PWGL LIBRARY

Jacopo Baboni Schilingi

JBS-CONSTRAINTS

jbs-constraints
(v 0.1)

January 18, 2010

Contents

1 Start-Here

This documentation has been written by Jacopo Baboni Schilingi and Julien Vincenot.
This is a library to control the Multi-PMC and the Multi-Score-PMC functions of PWGL.

(1) Multi-PMC

(2) Multi-Score-PMC

(3) Utils

(4) Examples

This library is the result of many years of work with Mikael Laurson, Mika Kuuskankare and Kilian Sprotte. All the functions or methods I formalized here are based on the syntax of Constraints done by Mikael Laurson. Please let me avoid any misunderstanding : this library is called JBS-Constraints exactly because it is my own way of using Mikael Laurson syntax to generate rules. That does not mean at all that the method I suggest is the only one. I developed this method because of my own hyper-systemic theory approach and because practice showed me that this way is the best for ME. Study also the tutorial of Mikael Laurson concerning constraints, because it might happen that my method is not the one you are looking for.
Please, if you find some bugs or anomalies, write directly to me at jbs@baboni-schilingi.com.

2 0-Multi-PMC

2.1 00-Introduction-to-Constraints

2.1.1 Introduction

This chapter is dedicated to explain how the Multi-PMC works. Please follow these tutorials from the beginning to the end.

2.1.2 0-The-Multi-PMC

1.00.0 - THE-MULTI-PMC
This function [1] is like an engine that finds out the solutions corresponding to a given problem. Imagine that we want to find all the permutations of a given list of 5 letters of the alphabet (a b c d e).
Just learn this :
In [a] you have to put a list of variables that will constitute the search space where the engine will look for the solution.
In [b] you have to put the rules called true/false : those rules will be applied without any tolerance.
In [c] you will put the heuristic rules : those rules will only be applied as much as possible.

In [d] you chose if you look for the solution following the order of the candidates - if you set NIL () - or in a random order - if you set T.
In [e] you decide how many solutions you are looking for: from 1 to :all.

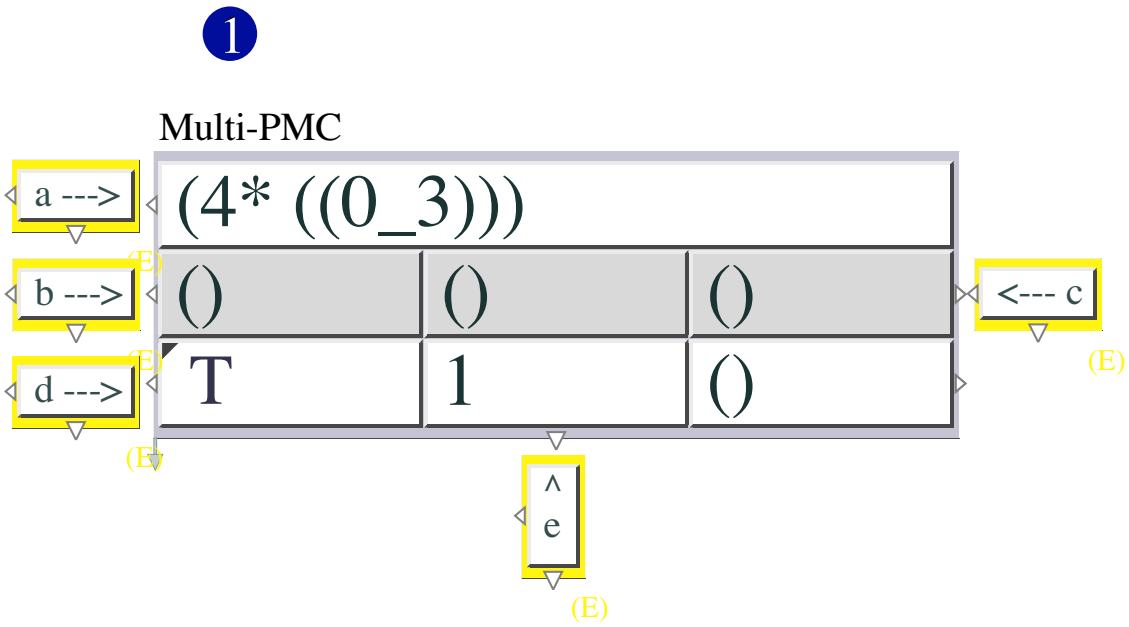

Figure 1: 1-00-0-The-Multi-PMC

2.1.3 1-Create-Candidates-for-the-Multi-PMC

1.00.1 - CREATE-CANDIDATES-FOR-THE-MULTI-PMC
Let (a b c d e) be your candidates [a]. In order to make the Multi-PMC look for one or all solutions, you have to duplicate your list of candidates for a number of times [b] to produce a list of variables (here we create 5 variables). The number of variables determines the length of the solutions.
Please, put 5 in [b] and evaluate the Multi-PMC [1]. A random solution occurs any time you evaluate this box.
Now change the number in [b]. If you put 2, for instance, your result will have only two elements. If you put 11, your result will have eleven elements chosen among (a b c d e).
Then change the menu [c] and choose the NIL position (). If you evaluate the Multi-PMC now you will see that the solution will have only a list of -A- with a length define

in [b].
Please change the value of the input [d], keeping the input [c] in the nil position (). You will obtain the amount of solutions that you defined in [d]. For instance, if [d] is equal to 4, you will have the following result : ((A A A A D) (A A A A C) (A A A A B) (A A A A A)). That means that the first result is (A A A A A), the second (A A A A B), the third (A A A A C), etc. The solutions are pushed from the first to the last.
Now, if you put again the [c] input in the T mode, you will have a number [d] of random solutions, each of them having the same length, defined in [b].

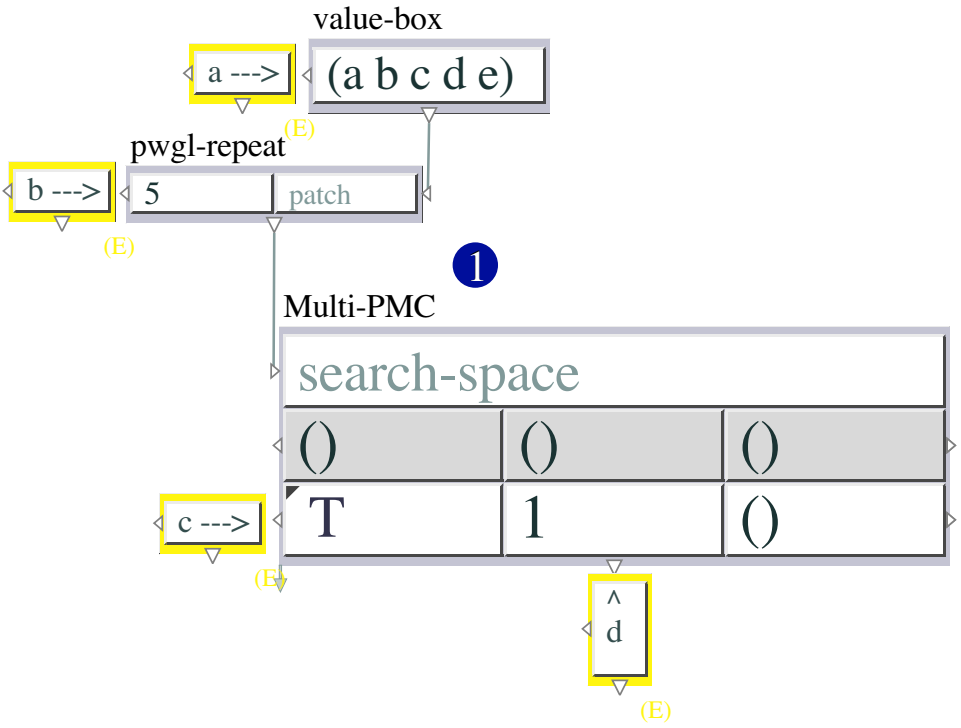

Figure 2: 1-00-1-Create-candidates-for-the-Multi-PMC

2.1.4 2-The-Multi-PMC-Rules-Application

1.00.2 - THE-MULTI-PMC-RULES-APPLICATION
Here the goal is to create one or all permutations of the (a b c d e) candidates, having the letter -A- always in the third position.
To do this you need a first concept that is called NO-REPETITION [e]. This function is in reality a rule for the Multi-PMC : it forbids the Multi-PMC to have any repetition of -A-, or -B-, or -C-, etc., inside the solution.
So in [a] you put your candidates. In [b] you define how many variables your solution

will be constituted with. In [c] if you set the T mode to ask for a random research. Put 1 in [d], in order to obtain only one solution for each evaluation.
The function [f] is another rule for the Multi-PMC [1]. This rule obliges the Multi-PMC to produce a result with the letter -A- in its third place. The COLLECT-RULES box [g] is a collector for all the rules you need.
Now evaluate many times the Multi-PMC and see the results in the PWGL output : each time you will produce different results keeping the letter -A- always in the third position.

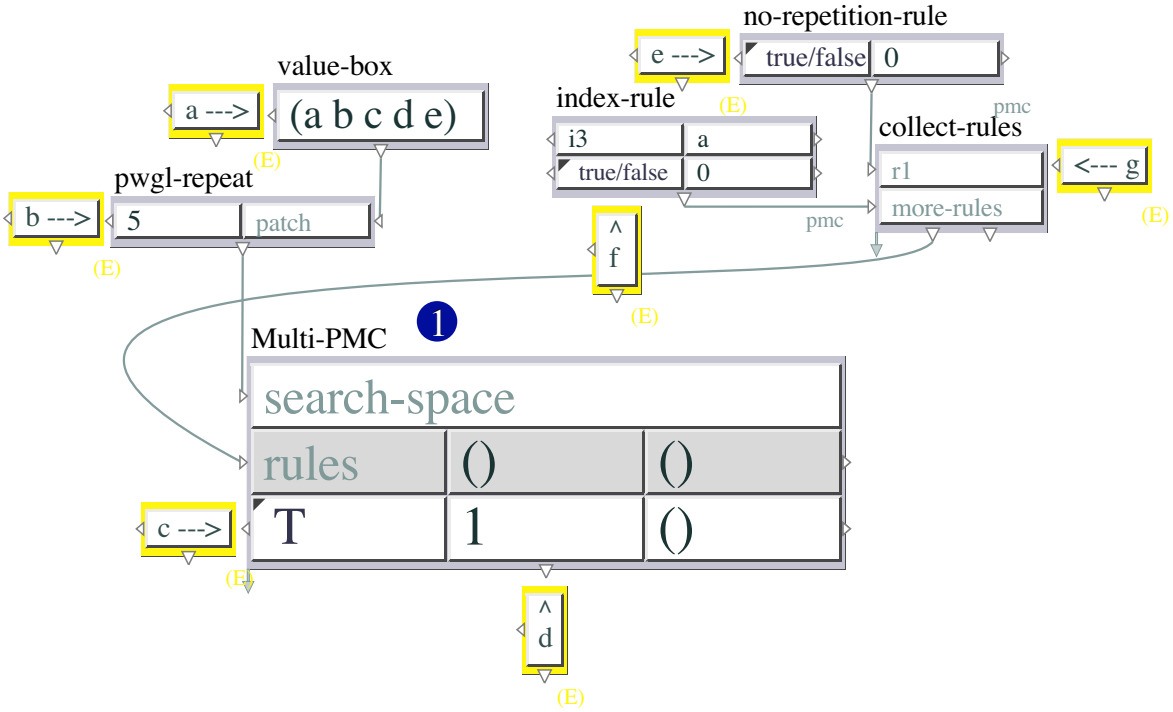

Figure 3: 1-00-2-The-Multi-PMC-rules-application

2.1.5 3-The-Logical-Conflict-and-the-Heuristic-Rules

1.00.3 - THE-MULTI-PMC-LOGICAL-CONFLICT-AND-THE-HEURISTIC-RULES
Look to this problem. This patch is identical to the 1.00.2, excepted that the input [b] is set to the value 6. If you evaluate the Multi-PMC [1] you do not obtain any solution. Why? Because you are asking the engine to find a solution of 6 elements with only 5 candidates (a b c d e) and without any repetition. It means that you are producing a logical conflict : this is impossible to have a solution of 6 different elements using only 5 different variables.
To solve this problem you have to imagine that each rule set in the true/false mode is

applied strictly. If no solution satisfies the rules set in the COLLECT-RULES [f], there can't be any result. For this reason chose, in the menu of the box NO-REPETITION-RULE [h], the heuristic mode. Put in the [i] input of the NO-REPETITION-RULE a value bigger than 1. This value is the weight of application of this rule. Then the Multi-PMC outputs solutions again. (Note : For this moment, please, do not ask more, just do what I suggest. I will explain in the next steps the concept of weight.)
To resume : In [a] you put the candidates that constitute the solution to find. In [b] you define the length of the solution, in this case a length of 6 elements.
The [h] input is set in the heuristic mode with a weight [i] bigger than 1. This means that the NO-REPETITION-RULE will be applied as much as possible (heuristically). The [g] rule, as set in the true/false mode, obliges the solution to have -A- in the third place. The [f] function collects the heuristic rule and the true/false rule. [c] is the input of the Multi-PMC accepting the true/false rules coming out of the left input of the COLLECT-RULES [f]. [d] is the input of the Multi-PMC accepting the heuristic rules coming out of the right input of the COLLECT-RULES [f]. The value 1 put in [e] asks for only one solution at a time.
OBVIOUSLY, as the NO-REPETITION-RULE is applied as much as possible, some repetitions appear.
For the moment REMEMBER that a true/false rule is applied strictly. If one true/false rule can not be applied (logical conflict) no solution is possible. A heuristic rule is applied as much as possible, depending on it weight.

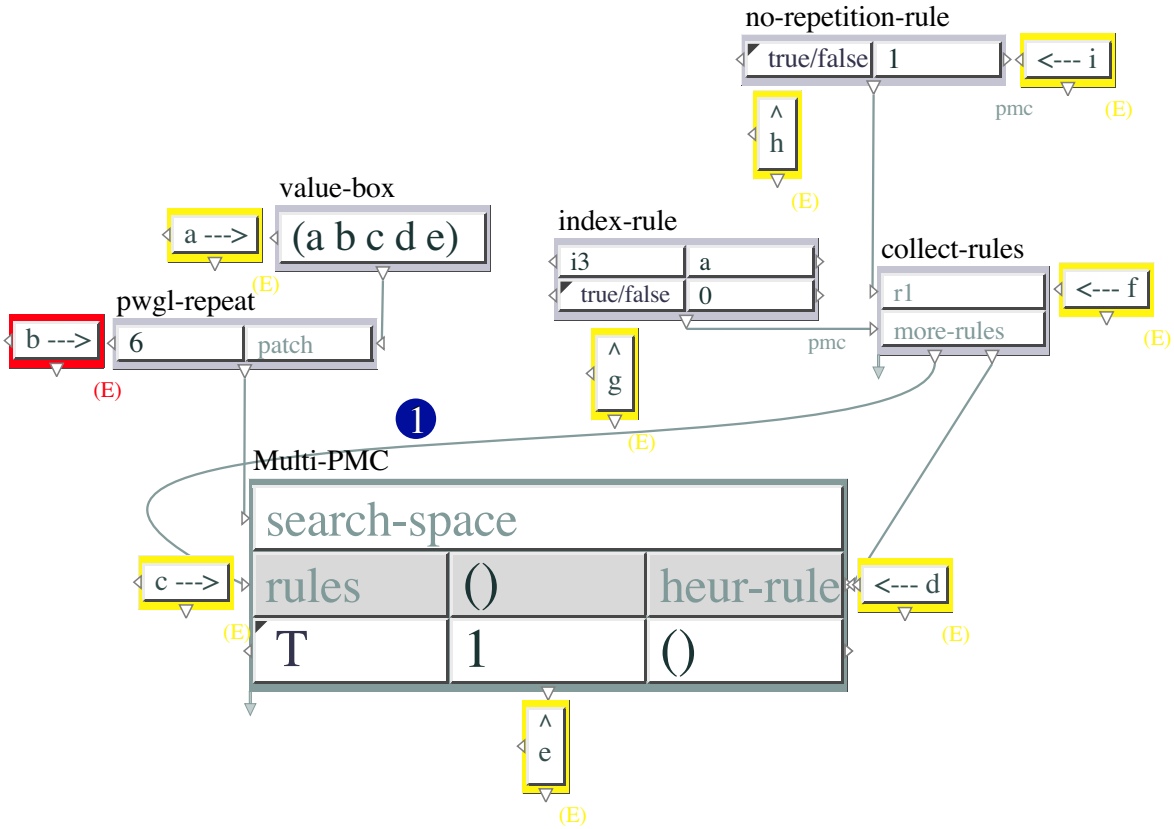

Figure 4: 1-00-3-The-logical-conflict-and-the-heuristic-rules

2.1.6 4-A-First-Musical-Example

1.00.4 - A-FIRST-MUSICAL-EXAMPLE
In [1] you put the candidates you want the solution to be constituted with, in this case a C-major scale. Put the value 5 in [2] in order to produce a solution of length 5.
[b] is a switch : you can choose by clicking in the left input not to apply the associated rule. If you click on the right input you make the correlated rule active.
[3] is a rule concerning intervals. Here it admits only intervals equals to a minor and major second and a major third. Please note that intervals are expressed in semitones : 1 is a minor second, 2 a major second, 3 a minor third, 4 a major third and so on.
[4] is, as we have already seen, a rule concerning the place of an element. In this case we ask the Multi-PMC to keep always the E note (64) in the third place.
[5] is the rule that forbids to have any repetition of any note in the result.
[6] is the collector of rules. Please OBSERV that the order of the input of the rules is irrelevant.
[7] is the Multi-PMC set in the T mode for a random research and looking for only one solution at a time.

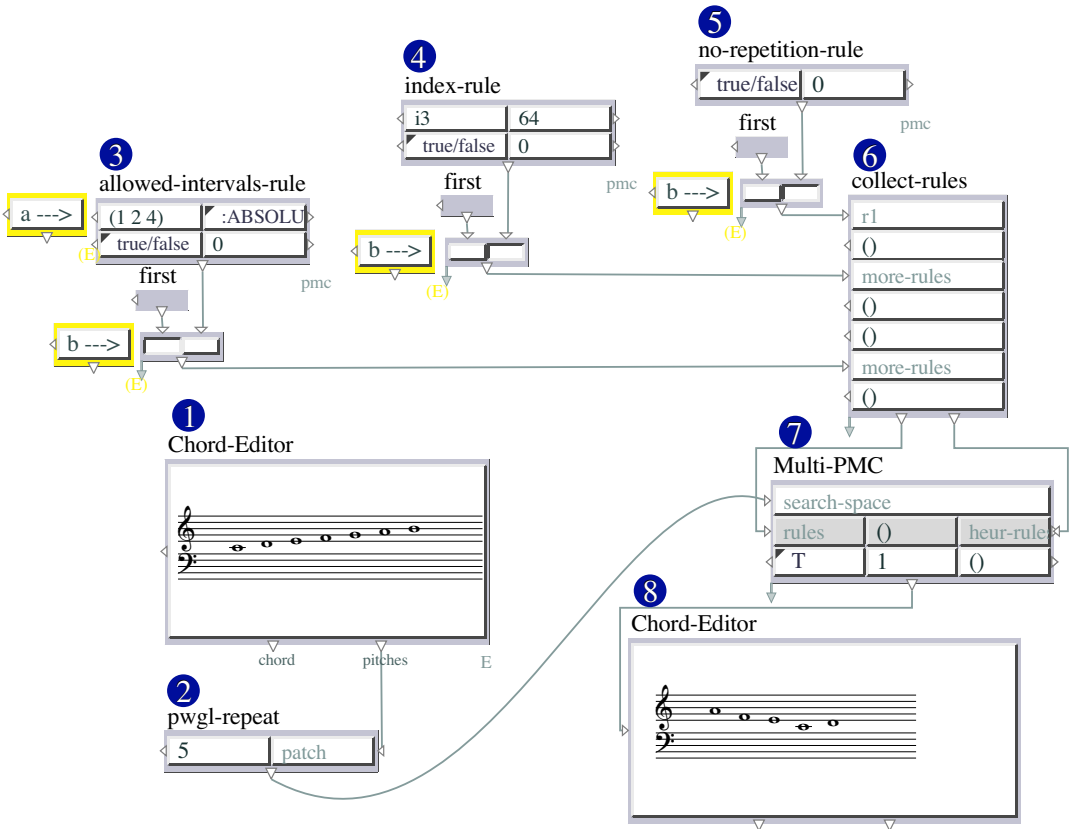

Figure 5: 1-00-4-A-first-musical-example

2.1.7 5-Specific-Candidates

1.00.5 - SPECIFIC-CANDIDATES
The values repeated several times, as we have seen before, are duplicated in order to define a search space for the Multi-PMC. We call these values CANDIDATES. Using the PWGL-REPEAT box you duplicates the candidates in order to create a certain amount of variables.
Sometimes you may need to define the variables not as duplicates of a given group of candidates, but as several subgroups. In this patch you can see how to do it.
[1], [2], [3], [4] and [5] are different groups of candidates that will correspond to different variables. The Multi-PMC is set to look randomly for research and to output only one solution for each evaluation. Please OBSERV that in [5] only the C (48 in midi note) is defined. So in this case the fifth element of any solution will be always this C (48 in midi note).
Please evaluate the Multi-PMC many time and look at the results. This way of preparing specific candidates optimizes the research for the Multi-PMC.

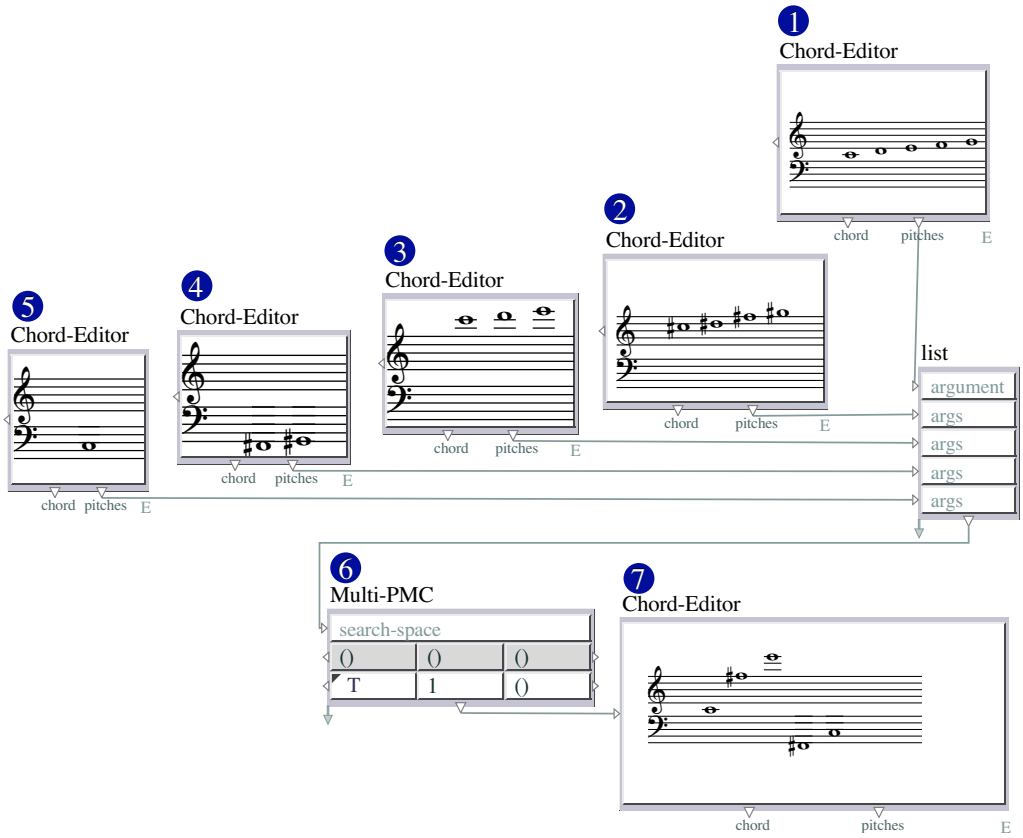

Figure 6: 1-00-5-Specific-candidates

2.1.8 6-Heuristic-Rules-and-No-Random-Reserach

1.00.6 - HEURISTIC-RULES-AND-NO-RANDOM-RESEARCH
Now I can explain you the notion of weight for the heuristic rules. You have to imagine the heuristic rules as a kind of wish. If we stay in this metaphor the concept will be easy to understand.
In [1] I have put a chromatic scale from midi note 60 to 71. In [2] I define how much notes will constitute my result.
[3] is a rule of shaping set in the heuristic mode with a weight of 1. We will see later in details, but for now imagine just know that this rule obliges the Multi-PMC to find a solution following as much as possible the shape designed in the 2D-EDITOR [4]. Here the NUM-BOX [2] defines also how much points will sample the 2D-EDITOR (or the BPF curve, in other words).
If you evaluate the Multi-PMC [6] you will see that, despite the fact that it is set in T mode (for a random research), the solutions [7] are always the same. That is because the heuristic rules re-dispose the candidates in an order that is set by the given rule.
For the moment, please, change the value in [2] using for instance the value 6 or 9 or 11 or 13, etc. You will see that all this results are stable : in other words, if you evaluate

many times with the same number set in [2] the solution does not change. They follow as much as possible the profile entered in the 2D-EDITOR.

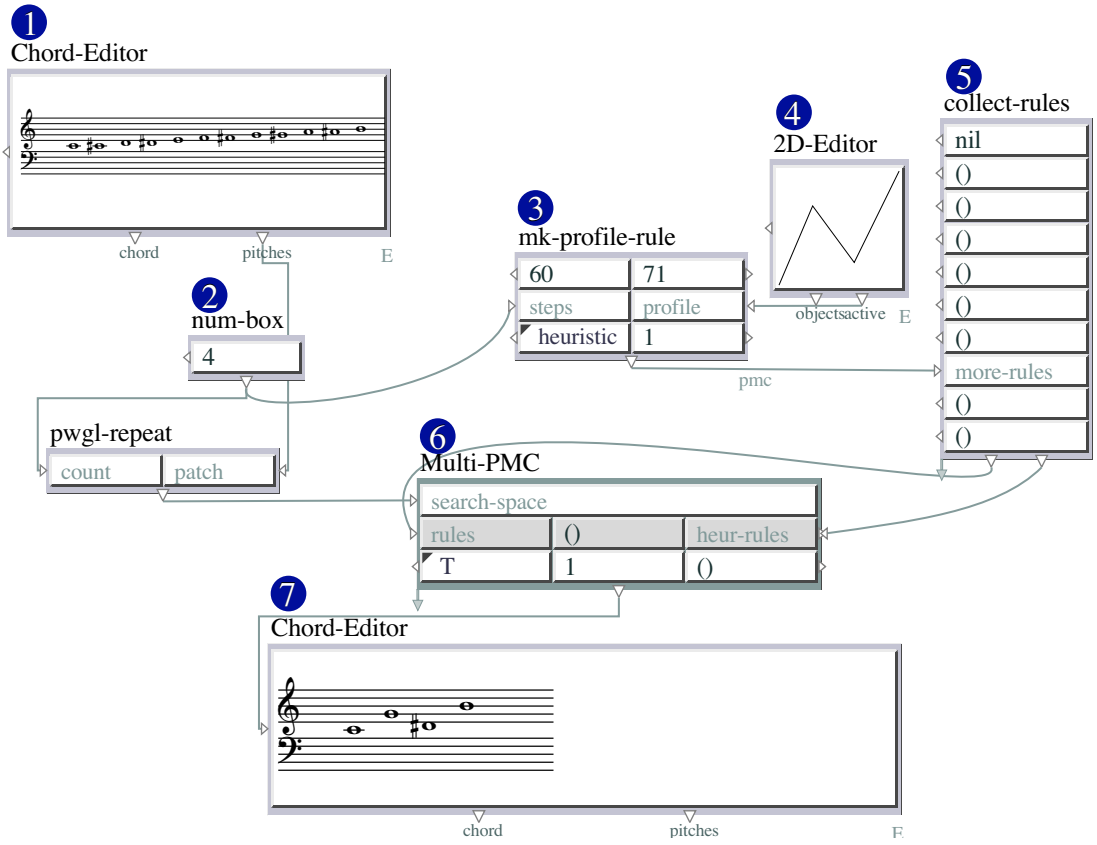

Figure 7: 1-00-6-Heuristic-rules-and-no-random-reserach

2.1.9 7-Heuristic-Rules-and-Weight

1.00.7 - HEURISTIC-RULES-AND-WEIGHTS
Now look at this patch. I have just added the NO-REPETITION-RULE [5] in the heuristic mode. Please give it a weight of 1.
The rest of the patch is equal to the previous (1.00.6). Evaluate the CHORD-EDITOR [8] and see what happens. The solutions you obtain are still not random but at the same time they try in the same time to avoid repetitions and to follow as much as possible the given profile.
Please, change the value of the weight [a] of the NO-REPETITION-RULE [5]. Try to put 1, then 2, then 3 and so on to 5. Take the time to evaluate many time the Multi-PMC with each of these values. You will see that from 1 to 4 the solutions change less and less. When the weight [a] of the NO-REPETITION-RULE is set to 5, the solution seems to be fixed. Between the two wishes (to have a solution as close as possible to the given profile [3], and not to have any repetition [5]) the weights, to keep the metaphor,

define which wish is stronger.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

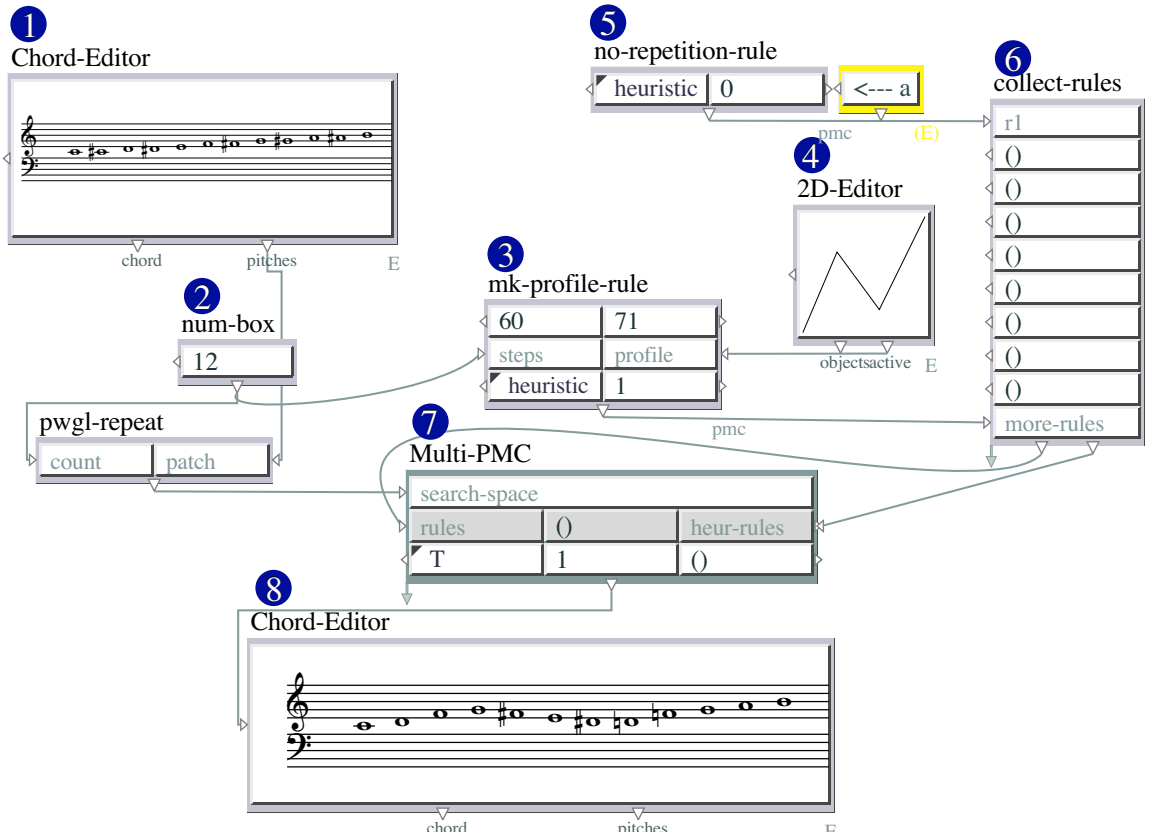

Figure 8: 1-00-7-Heuristic-rules-and-weight

2.1.10 8-Heuristic-Rules-and-Several-Weights

1.00.8 - HEURISTIC-RULES-AND-SEVERAL-WEIGHTS
Now look at this patch. The only difference from the previous (1.00.7) is the ALLOWED-INTERVALS-RULE [5]. This rules is set in the heuristic mode (as all the others) with a weight of 1 [a] (as the MK-FIX-PROFILE-RULE [3], and the NO-REPETITION-RULE [6]).
Evaluate many time the Multi-PMC and see that the results change a little. Then increase the weight [a] of the ALLOWED-INTERVALS-RULES [5]. Evaluate the Multi-PMC several times and look at the results. Then put the weight [a] of the ALLOWED-INTERVALS-RULES [5] to 1 and change the weight [b] of the NO-REPETITION-RULE [6]. Look again at the results.
Belonging to the weights you put the solutions are more and more close to a chosen

solution that try to balance different wishes. In this way no logical conflict is possible. In the heuristic mode, a solution is always possible.

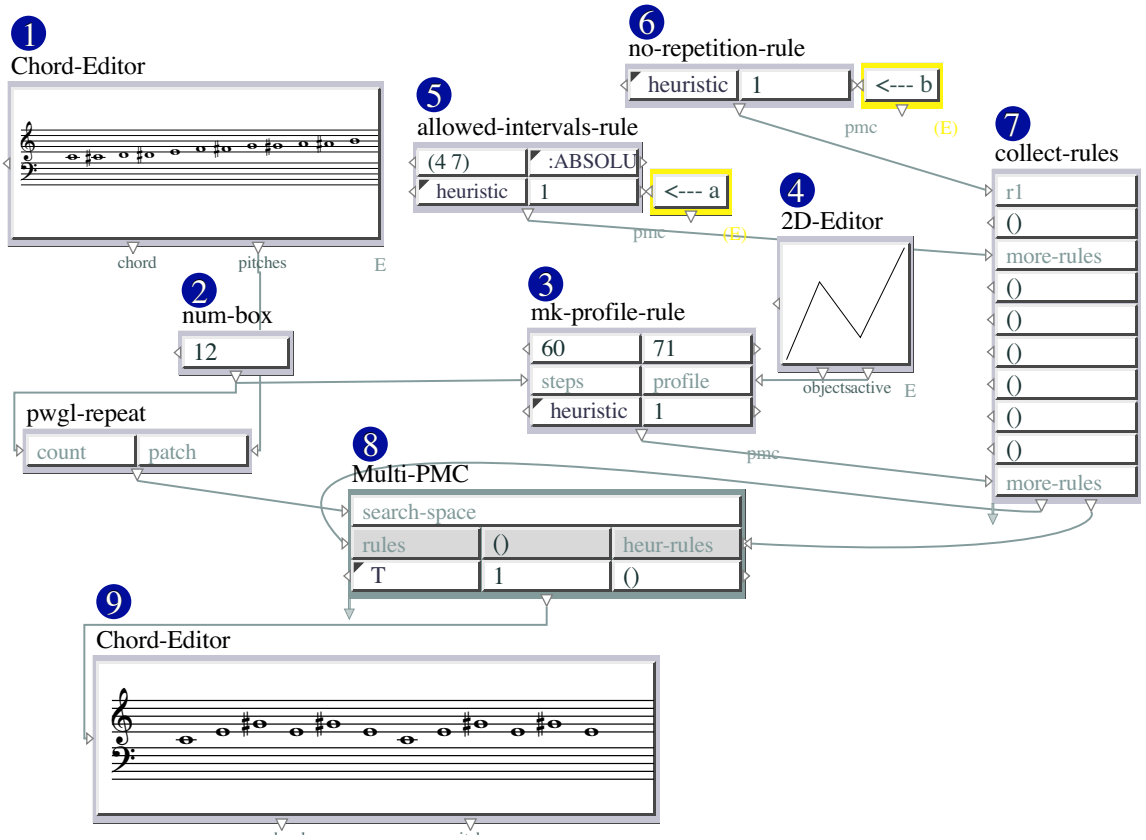

Figure 9: 1-00-8-Heuristic-rules-and-several-weights

2.1.11 9-Ergonomic-Disposition

1.00.9 - ERGONOMIC-DISPOSITION
When you start using the Multi-PMC very soon you will need to apply many rules at the same time. For this reason, and to better understand the following tutorials, we suggest this ergonomic disposition.
You can read it in the following way :
[1] is a fast way to generate a chromatic scale using the ARITHM-SER.
The MK-PITCH-CANDIDATES [2] is a tools to transpose quickly pitches into several octaves. The CHORD-EDITOR [3] it is just to see the range of the candidates.
The PWGL-REPEAT [4] creates as many groups of chromatic scales as the length of the solution you want to find. The NUM-BOX [5] needs to synchronize the length of the solution with some rules put inside the abstraction [6] containing the rules. Please open this abstraction and see how the rules are disposed.

This is just a suggestion dictated by experience. For beginners : use this disposition in order to read well what you are doing.

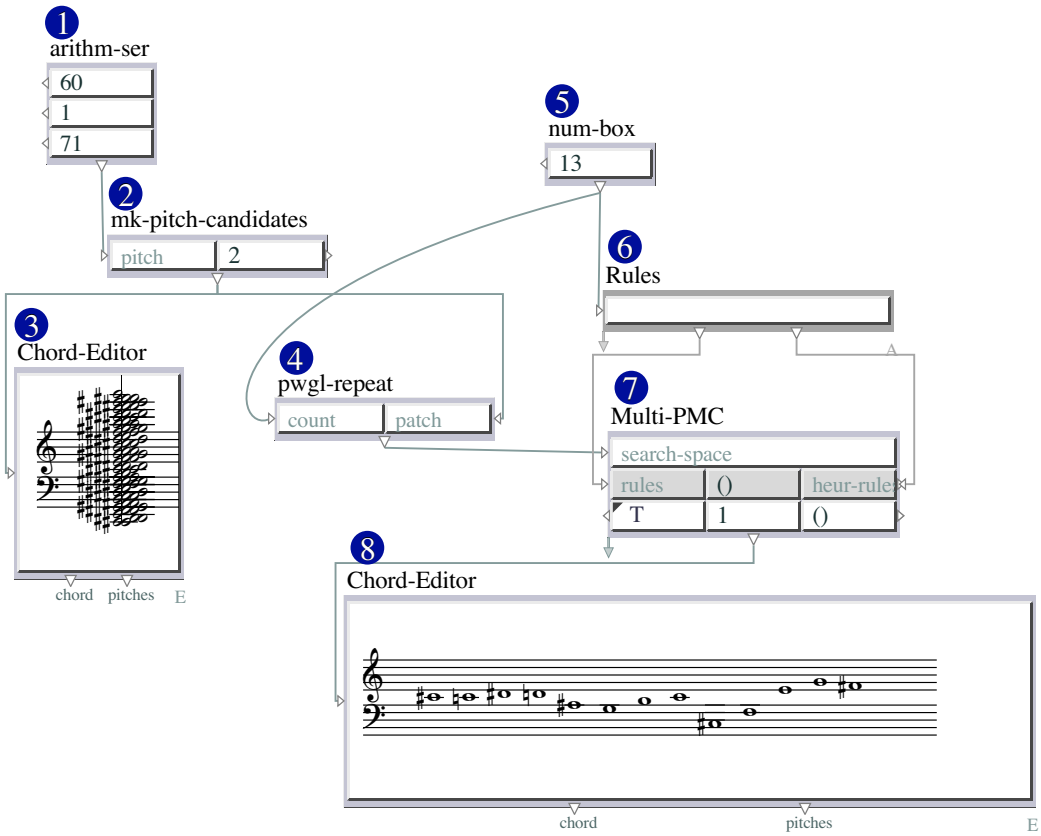

Figure 10: 1-00-9-Ergonomic-disposition

2.2 01-Generic-Rules

2.2.1 Generic-Rules

All the functions grouped here can be applied on any kind of parameters. That is why they are called generic.
Please note that often in this library you can find different rules, from different categories and menus, based on the same code or a really close one. These rules are redundant on purpose, in order to make easier your understanding of the musical concepts implemented.
You must understand that most of the rules described above are not necessarily dedicated to a single use. Most of them are generalizable to different musical parameters. The numerous patches in this tutorial only give you a partial sight of their possibilities.

2.2.2 01-Generic-Rules

1.01.01 - GENERIC-RULES
Each box [1] of this library has always at least two inputs. On [a] menu you can choose if you want to apply a rule in true/false mode or in heuristic one. The second is operational only when you are in the heuristic mode : it indicates the weight of application of the rule.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

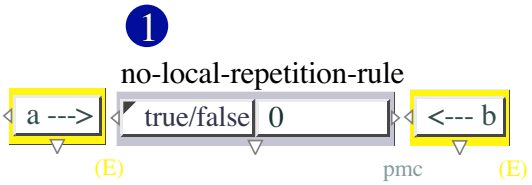

Figure 11: 1-01-01-generic-rules

2.2.3 02-Generic-Rules-with-Multi-PMC

1.01.02 - GENERIC-RULES-WITH-MULTI-PMC
The way to use all the rules [1] is to connect them obligatory to the COLLECT-RULES box you find in the Utils menu. Please connect always the left output [c] in the 'rules' input of the Multi-PMC and the right one [d] in the 'heur-rules' input of the same box.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of

weights to these rules.

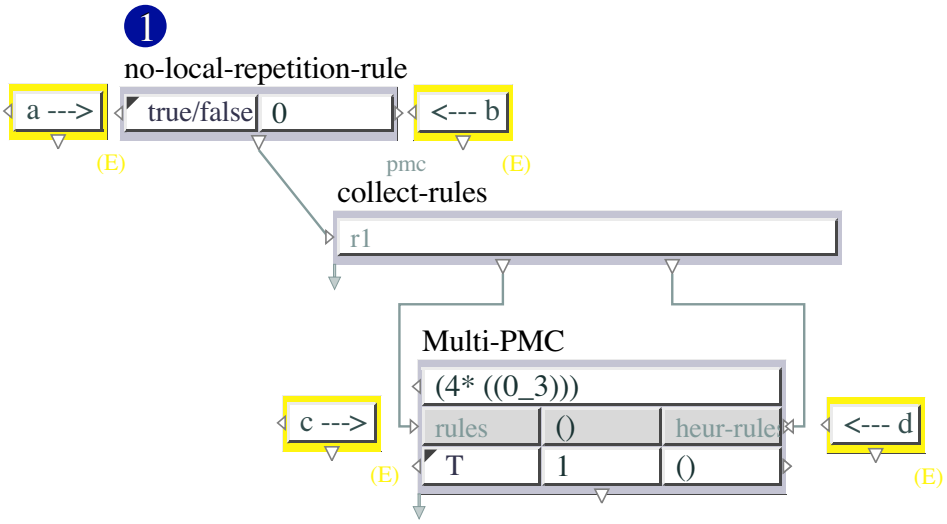

Figure 12: 1-01-02-generic-rules-with-multi-pmc

2.2.4 03-Generic-Rules-Candidates

1.01.03 - GENERIC-RULES-CANDIDATES
In [b] put a scale (the candidates) and duplicate it with the PWGL-REPEAT [c] box in order to define how many elements (the variables) will constitute the solutions. Evaluate [d] to look at the result in musical format.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules [1] are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

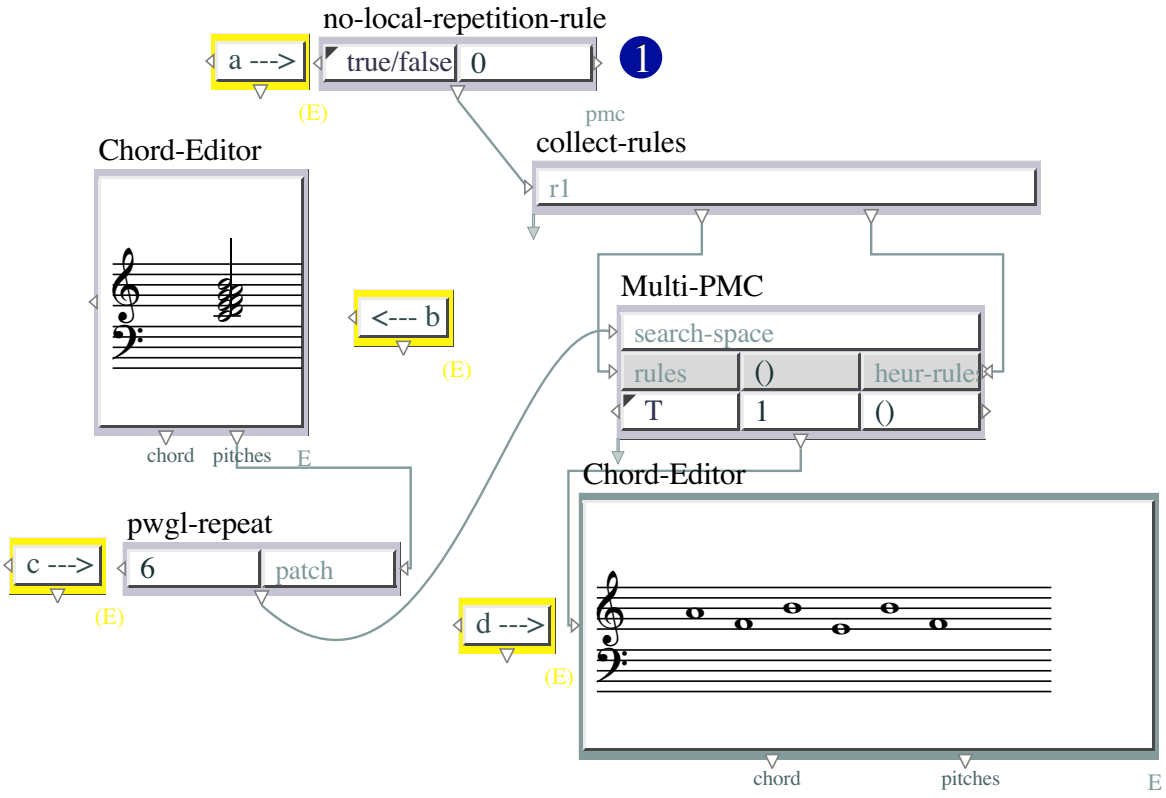

Figure 13: 1-01-03-generic-rules-candidates

2.2.5 04-Several-No-Repetitions-on-Durations

1.01.04 - SEVERAL-NO-REPETITIONS-ON-DURATIONS
This patch shows you how the functions NO-REPETITION, NO-LOCAL-REPETITION, NO-SPACED-REPETITION, etc. work.
In [a] you put, in this case, some ratio values of durations : remember that -1/4 is a quarter rest and 1/4 is a quarter note value. In [c] you use the PWGL-REPEAT to turn the candidates into several variables. For [d], please look at the tutorial of Kilian Sprotte in the library KSQuant. Evaluate [e] and look at the results.
Open the 'Rules' abstraction [b] :
Look at [f]. This is a SWITCH. If the first input is chosen the related rule is not applied. So, click on the right input [g] so the rule is operational.
[1], as we have already seen, is a rule forbidding repetition of any element all along a solution.
[2] forbids only repetitions for two consecutive elements.
[3] is pretty much like the contrary of NO-LOCAL-REPETITION. The 'candidates' input stands for a range between any element in the solution and another. For instance, "any element and the second after it" corresponds to the input (1 3). The rule makes sure

that any element in the solution won't be repeated by the second element after it (?3 after ?1).
[4] acts the same that NO-REPETITION-RULE, excepted that it forbids repetitions of absolute values. This is particularly useful here, because the rule will not make the difference between note values and rest values.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

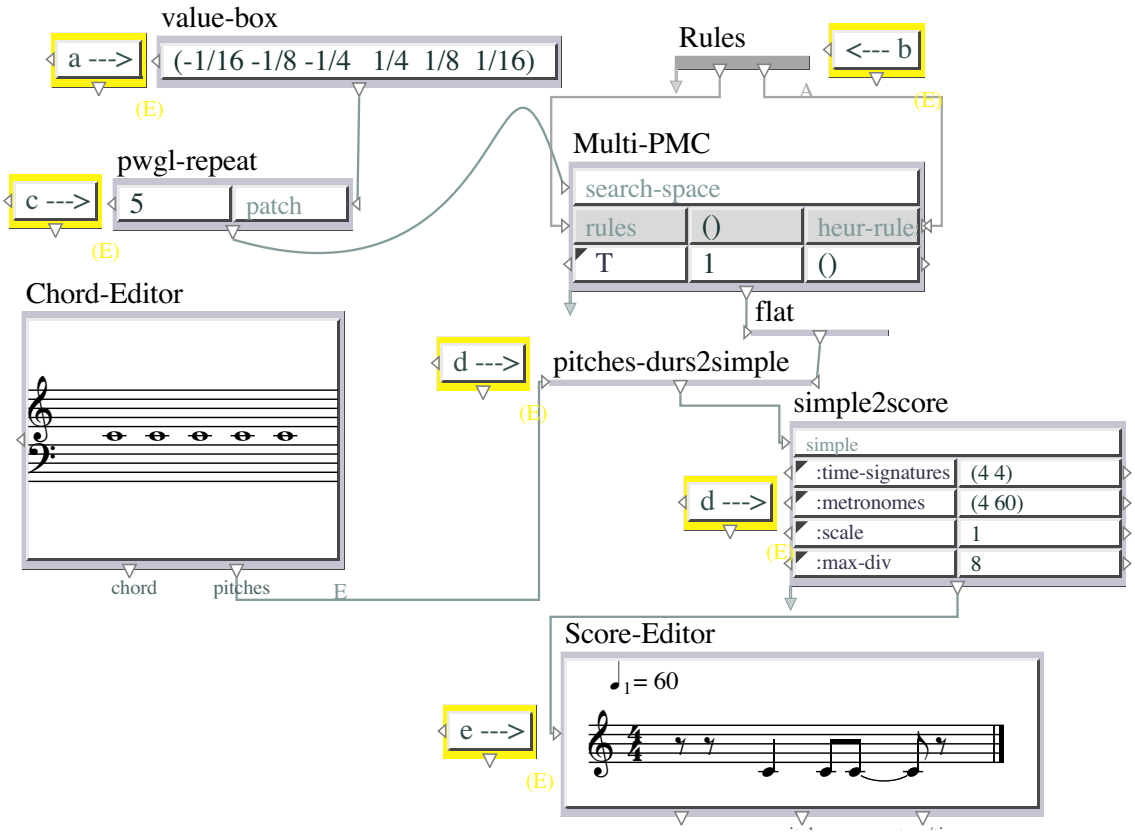

Figure 14: 1-01-04-several-no-repetitions-on-durations

2.2.6 05-Several-No-Repetitions-on-Intervals

1.01.05 - SEVERAL-NO-REPETITIONS-ON-INTERVALS
This patch shows you how the functions NO-REPETITION, NO-LOCAL-REPETITION, NO-SPACED-REPETITION, etc. work.
In [a] you put, in this case, some interval values : remember that -1 is a descending minor second and 2 is an ascending major second. In [c] you use the PWGL-REPEAT to turn the candidates into several variables. The DX->X function [d] rebuilds the sequence of notes from a starting point (60) with the sequence of intervals. Evaluate

[e] and look at the results.
Open the 'Rules' abstraction [b] :
Look at [f]. This is a SWITCH. If the first input is chosen the related rule is not applied. So, click on the right input [g] so the rule is operational.
[1], as we have already seen, is a rule forbidding repetition of any element all along a solution.
[2] forbids only repetitions for two consecutive elements.
[3] is pretty much like the contrary of NO-LOCAL-REPETITION. The 'candidates' input stands for a range between any element in the solution and another. For instance, "any element and the second after it" corresponds to the input (1 3). The rule makes sure that any element in the solution won't be repeated by the second element after it (?3 after ?1).
[4] acts the same that NO-REPETITION-RULE, excepted that it forbids repetitions of absolute values. This is particularly useful here, because the rule will not make the difference between ascending and descending intervals.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

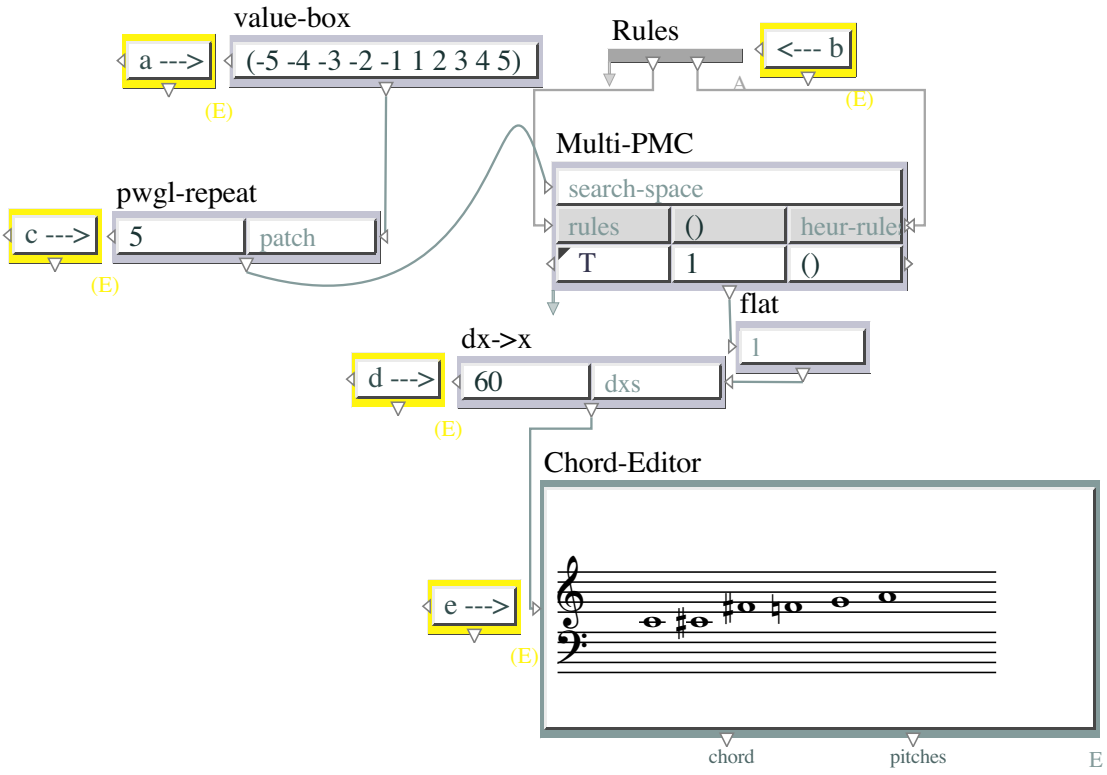

Figure 15: 1-01-05-several-no-repetitions-on-intervals

2.2.7 06-Modulo-X-Repetition

1.01.06 - MODULO-X-REPETITION-RULES
MODULO-X-REPETION-RULE [1] allows only repetitions of values having the modulo given in the 'modulo' input.
NOT-MODULO-X-REPETITION-RULE [2] forbids any repetition of a value having the modulo given in the 'modulo' input.
NOT-MODULO-X-LOCAL-REPETITION-RULE [3] does pretty much the same, except it forbids only consecutive repetitions of a modulo value.
Change the SWITCH in order to use or not the modulo-repetition-rule.
You can control the result by evaluating the G-MOD function [3] with the proper 'mod' [a]value. Every modulo value should be (normally...) turned into 0.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

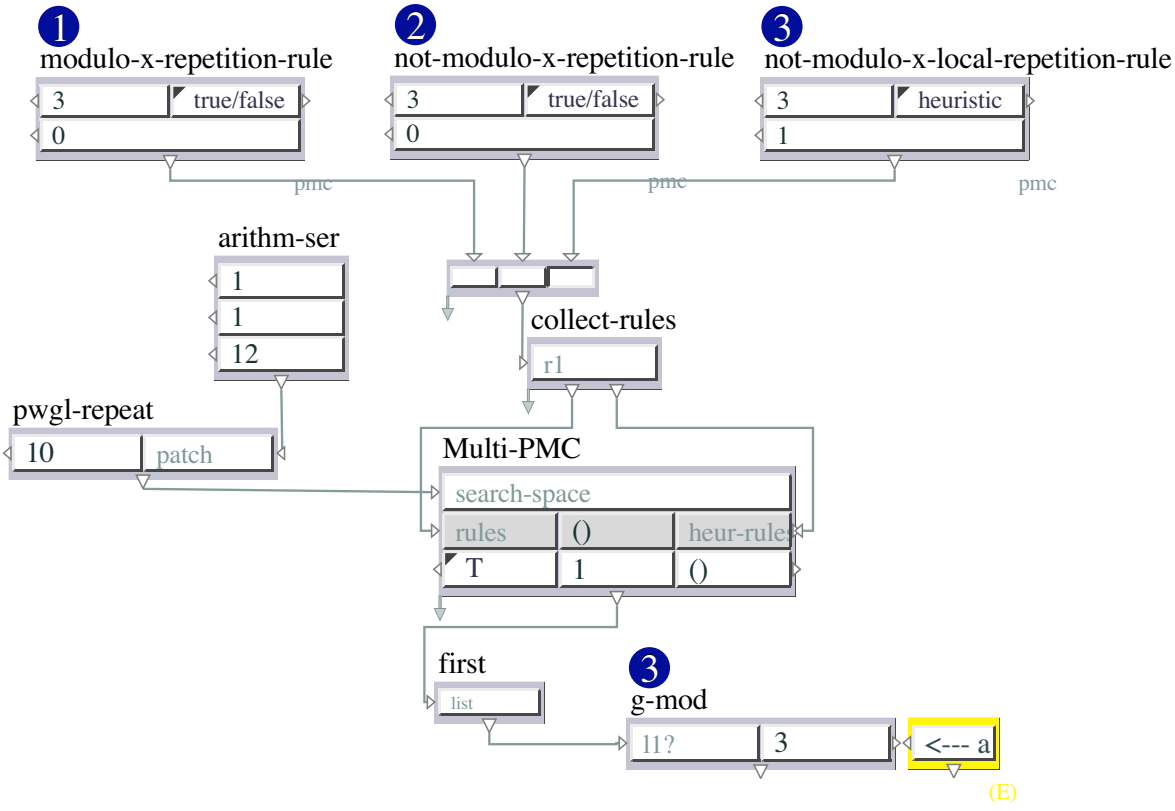

Figure 16: 1-01-06-modulo-x-repetition

2.2.8 07-Not-Consecutive-Rules

1.01.07 - NOT-CONSECUTIVE-RULES

This patch shows how to avoid some kinds of sequence. For instance:
NOT-CONSECUTIVE-NUMBER-RULE [1] does not allow any value to be followed by its consecutive value in the list of candidates. For instance, if the candidates are (1 2 3 4 5), the Multi-PMC cannot produce -1- followed by -2-. Sequences as -1- followed by -3-, -4- or -5- will be preferred.
NOT-CONSECUTIVE-ASCENDING-RULE [2] does not allow more ascending values than the value indicated in how-many.
NOT-CONSECUTIVE-EQUAL-RULE [3] does not allow more equal values than the values indicated in how-many.
NOT-CONSECUTIVE-DESCENDING-RULE [4] does not allow more descending values than the value indicated in how-many.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

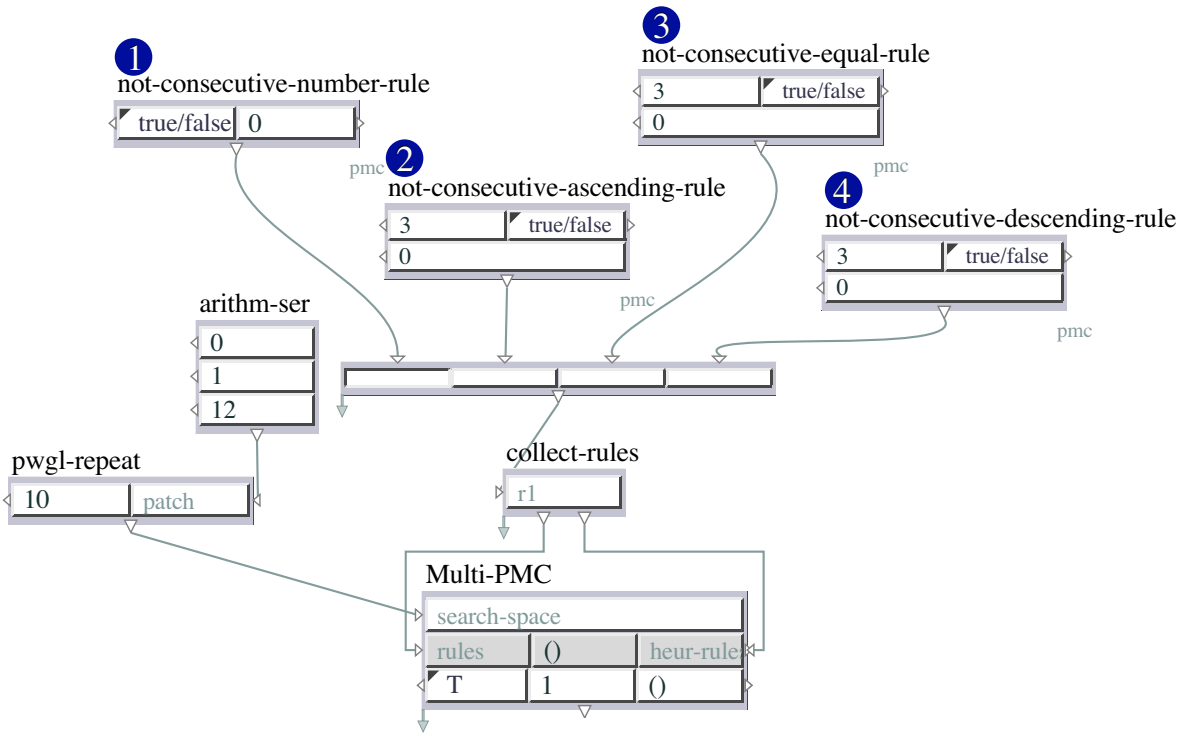

Figure 17: 1-01-07-not-consecutive-rules

2.2.9 08-Not-Repeated-Element-Sub-Group

1.01.08 - NOT-REPEATED-ELEMENT-SUB-GROUP

This rule [1] does not allow any repetition of an element into a given sub-group. The 'sub-group-length' input determines the length of the desired sub-group.
In [a] put the length of the sub-group. Now evaluate GROUP-LIST [b] and look at the result. Inside each sub-group of length 3 there is no repeated element.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

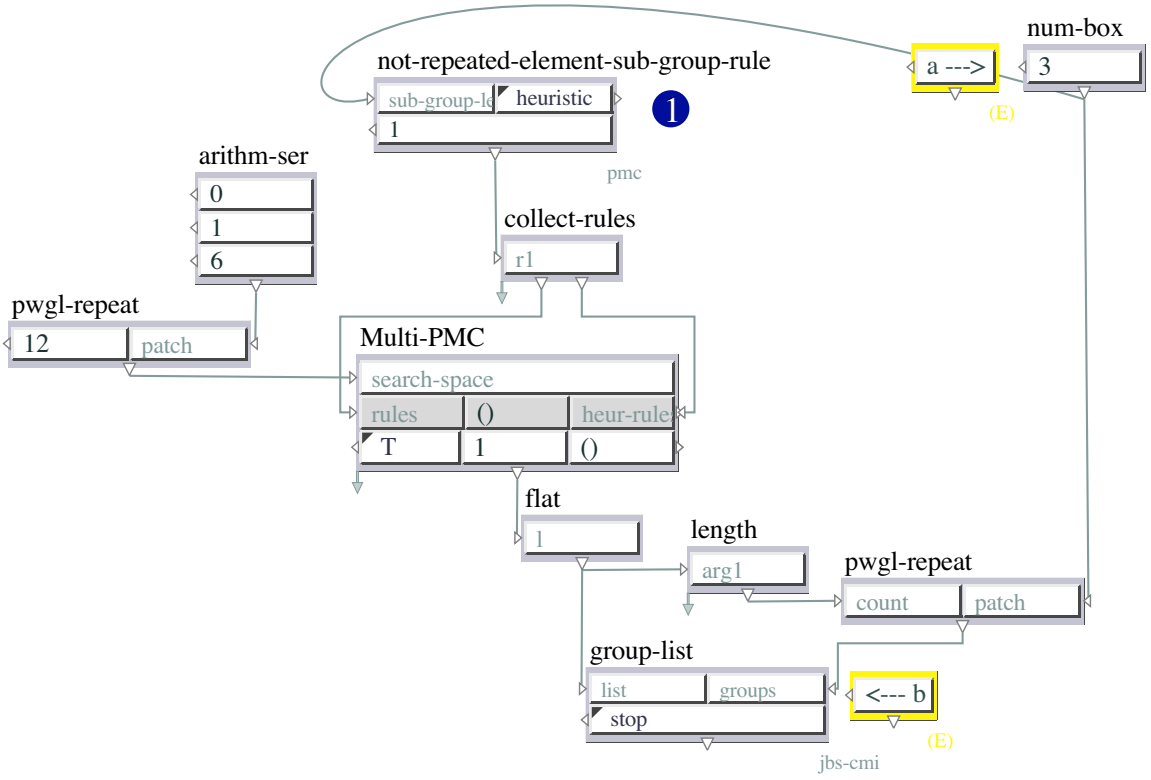

Figure 18: 1-01-08-not-repeated-element-sub-group

2.2.10 09-Not-Repeated-List-Sub-Group

1.01.09 - NOT-REPEATED-LIST-SUB-GROUP
NOT-REPEATED-LIST-SUB-GROUP-RULE [3] does not allow any repeated list into a given sub-group of a given length.
In [1] you define a list of lists as candidates. In [2] you set how many [a] elements have to be inside the solution.
Here we are : NOT-REPEATED-LIST-SUB-GROUP-RULE [3] needs to know the length of the sub-groups inside which you do not accept any repeated list. So in [b] we set 4 : that means that in any sub-group of four lists, there will be no repetition.

