Marco Stroppa

Structure, Categorization, Generation and Selection of Vertical Pitch Structures (VPS)

A Musical Application in
Computer Assisted Composition

(November 1988)

(IRCAM Document)

This text is a proposal of a relatively short and time-limited project of C.A.O. in the spirit of a "Reduced Musical Application" (RMA) or "maquette" which was called for by Andrew Gerzso as part of the initial phase of the IRCAM C.A.O. project. In this context, "Reduced" means that :

- there will be no special efforts made on the quality of the graphic interface, let alone the use of what already exists
- there will be no special efforts made on the efficiency and/or beauty of the formalization or of the programming phases
- only symbolic pitches will be used

In spite of its "Reduced" characteristics, however, the system will be built in a way as general as possible, so that both perceptual and compositional experiments of various kinds can be done by simply varying its data structures.
A first version will be implemented in Prolog on a Macintosh by Francis Courtot who will collaborate with me during the whole RMA. If the results are satisfactory, I shall first use them to deal with Vertical Pitch Structures (VPS) at the level of pre-compositional material of my piece for spatialized ensemble which will be premiered during the 1989 Festival d'Automne.

1) CONVENTIONS OF NOTATION

1.1) Pitches

Pitch names are upper-case and follow the Italian convention (i.e. DO, RE, MI, etc.). The half-tone sharp and flat signs are "#" and "b". Quarter tones are notated with an upward or downward arrow (i.e. " ≠ " or " | ") which means respectively higher and lower than the reference pitch (for instance, "DO # |" is the same as "DO ≠". Middle C is DO4.

1.2) Intervals

Intervals use two different notations.

1.2.1) Octave - based notation

<N> <ALT> ≠ / | (<OCT>)

where :

N : integer interval size, modulo 7 (1 for unisons, 2 for seconds, etc.). The number is positive if the interval is ascending, negative if it is descending.

ALT : for seconds, thirds, sixths and sevenths it may be either "+" or "-" (major or minor interval)
for unisons, fourths and fifths it may either be absent, or be "+" or "-" (natural, augmented or diminished interval).

≠/| : upward or downward quarter-tone alteration (present only when needed).

OCT : integer number of octaves which separate the two pitches. When it is zero, it will be omitted.

Augmented or diminished seconds, thirds, sixths ans sevenths are reduced to the corresponding adjacent minor or major interval.

1.2.2) Absolute-size notation

<N> . <M>

where :

N : integer number of semitones separating the two pitches (positive if the interval is ascending, negative if it is descending).

M : less-than-semitone extension, present only when required (".5" for quarter tones, ".25" for eighths of tone, etc...).

Examples :
DO4 - LAb4 = octave-based notation : 6- ; absolute-size notation : 9
SOL5 - RE3 = " " " : -4 (2) ; " " " : -
29
DO4 - DO5≠ = " " " : 1≠ (1) ; " " " :
12.5
FA4 - SOL#5 = octave-based notation : 3- (1) ; absolute-size notation :
15

1.3) Other Conventions

All the VPSs are generated starting from middle C and consist in a list of pitches with octave indication, separated by a comma and enclosed within parentheses. The VPS which most of the examples will refer to is : (DO4, LAb4, RE5, SOL5, DO#6).
VPS's attributes freely follow a Lisp-based format (for instance, lists of values are parenthe sized). Please note that the conventions of notations used in this text may not correspond to the ones adopted by the actual prolog-based system.

2) BACKGROUND

2.1) Proto-OIMs

The root of the project is to be found at the end of my article on "Musical Organisms" (OIMs), where I pointed to some possible extensions of the framework which I had put forward. One such extensions was to apply it to more traditional musical parameters (pitch, duration, dynamics, rhythm, harmony and the like). In this text, I shall be only concerned with the structure of sets of vertical pitches (i.e. pitches which are superposed, or which are dealt with as though they were superposed) from the standpoint of their attributes and of their underlying identity. Such sets, or VPSs, will be considered as simplified instances of OIMs, or "proto-OIMs".

