OMPitchField Reference

for version 2.0

contents

list-t-primeforms

inputs

card	cardinality of desired primeforms	integer in [2, n - 2] (or a list of such integers, to produce results for more than one cardinality)

optional inputs

n	modulus of the pc space	*menu selection:* 12 (default) or 24
tag	option to insert :t at head of each primeform	*menu selection:* NONE or :T

outputs

	list of all mod-n t-primeforms of specified cardinality	list (or list of lists) of pcsets (each pcset a list of mod-n integers)

A t-setclass is a family of pcsets related to one another by transposition, and its t-primeform is a member of the family, designated to represent the entire family.

This function lists the designated representative member of every t-setclass with the specified cardinality card. This input parameter, normally an integer, can also be a list of integers, to produce results for more than one cardinality. The modulus n can be one of two preset values: 12 for semitones or 24 for quartertones. The tag option facilitates the construction of the incl-classreps parameter required as an input to several functions in OMPF (see the entry for incl-classrep).

Compared to the orbites function in the Zn library, this function is much faster (because it looks up values in a database rather than calculating them on the fly) but much less general (mod-12 or -24 only).

`list-ti-primeforms`

inputs

`card`	cardinality of desired primeforms	integer in `[2, n - 2]` (or a list of such integers, to produce results for more than one cardinality)

optional inputs

`n`	modulus of the pc space	*menu selection:* `12` (default) or `24`
`tag`	option to insert `:t` at head of each primeform	*menu selection:* `NONE` or `:T`

outputs

	list of all mod-`n` t-primeforms of specified cardinality	list (or list of lists) of pcsets (each pcset a list of mod-`n` integers)

A ti-setclass is a family of pcsets related to one another by transposition and/or inversion, and its ti-primeform is a member of the family, designated to represent the entire family.

This function lists the designated representative member of every ti-setclass with the specified cardinality `card`. This input parameter, normally an integer, can also be a list of integers, to produce results for more than one cardinality. The modulus `n` can be one of two preset values: `12` for semitones or `24` for quartertones. The `tag` option facilitates the construction of the incl-classreps parameter required as an input to several functions in OMPF (see the entry for `incl-classrep`).

Compared to the `orbites` function in the Dn library, this function is much faster (because it looks up values in a database rather than calculating them on the fly) but much less general (mod-12 or -24 only).

t-primeform

inputs

`pcset`	any member of the t-setclass whose primeform is sought (or a list of these)	list (or list of lists) of mod-`n` integers

optional inputs

`n`	modulus of the pc space	positive integer (12 by default)

outputs

	primeform of the t-setclass to which `pcset` belongs (or a list of these)	list (or list of lists) of mod-`n` integers

A t-setclass is a family of pcsets related to one another by transposition, and its t-primeform is a member of the family, designated to represent the entire family. The t-primeform algorithm selects the pcset whose elements are maximally close to zero in a particular sense.

Will tolerate integers out of the mod-`n` range in `pcset`.

ti-primeform

inputs

`pcset`	any member of the ti-setclass whose primeform is sought (or a list of these)	list (or list of lists) of mod-`n` integers

optional inputs

`n`	modulus of the pc space	positive integer (12 by default)

outputs

	primeform of the ti-setclass to which `pcset` belongs (or a list of these)	list (or list of lists) of mod-`n` integers

A ti-setclass is a family of pcsets related to one another by transposition and/or inversion, and its ti-primeform is a member of the family, designated to represent the entire family. The ti-primeform algorithm selects the pcset whose elements are maximally close to zero in a particular sense. This algorithm is equivalent (when `n` = 12) to one introduced by John Rahn (1980) and adopted by Joseph Straus and Robert Morris among others; Allen Forte's original prime form algorithm (1973) produces different results in a small number of cases. (Essentially, the Rahn prime form minimizes the number of large pc integers, while the Forte prime form minimizes the largest one and then maximizes the number of small ones.)

Will tolerate integers out of the mod-`n` range in `pcset`.

