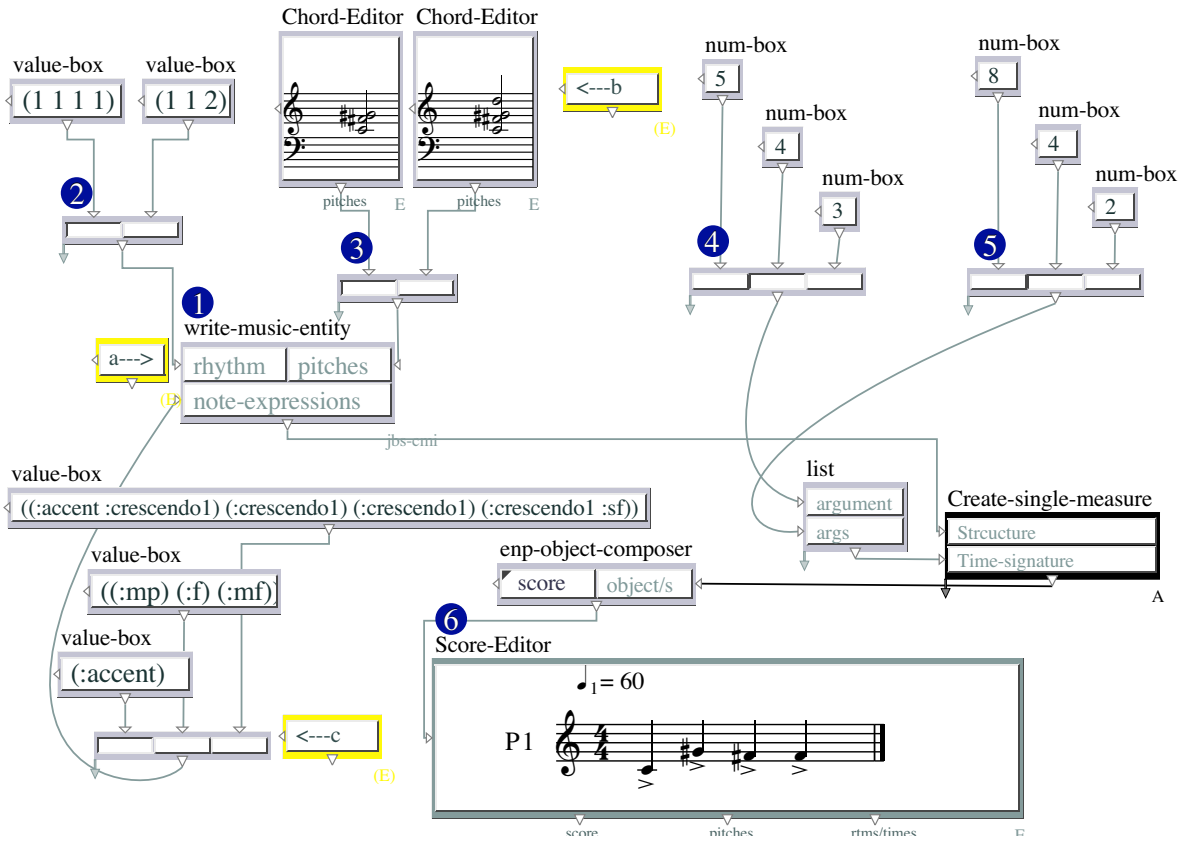

Figure 1: 1-1-write-music-entity

2.3 1-Write-Music-Entity

1.1.1 - WRITE-MUSIC-ENTITY
Here is a second example to study a little more this function [1]. As you know it allows you to write in a simple manner a rhythmical sequence (using rhythm tree structures) synchronized with pitches and music expressions.
Open the 'Pitch' abstraction [a] and look how I generate some pitch sequences. These sequences will define the rest of this example. (To understand it better, see also the JBS-Profile tutorial 5.4).
In [b] I generate some pulses sequences. Open the 'Rhythm' abstraction to understand this better. (Please look also at the JBS-Constraints tutorial 1.07.02)
In [c] I set the time signature nominators.
In [d] I set the time signature divisors.
In the abstraction 'create-expressions' [e] I generate two different expressions, according to the kind of pitch intervals generated in [a]. If the interval between two notes is bigger than a major third, then I ask for a slur, otherwise I ask for a staccato expression. Please open it in order to understand how it works.

The abstractions [2] and [3] are just ENP encoding for creating Score-Editor compatibility. I suggest you to study them in order to get more familiar with the ENP-Constructor. Please evaluate the Score-Editor [4] in order to obtain some results.

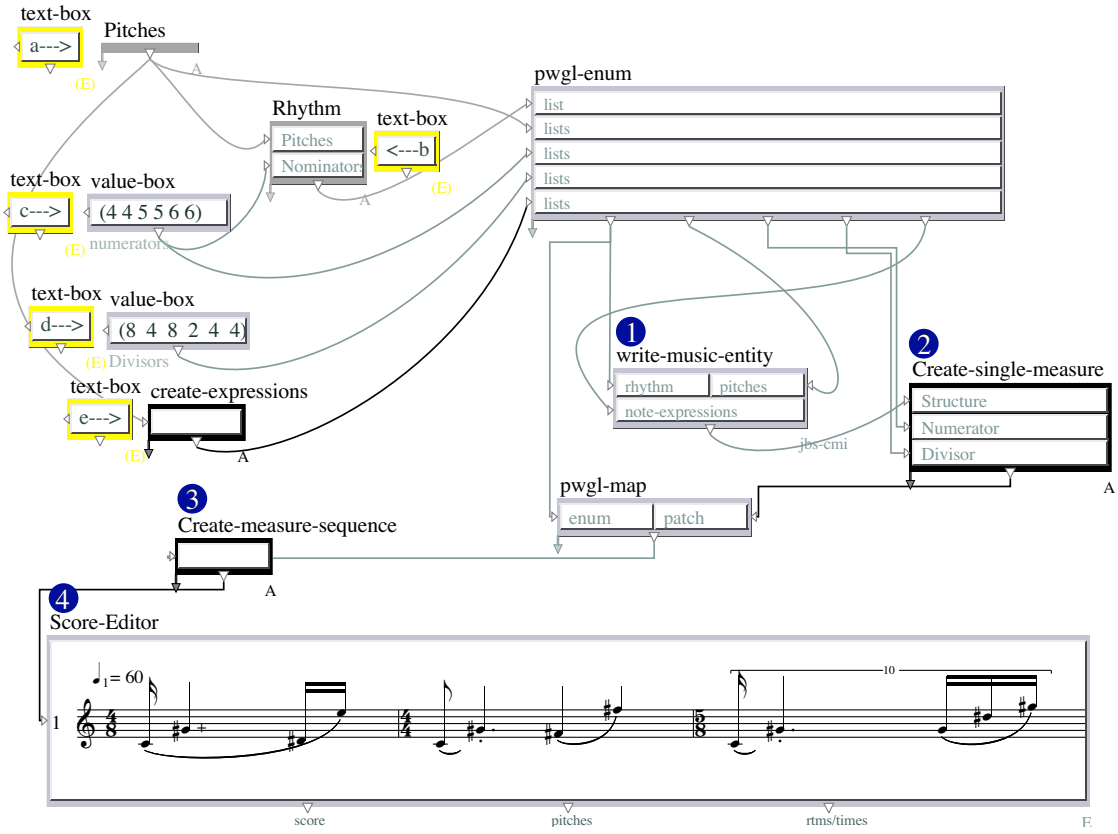

Figure 2: 1-1-1-write-music-entity

2.4 2-Write-Grace-Note-Music-Entity

1.2 - WRITE-GRACE-NOTE-MUSIC-ENTITY
(Sorry, not so elegant this one...)
This function [1] allows you to write in a simple manner a rhythmical sequence (using rhythm tree structures) synchronized with pitches and music expressions and inserting sequences of grace notes having music expressions for themselves.
In [a] you set the rhythm tree structure. Here we have (1 1 1 1), that means that the measure will be divided in 4 equal pulses.
In [b] you set the corresponding pitches.
ATTENTION : If the number of pitches is smaller than the number of pulses, the last pitch will be repeated in order to get the same length than pulses. If the number of pulses is smaller than the number of pitches, the last pulse will be repeated to get the

same length than the pitches. Please use the PWGL-SWITCH [2] to change the number of pulses.
In [c] you can define which music expression you want to be synchronized with pulses and pitches.
Syntactically the expressions have to be in parenthesis like this (:mf) or (:accent), but also like this: ((:accent :crescendo1) (:crescendo1) (:crescendo1) (:crescendo1 :sf)) (Please see the PWGL Help and the 02-ENP-Constructor patch in order to learn how many ENP expressions do exist.)
ATTENTION : If the number of expressions is smaller than the number of pulses or of pitches, the last expression will be repeated in order to get the same length of pitches.
In [d] you can enter either a flat list of pitches (in this case it will be considered as a single group of grace notes) or a list of lists of pitches (in this case each sublist will be considered as a separate group of grace notes).
In [e] you have to define where the grace notes will be appended. If you put 0, that means that the grace notes will appended to the nth 0 (the first) note you put in [a]. If you put 1 it will be in the second place, 2 in the third and so on.
In [4] you can define the grace note single group or a list of groups.
PLEASE use the four PWGL-SWITCH [3], [4], [5] and [6]: - with [3] you can decide single nth positions or groups of nth positions; - with [4] you can control if you have a single group of grace note or groups (a LIST of LISTS) of grace notes; - with [5] you can choose the NOTES expressions; - with [6] you can choose the GRACE-NOTES expressions.
NOTE THIS: - In [3] if you put a single number and in [4] you have set a list of groups, only the group corresponding to the nth in [3] will be chosen. - In [3] if you have chosen a group of nth and in [4] you have set only a group of grace notes, only this last one will be printed in the score in the first nth group set in [3].
- In [5] you have multiple choices (the behaviour for NOTE expressions). 1 - if you use a single expression like (:fff) this one will be propagated to all notes; 2 - if you use a flat list of expressions like (:sf :accent) the list will associate (:sf) to the first note, and (:accent) for all other notes; 3 - if you use a single list of list like ((:sf :accent)) the double expression (:sf :accent) will be propagated to all notes; 4 - if you use a list of separate sub lists like ((:p) (:fermata :accent)) and the length of notes is bigger, the last expression - in this case (:fermata :accent) - will be propagated till the last note; 5 - if you use a list of separate sublists like ((:p) (:f) (:mf) (:fermata :accent)) and the number of lists is equal to the length of notes, for each note you have a specific expression.
- In [6] you have multiple choices (the behaviour for GRACE-NOTE expressions)
ATTENTION : This behaviour is similar to NOTE expression but not exactly the same. 1 - if you use a single expression like (:fff) this one will be propagated to all grace-notes; 2 - if you use a single list of list like (:sf :accent) the double expression (:sf :accent) will be propagated to all grace-notes; 4 - if you use a list of separate sublists like ((:mf) (:accent :slur1)) and the length of notes is bigger than the length of expressions, the whole group of expressions will be propagated till the last grace-note.
ATTENTION : If the length of the sub-group grace-notes is smaller than the list of expressions, the list will stop accordingly to the length of the grace-note sub-group.
ATTENTION TOO : If the length of a sub-group is bigger than the length of the expres-

sion list, the last expression will be propagated to the last grace-note.
5 - if you use a list of separate sub-lists like ((:mf) (:accent) (:slur1 :crescendo1) (:crescendo1) (:crescendo1 :ff)) and the number of lists is equal to the length of grace-note list, for each grace-note you have a specific expression.

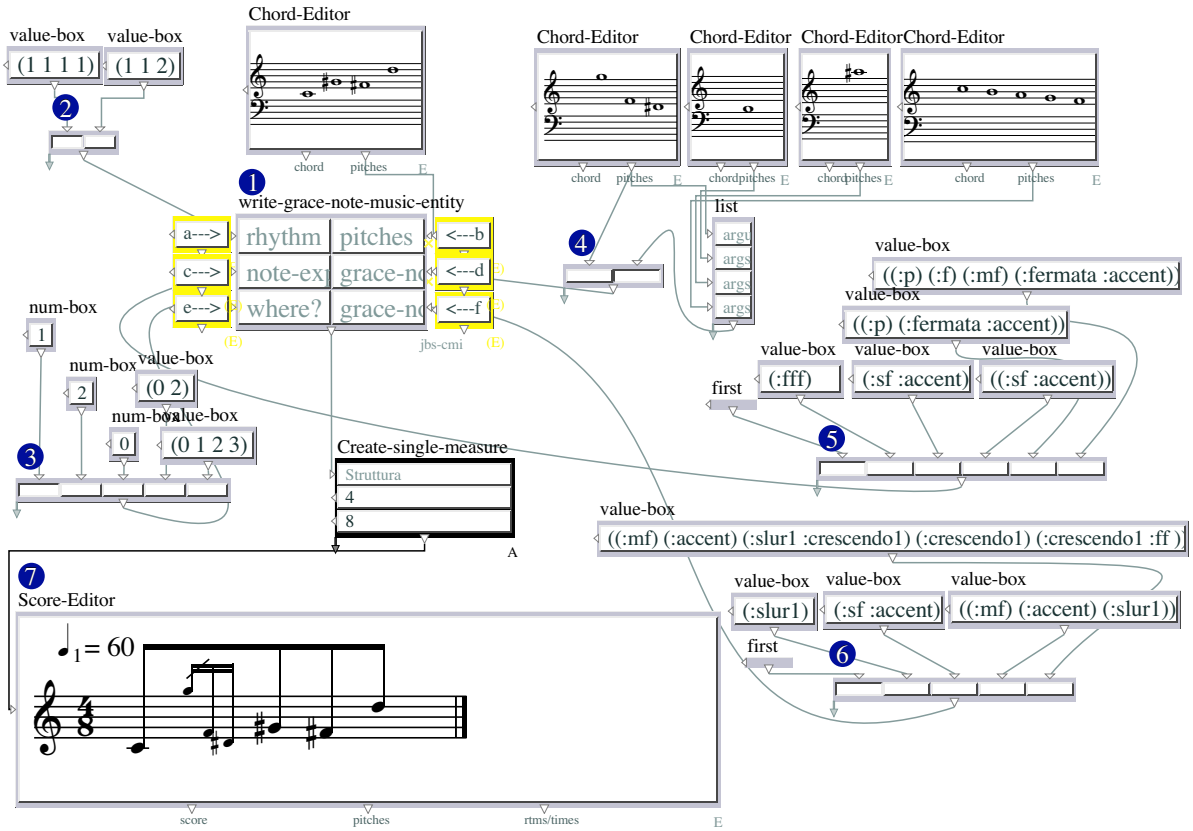

Figure 3: 1-2-write-grace-note-music-entity

2.5 1-Write-Grace-Note-Music-Entity

1.2.1 - WRITE-GRACE-NOTE-MUSIC-ENTITY
Here is a second example to study a little more this function [1]. As you know it allows you to write in a simple manner a rhythmical sequence (using rhythm tree structures) synchronized with pitches and music expressions, using also grace-note groups with expressions too.
Open the 'Pitch' abstraction [a] and look how I generate some pitch sequences. These sequences will define the rest of this example. (To better understand it, please see also the JBS-Profile tutorial 5.4).
In [b] I generate pulses sequences. Open it to better understand it, and please look at the JBS-Constraints tutorial 1.07.02.
In [c] I generate grace-note groups.