2.2) The Project

The first phase of the project will implement a certain amount of algorithms which specify how to compute the value of each VPS's attribute from a quantitative point of view. They will be described in the next section. The input of this phase is a single VPS, whereas the output will be a list of all its attributes with their own value.

The second, and hardest, phase will consist in reversing the direction of the first one. After specifying the value of some attributes, a list will be generated containing all the VPSs that fulfill the input conditions.

The final phase will make use of some software, which F. Courtot already wrote for other purposes, so as to transform and/or filter out the VPSs generated by the second phase according to certain constraints.

3) STRUCTURE AND CATEGORIZATION

This section will discuss the attributes of a VPS which give rise to its identity, as well as how to assign them a value. All but the last one are very simple to compute and deal with rather macroscopic characteristics.

3.1) Note List (NL)

It is the most straightforward attribute, that is the list of a VPS's pitches, from bottom to top, contiguously and without octave position.

Example : from (DO4, LAb4, RE5, SOL5 DO#6), NL = (DO, LAb, RE, SOL, DO#).

3.2) Interval Lists (CIL, AIL, GIL)

The intervallic structure of a VPS will be examined from three different perspectives, and as many attributes. Because of the two notations for intervals, each attribute will be made of two lists, one per notation.

Contiguous Interval List (**CIL**). It contains only a VPS's contiguous intervals, i.e. the intervals between the first and the second pitch, the second and the third one, and so on. For instance, the CIL of the reference VPS is : ((6-, 4+, 4, 4+) (8,6,5,6)).

Anchored Interval List (**AIL**). It is constituted by the intervals between a single reference pitch of a VPS and all the other pitches. The position of the reference pitch with respect to the VPS is placed at the beginning of the list (1 for the lowest pitch, 2 for the second pitch from the bottom, and so on).
Examples : from the lowest pitch of the reference VPS, AIL = (1 (6-, 2+(1), 5(1), 2-(2)) (8, 14, 19, 25)) ; from the highest pitch, AIL = (5 (-4+, -7+, -4(1), -2-(2)) (-6, -11, -17, -25)).

Global Interval List (**GIL**). It takes into account all the VPS's intervallic combinations, in only one direction. For instance, from (DO4, LAb4, RE5, SOL5, DO#6), all the ascending intervallic combinations are : DO - LAb, DO - RE, DO - SOL, DO - DO#, LAb - RE, LAb - DO#, RE - SOL, RE - DO#, SOL - DO#, and the corresponding GIL will be : ((6-, 2+(1), 5(1), 2-(2), 4+, 7+, 4(1), 4, 7+, 4+) (8, 14, 19, 25, 6, 11, 17, 5, 11, 6).

3.3) Number of Notes (NN), Number of Intervals (NCIL, NAIL, NGIL)

They are integer numbers representing nothing but what they say. Due to three interval lists, there will be three kinds of number of intervals :
NCIL = **NAIL** = (NN - 1) and **NGIL** = 1 + 2 + … + (NN - 1).

Example : NN = 5, NCIL = 4, NAIL = 4, NGIL = 10.

3.4) Surface (S)

This attribute measures the interval between a VPS's lowest and highest pitches, expressed in both notations.

Example : S = (2-(2), 25).

3.5) Density (D)

It is computed as NN / (S+1). A unit density will therefore indicate a chromatic cluster. A cluster of quarter tones will correspond to a density of 2, and so on.

Example : D = 5 / (25+1) = 0.192.

3.6) Coefficient of Homogeneity (H)

It is an attribute with two values. The first one is the difference between the largest and the smallest interval of a VPS's CIL (using both notations). The smaller the value, the more homogeneous (i.e. made of intervals of similar size) the VPS. The second value contains the actual size of both the largest and the smallest intervals separated by a "/".

Example : H = ((2+, 3) (6-/4, 8/5)).

3.7) Coefficient of Stability (CS)

It is the most complex attribute of a VPS and a measure of the interactions between the qualities of all the intervals contained in a GIL.

In order to understand its significance properly, one should start from the following considerations : let's suppose that all the possible intervals within an octave of a VPS be assigned a numeric weight, say between 0 and 10, which measures how "stable" or "unstable" that interval is. If, for instance, a low weight is a sign of a higher stability than a large weight, then 0 corresponds to a completely stable interval, whereas 10 will point to a totally unstable one.