`expand-t-setclass`

inputs

`pcset`	any member of the desired t-setclass (or list containing a member of each desired t-setclass)	list of mod-`n` integers (or list of such lists)

optional inputs

`n`	modulus of the pc space	positive integer (12 by default)

outputs

	family of pcsets containing pcset and all of its transpositions (or list of these families)	list of lists of mod-`n` integers (or list of these nested lists)

Given any member of a t-setclass, lists every member of that t-setclass. Will tolerate integers out of the mod-`n` range in `pcset`.

`expand-ti-setclass`

inputs

`pcset`	any member of the desired ti-setclass (or list containing a member of each desired ti-setclass)	list of mod-`n` integers (or list of such lists)

optional inputs

`n`	modulus of the pc space	positive integer (12 by default)

outputs

	family of pcsets containing pcset and all of its transpositions and inversions (or list of these families)	list of lists of mod-`n` integers (or list of these nested lists)

Given any member of a ti-setclass, lists every member of that ti-setclass. Will tolerate integers out of the mod-`n` range in `pcset`.

`xpose`

inputs

`p-or-pc`	pitch or pc (or flat or nested list of either)	integer or mod-`n` integer (or flat or nested list of either)
`interval`	directed interval of transposition	integer or mod-`n` integer

optional inputs

`n`	modulus of the pc space (if any)	nonnegative integer (12 by default)

outputs

	transposition of `p-or-pc` by designated interval	integer or mod-`n` integer (or flat or nested list of either)

Performs pc-space transposition (`n` > 0) or p-space transposition (`n` = `nil`). Will tolerate integers out of the mod-`n` range in `p-or-pc` and `interval` when performing pc-space transposition.

`nvert`

inputs

`p-or-pc`	pitch or pc (or flat or nested list of either)	integer or mod-`n` integer (or flat or nested list of either)
`index`	index of inversion	integer or mod-`n` integer

optional inputs

`n`	modulus of the pc space (if any)	nonnegative integer (12 by default)

outputs

	inversion of `p-or-pc` at designated index	integer or mod-`n` integer (or flat or nested list of either)

Performs pc-space inversion (`n` > 0) or p-space inversion (`n` = `nil`). The inversion of an element x at index i is $i - x$; thus `index` represents the constant sum of each element and its inversion. Will tolerate integers out of the mod-`n` range in `p-or-pc` and `index` when performing pc-space inversion.

`set-complement`

inputs

`pitch-or-pc-set`	list (or list of lists) of pitches or pitch classes	list (or list of lists) of integers

optional inputs

`space`	pitch space or pc space in which complementation is performed	list of integers

outputs

	list of all elements in `space` that are not members of `pitch-or-pc-set`	list (or list of lists) of integers

Computes the complement of `pitch-or-pc-set` with respect to `space`. In other words, returns `space` with `pitch-or-pc-set` removed.

`make-cyc-pfield`

inputs

`generator`	cycle of intervals from which `pfield` is generated	list of integers
`origin`	pitch in `pfield` coinciding with the start of `generator`	integer
`lo`	pitch below which `pfield` is truncated	integer
`hi`	pitch above which `pfield` is truncated	integer

outputs

`pfield`	space of pitches that unfolds `generator`	list of integers

Generates a space of pitches that repeatedly unfolds a cyclic interval pattern. For instance, the pitch field `(… -4 0 1 5 6 10 11 15 …)` unfolds the cycle `(1 4)`. Assuming the cycle cannot be partitioned exhaustively into copies of a smaller pattern (which rules out cycles like `(3 2 3 2)`), there will be *M* distinct transpositions of the field, where *M* is the sum of the generating intervals. These pitch fields, which extend infinitely low and high in theory, retain most of their interesting properties when truncated for use in composition. For more on the underlying theory, see the author's article, "Field Notes: A Study of Fixed-Pitch Formations," *Perspectives of New Music* 41.1 (Winter 2003): 180–239.

