Ircam documentation

- *Research reports*
- *Musical works*
- *Software*

OpenMusic

RC library

version 1.1

Second edition, March 2000

ircam Centre Pompidou

This documentation was written by Örjan Sandred, and was produced under the editorial responsibility of Marc Battier, Marketing Office, Ircam.

OpenMusic was conceived and programmed by
Gérard Assayag and Carlos Agon.

The RC library was conceived and programmed by Örjan Sandred.

First edition of the manual, March 2000.

This documentation corresponds to version 1.1 of the RC library, and to version 2.0 or higher of OpenMusic.

Ircam
1, place Igor-Stravinsky
F-75004 Paris
Tel. 01 44 78 49 62
Fax 01 44 78 15 40
E-mail ircam-doc@ircam.fr

Groupe d'utilisateurs Ircam

L'utilisation de ce programme et de sa documentation est strictement réservée aux membres des groupes d'utilisateurs de logiciels Ircam. Pour tout renseignement supplémentaire, contactez :

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

Courrier électronique: bousac@ircam.fr

Veuillez faire parvenir tout commentaire ou suggestion à :

M. Battier
Département de la Valorisation
Ircam
1, Place Stravinsky, F-75004 Paris, France

Courrier électronique : bam@ircam.fr

http://www.ircam.fr/forumnet

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

To move between pages,
use Acrobat Reader navigations buttons
or your keyboard's arrow keys

Content

Overview of OMRC 1.1 - A library for controlling rhythm by constraints

Introduction

Constraint satisfaction problems have been studied in artificial intelligence for almost 30

years. It is a technique particularly suitable for combinatorial problems. Instead of trying to find a proper algorithm for a problem, the user expresses his problem as a set of rules that has to be followed when searching for a solution to a problem. An interesting consequence of this is that it is often possible to find several correct answers to a single problem.

Traditional music (harmony) theory is often expressed as rules, and is therefore a suitable problem for a constraint-based system. Most work on music in this field has so far been focused on pitch. The difficulty in building a rule based system for rhythm is first of all a musical problem: it is hard to find rules for controlling rhythms that make musical sense.

The purpose of this library is to show a way to build rhythm with a rule-based system, and to create a platform for composers to work with constraints on rhythm.

A constraint-solving engine

There are two constraint-solving engines in the OpenMusic libraries: the Csolver (by Camilo Rueda) and the PMC (by Mikael Laurson). The RC library can communicate with both solvers, however the PMC has proven to be the fastest in connection with the RC library, and is therefore recommended.

A constraint solving engine needs two types of information/input:

• A domain. This is a set of possible answers to a problem. The output from the engine is a series of elements from the domain.

• A set of rules. These decide which elements from the domain will be accepted, and in what order they will appear in the answer.

The engine is told how many elements the answer should contain. The rules are tested on each separate element.

Characteristics of the RC library

In the RC library the elements in the domain are the smallest rhythmical building blocks in the system. An element is either a note value, a pause, a rhythm motif or a time signature.

Different strategies are possible when searching for a rhythm sequence. If the user defines the elements as rhythm motifs, he can already at this point have some control of the rhythmical language. The motifs will be the base for the rhythmical language. Their length and degree of difference are important for the result. If the user only gives the system single note values to work with, his rhythmical language will rely more on the rules.

The rules the composer defines will tell the system how the motifs/note values can be arranged and connected to each other. It is possible to work on hierarchical rhythm; i.e. some events are given higher importance than others. The hierarchy can either be a metrical hierarchy, or any other type of hierarchy the composer can imagine (for example harmonic rhythm).

There are several predefined rules in the library that the composer can use. However, an important feature is the possibility for the composer to define his own rules within the OpenMusic graphical interface.

Another type of rule is the heuristic rule. This type of rule lets the user prefer one answer out of many possible answers. The user can express how he wishes some characteristics of the answer to be. Heuristic rules are neither true nor false, but give different weights to different answers. Musically, this type of rule let the user tell the system things such as "I prefer an answer with fast note values".

Technical solutions

A major concern when making the RC library has been calculation speed. A big effort has been put into finding a good structure for the data and the rules. Several options were tried and trashed.

Already when building the domain, some basic calculations are done on the elements in the domain. The elements that are passed to the search engine are objects that carry the extra information, which fast can be accessed during the search.

The temporary solution is continuously stored in some internal vectors during the search, so that the search engine can reuse the result from calculations done on previous elements already found. This cuts down on the number of calculations, and increase speed.

It is very hard, if not impossible, to generalize a good strategy for how the engine should do the search. The user has the possibility to define his own strategy, for his particular calculation.

Experiences of the RC library

An interesting characteristic of the rule-based system in the RC library is the user's possibility to steer the system while searching for a solution. After evaluating an answer from the system he can for example tell the system "the result is OK, but I would like it more if you could change measure 23 to start with a whole note", and then ask for a new answer. With an algorithmic system this is very hard to do,

and the composer often ends up by changing the result without caring about the "error" he adds to the process.

The update to OMRC 1.1

The most important news in this update is the tutorial section of the manual.

There are also a few new predefined rules:

r-sync-over-barline, gr-layerorder, r-layerorder, r-pattern,

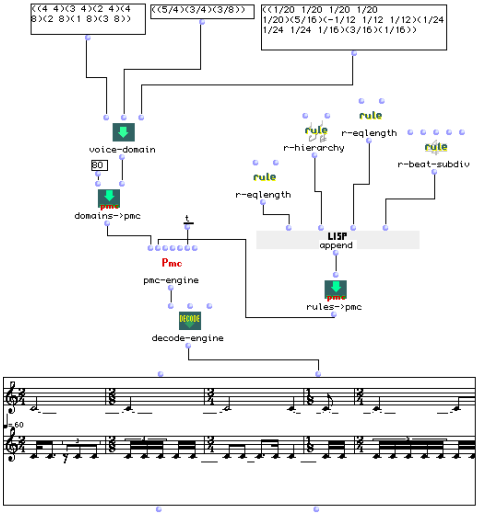
and a new user rule tool:

get-voicenumber

Limitations

There are some limitations in the library on how long the rhythm sequence can be, and how many search variables you can ask for. The limitation exist because of speed optimizations. To see what the limits are for this version (and how to change it), see the file "** README OMRC update 1.1".

Examples


((4 4)(3 4)(2 4)(4 8)(2 8)(1 8)(3 8))
((5/4)(3/4)(3/8))
((1/20 1/20 1/20 1/20 1/20)(5/16)(-1/12 1/12 1/12)(1/24 1/24 1/24 1/16)(3/16)(1/16))
voice-domain
80
domains->pmc
rule
r-hierarchy
r-eqlength
rule
r-beat-subdiv
rule
r-eqlength
LISP
append
Pmc
pmc-engine
rules->pmc
decode-engine