Please, evaluate the GROUP-LIST box [4] (Attention, this function comes from Profile library) and look at the results. In any group of length 4 there is no repeated list.

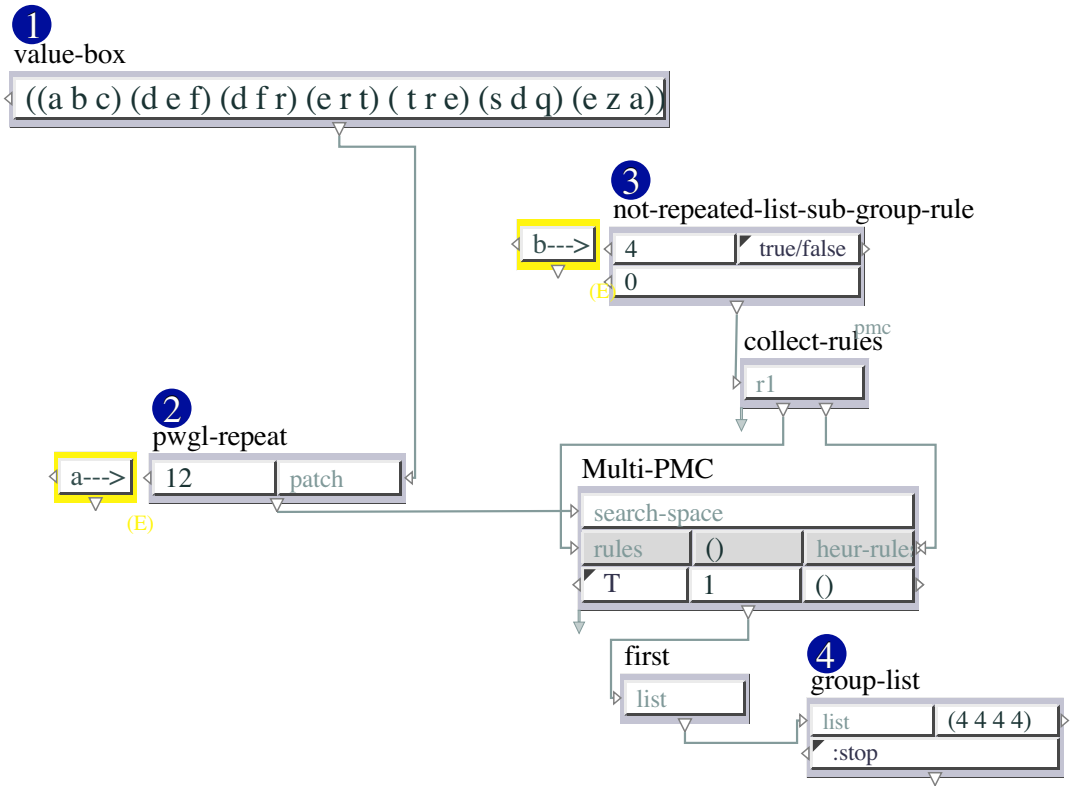

Figure 19: 1-01-09-not-repeated-list-sub-group

2.2.11 10-Item-Sub-Group-Member

1.01.10 - ITEM-SUB-GROUP-MEMBER
This rule [1] obliges the solution to be constituted by sub-groups (with length determined by 'sub-group-length') having in their index [a] position the possible elements put in [b].
Please evaluate the GROUP-LIST [c] and look at the result. At the beginning of each sub-group either a or b are allowed.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

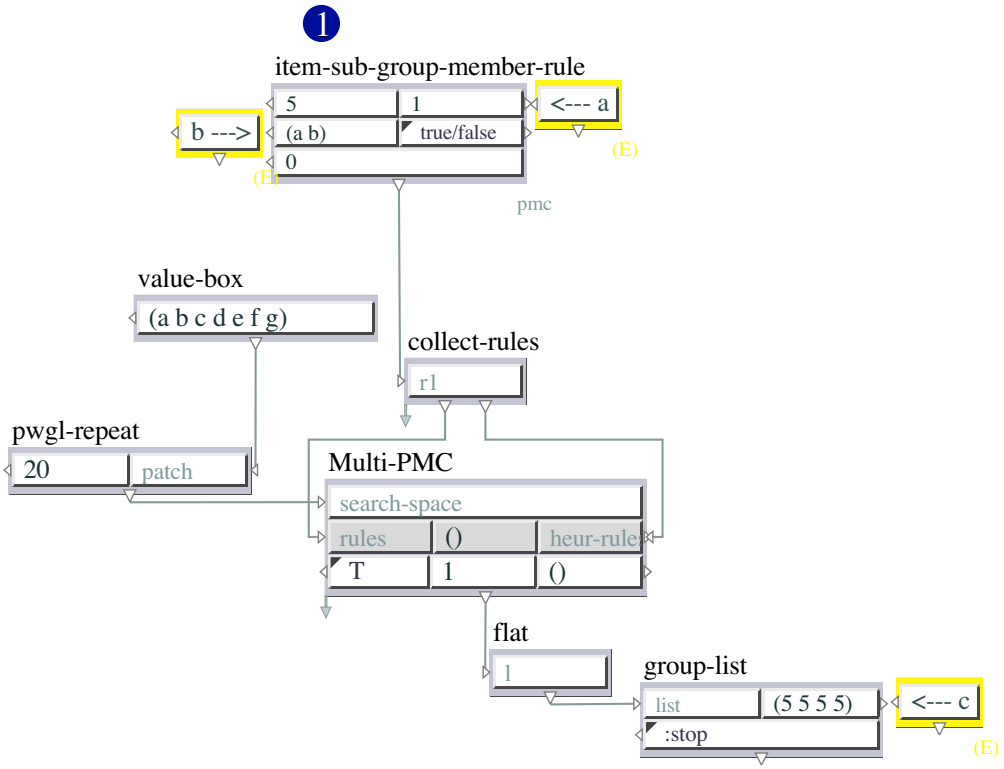

Figure 20: 1-01-10-item-sub-group-member

2.2.12 11-Allowed-Chain-Rules

1.01.11 - ALLOWED-CHAIN-RULES
ALLOWED-CHAIN-RULE [1] obliges the element set in [a] to be followed by one of the elements set in [b]. The elements in [b] are chosen randomly.
NOT-ALLOWED-CHAIN-RULE [2] does the opposite.
Evaluate the CHORD-EDITOR [c] and see how the first C note is followed either by a C-sharp (midi note 61) note or by a B (midi note 71).
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

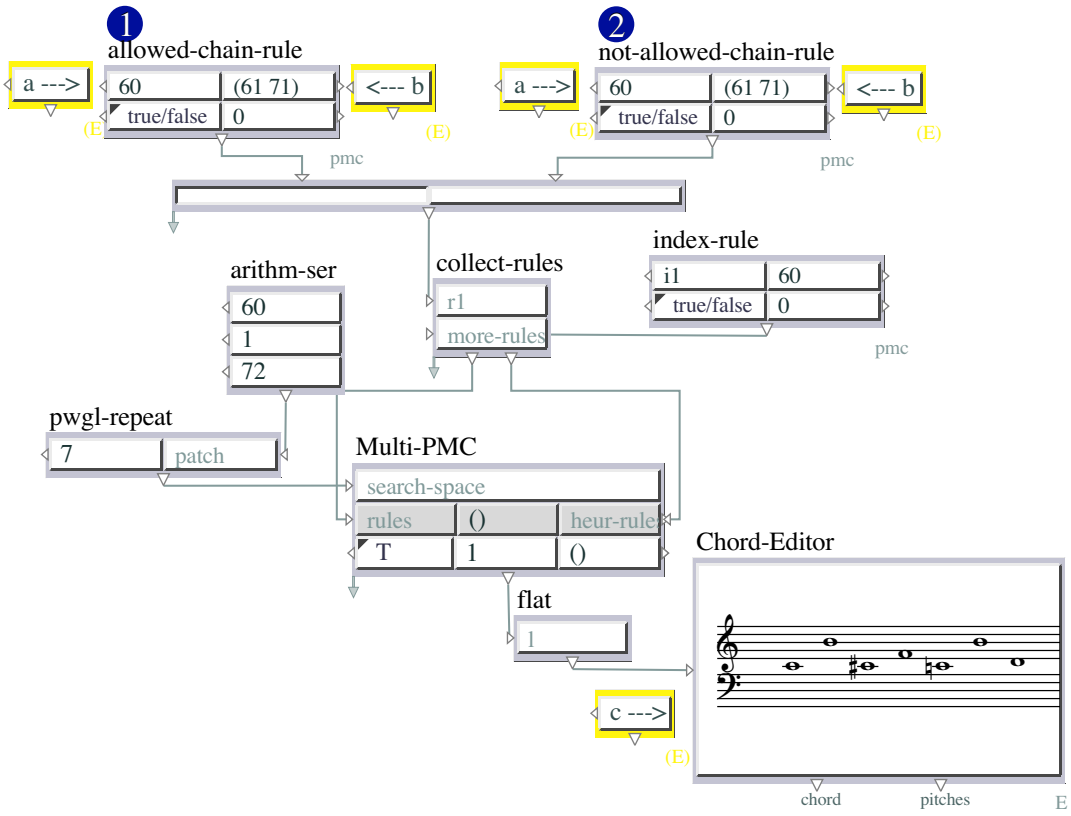

Figure 21: 1-01-11-allowed-chain-rules

2.2.13 12-Length-Rules

1.01.12 - LENGTH-RULES
Here are three different rules controlling the length of sub-lists belonging to a solution. Please open the abstraction 'Candidates' [a]. It generates 16 [d] random groups having random lengths with random notes. (Please keep it locked in order to obtain a result for each top-evaluation.)
Select with the PWGL-SWITCH [b] which rule you want to apply.
[1] NOT-CONSECUTIVE-EQUAL-LENGTH-RULE With this rule, the length of two consecutive groups cannot be equal. It acts pretty much like the NO-LOCAL-REPETITION-RULE.
[2] NOT-LENGTH-REPETITION-RULE With this rule, no repetition of sub-group equal length is allowed in the whole solution. This one could be compared with the NO-REPETITION-RULE.
[3] LENGTH-SUB-GROUP-RULE This rule obliges the sub-solutions to have a length accordingly to the list put in lengths. The list put in length has to be a bpf in the format 2D-EDITOR.

You can control the results by evaluating both the Score-Editor or the 'g-length' abstraction. [4]
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

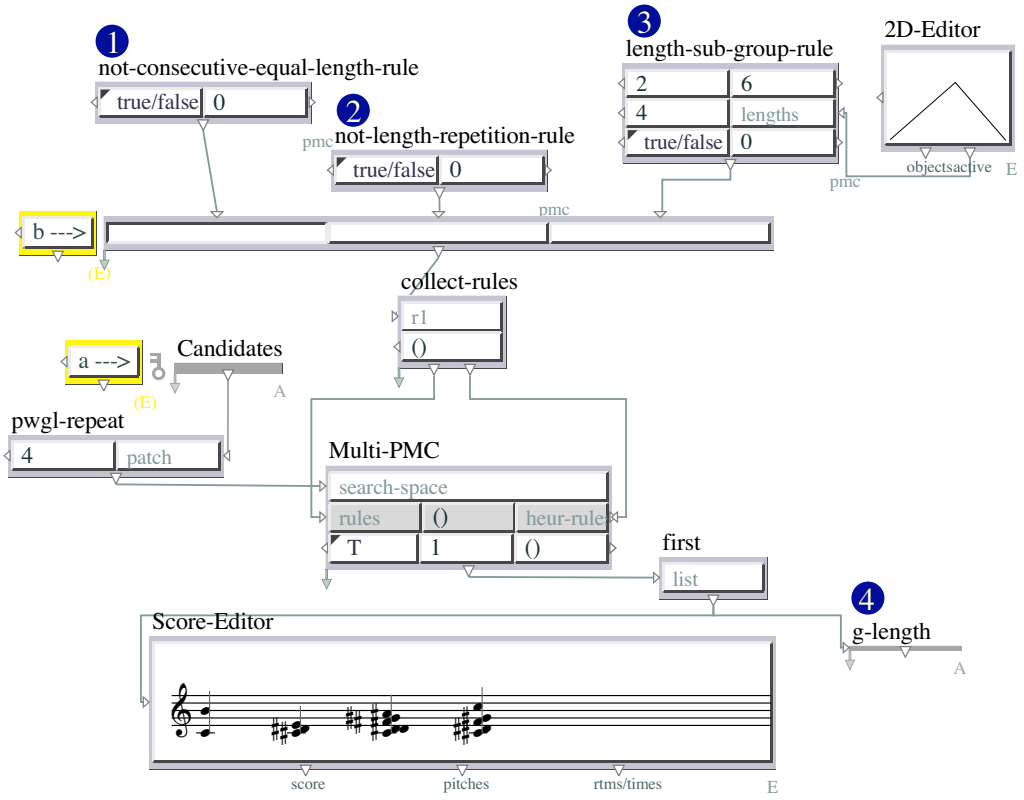

Figure 22: 1-01-12-length-rules

2.2.14 13-Several-Index-Rules

1.01.13 - SEVERAL-INDEX-RULES
This patch shows you four rules for indexes. Select with [a] which rule you want to apply, and then evaluate the FIRST box [5] to see the results.
INDEX-RULE [1] This rule obliges a variable in the solution (indicated by it index position) to have the value indicated in its right input.
NOT-INDEX-RULE [2] This rule obliges a variable of the solution (indicated by its index position) NOT to have the value indicated in its right input.
INDEX-HIGHER-RULE [3] This rule obliges a variable of the solution (indicated by its index position) to have a value higher than the one indicated in its right input.

INDEX-LOWER-RULE [4] This rule obliges a variable of the solution (indicated by its index position) to have a value lower than the one indicated in its right input.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

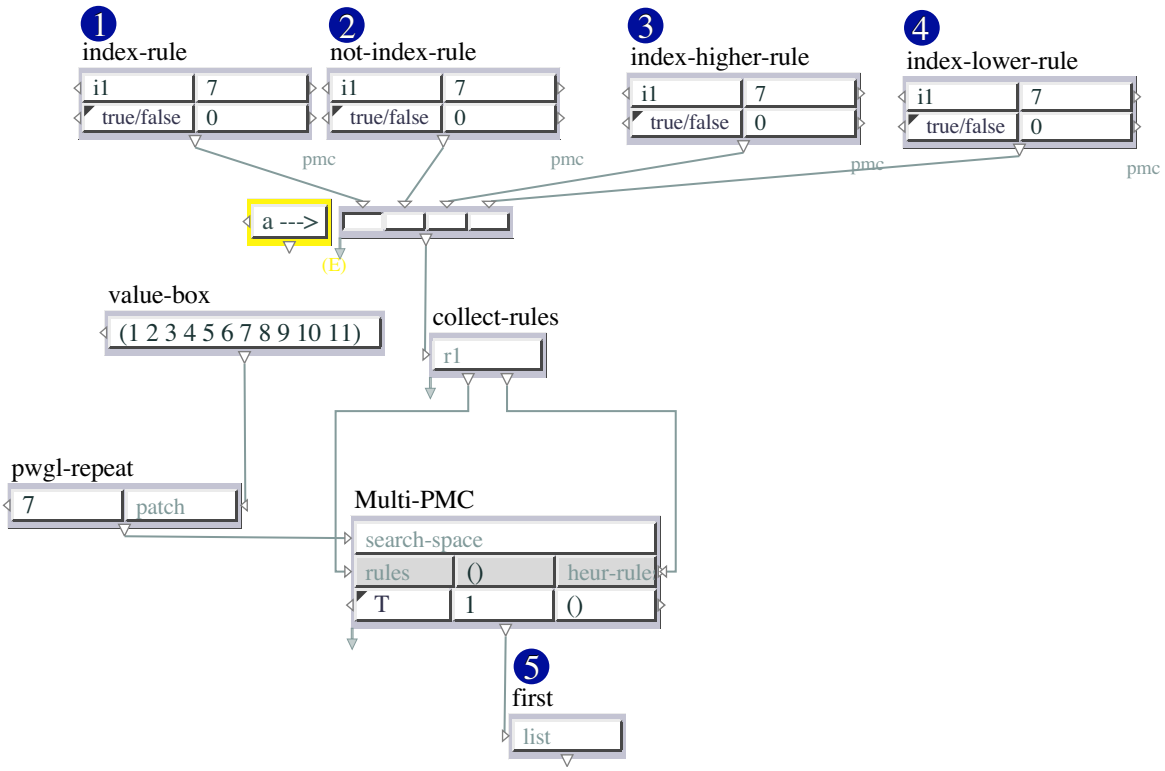

Figure 23: 1-01-13-several-index-rules

2.2.15 14-Index-Length-Rule

1.01.14- INDEX-LENGTH-RULE
This rule [1] obliges to have as a solution a list of lists in which the length of the list indicated by its [a] index has the length set in [b].
You can control the results by evaluating both the Score-Editor or the 'g-length' abstraction. [c]
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

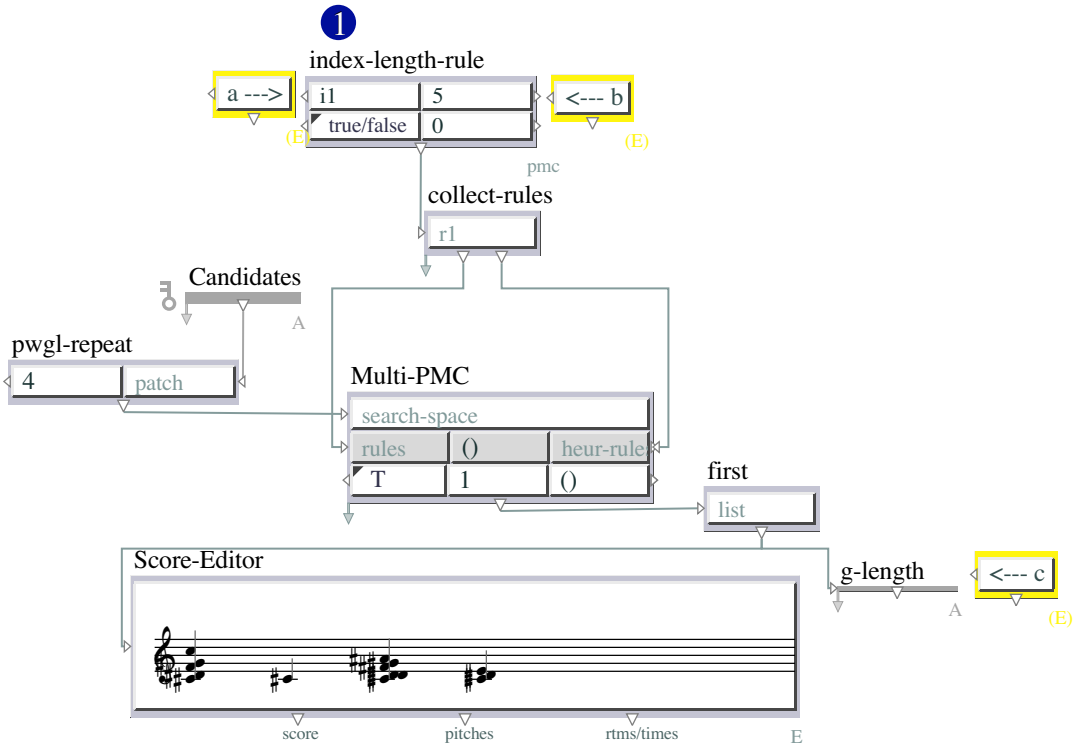

Figure 24: 1-01-14-index-length-rule

2.2.16 15-Index-Nth-Rule

1.01.15 - INDEX-NTH-RULE
This rule [1] obliges the nth element (indicated in [b] from 0 to n) of the list indicated by its index [a] to have the value put in [c].
Do not forget that the list representing a chord is not necessarily ordered from the lowest to the highest note.
Please, if you have difficulties to understand this rule, keep in mind that the first nth element is 0, and try evaluating the FIRST box [d] for a better reading of the results.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

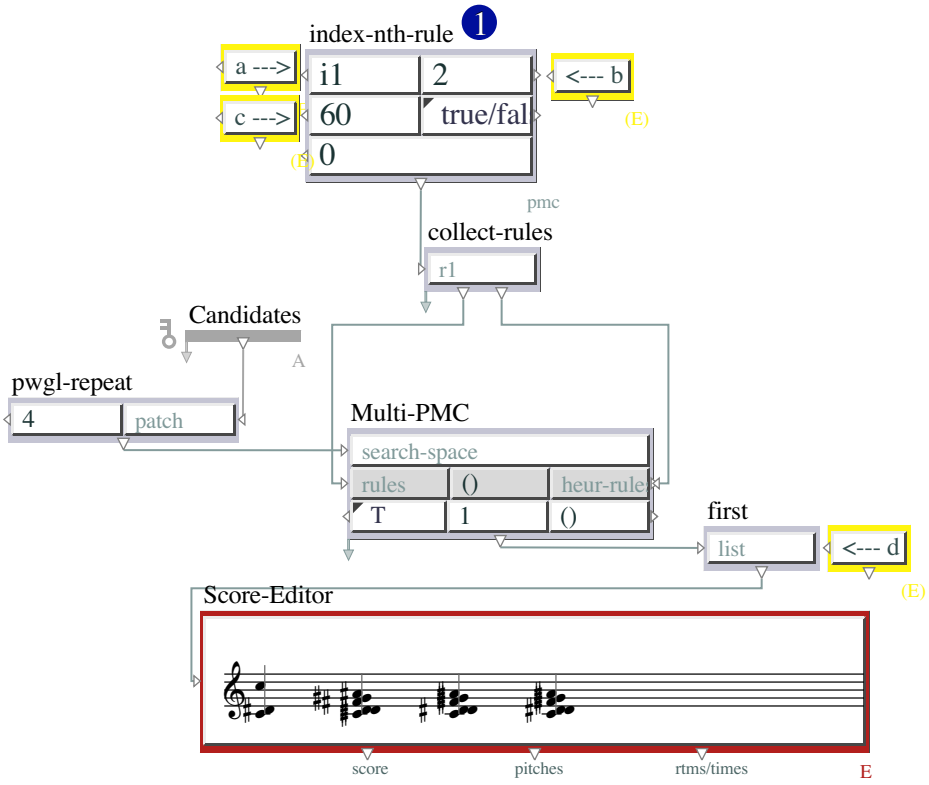

Figure 25: 1-01-15-index-nth-rule

2.2.17 16-Index-Applied-Sum-Rule

1.01.16 - INDEX-APPLIED-SUM-RULE
This rule [1] obliges the variable indicated by 'index' [a] to have its elements, summed together, give back the number put in the 'sum' input [b].
Here the candidates are pretty limited, so the Multi-PMC may not produce results for a certain amount of values in 'sum'. [b]
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

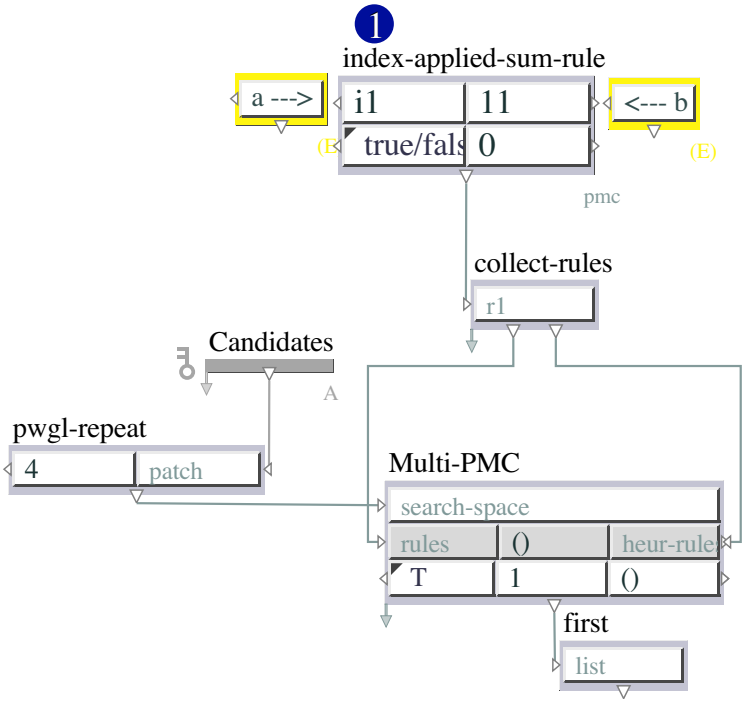

Figure 26: 1-01-16-index-applied-sum-rule

2.2.18 17-Member-Rules

1.01.17 - MEMBER-RULES
MEMBER-RULE [1] obliges any element of the solution to belong to the elements indicated in 'list' input [a].
NOT-MEMBER-RULE [2] does the opposite.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

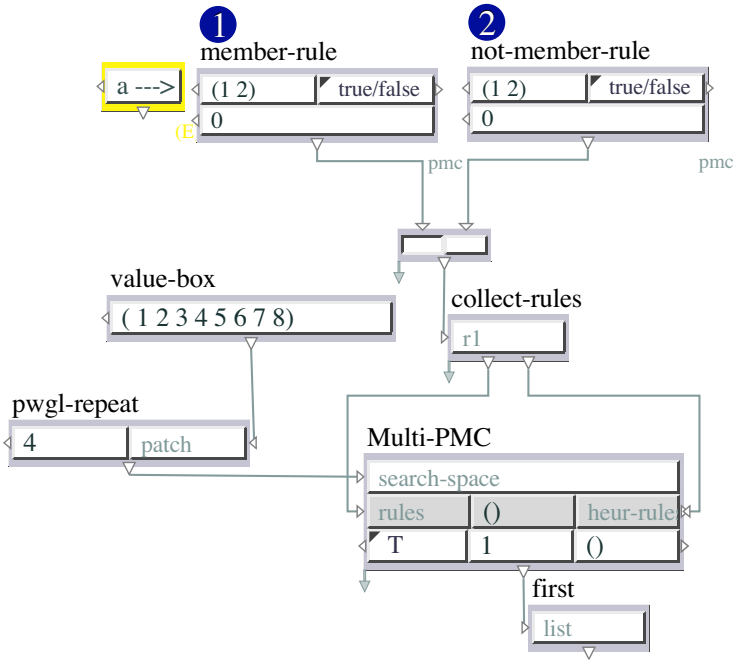

Figure 27: 1-01-17-member-rules

2.2.19 18-Not-Higher-or-Lower-than-Rules

1.01.18 - NOT-HIGHER-OR-LOWER-THAN-RULES
NOT-HIGHER-THAN-RULE [1] forbids any element of the solution to be higher than the value indicated in the 'limit' input [a].
NOT-LOWER-THAN-RULE [2] does the opposite.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

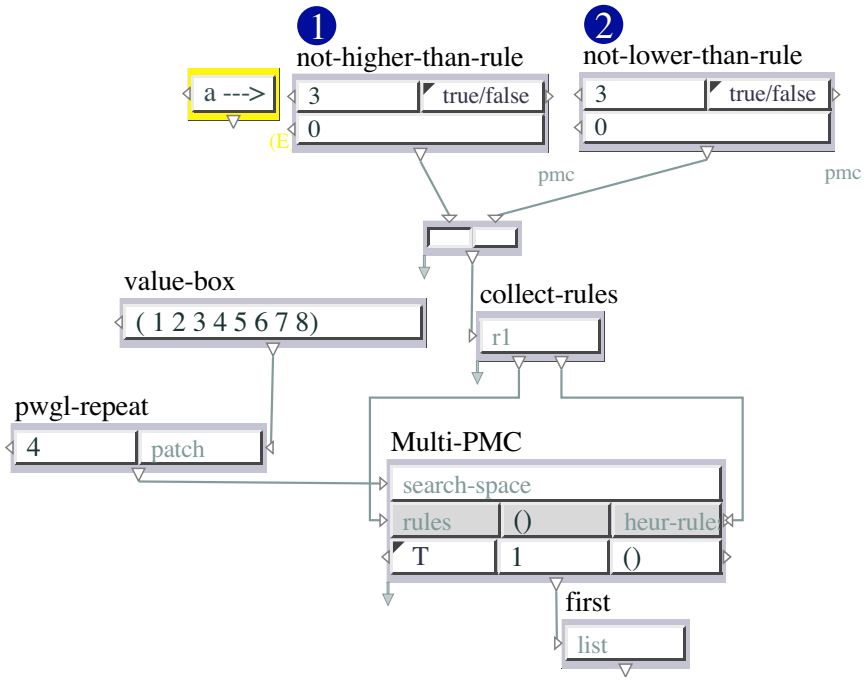

Figure 28: 1-01-18-not-higher-or-lower-than-rules

2.2.20 19-Count-Common-Elements-Rule

1.01.19 - COUNT-COMMON-ELEMENTS-RULE
This rule [1] obliges the sub-lists in the solution to have a desired number [a] of common elements. These common elements are not necessarily constant.
The abstraction [b] generates all sub-lists of length 4 with numbers from 1 to 9.
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

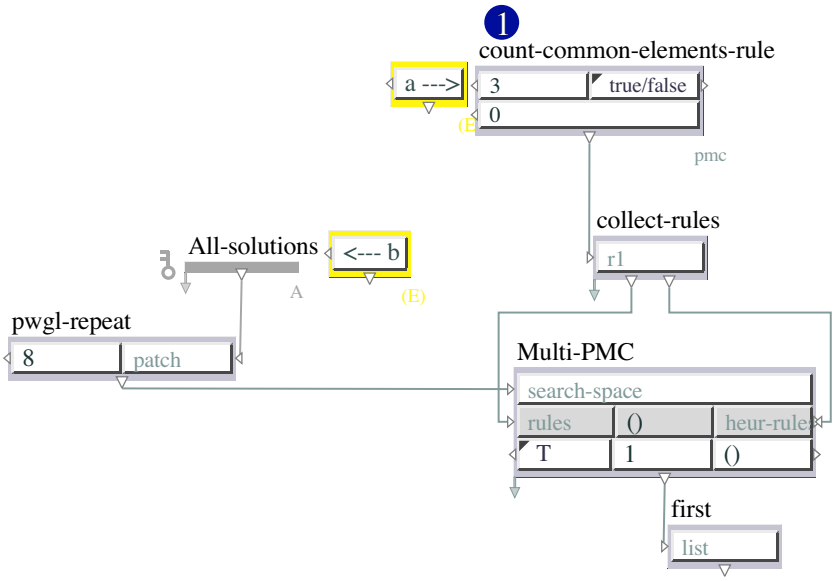

Figure 29: 1-01-19-count-common-elements-rule

2.2.21 20-Count-Any-Element-Rule

1.01.20 - COUNT-ANY-ELEMENT-RULE
This rule [1] obliges the solution to have any element repeated as many times as indicated in [a].
PLEASE BE CAREFUL Often in these tutorials the heuristic rules are initialized on purpose with a weight of 0. This was done in order to get used with the attribution of weights to these rules.

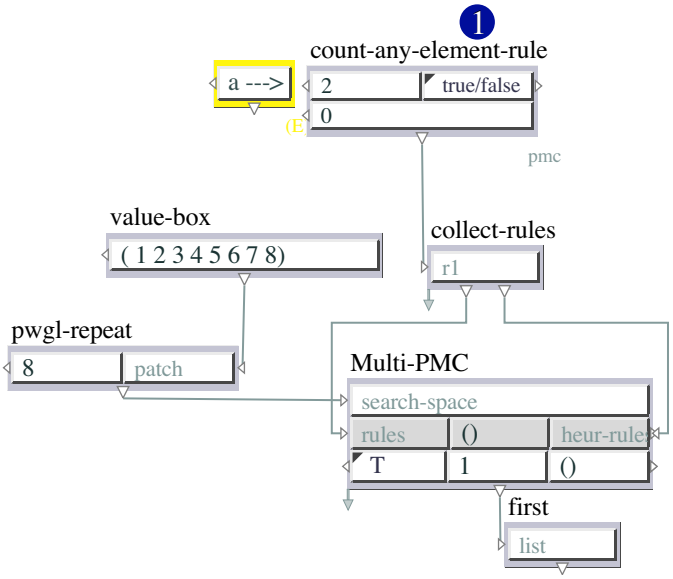

Figure 30: 1-01-20-count-any-element-rule

2.3 02-Interval-Rules

2.3.1 Interval-Rules

The following rules are conceived for intervals. That means that, as an interval is the distance between two points, these rules can be applied to any concept invoking the notion of distance like durations, melodic intervals or distance between intensities and so on.
Some of these rules have a particular menu called absolute? or up/down. If the menu absolute? is set in the absolute mode, that means that the intervals are intended in absolute mode (always positive). If this menu is set in up/down mode, that means that the intervals are divided into ascending (positive) and descending (negative).

2.3.2 01-Several-Interval-No-Repetitions

1.02.01 - SEVERAL-INTERVAL-NO-REPETITIONS
This patch shows three rules to forbid some interval successions.

NO-INTERVAL-LOCAL-REPETITION-RULE [1] This rule does not allow any local repetition of intervals. That means, for instance, that an ascending minor third can never be followed by another ascending minor third (in up/down mode) or any minor third (in absolute mode).
NO-INTERVAL-REPETITION-RULE [2] This rule does not allow any repetition of intervals.
If the menu absolute? is set on absolute, that means that intervals are considered in absolute mode. If this menu is set in the up/down mode, that means that the intervals are divided into ascending and descending. In this second case for instance a minor third going up can be followed by a minor third going down.
NO-LOCALLY-REPEATED-GIVEN-INTERVAL-RULE [3] This rule obliges a solution not to have a given interval [a] locally repeated. It looks like NO-INTERVAL-LOCAL-REPETITION-RULE but is limited to the given interval.
If the menu absolute? is set on absolute, that means that intervals are considered in absolute mode. If this menu is set in the up/down mode, that means that the intervals are divided into ascending and descending. In this second case for instance a minor third going up can be followed by a minor third going down.
Please try these rules by evaluating both the CHORD-EDITOR or the X->DX [4] function, if you need a better reading of intervals changes.

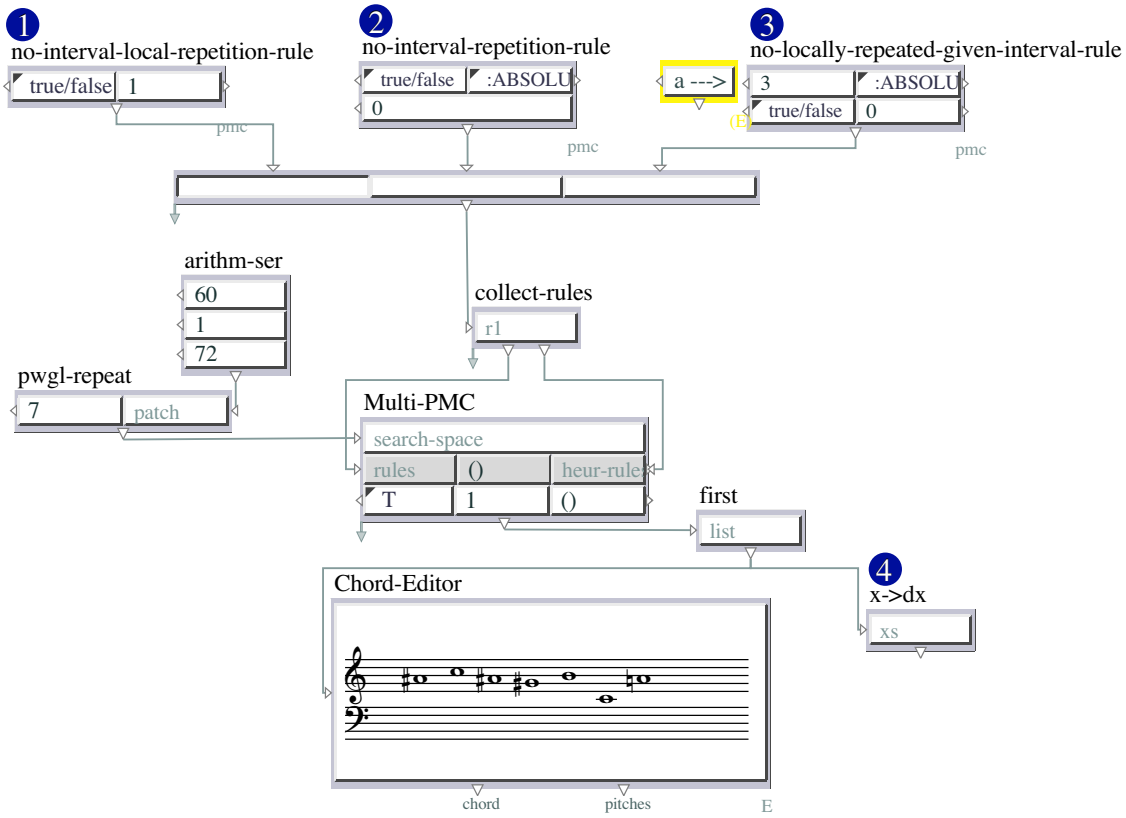

Figure 31: 1-02-01-several-interval-no-repetitions

2.3.3 02-Several-Allowed-or-Not-Interval-Rules

1.02.02 - SEVERAL-ALLOWED-OR-NOT-INTERVAL-RULES
In this patch there are three rules allowing or not some given intervals.
ALLOWED-INTERVALS-RULE [1] This rule allows only the intervals indicated in intervals.
If the menu absolute?' is set in the absolute mode, that means that intervals are intended in absolute mode. If this menu is set in up/down mode, that means that the intervals are divided into ascending and descending. So if you put -3 in the up/down mode that means that only descending minor third are admitted.
NOT-ALLOWED-INTERVALS-RULE [2] This rule does not allow the intervals indicated in intervals.
If the menu absolute?' is set in the absolute mode, that means that intervals are intended in absolute mode. If this menu is set in up/down mode, that means that the intervals are divided into ascending and descending. So if you put -3 in the up/down mode that means that only descending minor third are not admitted.
ALLOWED-DISTANT-INTERVALS-RULE [3] This rule allows a sequence of intervals to equal, within a given distance, one of the allowed intervals in the input 'intervals'. The

distance has to be described giving the first and the last note of the distance.
For instance, if in 'distance' you put 1 and 3 it means that the sum of the intervals between the first and the third notes of the sequence has to be a member of 'intervals'. Please remember that here the distance is expressed not in term of intervals but in term of notes.
Please try these rules by evaluating both the CHORD-EDITOR or the X->DX [4] function, if you need a better reading of intervals changes.

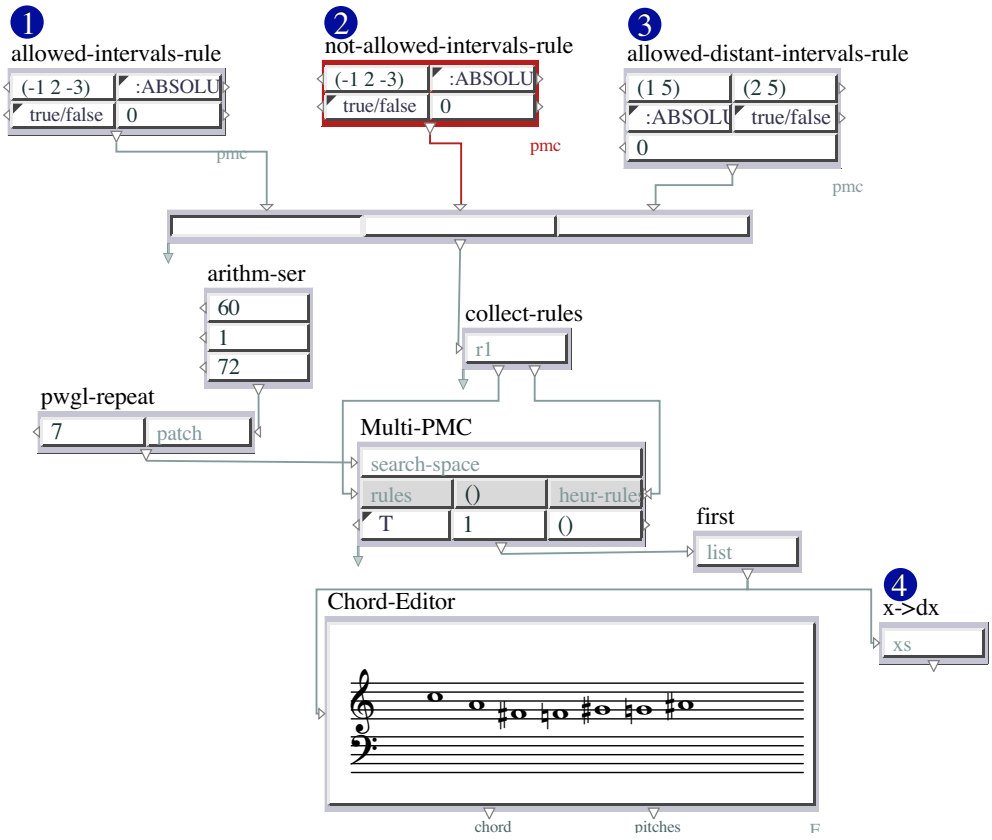

Figure 32: 1-02-02-several-allowed-or-not-interval-rules

2.3.4 03-No-Consecutive-Equal-Interval-Rules

1.02.03 - NO-CONSECUTIVE-EQUAL-INTERVALS-RULE
This rule [1] does not allow any repetition of intervals for a length put in how-many. If the menu absolute? is set in the absolute mode, that means that intervals are intended in absolute mode. If this menu is set in up/down mode, that means that the intervals are divided into ascending and descending. So if you put 3, in the how-many input, that means that any interval can not be repeat consecutively for more than two times.

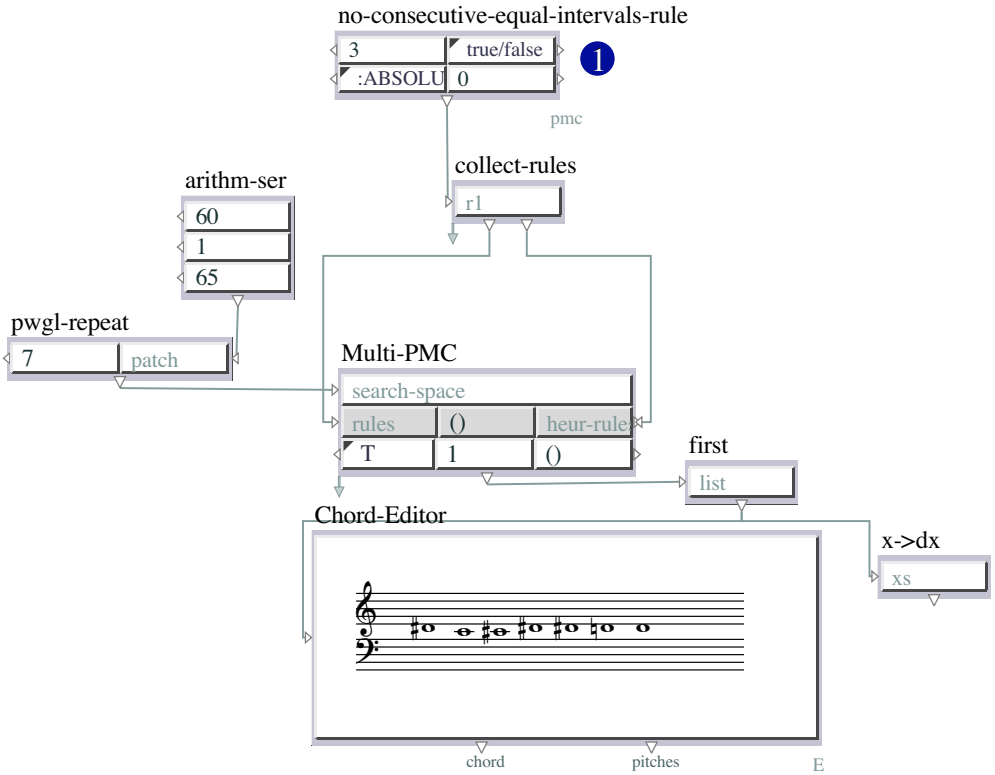

Figure 33: 1-02-03-no-consecutive-equal-interval-rules

2.3.5 04-Obliged-or-Not-Interval-Chain

1.02.04 - OBLIGED-OR-NOT-INTERVAL-CHAIN-RULE
OBLIGED-INTERVAL-CHAIN-RULE [1] This rule obliges an interval to be followed by those put in int-list.
If the menu absolute?' is set in the absolute mode, that means that intervals are intended in absolute mode. If this menu is set in up/down mode, that means that the intervals are divided into ascending and descending.
NOT-OBLIGED-INTERVAL-CHAIN-RULE [2] This rule obliges an interval NOT to be followed by those put in int-list.
If the menu absolute?' is set in the absolute mode, that means that intervals are intended in absolute mode. If this menu is set in up/down mode, that means that the intervals are divided into ascending and descending.

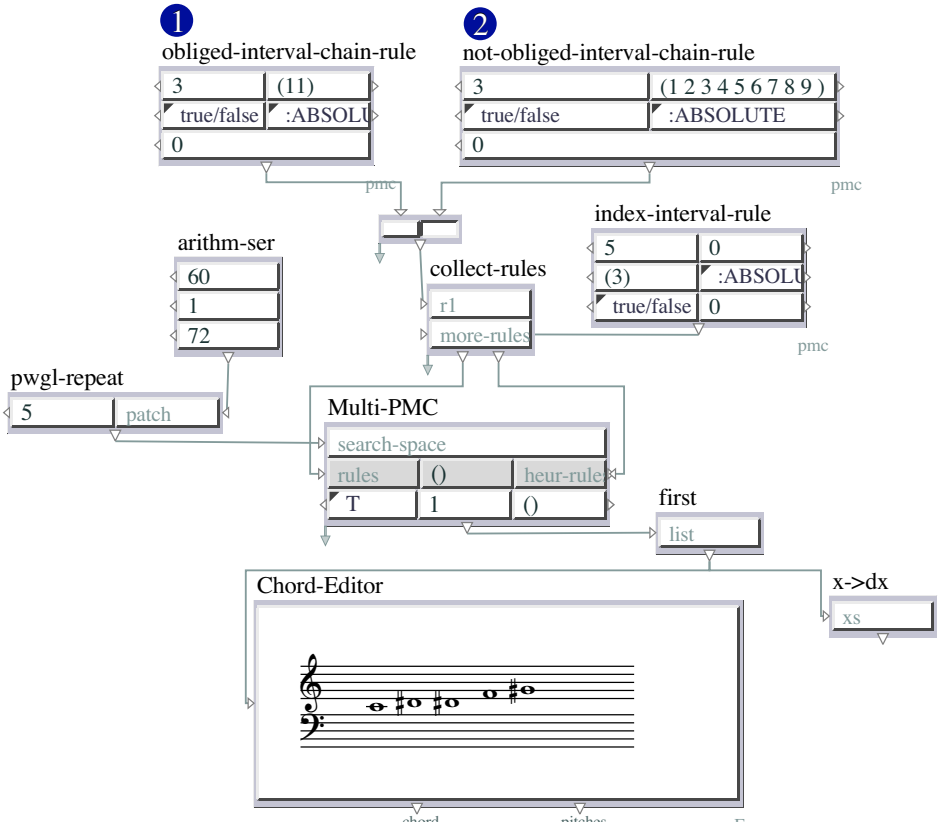

Figure 34: 1-02-04-obliged-or-not-interval-chain

2.3.6 05-Repeat-Interval

1.02.05 - REPEAT-INTERVAL-RULE
REPEAT-INTERVAL-RULE [1] obliges a solution to have a given interval [a]repeated many times as indicated in ”times” [b].
The interval is considered in the absolute mode but as you can see, in this case, the ASCENDING-WITHOUT-REPETITION-RULE [2] forbids descending intervals. You can bypass it with the SWITCH [c]
If the menu which? is set on $<$, it means that the given interval has to be repeated a number of time inferior to the one put in times. If the menu which? is set on $=$, it means that the given interval has to be repeated a number of time equal to the one put in times. If the menu which? is set on $>$, it means that the given interval has to be repeated a number of times bigger than the one put in times.

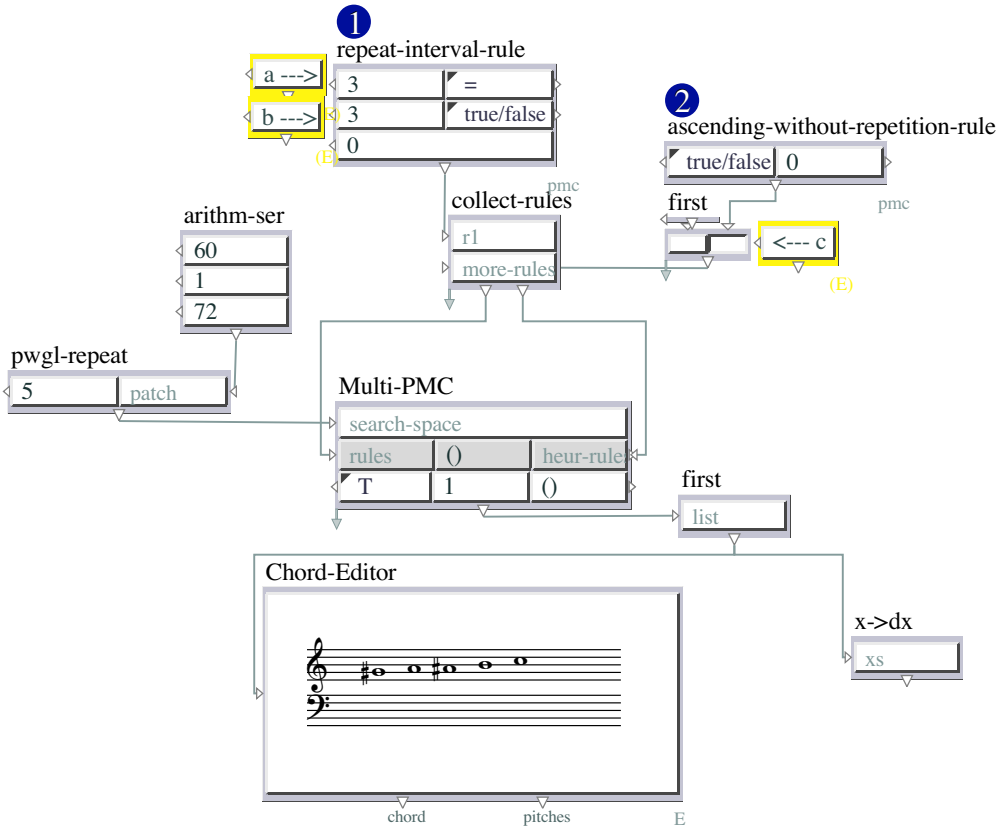

Figure 35: 1-02-05-repeat-interval

2.3.7 06-Repeat-Resulting-Interval

1.02.06 - REPEAT-RESULTING-INTERVAL-RULE
REPEAT-RESULTING-INTERVAL-RULE [1] obliges a solution to have a given resulting-interval repeated many times as indicated in the 'time' input. A resulting interval is an interval between a note and any possible other notes.
In this sense, look at the function FIND-ALL-INTERVALS (that you can call using the package JBS-CONSTRAINTS::FIND-ALL-INTERVALS). This function [a] gives all the intervals between all notes of a sequence.
If the menu which? is set on $<$, it means that the given interval has to be repeated a number of time inferior to the one put in times. If the menu which? is set on $=$, it means that the given interval has to be repeated a number of time equal to the one put in times. If the menu which? is set on $>$, it means that the given interval has to be repeated a number of times bigger than the one put in times.

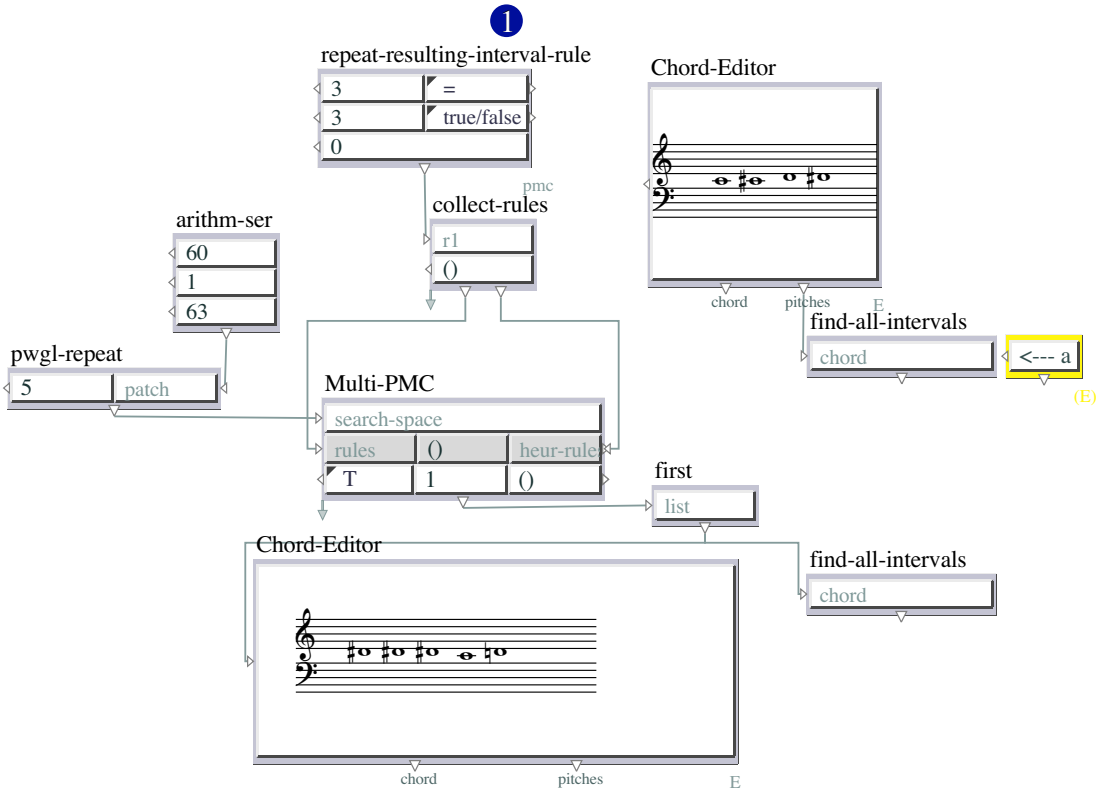

Figure 36: 1-02-06-repeat-resulting-interval

2.3.8 07-Index-or-Not-Index-Interval

1.02.07 - INDEX-INTERVAL-RULE
INDEX-INTERVAL-RULE [1] obliges a given interval indicated with 'index' to be a member of a list of possible intervals indicated in 'allowed'.
Attention ! The indexes start at zero as in Lisp.
If the menu absolute? is set in the absolute mode, it means that intervals are intended in absolute mode. If this menu is set in up/down mode, it means that the intervals are divided into ascending and descending.
NOT-INDEX-INTERVAL-RULE [2] does the opposite.

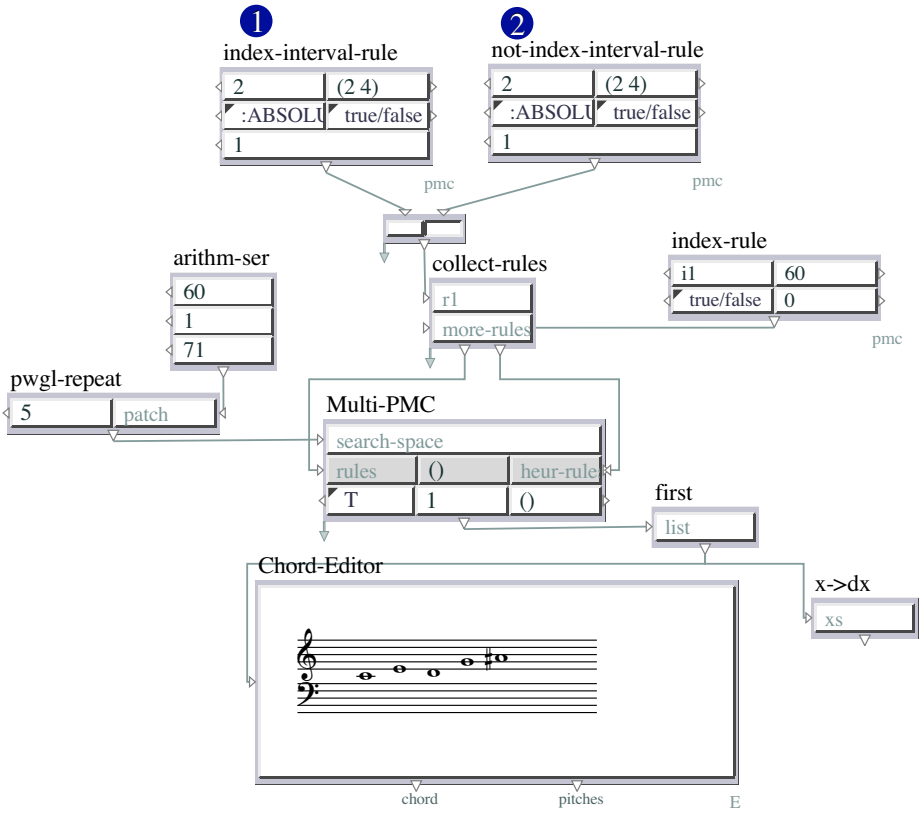

Figure 37: 1-02-07-index-or-not-index-interval

2.3.9 08-Not-Bigger-Not-Smaller-Interval

1.02.08 - NOT-BIGGER-NOT-SMALLER-INTERVAL-RULE
This patch shows you two functions with a particular behaviour.
To use NOT-BIGGER-INTERVAL-RULE [1], you first need to define if you work on positive, negative or absolute intervals using the menu 'sign?' [a].
If you choose '+', it means that this function does not allow intervals bigger than the one entered in 'limit' only for positive intervals. If you choose '-', it means that this function does not allow intervals bigger than the one entered in 'limit' only for negative intervals. If you choose 'absolute', it means that this function does not allow intervals bigger than the one entered in 'limit' both for positive and negative intervals.
NOT-SMALLER-INTERVAL-RULE [2] does the opposite.

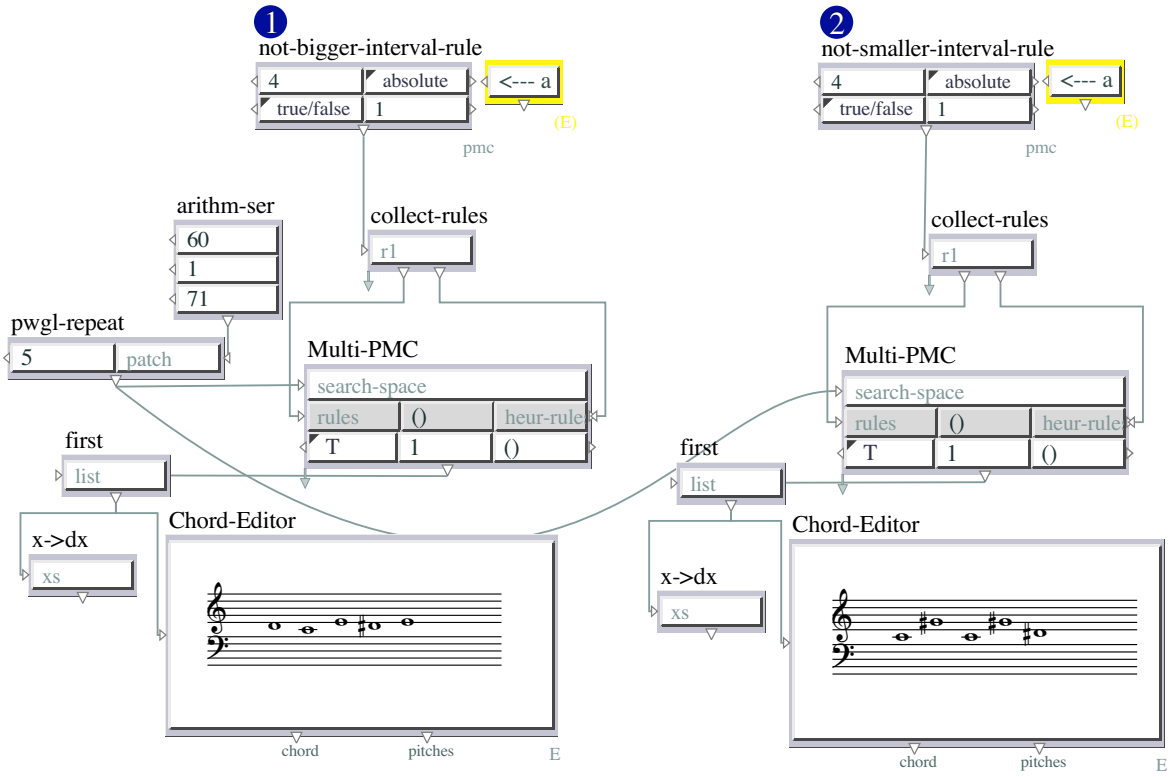

Figure 38: 1-02-08-not-bigger-not-smaller-interval

2.3.10 09-Resulting-Not-Resulting-Interval

1.02.09 - RESULTING-NOT-RESULTING-INTERVAL-RULE
This patch refers to the concept of a resulting interval. A resulting interval is an interval between a note of a sequence and any possible other notes in the same sequence.
The RESULTING-INTERVAL-RULE [1] obliges the solution to have, among all the intervals within a sequence, the defined interval.
The NOT-RESULTING-INTERVAL [2] does the opposite.
In this sense, look at the function FIND-ALL-INTERVALS (that you can call using the package JBS-CONSTRAINTS::FIND-ALL-INTERVALS). This function gives all the interval between all notes of a sequence. Evaluate the FIND-ALL-INTERVALS box [3] and look at the result.

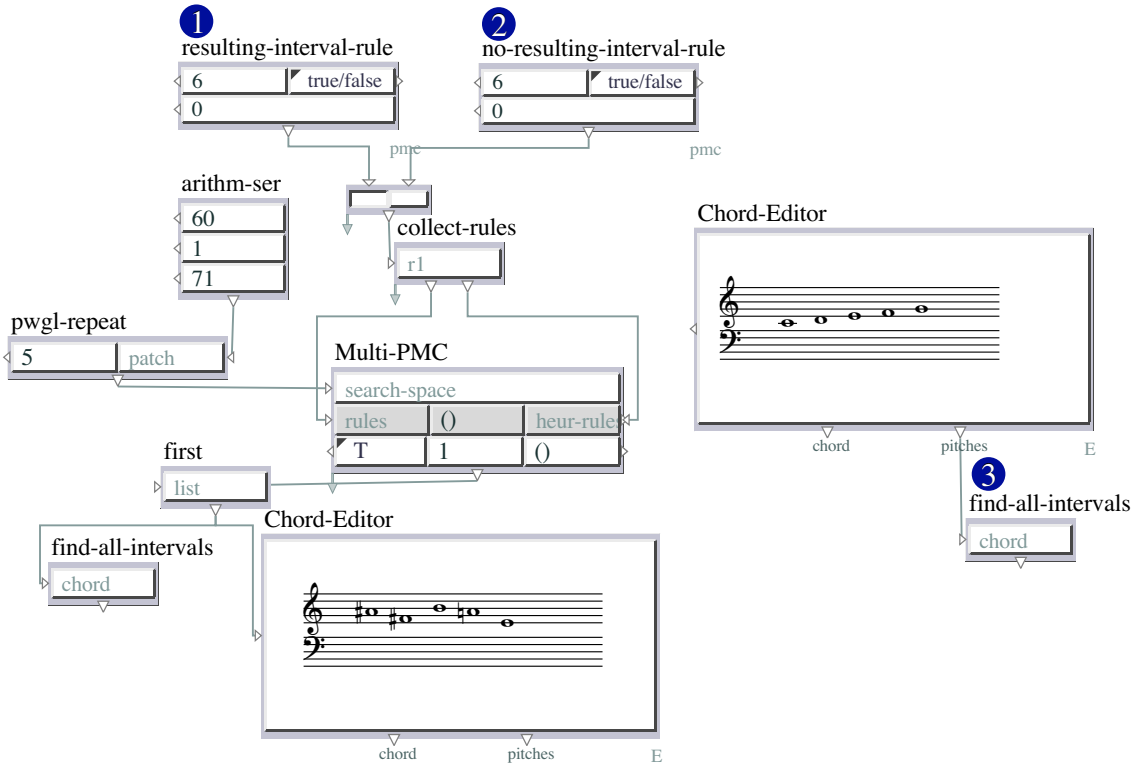

Figure 39: 1-02-09-resulting-not-resulting-interval

2.3.11 10-Jump-Resolution

1.02.10 - JUMP-RESOLUTION-RULE
If an interval is bigger than the value put in the 'interval' input, the next interval has to go in the opposite direction and it has to be smaller than the value put in 'resolution' input.