The resulting `pfield` is transposed so that a cycle of `generator` begins at `origin`, and it is truncated between `lo` and `hi`.

merge-pfields

inputs

any number of `pfield` *items:*

`pfield`	space of pitches	list of integers

outputs

	space of pitches	list of integers

Merges the contents of any number of pitch fields, with duplicate pitches removed.

find-pc-in-field

inputs

`pc`	pitch class or list of them	mod-`n` integer or list of them
`field`	space of pitches	list of integers

optional inputs

`n`	modulus of the pc space and number of equal steps per octave in the corresponding pitch space	positive integer (12 by default)

outputs

	list of all the pitches in `field` that are congruent mod-`n` to `pc` (or list containing one such list for each input pc)	list (or list of lists) of integers

Searches `field` (which can be output from `make-cyc-pfield`, or any list of pitches) and returns all the instances it finds of the pitch class `pc` (or of each pitch class, if `pc` is a list of them).

find-pcset-in-field

inputs

`pcset`	pitch class set or list of them	list (or list of lists) of mod-`n` integer
`field`	space of pitches	list of integers

optional inputs

`n`	modulus of the pc space and number of equal steps per octave in the corresponding pitch space	positive integer (12 by default)

outputs

	list of all the pitch sets in `field` that are pitch-space realizations of `pcset` (or list containing one such list for each input pcset)	list (or list of lists) of lists of integers

Searches `field` (which can be output from `make-cyc-pfield`, or any list of pitches) and returns all the instances it finds of the pitch class set `pcset` (or of each pitch class set, if `pcset` is a list of them).

find-bounded-chords-in-field

inputs

`bounds`	pairs (*lo hi*) representing registral bounds	list of pairs of integers
`field`	space of pitches	list of integers

outputs

	list of all the pitch sets in `field` composed of pitches within specified `bounds`	list of lists of integers

Each of the `bounds` defines a region of `field` whose pitches are at least *lo* and at most *hi*. Returns a list of all the pitch sets that can be formed by drawing one distinct pitch from each of these regions.

Can also process a list of fields: if `field` is a list of lists of integers, then returns a corresponding list of pitchsets for each sublist.

`filter-chordlist`

inputs

`test`	predicate function that tests a chord (list of integers) and returns `t` or `nil`	function or subpatch in lambda mode, or output from one of the modules named `make-`*foo*`-test`; to combine multiple tests, use the `and-tests` and `or-tests` functions
`chordlist`	list (possibly nested) of chords	list (possibly nested) of integers

optional inputs

`mode`	operating mode of filter	*menu selection:* `PASS` or `REJECT`

outputs

`filtered-chordlist`	list (possibly nested) of chords	list (possibly nested) of integers

The `chordlist` parameter is a (nested) list of lists of integers (for pitch sets) or mod-*n* integers (for pcsets). The `test` parameter is any predicate function returning `t` or `nil` when applied to a pitch set or pcset. Filtering works on elements of either type depending on the `test` parameter — although most of the `make-`*foo*`-test` modules yield tests for pitch sets only.

The return value `filtered-chordlist` is a list like `chordlist` but with certain chords removed. If `mode` is set to `PASS`, which is the default, then the chords for which `test` returns `t` will be passed through to `filtered-chordlist` and the others removed. If `mode` is set to `REJECT`, then the chords for which `test` returns `t` will be removed and the others passed through to `filtered-chordlist`.

This function performs as intended only when the input `chordlist` is a list with a particular structure S, which if not empty can contain chords (nonempty integer lists) or lists with structure S, but not a mix of the two.

Examples (each item `chord-`*n* is a nonempty list of integers)

- good: `((chord-1 chord-2) (chord-3) () (chord-4 chord-5))`

- good: `(chord-1 chord-2 chord-3)`

- good: `((chord-1 chord-2) (() (chord-3 chord-4) (chord-5)))`

- bad: `(chord-1 chord-2 nil chord-3)`
  list contains a mix of chords and S-structures

- bad: `((chord-1 (chord-2 chord-3)) (chord-4 chord-5)))`
  first sublist contains a mix of chords and S-structures

For other filtering tasks, consider the Common Lisp functions remove-if and remove-if-not, and OpenMusic functions like `filter-list`.

make-bounds-test

inputs

`lo`	lowest permissible pitch	integer
`hi`	highest permissible pitch	integer

outputs

	test that will return `t` or `nil`	compiled lexical closure

Returns a predicate, intended for use with `filter-chordlist`, to test if a pitch set fits entirely within the closed interval [`lo`, `hi`].