When all the weights are assigned, they can be ordered so as to constitute a "*space of stabilities*" (**S-space**), ranging from 0 to 10. The form of the S-space (linear, exponential, hand-made, etc.) as well as the choice of the assignments are completely free. For instance, they may result from compositional experience or intuitive rules, derive from perceptual models or even be computed so as to fit the data of psychological experiments on the stability of VPS's.

As a concrete example, the following are two semitonal S-spaces (a linear and a hand-made one) in which the assignments approximately follow the critical-band model :

Interval Type	**Linear space**	**Hand-made space**
2-	10	10
2+	9.09	9.5
7+	8.18	9
7-	7.27	8
4+	6.36	7
3-	5.45	4
3+	4.55	3.5
6-	3.64	3
6+	2.73	2.5
4	1.82	1
5	0.91	0.5
1	0	0

The perception of the quality of an interval, however, depends not only on its type modulo 7 (i.e. reduced within an octave), but also on how many octaves apart the two pitches are. The farther apart, the weaker the quality and the lighter the weight. For instance, the interval DO2 - DO#2 will not sound the same as DO2 - DO#7, even though the names of the pitches do not change. On the other hand, their respective qualities will be closer to each other than to the quality of the interval DO2 - SOL2.

To account for this difference within the same type of interval, another dimension must be added to the S-space, one corresponding to an "*octave scaler*" (**OS**). The OS's value is 1 for the first octave and decreases to 0 when the distance gets so large that the interval's quality is lost. As in the previous cases, the form of the evolution from 1 to 0 is free and need not be the same for all the intervals.

The actual weight, therefore, is the product of an interval's "*natural weight*" (**NW**), i.e. the one assigned to its type modulo 7, with OS. As an example, here is the S-space which I used in my string quartet : it is a semitonal space, loosely based on the critical-band model. OS is the same for all the intervals. Its evolution is defined intuitively.

Interval type	**NW**
2-	10
7+	9.5
2+	7.5
7-	7
4+	4
4	3.5
6+	3
6-	2.5
5	2
3-	1.5
3+	1
1	0

Distance in octaves	**OS**
0	1
1	0.85
2	0.7
3	0.45
4	0.25
5	0.1
>5	0

The whole two-dimensional S-space is the following :

Interval Type	**Octave : 0**	**1**	**2**	**3**	**4**	**5**	**>5**
(2-)	10	8.5	7	4.5	2.5	1	0
(7+)	9.5	8.075	6.65	4.275	2.375	0.95	0
(2+)	7.5	6.375	5.25	3.375	1.875	0.75	0
(7-)	7	5.95	4.9	3.15	1.75	0.7	0
(4+)	4	3.4	2.8	1.8	1	0.4	0

(4)	3.5	2.975	2.45	1.575	0.875	0.35	0
(6+)	3	2.55	2.1	1.35	0.75	0.3	0
(6-)	2.5	2.125	1.75	1.125	0.625	0.25	0
(5)	2	1.7	1.4	0.9	0.5	0.2	0
(3-)	1.5	1.275	1.05	0.675	0.375	0.15	0
(3+)	1	0.85	0.7	0.45	0.25	0.1	0
(1)	0	0	0	0	0	0	0

By definition, CS is a single number between 0 and 10 computed as follows :

$$\mathbf{CS} = \frac{\mathbf{\sum W\ (GIL)}}{\mathbf{NGIL}}$$

where **∑ W (GIL)** is the sum of the actual weights of all the intervals contained in GIL.