In [d] I set the indexes to choose where to put the grace-note groups.
In [e] I set the time signature nominators.
In [f] I set the time signature divisors.
The abstractions [3] and [4] are just ENP encoding for creating Score-Editor compatibility. I suggest you to study them in order to get more familiar with the ENP-Constructor. Please evaluate the Score-Editor [4] in order to obtain some results.

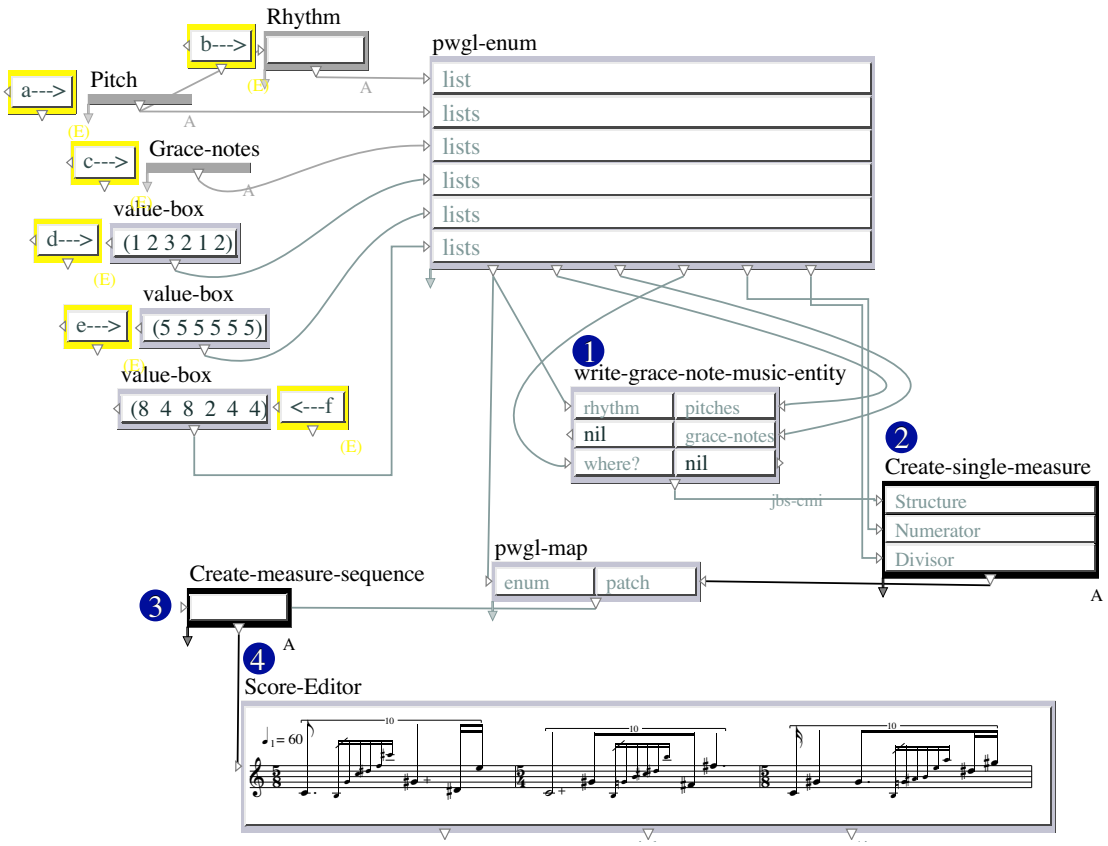

Figure 4: 1-2-1-write-grace-note-music-entity

2.6 3-Write-Grace-Note-Music-Entity

1.3 - WRITE-GRACE-NOTE-MUSIC-ENTITY
JUST ANOTHER COMPLETE EXAMPLE.
TRY TO STUDY IT.

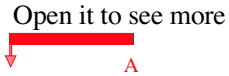

Figure 5: 1-3-write-grace-note-music-entity

3 0-Pitch

3.1 0-Pitch

This part of the library is devoted to two tools for controlling pitches.

3.2 1-Harmonic-Fields

2.1 - HARMONIC-FIELDS
This function [1] allows you to define as defvar variables some lists of chords and to recall them in three different ways.
First you have to define a set of chords.
Here is an example:
(defvar major-triads '((60 64 67) (61 65 68) (62 66 69) (63 67 70) (64 68 71) (65 69 72) (66 70 73) (67 71 74) (68 72 75) (69 73 76) (70 74 77) (71 75 78)))
and

(defvar my-clusters '((60 61 62 63 64) (61 62 63 64 65) (62 63 64 65 66) (63 64 65 66 67) (64 65 66 67 68) (65 66 67 68 69) (66 67 68 69 70) (67 68 69 70 71) (68 69 70 71 72) (69 70 71 72 73) (70 71 72 73 74) (71 72 73 74 75)))
Please open the Lisp-code-box [2]. The two variables have been evaluated when the patch was loaded, but you can modify and evaluate them again. To do it just select all the text, then type command-shift-E.
In [a] I just recall the whole classes of chords with their 12 transpositions. Please use the PWGL-SWITCH [3] and choose between the two classes of chords.
In [b] I set one class, the major triads, with a specific transposition value. This value belongs to the modulo 12 pitch result.
In [c] I set a specific order to recall some transpositions of the two classes of chords.
ATTENTION: I have already classified a lot of harmonic fields inside this library. These chords derive from E. Carter, I. Fedele, B. Ferneyhough, K. Stockhausen and myself. In order to know them, it is sufficient to read the doc of HARMONIC-FIELDS [1] inside a patch in PWGL.

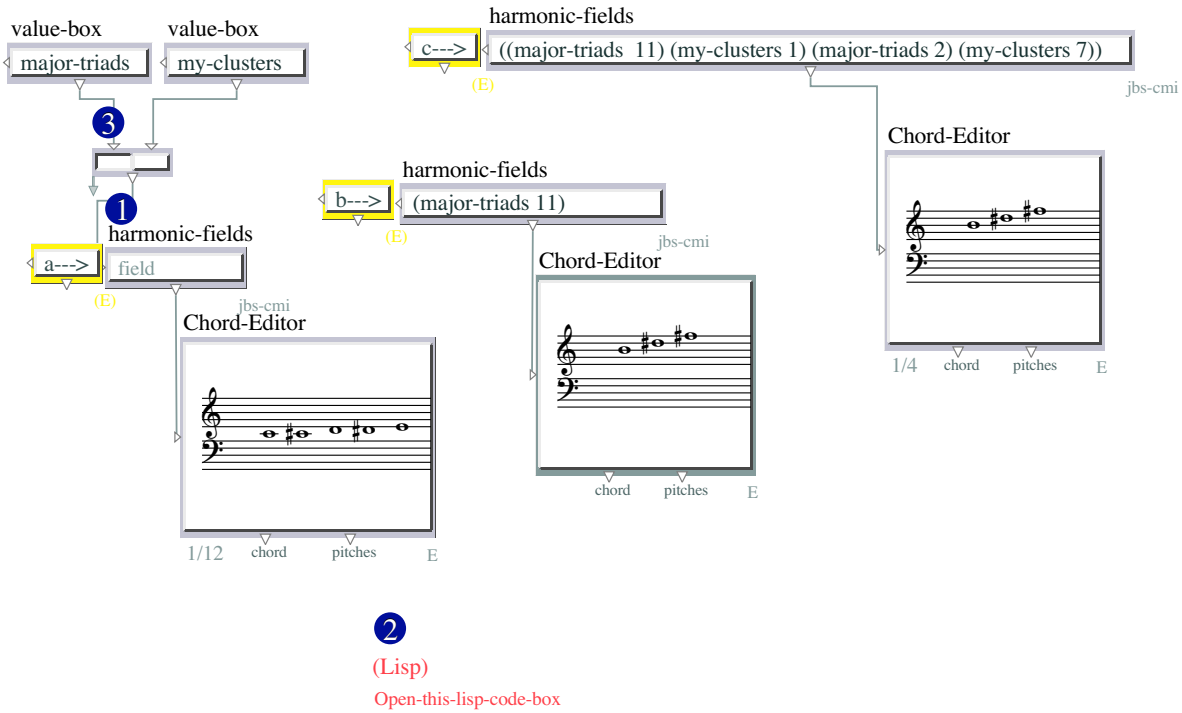

Figure 6: 2-1-harmonic-fields

3.3 2-Answer

2.2 - ANSWER
This function [1] is a simplified reproduction of the tonal answer of the fugue.
In [a] you put a melodic profile.
In [b] you define (as nth index) which note of the profile has to be considered as the dominant.
When you evaluate the ANSWER all the notes of the profile are transposed to the dominant, except the dominant, that is transposed on the tonic.

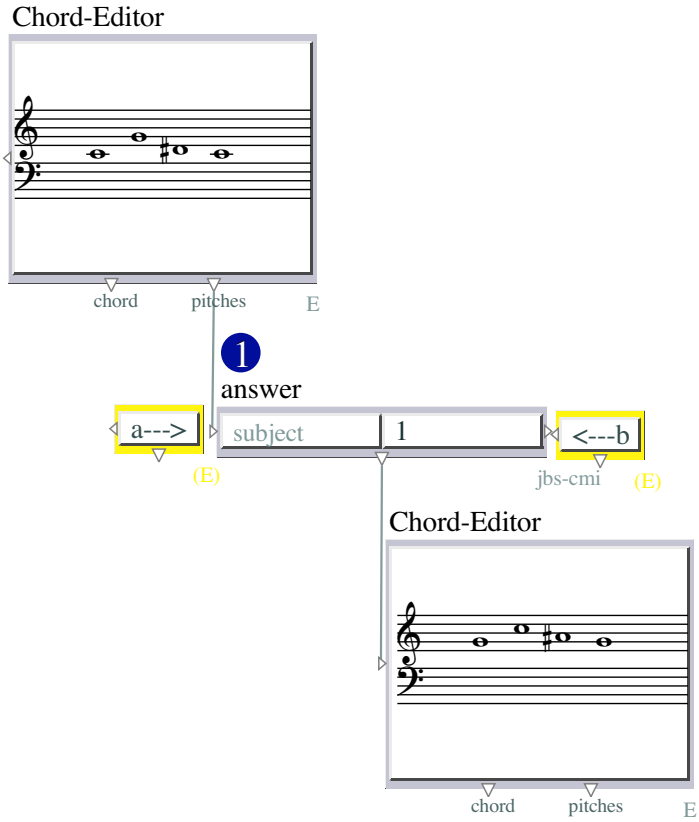

Figure 7: 2-2-answer

4 0-Matrix

4.1 0-Matrix

This part of the JBS-CMI library is devoted to the manipulation of matrices. By matrix I mean a list of lists of elements distributed in columns and rows. This distribution belongs to a given logic or tactic.

4.2 0-Numeric-Comment

4.2.1 1-Numeric-Comment

3.1.1 - NUMERIC-COMMENT
(Also know as Look-and-say sequence by John Horton Conway)
This function [1] reproduces the Look-and-say sequence as shown.
Put 1 in [a] and choose which level of recursion you want to use with [b].
To generate a member of the sequence from the previous member, read off the digits of the previous member, counting the number of digits in groups of the same digit.
For example:
1 is read off as 'one 1' or 11. 11 is read off as 'two 1's' or 21. 21 is read off as 'one 2, then one 1' or 1211. 1211 is read off as 'one 1, then one 2, then two 1' or 111221. 111221 is read off as 'three 1, then two 2, then one 1' or 312211.
In [c] I set a arithmetic series going from 1 to 10. Please evaluate the PRINT-LIST [2] in order to see the result.

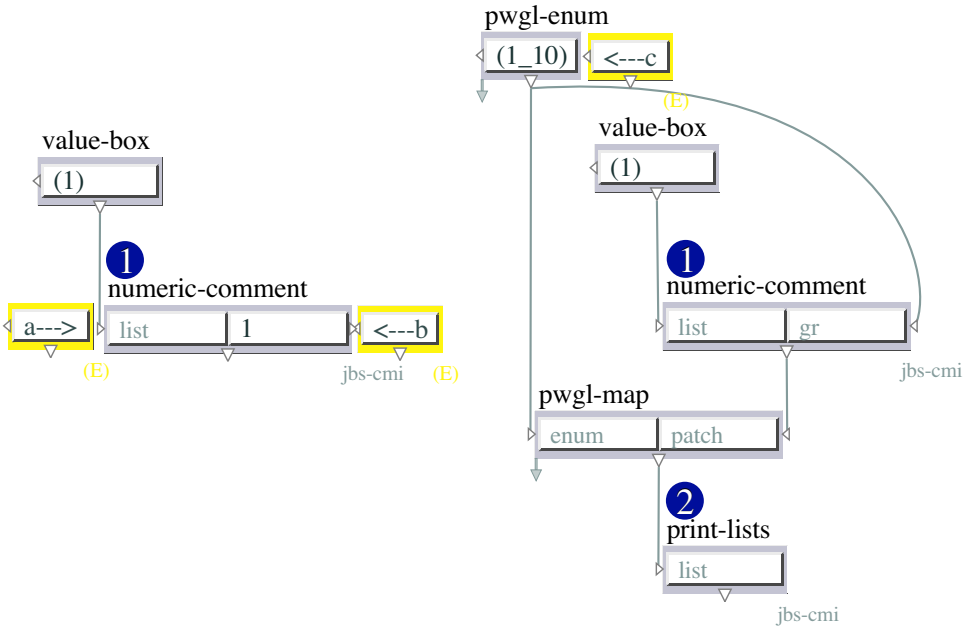

Figure 8: 3-1-1-numeric-comment

4.2.2 2-Numeric-Comment-Sort

3.1.2 - NUMERIC-COMMENT-SORT
This is a variation of the Look-and-say sequence by John Horton Conway.
Put 1 in [a] and choose which level of recursion you want to use with [b].
To generate a member of the sequence from the previous member, read off the digits of the previous member, counting the number of digits in groups of the same digit., BUT STARTING ALWAYS FROM THE SUM OF MORE LITTLE NUMBER TO THE BIGGER ONE.
1 is read off as 'one 1' or 11. 11 is read off as 'two 1' or 21. NOW SEE THE DIFFERENCE 21 is read off STARTING FROM THE MORE LITTLE NUMBER THAT IS 1 SO ' one 1' then 'one 2' or 1112. THEN AGAIN STARTING FROM THE MORE LITTLE : 1112 is read off as 'three 1', then 'one 2' or 3112. THEN AGAIN STARTING FROM THE MORE LITTLE : 3112 is read off as 'two 1', then 'one 2', then 'one 3' or 211213. THEN AGAIN STARTING FROM THE MORE LITTLE : 211213 is read off as 'three 1', then 'two 2', then 'two 3' or 312213. And so on.
A special property: After a while all sequences become the comment of themselves. Try to make the comment of this one : 2 1 3 2 2 3 1 4.