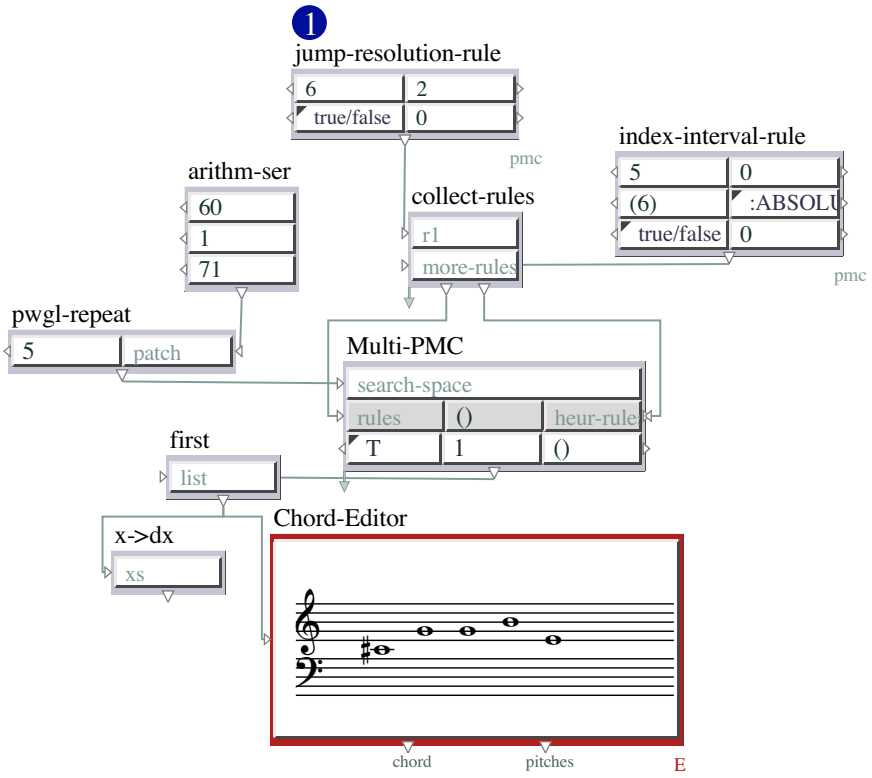

Figure 40: 1-02-10-jump-resolution

2.3.12 11-Do-Reach-Do-Not-Reach-That-Interval

1.02.11 - DO-REACH-DO-NOT-REACH-THAT-INTERVAL-RULE
DO-REACH-THAT-INTERVAL-RULE [1] obliges a solution to reach a given interval [a] within a given number of notes [b].
DO-NOT-REACH-THAT-INTERVAL-RULE [2] does the contrary.

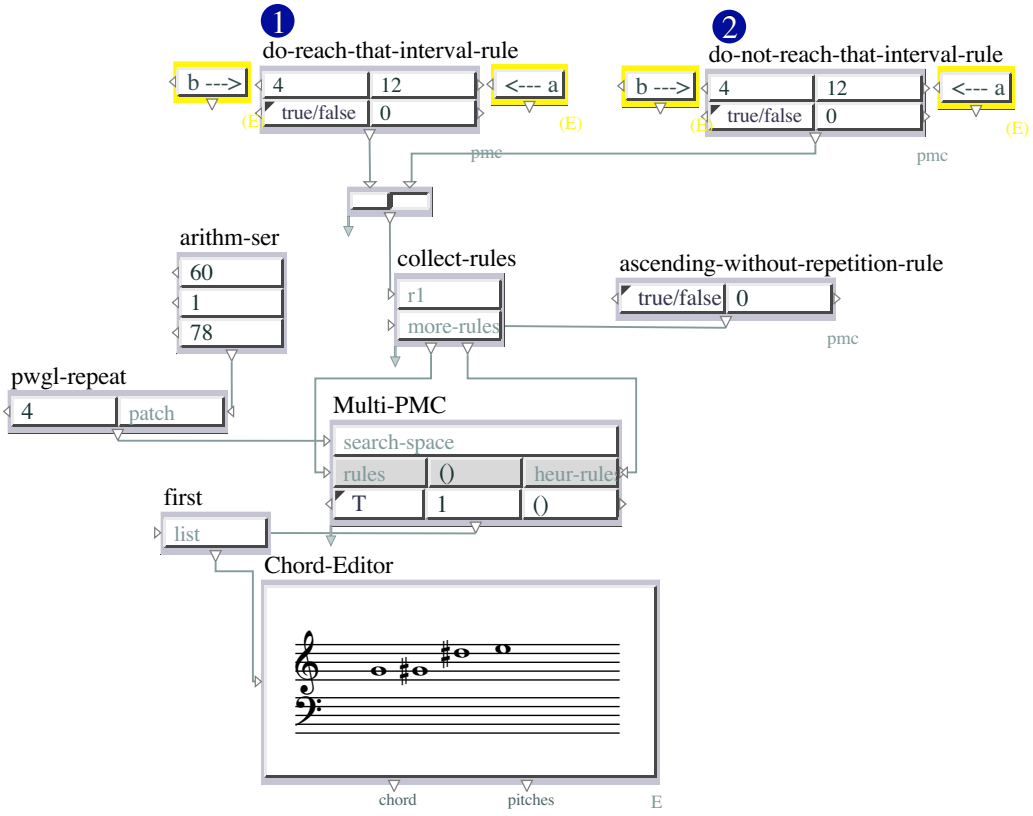

Figure 41: 1-02-11-do-reach-do-not-reach-that-interval

2.3.13 12-Apply-Interval-Sum

1.02.12 - APPLY-INTERVAL-SUM-RULE
APPLY-INTERVAL-SUM-RULE [1] outputs a solution having the sum of all intervals equal to the value put in 'sum' [a].
Evaluate the APPLY box [2] and see that the result is equal to the defined value.

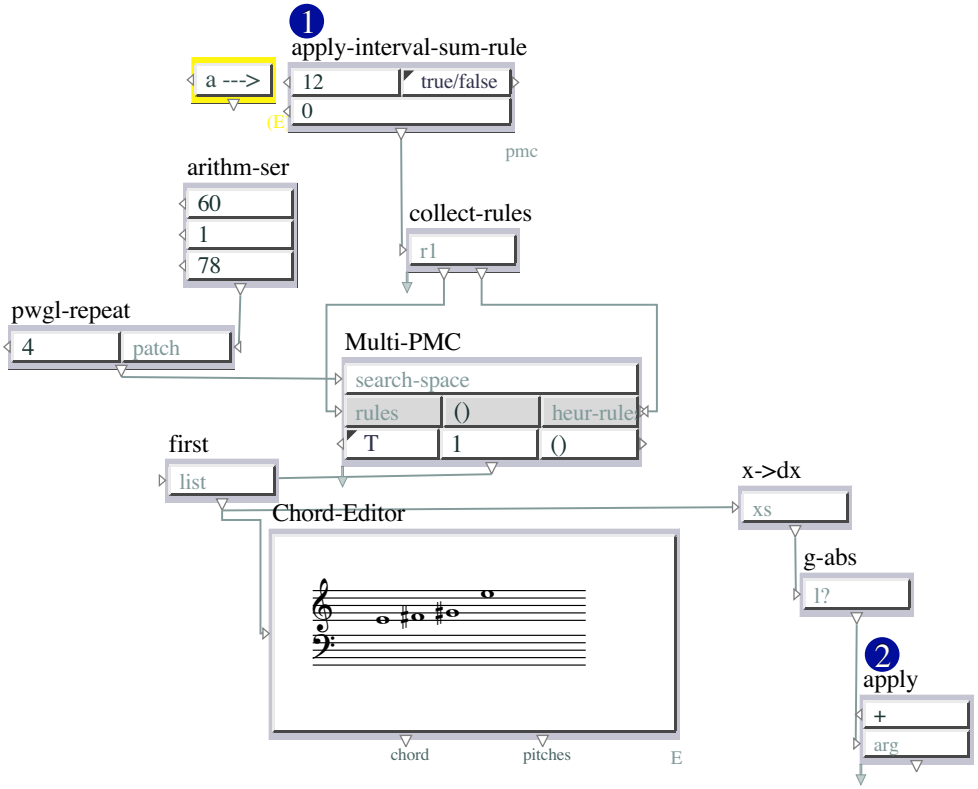

Figure 42: 1-02-12-apply-interval-sum

2.3.14 13-Apply-Interval-Global-Sum

1.02.13 - APPLY-INTERVAL-GLOBAL-SUM-RULE
This rule [1] outputs a solution having the sum of all intervals equal to the value put in 'sum'.
The difference from the APPLY-INTERVAL-SUM-RULE is to discover by yourself. [2]

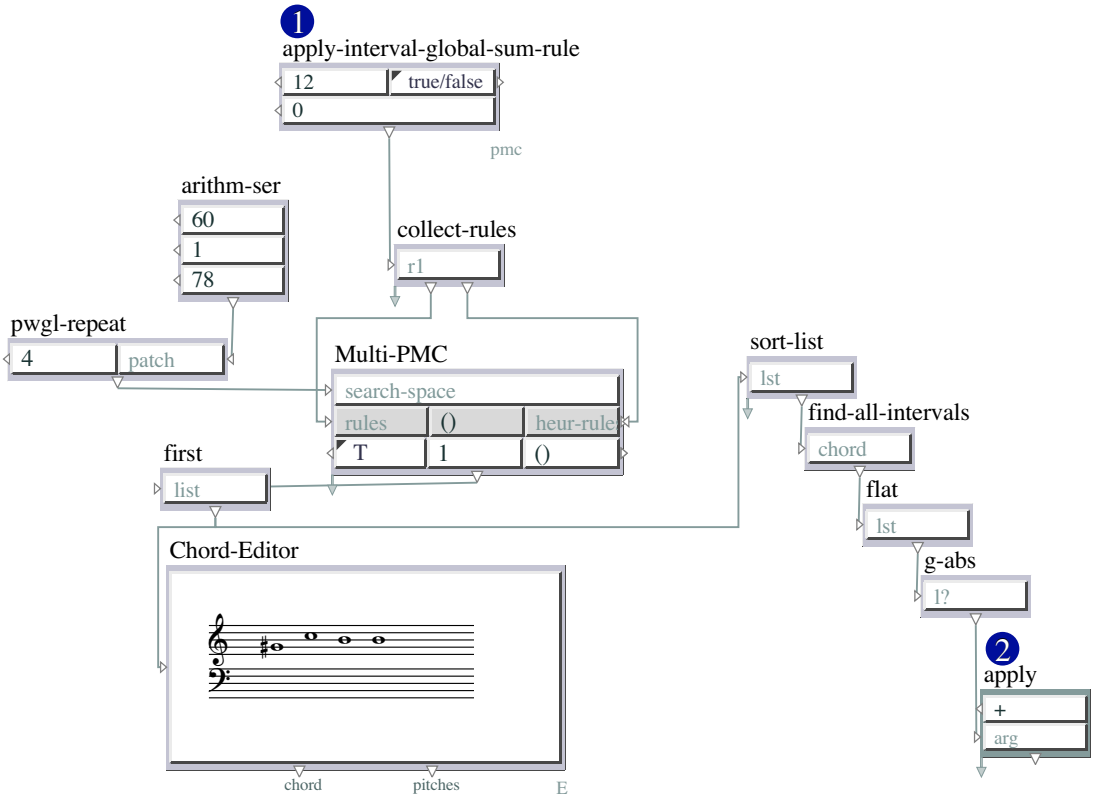

Figure 43: 1-02-13-apply-interval-global-sum

2.3.15 14-Not-Complementary-Interval

1.02.14 - NOT-COMPLEMENTARY-INTERVAL-RULE
This rule [1] does not allow the existence of a given interval set in [a] as the product of two consecutive intervals.
Please evaluate the APPLY box [b] and see that the interval defined in [a] is never produced by two complementary intervals.

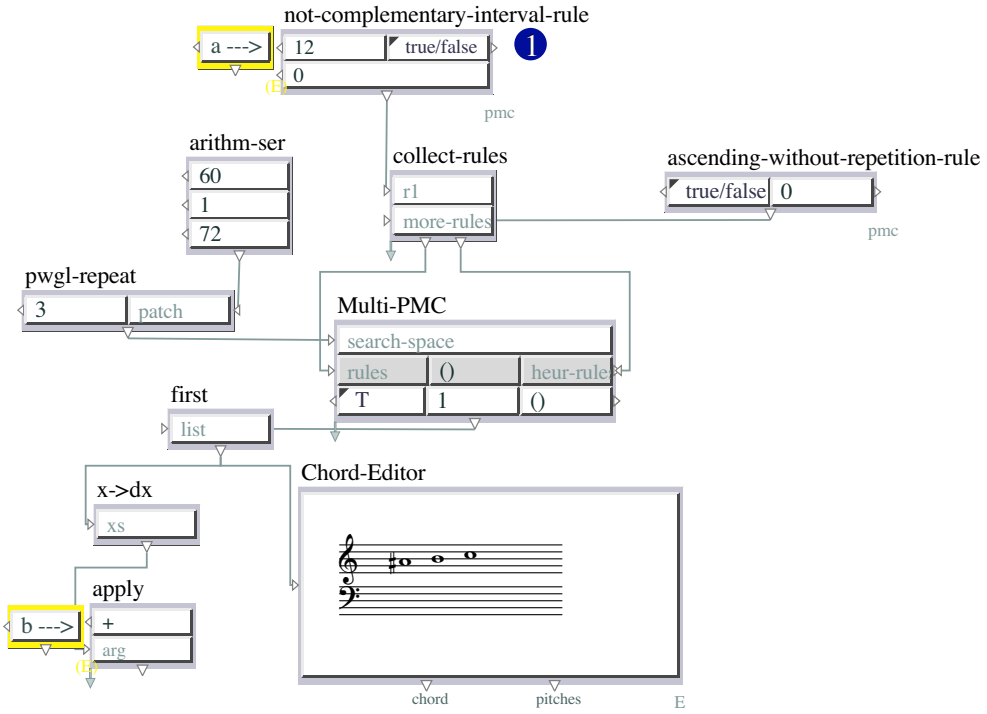

Figure 44: 1-02-14-not-complementary-interval

2.3.16 15-Interval-Structure

1.02.15 - INTERVAL-STRUCTURE-RULE
The INTERVAL-STRUCTURE-RULE [1] obliges a solution to have the given interval sequence defined in [a].
N.B. : BE CAREFULL ! The number of intervals put in interval-structure has to be one element less than the number of candidates you put in the search space [b].
The NOT-INTERVAL-STRUCTURE-RULE [2] does the opposite. It forbids a solution to have a given sequence of intervals defined in [a].
Here too, BE CAREFULL ! The number of intervals put in interval-structure has to be one element less than the number of candidates you put in the search space [b].

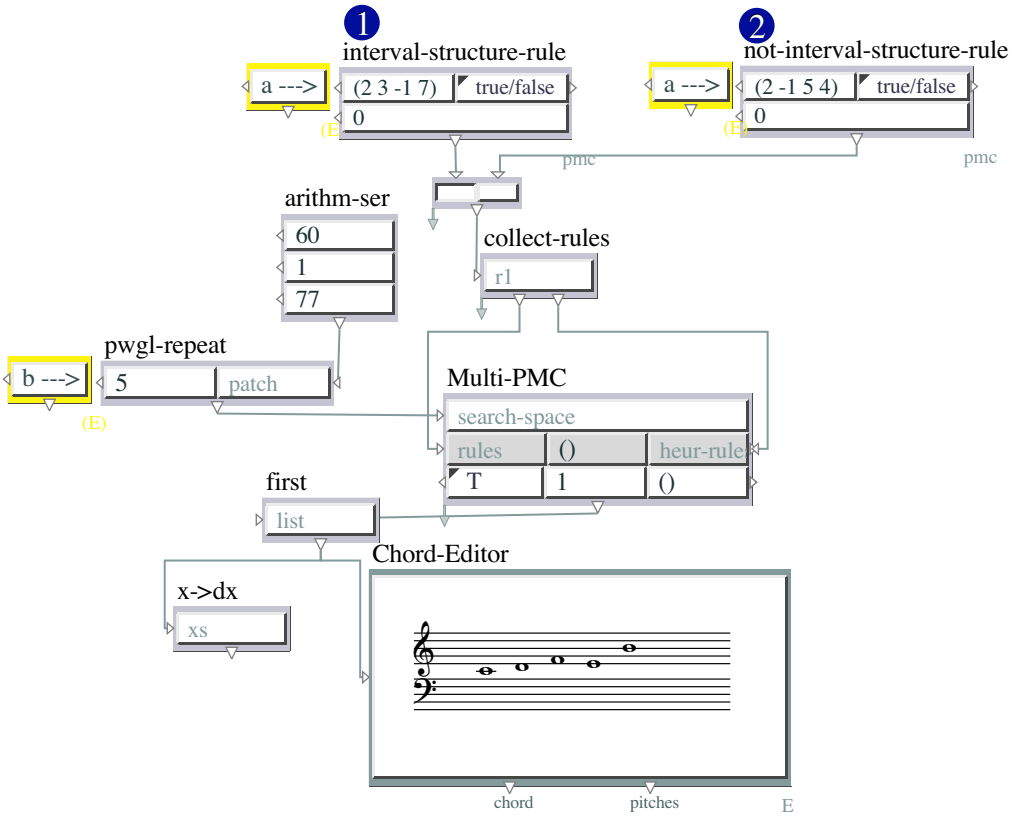

Figure 45: 1-02-15-interval-structure

2.3.17 16-Count-Positive-Negative-Intervals

1.02.16 - COUNT-POSITIVE-NEGATIVE-INTERVALS-RULE
If you use the COUNT-POSITIVE-INTERVALS-RULE [1], the solution will have a number of positive intervals as indicated in 'number' [a].
If you use the COUNT-NEGATIVE-INTERVALS-RULE [2], the solution will have a number of negative intervals as indicated in 'number' [a].

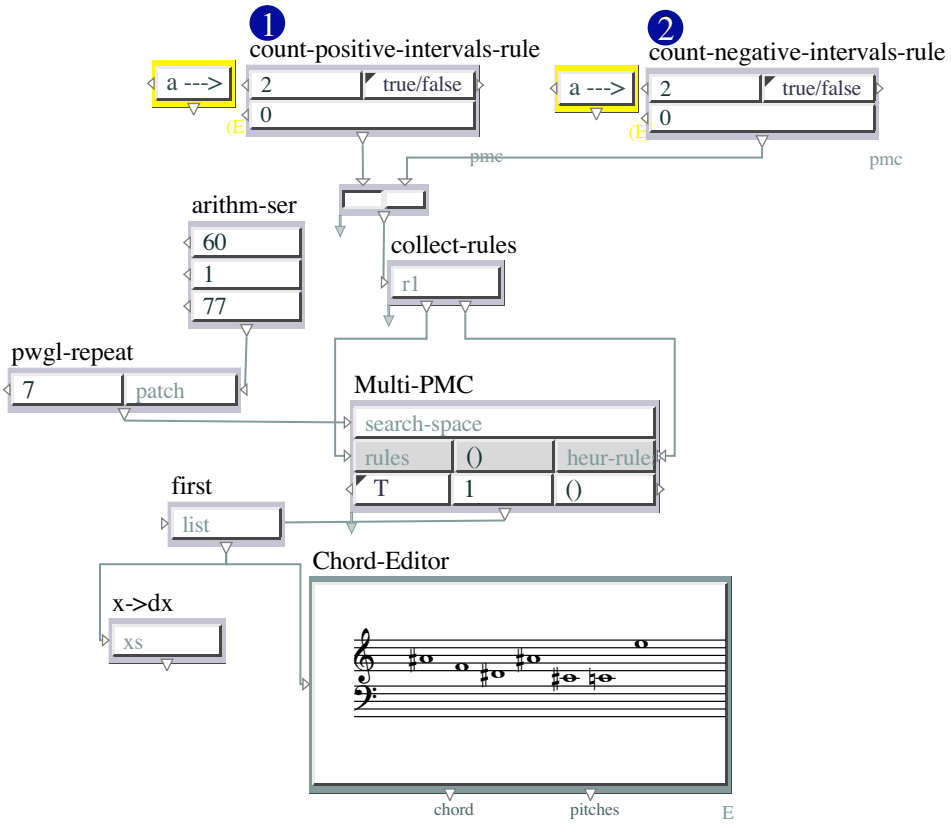

Figure 46: 1-02-16-count-positive-negative-intervals

2.4 03-Pitch-Rules

2.4.1 Pitch-Rules

The following rules are conceived for pitches. For that reason sometimes this tutorial refers to the modulo 12 (or 24, or 48 if we work with quarter or eighth of tones). Please if you do not know the function g-mod, see the PWGL documentation about it.
Just as a reminder : arithmetic modulo 12 is used by musicians, for instance, in consideration of the twelve-tones equal temperament system, where octave and enharmonic equivalency occurs.

2.4.2 01-Allowed-and-Not-Allowed-Pitches

1.03.01 - ALLOWED-PITCH-RULE
ALLOWED-PITCH-RULE [1] forces the solution to be constituted only by pitches indicated in 'pitch' input [a]. It differs from MEMBER-RULE because it considers pitches at any octave defined in the candidates.
NOT-ALLOWED-PITCH-RULE [2] does the opposite.

ALLOWED-POLARIZED-PITCH-RULE [3] allows only, for a given modulo 12 note, the pitch defined in [a]. In this example, the rule allows only, for the C and D notes, the pitch 60 and 62 and forbids all others.
ATTENTION ! You can control the results by evaluating both the CHORD-EDITOR and the G-MOD function [4], which reduces any value to its modulo 12. For instance, any C-sharp (49, 61 or 73) will be reduced to 1, any E (52, 64 or 76) will be reduced to 4, and so on...
ATTENTION AGAIN !!! If you work with microtones be sure that all pitches you define in [a] are floats. That means that if you admit 60 as a possible note, if you are using microtonal candidates (like quarter of tones for instance) you have to use 60.0 and not simply 60.

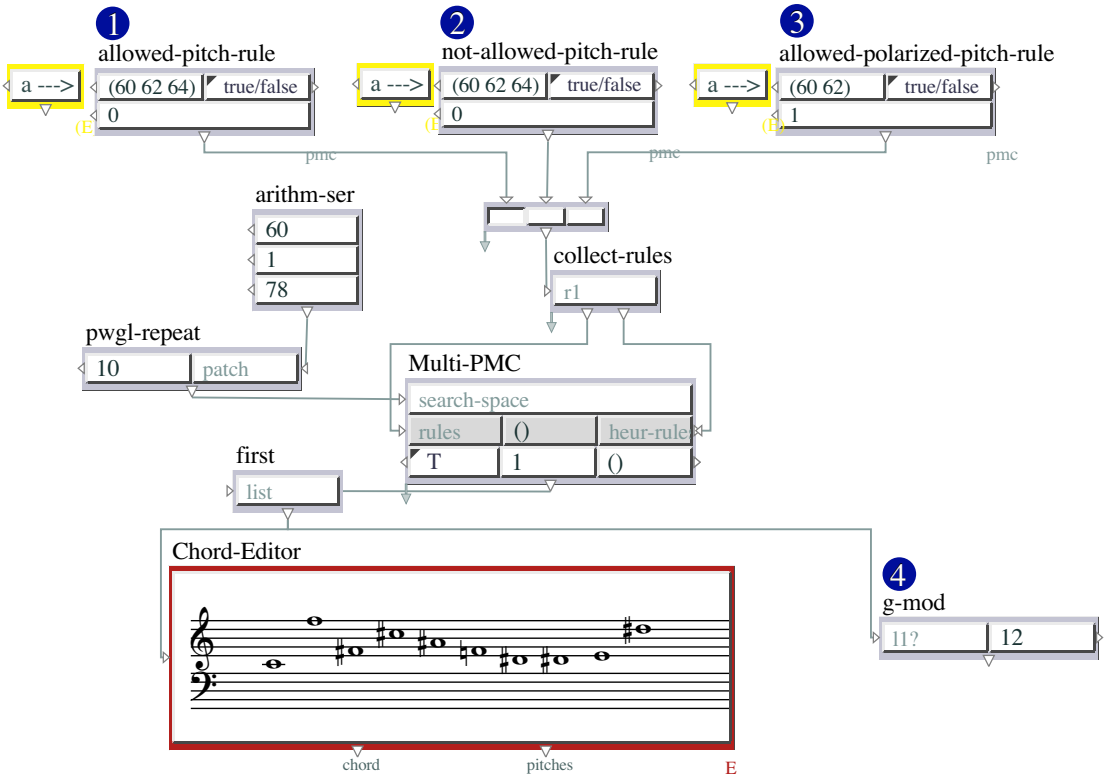

Figure 47: 1-03-01-allowed-and-not-allowed-pitches

2.4.3 02-Allowed-Pitches-Structure-and-Class

1.03.02 - ALLOWED-PITCHES-STRUCTURES-AND-CLASSES
This patch shows you three different ways to define the pitches you want within your solution. These rules are based on the concepts of pitch structure and pitch class.
ALLOWED-PITCH-STRUCTURE-RULE [1] creates a solution based on the set theory. In

this way only the set of pitches indicated in 'pitch' input will be allowed at any octave. N.B. : The chord produced will always be complete, or in other words it will always contain all the chosen pitches side by side.
ALLOWED-PITCH-CLASS-RULE [2] creates a solution based on a pitch class (a concept belonging to the set theory) derived from a chord put in [a]. In this case a C-minor triad is entered in [a]. The Multi-PMC will find any solution corresponding to the same class, i.e. at any possible octave, transposition, inversion or arrangement.
ALLOWED-PITCH-CLASS-SUB-LIST-RULE [3] allows an occurence of a defined pitch class (for instance a minor triad, indicated in 'pitch' input) among other notes in the solution So in this example I'm looking for a solution including a (consecutive) minor triad within a 5 notes chord.
ATTENTION ! You can control the results by evaluating both the CHORD-EDITOR and the G-MOD function [4], which reduces any value to its modulo 12. For instance, any C-sharp (49, 61 or 73) will be reduced to 1, any E (52, 64 or 76) will be reduced to 4, and so on...

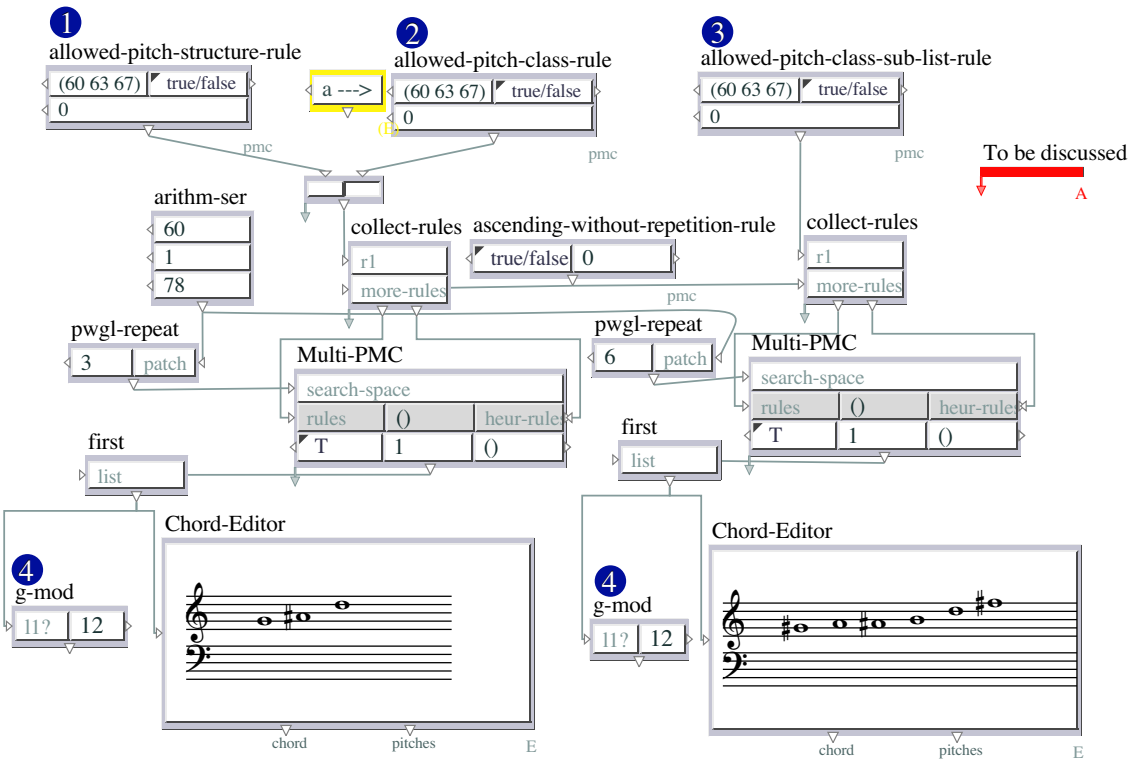

Figure 48: 1-03-02-allowed-pitches-structure-and-class

2.4.4 03-Not-Allowed-Pitches-Structure-and-Class

1.03.03 - NOT-ALLOWED-PITCHES-STRUCTURES-AND-CLASSES
This patch shows you three different ways to define the pitches you do not want to occur within your solution. These rules are based on the concepts of pitch structure and pitch class.
NOT-ALLOWED-PITCH-STRUCTURE-RULE [1] creates a solution based on the set theory. In this way, the set of pitches indicated in 'pitch' input will NOT be allowed at any octave. N.B. : The chord produced may contain some of the pitches defined in 'pitch' but never side by side.
NOT-ALLOWED-PITCH-CLASS-RULE [2] creates a solution that do NOT include the same pitch class (a concept belonging to the set theory) derived from a chord put in [a]. In this case a C-minor chord is entered in [a]. The Multi-PMC will find a solution NOT including this class, with any possible octave, transposition, inversion or arrangement.
The NOT-ALLOWED-PITCH-CLASS-SUB-LIST-RULE [3] forbids any occurence of a class, defined in 'pitch' input, among the notes of the solution. So in this example I am looking for a solution in which a minor triad cannot occur.
ATTENTION ! You can control the results by evaluating both the CHORD-EDITOR and the G-MOD function [4], which reduces any value to its modulo 12. For instance, any C-sharp (49, 61 or 73) will be reduced to 1, any E (52, 64 or 76) will be reduced to 4, and so on...

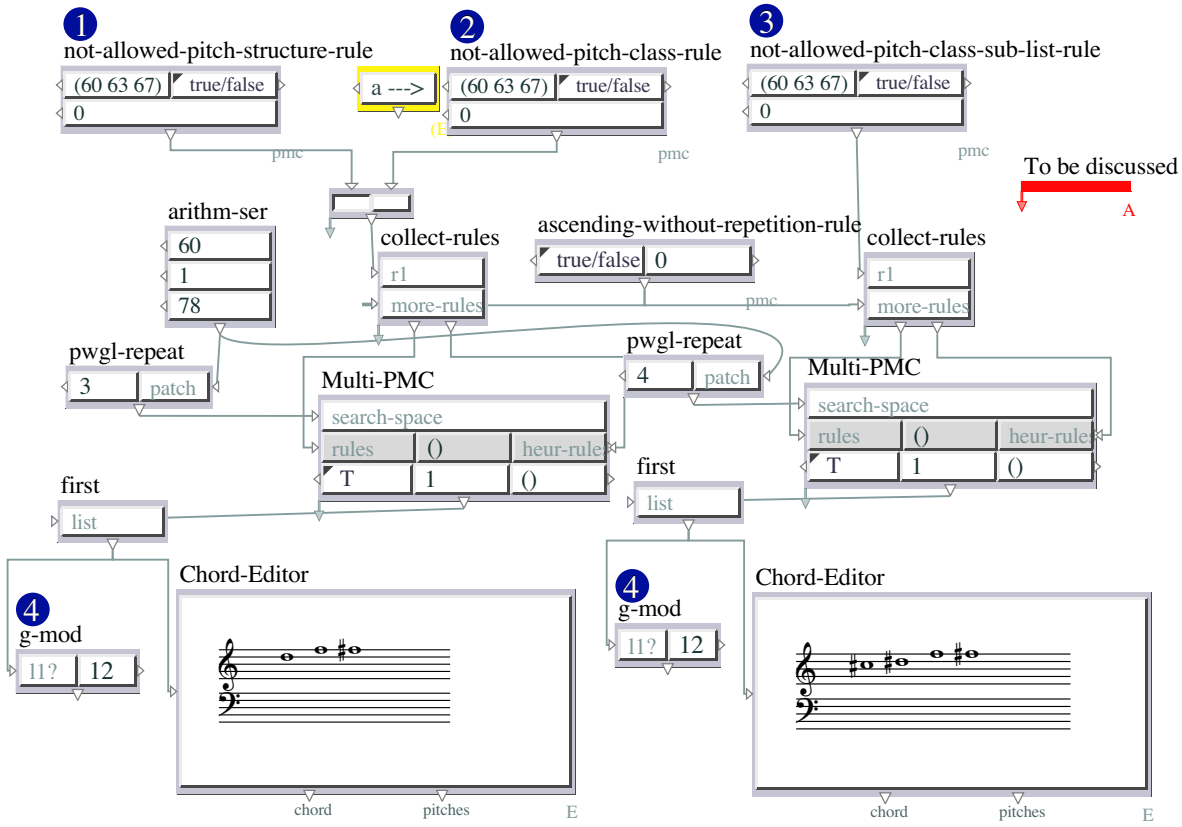

Figure 49: 1-03-03-not-allowed-pitches-structure-and-class

2.4.5 04-Index-and-Not-Index-Pitch

1.03.04 - INDEX-AND-NOT-INDEX-PITCH-RULE
[1] The INDEX-PITCH-RULE forces a given position (’index’ input)in the solution to be occupied by any pitch (at any octave) in the list defined with ’pitch’ input.
[2] The NOT-INDEX-PITCH-RULE does the opposite.

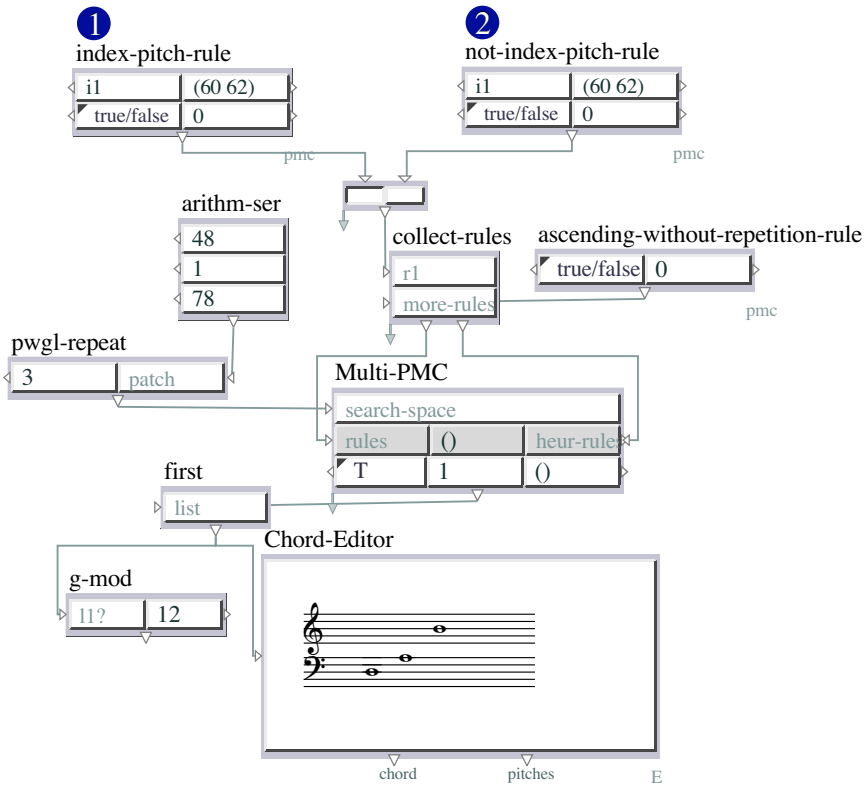

Figure 50: 1-03-04-index-and-not-index-pitch

2.4.6 05-Any-Note-Repeated

1.03.05 - ANY-NOTE-REPEATED-RULE
ANY-NOTE-REPEATED-RULE [1] verifies that any note (modulo 12) in the solution is repeated more, less or exactly the number of time defined in 'times' input.
N.B. BE CAREFUL The menu 'which?' defines less, equal or more. If you set $<$, the calculation is quite fast. If you set $=$, be sure to have a pair number of candidates in the search space. If you set $>$, the calculation can be very slow.

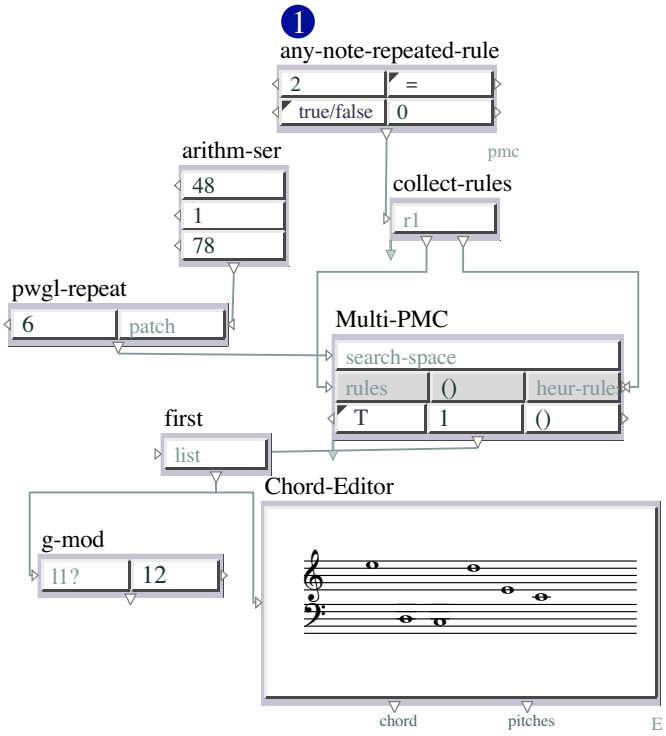

Figure 51: 1-03-05-any-note-repeated

2.4.7 06-Count-this-Note-and-Modulo

1.03.06 - COUNT-THIS-NOTE-AND-MODULO
This patch shows you how to count a single note or the modulo 12 of a given note.
COUNT-THIS-NOTE-RULE [1] obliges a solution to have the given note repeated many times as indicated in 'how-many' in the exact defined octave.
COUNT-THIS-MODULO-RULE [2] obliges a solution to have a given note repeated many times as indicated in 'how-many', in any possible octave.

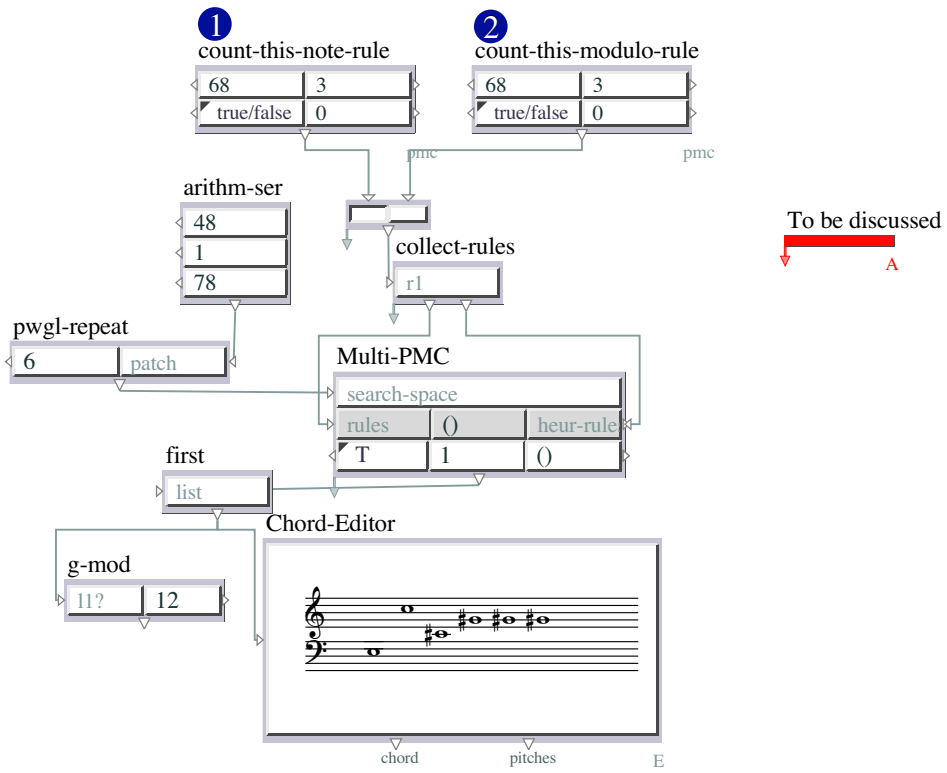

Figure 52: 1-03-06-count-this-note-and-modulo

2.4.8 07-Not-Repeated-Modulo-12

1.03.07 - NOT-REPEATED-MODULO-12
NOT-MODULO12-REPETITION-RULE [1] creates a solution without any modulo 12 repetition, i.e. in any octave.
NOT-MODULO12-LOCAL-REPETITION-RULE [2] creates a solution without any consecutive modulo 12 repetition.
You can control the result of these two rules by evaluating the first output [a] of the 'g-mod-group' abstraction [3].
NOT-REPEATED-MODULO12-SUB-GROUP-RULE [3] creates a solution without any modulo 12 repetition inside each sub-group whose length is defined by 'sub-group-length' input.
You can control the result of this rule by evaluating the second output [b] of the 'g-mod-group' abstraction [3]. Please verify that [c] in the abstraction is the same value than the one defined in 'sub-group-length' input of the rule.

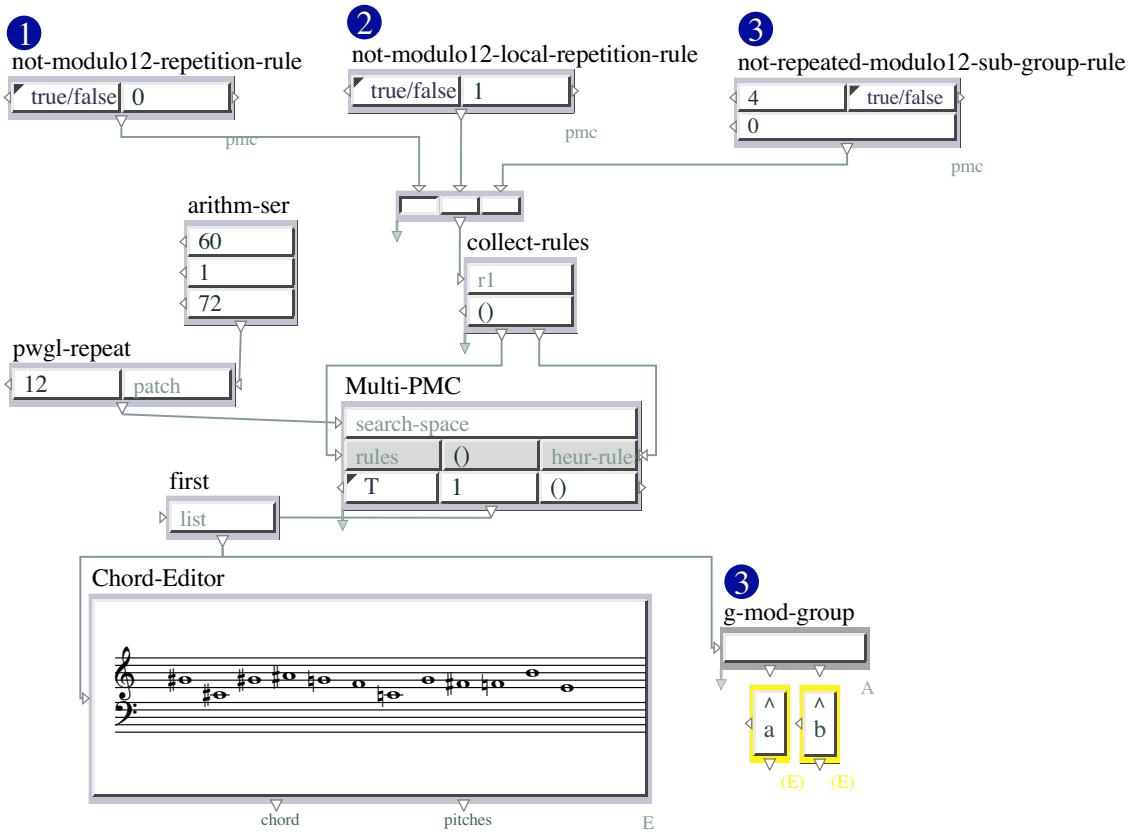

Figure 53: 1-03-07-not-repeated-modulo-12

2.4.9 08-Mk-Profile-Pitch

1.03.08 - MK-PROFILE-PITCH-RULE
MK-PROFILE-PITCH-RULE [1] asks the engine to put out a solution identical to the profile in bpf format [2].
The 'approx' input [a] defines the approximation of the mapping from the 2D-EDITOR's samples. 2 will output values for the chromatic scale, 4 will round the values to quarter of tones, 8 to eighth of tones, and so on... Take care to change the step of the ARITM-SER box [b] to 0.5 or 0.25 in order to produce a result with micro-tones.
ATTENTION ! This rule is essentially useful in heuristic mode, in order that other rules can contradict it.

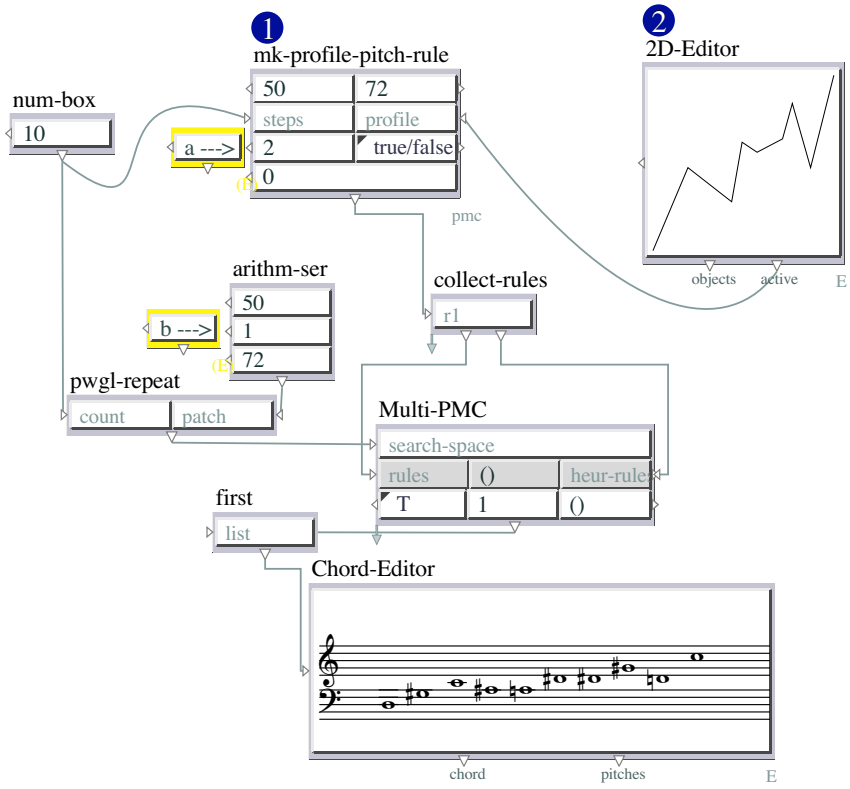

Figure 54: 1-03-08-mk-profile-pitch

2.4.10 09-Mk-Profile-Pitch-Modulo

1.03.09 - MK-PROFILE-PITCH-MODULO-RULE
MK-PROFILE-PITCH-MODULO-RULE [1] asks the engine to put out a list of pitches based on the same modulo 12 sequence than the given profile.
It creates a variation of a given pitch profile on any octave.

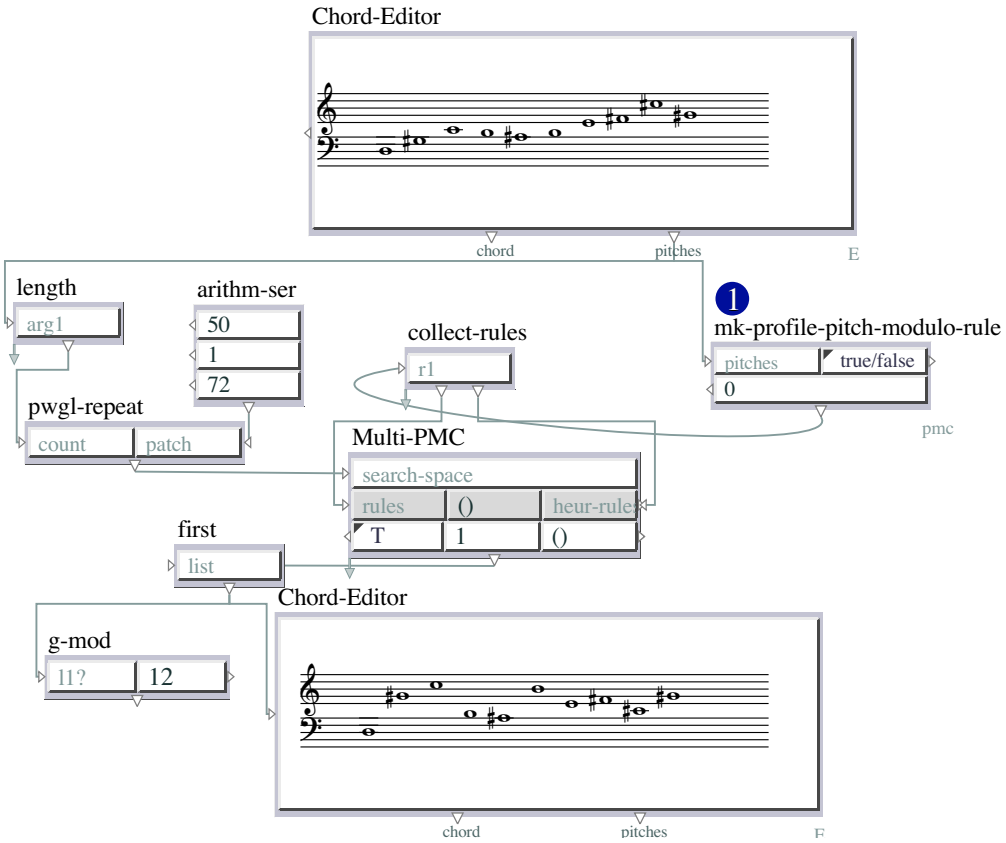

Figure 55: 1-03-09-mk-profile-pitch-modulo

2.5 04-Shaping-Rules

2.5.1 Shaping-Rules

The following rules have been conceived to control and generate shapes. The concept of shape is at the same time based on BPF representations and on melodic profile representations. In a sense, this part of the library concerns geometrical disposition of a sequence of elements.
Two particular functions (DIRECT-ANALYSIS and ENERGY-PROFILE) belong to the study of morphology. It is possible that in future these two functions' name and location will change. If it happens, I will keep these functions in the code, in order to make your patches run, but they'll probably disappear from this menu. Please look at the Morphologie library for more details.

2.5.2 01-Ascending-Descending-Rule

1.04.01 - ASCENDING-RULE
ASCENDING-RULE [1] obliges the solution to be always ascending.

ASCENDING-WITHOUT-REPETITION-RULE [2] obliges the solution to be always ascending and without any repetition.
DESCENDING-RULE [3] obliges the solution to be always descending.
DESCENDING-WITHOUT-REPETITION-RULE [4] obliges the solution to be always descending and without any repetition.

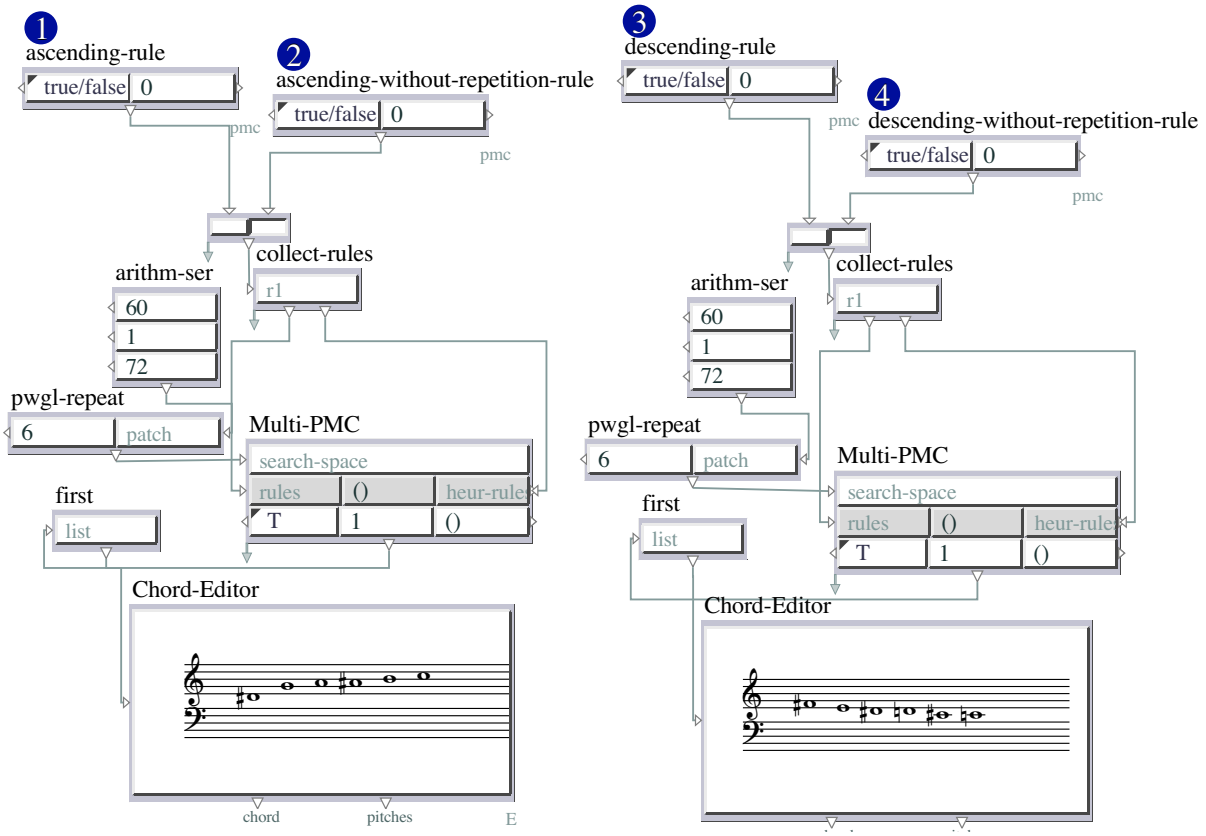

Figure 56: 1-04-01-ascending-descending-rule

2.5.3 02-Ascending-Descending-Sub-Group-Rule

1.04.02 - ASCENDING-DESCENDING-SUB-GROUP-NO-REPET-RULE
ASCENDING-SUB-GROUP-NO-REPET-RULE [1] obliges the nth value (put in 'nth-?') of each list of lists to be in an ascending order without any repetition. That means that all the first (if nth is equal to zero) elements of the all sub lists are in a sort (<) order without repetitions.
ASCENDING-SUB-GROUP-WITH-REPET-RULE [2] obliges the nth value (put in 'nth-?') of each list of lists to be in an ascending order with repetitions allowed. That means that all the first (if nth is equal to zero) elements of all sub-lists are in a sort (<) order with possible repetitions.
DESCENDING-SUB-GROUP-NO-REPET-RULE [3] obliges the nth value (put in 'nth-?')

of each list of lists to be in an descending order without any repetition. That means that all the first (if nth is equal to zero) elements of the all sub lists are in a sort (>) order without repetitions.
DESCENDING-SUB-GROUP-WITH-REPET-RULE [4] obliges the nth value (put in 'nth-?') of each list of lists to be in an descending order with repetitions allowed. That means that all the first (if nth is equal to zero) elements of the all sub lists are in a sort (>) order with possible repetitions.
ATTENTION To control the solutions you must read through all chord objects in the CHORD-EDITORS.

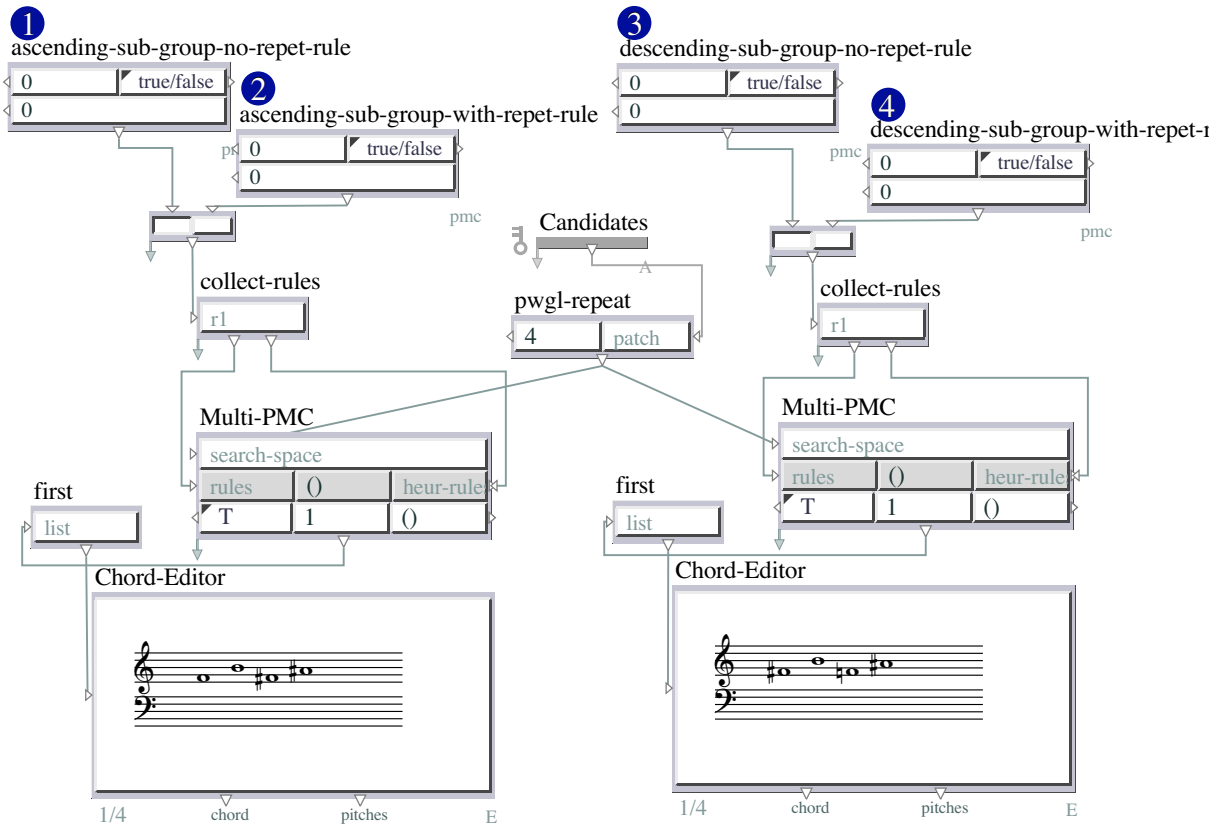

Figure 57: 1-04-02-ascending-descending-sub-group-rule

2.5.4 03-Mk-Fix-Profile-Rule

1.04.03 - MK-FIX-PROFILE-RULE
MK-FIX-PROFILE-RULE [1] asks the engine to put out a solution with a shape identical or similar to the list defined in 'profile' input [a].
About the uses of this rule, please look 'MK-Models' in the examples.

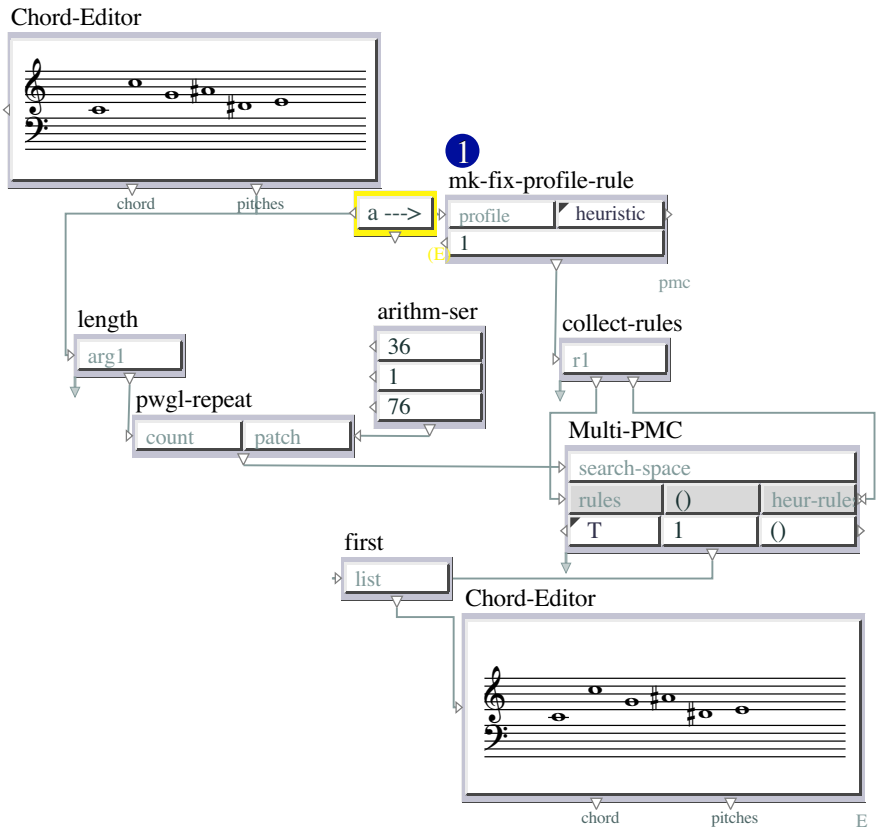

Figure 58: 1-04-03-mk-fix-profile-rule

2.5.5 04-Mk-Profile-Rule

1.04.04 - MK-PROFILE-RULE
MK-PROFILE-RULE [2] asks the engine to put out a solution with a shape similar to the bpf coming in 'profile' input. About the uses of this rule, please look 'MK-Models' in the examples.
In the NUM-BOX [1] choose how many notes have to constitute your profile. Then in the 2D-EDITOR [3] draw the curve you want to be the model of the profile.
In [a] you have to put the lowest note and in [b] the highest of the resulting profile. In [c] you can define how many decimals you use in order to approximate the values coming out of the 2D-EDITOR. This is particularly useful when you are dealing with candidates that are not midi values, like rationals.
Evaluate the CHORD-EDITOR [4] in order to see the results.