make-width-test

inputs

`width`	maximum permissible interval between lowest and highest pitches	integer

outputs

	test that will return `t` or `nil`	compiled lexical closure

Returns a predicate, intended for use with `filter-chordlist`, to test if the registral span of a pitch set (the interval between its lowest and highest pitches) is less than or equal to `width`.

make-cardinality-test

inputs

`lo`	lower bound on cardinality	positive integer
`hi`	upper bound on cardinality	positive integer

optional inputs

`n`	modulus of pc space	integer or nil (defaults to nil)

outputs

	test that will return `t` or `nil`	compiled lexical closure

Returns a predicate, intended for use with `filter-chordlist`, to test if the number of notes in a pcset or pitch set is at least `lo` and at most `hi`. If a non-nil modulus `n` is specified, reduces each chord mod `n` before computing its cardinality.

make-spacing-test

inputs

`spacing-lists`	list of specifications for pitch-set spacing	each element is a list of *spacing-pairs*; each *spacing-pair* is an integer pair (*lo hi*), *lo* ≤ *hi*

outputs

	test that will return `t` or `nil`	compiled lexical closure

Returns a predicate, intended for use with `filter-chordlist`, to test if the intervals between consecutive elements of a pitch set, traversed from bottom to top, are in the ranges determined by `spacing-lists`. Each item in `spacing-lists` is a list of *spacing-pairs*, which are pairs (*lo hi*) specifying the minimum and maximum permissible distances between consecutive pitches.

For a given chord *C*, testing proceeds as follows:

- The first item in `spacing-lists` with an appropriate number of *spacing-pairs* is located.
- The intervals of *C* are compared to the ranges of these *spacing-pairs*. If every interval is in range, then `make-spacing-test` returns `t`. Otherwise, the next item in `spacing-lists` with an appropriate number of *spacing-pairs* is located and the comparison step is repeated.
- If the intervals of *C* are not in the ranges determined by at least one list of *spacing-pairs*, then `make-spacing-test` returns `nil`.

`make-voicing-test`

inputs

`voicing-pairs`	list of specifications for pitch-set voicing	list of pairs; each pair is a list of the form `(`*ints* *lim*`)`; each *ints* is a list of integers; each *lim* is a number between –1 and 1

optional inputs

`n`	modulus of the pc space	positive integer (12 by default)

outputs

	test that will return `t` or `nil`	compiled lexical closure

Returns a predicate, intended for use with `filter-chordlist`, to test if specific interval classes (undirected mod-`n` pc intervals) in a chord are voiced according to the criteria specified in vspec-pairs, which is a list of pairs `(`*ints* *lim*`)`. Each *ints* is a list of undirected pitch intervals drawn from a single mod-n interval class — e.g. `(11 13)` with `n` at its default value of 12 —, and the absolute value of the corresponding *lim*, a positive (negative) number between 0 and 1 (–1), determines a lower (upper) bound on the ratio *J*:*K*, where *K* is the multiplicity of occurrence of the interval class represented in *ints*, and *J* is the combined multiplicity of occurrence of the undirected pitch intervals listed in *ints*.

Examples (with `n` = 12)

- With `voicing-pairs` = `(((10 22) 3/4))`, returns a test to see if at least 3/4 of the instances of interval class 2 in a chord are voiced as pitch intervals 10 or 22.
- With `voicing-pairs` = `(((1) 1/6) ((1) -1/2))`, returns a test to see if at least 1/6, but at most 1/2, of the instances of interval class 1 in a chord are voiced as pitch interval 1.

and-tests

inputs

any number of test *items:*

test	test that will return t or nil	compiled lexical closure

outputs

	test that will return t or nil	compiled lexical closure

Takes any number of predicate functions (each returning t or nil) and returns a test that, for a certain input, will return t if all the predicates return t for the same input, or nil if any of the predicates return nil.

or-tests

inputs

any number of `test` *items:*

`test`	test that will return `t` or `nil`	compiled lexical closure

outputs

	test that will return `t` or `nil`	compiled lexical closure

Takes any number of predicate functions (each returning `t` or `nil`) and returns a test that, for a certain input, will return `t` if any of the predicates return `t` for the same input, or `nil` if all the predicates return `nil`.

vector-dotprod

inputs

`v`	vector	list of numbers (NB: not a lisp vector)
`w`	vector of same order as `v`	list of numbers, equal in length to the `v` list

outputs

	dot product $v \cdot w$	number

The dot-product of the vectors $v = (v_1\ v_2\ \ldots\ v_n)$ and $w = (w_1\ w_2\ \ldots\ w_n)$ is the number $v_1w_1 + v_2w_2 + \ldots + v_nw_n$. Sometimes vector `w` is called a *weighting vector*; then $v \cdot w$ is called a *weighted sum* of the contents of `v`.