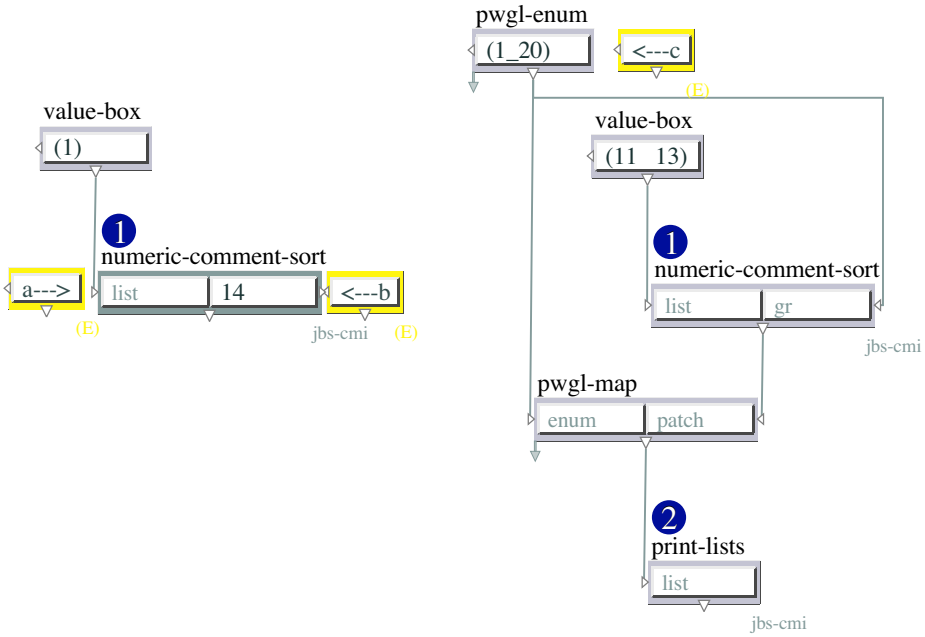

Figure 9: 3-1-2-numeric-comment-sort

4.3 0-Moving

4.3.1 1-All-to-1

3.2.1 - ALL-TO-ONE
This function [1] creates a linear interpolation between a list of numbers set in [a] and the number 1. When a number of the given list reaches 1, then it disappears.
If you put (10 7 5 2) as starting list, the result will be :
—> 10 7 5 2 —> 9 6 4 1 —> 8 5 3 —> 7 4 2 —> 6 3 1 —> 5 2 —> 4 1 —> 3 —> 2 —> 1
Please evaluate the PRINT-LIST [2] in order to see the result.
(This is an old function from Brian Ferneyhough.)

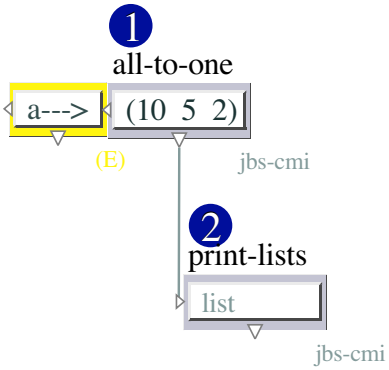

Figure 10: 3-2-1-all-to-1

4.3.2 2-All-to-X

3.2.2 - ALL-TO-X
This function [1] creates a linear interpolation between a list of numbers set in [a] and the number set in [b]. When a number of the given list reaches the value set in [b], then it disappears.

If you put (20 7 1) as starting list, the result will be :
—> 20 7 1 —> 19 8 2 —> 18 9 3 —> 17 10 4 —> 16 5 —> 15 6 —> 14 7 —> 13 8 —> 12 9 —> 11 10 —> 10
Please evaluate the PRINT-LIST [2] in order to see the result.

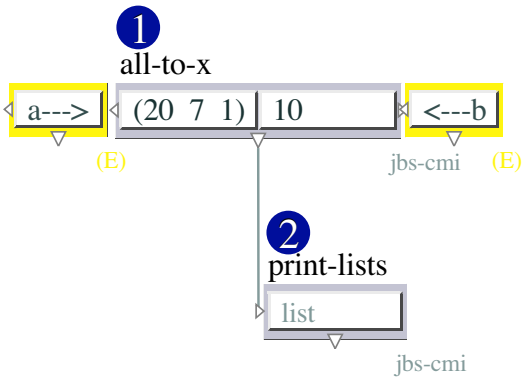

Figure 11: 3-2-2-all-to-x

4.4 0-Reading-Matrix

4.4.1 1-All-Reading-Matrix

3.3.1 - ALL-READING-MATRIX
This patch shows you all possible way of reading a matrix.
Please use the PWGL-SWITCH to choose which function you want to use, and evaluate the PRINT-MATRIX [2] in order to see the result.
Here is a very simple matrix :
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here are the results for each function in this patch: - RIGHT-DOWN-DIAGONAL-READING-MATRIX —> 1 —> 22 —> 333

- LEFT-UP-DIAGONAL-READING-MATRIX —> 333 —> 22 —> 1
- RIGHT-UP-DIAGONAL-READING-MATRIX —> 3 —> 22 —> 111
- LEFT-DOWN-DIAGONAL-READING-MATRIX —> 111 —> 22 —> 3
- DOWN-LEFT-DOWN-READING-MATRIX —> 1 —> 11 2 —> 111 22 3 —> 222 33 —> 333
- UP-RIGHT-UP-READING-MATRIX —> 333 —> 33 222 —> 3 22 111 —> 2 11 —> 1
- UP-LEFT-DOWN-READING-MATRIX —> 333 —> 222 33 —> 111 22 3 —> 11 2 —> 1
- DOWN-RIGHT-UP-READING-MATRIX —> 1 —> 2 11 —> 3 22 111 —> 33 222 —> 333
- ORDER-READING-MATRIX —> 1 11 111 —> 2 22 222 —> 3 33 333
- REVERSE-READING-MATRIX —> 3 33 333 —> 2 22 222 —> 1 11 111
- REVERSED-ORDER-READING-MATRIX —> 111 11 1 —> 222 22 2 —> 333 33 3
- REVERSED-ORDER-REVERSE-READING-MATRIX —> 333 33 3 —> 222 22 2 —> 111 11 1

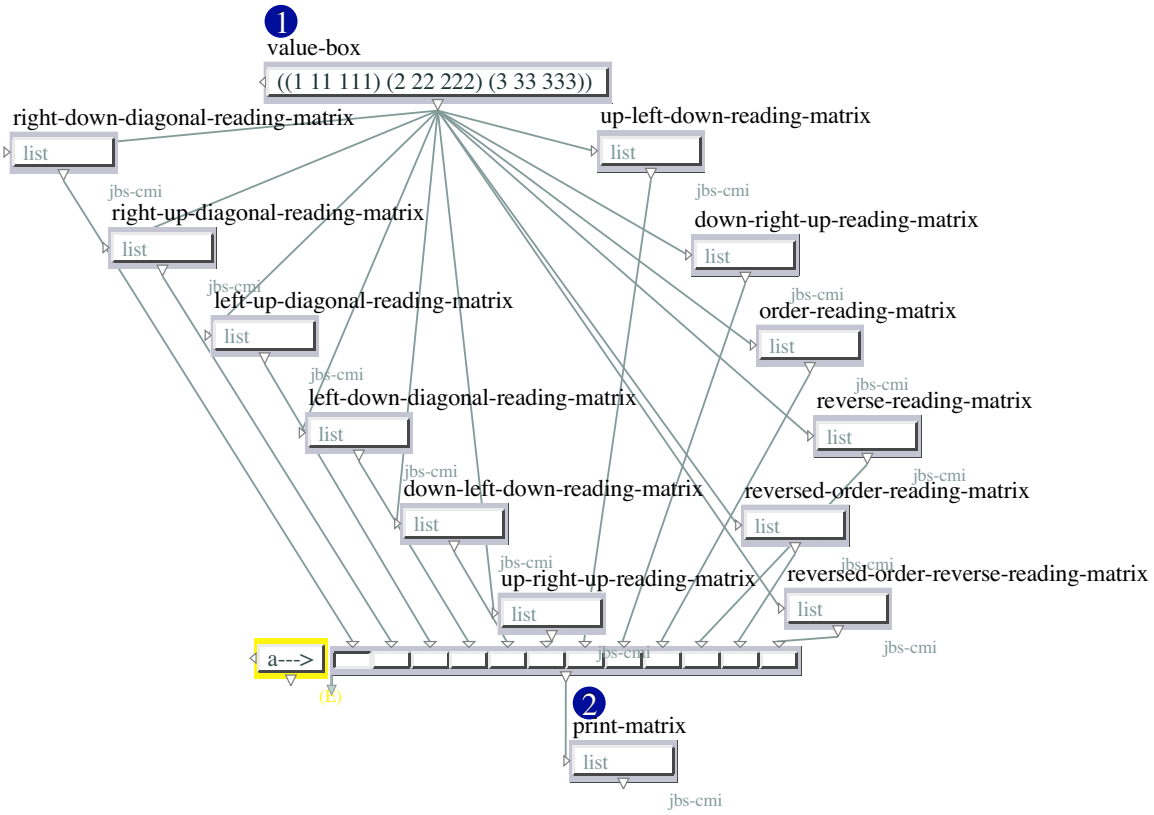

Figure 12: 3-3-1-all-reading-matrix

4.5 0-Print-List-Matrix

4.5.1 2-Print-Lists

3.3.2 - PRINT-LISTS
This function [1] is just a tool to print a list of lists in a more readable way.

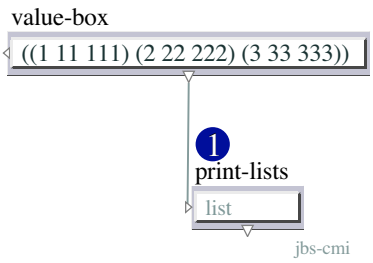

Figure 13: 3-3-2-print-lists

4.5.2 3-Print-Matrix

3.3.3 - PRINT-MATRIX
This function [1] is just a tool to print a list of lists in a matrix readable format.

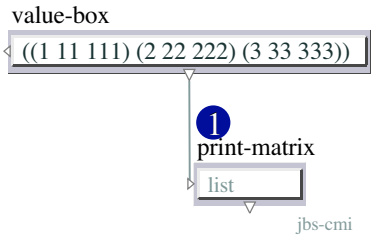

Figure 14: 3-3-3-print-matrix

5 0-Special-Combinations

5.1 0-Special-Combinations

This part of the library groups together some useful functions to generate every possibilities for some given operations on a list and also some special combinatorial cases.

5.2 1-All-Possibilities

5.2.1 1-All-Combinations

4.1.1 - ALL-COMBINATIONS
A function from the old PatchWork. It creates all combinations, with elements from the given list in 'vals', and with a length set in 'n'.

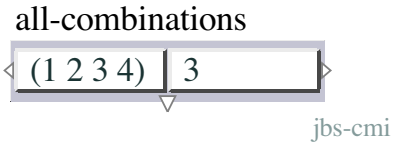

Figure 15: 4-1-1-all-combinations

5.2.2 2-All-Permutations

4.1.2 - ALL-PERMUTATIONS
A function from the old PatchWork. It creates all permutations with elements from a given list.

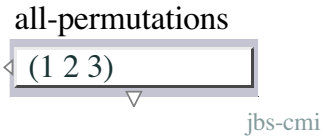

Figure 16: 4-1-2-all-permutations

5.2.3 3-All-Rotations

4.1.3 - ALL-ROTATIONS
This function gives all the circular permutations of a given list.

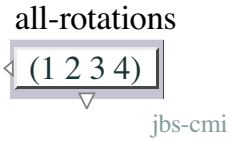

Figure 17: 4-1-3-all-rotations

5.3 2-Circular

5.3.1 1-Circ-down-Picth

4.2.1 - CIRC-DOWN-PITCH
This function [1] concerns pitches. It creates all circular permutations of a given list of pitches.
The specificity of this function is that each permutation is transposed on a LOWER octave from the preceding one.

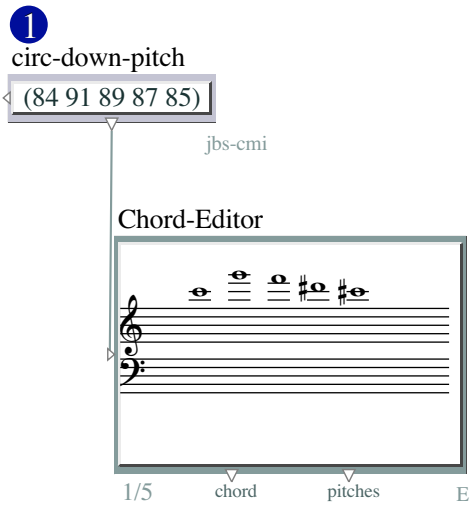

Figure 18: 4-2-1-circ-down-picth

5.3.2 2-Circ-up-Pitch

4.2.2 - CIRC-UP-PITCH
This function [1] concerns pitches. It creates all circular permutations of a given list of pitches.
The specificity of this function is that each permutation is transposed on a UPPER octave from the preceding one.

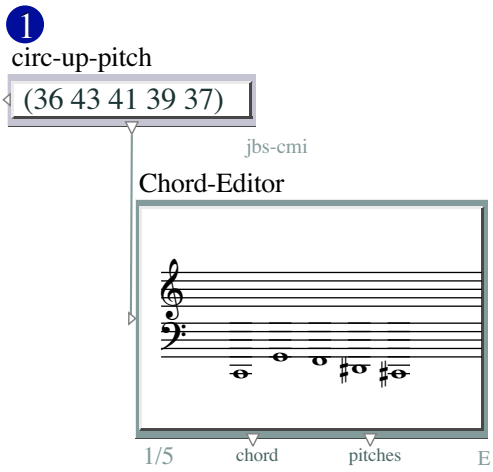

Figure 19: 4-2-2-circ-up-pitch

5.3.3 3-Circular-Groups10

4.2.3 - CIRCULAR-GROUPS10
This function [1] creates circular permutations between at maximum 10 groups.
In the ten list inputs [a] (list1, list2, list3, etc. list10) you can define the different elements. If the corresponding menu [b] (group1, group2, group3, etc. group10) is set on 'fix-val?', it means that the group will be repeated several times. If it is set on 'permut-val?', that means that the group will have all its circular permutations.
As the lists can have different lengths (as it is in this example), the number of repetitions of a group (if the corresponding menu is 'fix-val?') depends on the longest list of all the groups.
The same is for the circular permutations. The number of circular permutations of a given group is repeated in order to get the length of the longest group treatment.
If the menu 'global-permut?' [c] is set on 'fix-val?', that means that the treatment of each list stays in the exact order following the numeration of the lists.
If the menu 'global-permut?' [c] is set on 'permut-val?', that means that all the treatments of all the lists are permutated in a circular way.
Now please open the red abstraction.