CS is therefore a synthetic measure of how "stable" a VPS is with respect to a given S-space. When the S-space is also perceptually significant, CS is very strongly correlated with a VPS's perceptual identity.
For instance, with respect to the space above, the CS of the reference VPS is :

(2.5 + 6.375 + 1.7 + 7+ 4 + 9.5 + 2.975 + 3.5 + 9.5 + 4) / 10 = 5.105

An important feature of such a measure is that it allows one to classify all the possible CSs into a finite amount of groups which represent noticeable changes of a VPS's stability. As an example, one might establish the following six non-overlapping groups: CS >3, 3.5 (+/-0.5), 4.4 (+/-0.4), 4.95 (+/-0.25), 5.9 (+:-0.7), >6.6. Each group is identified by a <u>focal value</u> and by a <u>maximum range of variation</u> around it. If the focal CS corresponds to the value of the most typical VPS of that group, the range of variation defines the space within which a VPS is considered to be part of that group. So, the stability of two VPSs belonging to different groups will be substantially different, whereas the stability of various VPSs whose CS falls within the same group will rather be perceived as a sort of variation around a unique center. Unity and variety are therefore nicely assured.

4) GENERATION

To make it clear how one might use a system based on what I just described, I will briefly discuss two simple examples, one in each direction, presented in a language-independant syntax, where "?" and "---->" respectively precede an input command and the result of the computation.

Example 1 : from a given VPS to its attributes.

```
?       (DO4, LAb4, RE5, SOL5, DO#6)

--->    NL =    (DO, LAb, RE, SOL, DO#)
        NN =    5
        CIL =   ((6-, 4+, 4, 4+) (8,6,5,6))
        NCIL = 4
        AIL =   (1 (6-, 2+(1), 5(1), 2-(2)) (8, 14, 19, 25))
                (2 ... )
                  ...
                (5 (-4+, -7+, -4(1), -2-(2))  (-6, -11, -17, -25))
        NAIL = 4
        GIL =   ((6-, 2+(1), 5(1), 2-(2), 4+, 7+, 4(1), 4, 7+, 4+)
                 (8, 14, 19, 25, 6, 11, 17, 5, 11, 6))
        NGIL = 10
        S =     (2- (2), 25)
        D =      0.192
        H =      ((2+, 3) (6-/4, 8/5))
        CS =    5.105
```

Example 2 : from the specification of some attributes to the generation of all the VPSs which satisfy the input constraints. The specification can be absolute (e. g. NN = 4) or range-based (e. g. : either NN = [3 : 5, 7], meaning all the values between 3 and 5 and 7, or NN = 3 (2), meaning all the values between 3 - 2 and 3 + 2.

```
?      (NN = [4 : 5 ], CS = 4.3 (.25), S = 37 (10))

--->   <VPS 1>
       <VPS 2>
       …
```

5) SELECTION

In the final phase of the project, the VPSs which will have already been generated, will be further processed according to simple criteria and constraints on their attributes. Because their ultimate definition can be specified only within a compositional project, I will now limit myself to just a few isolated examples. During real applications, the various examples will be combined with one another.

5.1) Blind Transposition

Since, by rule, a VPS is always generated starting from middle C, this operation simply consists in transposing it by a desired interval.

5.2) Constraint-based Selection

It consists in a series of predicates to keep or eliminate the VPSs which do not satisfy certain conditions, without transposing them.

Examples in natural language :

1. Keep only the VPSs whose CIL contains a major third.
2. Keep only the VPSs containing a minor sixth between the first and fourth pitch.
3. Eliminate the VPSs whose GIL contains an augmented fourth.
4. Eliminate the VPSs whose NL contains either a LAb or the FA#.

5.3) Guided Transposition

It is a kind of combination of the two previous operations. During "Guided Transposition" a VPS is transposed, rather than kept or eliminated, so as to fulfill the input constraints on its attributes.

The following are some examples of single-constraint instructions (the result of their application to the reference VPS is shown within parenthesis) :

1. The third pitch must be a LAb

   (FA#3, RE4, LAb4, DO#5, SOL5)
2. The second-to-the-last pitch must be a RE7

   (SOL5, MIb6, LA6, RE7, LAb7)
3. The fourth SOL-DO must exist anywhere in the VPS's CIL

   (FA4, DO#5, SOL5, DO6, FA#6)
4. The major seventh RE6-DO#7 must exist and be non-contiguous

   (FA#5, RE6, LAb6, DO#7, SOL7)

6) EXTENSIONS

Even though it is not within the scope of the present RMA, I will nonetheless briefly skim through some extensions which immediately stem from the approach that has been put forward so far.