`vector-angle`

inputs

`v`	vector	list of numbers (NB: not a lisp vector)
`w`	vector of same order as `v`	list of numbers, equal in length to the `v` list

outputs

	angle from `v` to `w`	real number in [0, $\pi/2$]

Calculates the geometric angle (in radians) from vector `v` to vector `w` positioned at a common origin.

`incl-classrep` expression

undirected pc interval, possibly 0; or aggregation of them	integer in [0, n/2]; or list of them
undirected pitch interval, possibly 0, or range of them; or aggregation of intervals and/or ranges	`(:p` a_0 a_1 `…)`, a_i a nonnegative integer or pair `(`lo_i hi_i`)`, $lo_i < hi_i$, representing the range [lo_i, hi_i]; a single interval or range takes the same form, and is therefore a list with head `:p` and a one-element tail
t pcset class	`(:t` k_0 k_1 `…)`, k_i a mod-n integer
ti pcset class	`(:ti` k_0 k_1 `…)`, k_i a mod-n integer
t pitch set class	`(:tp` k_0 k_1 `…)`, k_i an integer
ti pitch set class	`(:tip` k_0 k_1 `…)`, k_i an integer

Examples

`4`	ic 4
`(1 2 6)`	ics 1, 2, 6
`(:p 3)`	undirected pitch interval 3
`(:p 5 (7 11) (13 17))`	undirected pitch intervals 5, 7, 8, 9, 10, 11, 13, 14, 15, 16, 17
`(:t 0 1 3)`	pcset {0 1 3} and its transpositions
`(:ti 0 1 5)`	pcset {0 1 5} and its transpositions and inversions
`(:tp 0 7 14)`	pitch set {0 7 14} and its transpositions
`(:tip 0 9 16)`	pitch set {0 9 16} and its transpositions and inversions

incl-vec

inputs

`chord`	chord whose inclusion vector is sought (or a list of chords)	list of integers (or list of lists of integers)
`classreps`	list of expressions representing equivalence classes of pcs or pitches	list of `incl-classrep` expressions

optional inputs

`n`	modulus of the pc space	positive integer (12 by default)

outputs

	inclusion vector reporting selected inclusion features of `chord`	list of nonnegative integers

The inclusion vector is a broad generalization of Alan Forte's interval vector (1973). For a given list of equivalence classes, the inclusion vector of a chord is a corresponding list indicating how many members of each equivalence class contain or are contained by `chord`. The equivalence classes are determined by the `classreps` parameter, an expression with special syntax described elsewhere. The equivalence classes can be particular undirected pc intervals, undirected pitch intervals, t or ti set classes of pcs, and/or t or ti set classes of pitches; multiple undirected pc or pitch intervals can also be aggregated and counted together, as explained in the discussion of `incl-classrep` syntax.

Depending on how `classreps` is configured, `incl-vec` calculations may or may not make sense when `chord` is interpreted as a mod-`n` pcset. No matter how `classreps` is configured, `incl-vec` calculations always make sense when `chord` is interpreted as a pitch set.

If n = 12 and `classreps` is a list representing all of the nonzero interval classes — namely `(1 2 3 4 5 6)` — then the inclusion vector is the interval vector whose uses have been discussed at length in the music-theory literature.

incl-vec-angle

inputs

`chord1`	chord (pcset or pitch set)	list of integers
`chord2`	chord (pcset or pitch set)	list of integers
`classreps`	list of expressions representing equivalence classes of pcs or pitches	list of `incl-classrep` expressions

optional inputs

`n`	modulus of the pc space	positive integer (12 by default)

outputs

	angle from the inclusion vector of `chord1` to that of `chord2`	real number in [0, π/2]

This function calculates inclusion vectors for `chord1` and `chord2` in terms of the *m* equivalence classes identified in `classreps`. It then situates these vectors at a common origin in *m*-dimensional space and computes the angle (in radians) from the `chord1` vector to the `chord2` vector.