Here there are only three lists in order to make this function more readable : - 1 11 111 1111 - 2 22 222 - a b c d e f g h
All the three menus [b1, b2 and b3] are set on 'fix-val?'.
Evaluate the PRINT-LISTS box [2] and the result is: —> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (1 11 111 1111) (2 22 222) (A B C D E F G H)
As you see, the number of repetitions of all the list is synchronized on the longest one (a b c d e f g h).
Please set the menu b1 on 'permut-val?'. Now the result is:
—> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (11 111 1111 1) (2 22 222) (A B C D E F G H) —> (111 1111 1 11) (2 22 222) (A B C D E F G H) —> (1111 1 11 111) (2 22 222) (A B C D E F G H) —> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (11 111 1111 1) (2 22 222) (A B C D E F G H) —> (111 1111 1 11) (2 22 222) (A B C D E F G H) —> (1111 1 11 111) (2 22 222) (A B C D E F G H)
As you can see, the number of circular permutations of the list (1 11 111) is not 3, but it depends again on the longest list (a b c d e f g h).
Now set the three menus (b1, b2 and B3) on 'permut-val?'. Now the result is :
—> (1 11 111 1111) (2 22 222) (A B C D E F G H) —> (11 111 1111 1) (22 222 2) (B C D E F G H A) —> (111 1111 1 11) (222 2 22) (C D E F G H A B) —> (1111 1 11 111) (2 22 222) (D E F G H A B C) —> (1 11 111 1111) (22 222 2) (E F G H A B C D) —> (11 111 1111 1) (222 2 22) (F G H A B C D E) —> (111 1111 1 11) (2 22 222) (G H A B C D E F) —> (1111 1 11 111) (22 222 2) (H A B C D E F G)
All the list have been permutated circularly, till when the longest one (a b c d e f g h) has finished.
Now set the menu [c] 'global-permut?' on 'permut-val?'. The result is too long to be printed here, so, please look in the PWGL OUTPUT window.

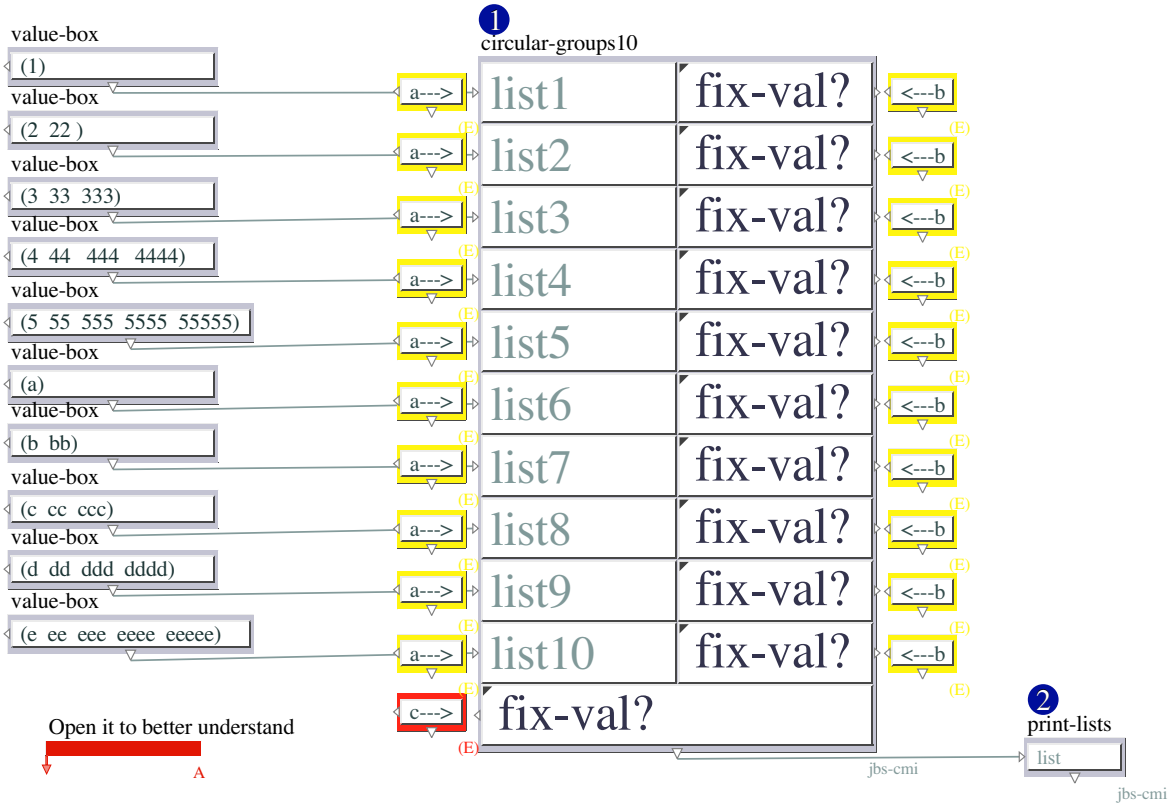

Figure 20: 4-2-3-circular-groups10

6 0-Grouping

6.1 0-Grouping

By grouping I mean a general concept based on the act or process of uniting elements into groups following a given criterion. This part of the library is divided in GROUPS and SEGMENTATIONS.
In GROUPS I put all the possible logical criteria I found, based on sets definition.
In SEGMENTATIONS I put some (not all) criteria developed by the theory of Morphologie (see also the library Morphologie) using the concept of new/old parameter.
If you do not know this theory, please look at the Paolo Aralla paper in PRISMA-01, Milan, 2003, Analisi morfologica - Un modello matematico per descrivere la relazione fra struttura morfologica del messaggio, attivit mnemonico-percettiva e risposta psichica.

6.2 1-Groups

6.2.1 1-Group-List

5.1.1 - GROUP-LIST
This function [1] can group a given list [a] into several groups with lengths specified in [b].
With the menu [c] you can: - stop - choose to finish the grouping action when the starting list arrives at its end; - circ - choose to continue to read the list in a circular way till when all the groups lengths have been used; - scale - choose to rescale proportionally the lengths of the subgroups in order to fit exactly in the length of the original list.

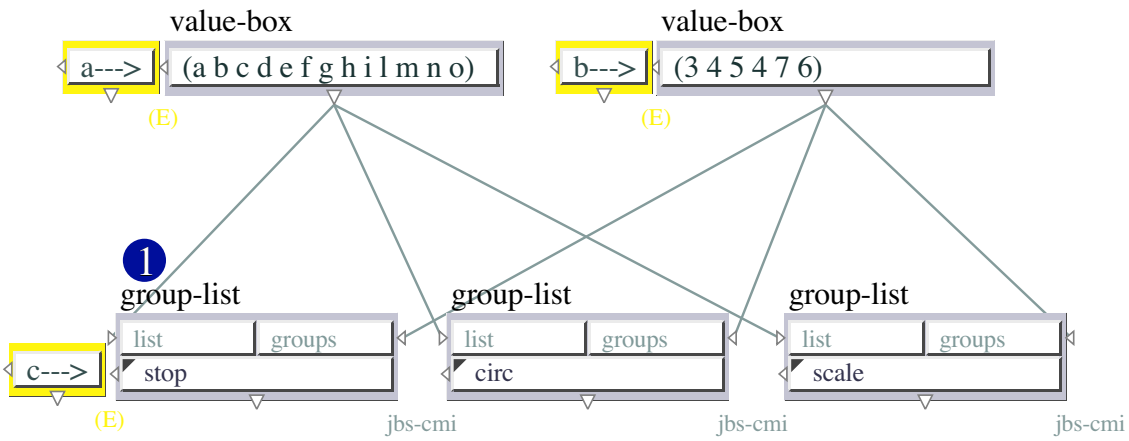

Figure 21: 5-1-1-group-list

6.2.2 2-Group-Equals

5.2 - GROUP-EQUALS
This function [1] groups every consecutive identical elements.

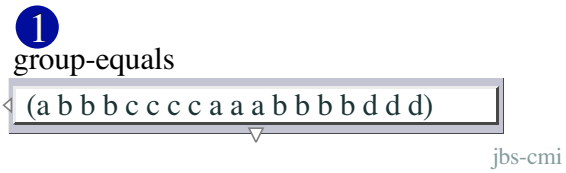

Figure 22: 5-1-2-group-equals

6.2.3 3-All-Sub-Groups

5.3 - ALL-SUB-GROUPS
This function [1] outputs all the possible subgroups of a given list.

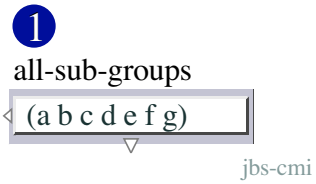

Figure 23: 5-1-3-all-sub-groups

6.2.4 4-All-Given-Sub-Groups

5.1.4 - ALL-GIVEN-SUB-GROUPS
This function [1] outputs all subgroups of a given list [a]. The subgroups lengths are set in [b].
For instance: If you put in [b] the atom 3, you will get only subgroups of length 3; if you put the atom 5 only the subgroups of length 5, etc.
But if [2] you put a list [c] like (1 5), you will get all the subgroups of length 1, 2, 3, 4 and 5.
ATTENTION: if you do not put the lengths in the 'sort' order, you get nil. So, look at PWGL-MAP [3, the red boxes] in order to see how to generate several subgroups with not linear lengths.

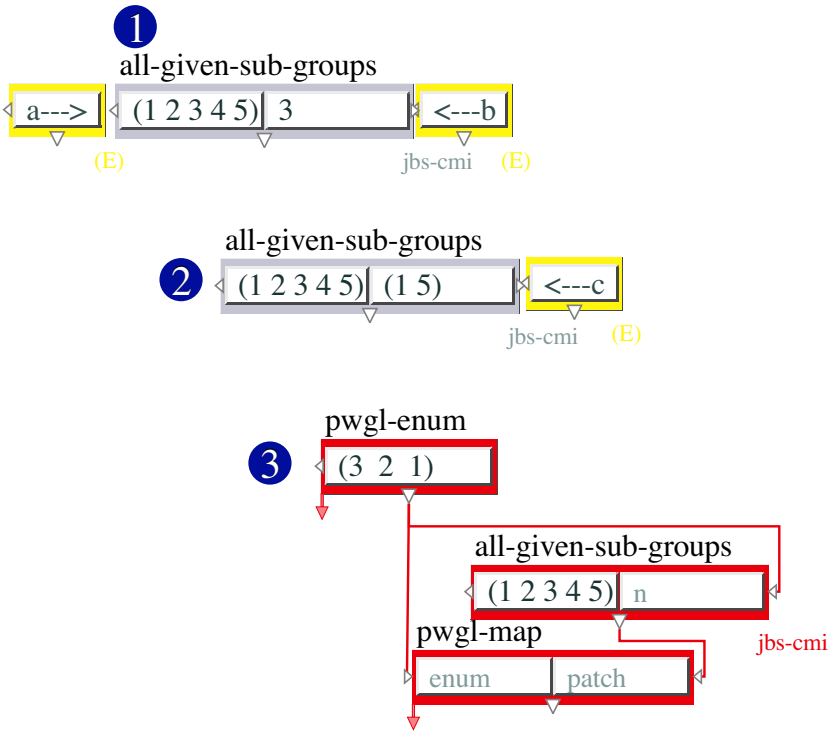

Figure 24: 5-1-4-all-given-sub-groups

6.2.5 5-Grouping-Including-Given

5.1.5 - GROUPING-INCLUDING-GIVEN-ELEMENT
This function [1] groups a given list [a] into new subgroups beginning with the given element set in [b]. In other words, every subgroups will start with the element set in [a].
ATTENTION : If the list does not start with the element set in [b], the first subgroup will not contain this element. Please use the PWGL-SWITCH [2] and choose which list you want to group and look at the result.

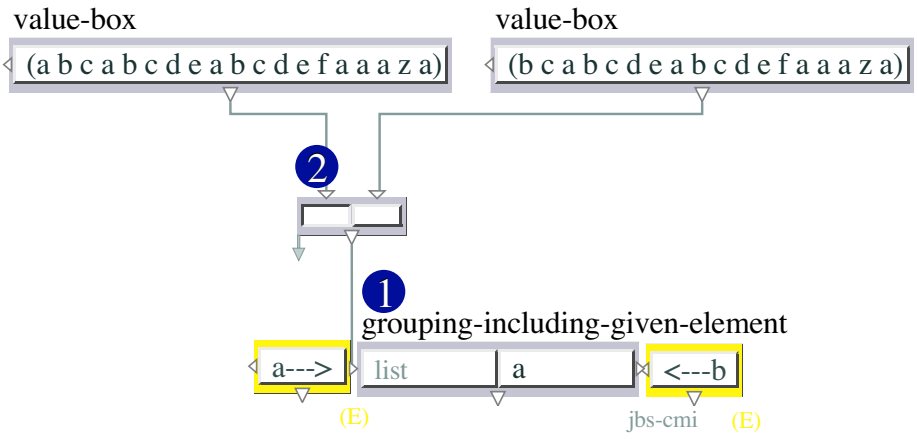

Figure 25: 5-1-5-grouping-including-given

6.2.6 6-Grouping-excluding-Given

5.1.6 - GROUPING-EXCLUDING-GIVEN-ELEMENT
This function [1] groups a given list [a] into new subgroups, isolating the given element set in [b] from any other elements. In other words, every subgroups will contain either the unique element set in [a], or groups of other elements.

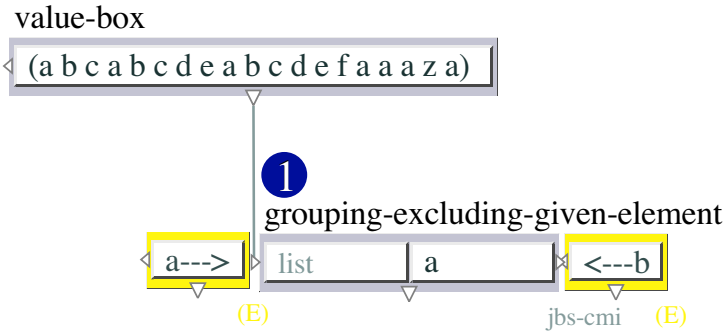

Figure 26: 5-1-6-grouping-excluding-given

6.2.7 7-All-Groups-by-All-Elements

5.1.7 - ALL-GROUPS-BY-ALL-ELEMENTS
This function [1] creates all sublists accordingly to every elements of the given list [a]. With the menu 'included?' [b] you can choose with 'yes' to include the element of the segmentation inside each sublist. With 'no' you can choose to exclude the element of the segmentation and to put it into a separate sublist.

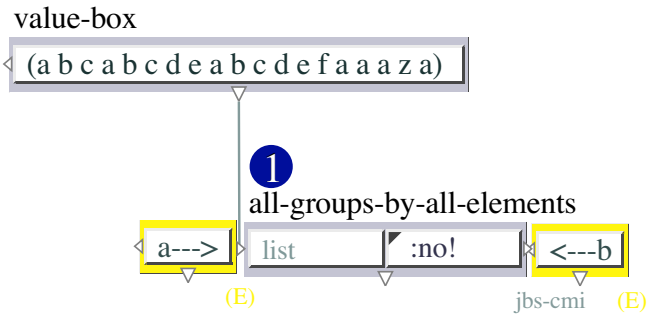

Figure 27: 5-1-7-all-groups-by-all-elements

6.2.8 8-Mixing-List-Groups

5.1.8 - CHAINING-GROUPS
This function [1] creates chained subgroups.
In [a] you put the list you want to chain, and in [b] you define the lengths of the subgroups.
In [c] you put how many last elements have to be in common between a group and its follower. For instance if you put (1 1 1) between the first group and the second there will be 1 common last element, the same between the second and the third, and the same too between the third and the fourth. But if you put (1 4 2), that means that between the first and the second group there will be 1 last common element; between the second and the third there will be 4 last common elements; between the third and the fourth there will be 2 last common elements.