The first one is to consider timbre as a kind of special, fused VPS, where "P" would stand for "partial", rather than for "pitch". The VPS's different attributes can then be used to characterize timbral structures from a perceptual and/or compositional standpoint. This extension is particularly well suited to my own compositional framework, due to the close interrelationships between harmony and timbre.

Another extension is to enlarge the S-space to a four-dimensional structure, by adding two more scalers, an "Exposure Scaler" and a "Register Scaler". The former will account for the relative position of a given interval within a VPS. Since intervals on the fringes are more easily perceivable, they should count more than inner intervals. The latter will refer to the absolute position of a VPS in the frequency space. The overall instability will be increased if the VPS is transposed towards the low register whereas it will be decreased if it is transposed towards the high register.

Finally, one may wish to have a graphic interface to determine the various dimensions of an S-space, so that function manipulation procedures can be used to modify their definition.

Addendum

Vertical Structures: Pitches and Spectra
User's Guide and Musical Applications
May 1998

INTRODUCTION

This text presents that part of Chroma that is dedicated to chords and spectra and to their implementation as polymorphic CLOS classes. It will follow a step-by-step bottom-up process raising from low-level concepts to my usage in both instrumental music (i.e. more standard CAO) and electronics music (synthesis control).

PITCHES AND INTERVALS: REPRESENTATION

Pitches are represented using either the Italian or the American spelling. The two spellings can be mixed within a single chord. It's a little confusing, but there is no problem for the system.

The class "symbolic-pitch" is used as the internal representation and is not to be addressed directly by the user. Normally the pitches are part of a VPS and are rarely employed alone.
EX: (setf note1 'DO)

Pitches

Possible names for pitches without alterations are: DO/UT/C, RE/D, MI/E, FA/F, SOL/G, LA/A, SI/B.

The alterations must be attached directly to the pitch without any space. They can be the following letters: "b, B, f, F" for flat and "d, D, s, S" for sharp.

An octave number can follow the pitch and transform it into an absolute pitch. When no octave is specified, the octave "0" is taken by default.

All the pitches may either be defined as lisp symbols (setf pitch 'DO) or as lisp strings (setf pitch "DO"). Hence, valid representations are: DOd, "Db2", MIB4, FS, "Ab2", etc.

Micro tones

Micro tones are represented by a lisp *cons* objects, whose *car* is a valid pitch symbol and whose *cdr* is either an integer number or the symbols "q" or "-q". The integer number indicates a deviation in cents, the symbols represent a quarter-tone higher and lower than the symbolic pitch.

Valid microtonal representations are: (REb . q), ("Af" . -6), (REb2 . 10), ("Gd4" . -q), etc.

Intervals

Intervals can be represented in the following ways:

REDUCED
• one of the these symbols: 1, 2-, 2+, 3-, 3+, 4-, 4, 4+, 5-, 5, 5+, 6-, 6+, 7-, 7+
They mean: unison, minor second, major second, minor third, major third, diminished fourth, ordinatry fourth, augmented fourth, etc.
These intervals are always well-tempered and within one octave.

EXPANDED
• a list containing one of the symbols above, an octave number and an optional frequency deviation in cents.
The octave number gives the distance in octaves. Hence, a minor ninth would be written (2-1), meaning a minor second one octave apart. In the frequency deviation, the symbols “q” and “-q” are not accepted.

EX of valid intervals: 3+, (3+ 2), (4+ 0 -12), 5-, etc.

PITCH CONVERSIONS

Different conversions are available. They are all in the form [source]**->**[destination] (ex. fq->midi). Unavailable direct conversions should use two direct conversions (ex. to convert from frequency to interval, one should convert from frequency to ratio or pitch and the from ration or pitch to interval).

Available representations:

• **fq:** frequency [Hz].
• **pch**: pitch (see above).
• **itvl**: interval (see above).
• **midi:** midi (with decimals if non tempered values are needed).
• **ratio**: the numerical ratio between the second and the first argument.
• **semitones**: the number of semitones (positive if ascending, negative if descending) between the first and the second argument.