When `n` = 12 and `classreps` is `(1 2 3 4 5 6)`, the inclusion vectors are interval vectors, and the return value is the interval angle proposed as a measure of pcset similarity (with smaller angles indicating greater similarity) in Damon Scott and Eric J. Isaacson, “The Interval Angle: A Similarity Measure for Pitch-Class Sets,” *Perspectives of New Music* 36.2 (Summer 1998): 107–142.

If `chord1` and/or `chord2` contains (or is contained by) *zero* members of *all* the equivalence classes specified in `classreps`, then it will have a zero-magnitude vector and undefined direction. In this case, a true angle measurement is impossible. To preserve the utility of this function as a generalized (dis)similarity measure, the following solution is adopted in zero-magnitude cases: if both vectors have zero magnitude, the angle reported is zero (for maximum similarity); if one vector has zero magnitude and the other has nonzero magnitude, the angle reported is π/2 (for maximum dissimilarity). To avoid zero-magnitude vectors and ensure a result based on true angle measures, include in `classreps` all possible interval classes or one of the complete prime-form lists produced by `list-t-primeforms` or `list-ti-primeforms` (using the `tag` option).

`prog-classrep` expression

directed pc interval, possibly 0; or aggregation of them	mod-n integer; or list of them
directed pitch interval or range of them; or aggregation of intervals and/or ranges	`(:p` a_0 a_1 `…)`, a_i an integer or pair of them (*cf* notation for undirected pitch intervals in `incl-classrep` syntax)

Examples

`11`	directed pc interval 11
`(8 9)`	directed pc intervals 8 and 9
`(:p 18)`	directed pitch interval 18
`(:p (-2 2) 6)`	directed pitch intervals –2, 1, 0, 1, 2, 6

prog-vec

inputs

`from-chord`	chord-of-departure for the progression whose vector is sought	list of integers
`to-chord`	chord-of-arrival for the progression whose vector is sought	list of integers
`classreps`	list of expressions representing directed pc or pitch intervals	list of `prog-classrep` expressions

optional inputs

`n`	modulus of the pc space	positive integer (12 by default)

outputs

	progression vector for the pair `(from-chord to-chord)`	list of nonnegative integers

The progression vector is an application, and in some respects a generalization, of David Lewin's interval function (*Generalized Music Interval and Transformations*, 1987). For a given list of directed intervals, as specified in `classreps`, the progression vector of a chord-pair `(from-chord to-chord)` is a corresponding list indicating how many instances of each interval can be formed from a member of `from-chord` to a member of `to-chord`.

Depending on how `classreps` is configured, `prog-vec` calculations may or may not make sense when `from-chord` and `to-chord` are interpreted as mod-`n` pcsets. No matter how `classreps` is configured, `prog-vec` calculations always make sense when these chords are intepreted as pitch sets.

`prog-vec-angle`

inputs

`from1`	chord (pcset or pitch set)	list of integers
`to1`	chord (pcset or pitch set)	list of integers
`from2`	chord (pcset or pitch set)	list of integers
`to2`	chord (pcset or pitch set)	list of integers
`classreps`	list of expressions representing directed pc or pitch intervals	list of `prog-classrep` expressions

optional inputs

`n`	modulus of the pc space	integer (12 by default)

outputs

	angle from the progression vector of `(from1 to1)` to that of `(from2 to2)`	real number in $[0, \pi]$

This function calculates progression vectors for the pairs `(from1 to1)` and `(from2 to2)` in terms of the m intervals identified in `classreps`. It then situates these vectors at a common origin in m-dimensional space and computes the angle (in radians) from the `(from1 to1)` vector to the `(from2 to2)` vector.