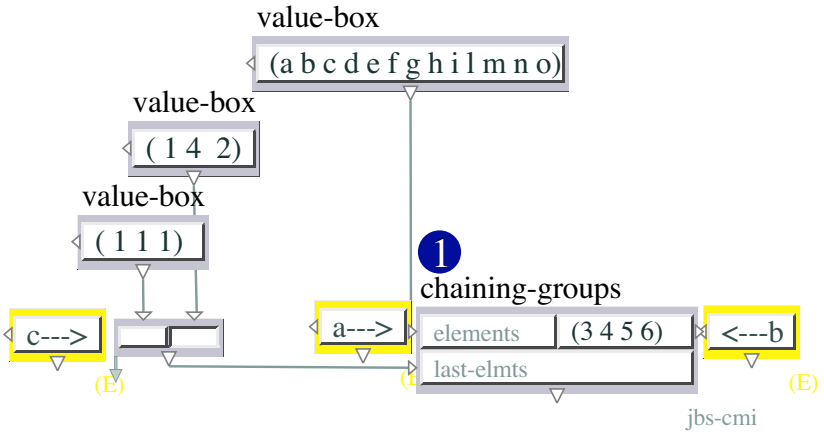

Figure 28: 5-1-8-mixing-list-groups

6.3 2-Segmentations

6.3.1 1-Up-down-Peaks-Segmentation

5.2.1 - MAXIMUM-MINIMUM-SEGMENTATIONS
(To better understand these two functions, please look also in mathematics, maxima and minima, known collectively as extrema.)
The function MAXIMUM-SEGMENTATION [1] creates a segmentation based on the primitive forms of a given list of numbers.
ATTENTION : This function works only with numbers (not with symbols). It creates a separate group of lists segmented before and after a point of maximum.
The function MINIMUM-SEGMENTATION [2] creates a segmentation based on the primitive forms of a given list of numbers.
ATTENTION : This function too works only with numbers. It creates a separate group of lists segmented before and after a point of minimum.

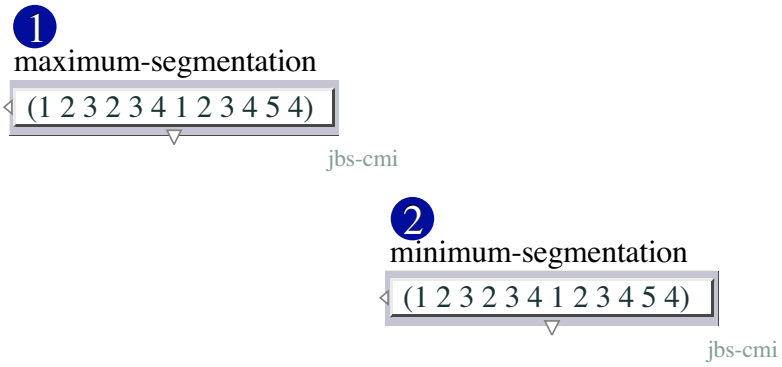

Figure 29: 5-2-1-up-down-peaks-segmentation

6.3.2 2-On-New-or-on-Old-Segmentation

5.2.2 - ON-NEW-OLD-SEGMENTATIONS
(These two functions refer to the morphology theory developed by Paolo Aralla.)
The ON-NEW-SEGMENTATION [1] creates a group on each NEW incoming element. To create this segmentation, it creates a group excluding the NEW element and another group including it.
The ON-OLD-SEGMNTATION [2] creates a group on each OLD element appearing in the given list.

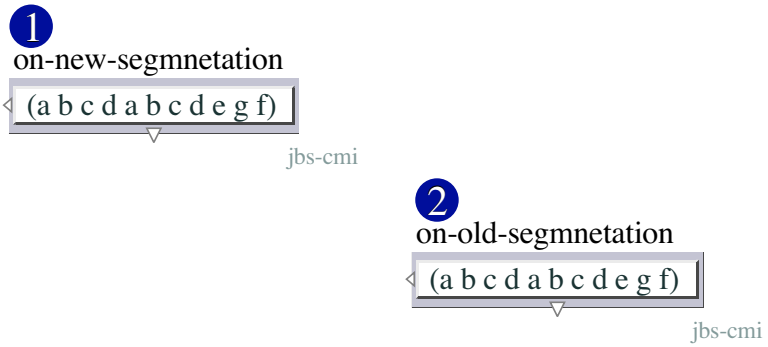

Figure 30: 5-2-2-on-new-or-on-old-segmentation

6.3.3 3-On-New-or-Old-on-New&Old-Segmentation

5.2.3 - ON-NEW-OR-OLD-NEW/OLD-ANALYSIS-SEGMENTATIONS
(These two functions refer to the morphology theory developed by Paolo Aralla.)
The ON-NEW/NEW-OLD-ANALYSIS-SEGMENTATION [1] creates a new group on each NEW element coming out of the NEW-OLD-ANALYSIS of the given list.
The ON-OLD/NEW-OLD-ANALYSIS-SEGMENTATION [1] creates a new group on each OLD element coming out of the NEW-OLD-ANALYSIS of the given list.

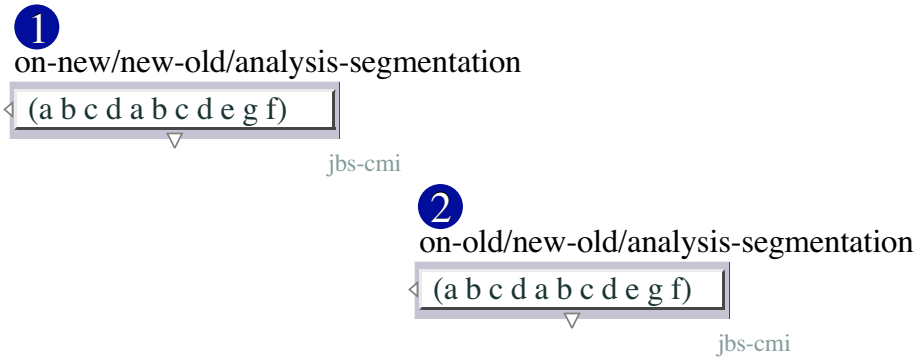

Figure 31: 5-2-3-on-new-or-old-on-new-old-segmentation

7 0-Utilities

7.1 0-Utils

Just some useful functions.

7.2 01-Depth

6.1 - DEPTH
This function [1] outputs each element of a given list with the corresponding level of nesting.

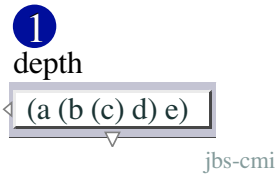

Figure 32: 6-01-depth

7.3 02-Tree-Extract

6.2 - TREE-EXTRACT
This function is a sort a recursive nth.
In [a] you put a list of lists, and in [b] you define the nth of the nth. For instance if you put (1 1), that means that you want to extract the second (nth 1) of the second (nth1).

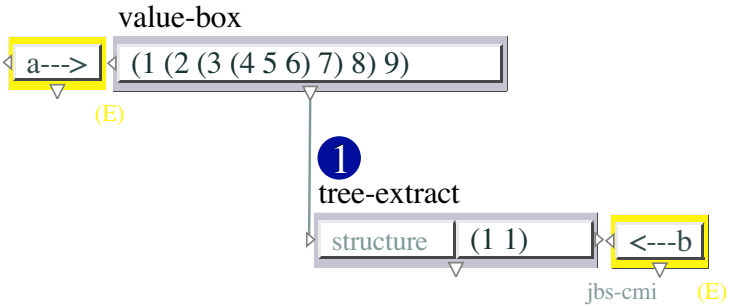

Figure 33: 6-02-tree-extract

7.4 03-Member-in-Sublists?

6.3 - MEMBER-IN-SUBLISTS
This function [1] is a sort of general member test. It outputs T or NIL.
In [a] you put the element you want to be checked as included or excluded.
In [b] you put a list where to check if the item set in [a] is found or not.
With the menu MODE you can choose between four different tests: - with EVERY this function puts T if the element set in [a] is in all sublists; - with SOME or NOTEVERY this function puts T if the element set in [a] is in certain sublists; - with NOTANY this function puts T if the element set in [a] is NOT in any sublists.
Please use the PWGL-SWITCH [2] in order to test several items.

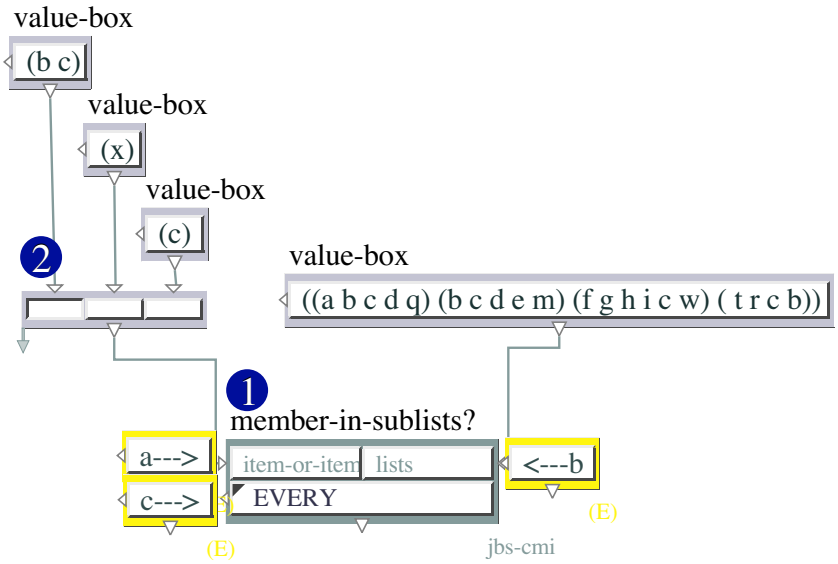

Figure 34: 6-03-member-in-sublists?

7.5 04-Circular-Reading

6.04 - CIRCULAR-LIST-READING
This function [1] allows you to synchronize the lengths of two lists.
If the lists set in [a] and [b] have the same length, it outputs the list set in [a].
If the length of [a] is smaller than the length of [b], it reads [a] in a circular way till when the two lengths are equal.
If the length of [b] is smaller than the length of [a], it reads [b] in a circular way till when the two lengths are equal.

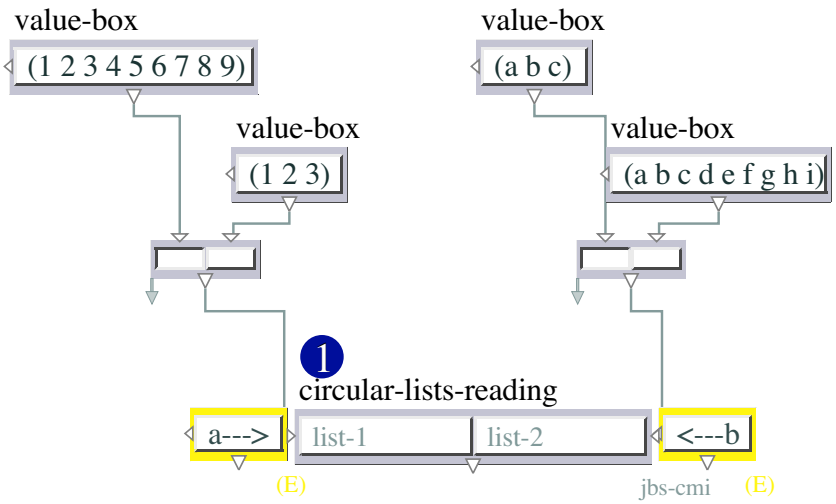

Figure 35: 6-04-circular-reading

7.6 05-First-N &-Last-N

6.05 - FIRST-OR-LAST-N
These two functions take the first elements [1] of a given list or the last [2] ones.
In [a] you put a list, and in [b] you define how many first or last elements you want.

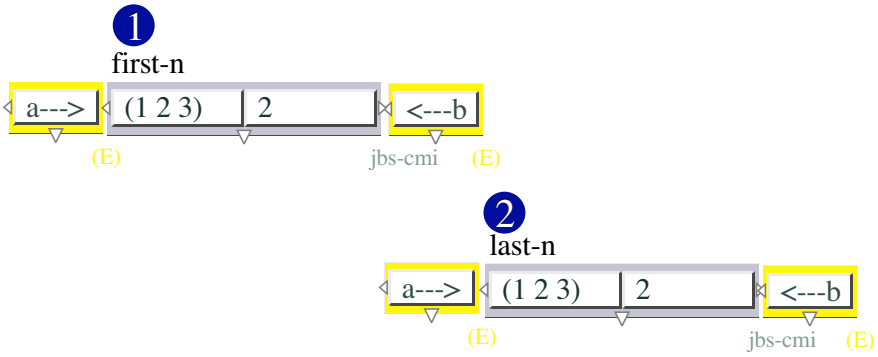

Figure 36: 6-05-first-n-last-n

7.7 06-Complete-List

6.06 - COMPLETE-LIST
This function [1] looks if the length of the left input is equal to the number put in the right one.
If the length is smaller, the function will repeat the last element of the list till when the length is equal to the number set in the right input.

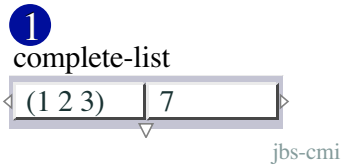

Figure 37: 6-06-complete-list

7.8 07-Index-Subst

6.07 - INDEX-SUBSTITUTE
This function [1] allows you to replace a list of elements giving the nth positions.
In [a] you put a list.
In [b] you define where you want to make a substitution. The values here correspond to nth indexes : 0 is the index for the first element, 1 for the second, 2 for the third, and so on.
In [c] you put the elements that will be put in the given nth indexes set in [b].

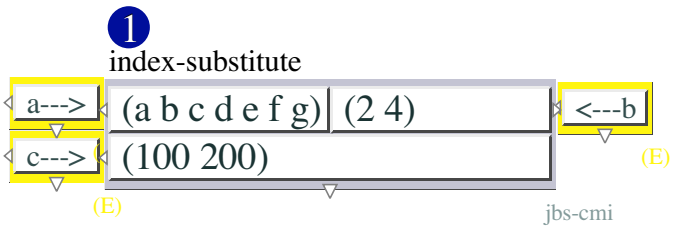

Figure 38: 6-07-index-subst

7.9 08-Arithmetic-Serie-Stop

6.08 - ARITHMETIC-SER-STOP
This function [1] creates an arithmetic series.
In [a] you set the starting point.
In [b] you set the step.
In [c] you define how many steps you want.

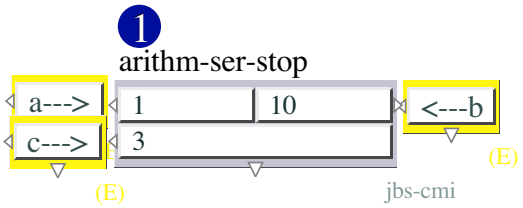

Figure 39: 6-08-arithmetic-serie-stop

7.10 09-Gold-Section

6.09 - GOLD-SECTION-MULTIPLICATION
This function [1] creates a multiplication by the gold section value (0.618). It is a function that can be recalled recursively.
In [a] you define a number you want to multiplied by the gold section factor.
In [b] you can define which level of the recursion you want. With 1 you call the function just one time; with 2 you apply the multiplication on the first result; with 3 you apply the multiplication on the second result that is the multiplication of the first one, and so on.