Summary of the available conversions

• = yes, - = not available

------->	fq	pch	midi	ratio	itvl	semitones
fq	\	*	*	*	-	-
pch	*	\	*	-	*	*
midi	-	*	\	-	-	*
ratio	*	-	-	\	*	*
itvl	*	-	*	*	\	*
semitones	-	-	-	*	*	\

FREQUENCY

(fq->pch freq [approx])
freq: single frequency or list of frequencies
approx: approximation when converting [cents]. An approximation of 50 will round the result up to a quarter tone. Default: 0 (=> the highest precision).
Remark: one can also include symbolic
EX: (fq->pch 443), (fq->pch ‘(441 442 443) 25), (fq->pch ‘(441 DO2 (RE3 . 23) 460) 25) [also possible, pitches will not be changed!], etc.

(fq->midi freq)
freq: single frequency or list of frequencies
EX: (fq->midi 443), (fq->midi '(441 442 443)), etc.

(fq->ratio freq)
freq: list of at least 2 frequencies
Result**:** always a list of N-1 elements
NB: values are always positive unless one of the frequencies are negative. Values between 0 and 1 indicate descending ratios, 1 being the unison.
EX: (fq->ratio '(443 552 234)), etc.

PITCH
NB: The argument of the "pitch" conversion can always be either a pitch or a number, usually meaning a frequency [Hz]. In this case, however, the value will be treated differently depending on the conversion.

(pch->fq pitch [diapason])
pitch: single pitch or list of pitches or frequencies. Frequencies will return the same unconverted value.
diapason: reference frequency for "LA4" [Hz]. Default: 440.
EX: (pch->fq "DO2"), (pch->fq '(DO3 . -23)), (pch->fq '(RE2 440 (MIb3 . 23) 550 SI2) 442), etc.

(pch->midi pitch [diapason])
pitch: single pitch or list of pitches or frequencies. Frequencies will return the same unconverted value. However, in this case the list will contain a mixture of midi values (ex. 69, 72, etc.) and frequencies in Hz that is not very useful.
diapason: reference frequency for "LA4" [Hz]. Default: 440.
EX: (pch->midi 'DO2), (pch->midi '(DO2 (RE3 . 8)) 442), etc.

(pch->itvl pitch)
pitch: list of at least 2 pitches or frequencies.
Bug: due to the internal implementation, here the frequencies will be considered MIDI values and not proper frequencies.
Result: always a list of N-1 elements. Interval is always in its expanded form (ex. (2 0) for a minor second).
EX: (pch->itvl '(DO2 RE2)), (pch->itvl '(LA4 (MIb3 . -12) SI2)), (pch->itvl '("DO3" ("MIb2" . -q) 69 SI2)), etc.

(pch->semitones pitch)
pitch: list of at least 2 pitches or frequencies.
Bug: due to the internal implementation, here the frequencies will be considered MIDI values and not proper frequencies.
Result: always a list of N-1 elements.
EX: (pch->semitones '(DO2 RE2)), (pch->semitones '(LA4 (MIb3 . -12) SI2)), (pch->semitones '("DO3" ("MIb2" . -q) 69 SI2)), etc.

MIDI
(midi->pch midi [approx])

midi: single midi value or a list of values.
EX: (midi->pch 69), (midi->pch 69.234), (midi->pch '(69 71.2 44)), (midi->pch 69.123 10), (midi->pch '(69 71.2 44) 50), etc.

(midi->semitones midi-list)
midi-list: list of at least 2 elements
Result**:** always a list of N-1 elements
NB: positive values indicate an ascending interval, negative values a descending one. Unison is 0.
EX: (midi->semitones '(69 70 73 45)), etc.

RATIO
(ratio->fq ratio ref-fq)
ratio: a single ratio or a list of contiguous ratios
ref-fq: starting frequency. The new frequencies will be computed by multiplying the current frequency with the current ratio.
Result**:** always a list of N+1 elements
EX: (ratio->fq '(0.5 1.5 2.0) 440) -> (440 220 330 660)
(ratio->fq () 440) -> (440.0)
(ratio->fq 0.5 440) -> (440.0 220.0), etc.