When either chord pair involves *zero* instances of *all* the intervals specified in `classreps`, then it will have a zero-magnitude vector and undefined direction. In this case, a true angle measurement is impossible. To preserve the utility of this function as a generalized (dis)similarity measure, the following solution is adopted in zero-magnitude cases: if both vectors have zero magnitude, the angle reported is zero (for maximum similarity); if one vector has zero magnitude and the other has nonzero magnitude, the angle reported is $\pi/2$ (for maximum dissimilarity). To avoid zero-magnitude vectors and ensure a result based on true angle measures, include in `classreps` all possible directed pc intervals 0, 1, …, `n`.

`sort+`

inputs

`elements`	items to sort	list

optional inputs

`test`	how to compare items for sorting	binary function name or function object (`#'<` by default)
`key`	operation to perform on items before comparison	function name or object (or `nil` by default)

outputs

`sorted-elements`	result of sorting	list
`equalities`	indicates runs of equal value (or key-value) in `sorted-elements`	list of positive integers

Resembles the `sort.` function in the OpenMusic kernel, with the addition of a second output, `equalities`, which reports how many elements score identically when they (or their `key` values) are subjected to `test`.

Example

Suppose `test` = `#'<` and `key` = `#'length`, with `elements` and return values as shown:

`elements:`	`((j k l m) (a b) (a b c) (c d) (e f) (d e f) (g h i))`
`sorted-elements:`	`((a b) (c d) (e f) (a b c) (d e f) (g h i) (j k l m))`
`equalities:`	`(3 3 1)`

Here the items to be sorted are lists such as `(a b)`, they are sorted based on their lengths, and the sort order is from shortest to longest. The `equalities` list indicates that three elements are tied for shortest, three more elements are tied for next shortest, and one element is longest. The actual result in `sorted-elements` may differ from what is shown in this example, because nothing is guaranteed about the order, relative to one another, of items with equal values (or equal `key` values). For instance, `sorted-elements` could also begin with `(c d)` or `(e f)` in this example.

sort+select

inputs

`elements`	items from which to select	list
`n`	how items to select	integer

optional inputs

`test`	how to compare items for sorting	binary function name or function object (`#'<` by default)
`key`	operation to perform on items before comparison	function name or object (or `nil` by default)

outputs

	`n` elements selected from top of sorted list	list

Sorts `elements` as they would be sorted by the `sort.` function in the OpenMusic kernel. Then selects the `n` items from the top of the sorted list. If certain elements (or their `key` values) are equal according to `test`, and `n` is such that some but not all of these elements should be selected, then this part of the selection is made randomly.

Example

Suppose `test` = `#'<`, `key` = `#'length`, and `n` = `4`, with `elements` as shown:

```
((j k l m) (a b) (a b c) (c d) (e f) (d e f) (g h i))
```

Here the three shortest sublists — `(a b)`, `(c d)`, `(e f)` — will be selected; and the fourth and final part of the selection will be selected at random from `(a b c)`, `(d e f)`, `(g h i)`.

sort-key_incl-vec-sum

inputs

classreps	list of expressions representing equivalence classes of pcs or pitches	list of incl-classrep expressions
weightlist	weighting applied to inclusion vector	list of numbers, one for each classreps item

optional inputs

n	modulus of the pc space	integer (12 by default)

outputs

	test that will return a number	compiled lexical closure

Returns a function that assigns a number to a chord according to a weighted sum of the positions in the chord's inclusion vector, calculated for the equivalence classes represented in classreps. The function returned by sort-key_incl-vec-sum is intended for use as a key function with sort+.

In one straightforward application, n = 12 and classreps is the list (1 2 3 4 5 6), so the inclusion vector is the familiar interval vector. With weightings that reflect the potential dissonance of each interval class, this application allows a list of chords to be sorted roughly in order of increasing or decreasing dissonance.

sort-key_incl-vec-angle

inputs

`classreps`	list of expressions representing equivalence classes of pcs or pitches	list of `incl-classrep` expressions
`refchord`	chord whose inclusion vector provides a reference from which angles are measured	list of integers

optional inputs

`n`	modulus of the pc space	integer (12 by default)

outputs

	test that will return a number	compiled lexical closure

Returns a function that assigns a number to a chord *C* based on the angle measured from the inclusion vector of `refchord` to the inclusion vector of *C*. The function returned by `sort-key_incl-vec-angle` is intended for use as a key function with `sort+`.