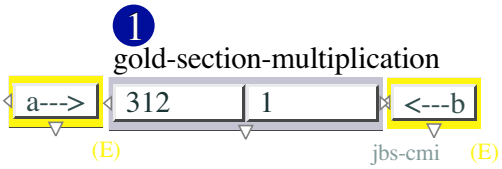

Figure 40: 6-09-gold-section

7.11 10-Several-for-Max-Coll

6.10 - SEVERAL-FOR-MAX-COLL
These 3 functions are just useful to sort a list or a list of lists in the text format of the coll object in MaxMSP and PureData environments.
FOR-MAX-COLL [1]
This function prepares a list like this one ((1 2 3) (a b c) (100 200 300)) in this format :
1, 1 2 3; 2, A B C; 3, 100 200 300;
FOR-MAX-SPECIAL-COLL [2]
This function prepares two lists, this one ((1 2 3) (a b c) (100 200 300)) and this one ((5 6 7) (d e f) (11 22 33)), in the following format:
(1 2 3), (5 6 7); (A B C), (D E F); (100 200 300), (11 22 33);
FOR-MAX-COLLECT-NAME-AND-INDEX [3]
This function prepares one list like this one (start start start) in the following format:
START1; START2; START3;
ATTENTION: With the input 'format?' [b] you can choose either to print the result in the PWGL output [to-listener], or to create a new file through a dialog box [to-file].

TO IMPORT IN MAX: If you create a file, you can open it directly in Max like any COLL text file.
ATTENTION 2: these two functions [2, 3] are not so easily usable directly in Max... Why? Why not!

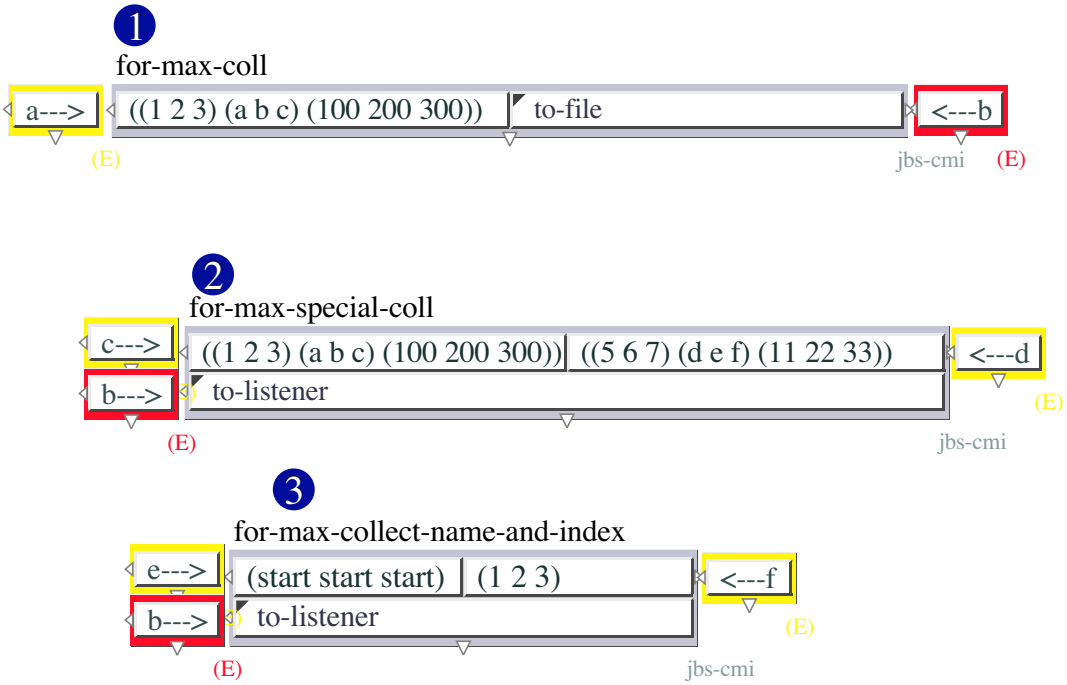

Figure 41: 6-10-several-for-max-coll

7.12 11-Find-All-Intervals

6.11 - FIND-ALL-INTERVALS
This function [1] comes from Brian Ferneyhough's old PatchWork Library.
It calculates all the melodic intervals of a given list of pitches.

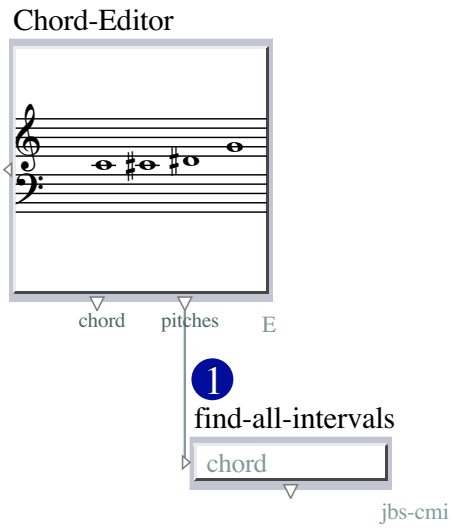

Figure 42: 6-11-find-all-intervals

A Box Reference

all-combinations

arglist: `(vals n)`
package: JBS-CMI
menu: Special-Combinations▶All-Possibilities
4.1.1 - ALL-COMBINATIONS
From old PatchWork: it creates all combinations of the given list in vals with a length set in n.

all-given-sub-groups

arglist: `(list1 n)`
package: JBS-CMI
menu: Groups-Segmentations▶Groups
5.1.4 - ALL-GIVEN-SUB-GROUPS
This function outputs all sub groups of a given list [list1]. The subgroups length is set in [n]. For instance: If you put in [n] the atom 3, you will get only subgroups of length 3; if you put the atom 5 only the subgroups of length 5. But if you put a list like (2 4), you will get all the subgroups of length 2, 3 and 4.

all-groups-by-all-elements

arglist: `(list included?)`
package: JBS-CMI
menu: Groups-Segmentations▶Groups
5.1.7 - ALL-GROUPS-BY-ALL-ELEMENTS
This function creates all sub lists accordingly to all elements of the given list [list]. With the menu INCLUDED? you can choose with YES to include the element of the segmentation inside each sub list; with NO! you can choose to exclude the element of the segmentation and to put into a separate sub list.

all-permutations

arglist: `(list)`
package: JBS-CMI
menu: Special-Combinations▶All-Possibilities
4.1.2 - ALL-PERMUTATIONS
From old PatchWork: it creates all permutations of the given list.

all-rotations

arglist: `(list)`
package: JBS-CMI

menu: Special-Combinations▶All-Possibilities
4.1.3 - ALL-ROTATIONS
It gives all the circular permutations of a given list.

all-sub-groups

arglist: `(list)`
package: JBS-CMI
menu: Groups-Segmentations▶Groups
5.3 - ALL-SUB-GROUPS
This function outputs all the possible sub groups of a given list.

all-to-one

arglist: `(list)`
package: JBS-CMI
menu: Numeric-Comment▶Moving
3.2.1 - ALL-TO-ONE (This is an old B. Ferneyhough function)
This function creates a linear interpolation between a list of numbers set in [list] and the number 1. When a number of the given list reaches 1, then it disappears.
I you put (10 7 5 2) as starting list. The result will be.
—> 10 7 5 2 —> 9 6 4 1 —> 8 5 3 —> 7 4 2 —> 6 3 1 —> 5 2 —> 4 1 —> 3 —> 2 —> 1

all-to-x

arglist: `(list x)`
package: JBS-CMI
menu: Numeric-Comment▶Moving
3.2.2 - ALL-TO-X
This function creates a linear interpolation between a list of numbers set in [list] and the number set in [x]. When a number of the given list reaches the value set in [x], then it disappears.
I you put (20 7 1) as starting list. The result will be.
—> 20 7 1 —> 19 8 2 —> 18 9 3 —> 17 10 4 —> 16 5 —> 15 6 —> 14 7 —> 13 8 —> 12 9 —> 11 10 —> 10

answer

arglist: `(subject dominant)`
package: JBS-CMI
menu: Pitch
2.2 - ANSWER
This function is a simplified reproduction of the tonal answer of the fugue.
In [subject] you put a melodic profile.

In [dominant] you define (as nth index) which note of the profile has to be considered as the dominant.
When you evaluate the ANSWER all the notes of the profile are transposed to the dominant, except the dominant, that is transposed on the tonic.

arithm-ser-stop

arglist: `(start step stop)`
package: JBS-CMI
menu: Utilities
6.08 - ARITHMETIC-SER-STOP
This function creates an arithmetic series.
In [start] you set the starting point.
In [step] you set the step.
In [stop] you define how many step you want.

chaining-groups

arglist: `(elements groups last-elmts)`
package: JBS-CMI
menu: Groups-Segmentations▶Groups
5.1.8 - CHAINING-GROUPS
This function creates chained sub groups. In [elements] you put the list you want to chain. In [groups] you define the length of the subgroups.
In [last-elemts] you put how many last elements have to be in common between a group and its follower. For instance if you put (1 1 1) between the first group and the second there will be 1 common last element; the same between the second and the third; and the same too between the third and the fourth. But if you put (1 4 2), that means that between the first and the second group there will be 1 last common element; between the second and the third there will be 4 last common elements; between the third and the fourth there will be 2 last common elements.

circ-down-pitch

arglist: `(list)`
package: JBS-CMI
menu: All-Possibilities▶Circular
4.2.1 - CIRC-DOWN-PITCH
This function concerns pitches.
It creates all circular permutations of a given list of pitches. The specificity of this function is that the each circular permutation is transposed on a LOWER octave from the beginning one.

circ-up-pitch

arglist: `(list)`
package: JBS-CMI
menu: All-Possibilities▶Circular
4.2.2 - CIRC-UP-PITCH
This function concerns pitches.
It creates all circular permutations of a given list of pitches. The specificity of this function is that the each circular permutation is transposed on a UPPER octave from the beginning one.

circular-groups10

arglist: `(list1 group1 list2 group2 list3 group3 list4 group4 list5 group5 list6 group6 list7 group7 list8 group8 list9 group9 list10 group10 global-permut?)`
package: JBS-CMI
menu: All-Possibilities▶Circular
4.2.3 - CIRCULAR-GROUPS10
This function creates the circular permutation between at maximum 10 groups.
In the ten lists (list1, list2, list3, etc. list10) you can define the different elements. If the corresponding menus (group1, group2, group3, etc. group10) are set on FIX-VAL?, it means that the group will be repeated several time. If it set on PERMUT-VAL?, that means that the group will have all its circular permutations.
As the lists can have different lengths (as it is in this example), the number of repetitions of a group (if the corresponding menu is FIX-VAL?) depends on the longest list of all the groups.
The same is for the circular permutations. The number of circular permutations of a given group is repeated in order to get the length of the longest group treatment.
If the menu GLOBAL-PERMUT? is set on FIX-VAL?, that means that the treatment of each list stays in the exact order following the numeration of the lists.
If the menu GLOBAL-PERMUT? is set on PERMUT)VAL?, that means that all the treatments of all the lists are permutated in a circular way.

circular-lists-reading

arglist: `(list-1 list-2)`
package: JBS-CMI
menu: Utilities
6.04 - CIRCULAR-LIST-READING
This function allows you to synchronize the length of two lists.
If the lists set in [list-1] and [list-2] have the same length, it outputs the list set in [list-1]
If the length of [list-1] is smaller than the length of [list-2], it reads [list-1] in a circular way till when the two lengths are equal.
If the length of [list-2] is smaller than the length of [list-1], it reads [list-2] in a circular way till when the two lengths are equal.

complete-list

arglist: `(list length)`
package: JBS-CMI
menu: Utilities
6.06 - COMPLETE-LIST
This function looks if the length of the left [list] input is equal to the number put in the right one [lenght].
If the length smaller, the function will repeat the last element of the list till when the length is equal till the number set in the right input.

depth

arglist: `(liste)`
package: JBS-CMI
menu: Utilities
6.1 - DEPTH
This function outputs each element of a given list with the corresponding level of nesting.

down-left-down-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It gives back the matrix reading in a diagonal way, going down from left to rigth while going down through the matrix.
Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result: —> 1 —> 11 2 —> 111 22 3 —> 222 33 —> 333

down-right-up-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It gives back the matrix reading in a diagonal way, going down from left to rigth while going down through the matrix.
Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result:
—> 1 —> 2 11 —> 3 22 111 —> 33 222 —> 333

find-all-intervals

arglist: `(chord)`
package: JBS-CMI
menu: Utilities
6.11 - FIND-ALL-INTERVALS
This function comes from B. FERNEYHOUGH old PatchWork Library.
It calculates all the melodic intervals of a given list of pitches.

first-n

arglist: `(list n)`
package: JBS-CMI
menu: Utilities
n first elements of a list

for-max-coll

arglist: `(list format?)`
package: JBS-CMI
menu: Utilities
6.10 - SEVERAL-FOR-MAX-COLL
FOR-MAX-COLL
This function prepares a list like this one ((1 2 3) (a b c) (100 200 300)) in this format:
1, 1 2 3; 2, A B C; 3, 100 200 300;
ATTENTION: with the input FORMAT? you can choose either to print the result in the PWGL output [to-listener], or to create a new file through a dialog box [to-file].

for-max-collect-name-and-index

arglist: `(list1 list2 format?)`
package: JBS-CMI
menu: Utilities
6.10 - SEVERAL-FOR-MAX-COLL
FOR-MAX-COLLECT-NAME-AND-INDEX
This function prepares one list like this one (start start start) in the following format:
START1; START2; START3;
ATTENTION: with the input FORMAT? [b] you can choose either to print the result in the PWGL output [to-listener], or to create a new file through a dialog box [to-file].

for-max-special-coll

arglist: `(list1 list2 format?)`
package: JBS-CMI
menu: Utilities

6.10 - SEVERAL-FOR-MAX-COLL
FOR-MAX-SPECIAL-COLL
This function prepares two lists, this one ((1 2 3) (a b c) (100 200 300)) and this one ((5 6 7) (d e f) (11 22 33)), in the following format:
(1 2 3), (5 6 7); (A B C), (D E F); (100 200 300), (11 22 33);
ATTENTION: with the input FORMAT? you can choose either to print the result in the PWGL output [to-listener], or to create a new file through a dialog box [to-file].

gold-section-multiplication

arglist: `(number gr)`
package: JBS-CMI
menu: Utilities
6.09 - GOLD-SECTION-MULTIPLICATION
This function creates a multiplication by the gold section value (0.618). It is a function that can be recalled recursively.
In [number] you define a number you want to divide by the gold section factor.
In [gr] you can define which level of the recursion you want. With 1 you call the function just one time; with 2 you apply the multiplication on the first value; with 3 you apply the multiplication on the second value that is the multiplication of the first one, and so on.

group-equals

arglist: `(list)`
package: JBS-CMI
menu: Groups-Segmentations▶Groups
5.2 - GROUP-EQUALS
This function groups identical elements with consecutive identical elements.

group-list

arglist: `(list groups mode?)`
package: JBS-CMI
menu: Groups-Segmentations▶Groups
5.1.1 - GROUP-LIST
This function can group a given list [list] into several groups whom lengths are specified in [groups].
With the menu [mode?] you can: - stop - choose to finish the grouping action when the starting list arrives at its end; - circ - choose to continue to read the list in a circular way till when all the sub-group have been completed; - scale - choose to rescale proportionally the lengths of the sub-groups in order to fit exactly in the length of the original list.