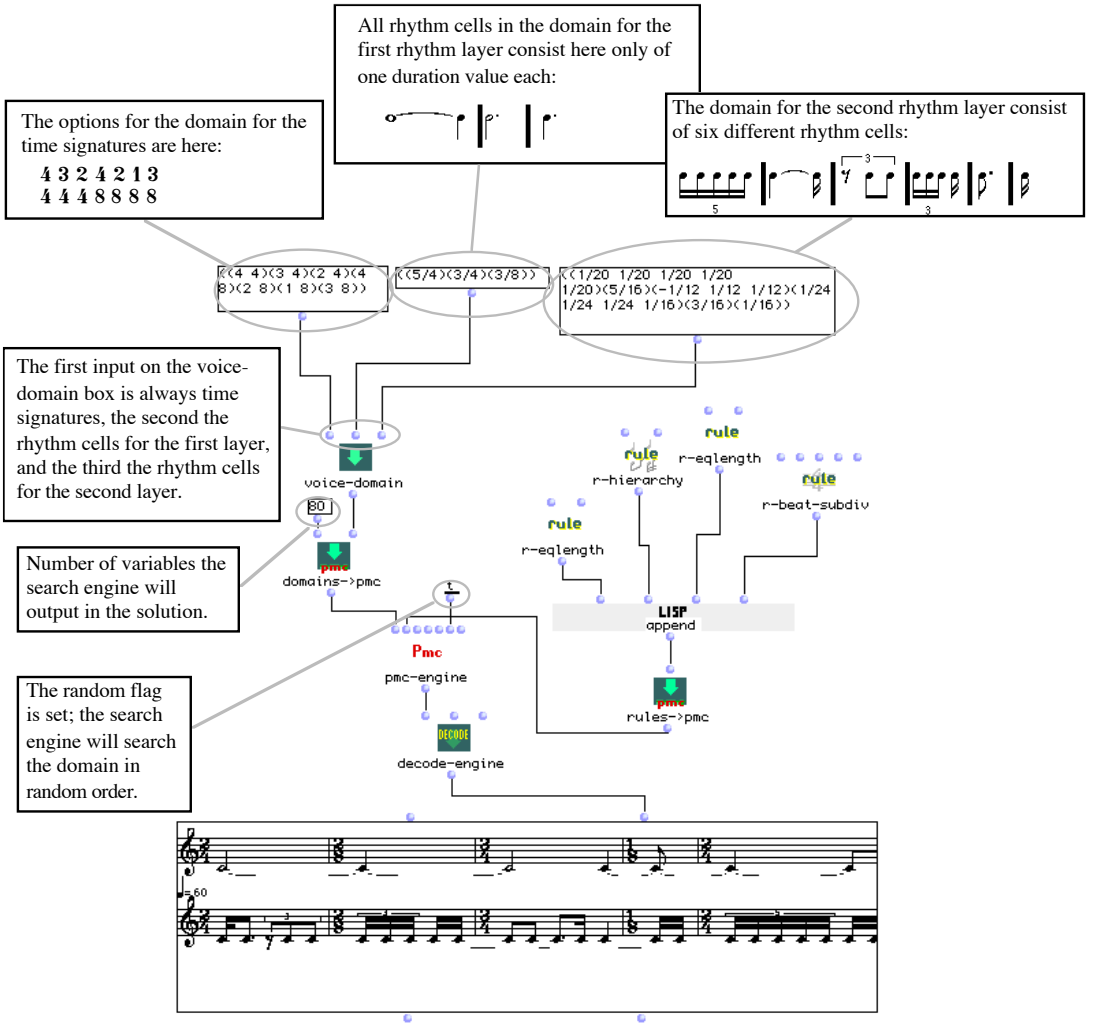


Fig.1 The domain and its boxes.

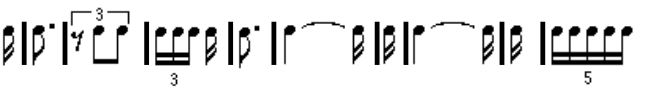


Fig.2 The second layer in the solution divided into its components (i.e. rhythm cells). Compare with the domain for the second rhythm layer above.

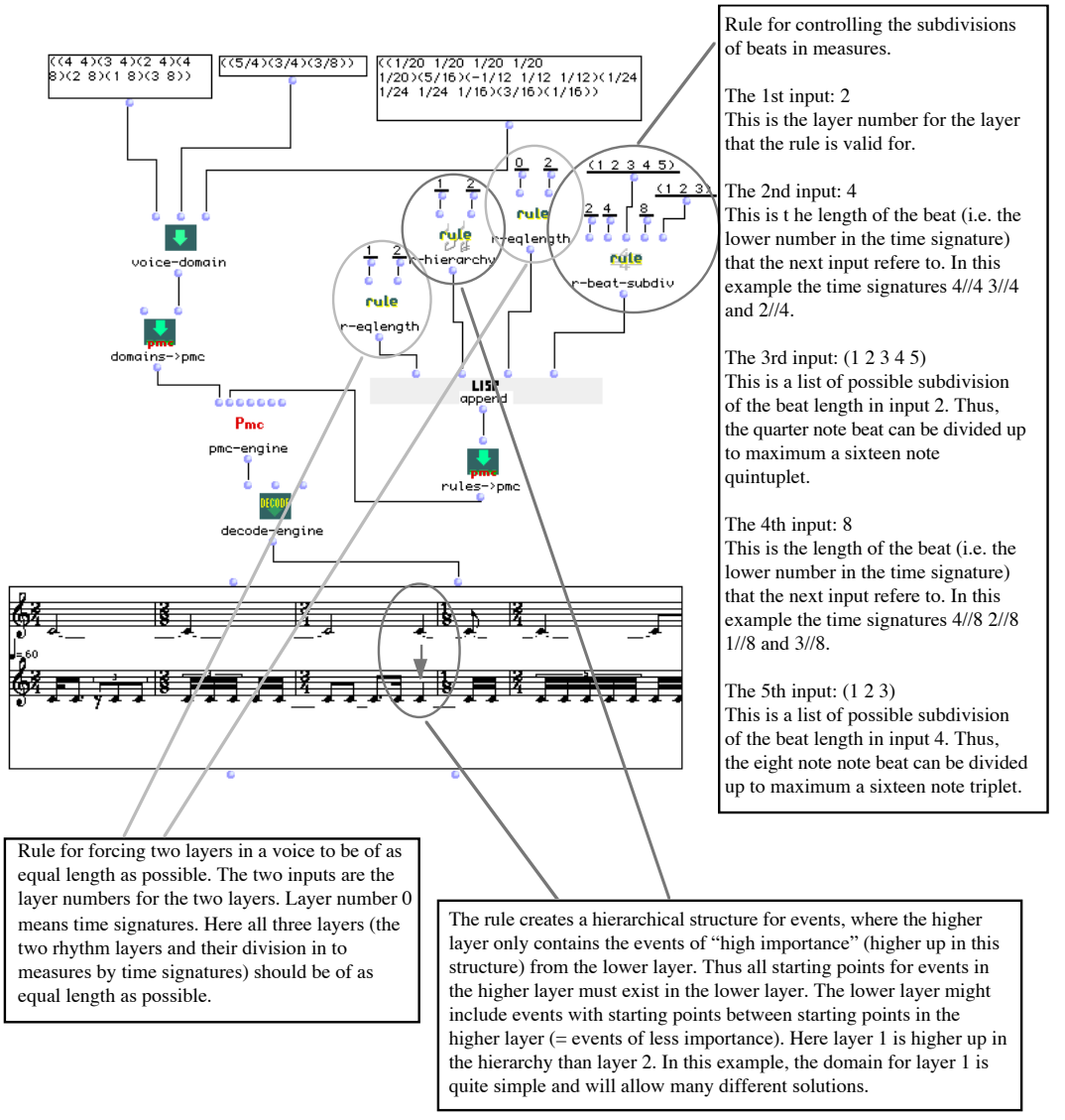
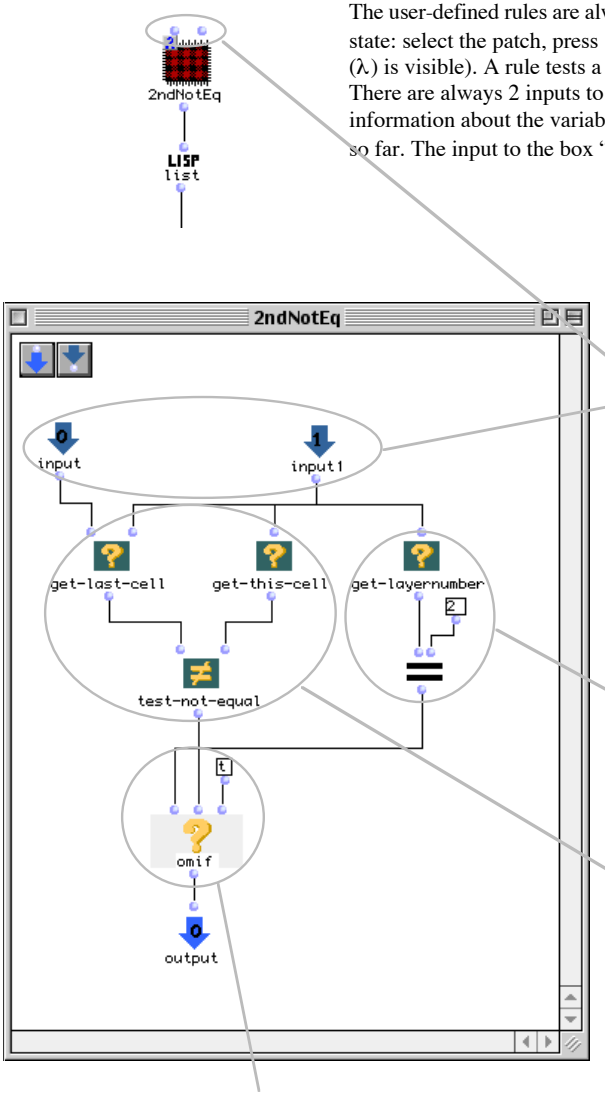