Because the angle measure between the inclusion vectors of two chords is often plausibly interpreted as a measure of their similarity (with smaller angles indicating greater similarity), this function allows a list of chords to be sorted in order of increasing or decreasing similarity to `refchord`.

sort-key_prog-vec-sum

inputs

`classreps`	list of expressions representing directed pc or pitch intervals	list of `prog-classrep` expressions
`from-chord`	chord (pcset or pitch set)	list of integers
`weightlist`	weighting applied to the progression vector	list of numbers, one for each `classreps` item

optional inputs

`n`	modulus of the pc space	integer (12 by default)

outputs

	test that will return a number	compiled lexical closure

Returns a function that assigns a number to a chord *C* according to a weighted sum of the positions in the progression vector of the pair `(from-chord` *C*`)`, calculated for the intervals represented in `classreps`. The function returned by `sort-key_prog-vec-sum` is intended for use as a key function with `sort+`.

sort-key_prog-vec-angle

inputs

`classreps`	list of expressions representing directed pc or pitch intervals	list of `prog-classrep` expressions
`from-chord`	chord (pcset or pitch set)	list of integers
`ref-from`	chord-of-departure for the pair whose progression vector provides reference from which angles are measured	list of integers
`ref-to`	chord-of-arrival for the pair whose progression vector provides reference from which angles are measured	list of integers

optional inputs

`n`	modulus of the pc space	integer (12 by default)

outputs

	test that will return a number	compiled lexical closure

Returns a function that assigns a number to a chord *C* based on the angle measured from the progression vector of the pair `(ref-from ref-to)` to the inclusion vector of the pair `(from-chord` *C*`)`. The function returned by `sort-key_prog-vec-angle` is intended for use as a key function with `sort+`.

Because the angle measure between the progression vectors of two chord pairs is often plausibly interpreted as a measure of their similarity (with smaller angles indicating greater similarity), this function allows a list of chords to be sorted in order of increasing or decreasing similarity of the pairs they complete to the reference pair `(ref-from ref-to)`.

sort-key_width

outputs

	test that will return a number	compiled lexical closure

Returns a function assigns a number to a chord representing the registral width of that chord (the distance between its lowest and highest pitches).

mc->p

inputs

mc	midicents value or list of them	integer or list of integers

optional inputs

n	number of equal steps per octave	integer (12 by default)

outputs

p	pitch-space value or list of them	integer or list of integers

Converts from midicent values to pitch-space values.

pitch space: middle-C = 0, minimal step (1/n octaves) = 1

midicents: middle-C = 6000, semitone = 100 (cent = 1)

p->mc

inputs

p	pitch-space value or list of them	integer or list of integers

optional inputs

n	number of equal steps per octave	integer (12 by default)

outputs

mc	midicents value or list of them	integer or list of integers

Converts from pitch-space values to midicent values.

pitch space: middle-C = 0, minimal step (1/n octaves) = 1

midicents: middle-C = 6000, semitone = 100 (cent = 1)

p->pc

inputs

p	pitch-space value or list of them	integer or list of integers

optional inputs

n	number of equal steps per octave	integer (12 by default)

outputs

pc	pc-space value or list of them	mod-n integer or list of them

Converts from pitch to pitch class.

parse-incl-classreps

inputs

incl-classreps	list of expressions representing equivalence classes of pcs or pitches	list of incl-classrep expressions

outputs

incl-classreps		the input is passed through unchanged

Prints (to the Listener window) a description of each item in the list incl-classreps, to assist in the construction of parameters that use the incl-classrep format.

parse-prog-classreps

inputs

prog-classreps	list of expressions representing directed pc or pitch intervals	list of prog-classrep expressions

outputs

prog-classreps		the input is passed through unchanged

Prints (to the Listener window) a description of each item in the list prog-classreps, to assist in the construction of parameters that use the prog-classrep format.

flatten2chordlist

inputs

`chordtree`	list (possibly nested) of chords	list (possibly nested) of lists of integers

outputs

`chordlist`	flat list of chords	list in which each element is a flat list of integers

Given a (possibly nested) list of chords, removes the nested structure and returns the same chords in a flat list. A nested list of chord must be processed with this function before it can be sorted.