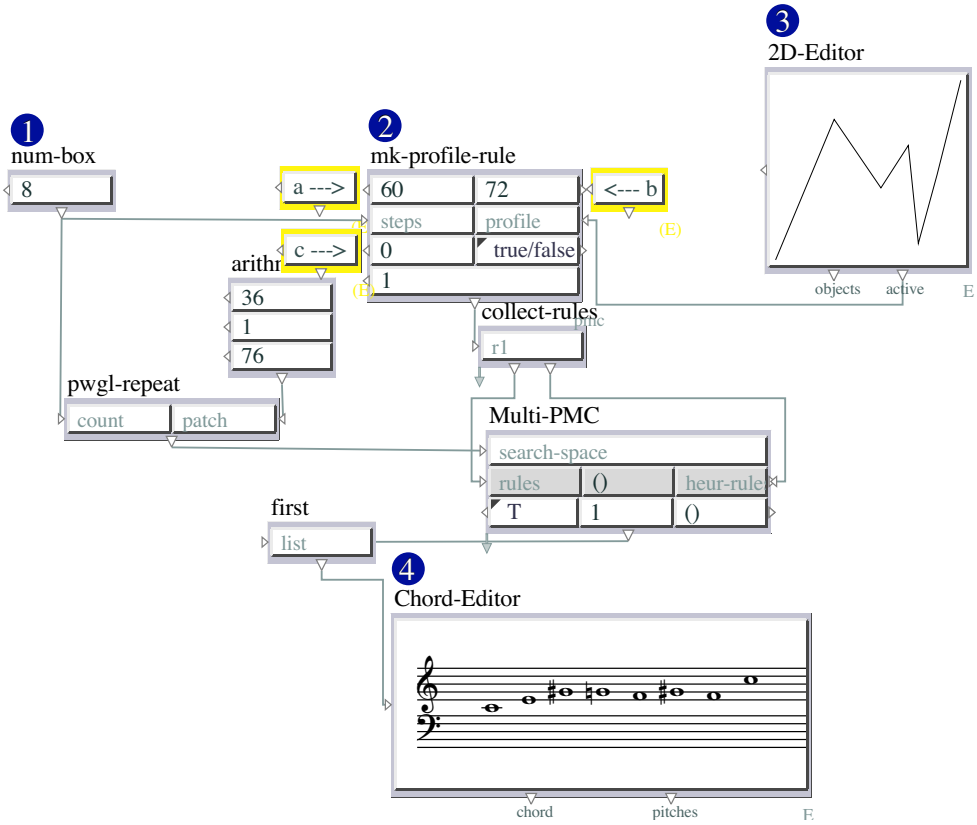

Figure 59: 1-04-04-mk-profile-rule

2.5.6 05-Sub-Group-Mk-Profile-Rule

1.04.05 - SUB-GROUP-MK-PROFILE-RULE
SUB-GROUP-MK-PROFILE-RULE [2] asks the engine to put out a solution in which, for each sub-groups, the nth (put in 'nth-?') has to be identical to follow the profile extracted from the bpf [3]. That means here that all the first elements (if nth is equal to zero) of all the sub-lists will take the order given by the profile.
In the NUM-BOX [1] choose how many notes have to constitute your profile. Then in the 2D-EDITOR [3] draw the curve you want to be the model of the profile. In [a] you have to put the lowest note and in [b] the highest of the resulting profile.
Evaluate the CHORD-EDITOR [4] and scroll through the succesive chords in order to see the results.
You can also evaluate the second 2D-EDITOR [c], produced by the g-first abstraction (it won't work if you change the 'nth-?' input of the rule) and see that the result follows as much as possible the original profile.
Finally, please open the 'sub-group-mk-fix-profile' abstraction. Here is a way to use SUB-GROUP-MK-PROFILE-RULE in order to restitute a given melodic profile, like with MK-FIX-PROFILE-RULE, in a particular index and through several sub-groups.

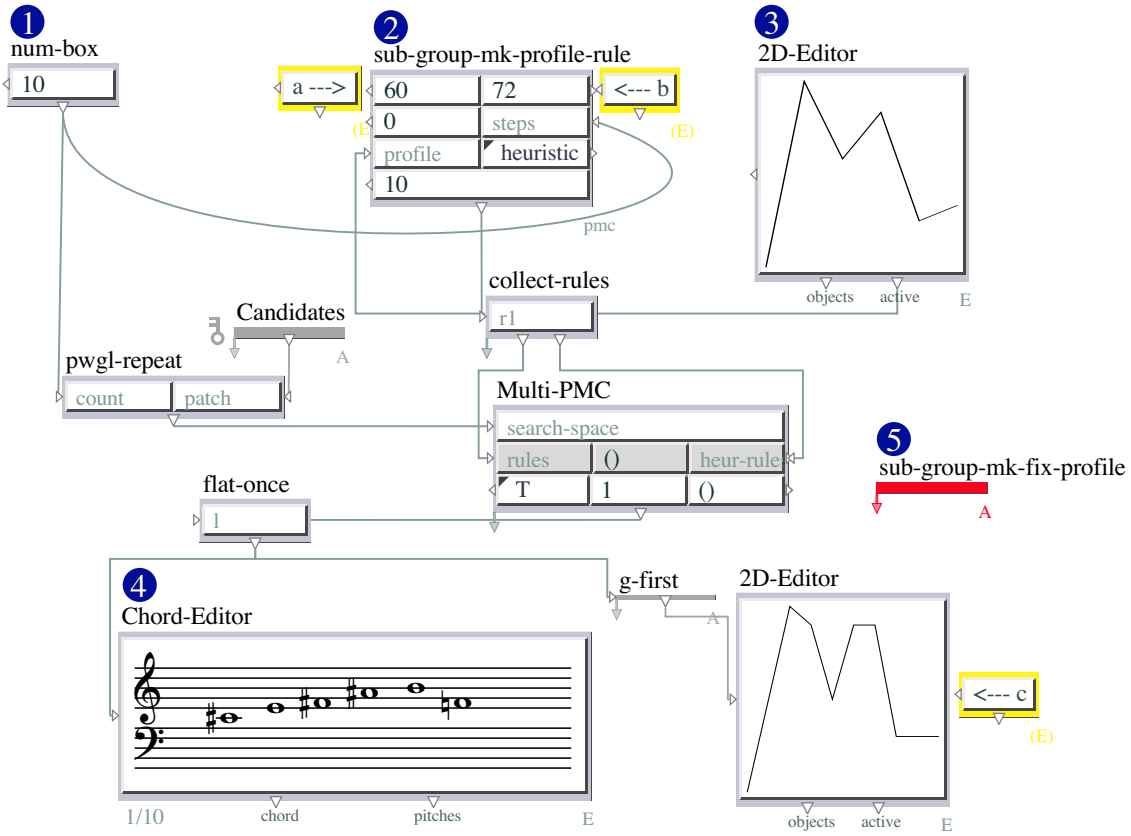

Figure 60: 1-04-05-sub-group-mk-profile-rule

2.5.7 06-Direct-Analysis-Rule

1.04.06 - DIRECT-ANALYSIS-RULE
DIRECT-ANALYSIS [2] is a function of the Morphologie library. Right now you just need to know that this function gives you the directions of intervals in a given sequence [1]. DIRECT-ANALYSIS-RULE [3] asks the engine to put out a solution that has an identical analysis in terms of directions. You can control this by evaluating the second DIRECT-ANALYSIS function [4].

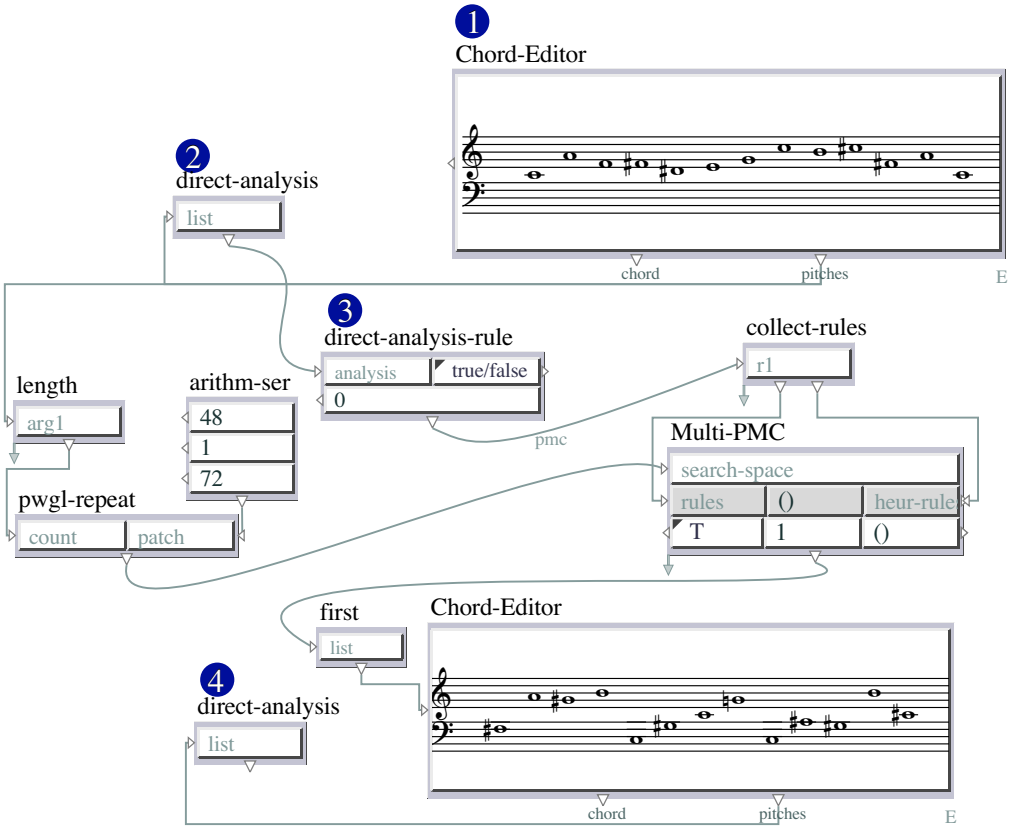

Figure 61: 1-04-06-direct-analysis-rule

2.5.8 07-Energy-Profile-Rule

1.04.07 - ENERGY-PROFILE-RULE
The CONTRASTS-LEV.1 [2] is a function of the Morphologie library. Right now you just need to know that this function gives you an analysis of the energy of information of a given sequence [1].
The ENERGY-PROFILE-RULE [3] asks the engine to put out a solution that has an identical analysis in terms of energy of information.

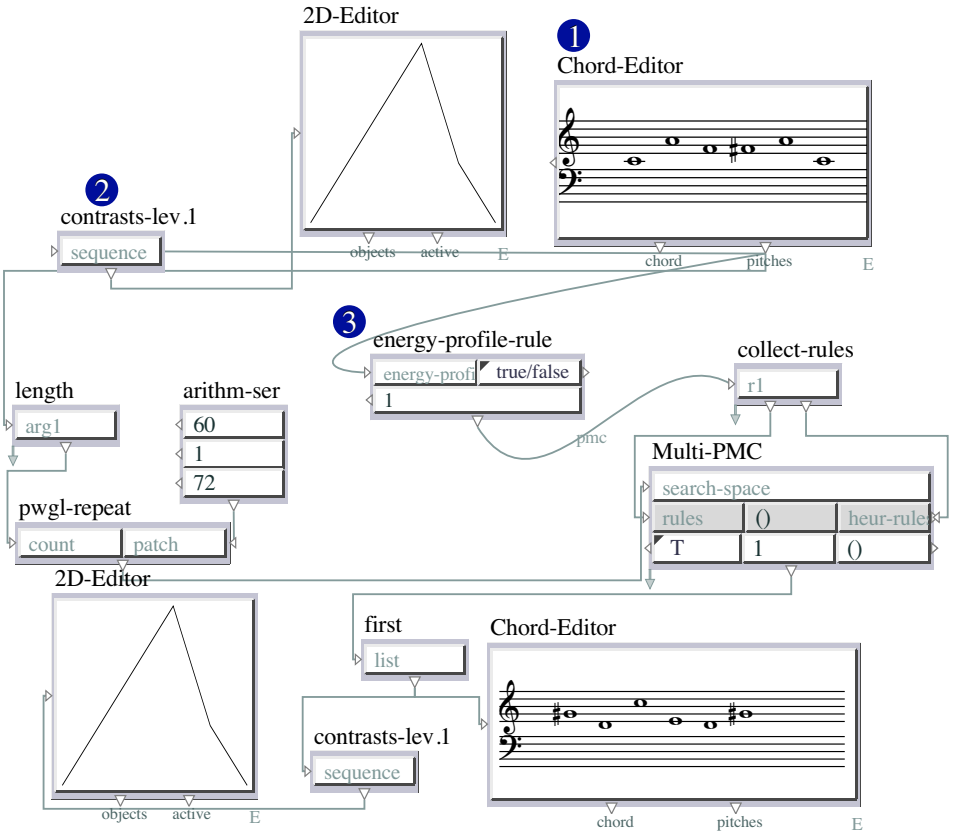

Figure 62: 1-04-07-energy-profile-rule

2.6 05-Pattern-Rules

2.6.1 Shaping-Rules

The following rules are useful to control and generate patterns. By pattern I mean a type of theme, of recurring events, or of objects, sometimes referred to as elements of a set. Patterns are based on repetition. Without repetitions there is no pattern.

2.6.2 01-Ptrn-Find-Not-Ptrn-Find-Rule

1.05.01 - PTRN-FIND-RULE
PTRN-FIND-RULE [1] looks for solutions including patterns with a given length of elements put in 'ptrn-length' input [a]. In 'repeated-ptrn' input [b] you have to put how many different patterns you want to be repeated along the solution.
If the menu 'which?' [c] is set on :
<, it means that you are looking for solutions having a number of repetitions of a pattern smaller than the one set in repeated-ptrn [b].

<=, it means that you are looking for solutions having a number of repetitions of a pattern smaller or equal to the one set in repeated-ptrn [b].
=, it means that you are looking for solutions having a number of repetitions of a pattern exactly equal to the one set in repeated-ptrn [b].
>=, it means that you are looking for solutions having a number of repetitions of a pattern bigger or equal to the one set in repeated-ptrn [b].
>, it means that you are looking for solutions having a number of repetitions of a pattern bigger than the one set in repeated-ptrn [b].
Please evaluate the PTRN-FIND function (which belong to the Morphologie library) and see that the solution includes as many repeated patterns than defined in [b].
NOT-PTRN-FIND-RULE does the opposite.

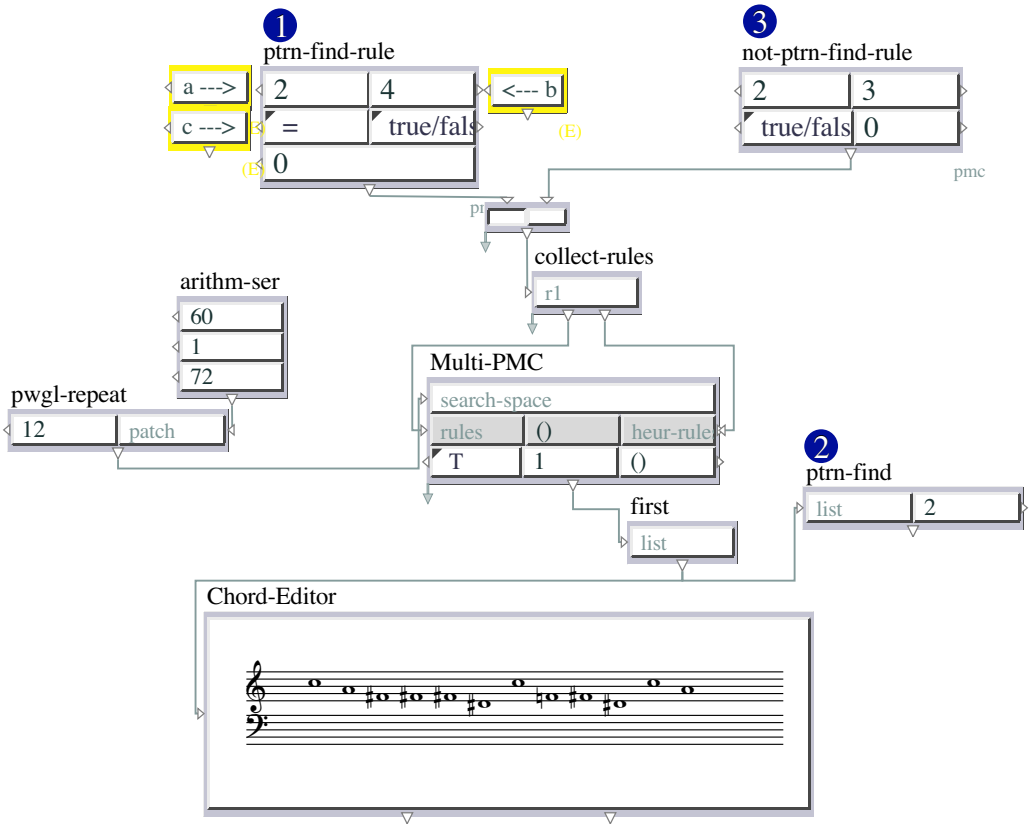

Figure 63: 1-05-01-ptrn-find-not-ptrn-find-rule

2.6.3 02-Find-this Ptrn-N-Times-Rule

1.05.02 - FIND-THIS-PTRN-N-TIMES-RULE
FIND-THIS-PTRN-N-TIMES-RULE [1] looks for a solution including the precise pattern you ask in 'pattern' input [a]. In 'repeated-ptrn' [b] you have to put how many times you want this pattern to be repeated in the solution.

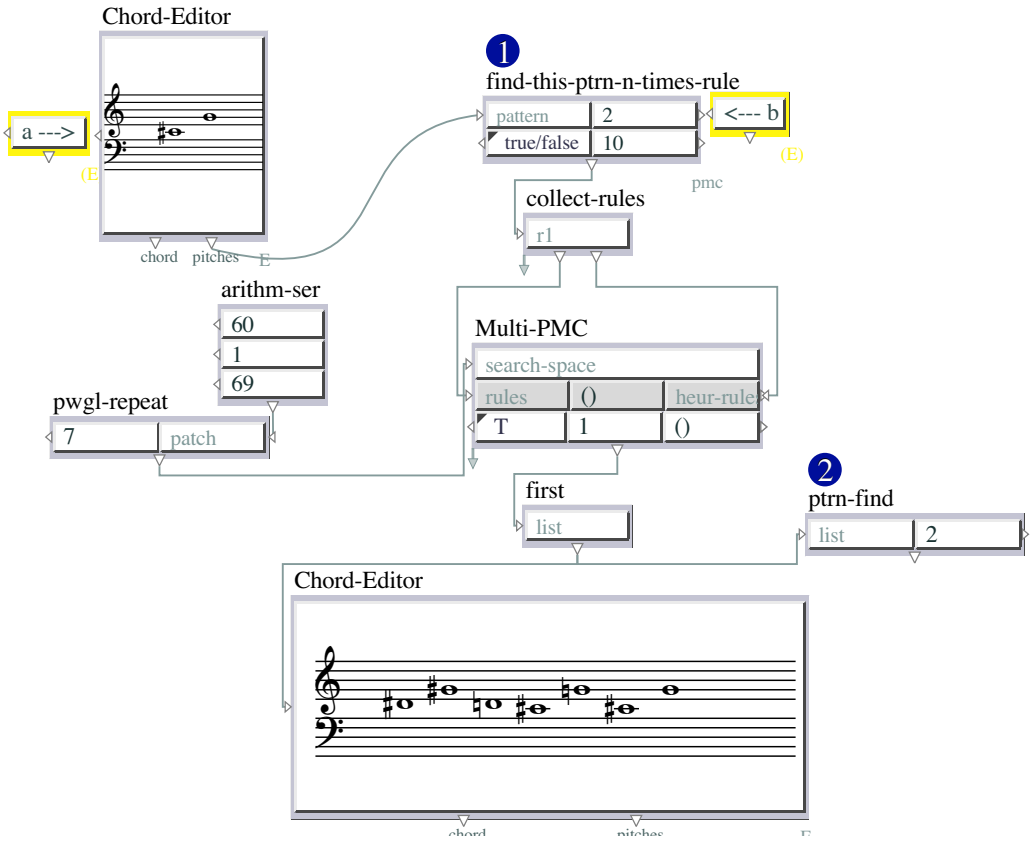

Figure 64: 1-05-02-find-this-ptrn-n-times-rule

2.6.4 03-More-First-Repeated-than-Second

1.05.03 - MORE-FIRST-REPEATED-THAN-SECOND-RULE
MORE-FIRST-REPEATED-THAN-SECOND-RULE [1] asks the engine to create a solution with the first element [a] repeated more times than the second [b].

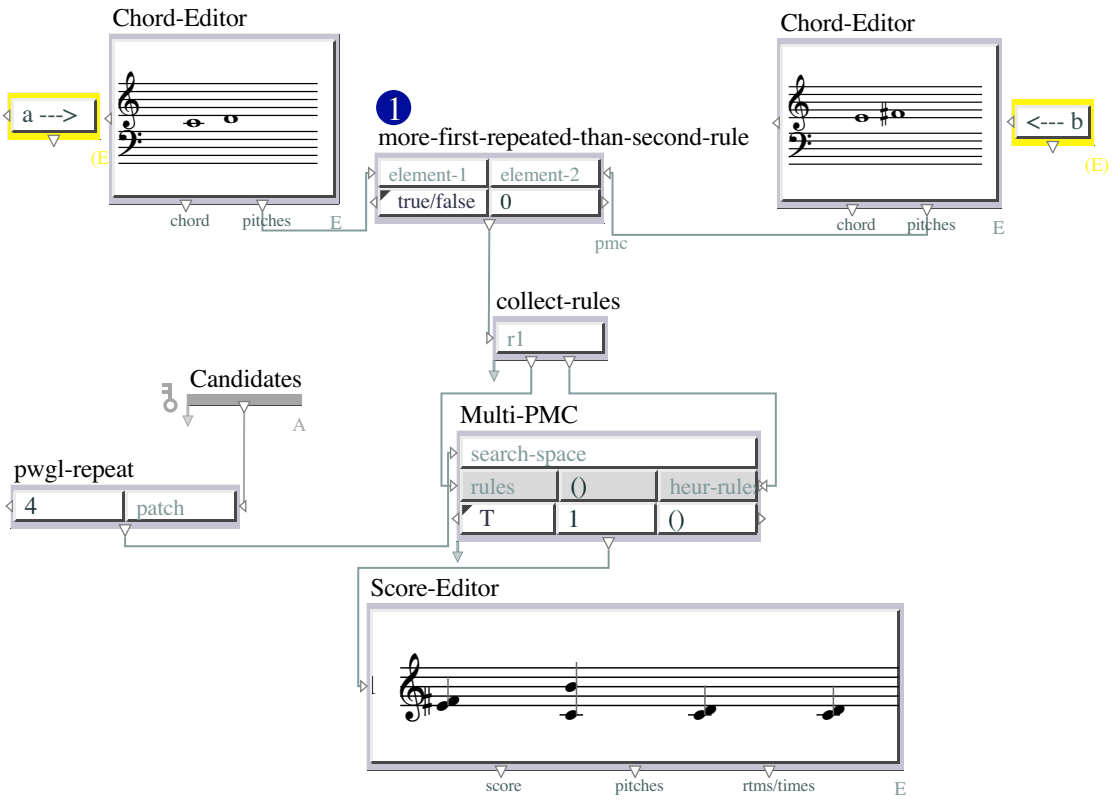

Figure 65: 1-05-03-more-first-repeated-than-second

2.6.5 04-Repeated-Pattern-Rule

1.05.04 - REPEATED-PATTERN-RULE
REPEATED-PATTERN-RULE [1] controls the amount of repetitions of a pattern given in [a] input.
In 'times' input [b] you put how many times you want the pattern to be repeated.
In 'which?' [c] you can choose among :
< means less times
= means an exact number of times
> means more times.
In 'absolute?' [d] you can choose if you are looking for positive elements (absolute) or not (up/down).
BE CAREFUL (full of care...) If you set [d] in the absolute mode, be sure that you are looking for positive values put in [a], otherwise you are committing a categorical mistake !

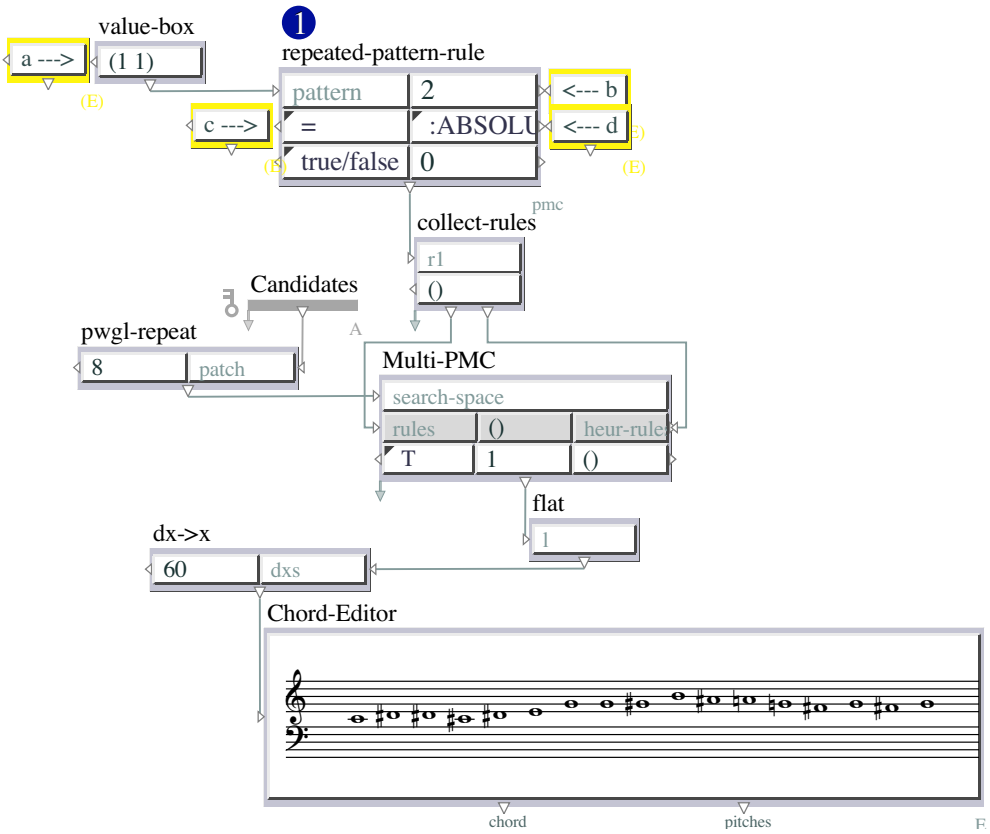

Figure 66: 1-05-04-repeated-pattern-rule

2.6.6 05-Always-More-Little-Included-Rule

1.05.05 - ALWAYS-MORE-LITTLE-INCLUDED-RULE
ALWAYS-MORE-LITTLE-INCLUDED-RULE [1] builds a solution in which, between two successive elements, the smaller is always included in the bigger.
Look at the 'Candidates' [2] abstraction. Here are generated all possible chords with the given lengths put in [a].

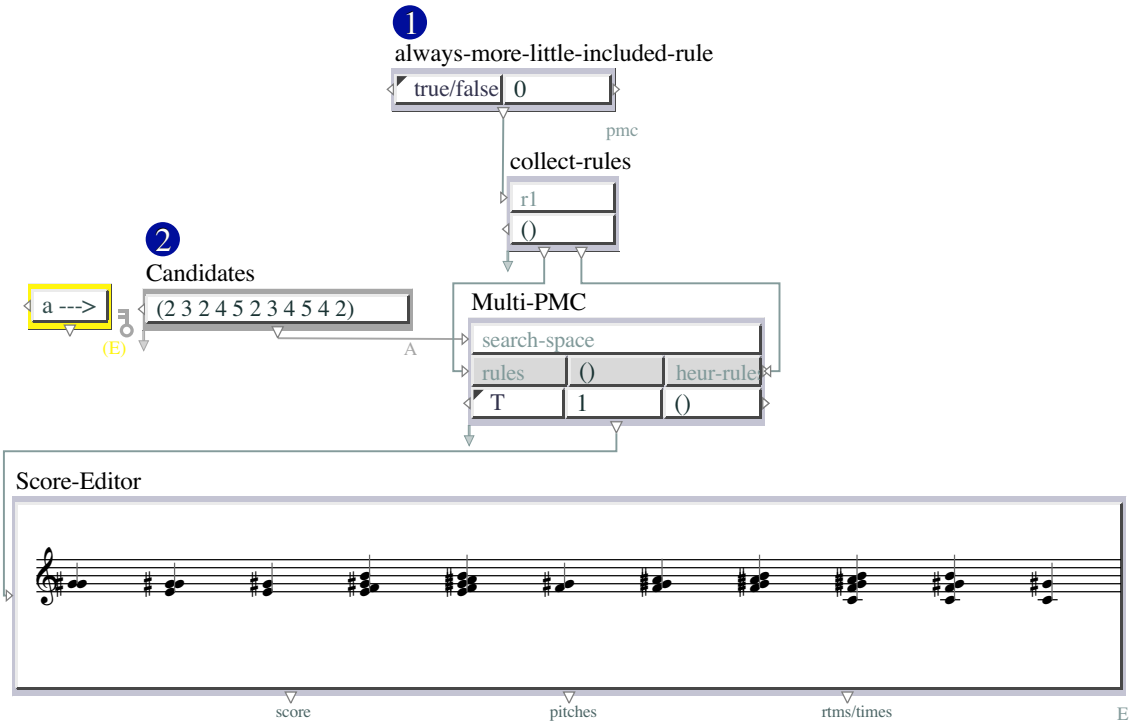

Figure 67: 1-05-05-always-more-little-included-rule

2.7 06-Distance-Rules

2.7.1 Distance-Rules

This menu has only one rule, which corresponds to the theory of morphological distance. Please look at the function distance in the Morphologie library for more details.

2.7.2 01-Distance-Rule

1.06.01 - DISTANCE-RULE
DISTANCE-RULE [1] is a morphological rule, based on the DISTANCE function of the Morphologie library.
It asks to the engine those solutions having a given 'distance' [a] with the pattern set in [b].
In 'which?' [c] you can chose if you want an equal distance '=', a smaller distance '<' or a bigger distance '>'.

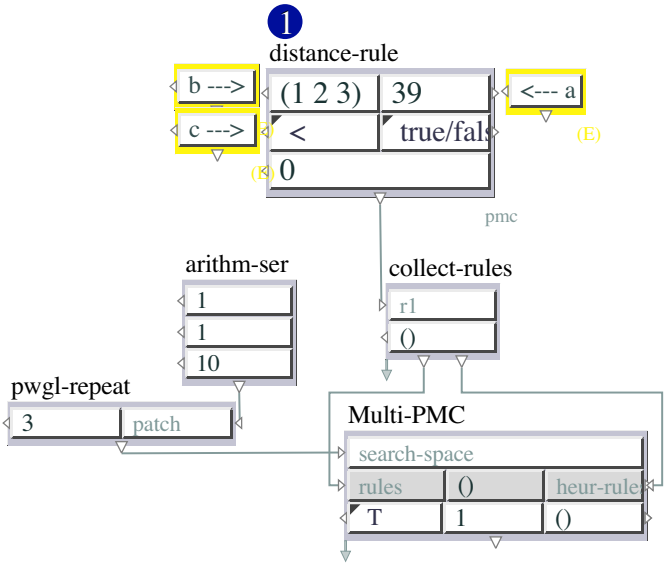

Figure 68: 1-06-01-distance-rule

2.7.3 02-Dynamic-Distance-Rule

1.06.02 - DYNAMIC-DISTANCE-RULE
This patch enlarges the possible results of the preceding one, still about DISTANCE-RULE based on the Morphogie library.
The aim is to produce a series of profiles, with the same length or not (this has to be defined with the SWITCH [1]), each with different distance in reference to a given profile [2].
The evolution of the distance through the profiles is defined in this case by a 2D-EDITOR [3]. There must be as many samples points taken from the bpf than length values for the profiles [a].
Both these values are going to the 'dynamic-distance-rule' abstraction [4] that creates as many Multi-PMC results than samples and length values. The model profile enters in the abstraction's third input and goes directly to the DISTANCE-RULE [b]
Here we chose to have a limited number of candidates [c], essentially a C-Major scale, in order to make the several researches faster and the results closer.
Please evaluate the CHORD-EDITOR [5] and look at the successive results by scrolling through the chords.

You can also control the results by evaluating the 'compare-distances' abstraction [6] and see if the result matches the result of the G-ROUND on the left.
Be careful, the distance-rule works better in '<' or '>' modes than in '=' mode. In this example, the '<' mode was prefered because it seems more intuitive for dynamic use, but you can use '>' mode as well. That's why the actual distances of the generated profiles (seen in 'compare-distances' abstraction) can be a little lower than the one you asked for.

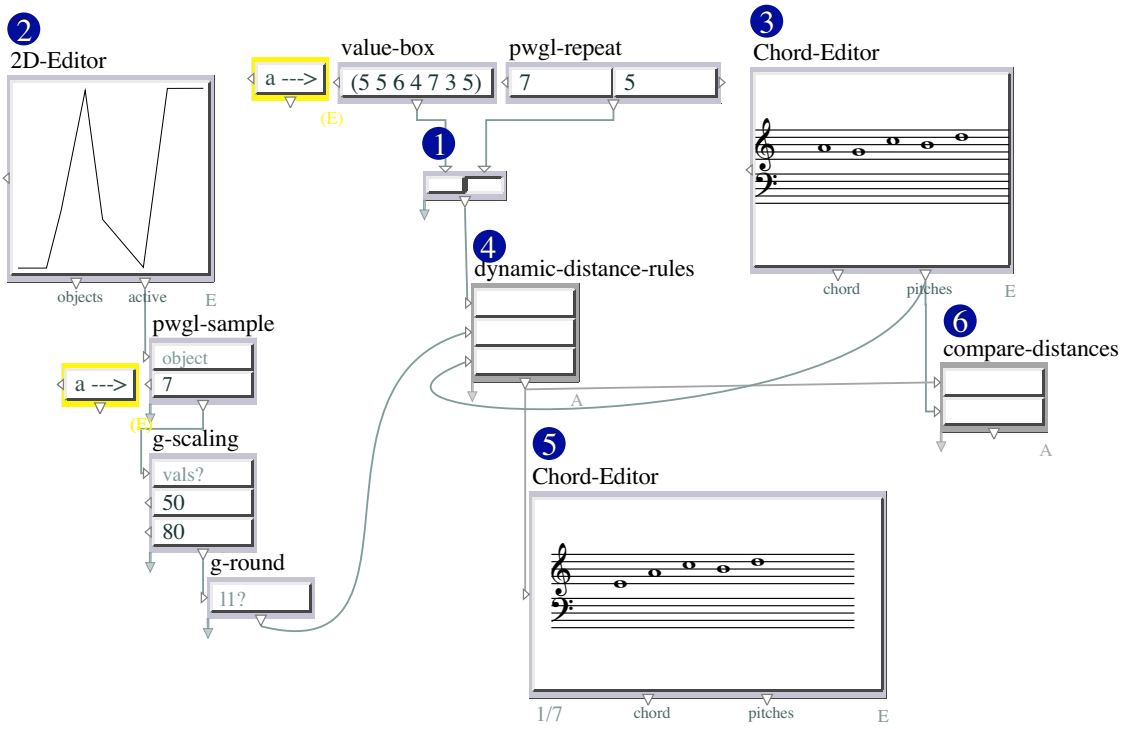

Figure 69: 1-06-02-dynamic-distance-rule

2.8 07-Structure-Rules

2.8.1 Structure-Rules

This part of the library is dedicated to the control of structures. The notion of a structure is hard to define in this kind of software. In general I make the difference between a list, a sequence and a structure. The interpretation of any phenomenon is articulated under the form of structures. It means here that the rules will generate some sequences but these sequences will be interpreted in function of a given context. In this sense, the same rule here can produce a structure that is meaningful for a rhythm sequence and another that is meaningful for an interval sequence.

2.8.2 01-Mk-Symbol-Structure-Rule

1.07.01 - FIND-APPLY-GLOBAL-AND-APPROX-SUM-RULE
MK-SYMBOL-STRUCTURE-RULE [1] asks the engine to put out a solution with a shape identical or similar to the list defined in 'profile' input [a].
I kept this rule, identical to MK-FIX-PROFILE-RULE, in order to let you know that a same concept can be useful in several musical domains.

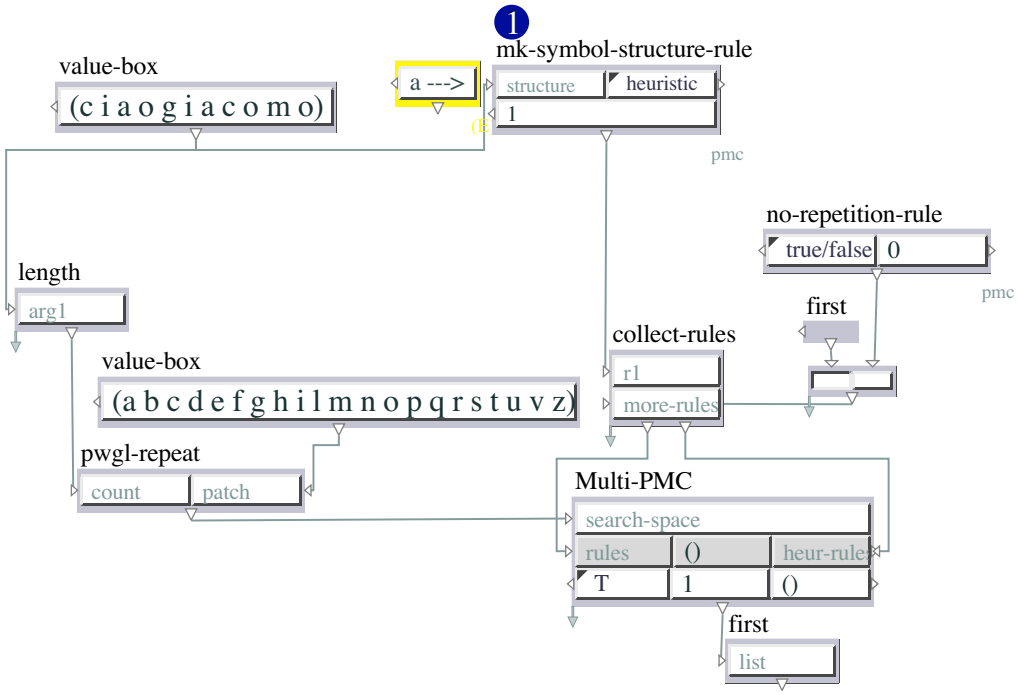

Figure 70: 1-07-01-mk-symbol-structure-rule

2.8.3 02-Find-Apply-Global-and-Approx-Sum-Rule

1.07.02 - FIND-APPLY-GLOBAL-AND-APPROX-SUM-RULE
FIND-APPY-GLOBAL-SUM-RULE [1] finds out a solution with elements having, summed together, the same value put in 'sum' input [a].
FIND-APPLY-GLOBAL-ABSOLUTE-SUM-RULE [2] does exactly the same but it considers also negative values.
FIND-APPLY-APPROX-SUM-RULE [2] does quite the same but it includes also a value of approximation (set in [b]), tolerated above or below the 'sum' value.
FIND-APPLY-GLOBAL-ABSOLUTE-SUM-RULE [4] does exactly the same but it considers also negative values.

Please use the SWITCH [c] in order to choose positive candidates only or both positive and negative candidates.

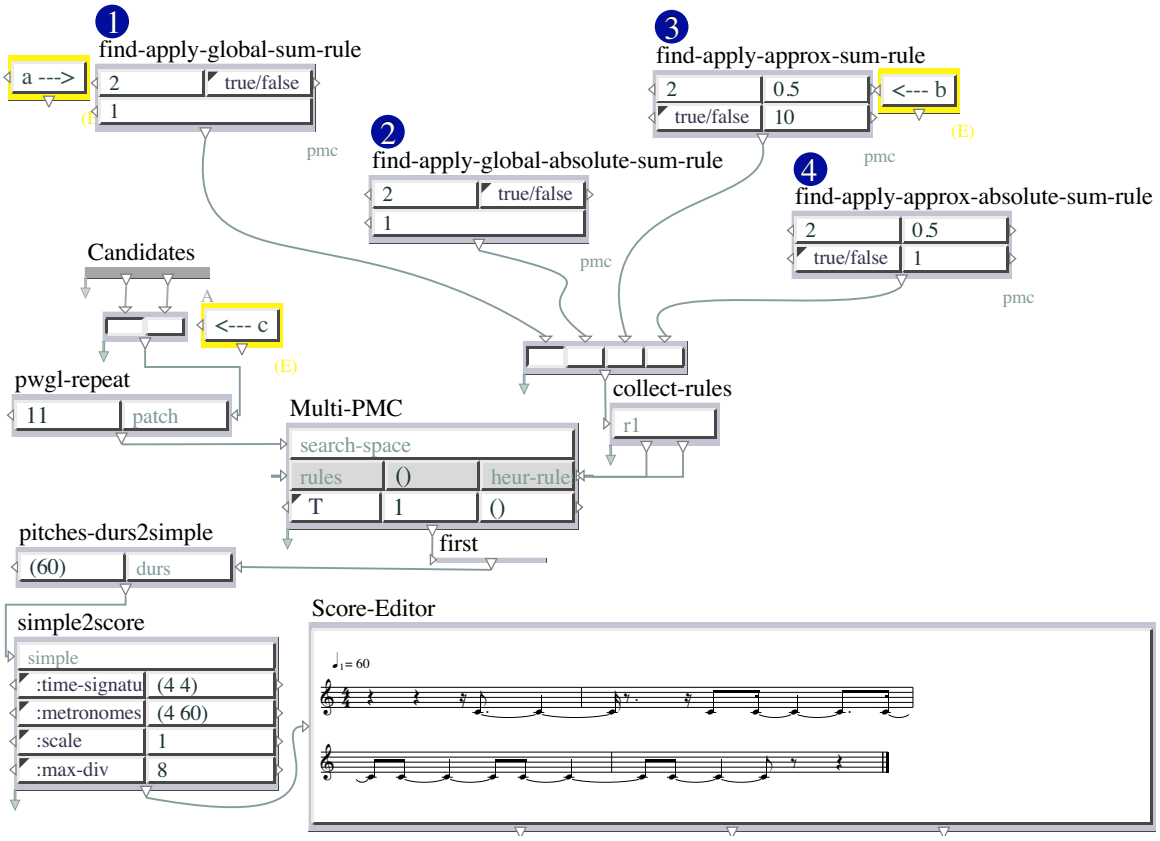

Figure 71: 1-07-02-find-apply-global-and-approx-sum-rule

2.8.4 03-Length-Sub-Group-Applied-Sum-Rule

1.07.03 - LENGTH-SUB-GROUP-APPLIED-SUM-RULE
LENGTH-SUB-GROUP-APPLIED-SUM-RULE [1] forces each sub-list of the solution to have an applied sum equal to the value set in 'length?' input [a].
You can control the result by evaluating the PWGL-MAP [2]. All the applied sum inside each sub-list is equal to the number set in [a].

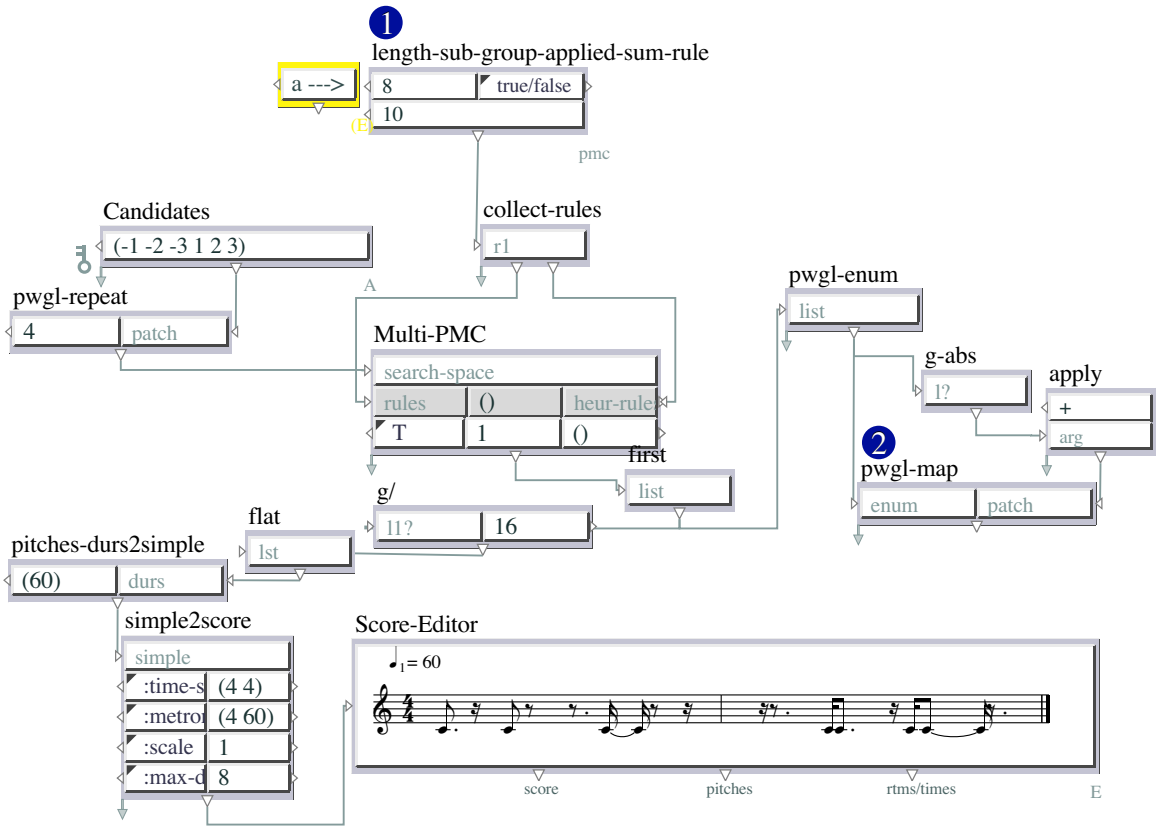

Figure 72: 1-07-03-length-sub-group-applied-sum-rule

2.8.5 04-Structured-Order-Sum-Rule

1.07.04 - STRUCTURED-ORDER-SUM-RULE
In candidates [a] you put, with Constraints symbols, the number of variables you are looking for. Ex. : ?1 ?2 ?3 Then in [b] you put a list of indexes that has to be applied by a POSN-MATCH function to the solution. Finally in sum [c] you put the value that the elements of the solution, once added together according to the POSN-MATCH order, have to produce.
STRUCTURED-ORDER-SUM-RULE [1] is very useful when you know the length of a metrical musical passage. In this case let say that you have to fill 32 1/16 (it means two full measure of (4 4)), using only three different values [a] which, summed together, must follow a given order [b].
You can control the result by evaluating the G/ function [2].

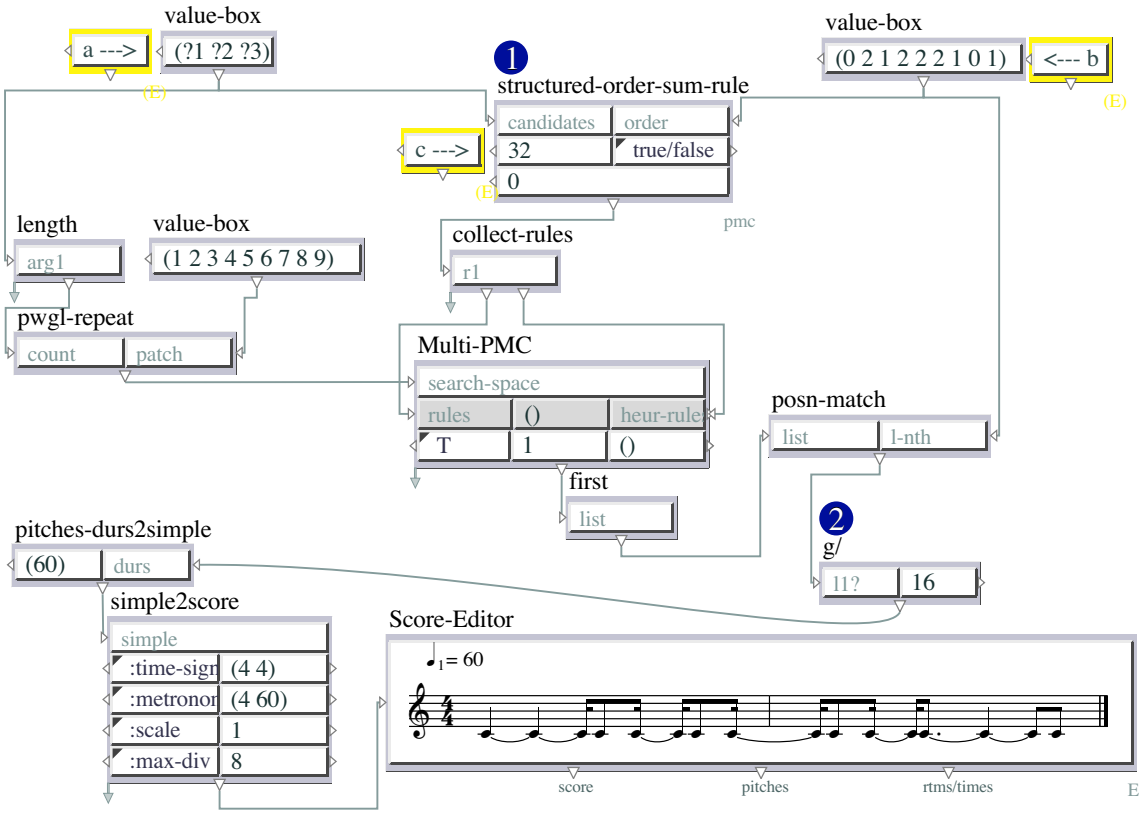

Figure 73: 1-07-04-structured-order-sum-rule

2.8.6 05-Count-Positive-and-Negative-Rule

1.07.05 - COUNT-POSITIVE-RULE
COUNT-POSITIVE-RULE [1] obliges the solution to include a number of positive values identical to its 'number' value [a].
COUNT-NEGATIVE-RULE [2] does exactly the same with negative values.

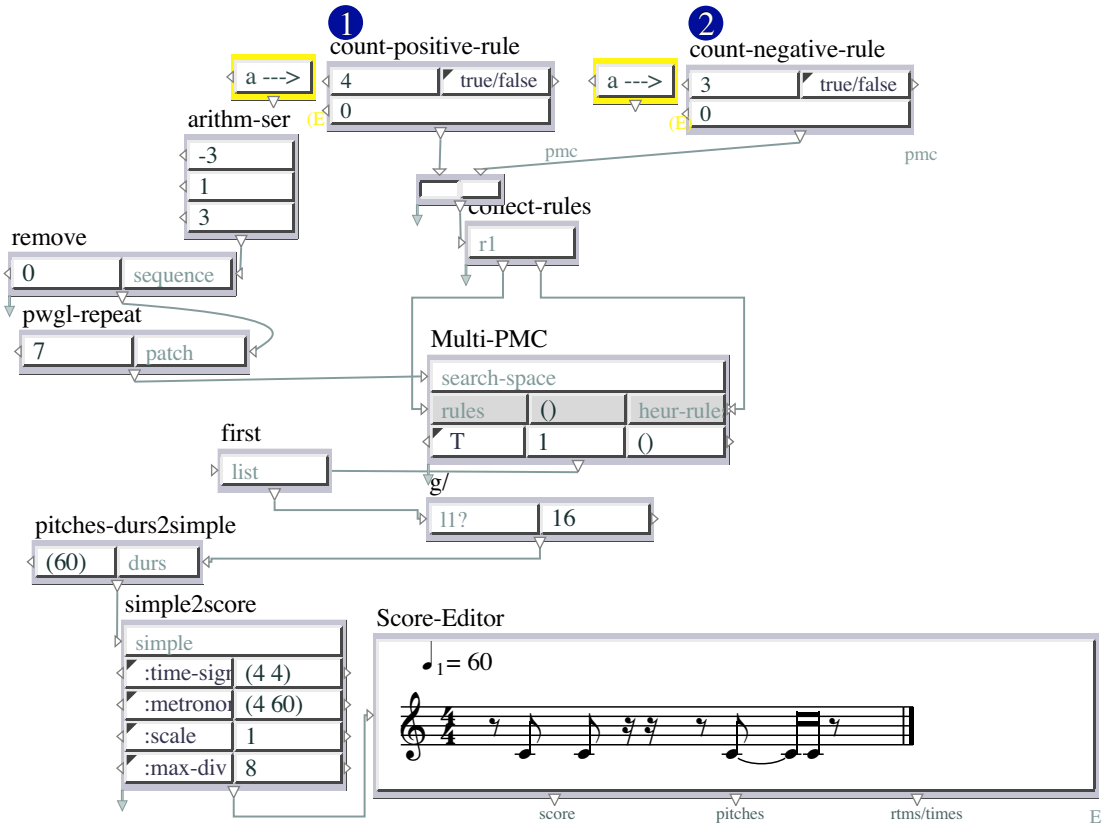

Figure 74: 1-07-05-count-positive-and-negative-rule

2.8.7 06-No-Consecutive-Rests-or-Pulses-Rule

1.07.06 - NO-CONSECUTIVE-RESTS-OR-PULSES
NOT-CONSECUTIVE-RESTS-RULE [1] forbids consecutive negatives values.
NOT-CONSECUTIVE-PULSES-RULE [2] forbids consecutive positive values.

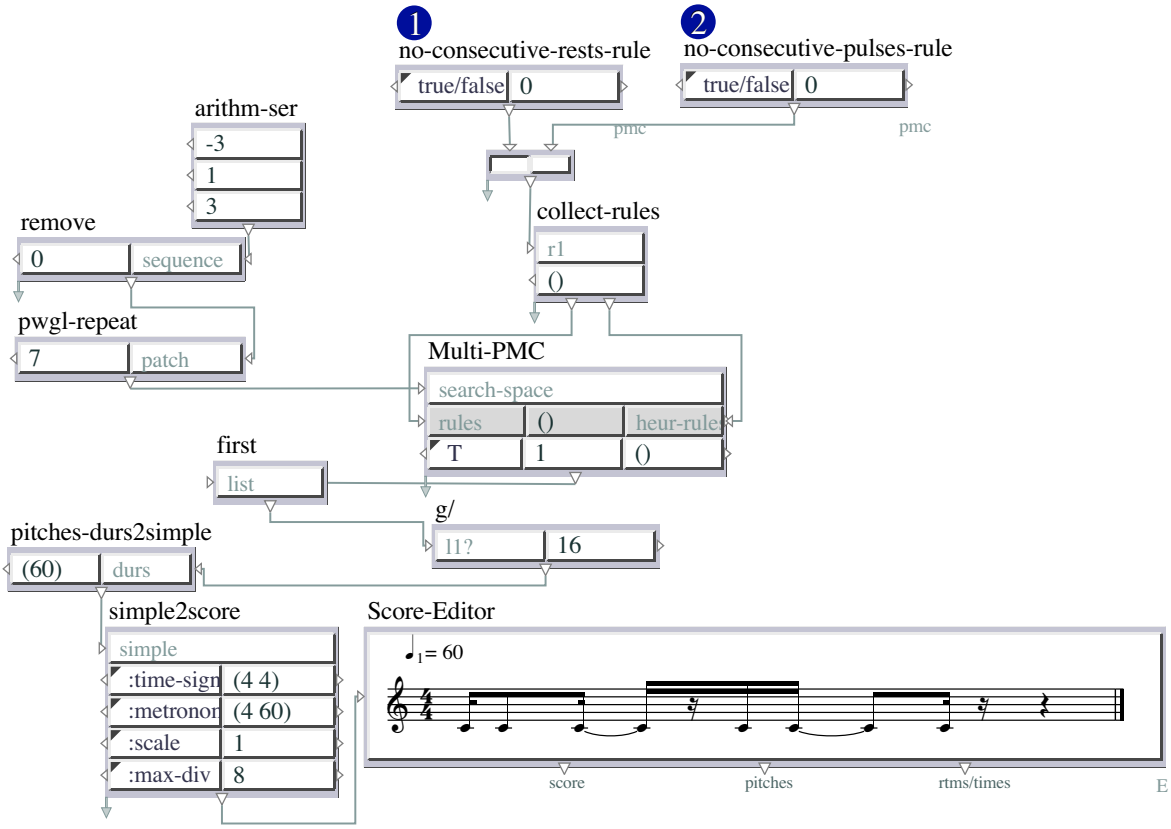

Figure 75: 1-07-06-no-consecutive-rests-or-pulses-rule

2.8.8 07-Alternating-Positive-Negative-Rule

1.07.07 - ALTERNATING-POSITIVE-NEGATIVE-RULE
ALTERNATING-POSITIVE-NEGATIVE-RULE [1] obliges a positive number to be followed by a negative number and vice versa.

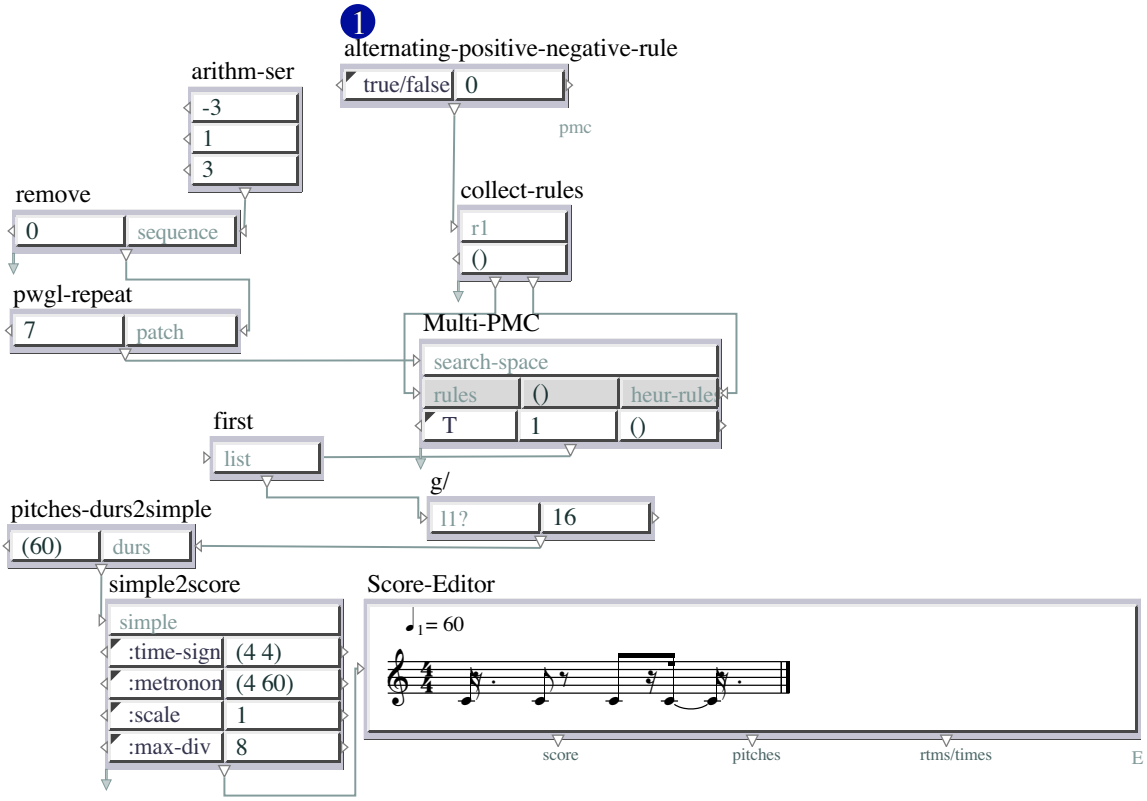

Figure 76: 1-07-07-alternating-positive-negative-rule

2.8.9 08-Alternating-plus-minus-First-or-Last-Elmt-Rule

1.07.08 - ALTERNATING-+/-FIRST-ELMT-RULE
ALTERNATING-+/-FIRST-ELMT-RULE [1] creates a list of sub-lists in which the FIRST element is, alternatively, before positive then negative an so on.
The ALTERNATING-+/-LAST-ELMT-RULE [2] creates a list of sub-lists in which the LAST element is, alternatively, before positive then negative an so on.
ATTENTION This rule requires list of lists as candidates to work. You can control the result by evaluating the G/ function [3]

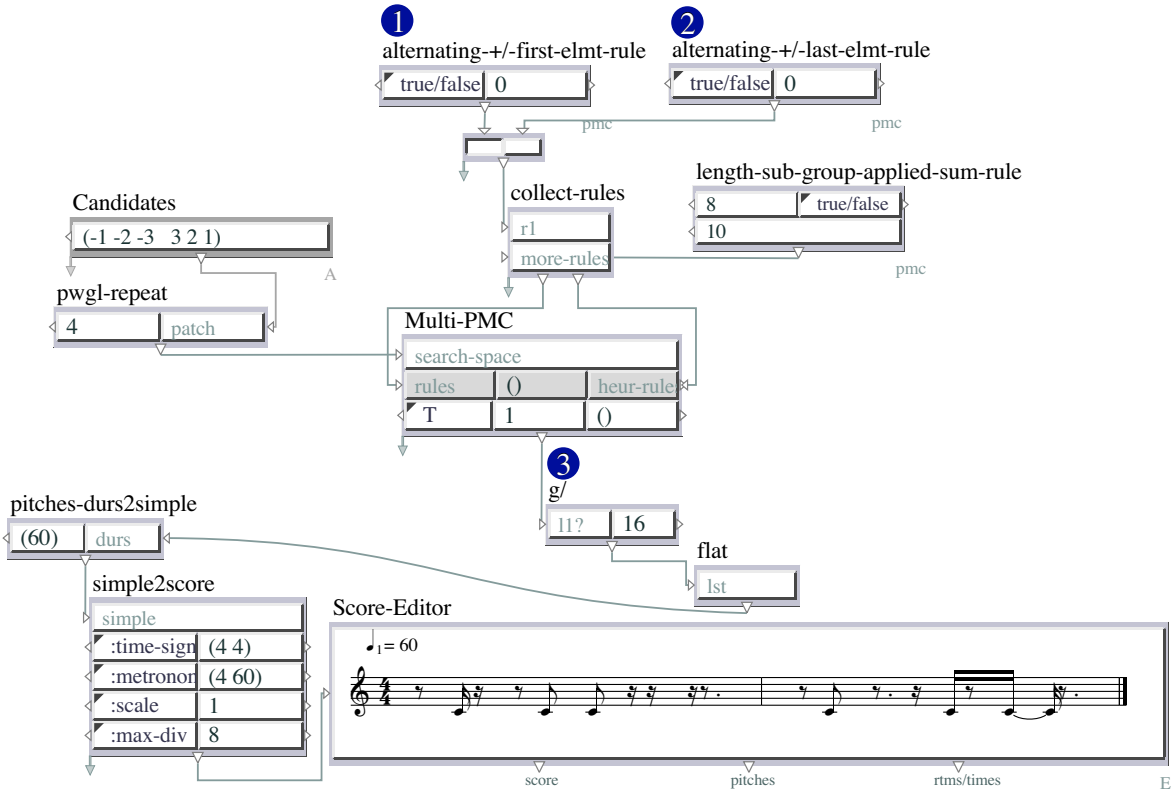

Figure 77: 1-07-08-alternating-plus-minus-first-or-last-elmt-rule

2.8.10 09-Structure-Identity-Rule

1.07.09 - STRUCTURE-IDENTITY
TO BE DISCUSSED WITH ORJAN SANDRED

to be discussed with Orjan Sandred energy-profile rule

Figure 78: 1-07-09-structure-identity-rule

2.9 08-Matrix-Rules

2.9.1 Matrix-Rules

This menu contains only two rules dealing with matrices. A matrix is a list of lists. A matrix is also a rectangular arrangement of numbers, where the horizontal and vertical lines are respectively called rows and columns.

2.9.2 01-Mk-Latin-Matrix-Rule

1.08.01 - MK-LATIN-MATRIX-RULE
MK-LATIN-MATRIX-RULE [1] creates a latin matrix like following :
1 3 2 4
2 1 4 3
3 4 1 2
4 2 3 1
What does it mean?... Simply that ther is no repetition in columns and in raws.

ATTENTION : It works with list of lists. So please look at the 'Candidates' abstraction [a].

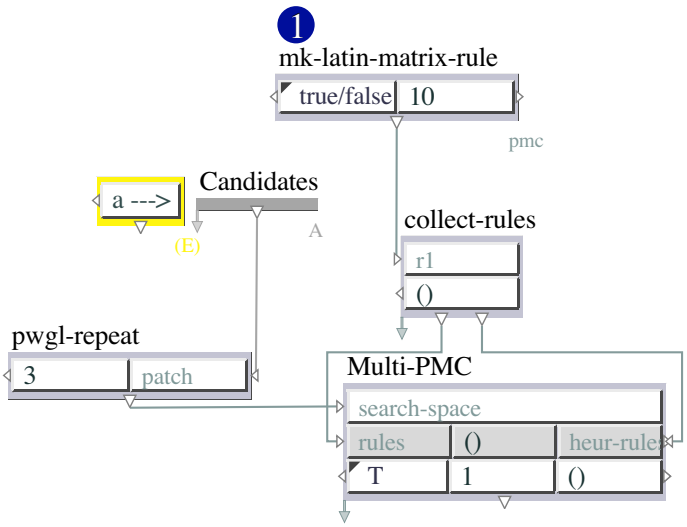

Figure 79: 1-08-01-mk-latin-matrix-rule

2.9.3 02-Chain-Common-Element-Lists-Rule

1.08.02 - CHAIN-COMMON-ELEMENT-LIST-RULE
CHAIN-COMMON-ELEMENT-LIST-RULE [1] looks for solutions of lists of lists in which the end of a sub-list in equal to the beginning of the following list.
First, open the abstraction [2] called in this patch 'LISTS'. Can you understand it or no? If not, please go back to the tutorials 1.01 and re-do them again. In any case this abstractions asks for an arbitrary number of solutions (defined in with the PWGL-REPEAT [3]) in which the first element has to be either 60 or 61, without any repetitions, and the length of each solution is 7.
In [a] you set HOW MANY LAST elements of a sub-list have to be included in the HOW MANY FIRST elements of the following one.
In this case I put 2. That means that the 2 last elements of the first sub-group list have to be equal with the 2 first elements of the second sub-list; then the 2 last elements of the second sub-list have to be equal to the 2 last elements of the third list, and so on.

If you evaluate the CHORD-EDITOR [5], you can see each sub list (one by one) throughout the CHORD-EDITOR option using your up and down keyboard shortcuts.
If you evaluate the CHORD-EDITOR [6] you can see all the solutions in one piece.
ATTENTION : IN THE HEURISTIC MODE IT DOES NOT WORK. WHY???