Fig.3 The general rules.



The user-defined rules are always created inside a sub patch (in the lambda state: select the patch, press 'b' and click on the button until the lambda sign (λ) is visible). A rule tests a variable and gives either true or false as a result. There are always 2 inputs to a rule. They will allow the user to access information about the variable itself and the solution the search engine found so far. The input to the box "rules->pmc" has to be a list of all rules.

The left input is the index for the current search variable (the search engine uses this index internal to keep track on what variable it is working). The right input is the current instanciated variable. Use the boxes in the "user rules tools" to access information about the search variable itself and other instanciated variables.

"get-layernumber" outputs the layer number for the current instanciated variable. The om= (from the kernel menu) will thus output true if the current variable belongs to layer number 2.

The current instanciated variable ("get-this-cell") is here compared with the last variable *in the same layer as the current* ("get-last-cell"). "test-not-equal" will output true only if they are not equal.

The rule will give true as output:
•if the current variable does not belong to layer number 2 (the rule will accept anything in other layers).
•if the current variable belongs to layer 2 and: if the one immediately before the current variable (also in layer 2) is not equal to the current variable.

Consequently: this rule will not allow immediate repetition of rhythm cells in layer 2.

Fig.4 A user defined rule.

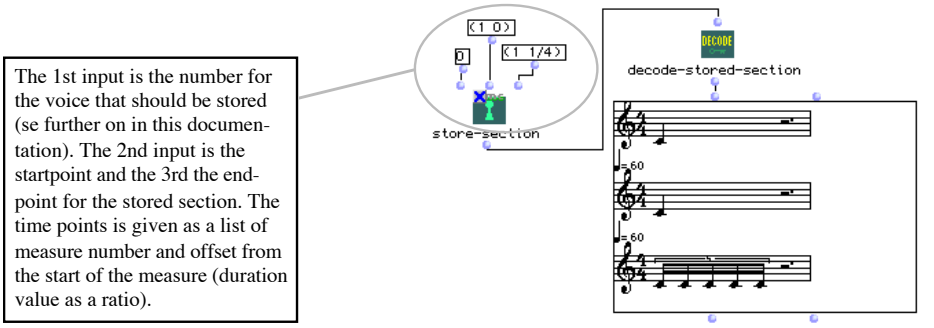


Fig.6 The stored section is from the example in fig.5. The content is printed in the notation window.

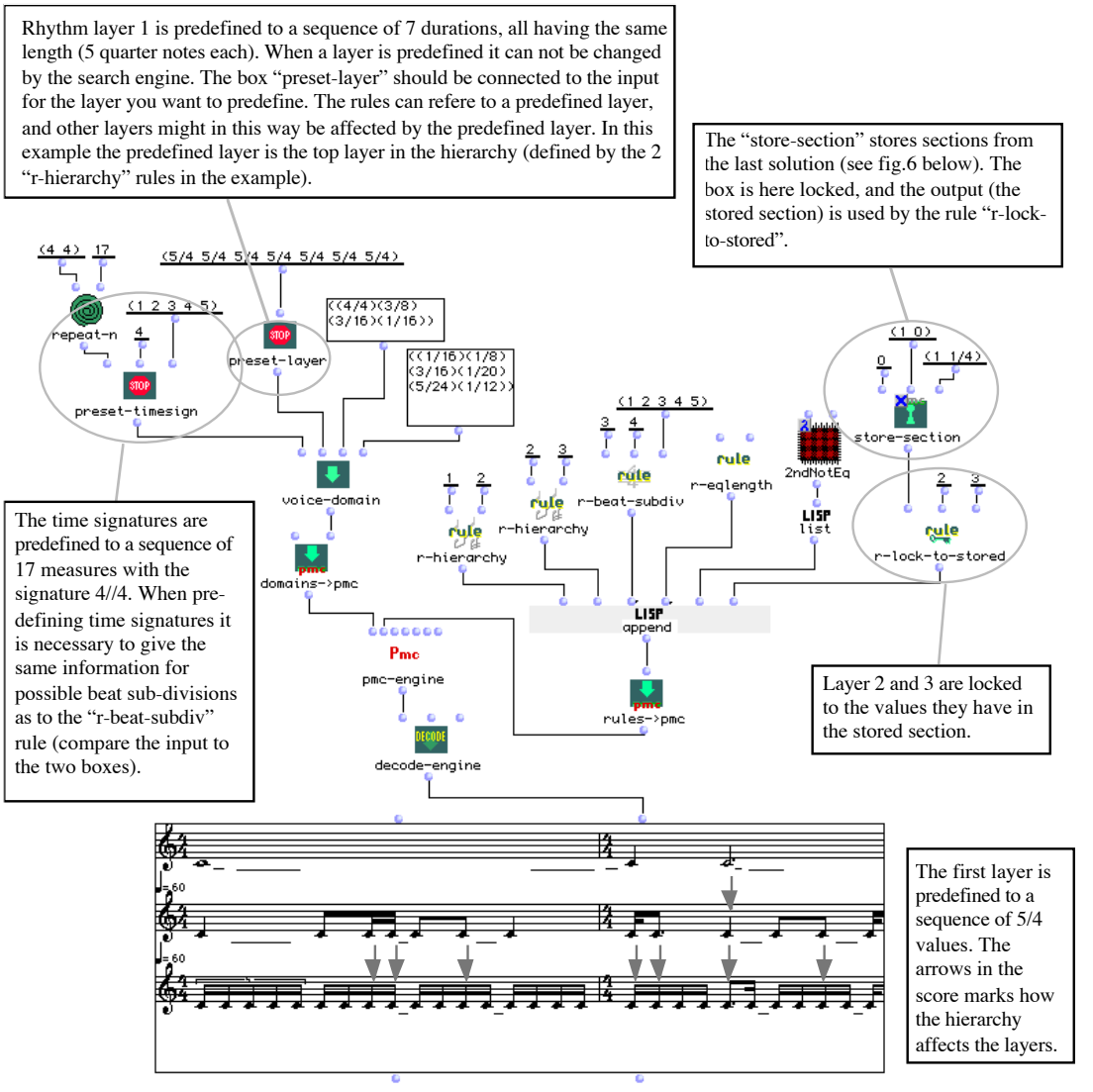
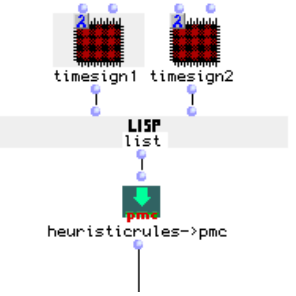


Fig.5 Predefined layers and rule to lock sections from stored partial solution.