grouping-excluding-given-element

arglist: `(list element)`
package: JBS-CMI
menu: Groups-Segmentations▶Groups
5.1.6 - GROUPING-EXLUDING-GIVEN-ELEMENT This function groups a given list [list] creating new subgroups: the given element set in [element] will be put into a separate list, and the other elements in other lists.

grouping-including-given-element

arglist: `(list element)`
package: JBS-CMI
menu: Groups-Segmentations▶Groups
5.1.5 - GROUPING-INCLUDING-GIVEN-ELEMENT This function groups a given list [list] creating new subgroups containing the given element set in [element]. That means that all subgroups will start with the element set in [list].
ATTENTION: if the list does not start with the element set in [element], the first subgroup will not contain this element.

harmonic-fields

arglist: `(field)`
package: JBS-CMI
menu: Pitch
This function allows you to define as defvar variables some list of chords and to recall them in three different ways. First you have to define a set of chords.
Here is an example: (defvar major-triad '((60 64 67) (61 65 68) (62 66 69) (63 67 70) (64 68 71) (65 69 72) (66 70 73) (67 71 74)
(68 72 75) (69 73 76) (70 74 77) (71 75 78)))
Then you can recall the whole classes of chords with their 12 transpositions, just typing the class name.
But you cane also recall one class with a specific transposition value. This value belongs to the modulo 12 pitch result. For instance : (major-triad 11)
Or you can set a specific order to recall some transpositions of the two classes of chords. Here is an example : ((major-triad 11) (o-a 1) (major-triad 2) (o-d 7)).
I've defined some classes chosen by my own taste: these calsses come out of E. Carter, K. Stockhausen, I. Fedele, and some of myself.
Here they are:
(defvar wood '(64 67 74 71 77))
(defvar -4-5-a '((60 61 62 64) (61 62 63 65) (62 63 64 66) (63 64 65 67) (64 65 66 68)
(65 66 67 69) (66 67 68 70) (67 68 69 71) (68 69 70 72) (69 70 71 73)
(70 71 72 74) (71 72 73 75)))
(defvar -4-5-b '((60 61 63 64) (61 62 64 65) (62 63 65 66) (63 64 66 67) (64 65 67 68)

```
(65 66 68 69) (66 67 69 70) (67 68 70 71) (68 69 71 72) (69 70 72 73)
(70 71 73 74) (71 72 74 75)))
(defvar -4-5-c '((60 62 63 64) (61 63 64 65) (62 64 65 66) (63 65 66 67) (64 66 67
68)
(65 67 68 69) (66 68 69 70) (67 69 70 71) (68 70 71 72) (69 71 72 73)
(70 72 73 74) (71 73 74 75)))
(defvar -3-6-a '((60 61 62 67) (61 62 63 68) (62 63 64 69) (63 64 65 70) (64 65 66
71)
(65 66 67 72) (66 67 68 73) (67 68 69 74) (68 69 70 75) (69 70 71 76)
(70 71 72 77) (71 72 73 78)))
(defvar -3-6-b '((60 61 66 67) (61 62 67 68) (62 63 68 69) (63 64 69 70) (64 65 70
71)
(65 66 71 72) (66 67 72 73) (67 68 73 74) (68 69 74 75) (69 70 75 76)
(70 71 76 77) (71 72 77 78)))
(defvar o-a '((60 61 63 67) (61 62 64 68) (62 63 65 69) (63 64 66 70) (64 65 67 71)
(65 66 68 72) (66 67 69 73) (67 68 70 74) (68 69 71 75) (69 70 72 76)
(70 71 73 77) (71 72 74 78)))
(defvar o-b '((60 61 64 66) (61 62 65 67) (62 63 66 68) (63 64 67 69) (64 65 68 70)
(65 66 69 71) (66 67 70 72) (67 68 71 73) (68 69 72 74) (69 70 73 75)
(70 71 74 76) (71 72 75 77)))
(defvar o-c '((60 62 65 66) (61 63 66 67) (62 64 67 68) (63 65 68 69) (64 66 69 70)
(65 67 70 71) (66 68 71 72) (67 69 72 73) (68 70 73 74) (69 71 74 75)
(70 72 75 76) (71 73 76 77)))
(defvar o-d '((60 64 66 67) (61 65 67 68) (62 66 68 69) (63 67 69 70) (64 68 70 71)
(65 69 71 72) (66 70 72 73) (67 71 73 74) (68 72 74 75) (69 73 75 76)
(70 74 76 77) (71 75 77 78)))
(defvar -4-a '((60 61 62 63 65) (61 62 63 64 66) (62 63 64 65 67) (63 64 65 66 68)
(64 65 66 67 69) (65 66 67 68 70) (66 67 68 69 71) (67 68 69 70 72)
(68 69 70 71 73) (69 70 71 72 74) (70 71 72 73 75) (71 72 73 74 76)))
(defvar -4-b '((60 61 62 64 65) (61 62 63 65 66) (62 63 64 66 67) (63 64 65 67 68)
(64 65 66 68 69) (65 66 67 69 70) (66 67 68 70 71) (67 68 69 71 72)
(68 69 70 72 73) (69 70 71 73 74) (70 71 72 74 75) (71 72 73 75 76)))
(defvar -4-c '((60 61 63 64 65) (61 62 64 65 66) (62 63 65 66 67) (63 64 66 67 68)
(64 65 67 68 69) (65 66 68 69 70) (66 67 69 70 71) (67 68 70 71 72)
(68 69 71 72 73) (69 70 72 73 74) (70 71 73 74 75) (71 72 74 75 76)))
(defvar -4-d '((60 62 63 64 65) (61 63 64 65 66) (62 64 65 66 67) (63 65 66 67 68)
(64 66 67 68 69) (65 67 68 69 70) (66 68 69 70 71) (67 69 70 71 72)
(68 70 71 72 73) (69 71 72 73 74) (70 72 73 74 75) (71 73 74 75 76)))
(defvar crom-up '(60 61 62 63 64 65 66 67 68 69 70 71))
(defvar crom-down '(71 70 69 68 67 66 65 64 63 62 61 60))
(defvar 4-up '((60 65 70 75 68 73 78 71 64 69 62 67)
(61 66 71 76 69 74 79 72 65 70 63 68)
(61 66 71 76 69 74 79 72 65 70 63 68)
(62 67 72 77 70 75 80 73 66 71 64 69)
(63 68 73 78 71 76 81 74 67 72 65 70)
```

```
(64 69 74 79 72 77 82 75 68 73 66 71)
(65 70 75 80 73 78 83 76 69 74 67 72)
(66 71 76 81 74 79 84 77 70 75 68 73)
(67 72 77 82 75 80 85 78 71 76 69 74)
(68 73 78 83 76 81 86 79 72 77 70 75)
(69 74 79 84 77 82 87 80 73 78 71 76)
(70 75 80 85 78 83 88 81 74 79 72 77)
(71 76 81 86 79 84 89 82 75 80 73 78)
))
(defvar 4< '((36 41 46 51 56 61 66 71 76 81 86 91 96)
(37 42 47 52 57 62 67 72 77 82 87 92 97)
(38 43 48 53 58 63 68 73 78 83 88 93 98)
(39 44 49 54 59 64 69 74 79 84 89 94 99)
(40 45 50 55 60 65 70 75 80 85 90 95 1)
(41 46 51 56 61 66 71 76 81 86 91 96 101)
(42 47 52 57 62 67 72 77 82 87 92 97 102)
(43 48 53 58 63 68 73 78 83 88 93 98 103)
(44 49 54 59 64 69 74 79 84 89 94 99 104)
(45 50 55 60 65 70 75 80 85 90 95 1 105)
(46 51 56 61 66 71 76 81 86 91 96 101 106)
(47 52 57 62 67 72 77 82 87 92 97 102 107)
))
(defvar 5-up '((60 67 74 69 76 71 66 61 68 63 70 65)
(61 68 75 70 77 60 67 62 69 64 71 66)
(62 69 76 71 78 61 68 63 70 65 60 67)
(63 70 77 72 67 62 69 64 71 66 61 68)
(64 71 78 73 68 63 70 65 60 67 62 69)
(65 72 79 74 69 64 71 66 61 68 63 70)
(66 73 68 75 70 65 60 67 62 69 64 71)
(67 74 69 76 71 66 61 68 63 70 65 60)
(68 75 70 77 60 67 62 69 64 71 66 61)
(69 76 71 78 61 68 63 70 65 60 67 62)
(70 77 72 67 62 69 64 71 66 61 68 63)
(71 78 73 68 63 70 65 60 67 62 69 64)
))
(defvar 5< '((36 43 50 57 64 71 78 85 92 99 106 113 120)
(37 44 51 58 65 72 79 86 93 1 107 114 121)
(38 45 52 59 66 73 80 87 94 101 108 115 122)
(39 46 53 60 67 74 81 88 95 102 109 116 123)
(40 47 54 61 68 75 82 89 96 103 110 117 124)
(41 48 55 62 69 76 83 90 97 104 111 118 125)
(42 49 56 63 70 77 84 91 98 105 112 119 126)
(43 50 57 64 71 78 85 92 99 106 113 120 127)
(44 51 58 65 72 79 86 93 1 107 114 121 128)
(45 52 59 66 73 80 87 94 101 108 115 122 129)
```

```
(46 53 60 67 74 81 88 95 102 109 116 123 130)
(47 54 61 68 75 82 89 96 103 110 117 124 131)
))
(defvar octphone '((60 61 63 64 66 67 69 70)
(61 62 64 65 67 68 70 71)
(62 63 65 66 68 69 71 72)))
(defvar enphone '((60 61 62 64 65 66 68 69 70)
(61 62 63 65 66 67 69 70 71)
(62 63 64 66 67 68 70 71 72)
(63 64 65 67 68 69 71 72 73)
))
(defvar pentaton-a '((60 61 65 69 70)
(61 62 66 70 71)
(62 63 67 71 72)
(63 64 68 72 73)
(64 65 69 73 74)
(65 66 70 74 75)
(66 67 71 75 76)
(67 68 72 76 77)
(68 69 73 77 78)
(69 70 74 78 79)
(70 71 75 79 68)
(71 72 76 68 69)
))
(defvar pentaton-b '((60 61 63 65 66)
(61 62 64 66 67)
(62 63 65 67 68)
(63 64 66 68 69)
(64 65 67 69 70)
(65 66 68 70 71)
(66 67 69 71 72)
(67 68 70 72 73)
(68 69 71 73 74)
(69 70 72 74 75)
(70 71 73 75 76)
(71 72 74 76 77)
))
(defvar octotone '((60 62 64 65 66 68 70 71)
(61 63 65 66 67 69 71 72)
(62 64 66 67 68 70 72 73)
(63 65 67 68 69 71 73 74)
(64 66 68 69 70 72 74 75)
(65 67 69 70 71 73 75 76)
))
(defvar eptatone-a '((60 63 64 65 67 68 69 72)
```

```
(61 64 65 66 68 69 70 73)
(62 65 66 67 69 70 71 74)
(63 66 67 68 70 71 72 75)
(64 67 68 69 71 72 73 76)
(65 68 69 70 72 73 74 77)
(66 69 70 71 73 74 75 78)
(67 70 71 72 74 75 76 79)
(68 71 72 73 75 76 77 68)
(69 72 73 74 76 77 78 69)
(70 73 74 75 77 78 79 70)
(71 74 75 76 78 79 68 71)
))
(defvar eptatone-b '((60 61 62 65 67 70 71 72)
(61 62 63 66 68 71 72 73)
(62 63 64 67 69 72 73 74)
(63 64 65 68 70 73 74 75)
(64 65 66 69 71 74 75 76)
(65 66 67 70 72 75 76 77)
(66 67 68 71 73 76 77 78)
(67 68 69 72 74 77 78 79)
(68 69 70 73 75 78 79 68)
(69 70 71 74 76 79 68 69)
(70 71 72 75 77 68 69 70)
(71 72 73 76 78 69 70 71)
))
(defvar openton-5 '((36 38 40 42 43 45 47 49 50 52 54 56
57 59 61 63 64 66 68 70 71 73 75 77
78 68 70 72 73 75 77 55 56 58 60 62
63 65 43 45 46 48 50 52 53 43 45 47
36)
(36 37 39 40 42 43 44 46 47 49 50 51
53 54 56 57 58 60 61 63 64 65 67 68
70 71 72 74 75 77 78 79 69 70 72 73
74 76 77 55 56 57 59 60 62 63 64 66
43 45 46 47 49 50 52 53 54 44 45 47
36)
(36 38 39 41 42 43 45 46 48 49 50 52
53 55 56 57 59 60 62 63 64 66 67 69
70 71 73 74 76 77 78 68 69 71 72 73
75 76 78 55 56 58 59 61 62 63 65 66
44 45 46 48 49 51 52 53 43 44 46 47
36)
))
(defvar openton-4 '((36 38 40 41 43 45 46 48 50 51 53 55
56 58 60 61 63 65 66 68 70 71 73 75
```

```
76 78 68 69 71 73 74 76 78 55 57 59
60)
(36 37 39 40 41 42 44 45 46 47 49 50
51 52 54 55 56 57 59 60 61 62 64 65
66 67 69 70 71 72 74 75 76 77 79 68
69 70 72 73 74 75 77 78 55 56 58 59
60)
(36 37 39 41 42 44 46 47 49 51 52 54
56 57 59 61 62 64 66 67 69 71 72 74
76 77 79 69 70 72 74 75 77 55 56 58
60)
))
(defvar piram-> '((36 47 57 66 74 81 87 92 96 99 101 102)
(37 48 58 67 75 82 88 93 97 1 102 103)
(38 49 59 68 76 83 89 94 98 101 103 104)
(39 50 60 69 77 84 90 95 99 102 104 105)
(40 51 61 70 78 85 91 96 1 103 105 106)
(41 52 62 71 79 86 92 97 101 104 106 107)
(42 53 63 72 80 87 93 98 102 105 107 108)
(43 54 64 73 81 88 94 99 103 106 108 109)
(44 55 65 74 82 89 95 1 104 107 109 110)
(45 56 66 75 83 90 96 101 105 108 110 111)
(46 57 67 76 84 91 97 102 106 109 111 112)
(47 58 68 77 85 92 98 103 107 110 112 113)
))
(defvar piram-< '((36 37 39 42 46 51 57 64 72 81 91 102)
(37 38 40 43 47 52 58 65 73 82 92 103)
(38 39 41 44 48 53 59 66 74 83 93 104)
(39 40 42 45 49 54 60 67 75 84 94 105)
(40 41 43 46 50 55 61 68 76 85 95 106)
(41 42 44 47 51 56 62 69 77 86 96 107)
(42 43 45 48 52 57 63 70 78 87 97 108)
(43 44 46 49 53 58 64 71 79 88 98 109)
(44 45 47 50 54 59 65 72 80 89 99 110)
(45 46 48 51 55 60 66 73 81 90 1 111)
(46 47 49 52 56 61 67 74 82 91 101 112)
(47 48 50 53 57 62 68 75 83 92 102 113)
))
(defvar majour-scale '(
(60 62 64 65 67 69 71 )
(61 63 65 66 68 70 72 )
(62 64 66 67 69 71 73 )
(63 65 67 68 70 72 74 )
(64 66 68 69 71 73 75 )
(65 67 69 70 72 74 76 )
```