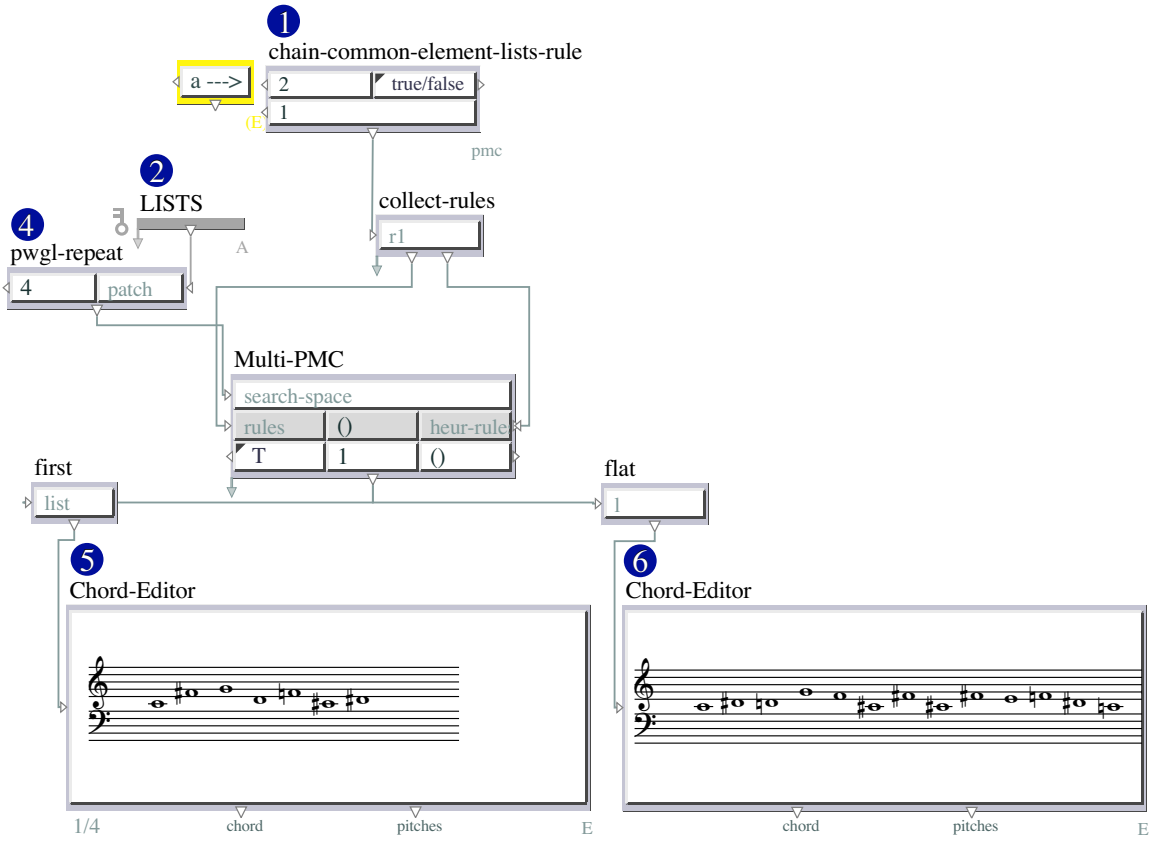

Figure 80: 1-08-02-chain-common-element-lists-rule

2.9.4 03-Chain-More-Little-Common-Rule

1.08.03 - CHAIN-MORE-LITTLE-INCLUDED-COMMON-LISTS-RULE
CHAIN-MORE-LITTLE-INCLUDED-COMMON-LISTS-RULE [1] looks for solutions of lists of lists in which the end of a sub-list in equal to the beginning of the following sub-list. But in this case the length of the first elements and the last elements can be different.
First, open the abstraction [2] called in this patch LISTS. Can you understand it or no? If not, please go back to the tutorials 1.01 and re-do them again. In any case this abstractions asks for an arbitrary number of solutions (defined with the PWGL-REPEAT [3]) in which the first element has to be either 60 or 61, without any repetitions, and the length of each solution is 7.
In [4] you define how many sub-lists you want to chain. (In this example they are 4).

In [b] you set HOW MANY LAST elements of a sub-list have to be included in the HOW MANY FIRST [a] elements of the following one. If [a] and [b] are equal, this rule [1] looks for solutions having the same elements at the end of a sub-list to be included in the beginning of the following one. The order of the common elements can be any.
If [a] is more little than [b], it means that all the elements in [a] have to be included even within some elements that do not belong to [b]. If is [b] the more little, it means that all the elements in [b] have to be included even within some elements that do not belong to [a].
If you evaluate the CHORD-EDITOR [5], you can see each sub-list (one by one) throughout the CHORD-EDITOR option using your up and down keys.
If you evaluate the CHORD-EDITOR [6] you can see all the solutions in a one piece.
ATTENTION: IN THE HEURISTIC MODE IT DOES NOT WORK. WHY???

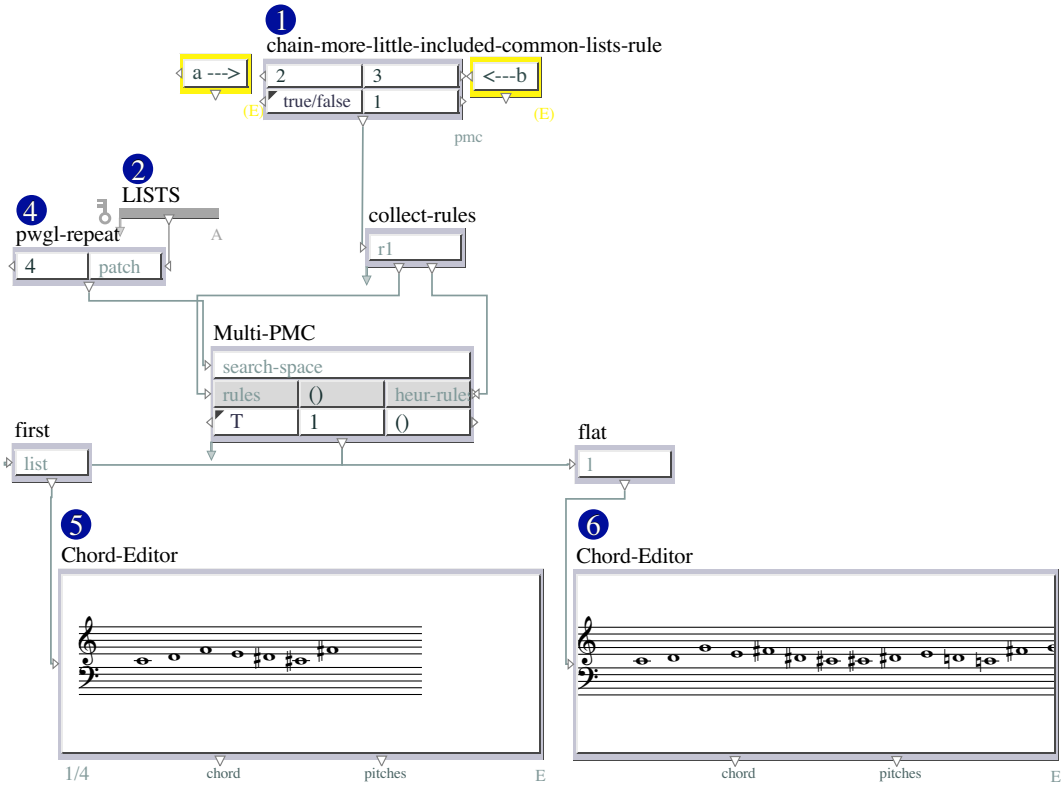

Figure 81: 1-08-03-chain-more-little-common-rule

3 0-Multi-Score-PMC

3.1 00-Introduction-to-Score-PMC

3.1.1 Introduction

This section is dedicated to explain how the Multi-Score-PMC works.

3.1.2 1-The-Rules-for-Multi-Score-PMC

2.00.1 - THE-RULES-FOR-MULTI-SCORE-PMC
The Multi-Score-PMC is a complex system dedicated to solve polyphonic and counterpoint musical problem. If you need to look for solutions for only one voice, the Multi-PMC is sufficient, and even more efficient. However, as you'll see in the next patches, to generate musical expressions with rules you need the Multi-Score-PMC.
BE CAREFUL Sometimes the syntax of the Multi-Score-PMC seems to be the same as the Multi-PMC. THIS IS NOT THE CASE. That's why the rules for the Multi-PMC are called just with a name invoking the related concept. On the contrary, the names of the Multi-Score-PMC rules are always preceded by S-PMC.
Look at the two rules in this patch. [1] is the rule for the Multi-PMC called ALLOWED-INTERVALS-RULE. [2] is the rule for the Multi-Score-PMC called S-PMC-ALLOWED-INTERVALS-RULE. If you use the Multi-Score-PMC for a single voice, the result of these two rules is strictly the same, but in this case it would be better to use the Multi-PMC.
As for all rules of this library, there is a true/false or heuristic menu [a]. If this menu is set in the true/false mode the Multi-Score-PMC will apply strictly the given rule. On the contrary, if the menu is set on heuristic the Multi-Score-PMC will apply the rule as much as possible, according to its weight [b]. (PLEASE SEE ALSO (AND AGAIN) THE PATCH 1.00.3-The-logical-conflict-and-the-heuristic-rules)
ATTENTION If you don't have studied yet the Multi-PMC, you have no chance to understand the Multi-Score-PMC. So, it is the case, please start this tutorial from the beginning.

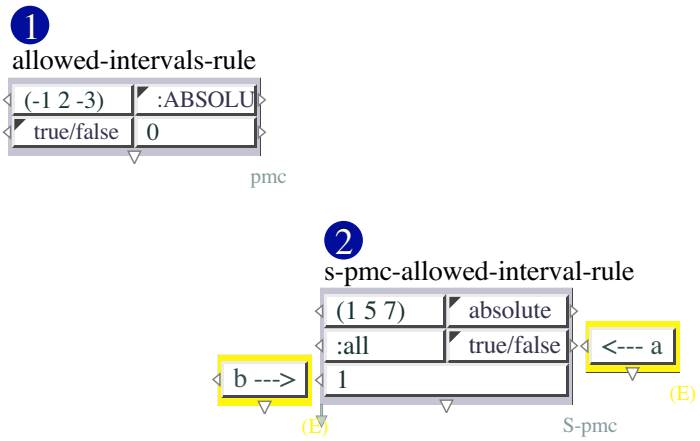

Figure 82: 2-00-1-The-rules-for-Multi-Score-PMC

3.1.3 2-The-Multi-Score-PMC

2.00.2 - THE-MULTI-SCORE-PMC
Here is the Multi-Score-PMC [1].
In [a] you have to connect the 'score' output of a SCORE-EDITOR that will provide the rhythmical structure of the searched solution.
In [b] you have to connect another SCORE-EDITOR (with its 'score' output) in which the solution will be stored and displayed.
In [c] you have to put the search-space.
In [d] you put the true/false rules.
In [e] the you put the heuristic rules.
In [f] you define if you want a random (T) or a sorted search ().
In [g] you choose how many solutions you are looking for.
In [h] you can set some particular data base.

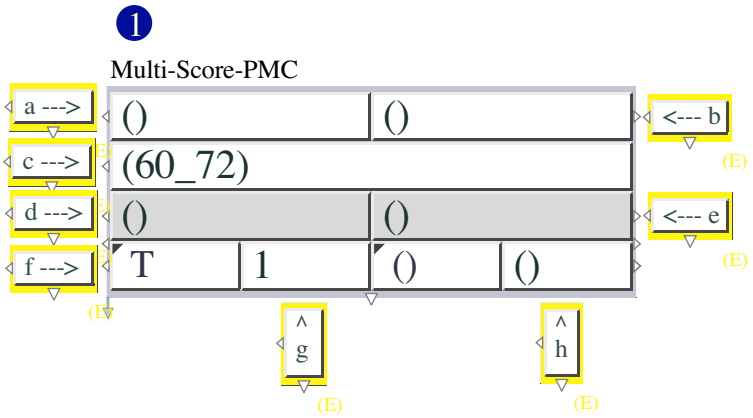

Figure 83: 2-00-2-The-Multi-Score-PMC

3.1.4 3-Multi-Score-PMC-Stantdard-Patch

2.00.3 - MULTI-SCORE-PMC-STANDARD-PATCH
Here I suggest you how to dispose the boxes in order to have as much as possible control on the entire process to find solutions.
You have to accept (at this level) that the rhythm of the solution is given by a SCORE-EDITOR. The rhythm can be provided by other calculations, or you could also have written it by hand, within the SCORE-EDITOR. So [1] is just a sort of default starting value. Without it the Multi-Score-PMC doesn't work.
In the other SCORE-EDITOR [2] you'll see the result when you evaluate the Multi-Score-PMC. One of the main differences between the Multi-Score-PMC and the Multi-PMC is that the first outputs its musical result (rhythm, pitches and expressions) in the SCORE-EDITOR on the right [2] and not with its own output.
The COLLECT-RULES [3] works exactly as for the Multi-PMC. It is the collector for all the rules you need. The order of insertion of the rules, or empty inputs (nil) do not affect the result. The right output is connected to the true/false rule input and the left to the heuristic rule input of the Multi-Score-PMC.
In the value-box [4] you can set the pitch search space in the 'expand-list' syntax. It

means that if you put (60_72) the value box will output this arithmetic series : (60 61 62 63 64 65 66 67 68 69 70 71 72).
Please evaluate the Multi-Score-PMC [5] and look at the right SCORE-EDITOR [2] in order to see the result. As in this patch the random mode [a] is set on T, for each evaluation you get a random research.
ATTENTION : I suggest you this configuration at the very beginning. Later we will see some variations concerning abstractions for rules or candidates.

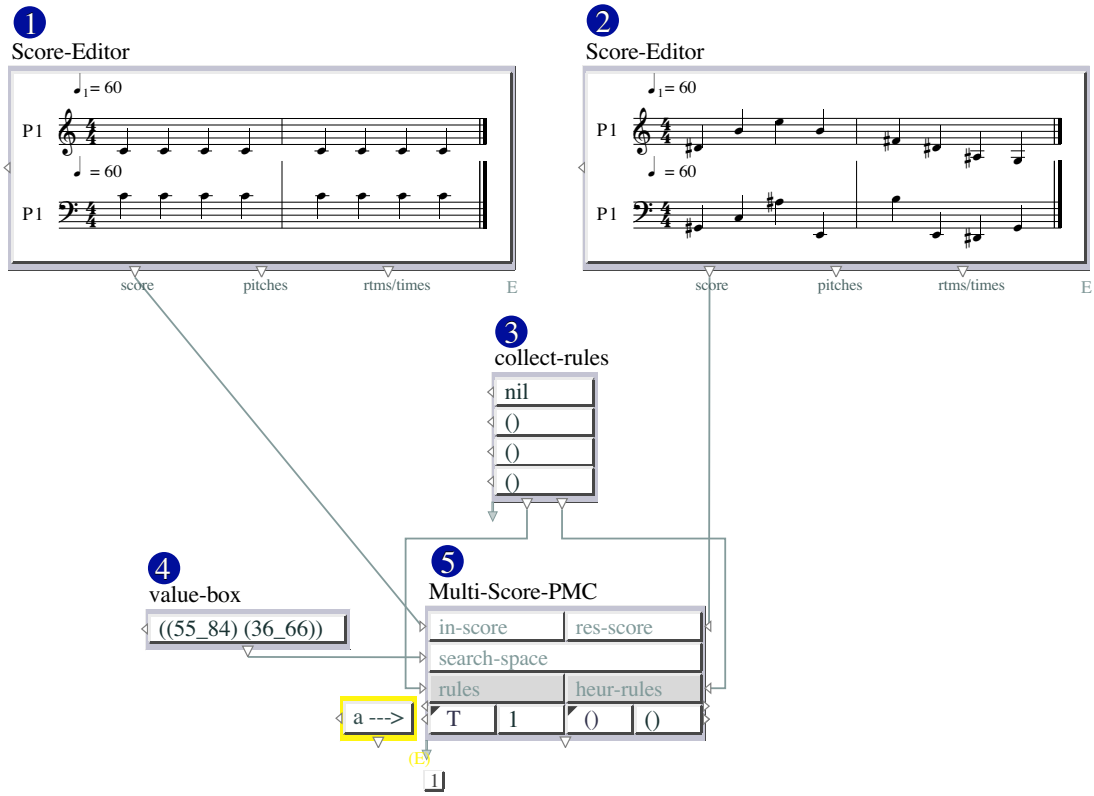

Figure 84: 2-00-3-Multi-Score-PMC-stantdard-patch

3.1.5 4-S-PMC-Rule-Voice-Attribution

2.00.4 - S-PMC-RULE-VOICE-ATTRIBUTION
This patch shows how to apply a rule to one specific voice or to all voices. As before, the left SCORE-EDITOR [1] is just needed to give a rhythmical structure for the result. The right SCORE-EDITOR [2] prints out the result.
In the VALUE-BOX [3] you define the search-space for the two voices. BE CAREFUL : The first sub-list defines the highest voice, the second defines the lowest.
In COLLECT-RULES [4] you can see the same S-PMC-INDEX-RULE [5] and [6] used twice.

Use the SWITCH [c] to activate the higher S-PMC-INDEX-RULE [5] and note that with the option :all [a], the two voices are forced to have a C (midi note 60) as a first note. Then, turn it off and activate the lower rule, with the SWITCH [d] When you evaluate the Multi-Score-PMC you'll see that the rule is only applied to the second voice [b], so that the first note of the bass voice is a C (midi note 60).

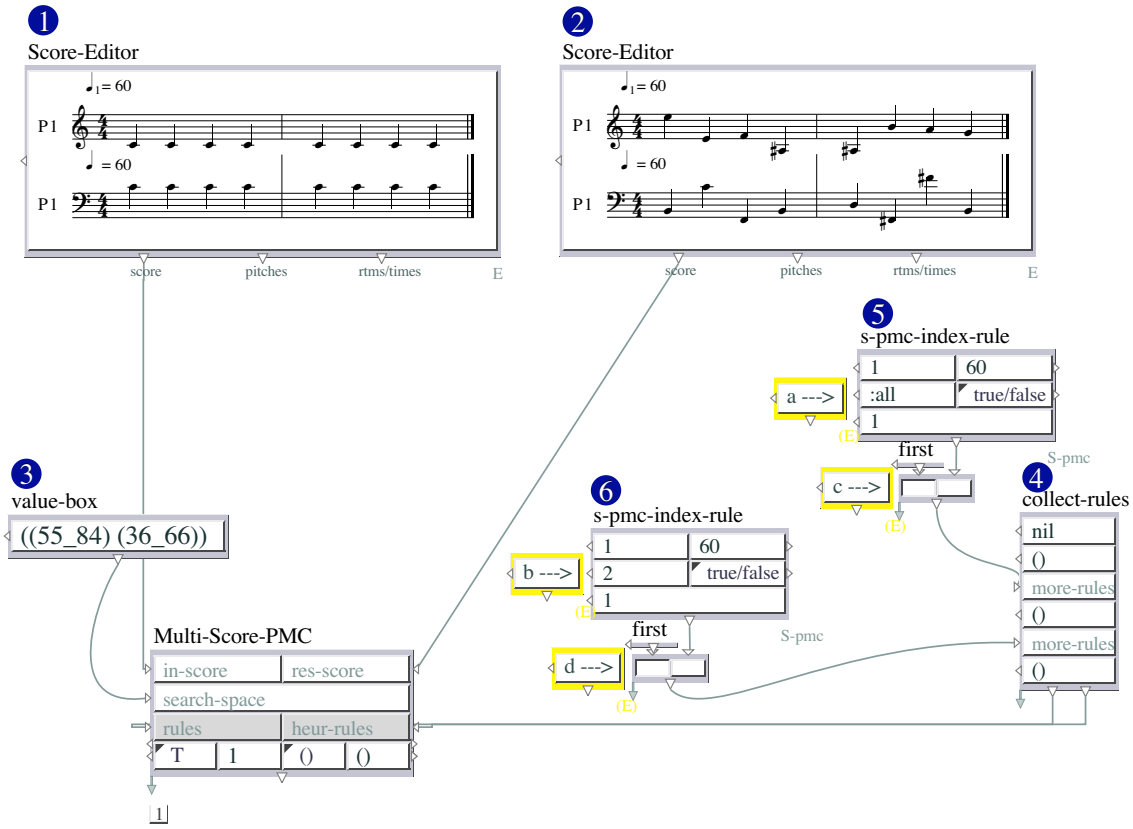

Figure 85: 2-00-4-s-pmc-rule-voice-attribution

3.1.6 5-S-PMC-Rule-Expressions-Recognition

2.00.5 - S-PMC-RULE-EXPRESSIONS-RECOGNITION
This patch shows you how to apply a rule to a specific voice or to all voices but only when some expressions have been previously defined. In this case, as before, the SCORE-EDITOR [1] gives only the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result.
In the VALUE-BOX [3] you define the search-space for the two voices. BE CAREFUL : The first sub-list defines the highest voice, the second defines the lowest.
S-PMC-ALLOWED-PITCH-RULE [4] forces all voices (defined in [b] with :all) just to include the notes C, D and E (set with midi values 60 62 64 [a]), but only when the expression :group [c] has been defined in the SCORE-EDITOR [1]. Attention, the C, D

and E note can be in any octave.
S-PMC-ALLOWED-INTERVAL-RULE [5] obliges only the first voice (the highest, set in [e]) to include minor and major seconds [d], but only when the expression :group has been defined in the SCORE-EDITOR [1].
Please evaluate the Multi-Score-PMC and look at the result. For the first voice (the highest), in correspondance with the :group expression, the C, D and E note can appears in any octave but always with adjacent intervals. For the bass voice these same notes can be anywhere with any interval.

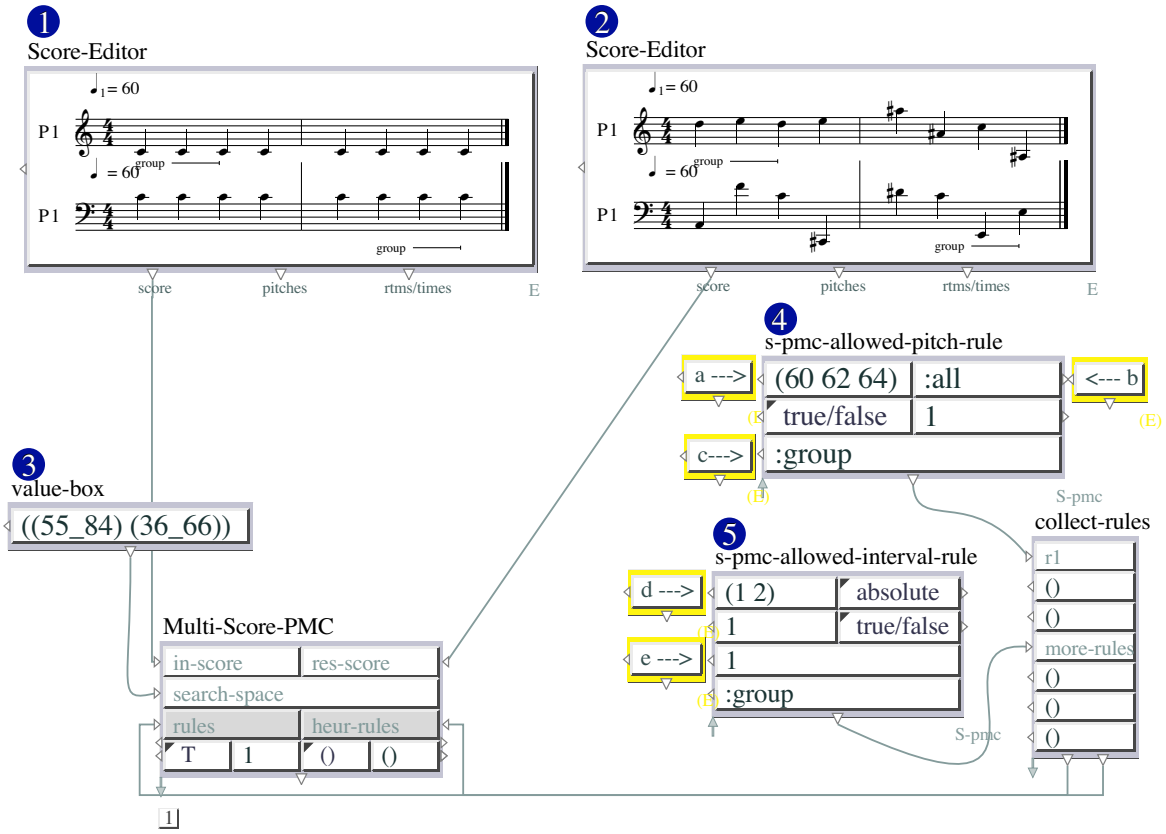

Figure 86: 2-00-5-s-pmc-rule-expressions-recognition

3.1.7 6-Logical-Conflict-between-Rules

2.00.6 - LOGICAL-CONFLICT-BETWEEN-RULES
If two rules are in conflict the Multi-Score-PMC outputs nil in the PWGL output.
In this patch two rules are in a logical conflict : we ask to the Multi-Score-PMC to find out a solution with only the C, D and E notes [4] in any octave of the search space (set in [a]) but at the same time we forbid [5] the solution NOT to have the same notes (set in [b]).

In this case no solution can be found, but in the SCORE-EDITOR's [2] last initialisation stays still printed out.
So BE CAREFUL ! When you see that in the printing out SCORE-EDITOR the result does not have any change look also to the PWGL output. The reason can be double : there is a logical conflict between rules (so in the PWGL output there is nil) or there is only one possible result.
Open the ONLY-ONE-POSSIBLE-RESULT abstraction [7] and look at the solution. When you evaluate the Multi-Score-PMC, the SCORE-EDITOR's printed result [2]never change. In this case there is no logical conflict but only one possible solution. If you look at the PWGL output you will see that instead of nil you find the pitches of the result : ((((60 62 60 62 60 62 60 62) (60 62 60 62 60 62 60 62)))).

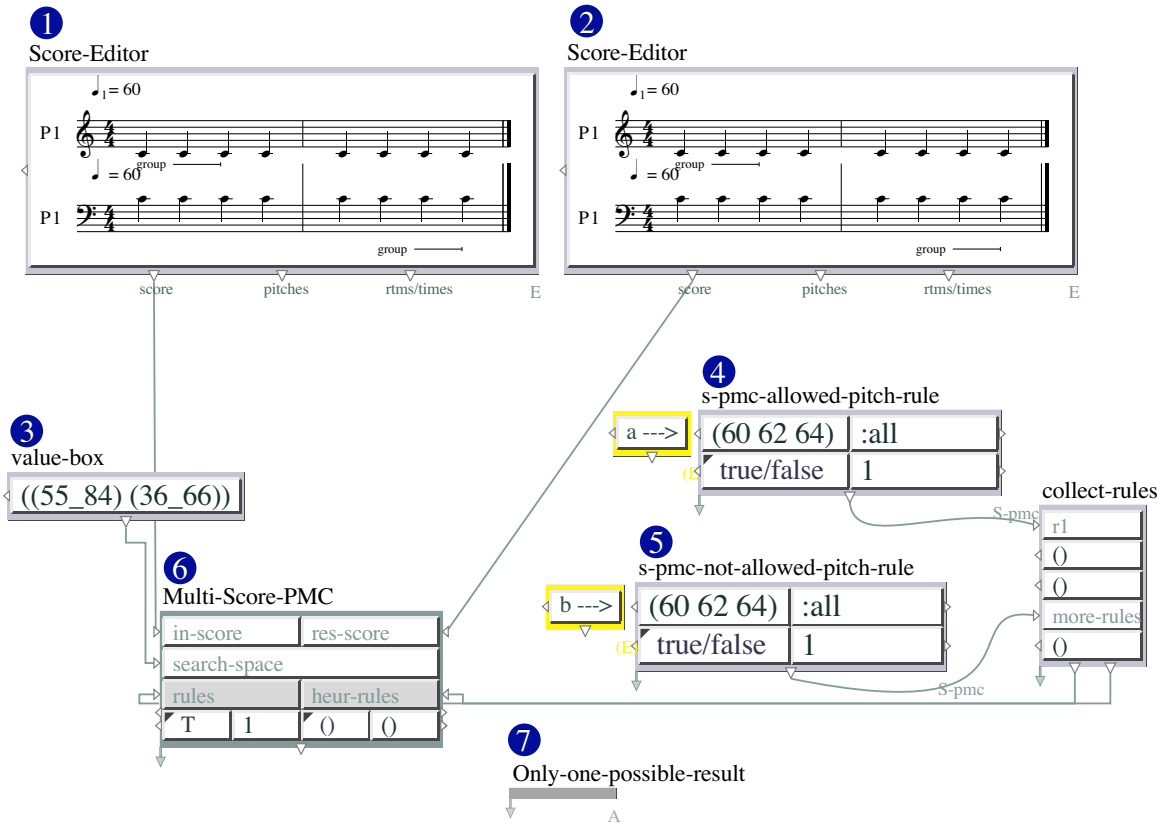

Figure 87: 2-00-6-logical-conflict-between-rules

3.2 01-Melodic-Rules

3.2.1 1-Generic-Poly-Rules

3.2.1.1 1-Several-Index-Rules

2.01.1.1 - S-PMC-SEVERAL-INDEX-RULES

This patch shows several index rules for polyphonic parts. The SCORE-EDITOR [1] only gives the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result. In the VALUE-BOX [3] you define the search-space for the two voices. The first sub-list is for the highest voice, the second for the lowest.
Apply S-PMC-INDEX-RULE [5] with the switch box (see previous patches). This rule obliges :all solutions to have the first element (index 1 set in [a]) equal to 60.
S-PMC-INDEX-HIGHER-RULE forces an element, according to its index, to be higher than 60 (in this example) for a given part. In this case it is set for all parts. (See also patch 2.00.4 - S-PMC-RULE-VOICE-ATTRIBUTION).
S-PMC-INDEX-LOWER-RULE forces an element, according to its index, to be lower than 60 (in this example) for a given part.
Please evaluate the Multi-Score-PMC and look at the results. Change the SWITCHES in order to apply only one rule at a time Attention, if two rules are in a logical conflict the result is nil, but in the SCORE-EDITOR [2] the last result stays printed out.

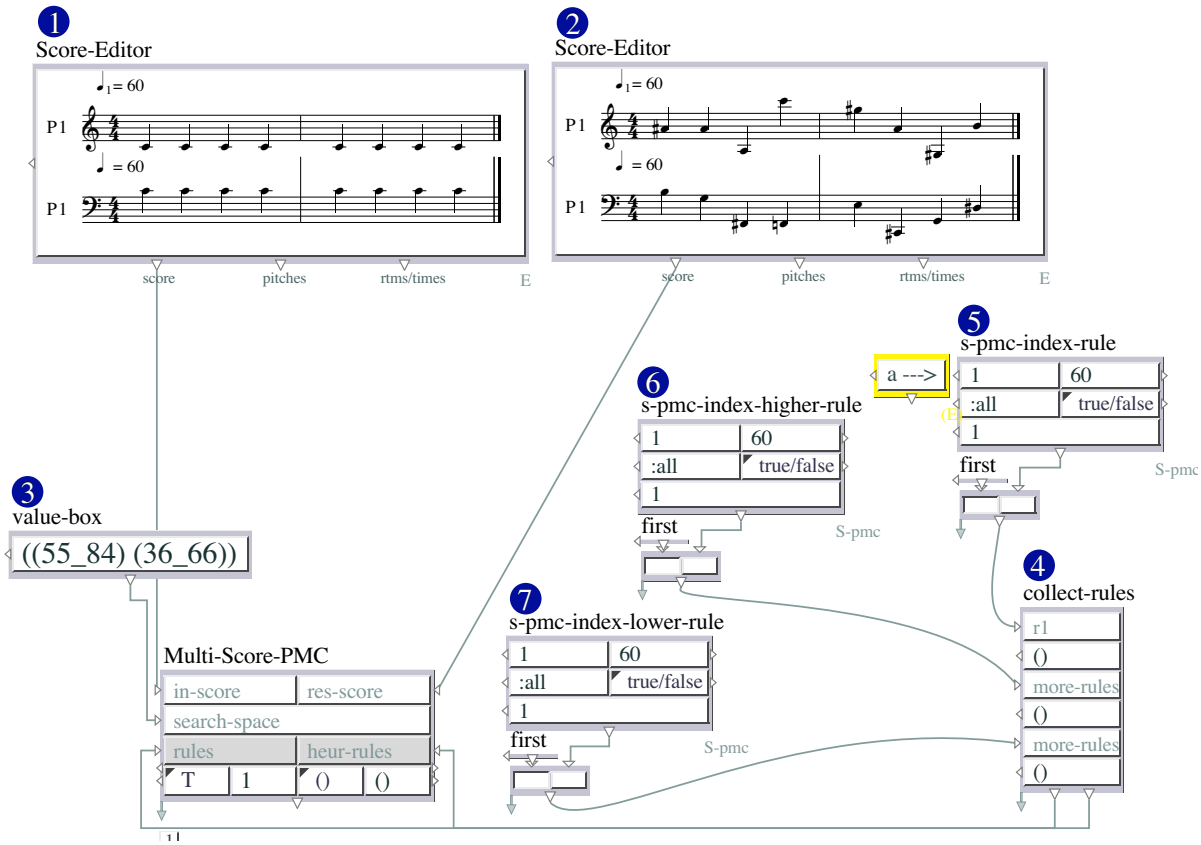

Figure 88: 2-01-1-1-several-index-rules

3.2.1.2 2-Not-Higher-or-Lower-Rules

2.01.1.2 - NOT-HIGHER-OR-LOWER-RULES

This patch shows you two complementary rules.
The SCORE-EDITOR [1] gives only the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result. In the VALUE-BOX [3] you define the search-space for the two voices. The first sub-list is for the highest voice, the second for the lowest.
S-PMC-NOT-HIGHER-RULE [5] forbids any value to be higher than a given number set in [a].
S-PMC-NOT-LOWER-RULE [6] forbids any value to be lower than a given number set in [a].

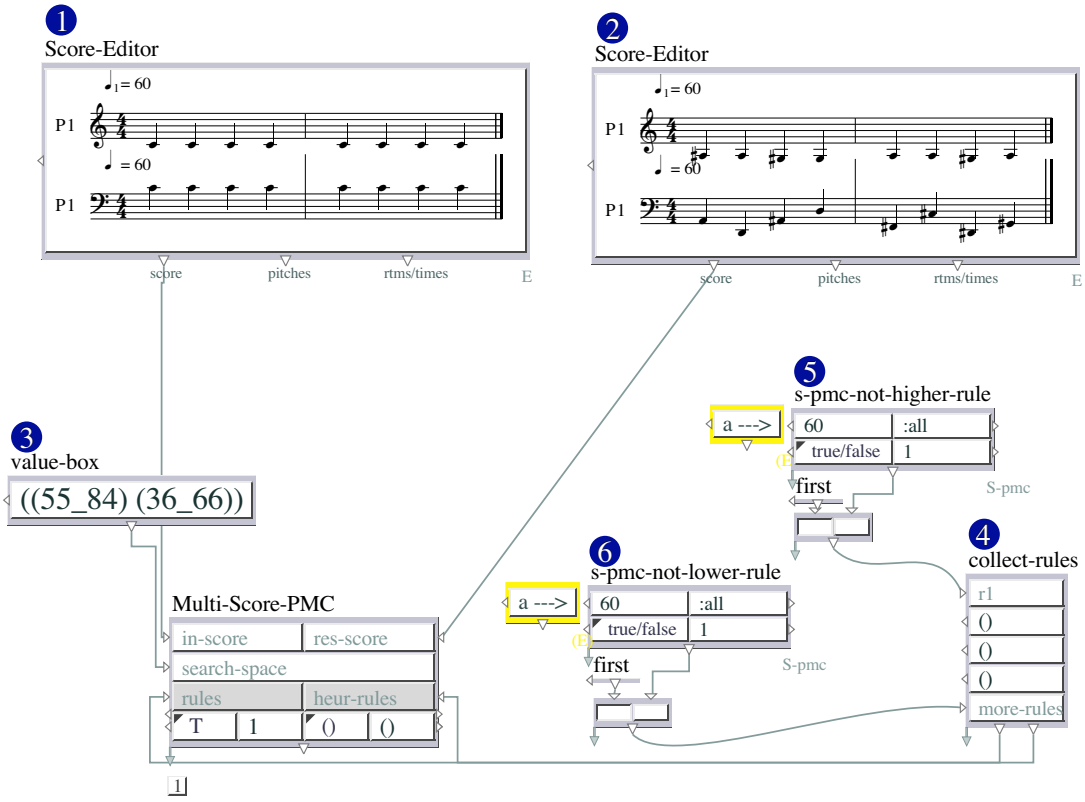

Figure 89: 2-01-1-2-not-higher-or-lower-rules

3.2.1.3 3-No-Lcl-Repetition-Rule

2.01.1.3 - S-PMC-NO-LCL-REPETITION-RULE
This patch applies the rule [4] that do not accept consecutive repetitions of notes in the given voices.
The SCORE-EDITOR [1] only gives the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result. In the VALUE-BOX [3] you define the search-space for the two voices. The first sub-list is for the highest voice, the second for the lowest.

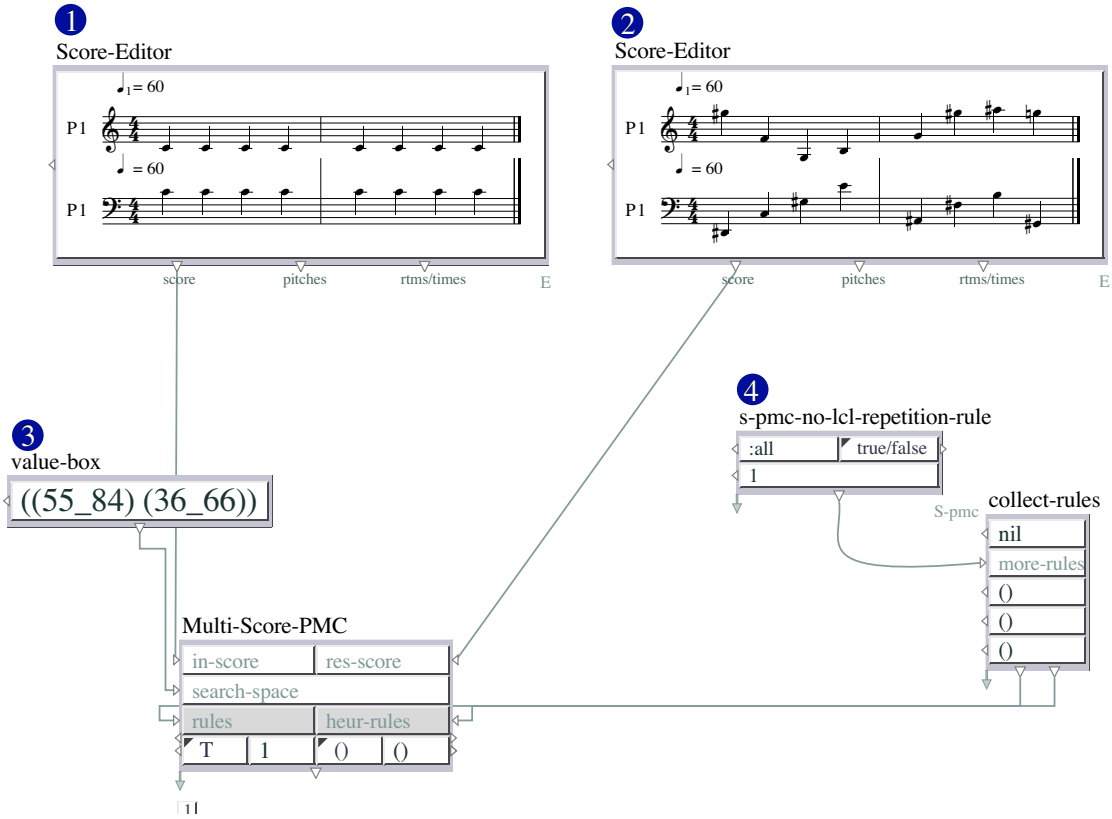

Figure 90: 2-01-1-3-no-lcl-repetition-rule

3.2.1.4 4-N-Ascending-N-Descending-Rules

2.01.1.4 - N-ASCENDING-N-DESCENDING-RULES
The SCORE-EDITOR [1] only gives the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result. In the VALUE-BOX [3] you define the search-space for the two voices. The first sub-list is for the highest voice, the second for the lowest.
S-PMC-N-ASCENDING-RULE [4] obliges a given voice (or all voices, as always) not to have more than 4 ascending notes (set in [a]).
S-PMC-N-DESCENDING-RULE [5] obliges a given voice (or all voices, as always) not to have more than 4 descending notes (set in [a]).

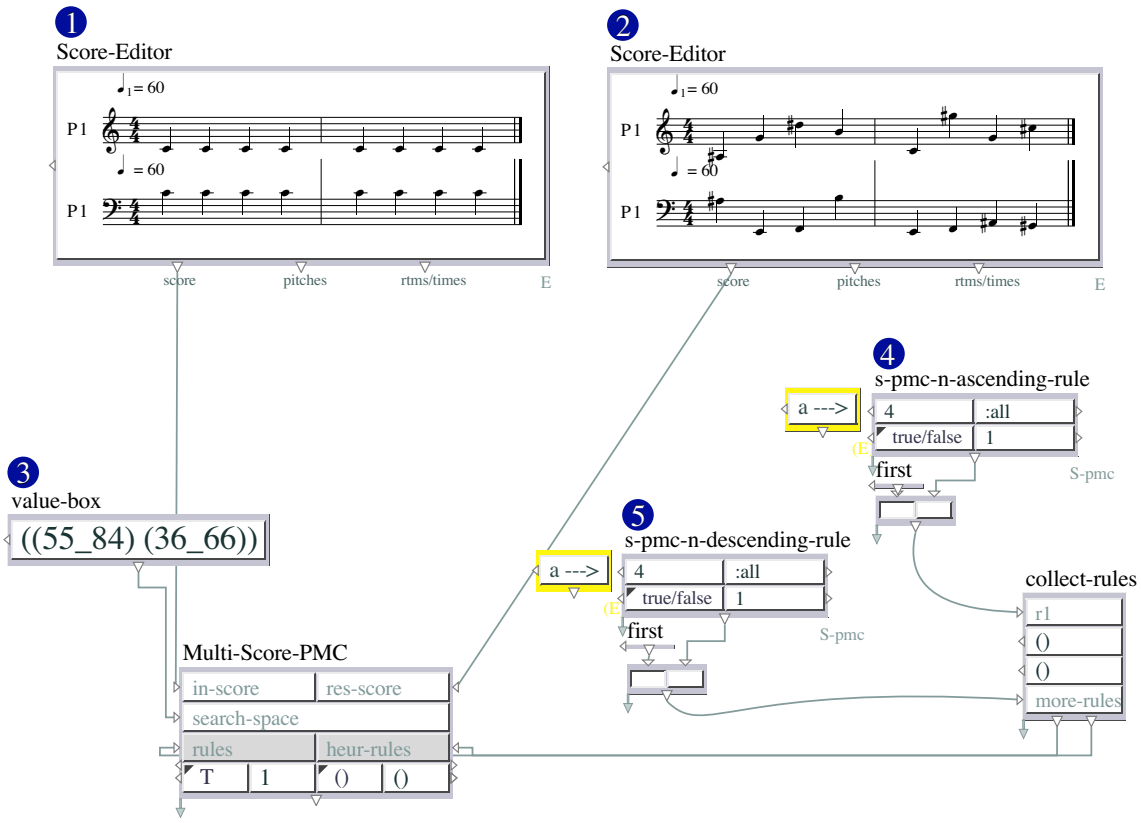

Figure 91: 2-01-1-4-n-ascending-n-descending-rules

3.2.2 2-Intervals-Poly-Rules

3.2.2.1 1-Allowed-Not-Allowed-Interval-Rules

2.01.2.1 - ALLOWED-NOT-ALLOWED-INTERVAL-RULES
The SCORE-EDITOR [1] only gives the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result. In the VALUE-BOX [3] you define the search-space for the two voices. The first sub-list is for the highest voice, the second for the lowest.
S-PMC-ALLOWED-INTERVAL-RULE [4] outputs a solution having only the intervals entered in the 'intervals' input [a]. With the menu 'absolute?' you can choose if the intervals are in absolute value or not.
S-PMC-NOT-ALLOWED-INTERVAL-RULE [5] outputs a solution without any of the interval defined in 'intervals' [a].
With the menu 'absolute?', you can choose if the intervals are in absolute value or not.
Please use the SWITCHs to activate the rules, then evaluate the Multi-Score-PMC to obtain a result.

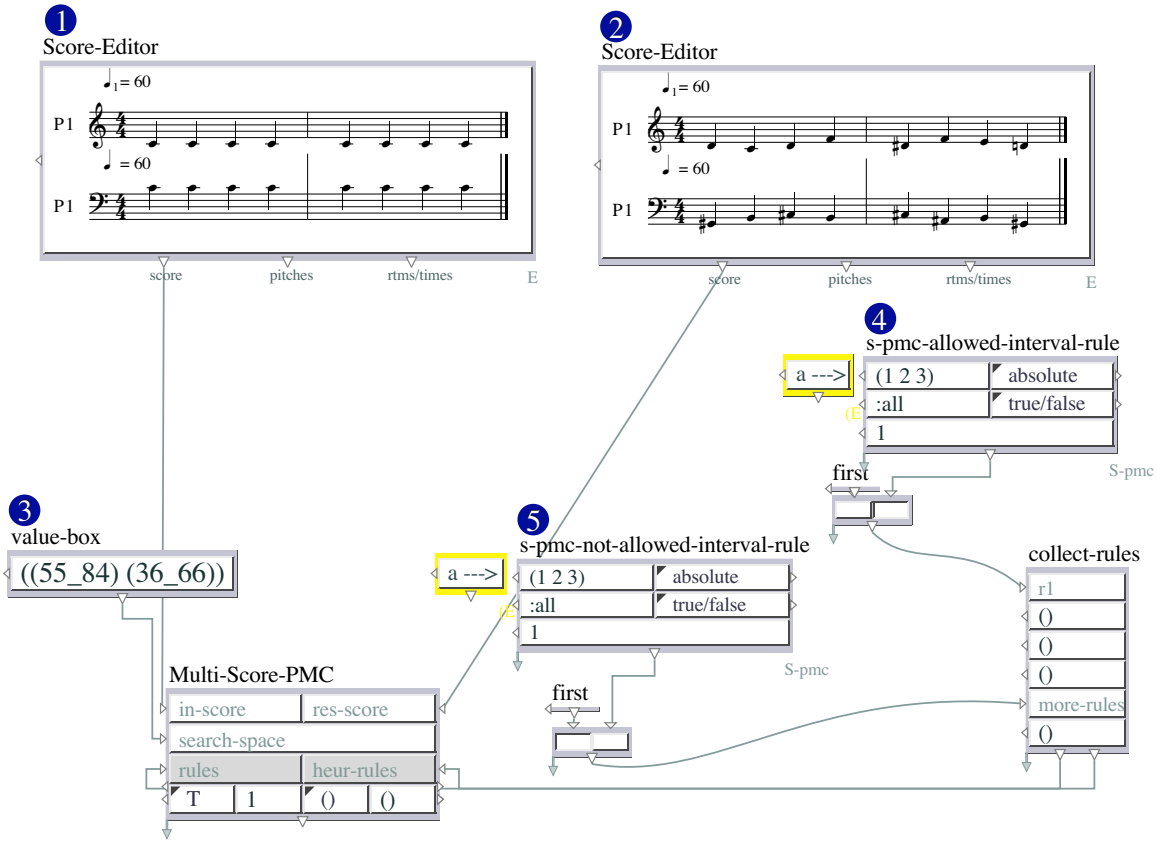

Figure 92: 2-01-2-1-allowed-not-allowed-interval-rules

3.2.2.2 2-Interval-Bigger-Smaller-Rules

2.01.2.2 - INTERVAL-BIGGER-SMALLER-RULES
The SCORE-EDITOR [1] only gives the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result. In the VALUE-BOX [3] you define the search-space for the two voices. The first sub-list is for the highest voice, the second for the lowest.
S-PMC-INTERVAL-BIGGER-RULE [4] outputs a solution including only intervals bigger than the value defined in the 'interval' input [a].
S-PMC-INTERVAL-SMALLER-RULE [5] outputs a solution including only intervals smaller than the value defined in the 'interval' input [a].
Please use the SWITCHs to activate the rules, then evaluate the Multi-Score-PMC to obtain a result.

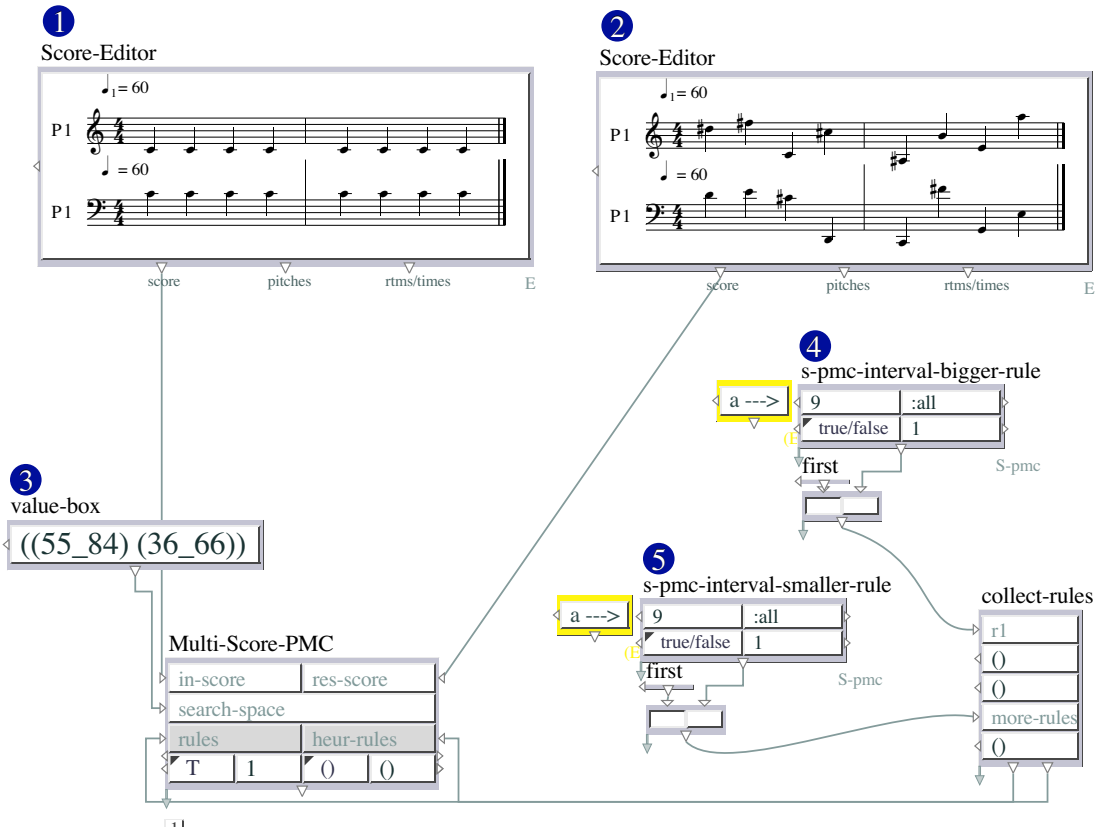

Figure 93: 2-01-2-2-interval-bigger-smaller-rules

3.2.2.3 3-No-Reached-Interval-Rule

2.01.2.3 - S-PMC-NO-REACHED-INTERVAL-RULE
The SCORE-EDITOR [1] gives only the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result. The VALUE-BOX [3] define the search-space for the two voices. The first sub-list is for the highest voice, the second for the lowest.
S-PMC-NO-REACHED-INTRV-RULE [4] obliges a solution NOT to reach a given interval [b] within a given number of notes [a].

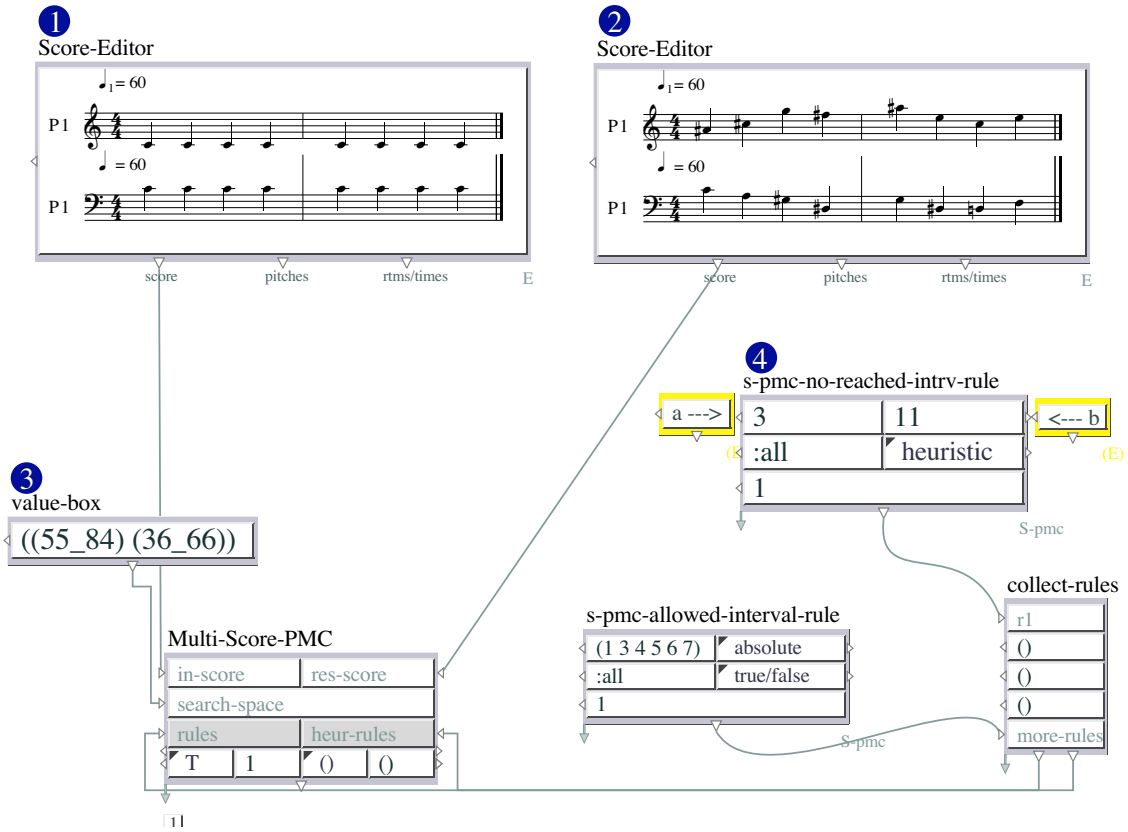

Figure 94: 2-01-2-3-no-reached-interval-rule

3.2.3 3-Pitch-Poly-Rules

3.2.3.1 1-Allowed-Not-Allowed-Pitch-Rule

2.01.3.1 - ALLOWED-NOT-ALLOWED-PICTH-RULE
The SCORE-EDITOR [1] only gives the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result. The VALUE-BOX [3] define the search-space for the two voices. The first sub-list is for the highest voice, the second for the lowest.
S-PMC-ALLOWED-PITCH-RULE [4] forces the solution to be constitued only by pitches (in modulo 12) entered in the 'pitch' input [a].
S-PMC-NOT-ALLOWED-PITCH-RULE [5] forbids the solution to include any pitch (in modulo 12) entered in the 'pitch' input [a].
Please use the SWITCHs to choose which rule you want to activate, then evaluate the Multi-Score-PMC to generate a result.

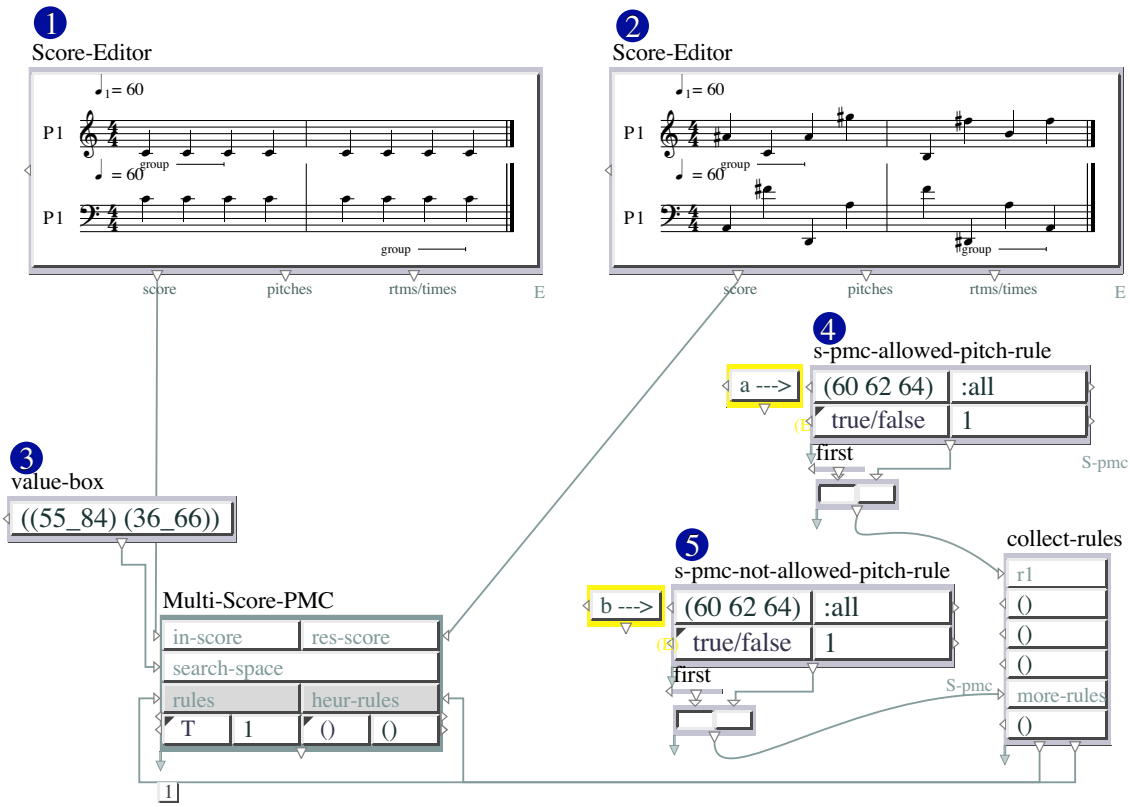

Figure 95: 2-01-3-1-allowed-not-allowed-pitch-rule

3.2.3.2 2-Allowed-Not-Allowed-Pitch-Class-Rule

2.01.3.2 - ALLOWED-NOT-ALLOWED-PITCH-CLASS-SUB-GROUP-RULE
The SCORE-EDITOR [1] only gives the rhythmical structure of the result. The SCORE-EDITOR [2] prints out the result. The VALUE-BOX [3] define the search-space for the two voices. The first sub-list is for the highest voice, the second for the lowest.
S-PMC-ALLOWED-PITCH-RULE [4] forces the solution to be constitued only by pitches (in modulo 12) entered in the 'pitch' input [a].
S-PMC-NOT-ALLOWED-PITCH-CLASS-SUB-GROUP-RULE [5] outputs a solution in which the PC-set (for instance minor triad) indicated in 'pitch' will NOT be allowed in any octave. That means that I'm looking for a solution without any minor triad.
S-PMC-ALLOWED-PITCH-CLASS-SUB-GROUP-RULE [6] outputs a solution in which the PC-set (for instance minor triad) indicated in 'pitch' will be allowed in any octave but also including other notes. That means that I'm looking for a solution including a minor triad.
Please use the SWITCHs to choose which rule you want to activate, then evaluate the Multi-Score-PMC to obtain a result.

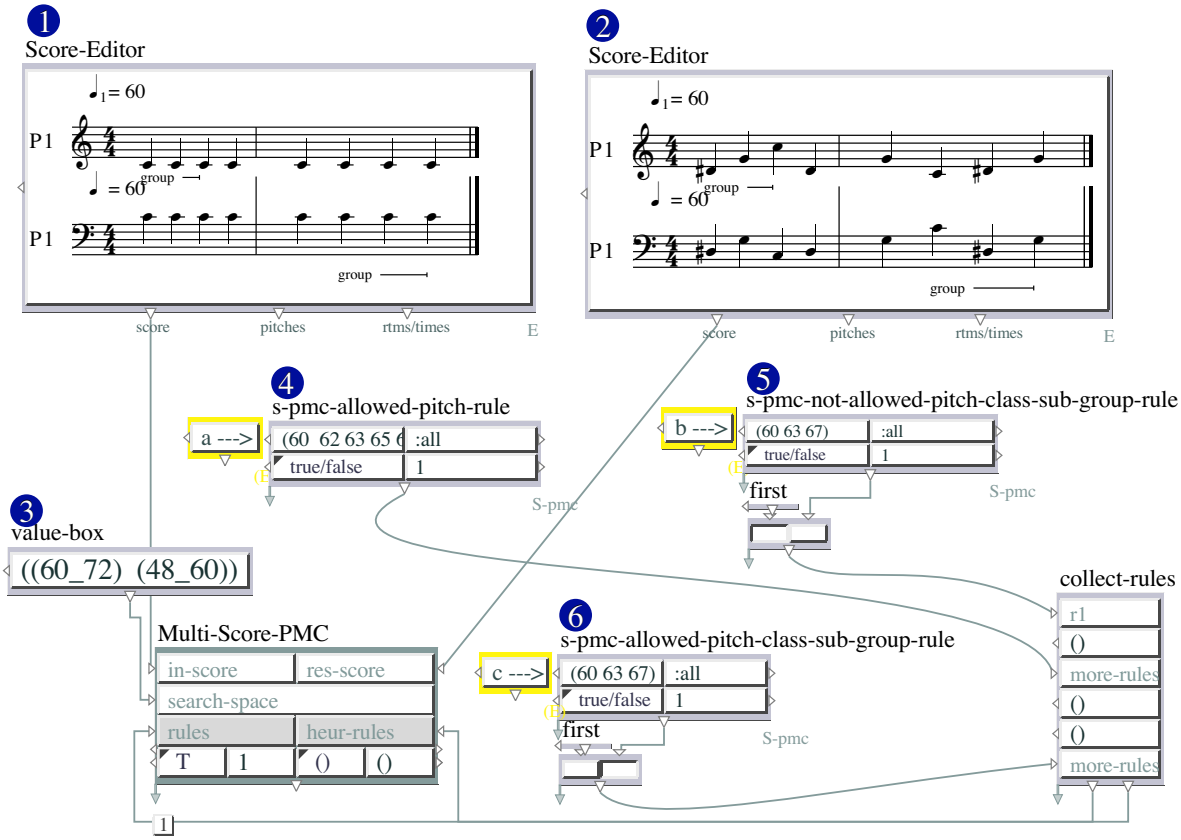

Figure 96: 2-01-3-2-allowed-not-allowed-pitch-class-rule

3.2.4 4-Resolution-Poly-Rules

3.2.4.1 1-Tone-Not-Tone-Resolution-Rule

2.01.4.1 - S-PMC-TONE-RESOLUTION-RULE
S-PMC-TONE-RESOLUTION-RULE [4] is like the 'sensibile' rule. In this case if a B note [a] appears, it has to be followed by a C note [b].
S-PMC-NOT-TONE-RESOLUTION-RULE [5] does the contrary.
ATTENTION With these two rules (S-PMC-TONE-RESOLUTION-RULE and S-PMC-NOT-TONE-RESOLUTION-RULE) you can put a list in [b] so that a single pitch can or cannot be followed by a list of possible pitches.