Heuristic rules only exist as user defined. Their construction is similar to regular rules except for their output: instead of either being true or false they output weights. The search engine compare all possible answers and chose the one with the highest weight. Heuristic rules do therefore not discualify any answer, they only “prefere” certain answers.

All heuristic rules should be collected in a list and then input to the “heuristicrules->pmc” box.

Compare the example that follows with how the user defined rule was made.

The 2 inputs are identical to the inputs for the user defined rule: index and search variable (see fig.4).

Check if this variable is a time signature.

If the variable is a time signature, output the number of beats (= the first of the 2 numbers in a time signature).

If the variable is not a time signature, output 0.

Consequently: this heuristic rule will prefere time signatures with as many beats as possible.

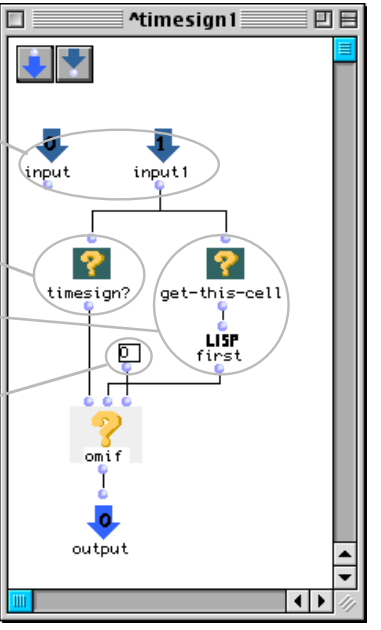


Fig.7 A heuristic rule.

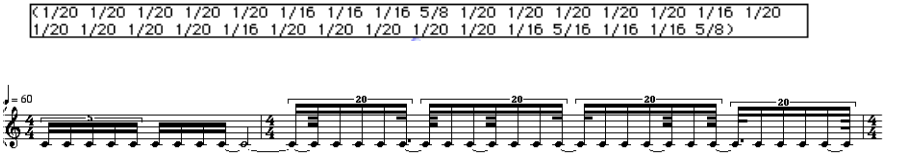


Fig.8

Given is a rhythm sequence of exact duration values. Only 1/16, 5/16, 5/8 and 1/20 (= a sixteen note quintuplet) exist. When trying to notate the rhythm using only 4//4 as time signature we discover a problem: because of a sixteen-note offset at the start of the second measure, the rest of the measure has a very complicated notation. In the below example we will let the search engine put time signature on this rhythm sequence. We will use the rules to get an easy-to-read notation.

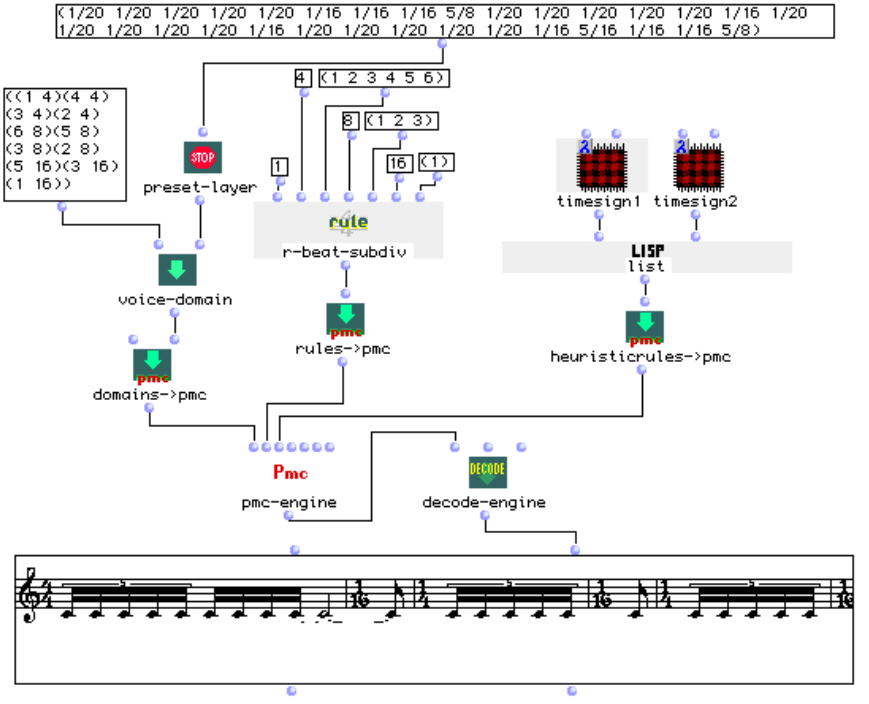
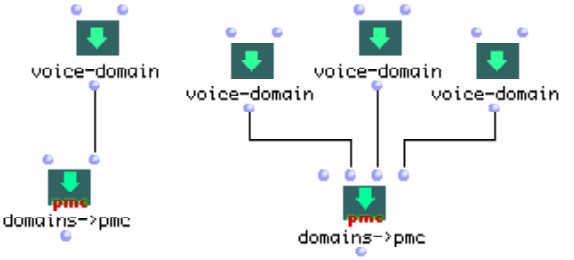


Fig.9

The domain of time signature includes three different beat lengths: quarter note, eight-note or sixteen-note. “r-beat-subdiv” restricts a quarter note beat to have up to maximum sixteen-note six-tuplets (the subdivision has to be on-beat, it can not start on a syncope). A eight-note beat can maximum have a sixteen-note triplet and a sixteen-note beat can not be subdivided. There are two heuristic rules. The first one is explained in fig.7 (it prefers time signatures with many beats). The second one prefers quarter note as beat value in front of eight-note beat value, and eight-note beats in front of sixteen-note beats. The solution is a much more understandable notation than the one in fig.8. The rhythms are identical; it is only the notation that differs.

1. BUILD DOMAINS

In order to build a domain you need two boxes to format the information for a search engine: “voice-domain” and “domains->pmc”. A “voice-domain” can not be directly connected to the search engine. It has always to be connected to the “domains->pmc” before input to the search engine.



domains->pmc

Syntax

(rc::domains->pmc *n-var voice0*)

&optional *voice1 voice2 voice3 voice4 voice5 voice6*

Input

n-var	is the number of variables that will be given in the solution.
voice0	should be connected to the “voice-domain” output for voice number 0.
voice1	(optional) should be connected to the “voice-domain” output for voice number 1.
…voice6	

Output

Should be connected to the “pmc” <s-space> input.

Description

Format all voice-domains for the “pmc” <s-space> input. A “voice-domain” is identified by which entrance it is connected to. Only rules that are connected to the corresponding input on the “rules->pmc” will affect the domain.

voice-domain

Syntax

(rc::voice-domain *timesign-list rhythmcell-list1*)
&optional *rhythmcell-list2 rhythmcell-list3 rhythmcell-list4*

Input

timesign-list	is a list of possible time signatures in the voice (or is connected to a “preset-time-sign”).
rhythmcell-list1	is a list of possible rhythm cells that can exist in layer 1 (or is connected to a “preset-layer”).
rhythmcell-list2	(optional) is a list of possible rhythm cells that can exist in layer 2 (or is connected to the “preset-layer”).
…rhythmcell-list4	

Output

Should be connected to a “domains->pmc” <voice> input.

Description

Build a domain for one voice.

If an input is set to nil, that layer is not used (for the first input this means that the solution will not be divided into bars. 4//4 will be used in notation windows). A layer can be predefined; i.e. the layer can not be changed by the search engine (but will affect other layers in the same way as for “regular” domains). Connecting the “preset-layer” or ”preset-timesign” box to the input for a layer does this.

preset-layer

Syntax

(rc::preset-layer *rhythmseq*)

Input

rhythmseq is a list of duration values (ratios). Negative values indicate pauses.

Output

Should be connected to a “voice-domain” <rhythmcell-list> input.

Description

Predefine a rhythm layer.