(66 68 70 71 73 75 77)
(67 69 71 72 74 76 78)
(68 70 72 73 75 77 79)
(69 71 73 74 76 78 68)
(70 72 74 75 77 79 69)
(71 73 75 76 78 68 70)))
(defvar minor-scale '(
(60 62 63 65 67 68 70)
(61 63 64 66 68 69 71)
(62 64 65 67 69 70 72)
(63 65 66 68 70 71 73)
(64 66 67 69 71 72 74)
(65 67 68 70 72 73 75)
(66 68 69 71 73 74 76)
(67 69 70 72 74 75 77)
(68 70 71 73 75 76 78)
(69 71 72 74 76 77 79)
(70 72 73 75 77 78 68)
(71 73 74 76 78 79 69)))
(defvar minor-triad '((60 63 67) (61 64 68) (62 65 69) (63 66 70)
(64 67 71) (65 68 72) (66 69 73) (67 70 74)
(68 71 75) (69 72 76) (70 73 77) (71 74 78)))
(defvar major-triad '((60 64 67) (61 65 68) (62 66 69) (63 67 70)
(64 68 71) (65 69 72) (66 70 73) (67 71 74)
(68 72 75) (69 73 76) (70 74 77) (71 75 78)))

index-substitute

arglist: `(list indexes elements)`
package: JBS-CMI
menu: Utilities
6.07 - INDEX-SUBSTITUTE
This function allows you to replace a list of elements giving the nth positions.
In [list] you put a list.
In [indexes] you define where you want to make a substitution. The values here correspond to nth indexes. So 0 is the index if for the first element, 1 for the second, 2 for the third and so on.
In [elements] you put the elements that will be put in the given nth indexes set in [indexes]

last-n

arglist: `(list n)`
package: JBS-CMI
menu: Utilities

n last elements of a list

left-down-diagonal-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It reads the matrix through its diagonal, starting to the last element of the first line to the first element of the last line. Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result: —> 111 —> 22 —> 3

left-up-diagonal-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It reads the matrix through its diagonal, starting to the first element of the first line to the last element of the last line.
Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result: —> 333 —> 22 —> 1

maximum-segmentation

arglist: `(list)`
package: JBS-CMI
menu: Groups▶Segmentations
From PAolo Aralla MusicTopology 5.2.1 - MAXIMUM-MINIMUM-SEGMENTATIONS
(To better understand this function, please look also in mathematics, maxima and minima, known collectively as extrema.)
The function MAXIMUM-SEGMNTATION creates a segmentation based on the primitive forms of a given list of numbers. ATTENTION: this function works only with numbers (not with symbols). It creates a separate group of lists segmented before and after a point of maximum.

member-in-sublists?

arglist: `(item-or-items lists mode)`
package: JBS-CMI
menu: Utilities
6.3 - MEMBER-IN-SUBLISTS?
This function is a sort of general member test. It outputs T or NIL.

In [item-or-items] you put the element you want to be checked as included or excluded. In [lists] you put a list where to check if the item set in [item-or-items] is found or not. With the menu MODE you can choose between four different tests: - with EVERY this function puts T if the element set in [item-or-items] is in all sub lists; - with SOME or NOTEVERY this function puts T if the element set in [item-or-items] is in certain sub lists; - with NOTANY this function puts T if the element set in [item-or-items] is NOT in any sub lists.

minimum-segmentation

arglist: `(list)`
package: JBS-CMI
menu: Groups▶Segmentations
From PAolo Aralla MusicTopology
5.2.1 - MAXIMUM-MINIMUM-SEGMENTATIONS
(To better understand this function, please look also in mathematics, maxima and minima, known collectively as extrema.)
The function MINIMUM-SEGMNTATION creates a segmentation based on the primitive forms of a given list of numbers. ATTENTION: this function works only with numbers. It creates a separate group of lists segmented before and after a point of minimum.

numeric-comment

arglist: `(list gr)`
package: JBS-CMI
menu: Matrix▶Numeric-Comment
3.1.1 - NUMERIC-COMMENT
(Also know as Look-and-say sequence by John Horton Conway)
This function reproduces the Look-and-say sequence as shown.
Put 1 in [list] and choose which level of recursion you want to use with [gr].
To generate a member of the sequence from the previous member, read off the digits of the previous member, counting the number of digits in groups of the same digit.
For example:
1 is read off as 'one 1' or 11. 11 is read off as 'two 1's' or 21. 21 is read off as 'one 2, then one 1' or 1211. 1211 is read off as 'one 1, then one 2, then two 1' or 111221. 111221 is read off as 'three 1, then two 2, then one 1' or 312211.

numeric-comment-sort

arglist: `(list gr)`
package: JBS-CMI
menu: Matrix▶Numeric-Comment
3.1.2 - NUMERIC-COMMENT-SORT
This is a variation of the Look-and-say sequence by John Horton Conway.
Put 1 in [list] and choose which level of recursion you want to use with [gr].

To generate a member of the sequence from the previous member, read off the digits of the previous member, counting the number of digits in groups of the same digit., BUT STARTING ALWAYS FROM THE SUM OF MORE LITTLE NUMBER TO THE BIGGER ONE.
1 is read off as 'one 1' or 11. 11 is read off as 'two 1's' or 21. NOW SEE THE DIFFERENCE 21 is read off STARTING FROM THE MORE LITTLE NUMBER THAT IS 1 SO: as ' one 1' then one 2' or 1112. THEN AGAIN SARTING FROM THE MORE LITTLE: 1112 is read off as 'three 1, then one 2' or 3112. THEN AGAIN SARTING FROM THE MORE LITTLE: 3112 is read off as 'two 1, then one 2, then one 3' or 211213. THEN AGAIN SARTING FROM THE MORE LITTLE: 211213 is read off as 'three 1, then two 2, then two 3' or 312213. And so on.
A special property: After a while all sequences become the comment of themselves. Try to make the comment of this one: 2 1 3 2 2 3 1 4.

on-new-segmnetation

arglist: `(list)`
package: JBS-CMI
menu: Groups▶Segmentations
From PAolo Aralla MusicTopology
5.2.2 - ON-NEW-OLD-SEGMENTATIONS
(These two functions refer to the morphology theory developed by Paolo Aralla.)
The ON-NEW-SEGMENTATION creates a group on each NEW incoming element. To create this segmentation, it creates a group excluding the NEW element and another group including it.

on-new/new-old/analysis-segmentation

arglist: `(list)`
package: JBS-CMI
menu: Groups▶Segmentations
From PAolo Aralla MusicTopology
5.2.3 - ON-NEW-OR-OLD-NEW/OLD-ANALYSIS-SEGMENTATIONS
(These two functions refer to the morphology theory developed by Paolo Aralla.)
The ON-NEW/NEW-OLD-ANALYSIS-SEGMENTATION creates a new group on each NEW element coming out of the NEW-OLD-ANALYSIS iof the given list.

on-old-segmnetation

arglist: `(list)`
package: JBS-CMI
menu: Groups▶Segmentations
From PAolo Aralla MusicTopology
5.2.2 - ON-NEW-OLD-SEGMENTATIONS
(These two functions refer to the morphology theory developed by Paolo Aralla.)

The ON-OLD-SEGMNTATION creates a group on each OLD element appearing in the given list.

on-old/new-old/analysis-segmentation

arglist: `(list)`
package: JBS-CMI
menu: Groups▶Segmentations
From PAolo Aralla MusicTopology
5.2.3 - ON-NEW-OR-OLD-NEW/OLD-ANALYSIS-SEGMENTATIONS
(These two functions refer to the morphology theory developed by Paolo Aralla.)
The ON-OLD/NEW-OLD-ANALYSIS-SEGMENTATION creates a new group on each OLD element coming out of the NEW-OLD-ANALYSIS of the given list.

order-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
Probably quite a stupid function: it just gives back the matrix in the same order (list of lists) as it is. It's usefull in order to compare with the other functions that read the matrix.
Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result:
—> 1 11 111 —> 2 22 222 —> 3 33 333

print-lists

arglist: `(list)`
package: JBS-CMI
menu: Reading-Matrix▶Printing-Lists-&-Matrix
It prints out in the listener in a more readable way.

print-matrix

arglist: `(list)`
package: JBS-CMI
menu: Reading-Matrix▶Printing-Lists-&-Matrix
It prints in the listener in a matrix redeable format.

reverse-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It gives back the last line up to the first, keeping the same order of elements, inside each lines.
Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result:
—> 3 33 333 —> 2 22 222 —> 1 11 111

reversed-order-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It gives back the first line down to the last, reversing the order of elements, inside each lines.
Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result:
—> 111 11 1 —> 222 22 2 —> 333 33 3

reversed-order-reverse-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It gives back the last line up to the first, reversing the order of elements, inside each lines.
Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result:
—> 333 33 3 —> 222 22 2 —> 111 11 1

right-down-diagonal-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It reads the matrix through its diagonal, starting to the first element of the first line to the last element of the last line.

Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result: —> 1 —> 22 —> 333

right-up-diagonal-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It reads the matrix through its diagonal, starting to the last element of the first line to the first element of the last line. Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result: —> 3 —> 22 —> 111

tree-extract

arglist: `(structure index)`
package: JBS-CMI
menu: Utilities
6.2 - TREE-EXTRACT
This function is a sort a recursive nth.
In [structure] you put a list of lists. In [index] you define the nth of the nth. For instance if you put (1 1), that means that you want to extract the second (nth 1) of the second (nth1).

up-left-down-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix
It gives back the matrix reading in a diagonal way, going up from right to left while going down through the matrix.
Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result:
—> 333 —> 222 33 —> 111 22 3 —> 11 2 —> 1

up-right-up-reading-matrix

arglist: `(list)`
package: JBS-CMI
menu: Moving▶Reading-Matrix

It gives back the matrix reading in a diagonal way, going up from right to left while going down through the matrix.
Here is a very simple matrix:
BEGIN-OF-MATRIX 0 1 2 3 _____ 1, 1 11 111 _____ 2, 2 22 222 _____ 3, 3 33 333 _____ END-OF-MATRIX
Here is the result:
—> 333 —> 33 222 —> 3 22 111 —> 2 11 —> 1

write-grace-note-music-entity

arglist: `(rhythm pitches note-expressions grace-notes where? grace-note-expressions)`
package: JBS-CMI
menu: Write-Music-Entities
1.2 - WRITE-GRACE-NOTE-MUSIC-ENTITY
(Sorry, not so elegant this one...)
This function allows you to write in a simple mode a rhythmical sequence (using rhythm tree structures) synchronised with pitches and music expressions and inserting sequences of grace notes having music expressions too.
In [rhythm] you set the rhythm tree structure: in this example it is (1 1 1 1), that means that the measure will be divided in 4 equal pulses.
In [pitches] you set the corresponding pitches.
FIRST ATTENTION: if the number of pitches is smaller than the number of pulses, the last pitch will be repeated in order to get the same length of pulses. If the number of pulses is smaller than the number of pitches, the last pulse will be repeated to get the same length of the pitches.
In [note-expressions] you can define which music expression you want to be synchronized with pulses and pitches.
Syntactically the expressions have to be inside a parenthesis as shown: (:mf) or (:accent); but also like this: ((:accent :crescendo1) (:crescendo1) (:crescendo1) (:crescendo1 :sf)) (Please see the PWGL help and the 02-enp-constructor patch in order to learn how many ENP expressions are.)
SECOND ATTENTION: if the number of expression is smaller than the number of pulses or of pitches the last expression will be repeated in order to get the same length of pitches.
In [grace-notes] you can enter either a flat list of pitches (in this case it will consider as a single group of grace notes) or a list of lists of pitches (in this case each sub list will be considered as a separate group of grace notes).
In [where?] you have to define where the grace notes are appended. If you put 0, that means that the grace notes will appended to the nth 0 (the first) note you put in [rhythm]; if you put 1 it will be in the second place, 2 in the third and so on.
In [grace-note-expressions] you can define the grace note single group or a list of groups.
With this function: - you can decide single nth positions or groups of nth positions; - you can control if you have a single group of grace note or groups (a LIST of LISTS) of grace

notes; - you can choose the NOTES expressions; - you can choose the GRACE-NOTES expressions.
NOTE THIS: - In [where?] if you put a single number and in [grace-note-expressions] you have set a list of groups, only the group corresponding to with the nth in [3] will be chosen. - In [where?] if you have chosen a group of nth and in [grace-note-expressions] you have set only a group of grace notes, only this last one will be printed in the score in the first nth group set in [where?].
- In [note-expressions] you have multiple choices (the behaviour for NOTE expressions). 1 - if you use a single expression like (:fff) this one will be propagated to all notes; 2 - if you use a flat list of expressions like (:sf :accent) the list will associate (:sf) to the first note, and (:accent) for all other notes; 3 - if you use a single list of list like ((:sf :accent)) the double expression (:sf :accent) will be propagated to all notes; 4 - if you use a list of separate sub lists like ((:p) (:fermata :accent)) and the length of notes is bigger, the last expression - in this case (:fermata :accent) - will be propagated till the last note; 5 - if you use a list of separate sub lists like ((:p) (:f) (:mf) (:fermata :accent)) and the number of lists is equal to the length of notes, for each note you have a specific expression.
- In [grace-note-expressions] you have multiple choices (the behaviour for GRACE-NOTE expressions) ATTENTION: this behaviour is similar to NOTE expression but not exactly the same. 1 - if you use a single expression like (:fff) this one will be propagated to all grace-notes; 2 - if you use a single list of list like (:sf :accent) the double expression (:sf :accent) will be propagated to all grace-notes; 4 - if you use a list of separate sub lists like ((:mf) (:accent :slur1)) and the length of notes is bigger than the length of expressions, the whole group of expressions will be propagated till the last grace-note. ATTENTION: if the length of the sub-group grace-notes is smaller than the list of expressions, the list will stop accordingly to the length of the grace-note sub group. ATTENTION TOO: if the length of a sub group is bigger than the length of the expression list, the last expression will be propagated to the last grace-note. 5 - if you use a list of separate sub lists like ((:mf) (:accent) (:slur1 :crescendo1) (:crescendo1) (:crescendo1 :ff)) and the number of lists is equal to the length of grace-note list, for each grace-note you have a specific expression.

write-music-entity

arglist: `(rhythm pitches note-expressions)`
package: JBS-CMI
menu: Write-Music-Entities
1.1 - WRITE-MUSIC-ENTITY
This function allows you to write in a simple mode a rhythmical sequence (using rhythm tree structures) synchronised with pitches and music expressions.
In [rhythm] you set the rhythm tree structure: in this example it is (1 1 1 1), that means that the measure will be divided in 4 equal pulses.
In [pitches] you set the corresponding pitches.
FIRST ATTENTION: if the number of pitches is smaller than the number of pulses, the last pitch will be repeated in order to get the same length of pulses. If the number of

pulses is smaller than the number of pitches, the last pulse will be repeated to get the same length of the pitches.
In [note-expressions] you can define which music expression you want to be synchronized with pulses and pitches. Syntactically the expressions have to inside a parenthesis as shown: (:mf) or (:accent); but also like this: ((:accent :crescendo1) (:crescendo1) (:crescendo1) (:crescendo1 :sf)) (Please see the PWGL help to learn how many ENP expressions are.)
SECOND ATTENTION: if the number of expression is smaller than the number of pulses or of pitches the last expression will be repeated in order to get the same length of pitches.

Box Index