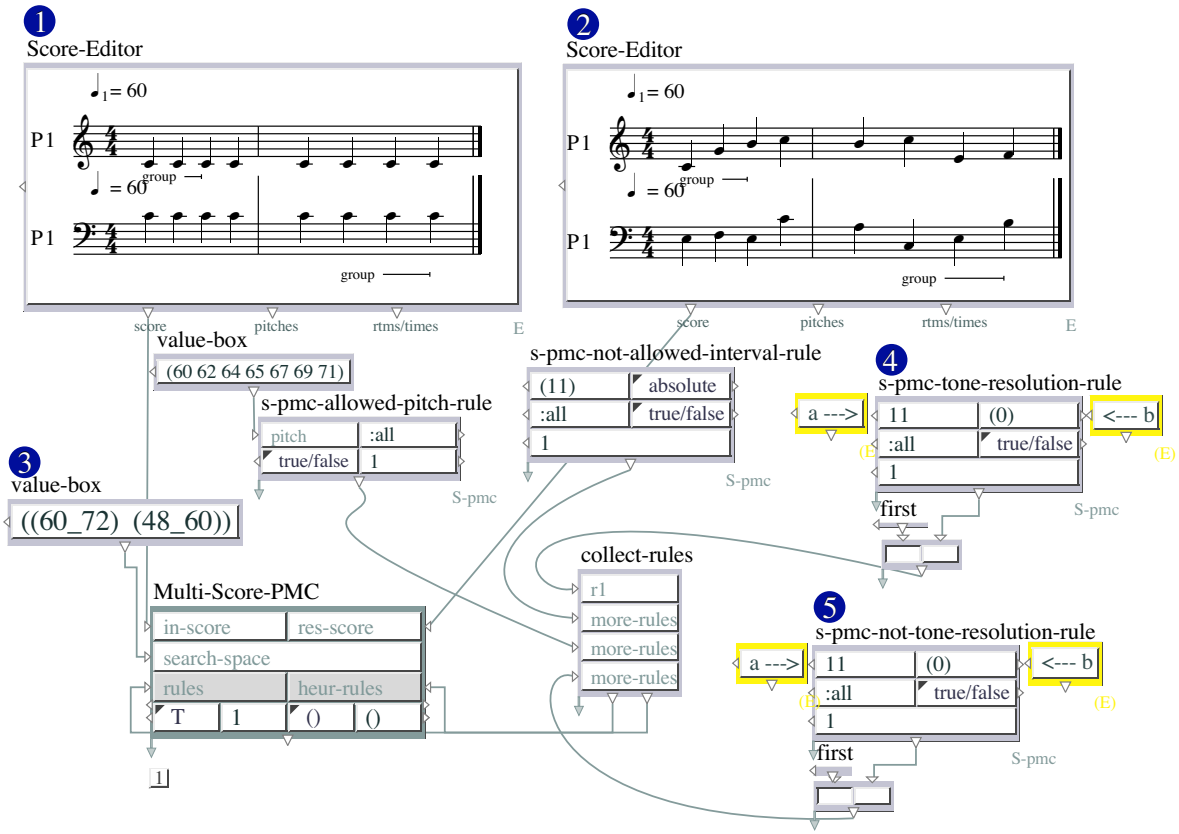

Figure 97: 2-01-4-1-tone-not-tone-resolution-rule

3.2.4.2 2-Jump-Resolution-Rule

2.01.4.2 - S-PMC-JUMP-RESOLUTION-RULE
S-PMC-JUMP-RESOLUTION-RULE [4] makes sure that if there's a jump bigger than an augmented fourth [a], the next interval has to be smaller than a major second [b] and in the opposite direction.

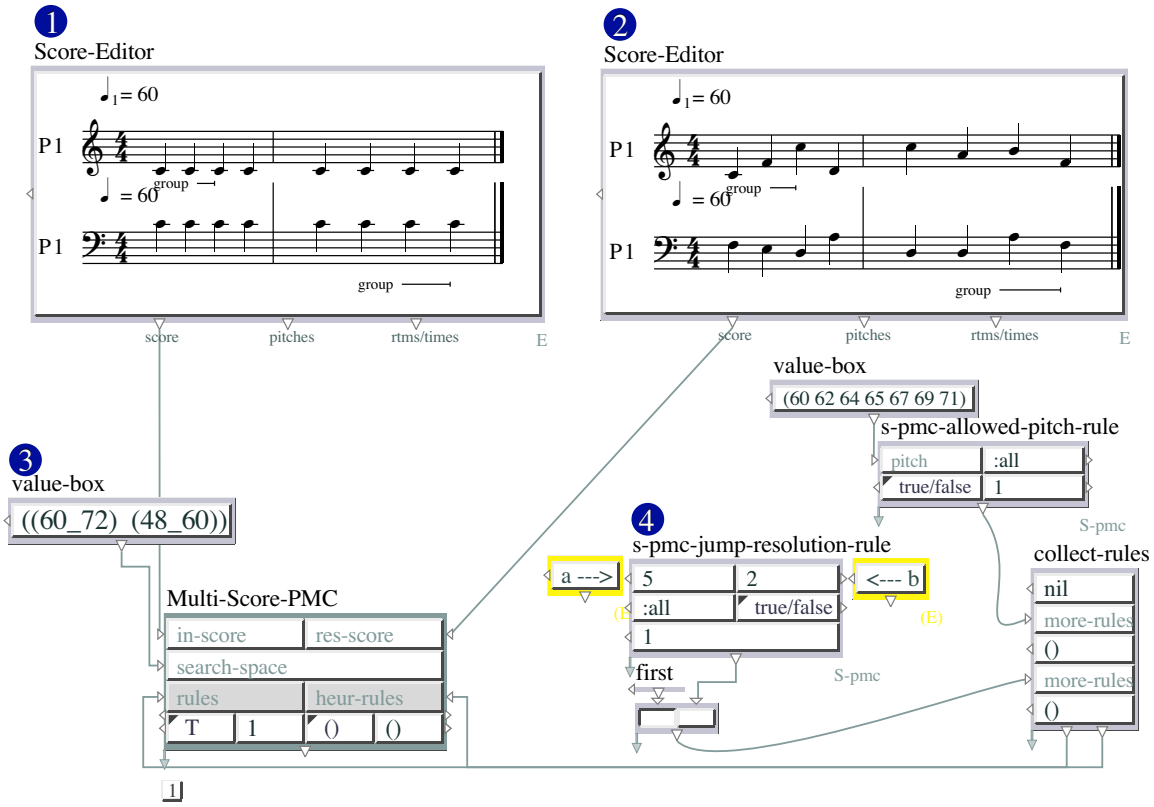

Figure 98: 2-01-4-2-jump-resolution-rule

3.2.5 5-Shaping-Poly-Rules

3.2.5.1 1-Given-Voice-Rule

2.01.5.1 - S-PMC-GIVEN-VOICE-RULE
The S-PMC-GIVEN-VOICE-RULE [4] obliges the pitches of the second voice set in [b] (the lowest in this example), to reproduce the sequence from the SCORE-EDITOR [a]. Pay attention! A voice can be polyphonic: for this reason we have chosen to call this function S-PMC-GIVEN-VOICE-RULE. But if you use monodic voice, this function works as the MK-FIX-PROFILE-RULE for the Multi-PMC. (Look at the tutorial 1.04.03-mk-fix-profile-rule.pwgl)

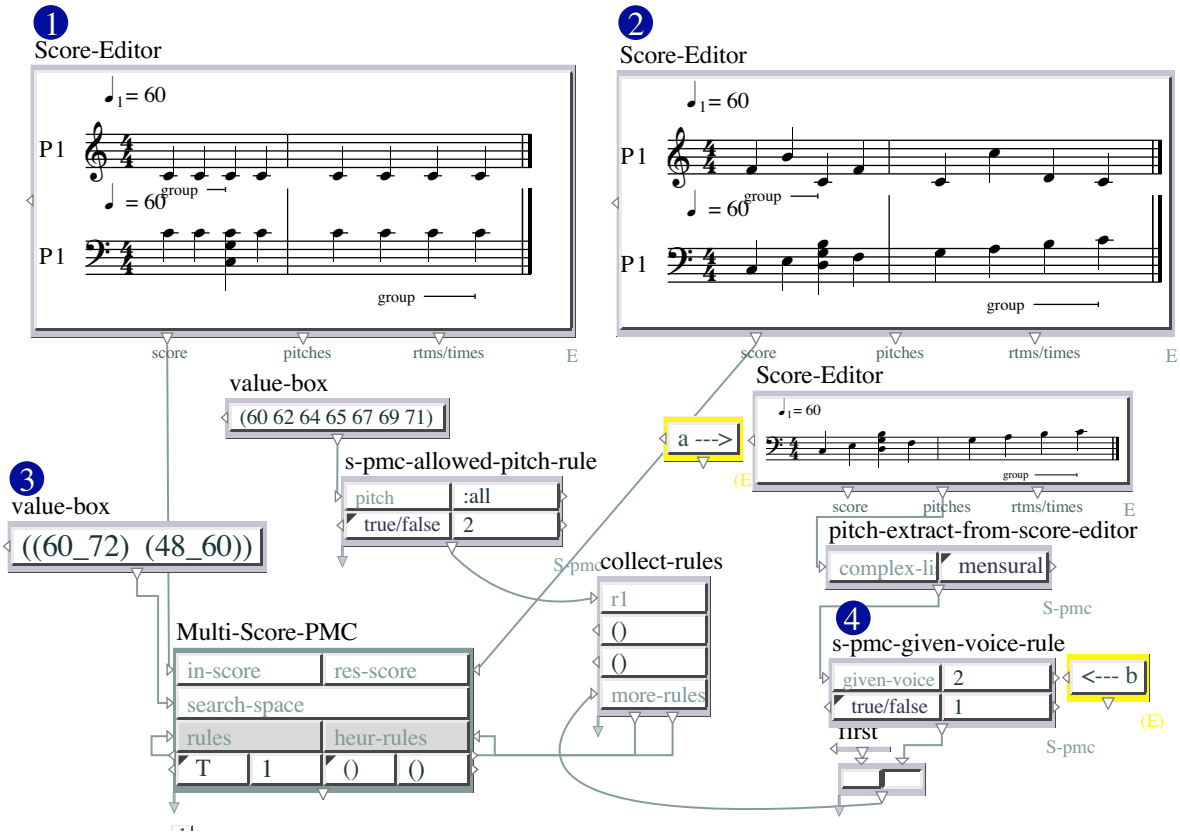

Figure 99: 2-01-5-1-given-voice-rule

3.2.5.2 2-Mk-Profile-Rule

2.01.5.2 - S-PMC-MK-FIX-PROFILE-RULE
S-PMC-MK-PROFILE-RULE [4] obliges a given voice (in this case the lowest set in [b]) to follow strictly (true/false) or as much as possible (heuristic) the shape of the bpf [a]. In this case, the rule is used in true/false mode, so the lowest voice is totally fixed, but the highest voice remains untouched.
Please make sure that the 'curve-min' [c] and 'curve-max' [d] inputs are set in the good interval according to the voice you want to constrain.

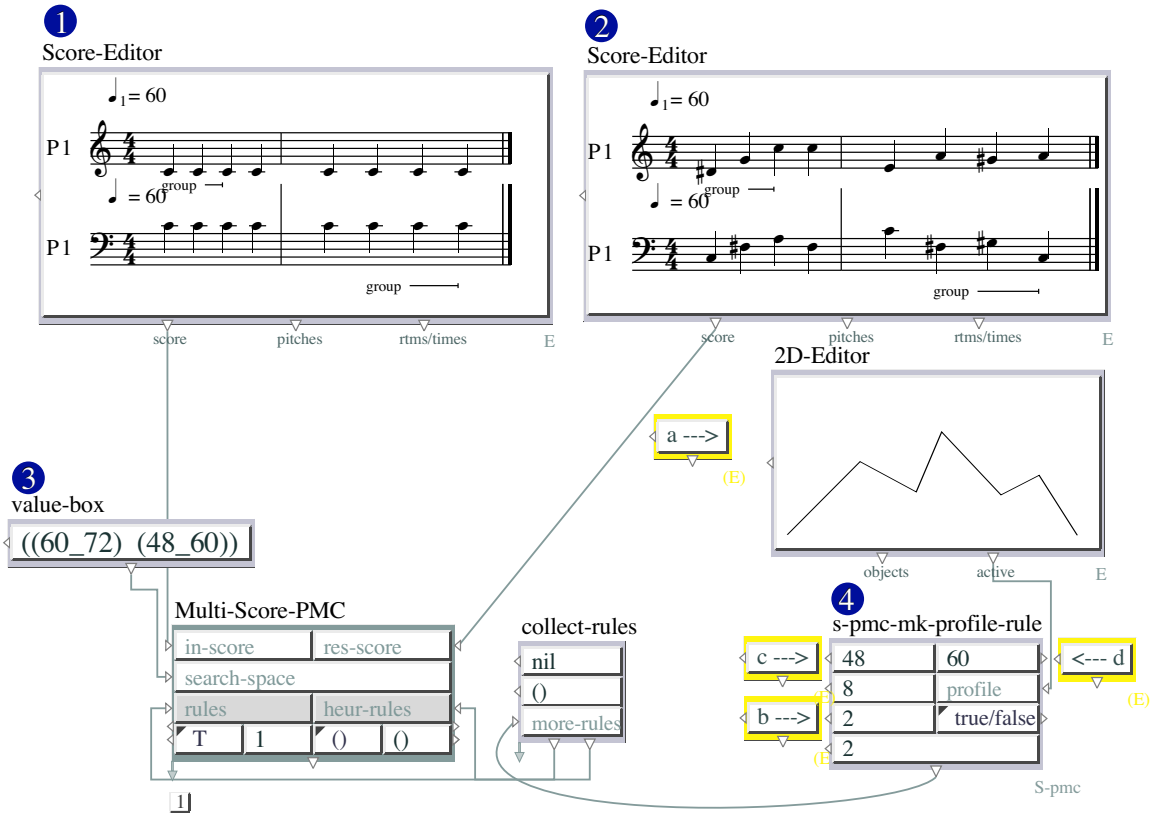

Figure 100: 2-01-5-2-mk-profile-rule

3.3 02-Harmonic-Rules

3.3.1 01-Index-Allowed-Harmony

2.02.01 - S-PMC-INDEX-ALLOWED-&-NOTALLOWED-HARMONY-RULE
In the SCORE-EDITOR [1] you enter the rhythmical structure for the solution. In this patch, the result will be printed out in the same SCORE-EDITOR. (See also in the Examples 'WHEN-TO-USE-TWO-SCORE-EDITORS').
In the VALUE-BOX [2] you define the range for the four voice search-space. Remember that the first voice is (in this case) the soprano and the last, the bass voice.
S-PMC-INDEX-ALLOWED-HARMONY-RULE [3] obliges the solution to follow the harmony defined in [a], but only on the index defined in [b].
In this example, the solution must produce C-Major chords (modulo 12) at measures 1 and 3.
S-PMC-INDEX-NOT-ALLOWED-HARMONY-RULE [4] does the opposite.

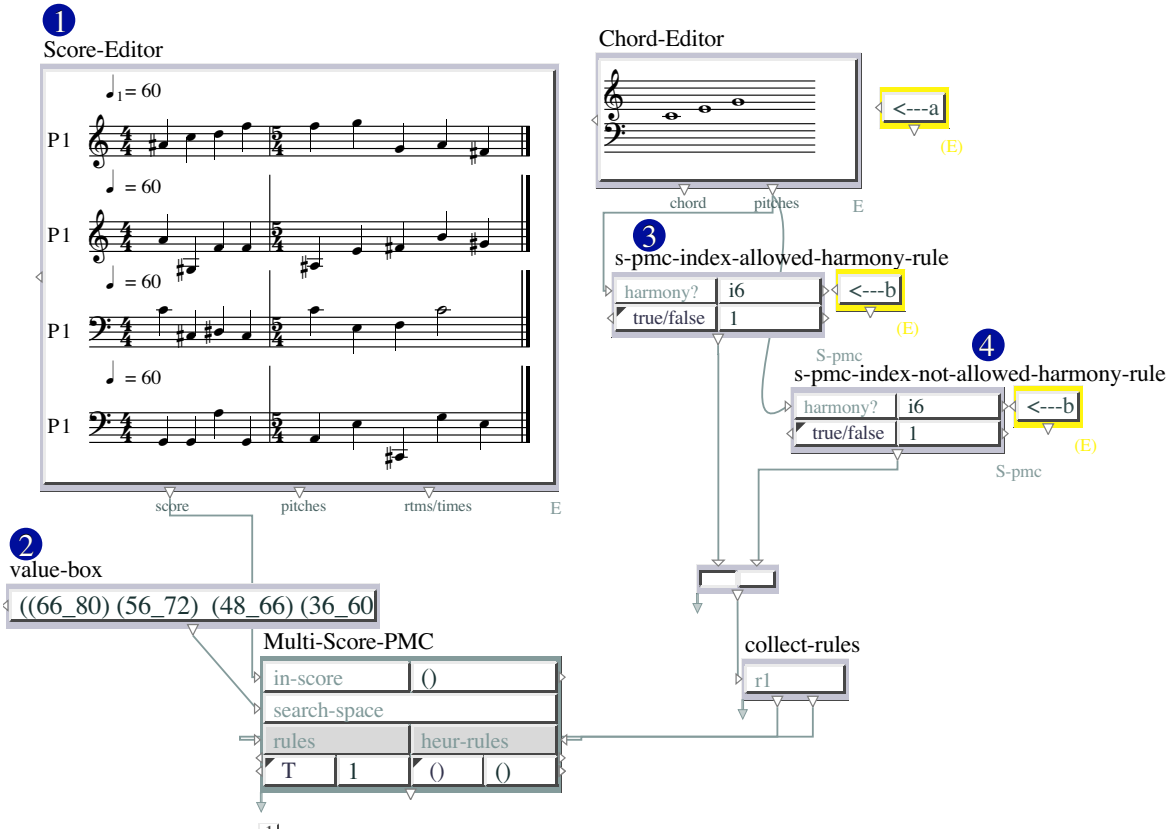

Figure 101: 2-02-01-index-allowed-harmony

3.3.2 02-Allowed-&-Not-Harmony-in-Given-Measures

2.02.02 - S-PMC-ALLOWED-&-NOT-HARMONY-IN-GIVEN-MEASURES-RULE
In the SCORE-EDITOR [1] you enter the rhythmical structure for the solution. In this patch, the result will be printed out in the same SCORE-EDITOR. (See also in the Examples 'WHEN-TO-USE-TWO-SCORE-EDITORS').
In the VALUE-BOX [2] you define the range for the four voice search-space. Remember that the first voice is (in this case) the soprano and the last, the bass voice.
S-PMC-ALLOWED-HARMONY-IN-GIVEN-MEASURES-RULE [3] obliges the solution to follow the harmony defined in [a], but only at the measures defined in [b].
In this example, the solution must produce C-Major chords (modulo 12) at measures 1 and 3.
S-PMC-NOT-ALLOWED-HARMONY-IN-GIVEN-MEASURES-RULE [4] does the opposite.

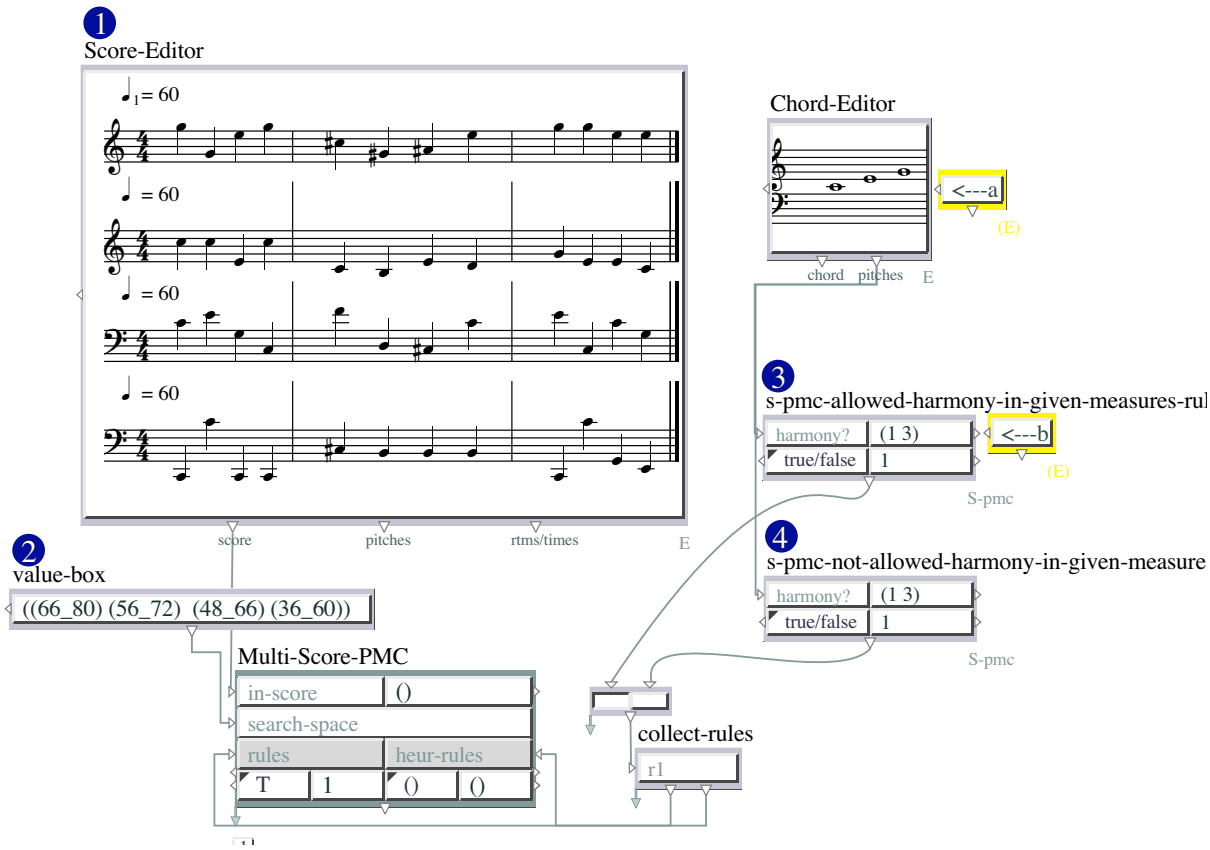

Figure 102: 2-02-02-allowed-not-harmony-in-given-measures

3.3.3 03-Allowed-&-Not-Harmony-on-Beat

2.02.03 - S-PMC-ALLOWED-&-NOT-HARMONY-ON-BEAT-RULE
S-PMC-ALLOWED-HARMONY-ON-BEAT-RULE [3] obliges the solution to follow the harmony set in [a], but only on the beats defined in [b].
In this example, a C-Major chord (modulo 12) is produced on the third beat of each measures, but also in the fifth one. Because only the second measure has 5 beats, the fifth beat will be concerned only in the second measure.
The S-PMC-NOT-ALLOWED-HARMONY-ON-BEAT-RUL [4] does the contrary.

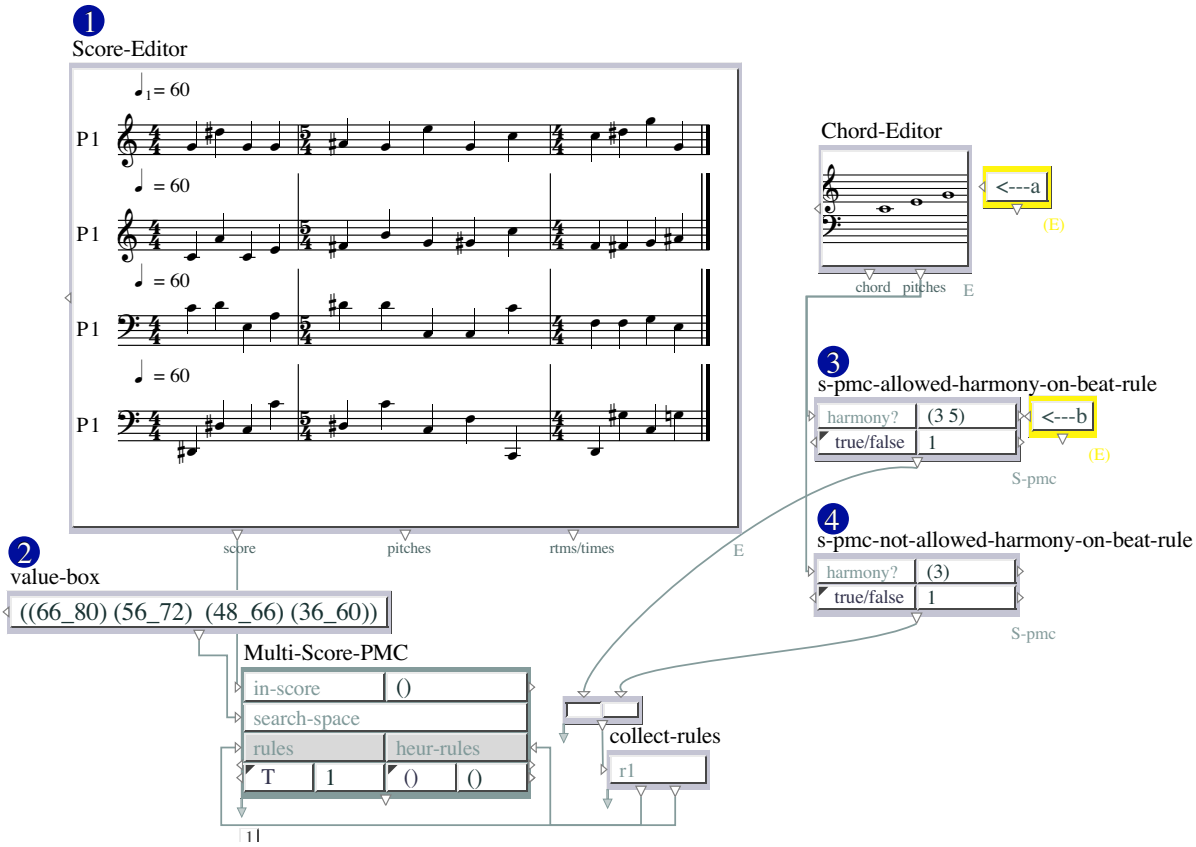

Figure 103: 2-02-03-allowed-not-harmony-on-beat

3.3.4 04-Allowed-&-Not-Harmonic-Interval

2.02.04 - S-PMC-ALLOWED-&-NOT-ALLOWED-HARMONIC-INT-RULE
S-PMC-ALLOWED-HARMONIC-INT-RULE [3] allows only a given set of harmonic intervals, defined in ”intervals’ input, between every parts.
S-PMC-NOT-ALLOWED-HARMONIC-INT-RULE [3] forbids a given set of harmonics intervals, defined in ”intervals’ input, between every parts.

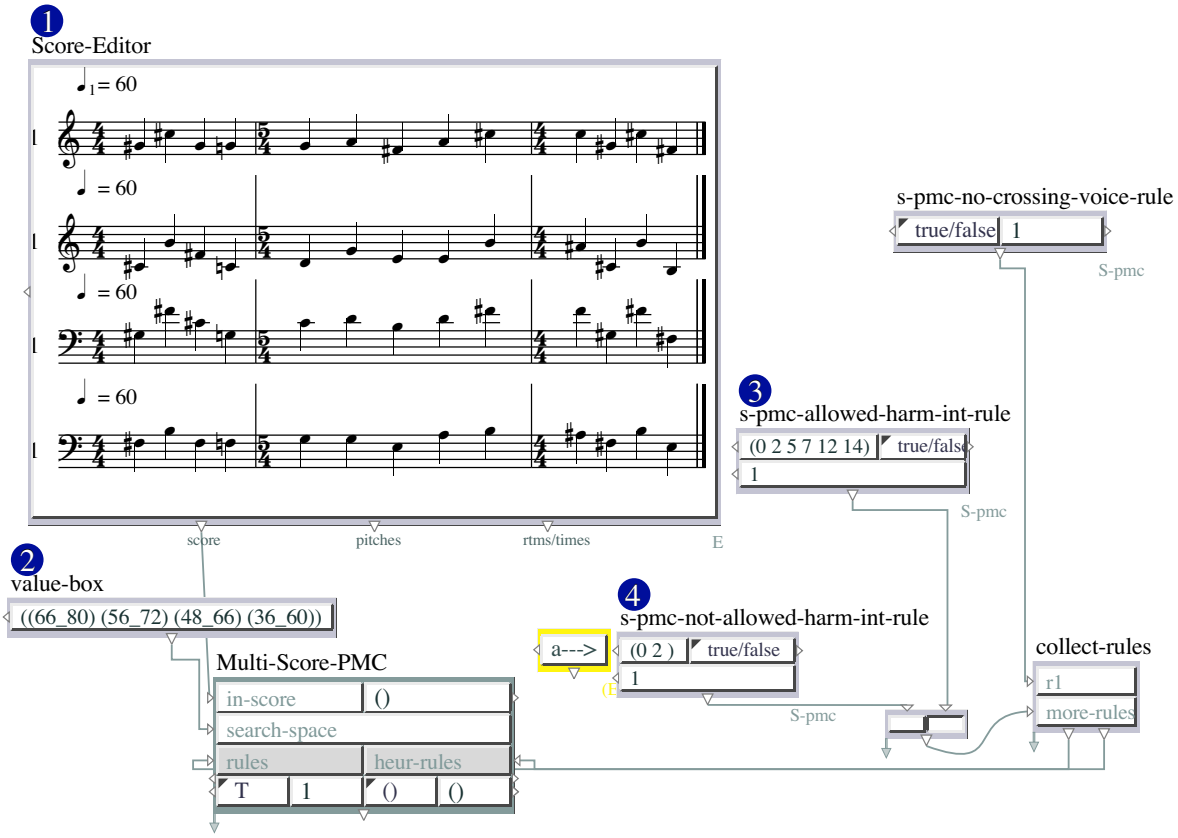

Figure 104: 2-02-04-allowed-not-harmonic-interval

3.3.5 05-All-Notes-Included

2.02.05 - S-PMC-ALL-NOTES-INCLUDED-RULE
S-PMC-ALL-NOTES-INCLUDED-RULE [3] obliges each harmony to have exactly the number of different pitches set in [a].
In this example, because of the S-PMC-ALLOWED-PITCH-RULE [4], only four notes are allowed (C, D, E and F). S-PMC-ALL-NOTES-INCLUDED-RULE [3] creates solutions including all the four notes in each chord.

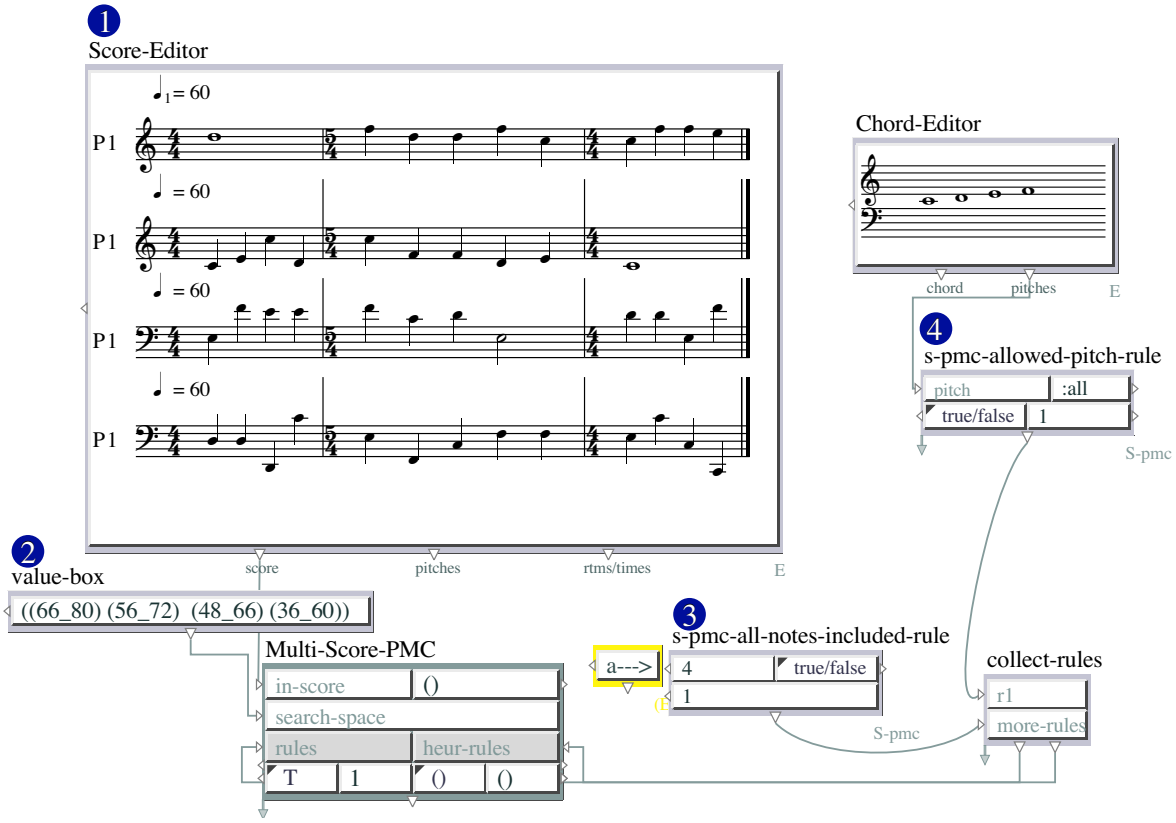

Figure 105: 2-02-05-all-notes-included

3.3.6 06-Index-All-Notes-Included

2.02.06 - INDEX-ALL-NOTES-INCLUDED
In the SCORE-EDITOR [1] you enter the rhythmical structure for the solution. In this patch, the result will be printed out in the same SCORE-EDITOR. (See also in the Examples 'WHEN-TO-USE-TWO-SCORE-EDITORS').
In the VALUE-BOX [2] you define the range for the four voice search-space. Remember that the first voice is (in this case) the soprano and the last, the bass voice.
S-PMC-INDEX-ALLOWED-HARMONY-RULE [3] obliges the solution to follow the harmony defined in [a], but only on the index defined in [b]. In this case pitches are (in modulo 12) 0, 4, 7 and 9.
S-PMC-INDEX-ALL-NOTES-INCLUDED-RULE [4] obliges the solution to have all the pitches defined in [a] on the index defined in [b].
ATTENTION: in fact the S-PMC-INDEX-ALL-NOTES-INCLUDED-RULE [4] asks for a solution having a number of different pitches set in [c].

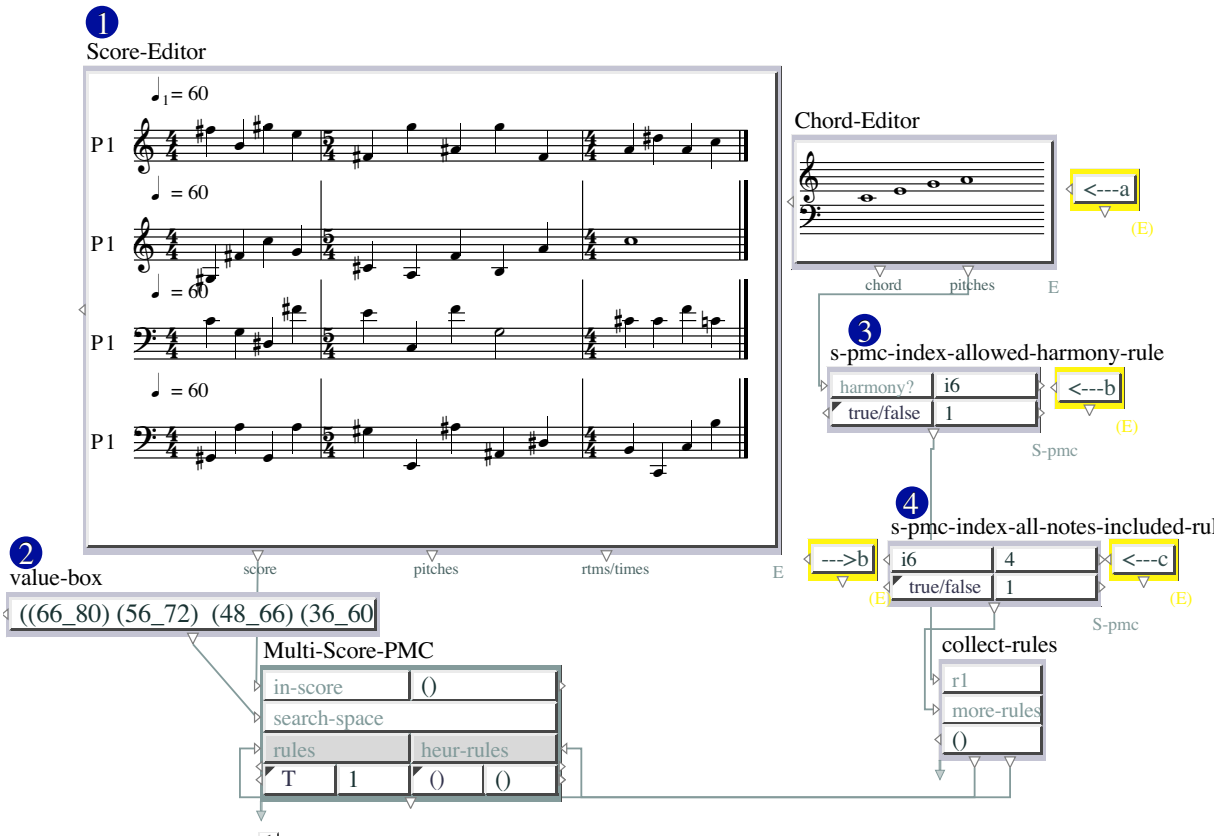

Figure 106: 2-02-06-Index-all-notes-included

3.3.7 07-All-Notes-Included-on-Beat

2.02.07 - S-PMC-ALL-NOTES-INCLUDED-ON-BEAT-RULE
S-PMC-ALL-NOTES-INCLUDED-ON-BEAT-RULE [3] obliges the solution to have, on each beat set in [a], the number of different pitches set in [b].

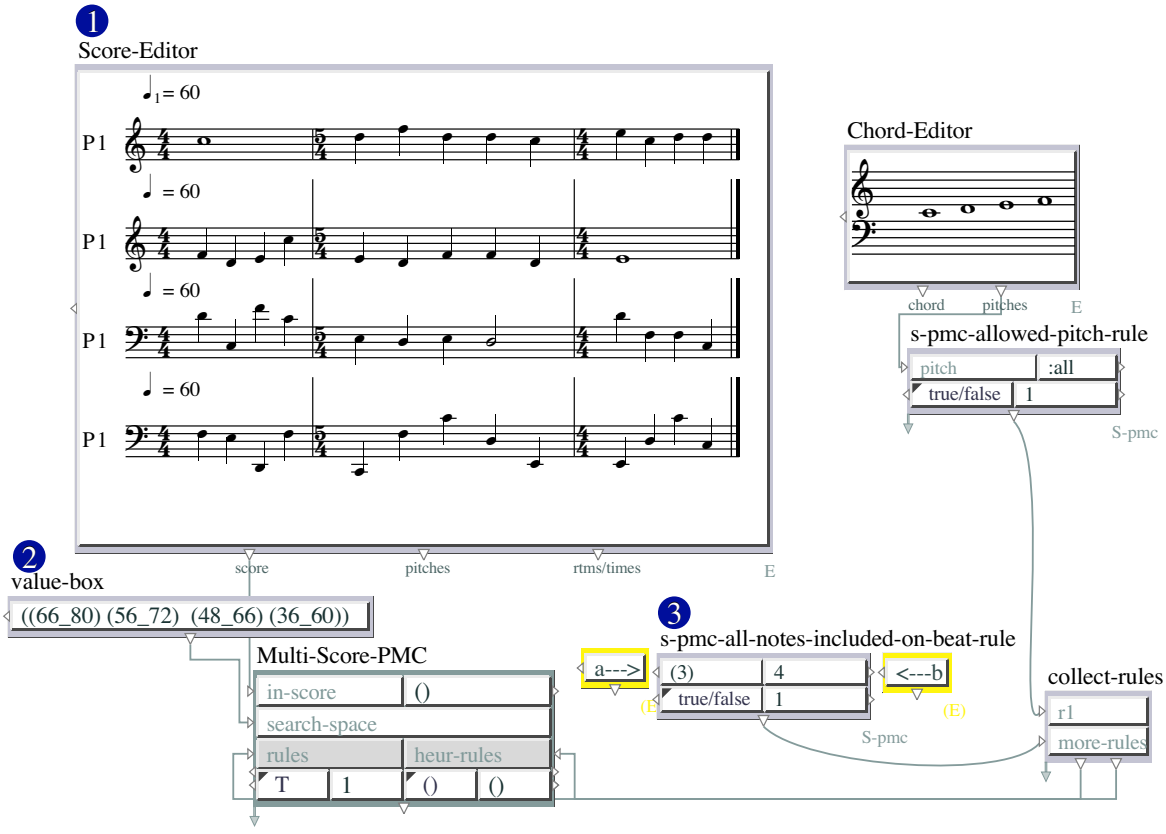

Figure 107: 2-02-07-all-notes-included-on-beat

3.3.8 08-Forbidden-Inversion

2.02.08 - S-PMC-FORBIDDEN-INVERSIONS-RULE
S-PMC-FORBIDDEN-INVERSIONS-RULE [3] forbids a solution to include one or more chord inversions. The forbidden inversions are set in [a].
Please evaluate the Multi-Score-PMC and observe that the G note will never appear in the bass voice when a C-major chord appears.
ATTENTION This rule does not work with heterogeneous rhythmical values. Open the 'See-when-does-not-work' abstraction to see it.

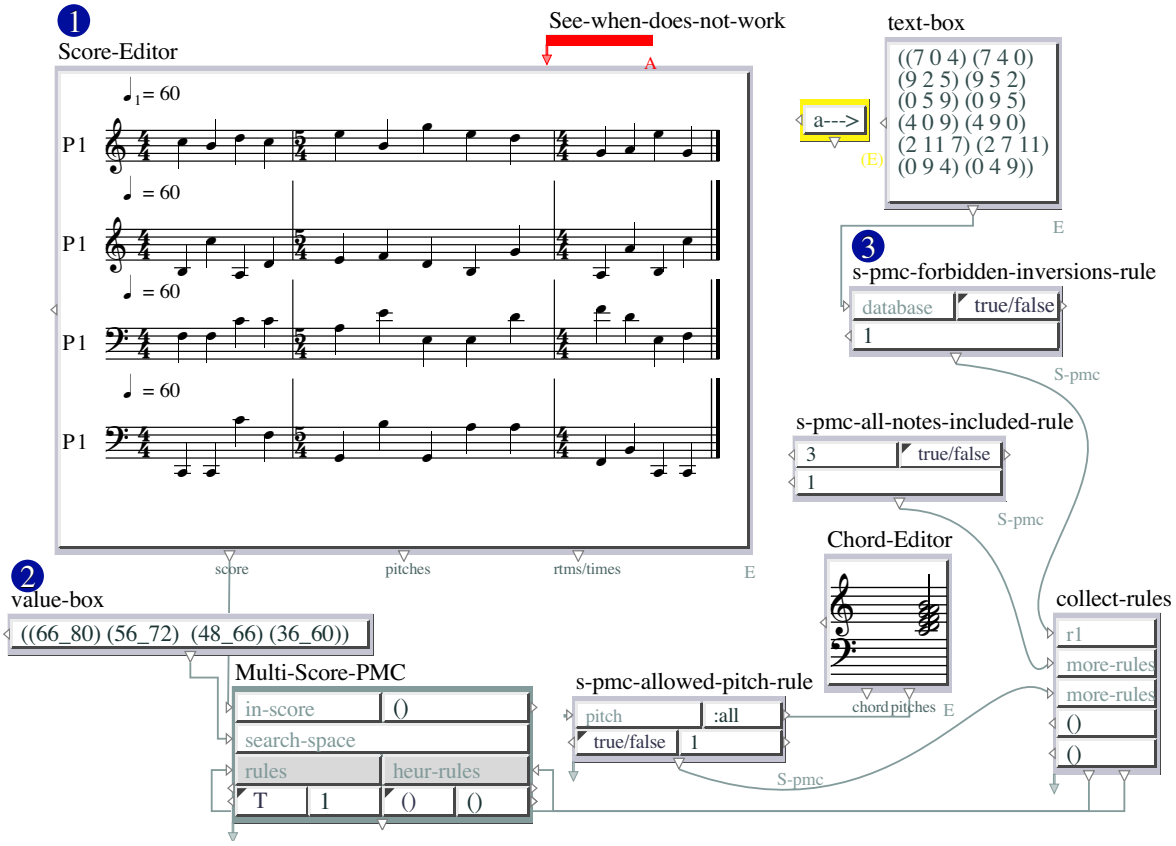

Figure 108: 2-02-08-forbidden-inversion

3.3.9 09-Preferred-Duplicates

2.02.09 - S-PMC-PREFERRED-DUPLICATE-RULE
S-PMC-PREFERRED-DUPLICATE-RULE [3] makes sure that the solution, having four notes each time but only three notes per chord, will choose the pitch (in modulo 12) set in [a] as the preferred duplicate.

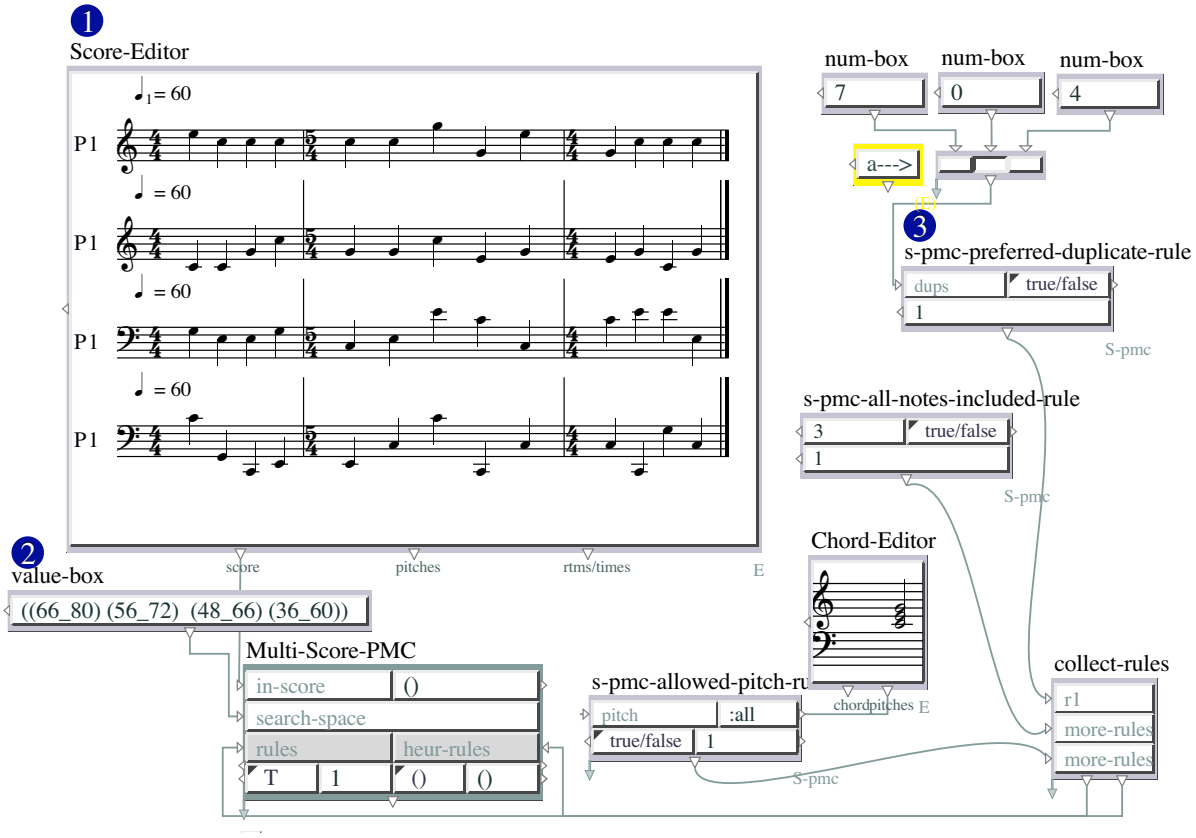

Figure 109: 2-02-09-preferred-duplicates

3.3.10 10-Allowed-Harmony

2.02.10 - S-PMC-ALLOWED-HARM-RULE
S-PMC-ALLOWED-HARM-RULE [3] defines which chords are admitted in the solution. Please open the 'Allowed-chords' abstraction [a]. In [b] you define the chords you want to be allowed in the solution.
Then it is just a set-theory organization. In [c] are defined the possible transpositions of the sets. In [d] I define also the possible sub-groups of a chord that can be allowed.

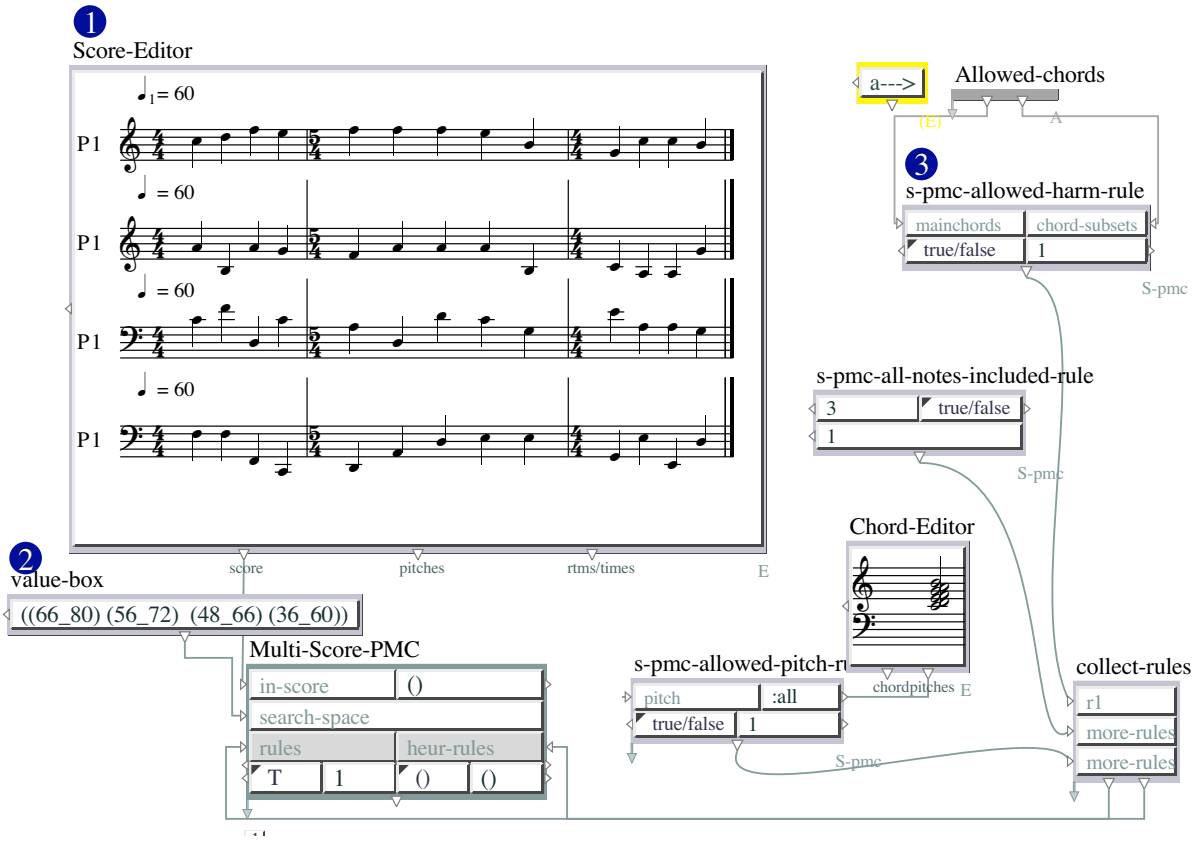

Figure 110: 2-02-10-allowed-harmony

3.3.11 11-Chord-Succession

2.02.11 - S-PMC-CHORDS-SUCCESSION-RULE
S-PMC-CHORDS-SUCCESSION-RULE [6] looks for solutions having the chord-successions set in 'database' input [a].
Please open the 'Chords-successions' abstraction. This patch repeats 6 times the same procedure. In a CHORD-EDITOR [b] you define a chord and in a list of CHORD-EDITORS [c] you define the chords allowed to follow the one set in [b].
ATTENTION The chords are entered as midi notes in CHORD-EDITORS, but S-PMC-CHORDS-SUCCESSION-RULE recognizes only the Set-Theory format.

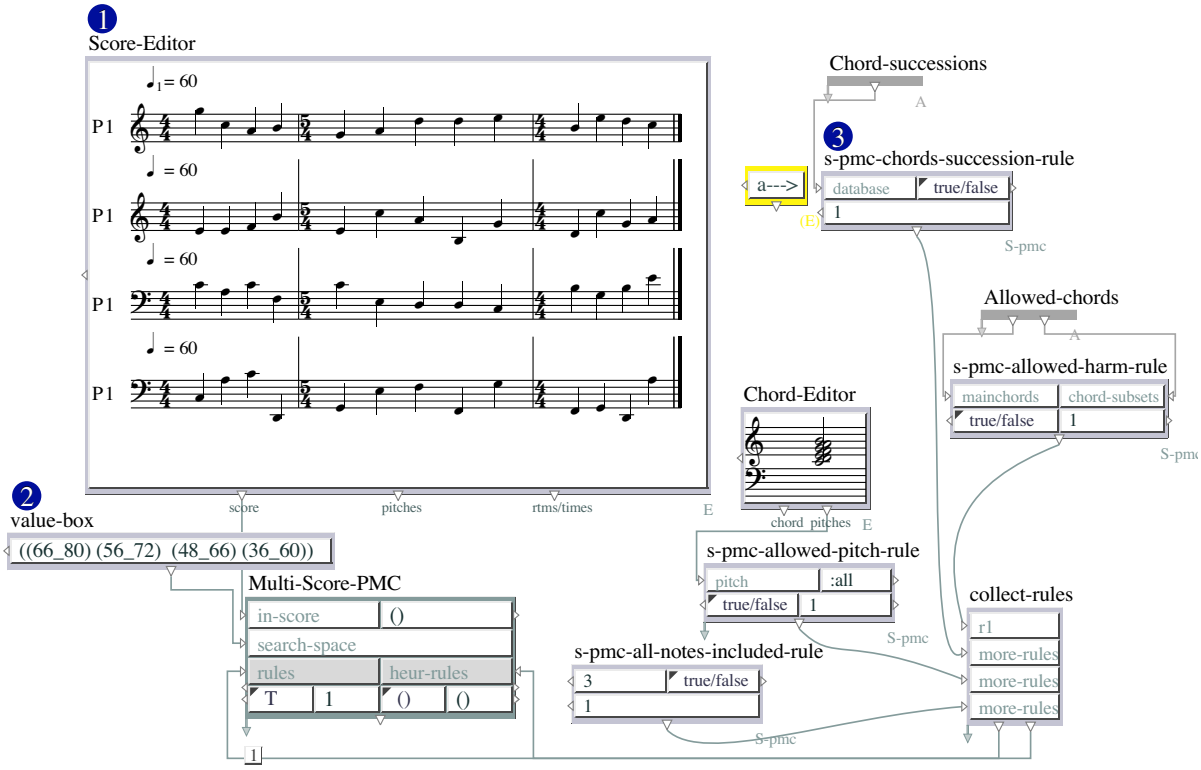

Figure 111: 2-02-11-chord-succession

3.3.12 12-Allowed-Interval-between-2-Parts

2.02.12 - S-PMC-INTV-BETWEEN-2-PARTS-RULE
S-PMC-INTV-BETWEEN-2-PARTS-RULE [3] deals with vertical intervals between two voices.
In the input [a] you define a part and in [b] another one. Then in [c] you set which intervals are allowed between the voices defined in [a] and [b].

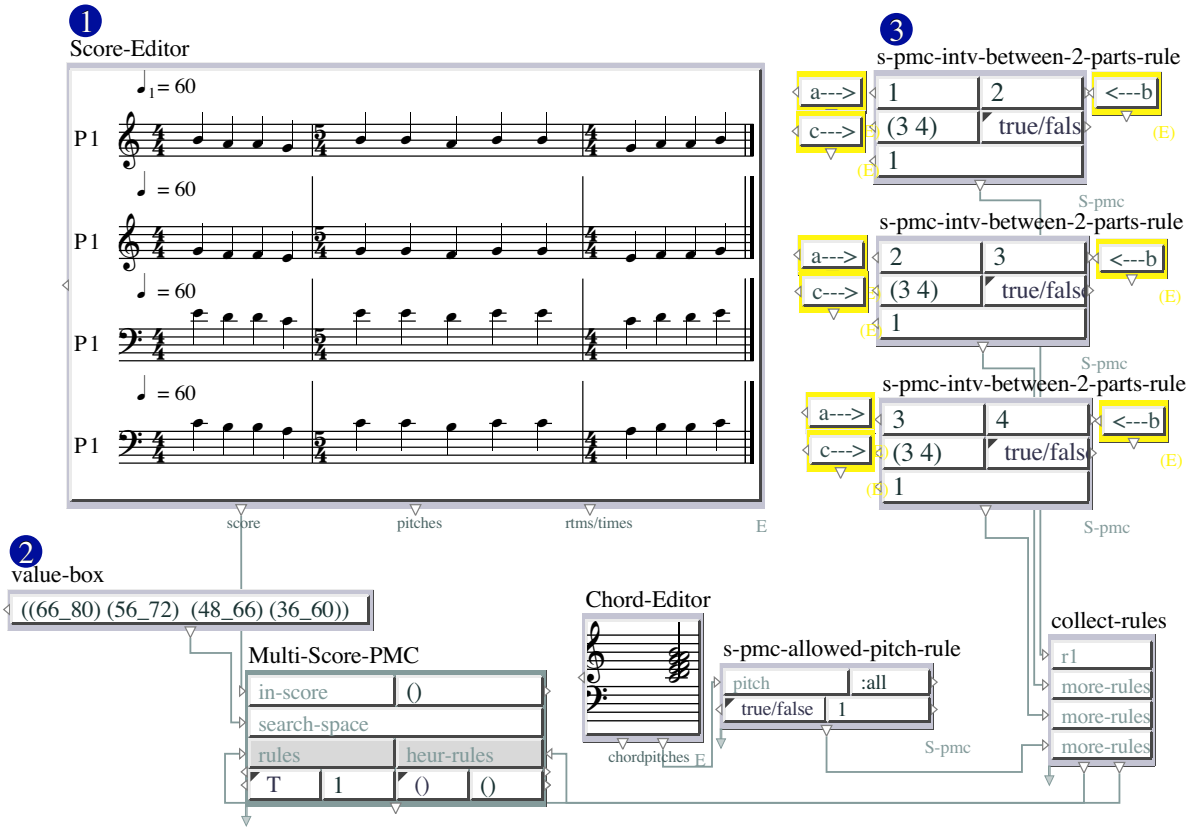

Figure 112: 2-02-12-allowed-interval-between-2-parts

3.3.13 13-Not-Allowed-Interval-between-2-Parts

2.02.13 - S-PMC-NOT-INTV-BETWEEN-2-PARTS-RULE
S-PMC-NOT-INTV-BETWEEN-2-PARTS-RULE [3] deals with vertical intervals between two voices.
In the input [a] you define a part and in [b] another one Then in [c] you set which intervals are NOT allowed between the voices defined in [a] and [b].

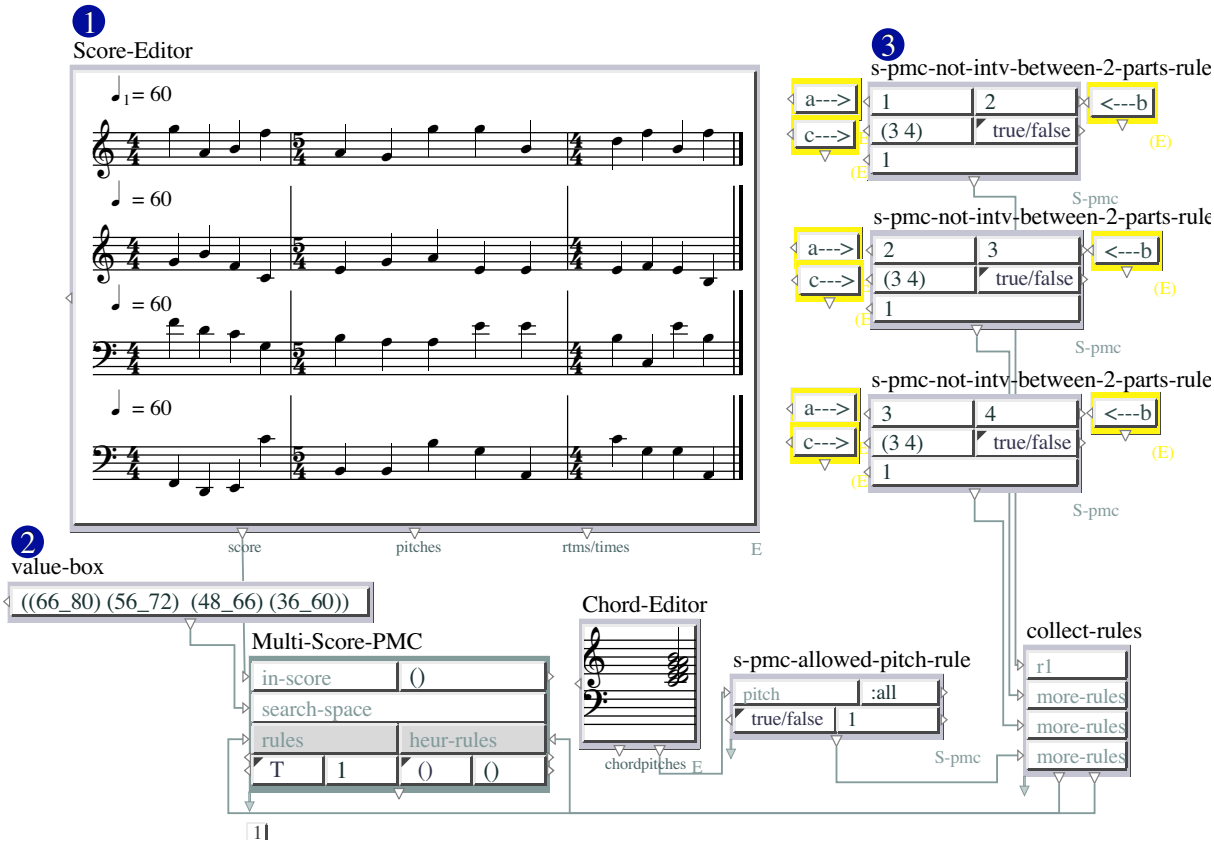

Figure 113: 2-02-13-not-allowed-interval-between-2-parts

3.3.14 14-To-Be-Done

2.02.14

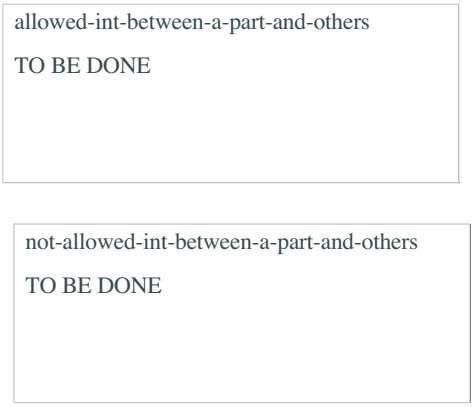

Figure 114: 2-02-14-to-be-done

3.3.15 15-Smaller-and-Bigger-Int-between-Parts

2.02.15 - SMALLER-AND-BIGGER-INT-BETWEEN-PARTS
S-PMC-SMALLER-INT-BETWEEN-2-PARTS-RULE [3] deals with vertical intervals between two parts, defined in [a] and [b] inputs.
The solution is forced to include harmonic intervals smaller than the given interval value [c].
S-PMC-BIGGER-INT-BETWEEN-2-PARTS-RULE [4] does the opposite, so the solution is forced to included harmonic intervals bigger than a given interval [c] between two voices.

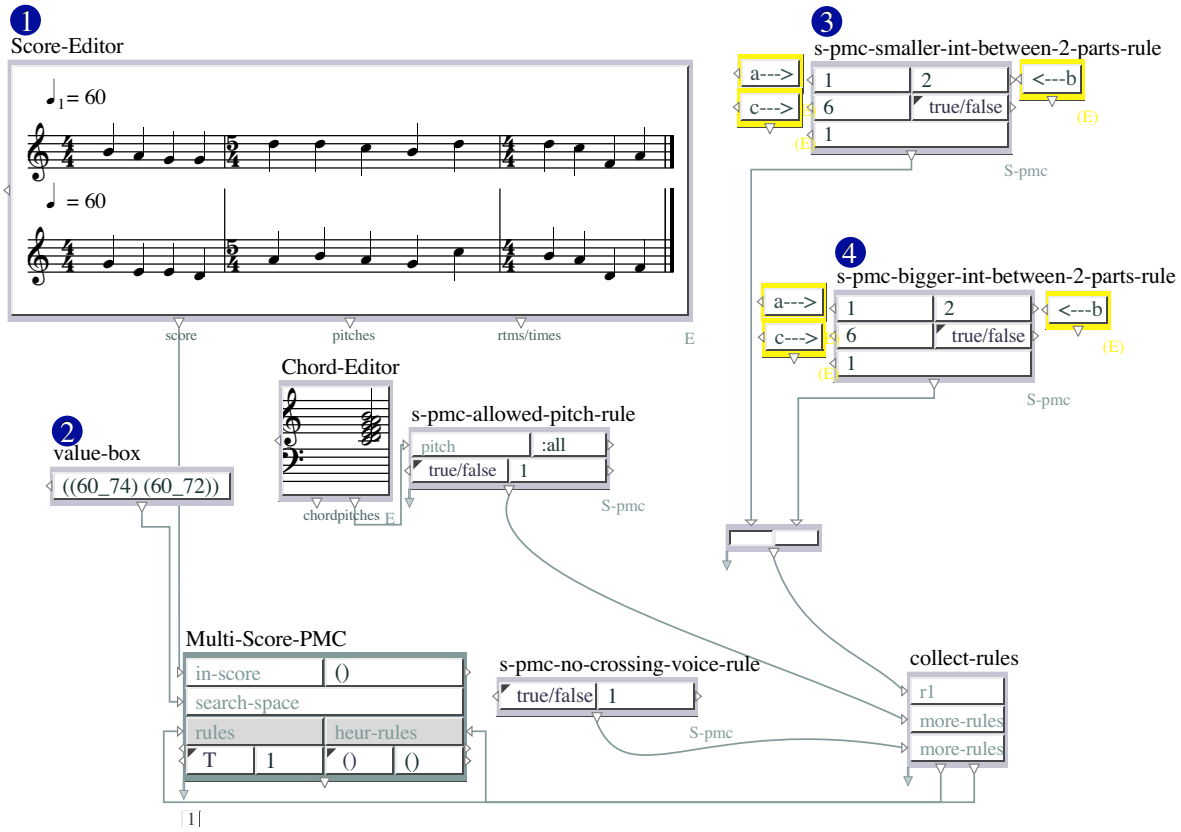

Figure 115: 2-02-15-smaller-and-bigger-int-between-parts

3.3.16 16-Forbidden-Interval-Relation

2.02.16 - S-PMC-FORBIDDEN-INT-RELATION-BETWEEN-2-PARTS-RULE
This rule is not so easy to understand. It is a generalisation of the ancient 'false relation'. This rule does not admit, in a close succession in two independent parts, that two different pitches can be swapped between the two parts. For instance, let take the pitches C and D and just two voices. The S-PMC-FORBIDDEN-INT-RELATION-BETWEEN-2-PARTS-RULE [3] does not admit that, if D follows C in the first voice, and in the second voice D is the second note, C can not be before D in the second voice.
In [2] you define the search space.
In [a] you set the first voice, and in [b] the second one. In [c] you have to specify which of the intervals between the two voices forbids the succession of pitches.
In [1] you chose which Score-Editor you want to use. BE CAREFUL: the first Score-Editor [d], as I constrained the first two pitches in the two voices, does not have solutions. That is because I am asking not to have two consecutive pitches producing a major second (set in [c]), swapped between the two voices. If in [1] you chose [e], you will get several possible solutions.
With S-PMC-ALLOWED-PITCHES-RULE [4] I am asking only C major scale notes.