The layer freezes to the rhythm sequence and can not be changed by the search engine. The box replaces a rhythm domain for one layer and the output should be connected to the input on the “voice-domain” box for the layer you want to predefine.

preset-timesign

Syntax

(rc::preset-timesign *timesign-list beatvalue1 subdiv1*)

&optional *beatvalue2 subdiv2 beatvalue3 subdiv3 beatvalue4 subdiv4 beatvalue5 subdiv5*

Input

timesign-list is a list of time signatures.

beatvalue1 and subdiv1

come in a pair, and the rule "r-beat-subdiv" depends on them. subdiv1 is a list of possible subdivisions of the beat in a measure. beatvalue1 is the length of the beat (i.e. the lower number in the time signature) that subdiv1 refers to.

beatvalue2 and subdiv2 (optional)

…beatvalue5 and subdiv5

Output

Should be connected to a "voice-domain" <timesign-list> input.

Description

Predefine a time signature layer.

The time signatures in the solution freeze to the sequence timesign-list (the sequence can not be changed by the search engine). The box replaces a domain of time signatures and the output should be connected to the first input on the "voice-domain" box for the voice you want to affect.

subdiv1 and beatvalue1 have the same meaning as on the rule "r-beat-subdiv", and they should be set identical to these (or exceptionally to a higher number for the maximum subdivision). An example: If beatvalue1 = 4, then subdiv1 tells how the beats in a measure with a time signature with the beat value a quarter note can be subdivided. If subdiv1 = (1 2 3 4 5) this means that the beat can be subdivided up to a quintuplet, however only if this is on the beat (i.e. not as a syncopation). See further "r-beat-subdiv".

2. DECODE SOLUTION

In order to have the solution from the search engine in a readable format it is necessary to decode it. This is normally done directly on the output from the engine. However, it is possible to access the internal vectors where the engine store temporary solutions. This might be more dangerous, since this information is overwritten by every new calculation by the search engine.

decode-engine

Syntax

(rc::decode-engine *sol tempo)*

&optional *output presets?*

Input

sol	is the solution from the search engine (“pmc” or “Csolver”).
tempo	is the tempo that a notation window (“voice” or “poly”) will use.
output	(optional) is the format for the output of this box. Default you will get a list of poly-objects. You can also choose to get a list of voice-objects or a list in “simple format”. Choose with the help of the pop-up menu.
presets?	(optional) indicates whether the preset layers are included in the output. Choose with the help of the pop-up menu.

Output

The decoded solution (suitable for either a notation object, or as a readable list).

Description

Decode the solution from the search engine (works for both “pmc” and “Csolver”).

There are different options for the format the box will output:

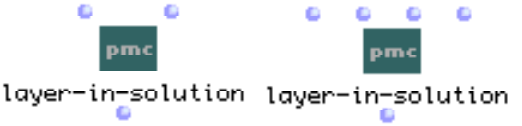
• “poly-list” will output a list of poly-objects, each one representing one voice in the solution. Every new voice in a poly-object will address a new MIDI channel.

• “voice-list” will output a list of voice-objects, each one representing one rhythm layer in the solution. Every new voice will address a new MIDI channel.

• “simple” will output a list in “simple format”: this is a list with a sublist for each voice, containing sublists for the time signatures and every existing rhythm layer.

If no time signatures exist in the solution, 4//4 is used as default.

layer-in-solution



Syntax

(rc::layer-in-solution *voice layer)*

&optional *tempo output*

Input

voice	is the voice number for the layer.
layer	is the layer number for the layer (0 will output the time signatures).
tempo	(optional) is the tempo that a notation window ("voice") will use.
output	(optional) is the format for the output of this box. Default you will get a list of all duration values or time signatures ("simple"). You can also get it as a voice-object ("voice"). Choose with the help of the pop-up menu.

Output

One layer from the last found solution in the chosen format.

Description

Decode one layer from the last found solution (only "pmc"). The information is taken from the internal vectors the search engine use while searching for a solution. Every time the engine runs, it will overwrite these vectors. Before the engine has been run, the vectors are empty. If the search is interrupted, it is also possible to access the partial solution with this box.

view-presets

Syntax

(rc::view-presets *voice*)

&optional *tempo output*

Input

voice	is the number for the voice to be viewed.
tempo	(optional) is the tempo that the notation window (“voice” or “poly”) will use.
output	(optional) is the format for the output of this box. Default you will get a list in “simple format”. You can also choose to get a poly-object or a list of voice-objects. Choose with the help of the pop-up menu.

Output

Description

View all predefined layers in one voice.

There are different options for the format the box will output:

• “poly ” will output a poly-object. Every new voice in the poly-object will address a new MIDI channel.

• “voices ” will output a list of voice-objects, each one representing one rhythm layer in the solution. Every new voice will address a new MIDI channel.

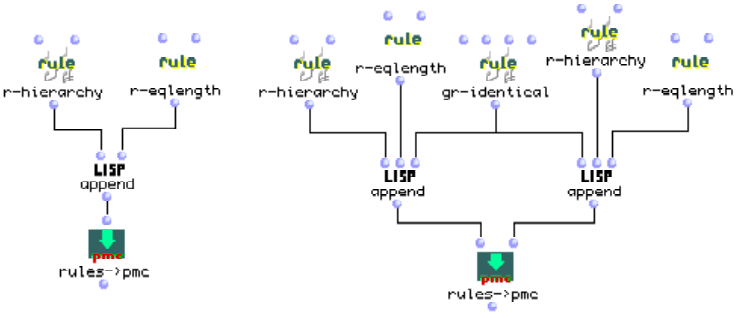
• “simple” will output a list in “simple format”: this is a list with a sub-list for the time signatures and every existing rhythm layer.

If no time signature is present, 4//4 is used as default.

3. RULES

Rules are collected in lists and then input to the “rules->pmc”. The “rules->pmc” resembles the “domain->pmc” box: in the same way the “domain->pmc” collects voice domains, the “rules->pmc” collects sets of rules. In “domain->pmc” the domain for a voice is identified by the entrance it is connected to, in “rules->pmc” a set of rules is valid for the voice corresponding to the input it is connected to. The “heuristicrules->pmc” works the same way (heuristic rules have to be user defined, see section 4).

Global rules (starts with the prefix “gr-”) are rules between layers in different voices. They have to be input separately for every voice they are valid for (“gr-canon” is an exception).



rules->pmc

Syntax

(rc::rules->pmc *voice0*)

&optional *voice1 voice2 voice3 voice4 voice5 voice6*

Input

voice0	is a list of all rules for voice number 0.
voice1	(optional) is a list of all rules for voice number 1.
…voice6	

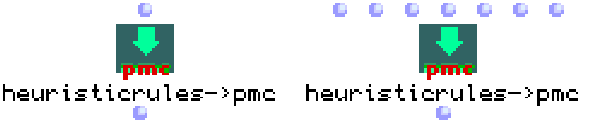
Output

Should be connected to the “pmc” <rules> input.

Description

Format all rules for the pmc <rules> input.
A voice is identified by which entrance the box “voice-domain” is connected to on the “domains->pmc” box. Rules must be connected to the corresponding input on this box to be valid for a voice. See also “domains->pmc”.

heuristicrules->pmc



Syntax

(rc::heuristicrules->pmc *voice0*)

&optional *voice1 voice2 voice3 voice4 voice5 voice6*

Input

voice0	is a list of all heuristic rules for voice number 0.
voice1	(optional) is a list of all heuristic rules for voice number 1.
…voice6	

Output

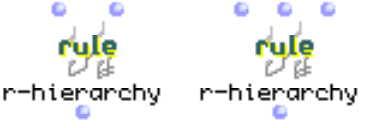
Should be connected to the “pmc” <heuristic-rules> input.

Description

Format all heuristic rules for the pmc <heuristic-rules> input.