(ratio->itvl ratio)
ratio: a single ratio or a list of ratios
EX: (ratio->itvl '0.5) -> (1 -1)
(ratio->itvl '(0.5)) -> ((1 -1))
(ratio->itvl '(3.4 4.5 0.99)) -> ((6+ 1 19) (2+ 2 4) (1 0 -17)), etc.

(ratio->semitones ratio)
ratio: a single ratio or a list of ratios
EX: (ratio->semitones 0.5) -> -12
(ratio->semitones '(0.5 1.6 2.0) -> (-12.0 8.136862 12.0), etc.

INTERVALS
(itvl->fq itvl ref)
itvl: single interval symbolic interval or list of contiguous intervals.
ref: starting frequency.
Result**:** always a list of N+1 elements
Bug: a single interval must be a unique symbol (ex. '6+); if it is microtonal (ex. (6+ 0 -12)) it has to be used in the form in a list, otherwise the system will treat the argument as a list of symbolic intervals.
EX: (itvl->fq '6+ 440) -> (440.0 739.988)
(itvl->fq '(6+ 3- (3- 0 -12)) -> (440.0 739.988 880.0 1039.2735), etc.

(itvl->midi itvl ref)
itvl: single interval symbolic interval or list of contiguous intervals.
ref: starting midi note
Result**:** always a list of N+1 elements

Bug: a single interval must be a unique symbol (ex. ‘6+); if it is microtonal (ex. (6+ 0 -12)) it has to be used in the form in a list, otherwise the system will treat the argument as a list of symbolic intervals.

EX: (itvl->midi ‘6+ 60) -> (60 69)
(itvl->midi ‘(6+) 60) -> (60 69)
(itvl->midi ‘((6+ 0 12)) 60) -> (60 1728/25)
(itvl->midi ‘(6+ (3- 0 -50) (1 0 50)) 60) -> (60 69 143/2 72), etc.

(itvl->ratio itvl)
itvl: single interval symbolic interval or list of contiguous intervals.
Result**:** always a list of N elements
Bug: a single interval must be a unique symbol (ex. ‘6+); if it is microtonal (ex. (6+ 0 -12)) one has to use the form in a list, otherwise the system will treat the argument as a list of symbolic intervals.
EX: (itvl->ratio ‘6+) -> (1.68179)
(itvl->ratio ‘(2- (3- 1) (2- 0 -50))) -> (1.059 2.378 1.029), etc.

(itvl->semitones itvl)
itvl: single interval symbolic interval or list of contiguous intervals.
Result**:** always a list of N elements
Bug: a single interval must be a unique symbol (ex. ‘6+); if it is microtonal (ex. (6+ 0 -12)) one has to use the form in a list, otherwise the system will treat the argument as a list of symbolic intervals.
EX: (itvl->semitones ‘6+) -> (9)
(itvl->semitones ‘(2- (3- 1) (2- 0 -50))) -> (9 15 1/2), etc.

SEMITONES
(semitones->ratio val)
val: single number of semitones or list of semitones.
EX: (semitones->ratio 6) -> 1.414
(semitones->ratio ‘(6)) -> (1.414)
(semitones->ratio ‘(6 1 0 12) -> (1.414 1.059 1 2), etc.
(semitones->ratio ‘(6.1 1.5 0.5 -12) -> (1.422 1.090 1.029 1/2), etc.
etc.

(semitones->itvl val)
val: single midi value or a list of values.
EX: (semitones->itvl 6) -> (4+ 0)
(semitones->itvl ‘(6)) -> ((4+ 0))
(semitones->itvl ‘(6 1 0 12) -> ((4+ 0) (2- 0) (1 0) (1 1)), etc.
(semitones->itvl ‘(6.1 1.5 0.5 -12) -> ((4+ 0 10) (2- 0 -50) (1 0 50) (1 1)),

POLYMORPHISM
All the system is conceived polymorphically, so that several types of data can be processed by the same method. The result will be of the same type as the input data. For example, if the function “transpose” is applied to a single pitch, the result will be another single pitch, whereas if the same function is applied to a chord, the result will be another chord, with all the pitches correctly transposed.