(Please look at the tutorial n. 2.01.3.1 of this same tutorial library).
With the S-PMC-NO-CROSSING-VOICE-RULE [5] I am looking for solutions in which the first voice has higher notes than the second voice. (Please look at the tutorial n. 2.03.01 of this same tutorial library).

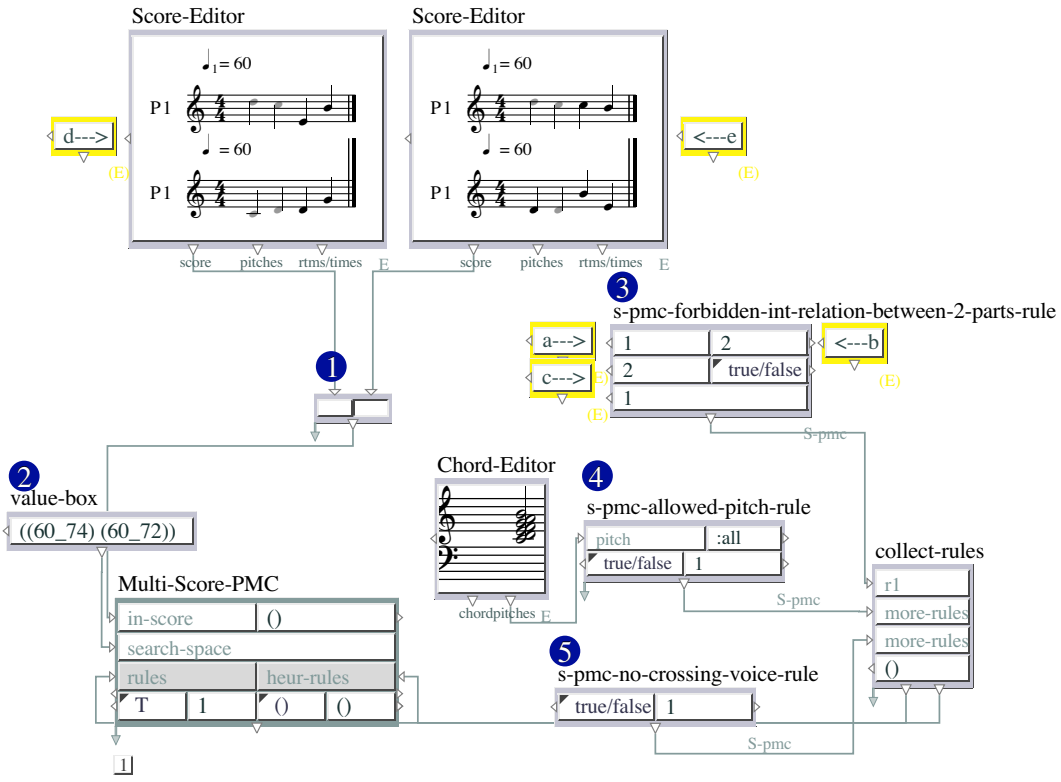

Figure 116: 2-02-16-forbidden-interval-relation

3.3.17 17-Not-N-Consecutive-Equal-Intervals

2.02.17 - S-PMC-NOT-N-CONSECUTIVE-HARM-INT-RULE
S-PMC-NO-CROSSING-VOICE-RULE [3] forbids two or more voices to cross each other's pitches. (See also the tutorial 2.03.01 - NO-CROSSING-VOICE-RULE).
S-PMC-NOT-N-CONSECUTIVE-HARM-INT-RULE [4] forbids two given voices, set in [a] and [b], to have more than a given number [c] of consecutive equal harmonic intervals.

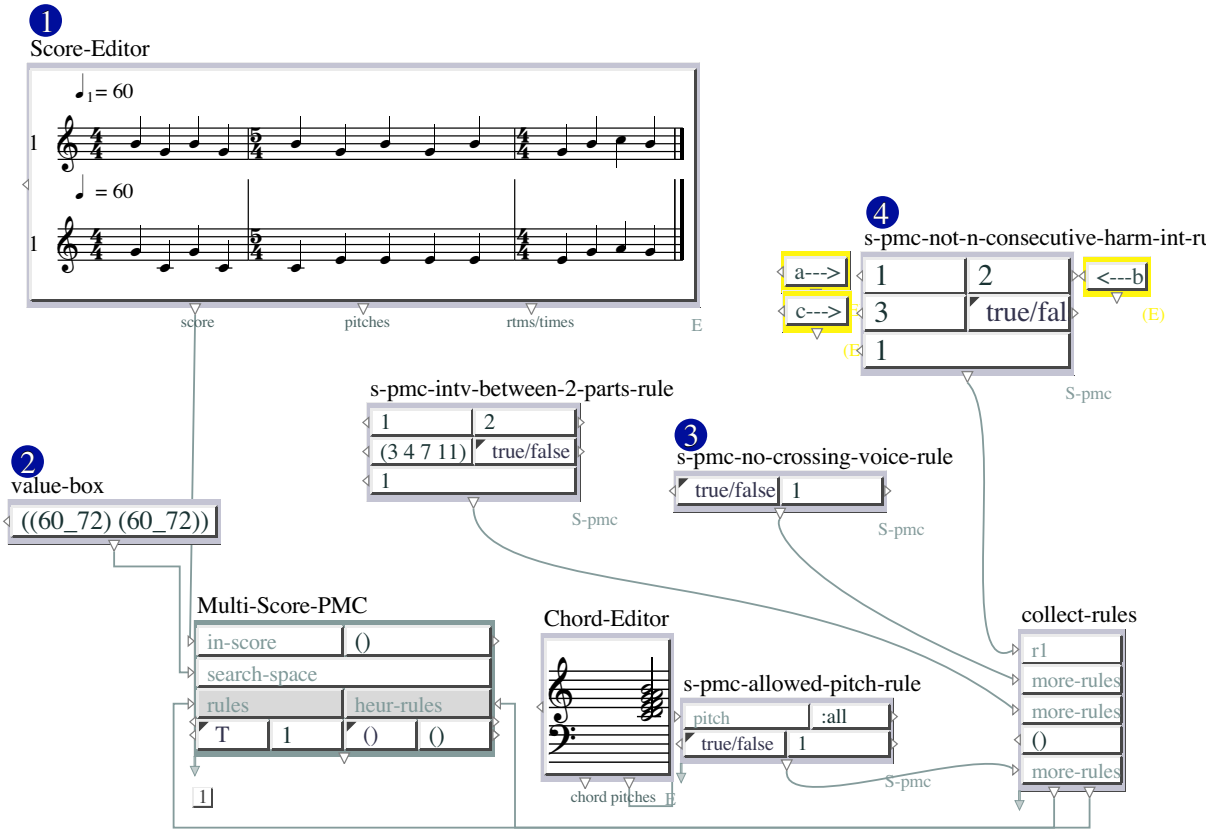

Figure 117: 2-02-17-not-n-consecutive-equal-intervals

3.3.18 18-Not-N-Same-Directions

2.02.18 - S-PMC-NOT-N-SAME-DIRECTIONS-RULE
S-PMC-NOT-N-SAME-DIRECTIONS-RULE [3] forbids two given voices, set in [a] and [b], to go more than a given number of times [c] in the same direction.

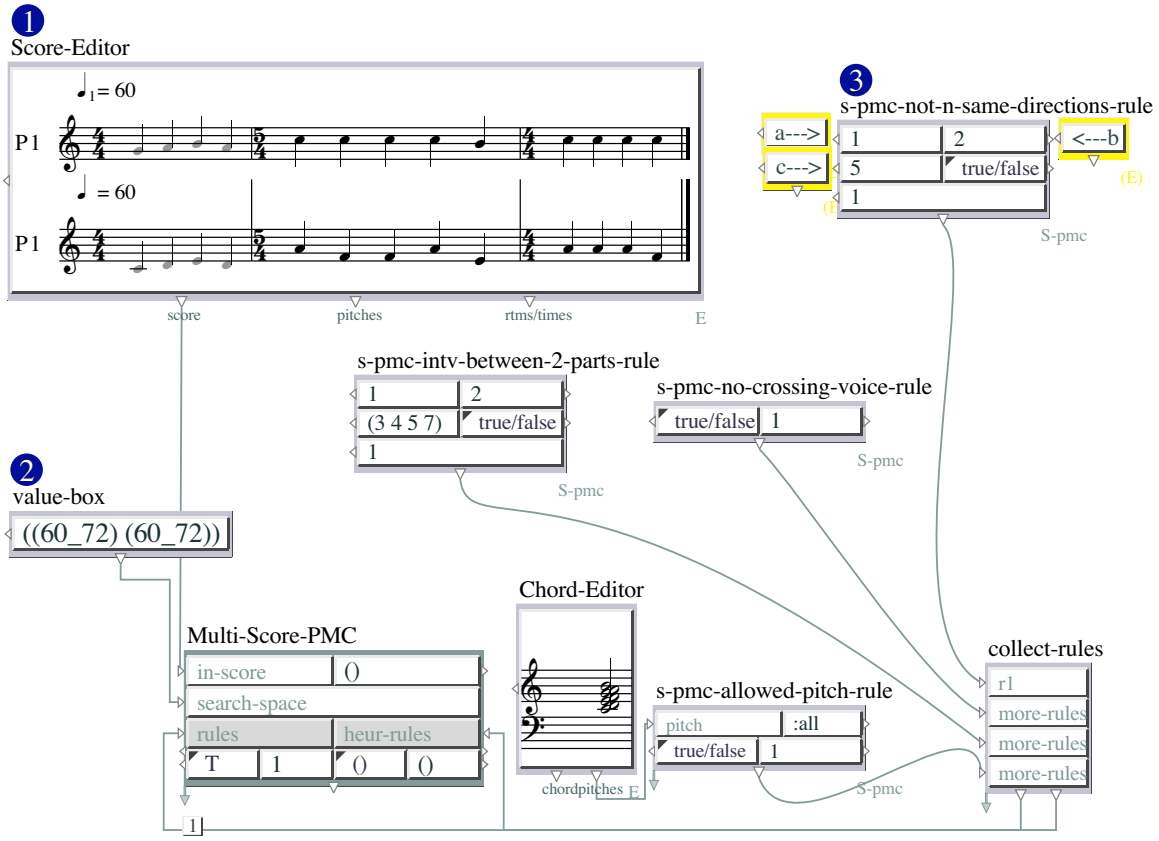

Figure 118: 2-02-18-not-n-same-directions

3.3.19 BPF-Delay

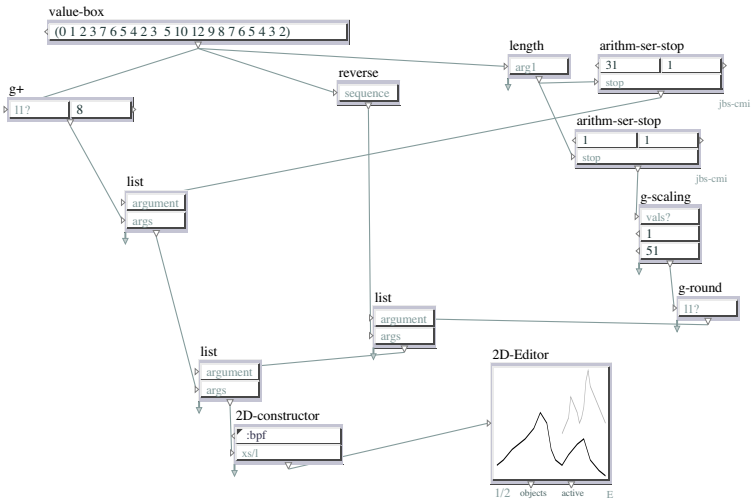

Figure 119: Bpf-Delay

3.4 03-Voice-Leading-Rules

3.4.1 01-No-Crossing-Voice-Rule

2.03.01 - S-PMC-NO-CROSSING-VOICE-RULE
S-PMC-NO-CROSSING-VOICE-RULE [3] does not allow : - the first voice to have lower notes than the second voice, - the second voice to have lower notes than the third voice, - the third voice to have lower notes than the fourth, and so on...
ATTENTION In this example there are only four voices, but THERE IS NO LIMITATION in terms of number of voices.

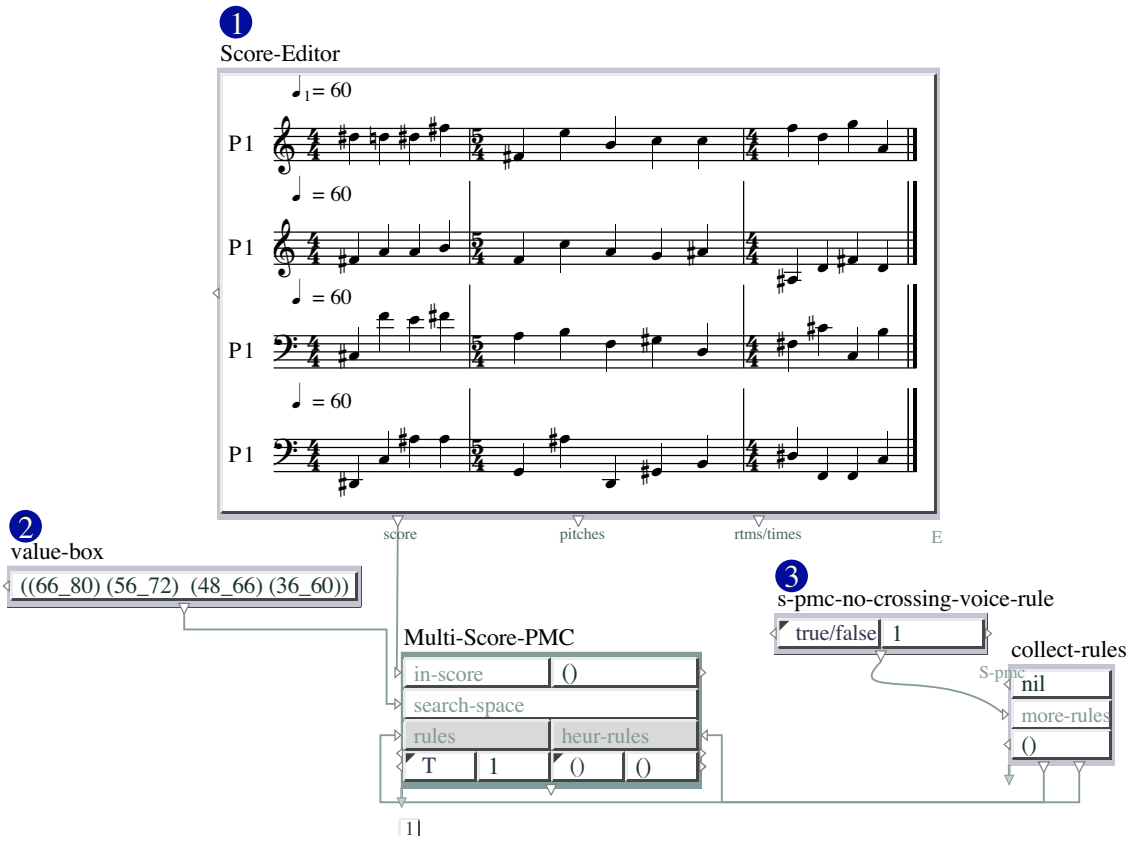

Figure 120: 2-03-01-no-crossing-voice-rule

3.4.2 02-No-Open-Parallel-Rule

2.03.02 - S-PMC-NO-OPEN-PARALLEL-RULE
S-PMC-NO-OPEN-PARALLEL-RULE [5] forbids two voices to have two consecutive octaves (0 as an interval of unison [a]) and any consecutive fifths (7 as an interval of fifth [a]).

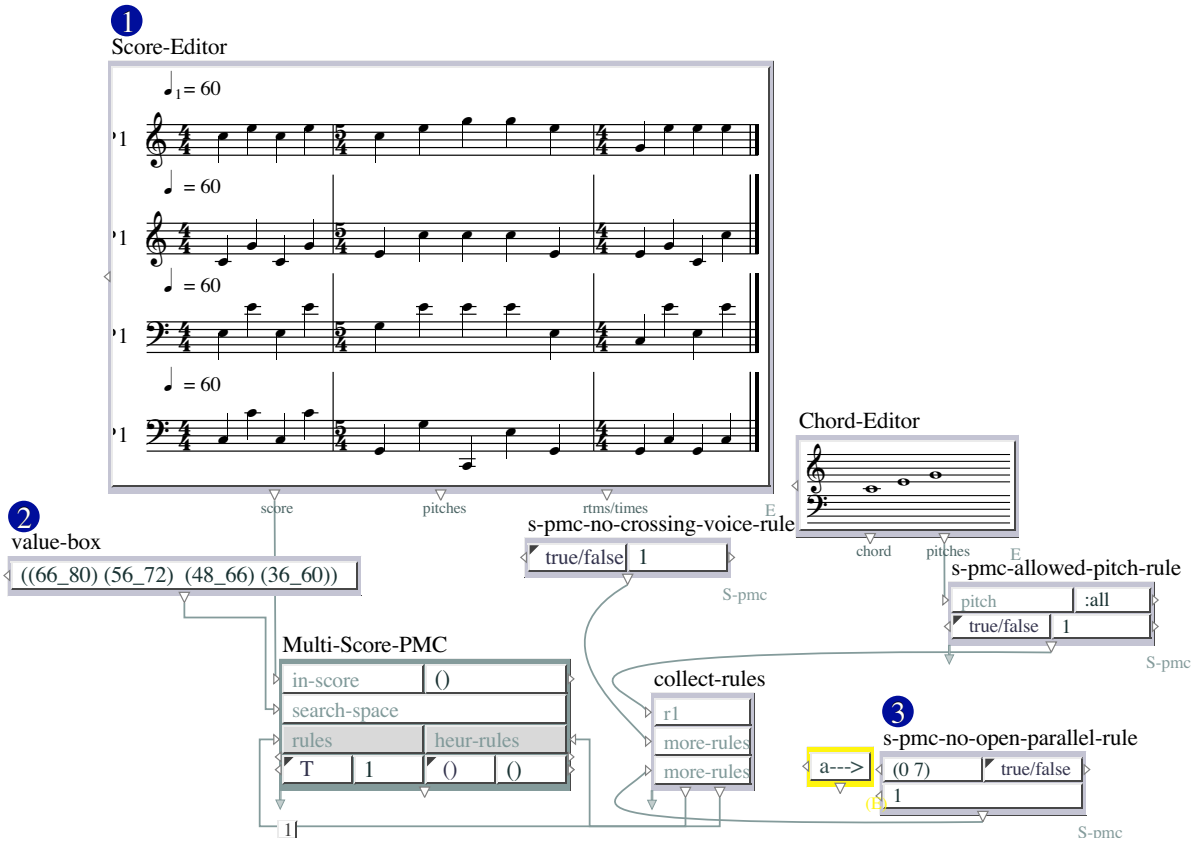

Figure 121: 2-03-02-no-open-parallel-rule

3.4.3 03-Forbidden-Succession-Rule

2.03.03 - S-PMC-FORBIDDEN-SUCCESSION-RULE
The S-PMC-FORBIDDEN-SUCCESSION-RULE [3] forbids to have a given succession of intervals between two voices.
In this case we define in [a] an augmented fourth (6 semitones) and in [b] a perfect fifth (7 semitones). The rule forbids any solution to have, between two voices, an interval of augmented fourth followed by a perfect fifth.

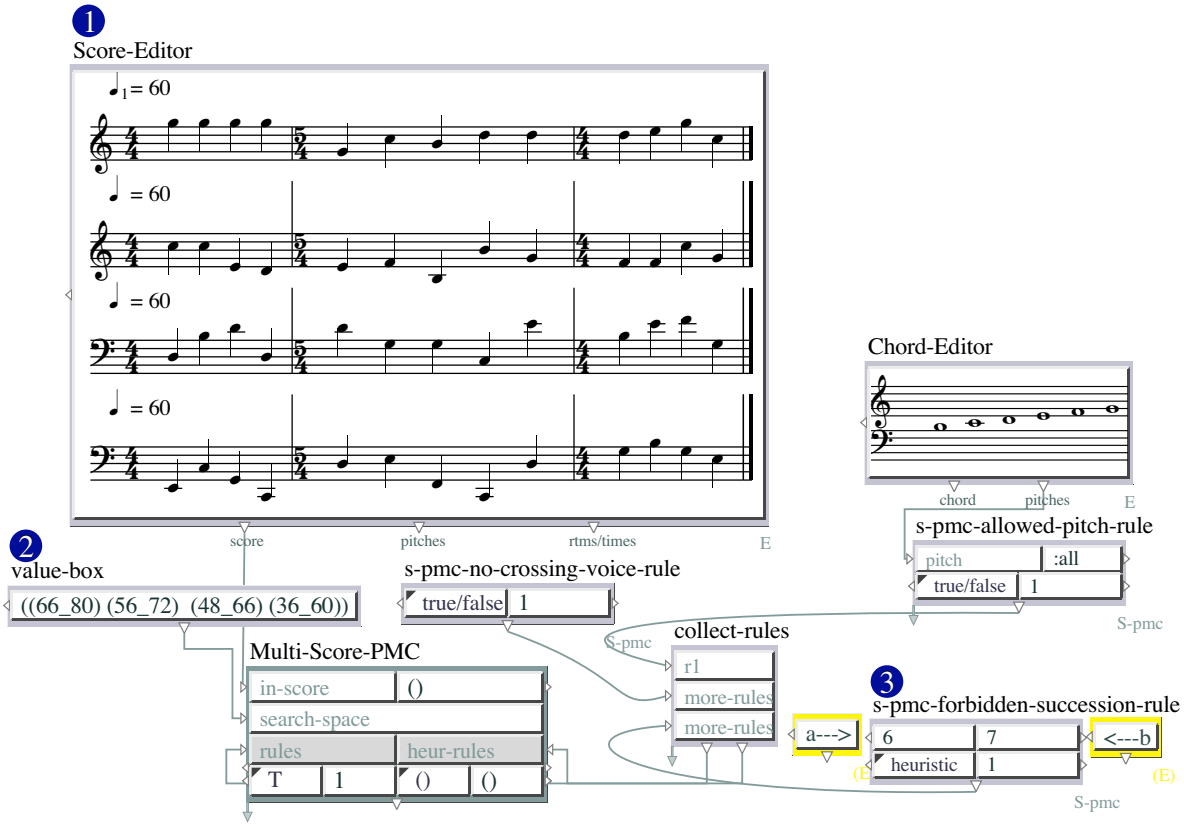

Figure 122: 2-03-03-forbidden-succession-rule

3.4.4 04-Hidden-Parallel-Rule

2.03.04 - S-PMC-HIDDEN-PARALLEL-RULE
S-PMC-HIDDEN-PARALLEL-RULE [3] needs to set in [a] which are the intervals to focus on, in this case the octave (set with 0 as a unison interval) and the perfect fifth (set with 7).
This rule only permits to get a fifth or an octave when the upper voice (between any two voices) is moving stepwise.

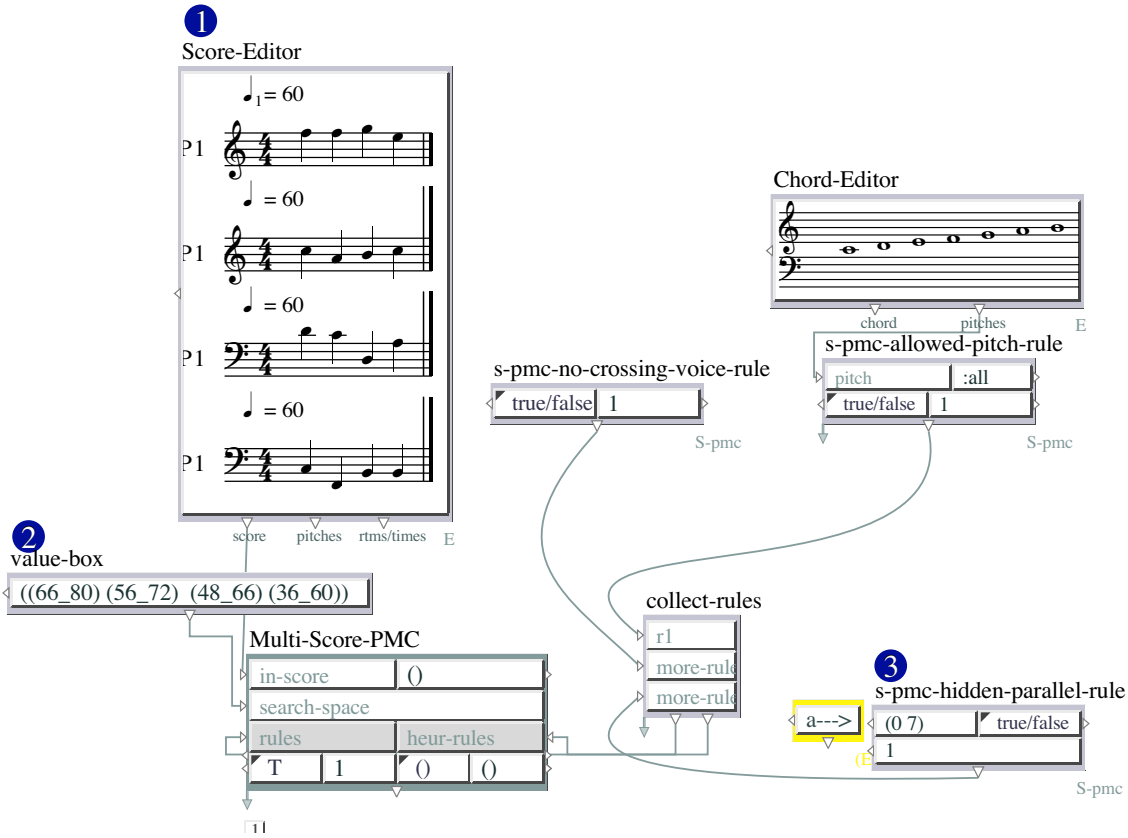

Figure 123: 2-03-04-hidden-parallel-rule

3.5 04-Create-Expressions-Tools

3.5.1 01-Create-Individual-Expression

2.04.01 - CREATE-INDIVIDUAL-EXPRESSION
All the expression functions work with the ENP-SCRIPT box [4]. The goal is to apply a given rule (in this example the S-PMC-ALLOWED-PITCH-RULE [6]), only where a given expression can be found. So first of all type R after having selected the ENP-SCRIPT [4]. You will see that in the SCORE-EDITOR [1] all expression symbols will disappear.
Now evaluate the ENP-SCRIPT [4]. According to the boxes connected to the COLLECT-SCRIPT-RULES [5] you will generate some expressions in the SCORE-EDITOR [1].
CREATE-INDIVIDUAL-EXPRESSION [3] works like this : In [a] you define on which note you want to assign the expression. In [b] you define which expression. In [c] you define in which voice you want to set the expressions : 1 is for the first voice, 2 for the second and so on.
ATTENTION Indexes are written in constraints numeration, they are not like the Lisp nth notation. So, do not forget that 1 is the first note, 2 for the second, 3 for the third and so on. To know which are the expressions, please look at the ENP tutorial.

Now, if you evaluate the Multi-Score-PMC [7] you will see that S-PMC-ALLOWED-PITCH-RULE [6] is applied only to the notes having a :Conductor-Mark expression. On these notes, the only pitch allowed is the C modulo 12 [d].

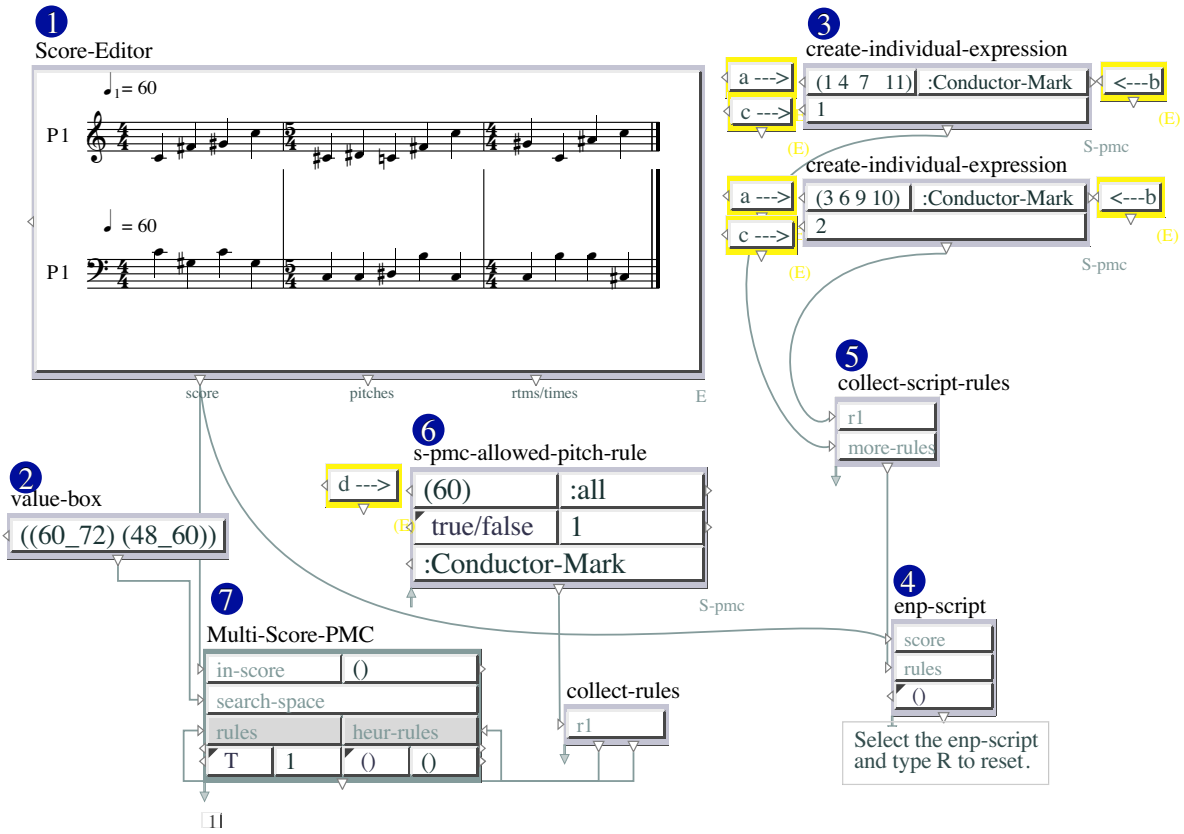

Figure 124: 2-04-01-create-individual-expression

3.5.2 02-Create-Group-Expression

2.04.02 - CREATE-GROUP-EXPRESSIONS
All the expression functions work with the ENP-SCRIPT box [4]. The goal is to apply a given rule (in this example the S-PMC-ALLOWED-PITCH-RULE [6]), only where a given expression can be found. So first of all type R after having selected the ENP-SCRIPT [4]. You will see that in the SCORE-EDITOR [1] all expression symbols will disappear.
Now evaluate the ENP-SCRIPT [4]. According to the boxes connected to the COLLECT-SCRIPT-RULES [5] you will generate some expressions in the SCORE-EDITOR [1].
CREATE-GROUP-EXPRESSION [3] works like this : In [a] you define on which group of notes you want the assign the expression.. In [b] you define which expression. In [c] you define in which voice you want to set the expressions : 1 is for the first voice, 2 for the second and so on.
ATTENTION Indexes are written in constraints numeration, they are not like the Lisp

nth notation. If you put (1 7), that means that from the first note to the seventh note you want to apply an expression. In this example the expression is a :slur. To know which are the expressions, please look at the ENP tutorial.
Now, if you evaluate the Multi-Score-PMC [7] you will see that S-PMC-ALLOWED-PITCH-RULE [6] is applied only on the notes having the :slur expression. On these notes, the only pitch allowed is the C modulo 12 [d].

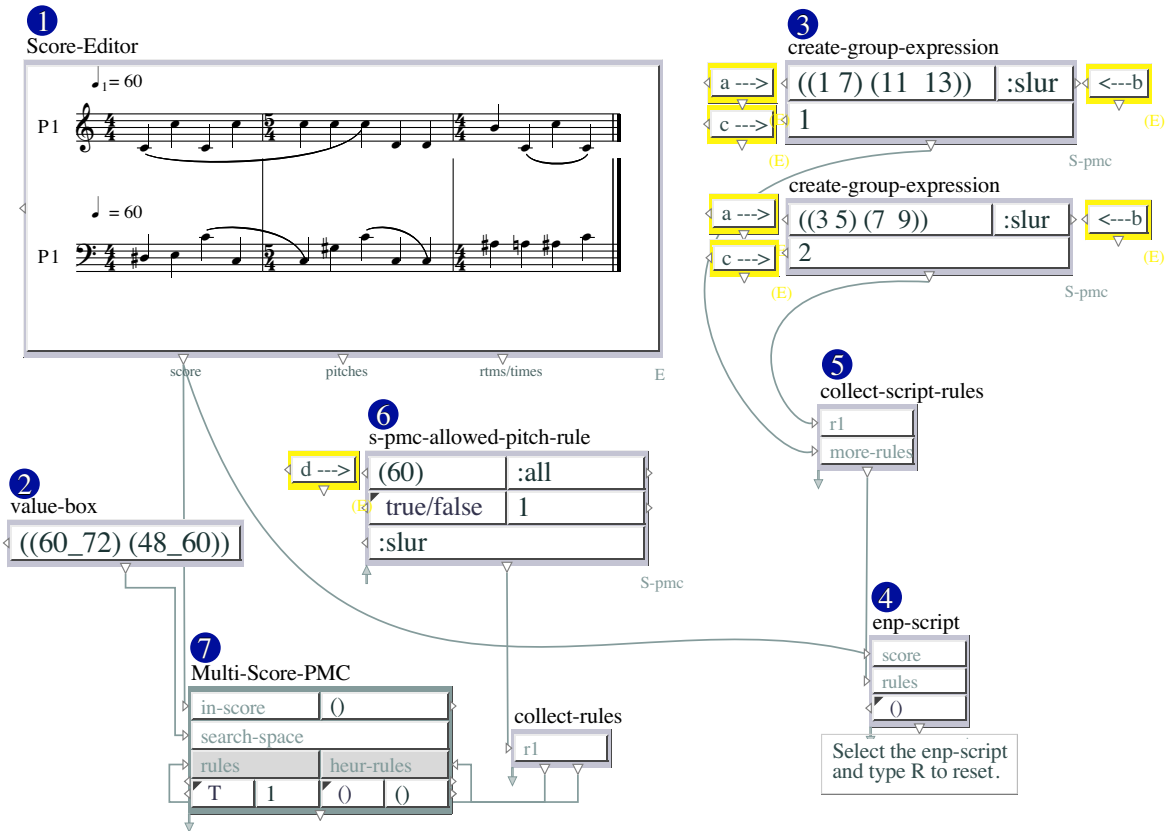

Figure 125: 2-04-02-create-group-expression

3.5.3 03-Create-Face-Value-Expression

2.04.03 - CREATE-FACE-VALUE-EXPRESSION
All the expression functions work with the ENP-SCRIPT box [4]. The goal is to apply a given rule (in this example the S-PMC-ALLOWED-PITCH-RULE [6]), only where a given expression can be found. So first of all type R after having selected the ENP-SCRIPT [4]. You will see that in the SCORE-EDITOR [1] all expression symbols will disappear.
Now evaluate the ENP-SCRIPT [4]. According to the boxes connected to the COLLECT-SCRIPT-RULES [5] you will generate some expressions in the SCORE-EDITOR [1].
CREATE-GROUP-EXPRESSION [3] works like this : In [a] you define on which rhythmic values you want to assign the expression. In [b] you define which expression. In [c]

you define in which voice you want to set the expressions : 1 is for the first voice, 2 for the second and so on.
In this example the expression is a :slur. To know which are the expressions, please look at the ENP tutorial.
Now, if you evaluate the Multi-Score-PMC [7] you will see that S-PMC-ALLOWED-PITCH-RULE [6] is applied only to the notes having an :accent expression. On these notes, the only pitch allowed is the C modulo 12 [d].

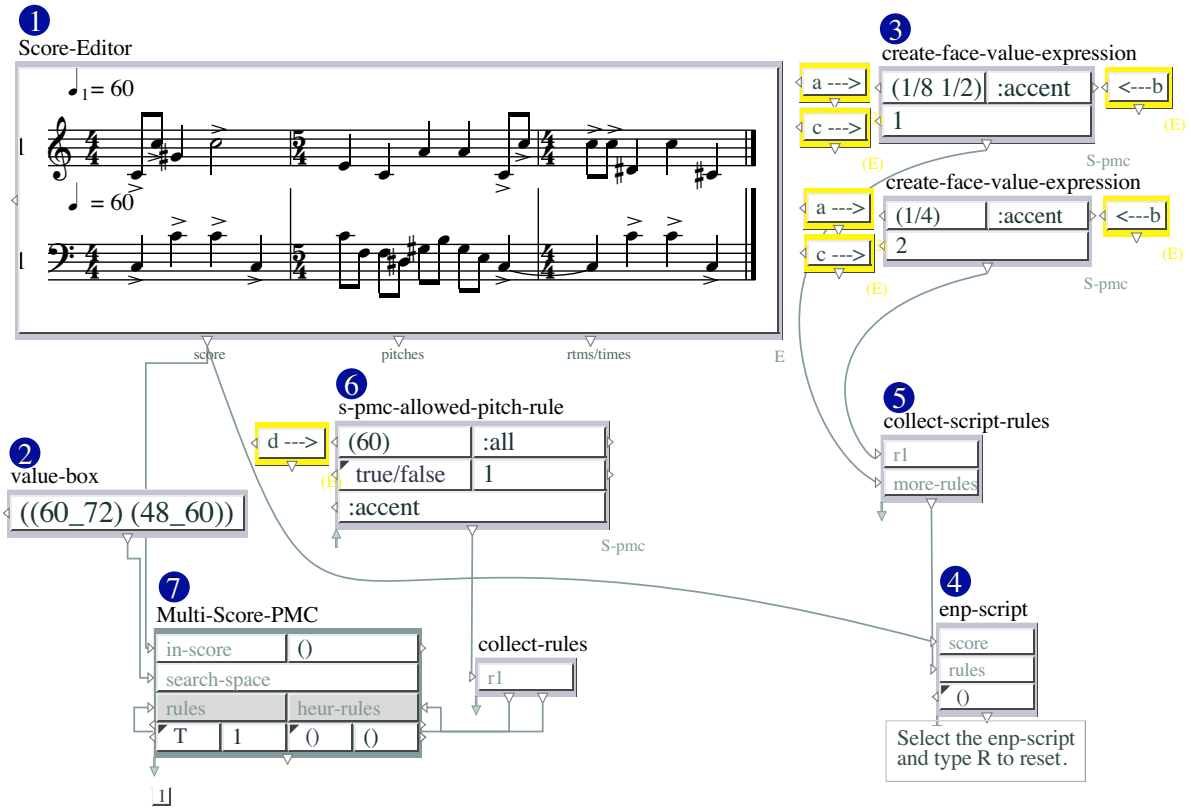

Figure 126: 2-04-03-create-face-value-expression

3.5.4 04-Create-Expression-on-Note-Sequence

2.04.04 - CREATE-EXPRESSION-ON-NOTE-SEQUENCE
All the expression functions work with the ENP-SCRIPT box [4]. The goal is to apply a given rule (in this example the S-PMC-ALLOWED-PITCH-RULE [6]), only where a given expression can be found. So first of all type R after having selected the ENP-SCRIPT [4]. You will see that in the SCORE-EDITOR [1] all expression symbols will disappear.
Now evaluate the ENP-SCRIPT [4]. According to the boxes connected to the COLLECT-SCRIPT-RULES [5] you will generate some expressions in the SCORE-EDITOR [1].
CREATE-EXPRESSION-ON-NOTE-SEQUENCE [3] works like this : In [a] you define

which expression (a :slur in the example) you want to assign. In [b] you define in which voice you want to set the expressions. In this case ':all' means that the expression will be applied to all voices.
CREATE-EXPRESSION-ON-NOTE-SEQUENCE [3] will create a slur on all sequences of single notes. When a chord appears, the slur is closed.
Now, if you evaluate the Multi-Score-PMC [7] you will see that S-PMC-ALLOWED-PITCH-RULE [6] is applied only to the notes having a :slur expression. On these notes, the only pitch allowed is the C modulo 12 [c].

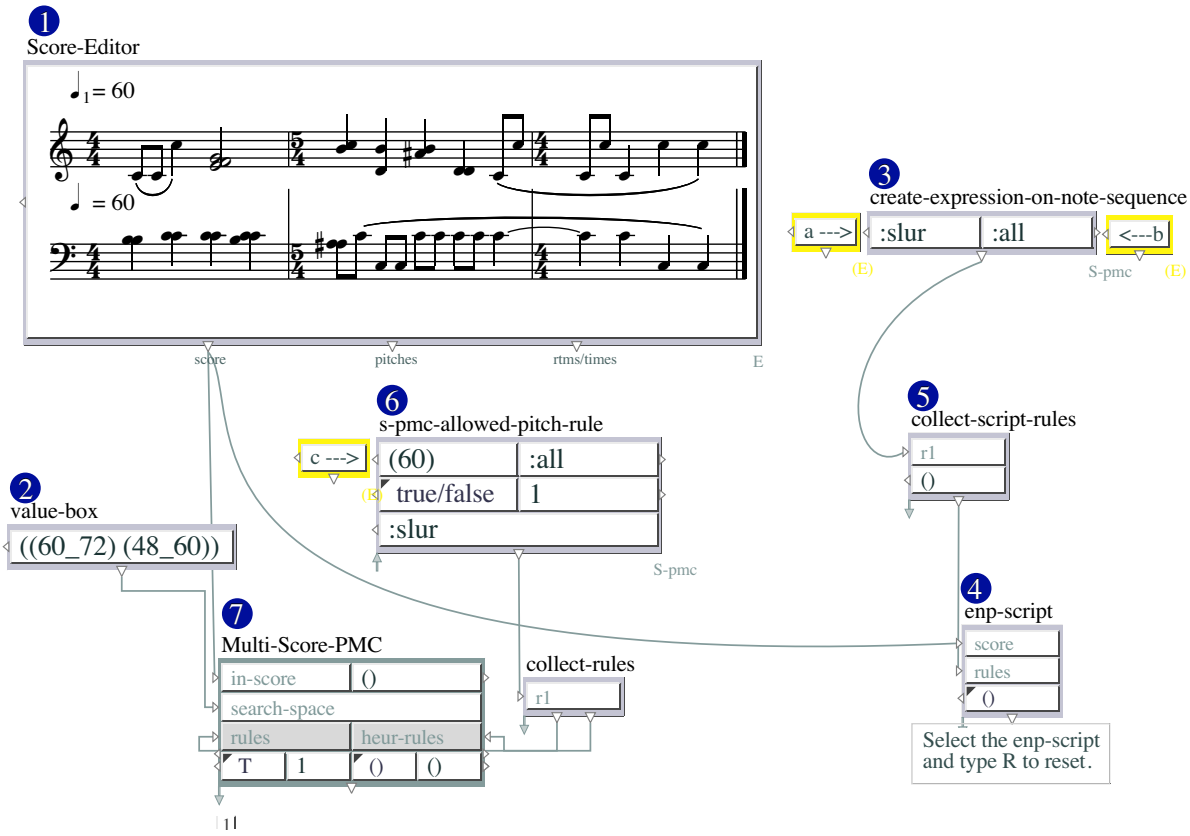

Figure 127: 2-04-04-create-expression-on-note-sequence

3.5.5 05-Create-Expression-on-Chord-Sequence

2.04.05 - CREATE-EXPRESSION-ON-CHORD-SEQUENCE
All the expression functions work with the ENP-SCRIPT box [4]. The goal is to apply a given rule (in this example the S-PMC-ALLOWED-PITCH-RULE [6]), only where a given expression can be found. So first of all type R after having selected the ENP-SCRIPT [4]. You will see that in the SCORE-EDITOR [1] all expression symbols will disappear.
Now evaluate the ENP-SCRIPT [4]. According to the boxes connected to the COLLECT-SCRIPT-RULES [5] you will generate some expressions in the SCORE-EDITOR [1].

CREATE-EXPRESSION-ON-CHORD-SEQUENCE [3] works like this : In [a] you define which expression (a :slur in the example) you want to assign. In [b] you define in which voice you want to set the expressions. In this case ':all' means that the expression will be applied to all voices.
CREATE-EXPRESSION-ON-CHORD-SEQUENCE [3] will create a slur on all sequences of chords. When a single note appears, the slur is closed.
Now, if you evaluate the Multi-Score-PMC [7] you will see that S-PMC-ALLOWED-PITCH-RULE [6] is applied only to the notes having a :slur expression. On these notes, the pitches allowed are C, E-flat and G, modulo 12 [c].

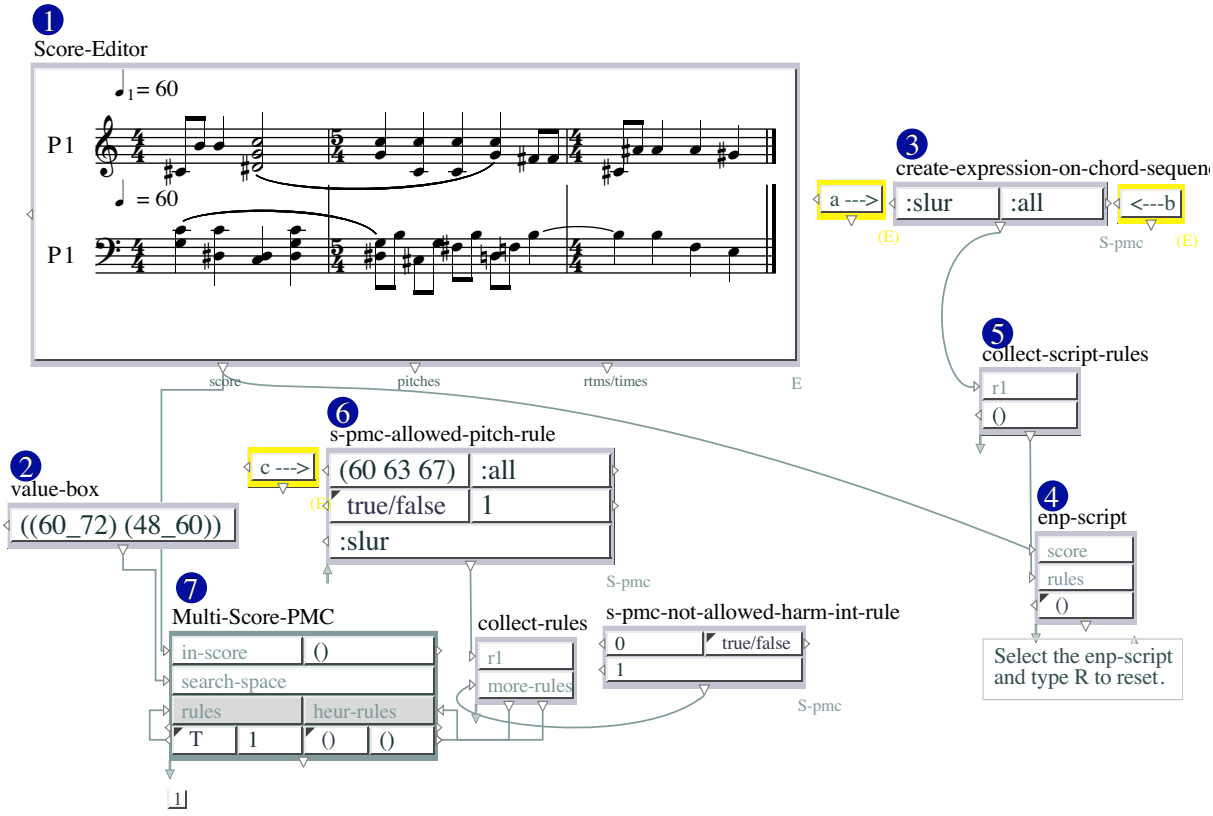

Figure 128: 2-04-05-create-expression-on-chord-sequence

3.5.6 06-Create-Expression-on-Grace-Note-Sequence

2.04.06 - CREATE-EXPRESSION-ON-GRACE-NOTE-SEQUENCE
All the expression functions work with the ENP-SCRIPT box [4]. The goal is to apply a given rule (in this example the S-PMC-ALLOWED-PITCH-RULE [6]), only where a given expression can be found. So first of all type R after having selected the ENP-SCRIPT [4]. You will see that in the SCORE-EDITOR [1] all expression symbols will disappear.

Now evaluate the ENP-SCRIPT [4]. According to the boxes connected to the COLLECT-SCRIPT-RULES [5] you will generate some expressions in the SCORE-EDITOR [1].
CREATE-EXPRESSION-ON-GRACE-NOTE-SEQUENCE [3] works like this : In [a] you define which expression (a :slur in the example) you want to assign. In [b] you define in which voice you want to set the expressions. In this case ':all' means that the expression will be applied to all voices.
CREATE-EXPRESSION-ON-GRACE-NOTE-SEQUENCE [3] will create a slur on all sequences of grace notes.
Now, if you evaluate the Multi-Score-PMC [7] you will see that S-PMC-ALLOWED-PITCH-RULE [6] is applied only to the notes having a :slur expression. On these notes, the only pitch allowed is C, modulo 12 [c].

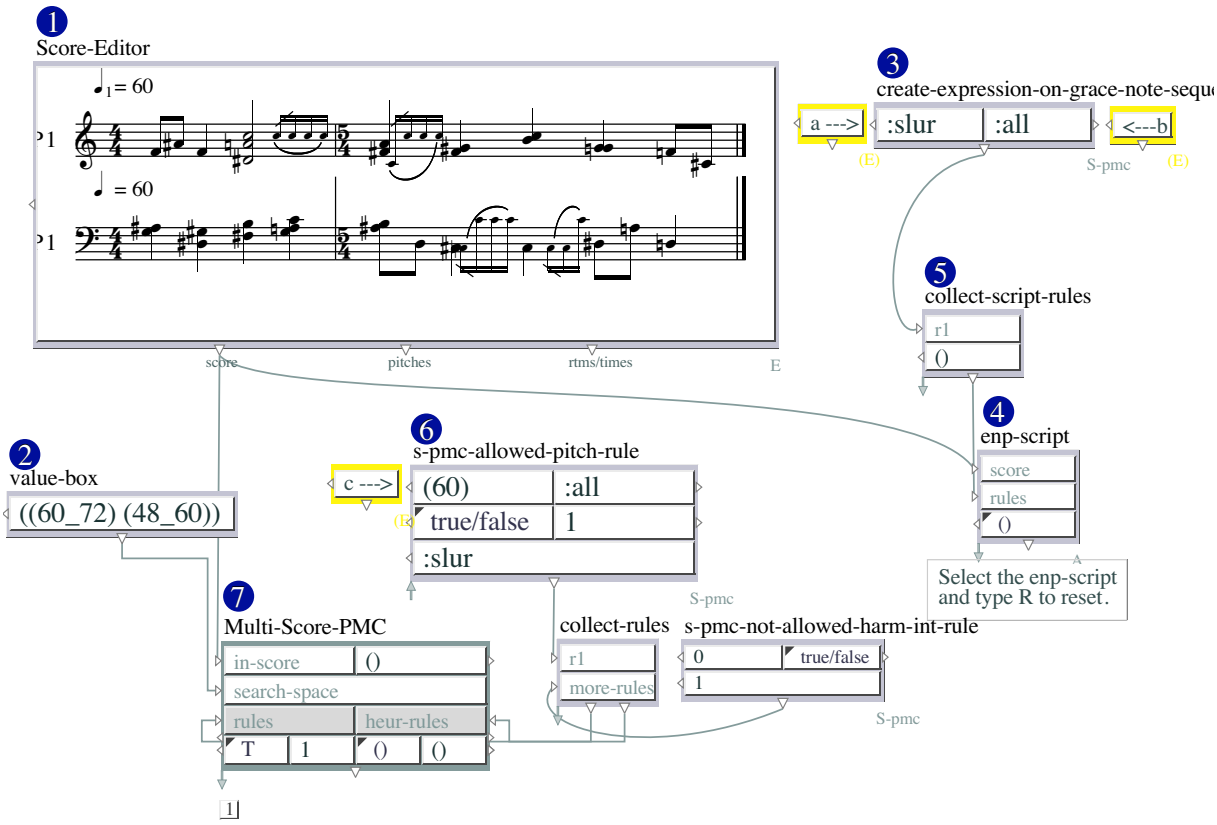

Figure 129: 2-04-06-create-expression-on-grace-note-sequence

3.5.7 07-Create-Expression-on-Main-Beat

2.04.07 - CREATE-EXPRESSION-ON-MAIN-BEAT
All the expression functions work with the ENP-SCRIPT box [4]. The goal is to apply a given rule (in this example the S-PMC-ALLOWED-PITCH-RULE [6]), only where a given expression can be found. So first of all type R after having selected the ENP-SCRIPT [4].

You will see that in the SCORE-EDITOR [1] all expression symbols will disappear.
Now evaluate the ENP-SCRIPT [4]. According to the boxes connected to the COLLECT-SCRIPT-RULES [5] you will generate some expressions in the SCORE-EDITOR [1].
CREATE-EXPRESSION-ON-MAIN-BEAT [3] works like this : In [a] you define which expression (a fermata in the example) you want to assign. In [b] you define in which voice you want to set the expressions. In this case ':all' means that the expression will be applied to all voices.
CREATE-EXPRESSION-ON-MAIN-BEAT [3] will create a fermata on all main beats of each measure.

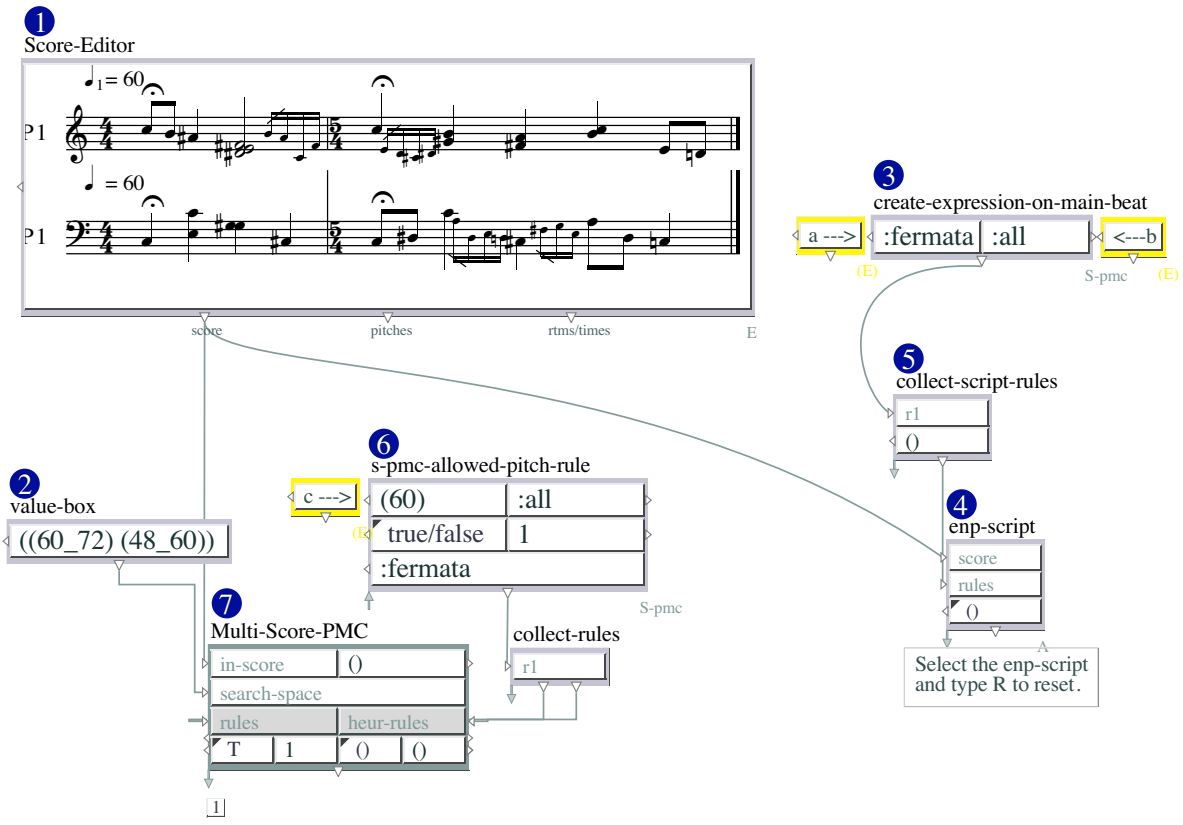

Figure 130: 2-04-07-create-expression-on-main-beat

3.5.8 08-Create-Expression-Not-on-Main-Beat

2.04.08 - CREATE-EXPRESSION-NOT-ON-MAIN-BEAT
CREATE-EXPRESSION-NOT-ON-MAIN-BEAT [3] works like this : In [a] you define which expression (an accent in the example) you want to assign. In [b] you define in which voice you want to set the expressions. In this case :all means that the expression will be applied to all voices.

CREATE-EXPRESSION-NOT-ON-MAIN-BEAT [3] will create a accent on every notes, except on main beats, of each measure.

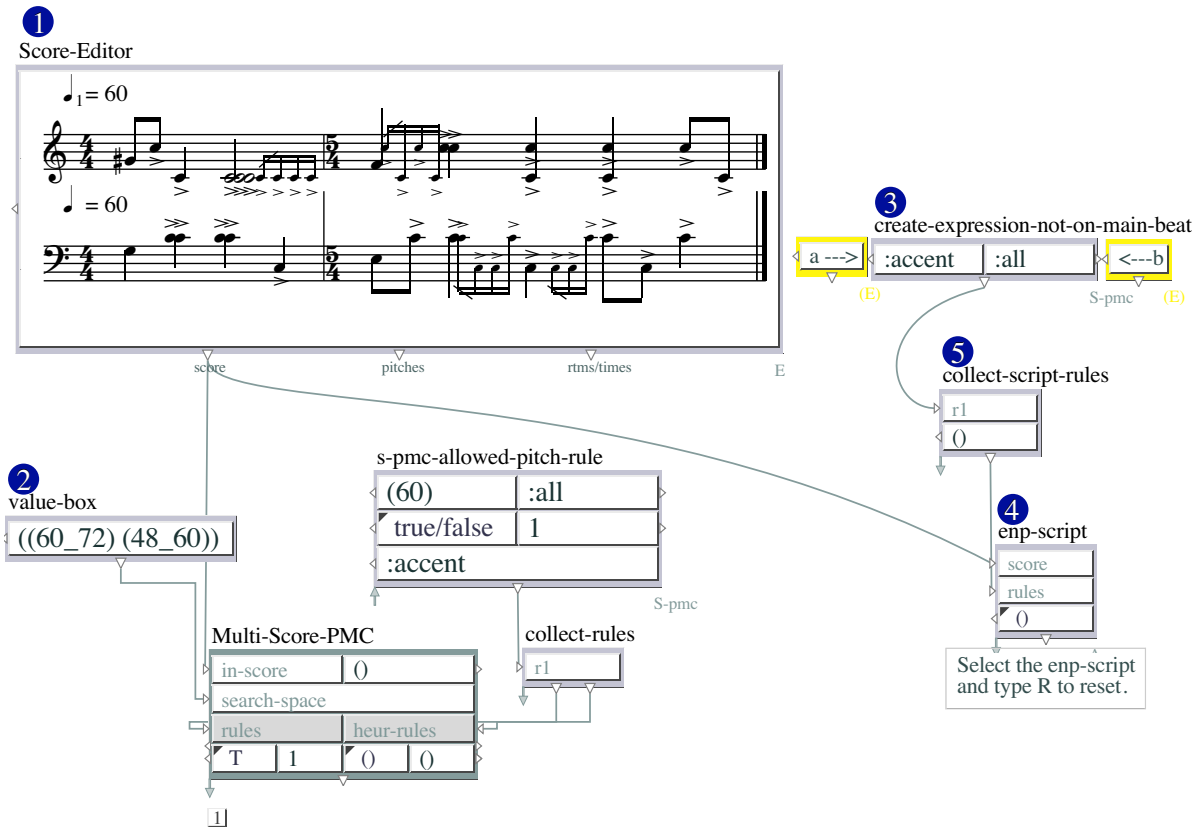

Figure 131: 2-04-08-create-expression-not-on-main-beat

3.5.9 09-Create-Expression-for-Beats

2.04.09 - CREATE-EXPRESSION-FOR-BEATS
CREATE-EXPRESSION-FOR-BEATS [3] works like this : In [a] you define on which beats, the second and the fifth in the example, of each measure you want to apply an expression. In [b] you define which expression, here a :fermata, you want to assign. In [c] you define in which voice you want to set the expressions. In this case, 2 means that the expression will be applied only to the second voice.
CREATE-EXPRESSION-FOR-BEATS [3] will create a fermata on all the second beats of each measure, and on the fifth of the last measure, as it is a (5 4) measure.

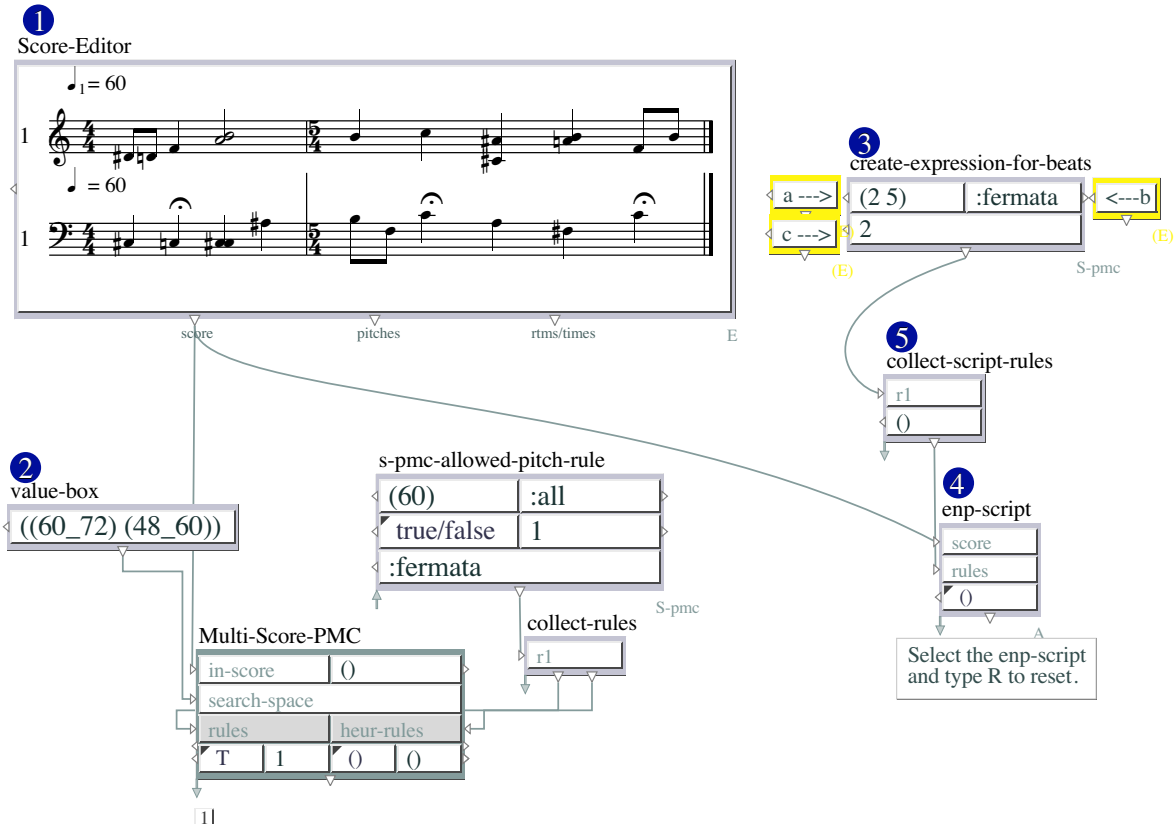

Figure 132: 2-04-09-create-expression-for-beats

3.5.10 10-Create-Expression-for-Measures

2.04.10 - CREATE-EXPRESSION-FOR-MEASURES
CREATE-EXPRESSION-FOR-MEASURES [3] works like this : In [a] you define on which measure, here the second, you want to apply an expression. In [b] you define which expression, here an :accent, you want to assign. In [c] you define in which voice you want to set the expressions. In this case 1 means that the expression will be applied only to the first voice.
CREATE-EXPRESSION-FOR-MEASURES [3] will create a series of accents on the second measure.