A voice is identified by which entrance the box “voice-domain” is connected to on the “domains->pmc” box. Heuristic rules must be connected to the corresponding input on this box to be valid for the voice. See also “domains->pmc”.

r-hierarchy



Syntax

(rc::r-hierarchy *layerhigh layerlow*)

&optional *mode*

Input

<layerhigh>	is the layer number for the higher layer in the hierarchy.
<layerlow>	is the layer number for the lower layer in the hierarchy.
<mode>	(optional) indicates if the hierarchy accepts any event in the lower layer (“normal”), or only starting points for the rhythm cells in the lower layer (“cell-starts”).

Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the “rules->pmc” box.

Description

Rule for making a hierarchical connection between two layers in a voice.

The rule creates a hierarchical structure for events, where the higher layer only contains the events of “high importance” (higher up in this structure) from the lower layer. Thus all starting points for events in the higher layer must exist in the lower layer. The lower layer might include events with starting points between starting points in the higher layer (= events of less importance).
“Cell-start” mode is a stricter rule. Here the starting points in the higher layer has to occur at starting points for a whole cell in the lower layer (i.e. it is not enough that any event in the lower layer occur simultaneously as events in the higher layer).

r-eqlength

Syntax

(rc::r-eqlength *layer1 layer2*)

Input

layer1	is the layer number for one of the layers.
layer2	is the layer number for the other layer.

Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the “rules->pmc” box.

Description

Rule for forcing two layers in a voice to be of as equal length as possible.

The rule will only accept a cell for the for the moment shortest layer as the next cell in the solution. It also works for layer 0 (the time signatures) – it will then make sure the sequence of time signature makes up a long enough sequence for the rhythm layer.

r-identical

Syntax

(rc::r-identical *layer1 layer2)*

Input

layer1	is the layer number for one of the layers.
layer2	is the layer number for the other layer.

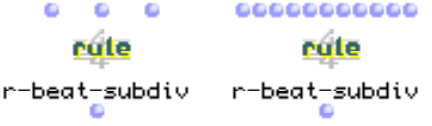
Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the "rules->pmc" box.

Description

Rule to force two layers in a voice to be identical.

r-beat-subdiv



Syntax

(rc::r-beat-subdiv *rhythm-layer-nr beatvalue1 subdiv1*)

&optional *beatvalue2 subdiv2 beatvalue3 subdiv3 beatvalue4 subdiv4 beatvalue5 subdiv5*

Input

rhythm-layer-nr is the number for the rhythm layer the rule is valid for.

beatvalue1 and subdiv1

come in a pair. subdiv1 is a list of possible subdivisions of the beat in a measure. beatvalue1 is the length of the beat (i.e. the lower number in the time signature) that subdiv1 refers to.

beatvalue2 and subdiv2 (optional)
…beatvalue5 and subdiv5

Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the “rules->pmc” box.

Description

Rule for controlling the subdivisions of beats in measures.

Example: If beatvalue1 = 4, then subdiv1 tells how the beats in a measure with a time signature with the beat length a quarter note (for example 4//4, 3//4) can be subdivided. If subdiv1 = (1 2 3 4 5) this means that the beat can be subdivided down to maximum a quintuplet, however only if this is on the beat (i.e. not as a syncopation). In this example the rule will not affect measures with time signatures that have for example eight-note as the beat length; every beat length has to be defined separately (in pairs of betvalue and subdiv).

r-canon

Syntax

(rc::r-canon *layercomes layerdux offset*)

Input

layercomes is the layer number for the layer that will imitate the original rhythm.
layerdux is the layer number for the original rhythm.
offset is the distance in time (duration value as a ratio) between the original rhythm and its imitation.

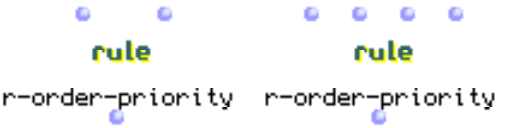
Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the “rules->pmc” box.

Description

Rule to create a rhythmical canon. The layers have to be in the same voice (compare with “gr-canon”).

r-order-priority



Syntax

(rc::r-order-priority *layer1 layer2*)

&optional *layer3 layer4*

Input

layer1	are the layer number for the layer that the engine will look at first of the specified layers in this rule when searching for a solution..
layer2	are the layer number for the layer that the engine will look at second of the specified layers in this rule when searching for a solution..
layer3	(optional) are the layer number for the layer that the engine will look at third of the specified layers in this rule when searching for a solution..

…layer5

Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the “rules->pmc” box.

Description

Rule to force the engine to find the solution looking at the layers in a specified order. The inputs indicates what order (from left to right) the engine should look at the layers when searching for a solution.

This rule is experimental. It might speed up the search process (or might slow it down, if used in a wrong way). It might also help to find a specific answer, giving priority to certain layers. WARNING: The rule might forbid solutions that otherwise would have been possible.

gr-hierarchy

Syntax

(rc::r-hierarchy *layerhigh voicehigh layerlow voicelow*)

&optional *mode*

Input

layerhigh	is the layer number for the higher layer in the hierarchy.
voicehigh	is the voice number for the higher layer in the hierarchy.
layerlow	is the layer number for the lower layer in the hierarchy.
voicelow	is the voice number for the lower layer in the hierarchy.
mode	(optional) indicates if the hierarchy accepts any event in the lower layer (“normal”), or only starting points for the rhythm cells in the lower layer (“cell-starts”).

Output

is a list with the rule. The rule should be collected together with other rules for a voice (use append) and then input to the “rules->pmc” box. It has to be connected to each set of rules for the two voices.

Description

Rule for making a hierarchical connection between two layers in different voices.

The rule creates a hierarchical structure for events, where the higher layer only contains the events of “high importance” (higher up in this structure) from the lower layer. Thus all starting points for events in the higher layer must exist in the lower layer. The lower layer might include events with starting points between starting points in the higher layer (= events of less importance).

“Cell-start” mode is a stricter rule. Here the starting points in the higher layer has to occur at starting points for a whole cell in the lower layer (i.e. it is not enough that any event in the lower layer occur simultaneously as events in the higher layer).

gr-eqlength

Syntax

(rc::r-eqlength *layer1 voice1 layer2 voice2*)

Input

layer1	is the layer number for the first layer.
voice1	is the voice number for the first layer.
layer2	is the layer number for the second layer.
voice2	is the voice number for the second layer.

Output

is a list with the rule. The rule should be collected together with other rules for a voice (use append) and then input to the “rules->pmc” box. It has to be connected to each set of rules for the two voices.

Description

Rule for forcing two layers in different voice to be of as equal length as possible.

The rule will only accept a cell for the for the moment shortest layer as the next cell in the solution. It also works for layer 0 (the time signatures) – it will then make sure the sequence of time signature makes up a long enough sequence for the rhythm layer.

gr-identical

Syntax

(rc::r-identical *layer1 voice1 layer2 voice2*)

Input

layer1	is the layer number for the first layer.
voice1	is the voice number for the first layer.
layer2	is the layer number for the second layer.
voice2	is the voice number for the second layer.

Output

is a list with the rule. The rule should be collected together with other rules for a voice (use append) and then input to the “rules->pmc” box. It has to be connected to each set of rules for the two voices.

Description

Rule to force two layers in different voices to be identical.

gr-canon

Syntax

(rc::gr-canon *layercomes layerdux voicedux offset*)

Input

layercomes	is the layer number for the layer that will imitate the original rhythm.
layerdux	is the layer number for the original rhythm.
voicedux	is the voice number for the original rhythm.
offset	is the distance in time (duration value as a ratio) between the original rhythm and its imitation.

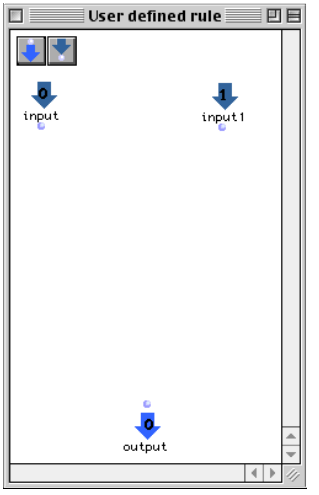
Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the "rules->pmc" box. The rule should be connected to the voice where the imitation is (not the voice for the original rhythm).