Some methods are defined only for a given representation (ex. get-cs works only with spl's, see below). When passing a different representation

VPS: MULTIPLE REPRESENTATIONS
A VPS is a Vertical Pitch Structure (see below) and is implenented as a set of CLOS classes.
Class structure:

VS	CHORD	SPL
		RPL
		CIL
		AIL
		PPL
	SPECTRUM	FQL
		CRL
		ARL

Schematic Representation
We will first describe the procedures to instanciate and access the objects using CLOS. Further down we will present a "user-friendlier" interface providing some macros that mask the CLOS syntax and do not require keywords.

VS: Vertical Structure, the root of everything. The only test performed at this stage is on the existance of the input arguments + some very generic functions such as print.

VPS: Vertical Pitch Structure, the son of the root. For "vertical" we mean that it consists of a list of items that do not repeat themselves (no unisons, for example) and are in an ascending order, either explicit or implicit. Other information, such as the current diapason is also initialized here.

CHORD: A list of Pitches of symbolic Intervals (formerly called a VPS).
- **SPL:** Symbolic Pitch List, a list of symbolic pitches. Ex. '(RE3 LA3 FAd4)
- **RPL:** Relative Pitch List, a list of pitches whose octave number is relative to the first pitch in the list. Ex. '(RE LA FAd1). Here the F sharp is 1 octave above the low D.
- **CIL:** Contiguous Interval List, a list of contiguous intervals. Ex. '(5 6+). This object contains N-1 elements.
- **AIL:** Anchored Interval List, a list of intervals with respect to a reference pitch (anchor). Ex. '(-4+ 2- 7-) with anchor = SOLd3.
- **PPL:** Pure Pitch List, internal represention not to be used.

SPECTRUM: A list of Frequencies of Ratios.
- **FQL:** Frequency List, a list of frequencies [Hz]. When this class is used to represent analysis data (from AudioSculpt, for example) two parallel lists of Amplitudes and Band Widths are associated to the list of frequencies. They must have the same number of elements. Ex. '(100 150 300 450).

CRL: Contiguous Ratios List, a list of contiguous ratios. Ex. ‘(1.5 2 1.6). This object contains N-1 elements. Since the elements must be in ascending order, the ratios are always > 1.0

ARL: Anchored Ratios List, a list of ratios with respect to a reference frequency (anchor). When the anchor is low, this corresponds to a spectrum (anchor = f0). Ex. ‘(0.5 0.75 1.5 2.22) with an anchor of 200 or ‘(10 15 30 45) with an anchor (f0) of 10.

Schematic examples

;**SPL** (Symbolic Pitch List)
```
;    '(DO4 LAb4 RE5 SOL5 REb6)
;    '(DO4 (LAb4 23) RE5 (SOL5 -12) REb6)
```

;**RPL** (Relative Pitch List)
```
;    '(DO LAb RE1 SOL1 REb2)
;    '((DO . 23) (LAb . -34) RE1 SOL1 (REb2 . 77))
```

;**PPL** (Pure Pitch List)
```
;    '(DO LAb RE SOL REb)
;    '((DO . 23) (LAb . -34) RE SOL (REb . 77))
```

;**CIL** (Contiguous Interval List)
```
;    '(6- 4+ 4 4+)
;    '((6- 0 12) (4+ 1 -21) (4 0 7) (4+ 1 -2))
```

;**AIL** (Anchored Interval List)
```
;    '(-6+ -2- 4 7- (3+ 1)) 'LA4
;    '((-6+ 0 12) -2- (4 0 -22) 7- (3+ 1 17)) + reference: 'LA4
```

;**FQL** (Frequency List)
```
;    '(261 415 587 784 1108)
```

;**CRL** (Contiguous Ratios)
```
;    '(1.5873 1.4142 1.3345 1.4145)
```

;**ARL** (Spectrum / Anchored Ratios)
```
;    '(2.61 4.153 5.873 7.84 11.07) + reference: 100.0
```