Description

Rule to create a rhythmical canon. The layers can be in different voices (compare with "r-canon").

4. USER DEFINED RULES



A user defined rule is always done within a sub-patch in the lambda state (see the OpenMusic Manual). It should always have two inputs (which give the search engine access to the rule) and one output. The left input is the index the search engine uses internal while searching for a solution (called indexx in the input in the functions below). The right input is the current search variable (called x in the input to the functions below). A rule tests a variable and gives the result true or false back to the search engine.

Heuristic rules are identical with one exception: instead of giving true or false they give weights back to the search engine. The engine will choose the variable that gives the highest weight.

get-this-cell

Syntax

(rc::get-this-cell *x*)

Input

x	is the current variable. It should be connected to the second input inside a user rule patch.

Output

The content of the current variable.

Description

Get the content of the current instantiated variable (i.e. the current rhythm cell or time signature). NO DIFFERENCE WILL BE MADE BETWEEN PAUSES AND NOTES IN RHYTHM CELLS.

get-this-cell-dur

Syntax

(rc::get-this-cell-dur *x)*

Input

x	is the current variable. It should be connected to the second input inside a user rule patch.

Output

The length of the current variable.

Description

Get the length of the current variable (i.e. the length of the current rhythm cell or measure).

get-last-cell

Syntax

(rc::get-last-cell *indexx x*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.

Output

The content of the variable in the same layer as the current variable, before the current variable.

Description

Get the content of the variable in the same layer as the current one, but before the current one (i.e. a rhythm cell or time signature). NO DIFFERENCE WILL BE MADE BETWEEN PAUSES AND NOTES IN RHYTHM CELLS.

get-cell-before-last

Syntax

(rc::get-cell-before-last *indexx x*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.

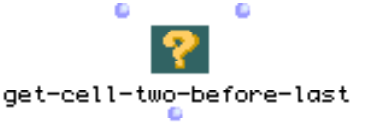
Output

The content of the variable in the same layer as the current variable, two before the current variable.

Description

Get the content of the variable in the same layer as the current one, but two before the current one (i.e. a rhythm cell or time signature). NO DIFFERENCE WILL BE MADE BETWEEN PAUSES AND NOTES IN RHYTHM CELLS.

get-cell-two-before-last



Syntax

(rc::get-cell-two-before-last *indexx x*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.

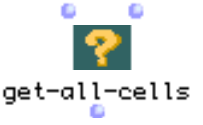
Output

The content of the variable in the same layer as the current variable, three before the current variable.

Description

Get the content of the variable in the same layer as the current one, but three before the current one (i.e. a rhythm cell or time signature). NO DIFFERENCE WILL BE MADE BETWEEN PAUSES AND NOTES IN RHYTHM CELLS.

get-all-cells



Syntax

(rc::get-all-cells *indexx x*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.

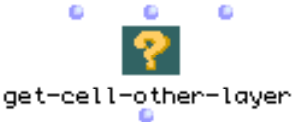
Output

The content of all variables in the same layer as the current one (except the current one) up till now.

Description

Get the content of all variables in the same layer as the current one (i.e. a list of all rhythm cells or time signatures). The current variable is not included. NO DIFFERENCE WILL BE MADE BETWEEN PAUSES NOTES IN RHYTHM CELLS.

get-cell-other-layer



Syntax

(rc::get-cell-other-layer *indexx x layer*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.

layer is the layer number for the layer where the variable exist.

Output

The content of the variable in another layer, but in the same voice, as the current variable, that exists at the starting point for the current variable.

Description

Get the content of the variable in another layer, but in the same voice, as the current instantiated variable. The variable exists at the starting point for the current variable (i.e. a rhythm cell or a time signature). If no variable exist yet, nil will be returned. NO DIFFERENCE WILL BE MADE BETWEEN PAUSES AND NOTES IN RHYTHM CELLS.

get-cell-any-layer

Syntax

(rc::get-cell-any-layer *indexx x layer voice*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.
layer	is the layer number for the layer where the variable exist.
voice	is the voice number for the voice where the variable exist.

Output

The content of a variable in any layer and voice that exists at the starting point for the current variable.

Description

Get the content of a variable in any layer and voice that exists at the starting point for the current variable (i.e. a rhythm cell or a time signature). If no variable exist yet, nil will be returned. NO DIFFERENCE WILL BE MADE BETWEEN PAUSES AND NOTES IN RHYTHM CELLS.

get-rhythm-other-layer

Syntax

(rc::get-rhythm-other-layer *indexx x layer*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.
layer	is the layer number for the layer where the rhythm exist.

Output

The rhythm in another layer, but in the same voice, as the current variable, that occur within the timeframe for the current instanced variable.

Description

Get the rhythm in another layer, but in the same voice, as the current variable, that occur within the timeframe for the current instanced variable. This is not similar to a rhythm cell, since parts of one or several cells might occur within the timeframe. If no rhythm exist yet, nil will be returned. NO DIFFERENCE WILL BE MADE BETWEEN PAUSES AND NOTES.

get-rhythm-any-layer

Syntax

(rc::get-rhythm-any-layer *indexx x layer voice*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.
layer	is the layer number for the layer where the rhythm exist.
voice	is the voice number for the voice where the rhythm is.

Output

The rhythm that occurs within the timeframe for the current variable in any layer and voice.

Description

Get a rhythm that occurs within the timeframe for the current variable in any layer and voice. This is not similar to a rhythm cell, since parts of one or several cells might occur within the timeframe. If no rhythm exist yet, nil will be returned. NO DIFFERENCE WILL BE MADE BETWEEN PAUSES AND NOTES.

get-all-cells-any-layer

Syntax

(rc::get-all-cells-any-layer *indexx layer voice*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.
layer	is the layer number for the layer.
voice	is the voice number for the voice where the layer is.

Output

The content of all variables in any layer and voice.

Description

Get the content of all variables in any layer and voice. NO DIFFERENCE WILL BE MADE BETWEEN PAUSES AND NOTES IN RHYTHM CELLS.

get-time

Syntax

(rc::get-time *indexx x*)

Input

indexx	is the index for the current variable (index is used internal by the search engine). It should be connected to the first input inside a user rule patch.
x	is the current variable. It should be connected to the second input inside a user rule patch.

Output

The start time for the current instantiated variable.

Description

Get the start time for the current variable. The time will be given as a ratio, equivalent to the total note duration from the start of the sequence.

get-layernumber

Syntax

(rc::get-layernumber *x*)

Input

x	is the current variable. It should be connected to the second input inside a user rule patch.

Output

The layer number for the current variable.

Description

Get the layer number for the current variable.

rhythmcell?

Syntax

(rc::rhythmcell? *x*)

Input

x	is the current variable. It should be connected to the second input inside a user rule patch.

Output

If the current variable is a rhythm cell: true, otherwise: false.

Description

Test if the current variable is a rhythm cell.

timesign?

Syntax

(rc::timesign? *x*)

Input

x	is the current variable. It should be connected to the second input inside a user rule patch.

Output

If the current variable is a time signature cell: true, otherwise: false.

Description

Test if the current variable is a time signature cell.

test-equal

Syntax

(rc::test-equal *ev1 ev2*)

Input

ev1	is one of the events to compare.
ev2	is the other.

Output

If ev1 and ev2 are identical: true, otherwise: false.

Description

Test if two cells (rhythm cells or time signatures, also works for numbers and lists) are identical.

test-not-equal

Syntax

(rc::test-not-equal *ev1 ev2*)

Input

ev1	is one of the events to compare.
ev2	is the other.

Output

If ev1 and ev2 are different: true, otherwise: false.

Description

Test if two cells (rhythm cells or time signatures, also works for numbers and lists) are different from each other.

5. STORE AND LOCK TO PARTIAL SOLUTIONS

store-section

Syntax

(rc::store-section *voice timepoint1 timepoint2*)

Input

voice — is the number for the voice from which the section will be taken.

timepoint1 and timepoint2 — the start time point and the end time point in the last solution between which the section will be taken. The time point is given as a list of measure number and offset, for example (2 1/4) means one quarter note after the start of measure number 2.

Output

should be connected to the rule "r-lock-to-stored", or optional you can generate instances of the class "stored-section". If the box is used connected directly to the "r-lock-to-stored", it has to be locked first.

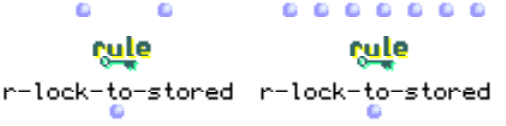
Description

Store a section from the last solution (pmc).

It is highly recommended that you always lock this box before evaluating it – in the new solution the bar lines might be on new positions, and the <timepoint1> and <timepoint2> will then not have the same meaning.

Do not confuse the last found solution with what might be visible in a notation window (the information is taken from internal vectors connected to the search engine – every time the search engine is run, the last solution will be overwritten).
See below "r-lock-to-stored".

r-lock-to-stored



Syntax

(rc::r-lock-to-stored *section layernr0*)

&optional *layernr1 layernr2 layernr3 layernr4 offset*)

Input

section	should either be connected to a “store-section” box, or an instance of the class “stored-section”.
layernr0	is the layer number for a layer to lock.
layernr1	(optional) is the layer number for a layer to lock.
…layernr4	
offset	You can optional move the stored section to a new position in the sequence. Offset is the distance (duration value as a ratio) from the start of the sequence. If offset is nil, the section will be locked to its original position.

Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the “rules->pmc” box.

Description

Rule that lock a section in the solution to be identical to a stored section from an earlier solution. One or several layers (including time signatures) can be locked.

decode-stored-section

Syntax

(rc::decode-stored-section *section*)

&optional *output*

Input

section	should be connected to the box “store-section” or an instance of the class “stored-section”.
output	(optional) is the format for the output of this box. Default you will get a poly-object. You can also choose to get a list of voice-objects or a list in “simple format”. Choose with the help of the pop-up menu.

Output

the decoded section in the chosen format.

Description

Decode the box “store-section” or an instance of the class “stored-section” to a score or to simple format.

There are different options for the format the box will output:
• “poly ” will output a poly-object. Every new voice in the poly-object will address a new MIDI channel.
• “voices ” will output a list of voice-objects. Every new voice will address a new MIDI channel.

• “simple” will output a list in “simple format”: this is a list with sub-lists for the time signatures and every existing rhythm layer.

simpleformat->tree

Syntax

(rc::simpleformat->tree *rhythm timesigns*)

Input

rhythm	is a list of note values as ratios.
timesigns	is either a list of time signatures, or one time signature.

Output

is an OpenMusic standard rhythm tree.

Description

Convert a list of note values to a hierarchical rhythm-tree. A negative note value indicates a pause. If a list of time signatures is given, this sequence will be used. If only one time signature is given, this will be used as many times as needed to notate the sequence.

tree->simpleformat

Syntax

(rc::tree->simpleformat *tree*)

Input

tree	is a rhythm tree.

Output

A list of note values as ratios. A negative value indicates a pause.

Description

Convert an OpenMusic standard rhythm tree to a list of note values (ratios).
Pauses following each other will be fused to one (long) pause. Slured note values will be fused to one note value.

rhythmdomain->voices

Syntax

(rc::rhythmdomain->voices *domain*)

Input

domain is a list of rhythm cells, notated as lists of ratios.

Output

is a list of voices.

Description

Convert a domain (or any list of rhythm cells) to a list of voices. 4/4 will be used as meter. The longest cell decides the length for the notation.

7. CSOLVER COMPATIBILITY

Normally the pmc is used as the search engine for this library; However the Csolver from the Situation library can be used as well. Different engines work with data structured in different ways. To be able to use the Csolver, you have to change 4 of the boxes for the pmc documented above, to the equivalent boxes that support the Csolver.

domains->pmc	=>	rc::domains->csolver
rules->pmc	=>	rc::rules->csolver
layer-in-solution	=>	rc::layer-in-sol/csolv
store-section	=>	rc::store-section-csolver

These boxes are not present in the menus, but they can be used by command-clicking and dragging the mouse in a patch, and then typing there names (see the OpenMusic manual).

r-layerorder

Syntax

(rc::r-layerorder *layer1 layer2 endtime*)

Input

layer1	is the number for the layer that should be calculated first.
layer2	is the number for the layer that should be calculated after the first layer.
endtime	is the point to which the first layer should reach before starting calculating the second layer. This is defined as a note value (ratio).

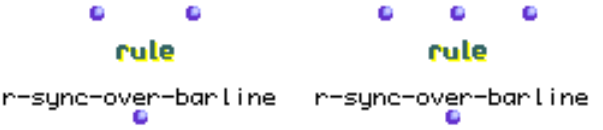
Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the “rules->pmc” box.

Description

Rule for forcing one layer in a voice to be calculated before another layer in the same voice up to a certain time point. After this point the rule works in the same way as r-eqlength.

r-sync-over-barline



Syntax

(rc::r-sync-over-barline *rhythm-layer-nr allowed-syncopes*)

&optional *special-case*

Input

rhythm-layer-nr	is the number for the rhythm layer the rule is valid for.
allowed-syncopes	is a list of possible positions for the first event in a measure. They are defined as offsets (note value as a ratio) from the bar line.
special-case	is a popup menu where you can choose if it is allowed that a measure has no events, i.e. that no starting points for events occur in the measure.

Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the “rules->pmc” box.

Description

Rule for controlling the position of the first event after a bar line, i.e. control of syncopations over bar lines.

Ex: If the second input is '(0 1/8), and the option no-empty-measures is chosen in the third input, every new measure must either start with a new event at the first beat, or have the first event starting one eight note after the bar line (the first eight note in the measure is then slured from the previous measure).

r-pattern

Syntax

(rc::r-pattern *layer nr-of-ev*)

&optional *duration*

Input

layer	is the number for the rhythm layer the pattern will be built.
nr-of-ev	is the number of events in the pattern.
duration	is the length of the pattern.

Output

is a list with the rule. Should be collected together with other rules for a voice (use append) and input to the “rules->pmc” box.

Description

Rule for building a pattern. The rule does not understand the difference between notes and pauses. A pause will be treated as a note.

Ex: If nr-of-ev is 4 and duration is 1, then a pattern with the total length of a whole note with four rhythm events in it will be built (and repeated over and over again) in the layer.

gr-layerorder

Syntax

(rc::gr-layerorder *layer1 voice1 layer2 voice2 endtime*)

Input

layer1	is the number for the layer that should be calculated first.
voice1	is the voice number for the layer that should be calculated first.
layer2	is the number for the layer that should be calculated after the first layer.
voice2	is the voice number for the layer that should be calculated after the first layer.
endtime	is the point to which the first layer should reach before starting calculating the second layer. This is defined as a note value (ratio).

Output

is a list with the rule. The rule should be collected together with other rules for a voice (use append) and then input to the “rules->pmc” box. It has to be connected to each set of rules for the two voices.

Description

Rule for forcing one layer in a voice to be calculated before another layer in another voice up to a certain time point. After this point the rule works in the same way as r-eqlength.

The rule also works with layer 0 (the time signatures).

The rule has to be connected to the input for both layers on “rules->pmc”.

get-voicenumber

Syntax

(rc::get-voicenumber *x*)

Input

x	is the current variable. It should be connected to the second input inside a user rule patch.

Output

The layer number for the current variable.

Description

Get the voice number for the current variable.

Index

T

U

V