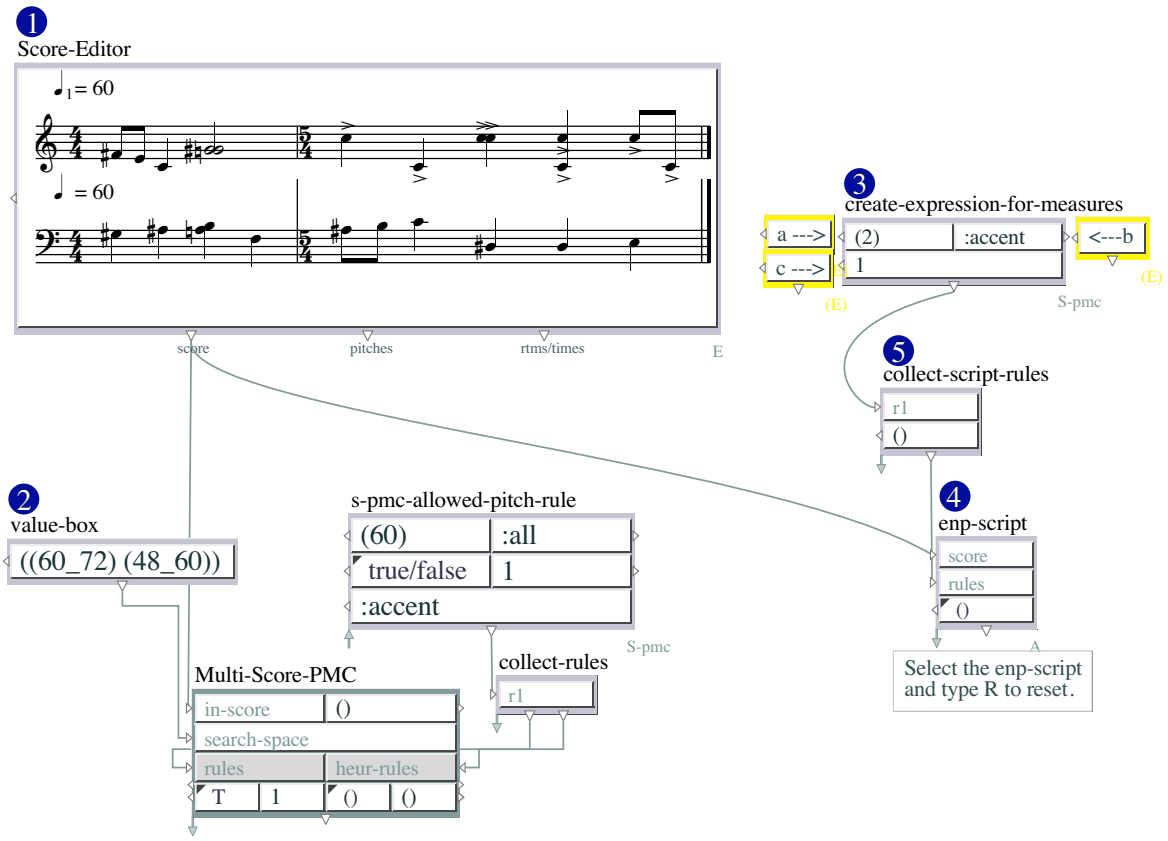

Figure 133: 2-04-10-create-expression-for-measures

4 0-Utils

4.1 Utils

This menu is dedicated to some functions useful to prepare datas for the Multi-PMC and the Multi-Score-PMC.

4.2 01-Collect-Rules

3.01 - COLLECT-RULES
This function [1] collects the rules and separate true/false from heuristic rules. Use the two outputs in order to connect to the Multi-PMC and the Multi-Score-PMC. The COLLECT-RULES [1] sends automatically a true/false rule or a heuristic one in the correct inputs.
The order in which you enter the rules in the COLLECT-RULES is not important. If some inputs of the COLLECT-RULES are empty, it does not affect the Multi-PMC and the Multi-Score-PMC.

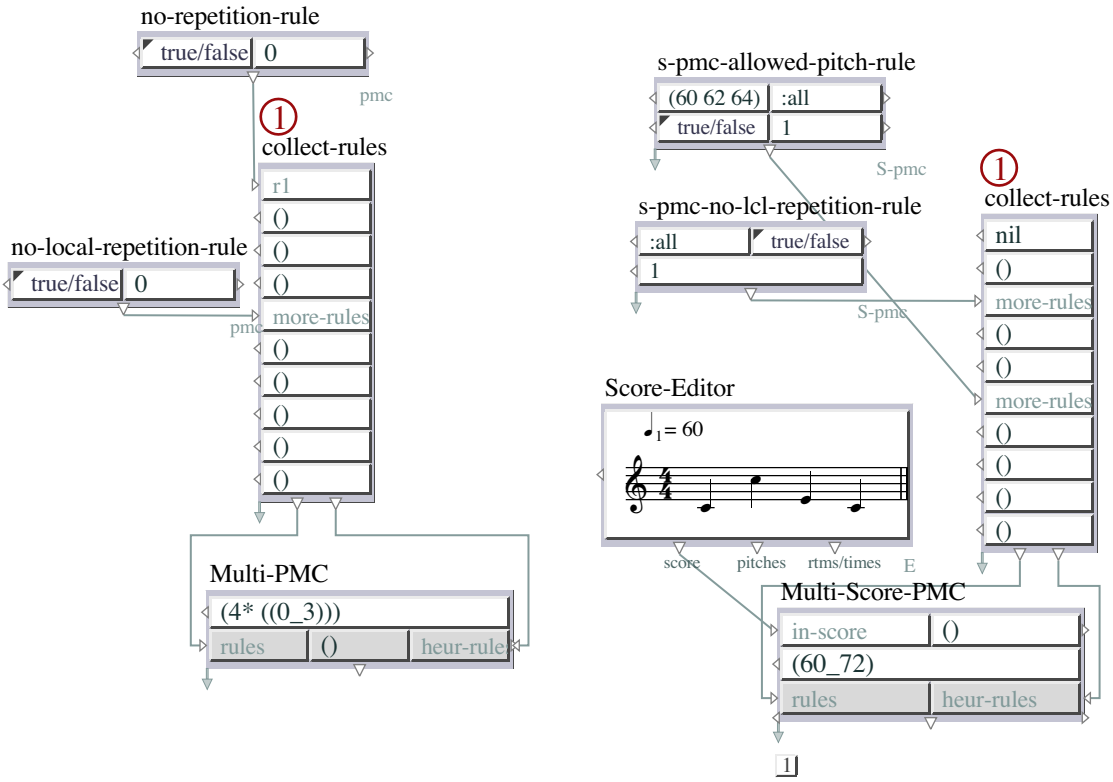

Figure 134: 3-01-collect-rules

4.3 02-Collect-Script-Rules

3.02 - COLLECT-SCRIPT-RULES
This function [1] collects the rules for the ENP-SCRIPT.
The order in which you enter the rules in the COLLECT-SCRIPT-RULES is not important. If some inputs of the COLLECT-SCRIPT-RULES are empty, it does not affect the ENP-SCRIPT.

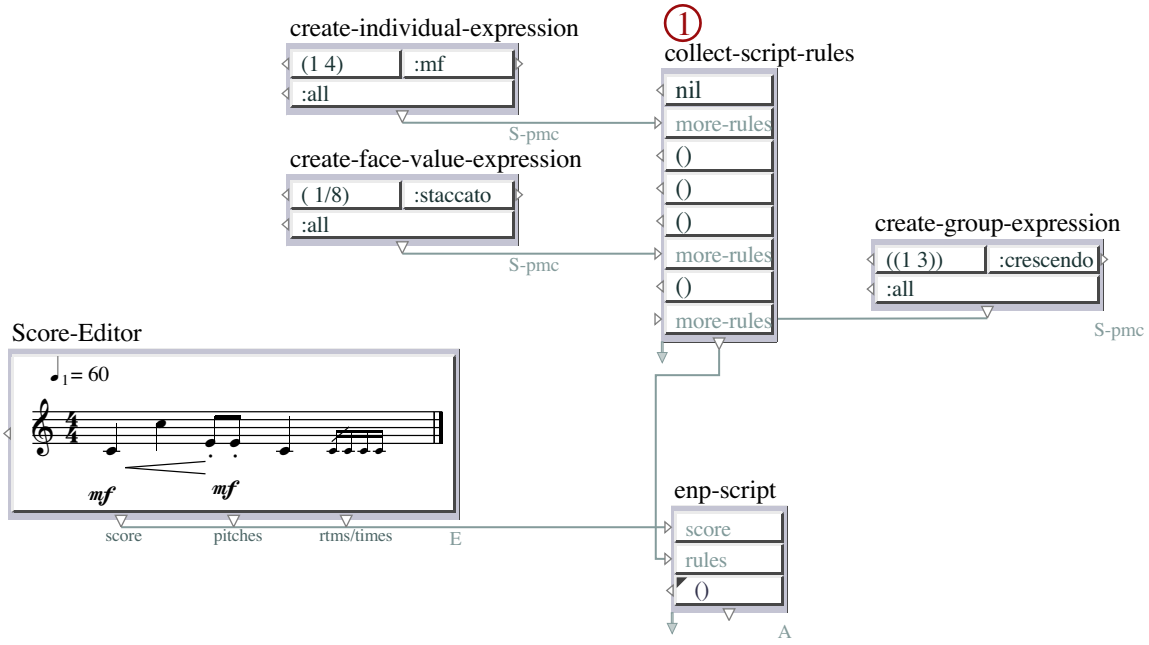

Figure 135: 3-02-collect-script-rules

4.4 03-Make-?1-and-Make-I1

3.03 - MAKE-?1-AND-MAKE-I1
[1] creates a list of constraints candidates like ?1 ?2 ?3, etc.
[2] creates a list of constraints candidates, like from ?1 to ?8.
[3] creates a list of constraints indexes, like i1 i2 i3, according with the list of number you put in 'list'.
[4] creates a list of constraints indexes, like from i1 to i10.

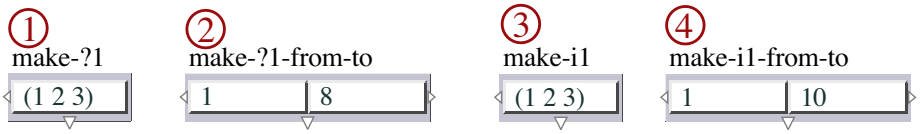

Figure 136: 3-03-make-?1-and-make-i1

4.5 04-Make-Candidates

3.04 - MK-CANDIDATES
MK-LINEAR-CANDIDATES [1] creates a sequence, for each element in 'list', going both up and down for a number of time set in 'step'.
MK-RANGE-CANDIDATES [2] creates a sequence, for each elements in 'list', going up and down for a number of time set in 'step' and with a given 'range'.

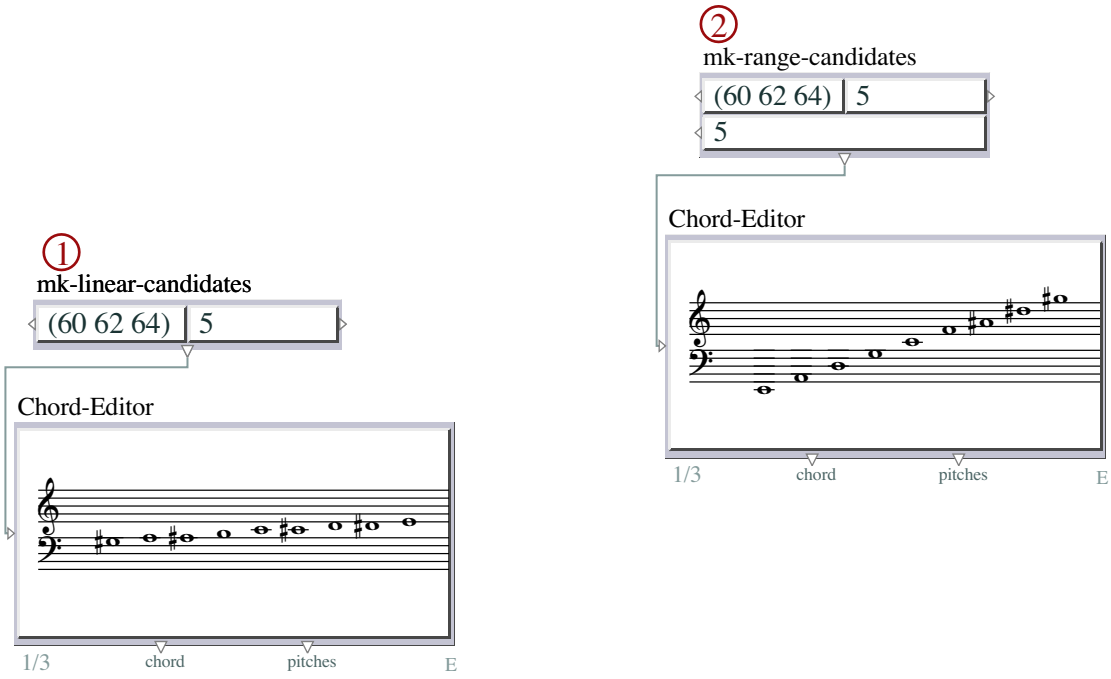

Figure 137: 3-04-make-candidates

4.6 05-Mk-Chain-Candidates

3.05 - MK-CHAIN-CANDIDATES
This function comes from Mikael Laurson's code. It creates a list of candidates that share a high level of common elements.
In 'list' you put the range of candidates, and in 'groups' you define how many sub-groups you want to create.

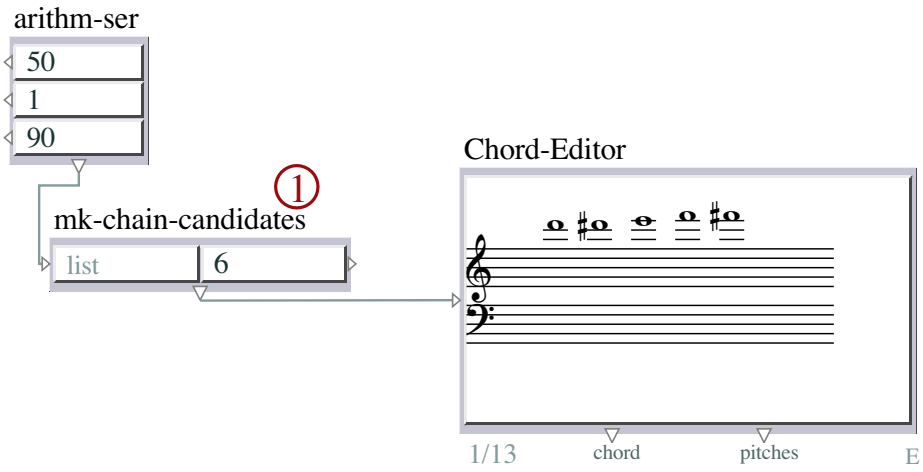

Figure 138: 3-05-mk-chain-candidates

4.7 06-Make-Pitch-Candidates

3.06 - MK-PITCH-CANDIDATES
MK-PITCH-CANDIDATES [1] creates a list of from the pitches entered in 'pitch' for a number of octaves indicated in 'octave'. If you put 1 you will obtain one octave lower and upper of the original one.
MK-PITCH-CANDIDATES-NOT-SYMMETRIC [2] creates a list of pitch candidates. In 'pitch' you put the note you want to be repeated. In 'up' and 'down' you define how many octaves you want the pitch to be transposed up and down, independantely.

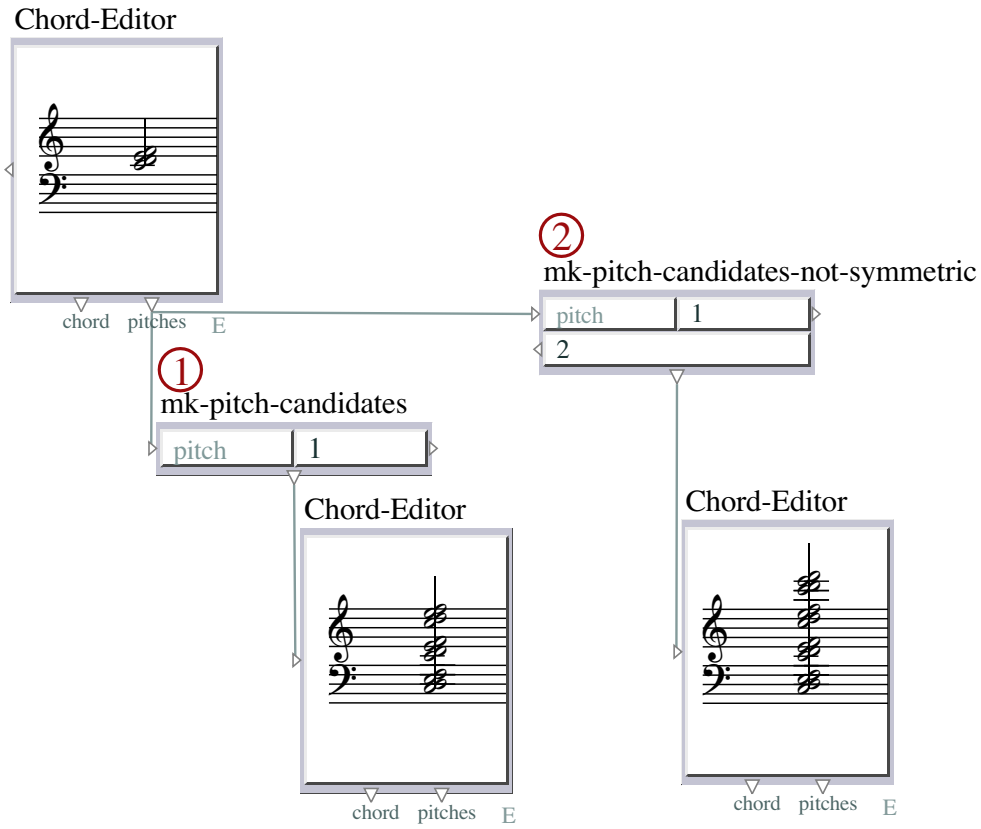

Figure 139: 3-06-make-pitch-candidates

4.8 07-Logic-or-Condition

3.07 - LOGIC-OR-CONDITION
LOGIC-OR-CONDITION [1] is a function that has to be used with some precautions, because you can easily not understand what is happening in the Multi-PMC.
This function adds the OR conditional to a series of rules. In this case I am looking for a solution that can have OR the FIND-APPLY-GLOBAL-SUM-RULE set in [2] OR the one set in [3].
Please evaluate the APPLY box [4] and you will see that the result can be randomly one of the two structures set in [2] and [3].

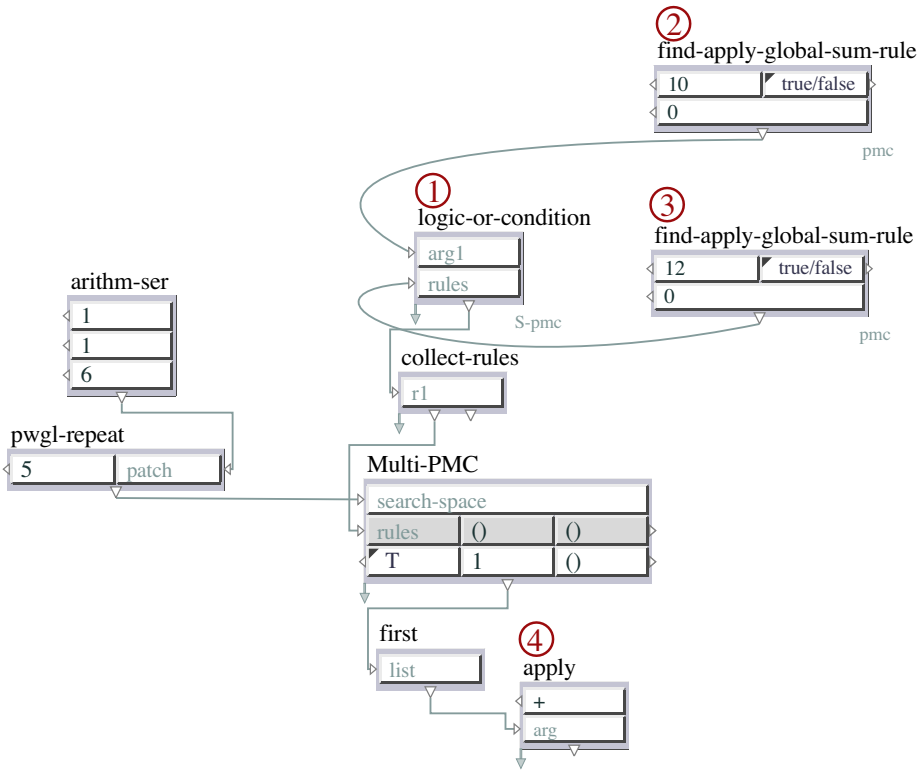

Figure 140: 3-07-logic-or-condition

4.9 08-Pitch-Extract-from-Score-Editor

3.08 - PITCH-EXTRACT-FROM-SCORE-EDITOR
PITCH-EXTRACT-FROM-SCORE-EDITOR [1] function extracts pitches from a SCORE-EDITOR in the chord format. It means that the output for a single note will be like this (60) and for a chord like this (60 62 69).
Look at the CHORD-EDITOR [2] in which all notes are entered in a flat format list.
Then look at the CHORD-EDITOR [3] in which the pitches coming out of the PITCH-EXTRACT-FROM-SCORE-EDITOR keep their structure as single notes or chords.

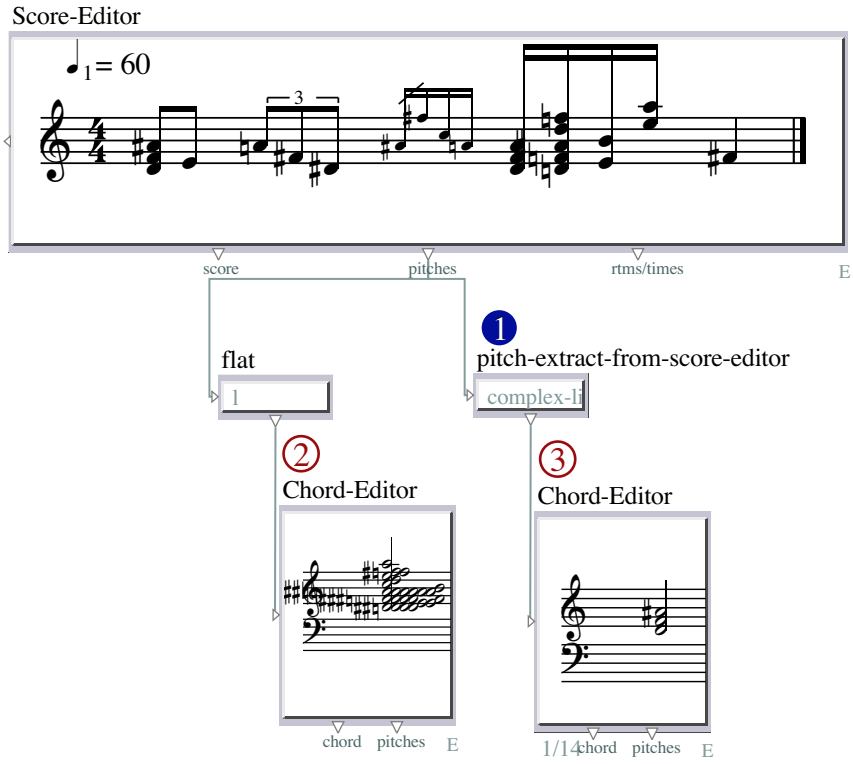

Figure 141: 3-08-pitch-extract-from-score-editor

5 0-Examples

5.1 01-Collect-Other-Rules

5.1.1 Collect-Other-Rules-01

— COLLECT-RULES-01 —
This patch gives you an example of how you can apply some rules written in the text syntax of Mikael Laurson.
The PWGL-REPEAT [1] produces a list of 10 arithmetic series in order to create the search space.
In the COLLECT-RULES [2] I just put a true/false rule and a heuristic one. The first asks for no local repetitions. The second (ALLOWED-INTERVALS) looks heuristically solutions having only the following interval list : (1 2 3 4 5 6 7).
In the TEXT-BOX [3] I wrote a rule using Mikael Laurson text syntax defining that if an interval between two notes is equal to a minor second, the following two intervals have to be two octaves (equal to 12). This rule is connected through an X-APPEND [4] with the left output of the COLLECT-RULES. That means that the previous true/false rule

(NO-LOCAL-REPETITION-RULE) will be added to the TEXT-BOX rule within the MULTI-PMC. On the contrary the heuristic rule still works as much as possible, accordingly to the true/false rules.
Please evaluate the CHORD-EDITOR [5] and see how the rules are applied.

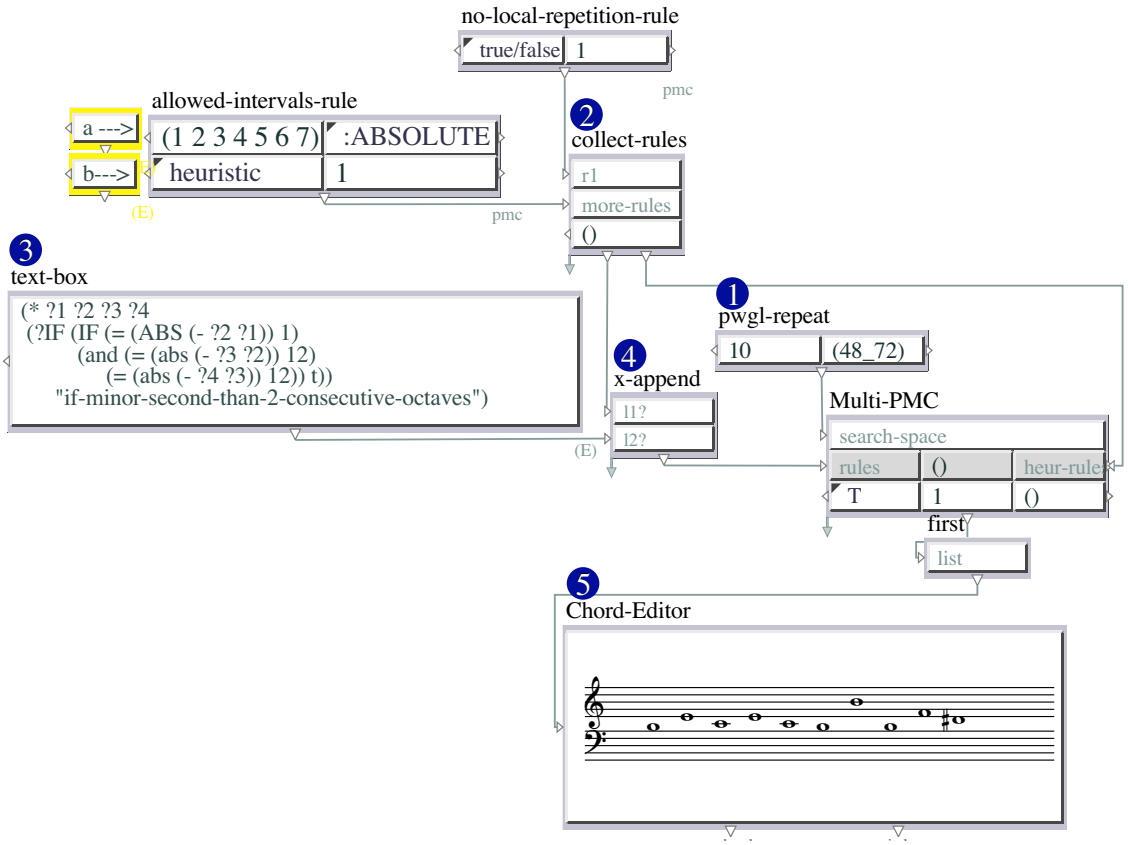

Figure 142: Collect-other-rules-01

5.1.2 Collect-Other-Rules-02

— COLLECT-RULES-02 —
This patch gives you another example of how you can apply some rules written in the text syntax of Mikael Laurson.
In this case I used the Multi-Score-PMC [5] to solve a polyphonic problem.
In the Score-Editor [1] I set a given score.
In the COLLECT-RULES [2] I just put one true/false rule and two heuristic ones. The true/false S-PMC-NO-CROSSING-VOICE) does not allow : - the first voice to have lower notes than the second voice, - the second voice to have lower notes than the third voice, - the third voice to have lower notes than the fourth, and so on...
Then there are two heuristic rules : the first (S-PMC-ALLOWED-HARMONY-IN-GIVEN-MEASURES-RULE) obliges the solution to follow the harmony of C-major, but only in

the first measure. The second heuristic rule (S-PMC-INTERVAL-SMALLER-RULE) asks for a solution including only intervals smaller than the value defined in the 'interval' input (in this case an augmented fourth).
In the TEXT-BOX [3] I wrote a rule using Mikael Laurson text syntax. This rule concerns only the second part. If a note is a C the following note has to be a C-sharp. This rule is connected through an X-APPEND [4] with the left output of the COLLECT-RULES. That means that the previous true/false rule will be added to the TEXT-BOX rule within the MULTI-PMC. On the contrary the heuristic rules still work as much as possible, accordingly to the true/false rules.
Please evaluate the Multi-Score-PMC [5] and see how the rules are applied.

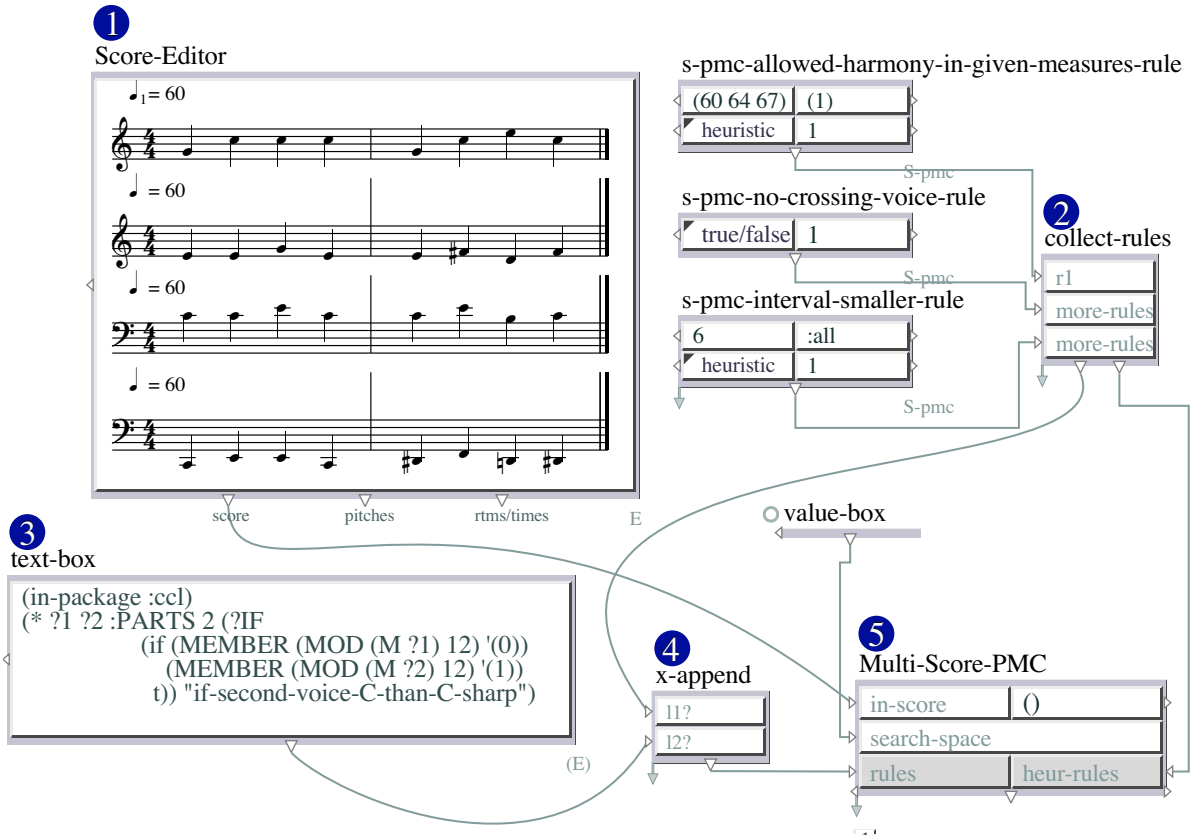

Figure 143: Collect-other-rules-02

5.1.3 Collect-Other-Rules-03

— COLLECT-RULES-03 —
This patch gives you another example of how you can apply some rules written in the text syntax of Mikael Luarson.
In this case I used the ENP-SCRIPT [5] to solve a score treatment.
In [1] I wrote a score.

In the COLLECT-SCRIPT-RULES [2] I define one rule for the fourth voice (CREATE-EXPRESSION-ON-MAIN-BEAT) that puts an accent on each main beat for each measure. I define also that for the first part I want to put an, accent on each not main beat of each measure (CREATE-EXPRESSION-NOT-ON-MAIN-BEAT).
In the TEXT-BOX [3] I defined four rules written in Laurson syntax. Here is the detail: 1. When it finds an interval bigger than a minor third, I ask for a slur; 2. When it finds four different pitches (in any modulo 12) I ask for a fermata; 3. When there are three consecutive descending notes, I ask for a diminuendo; 4. When there are three consecutive descending notes and the first interval is smaller than a minor third, then I ask for a group.
Please choose with the RULE-FILTER [a] which rule you want to apply. You can chose all of them if you want. Then select the ENP-SCRIPT [5] : first type R on the keyboard in order to reset the Score-Editor, then type V on the keyboard and look at the result. If you change the selection of rules inside the RULE-FILTER [a], to better see the result, please select again the ENP-SCRIPT [5] : first type R to reset and then V and look at the result.

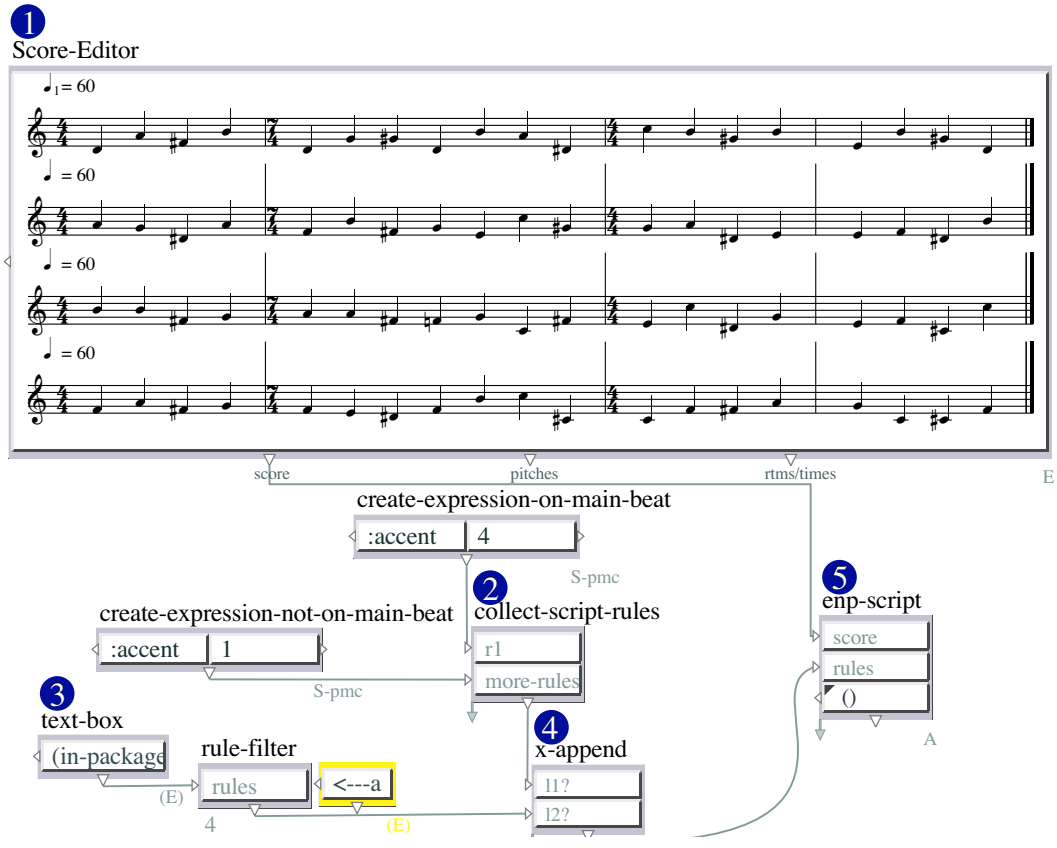

Figure 144: Collect-other-rules-03

5.2 02-Contrepoint

5.2.1 Counterpoint

This series of patches is conceived as part of the tutorial for the Multi-Score-PMC.
This exercise (which is not at all a goal in itself) tries to reproduce the Palestrina counterpoint of first specie. That means a four voices counterpoint in which all the notes have the same face value (1/4).
The conception of these patches is absolutely consistent. Go from the first to the last in the order. What is explained in one patch is not longer explained in the next ones.

5.2.2 Counterpoint-01

— COUNTERPOINT-01 —
In the SCORE-EDITOR [a] I entered four voices with random notes.
The search-space is in the VALUE-BOX [b] using the expand list code. Look at the CHORD-EDITOR [c] how the candidates are distributed in order to generate only useful notes.
The first (soprano voice) is the highest, the contralto voice is below till the bass voice (the fourth), that is the lowest.
In the abstraction 'Rules' [d], I will put the rules. For the moment it is empty.
Evaluate the Multi-Score-EDITOR [e] in order to see how it works with the random mode set on T.

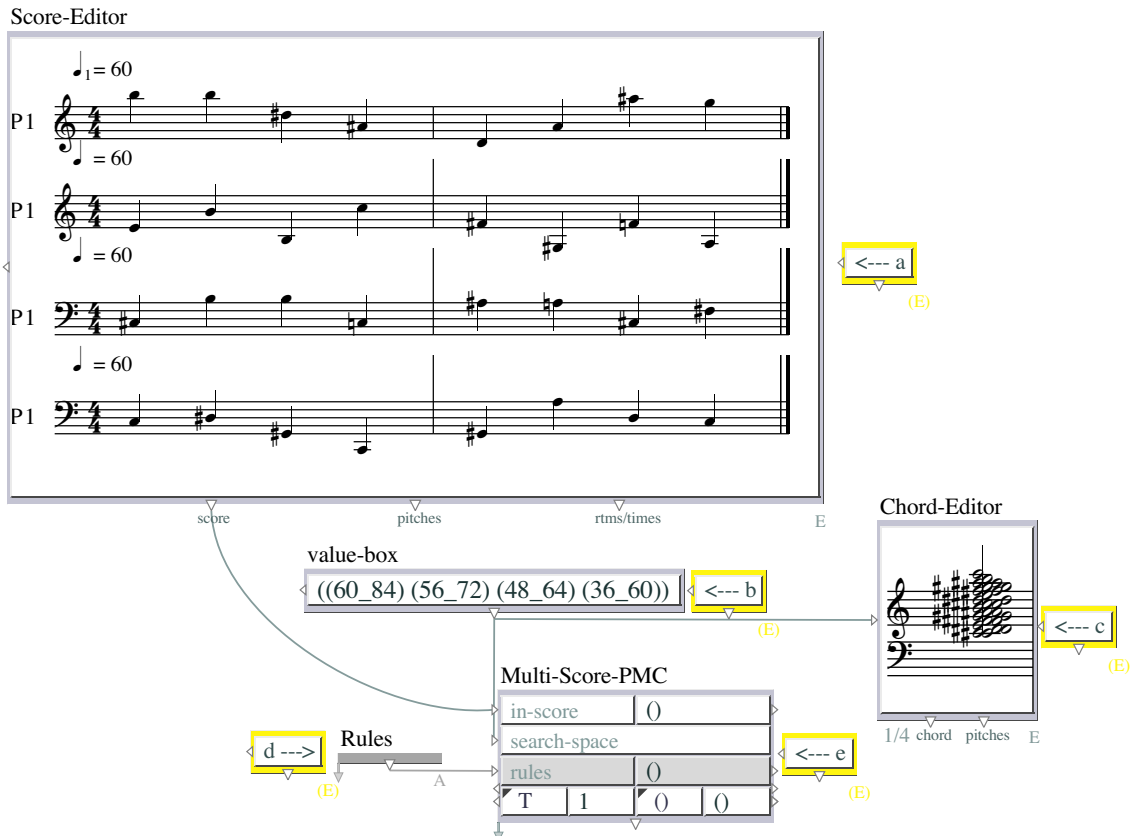

Figure 145: Counterpoint-01

5.2.3 Counterpoint-02

— COUNTERPOINT-02 —
This is the next step. In the abstraction 'Count-notes' [a], I calculate how long is a sequence of notes for each voice.
Open the abstraction 'Rules' [b].
S-PMC-INDEX-RULE [1] and [2] obliges the first note (index 1) of the soprano voice [c] to be equal to C (midi note 60)
In [2] the C (midi note 72) will be at the last place, defined by the abstraction 'Count-notes' result coming in the [d] input.

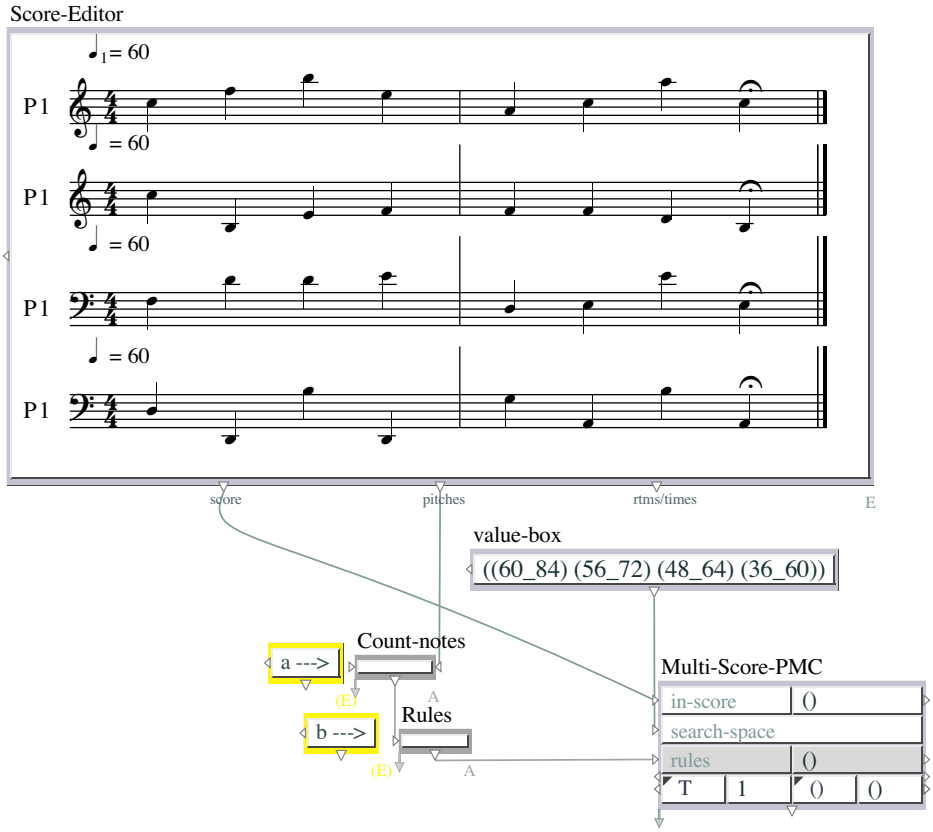

Figure 146: Counterpoint-02

5.2.4 Counterpoint-03

— COUNTERPOINT-03 —
As I told you in the introduction, these patch are put in a consistent order, so I will describe only the integrations and the differences from the previous patches.
Please open the 'Candenza-rules' abstraction [a].
In addition of what we have seen before in [1], I obliged the bass voice (set in [b]) to finish on a C in the last measure. At the same time, the rule [2] allows only, on the fermata, the notes of a C-Major chord.

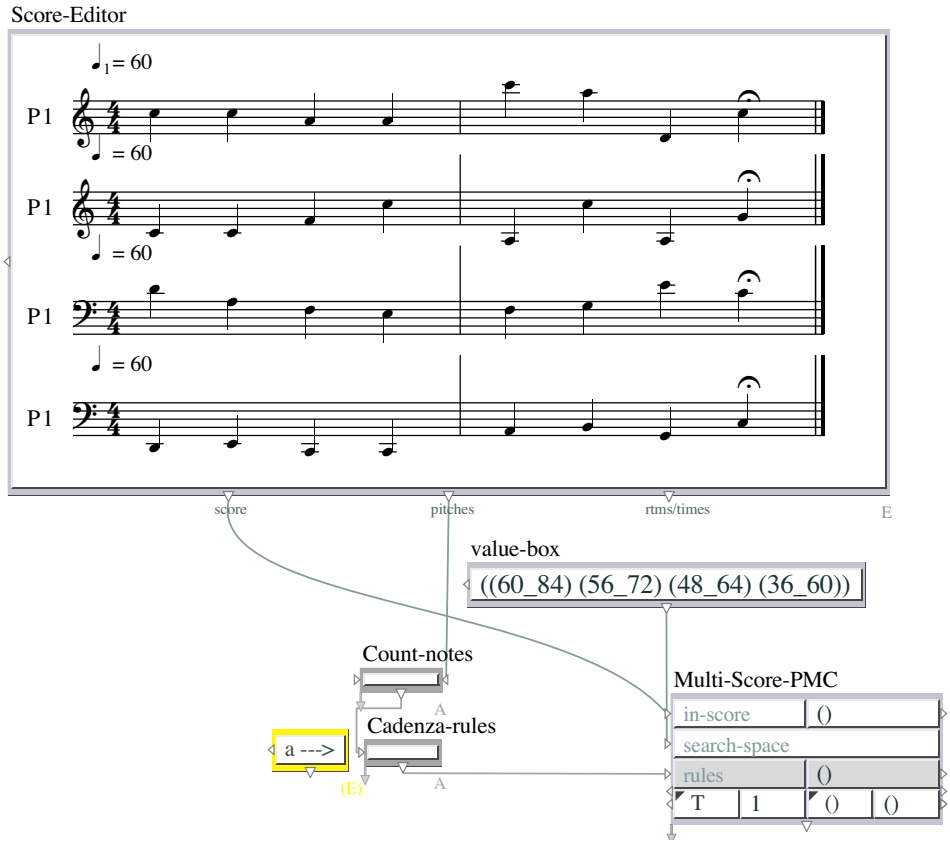

Figure 147: Counterpoint-03

5.2.5 Counterpoint-04

— COUNTERPOINT-04 —
Open the 'Interval-rules' abstraction [a].
Here you can easily understand the rules I apply. In [1] I ask for intervals smaller than a major sixth. In [2] I oblige the bass voice [b] not to have local repetitions. In [3] I ask for a solution where, for every voices, I don't want more than 5 notes going in the same descending direction. In [4] I ask the same thing but for ascending notes.

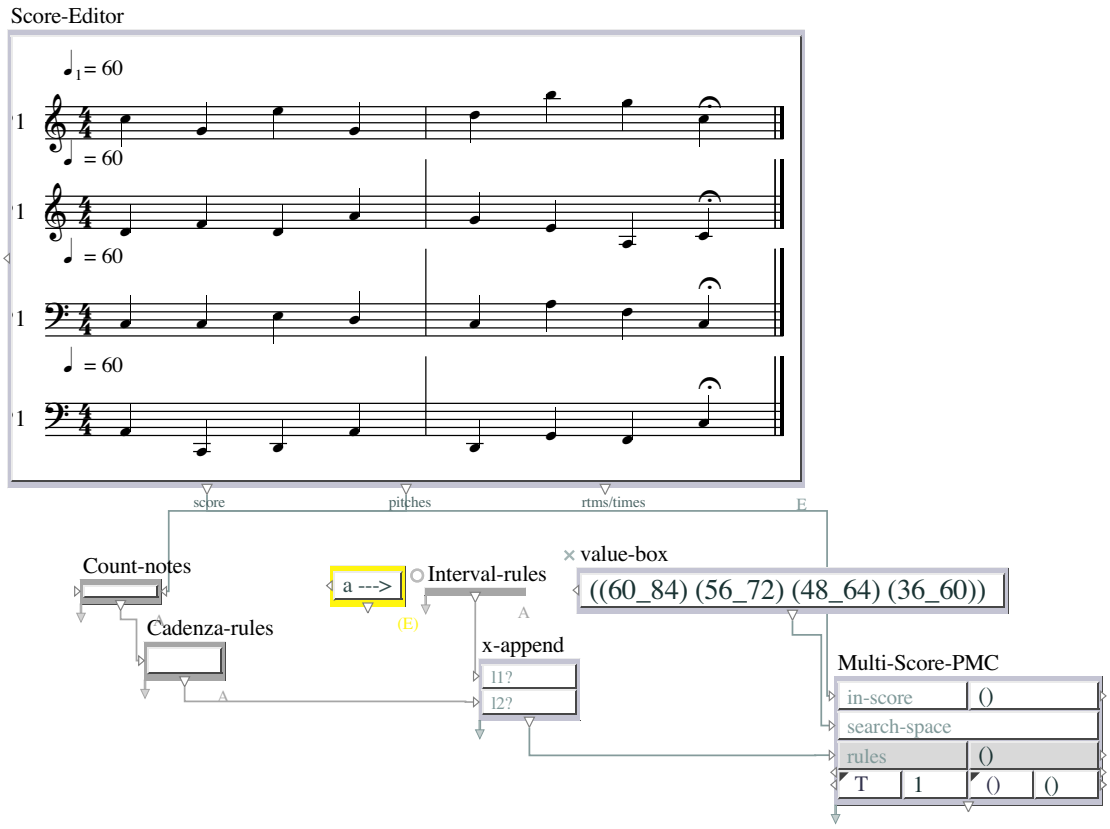

Figure 148: Counterpoint-04

5.2.6 Counterpoint-05

— COUNTERPOINT-05 —
Open the 'Interval-rules' [a] abstraction.
In [1] I ask for a solution that never reaches in 3 notes [b] an interval of major seventh [c]. In [2] I apply the same rule but forbidding to reach, in 4 notes [d] an interval of octave [e]. In [3] I use the JUMP-RESOLUTION RULE : when an interval of the solution is bigger than an augmented fourth [f], the next interval has to be in the opposite direction, and must be to be smaller than a minor third [g]. In [4] I forbid local repetitions for all voices, for this reason the previous rule [5] is now inactive.

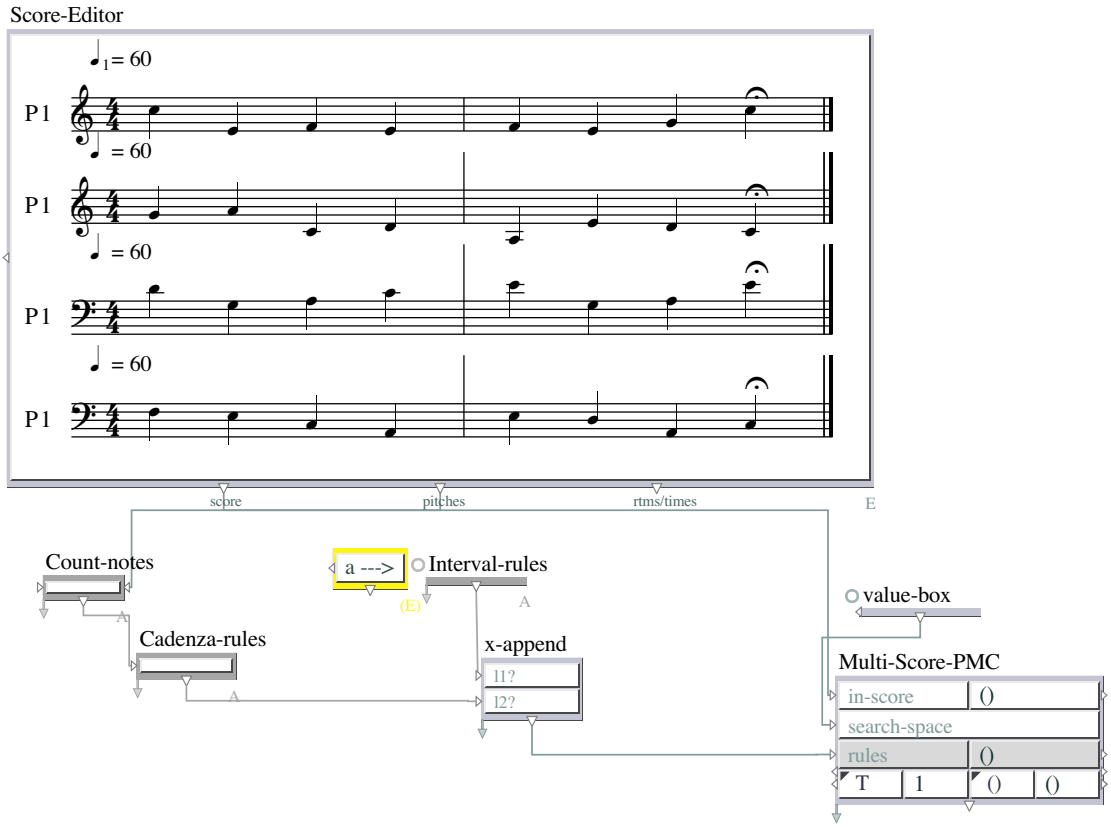

Figure 149: Counterpoint-05

5.2.7 Counterpoint-06

— COUNTERPOINT-06 —
Open the 'Harmonic-rules' [a] abstraction.
In [1] I forbid the unison [b] as a possible interval between two voices. In [2] I ask for a solution where, for each chord, I have at least three [c] different pitches. In [3] I define that I prefer to duplicate octaves [d]. In [4] I give the Multi-Score-PMC all forbidden inversions set in [e].

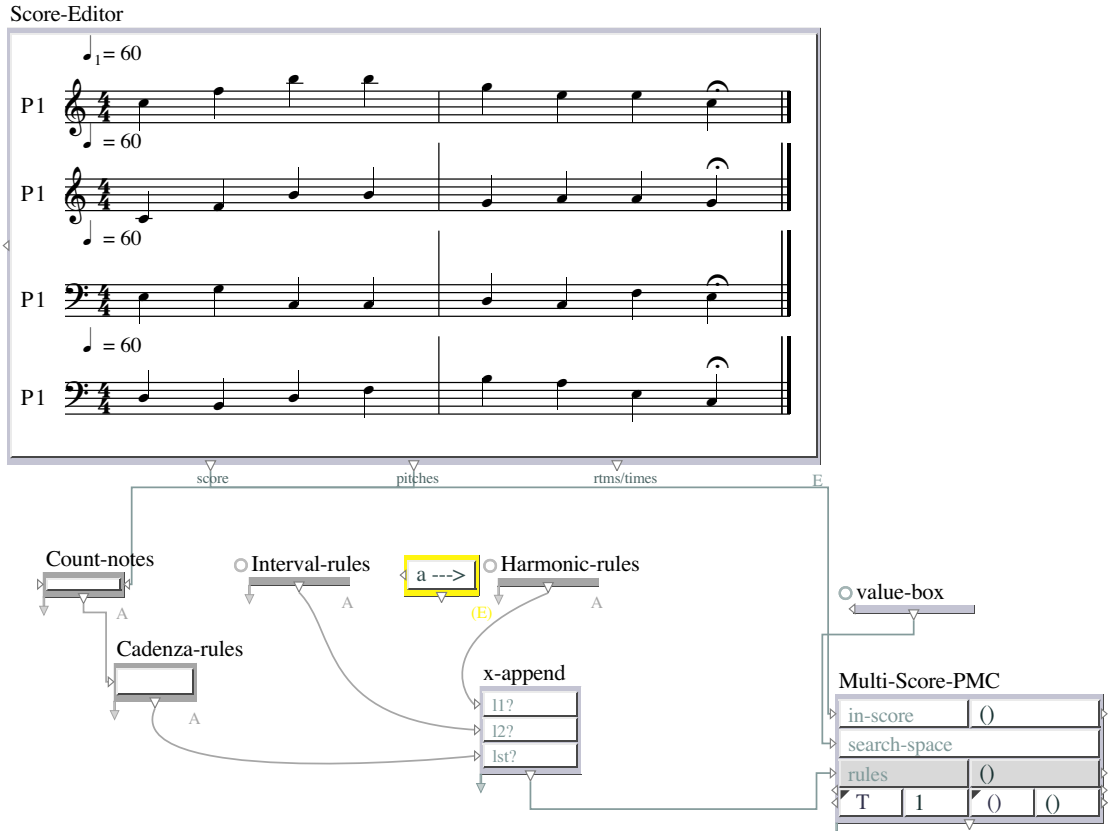

Figure 150: Counterpoint-06

5.2.8 Counterpoint-07

— COUNTERPOINT-07 —
Open the 'Harmonic-II-rules' [a] abstraction.
In [1] I define the allowed chord successions with input [b].
Open the 'Chord-successions' [b] abstraction. Inside you'll see how I define the successions. In [d] I give a chord and in [e] I put all possible following chords.
In [2] I give all the allowed chords set in the 'Allowed-chords' [c] abstraction. Please open it. In [f] I entered all the chord I do accept. Please if you do not understand this patch, see the patch 2.02.09 of the same tutorial.

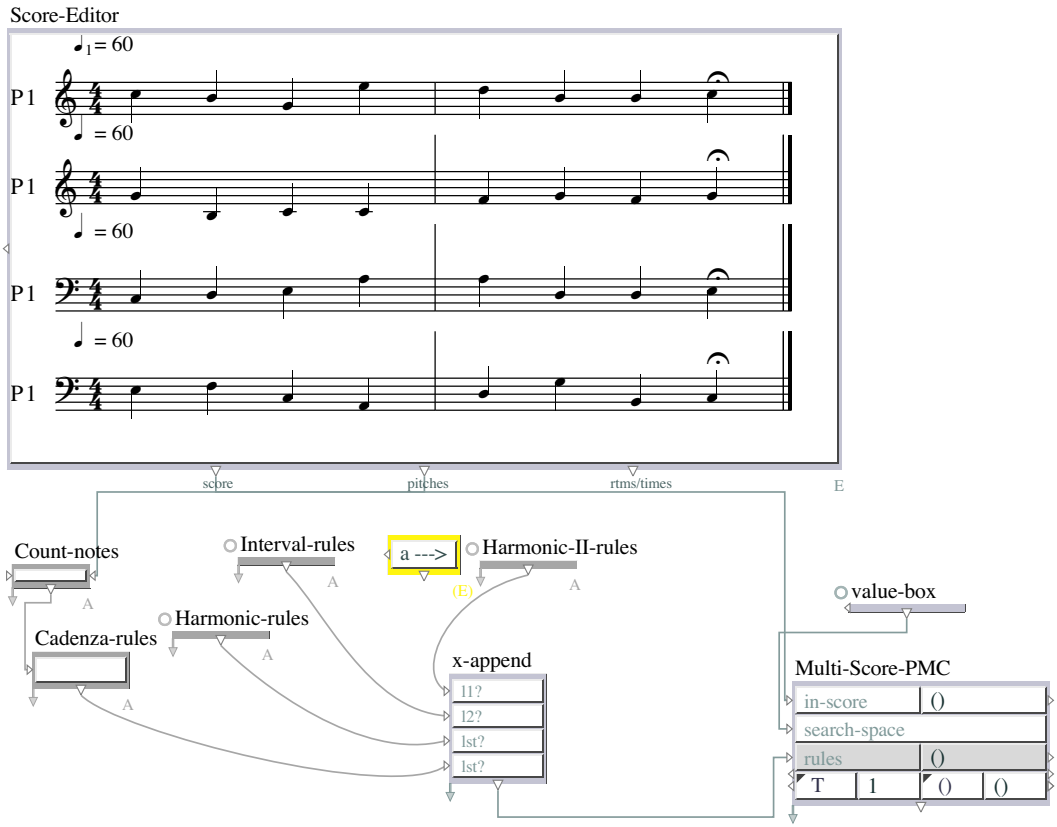

Figure 151: Counterpoint-07

5.2.9 Counterpoint-08

— COUNTERPOINT-08 —
Open the 'Voice-leading' [a] abstraction.
In [1] I ask not to have open parallel octaves and fifths [b]. In [2] I do not accept a succession from an augmented fourth [c] to a perfect fifth [d]. In [3] I forbid hidden parallel octaves and fifths [e].
Some little changes came since the last patch. Please open the abstraction 'Harmonic-rules' [f] and see that I added the rule NO-CROSSING-VOICE [4], to avoid any intersections between pitches of different parts.
Now open the 'Interval-rules' abstraction [g]. Here I added TONE-RESOLUTION- RULE [5], in which I specify that the sensible note (B in C-major mode [h]) has to be followed by a C note [i].
Why do I add these rules now? Because I forgot to do it before. :-)

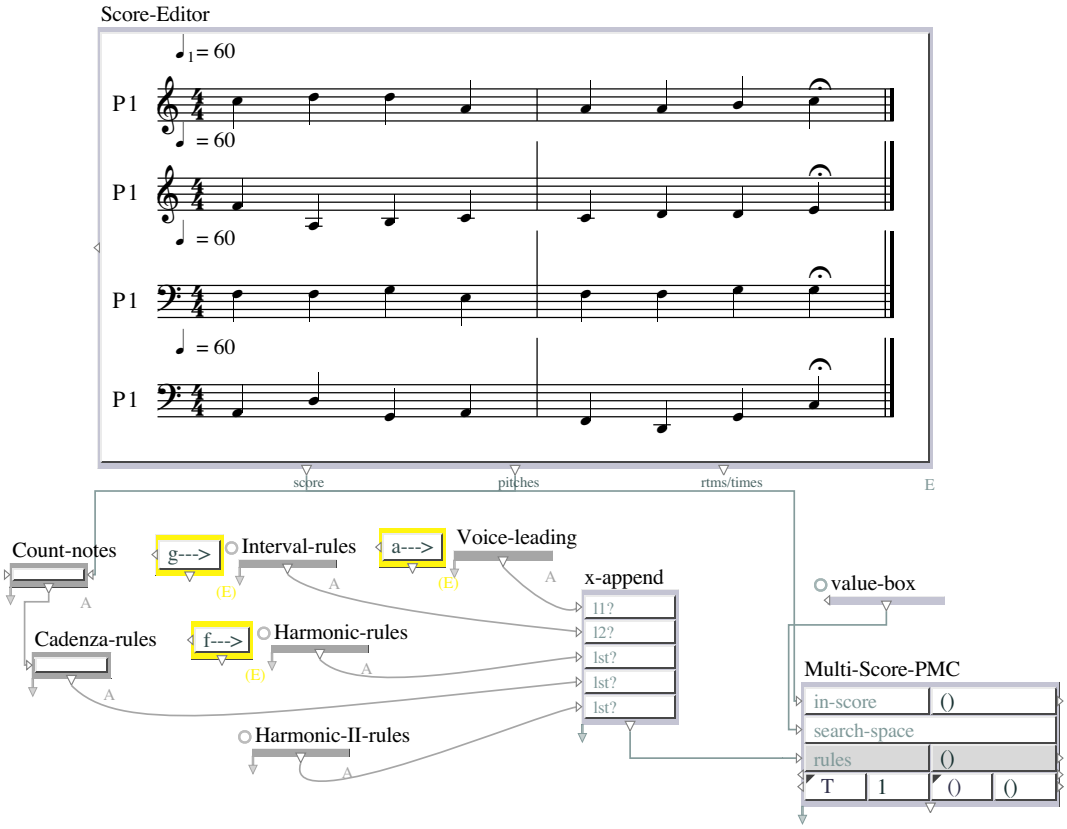

Figure 152: Counterpoint-08

5.2.10 Counterpoint-09

— COUNTERPOINT-09 —
Open the 'Given-voice' [a] abstraction.
Here you see that I give the bass voice. When you evaluate the Multi-Score-PMC, you'll see that the bass voice does not change, and the other voices follow all the rules.

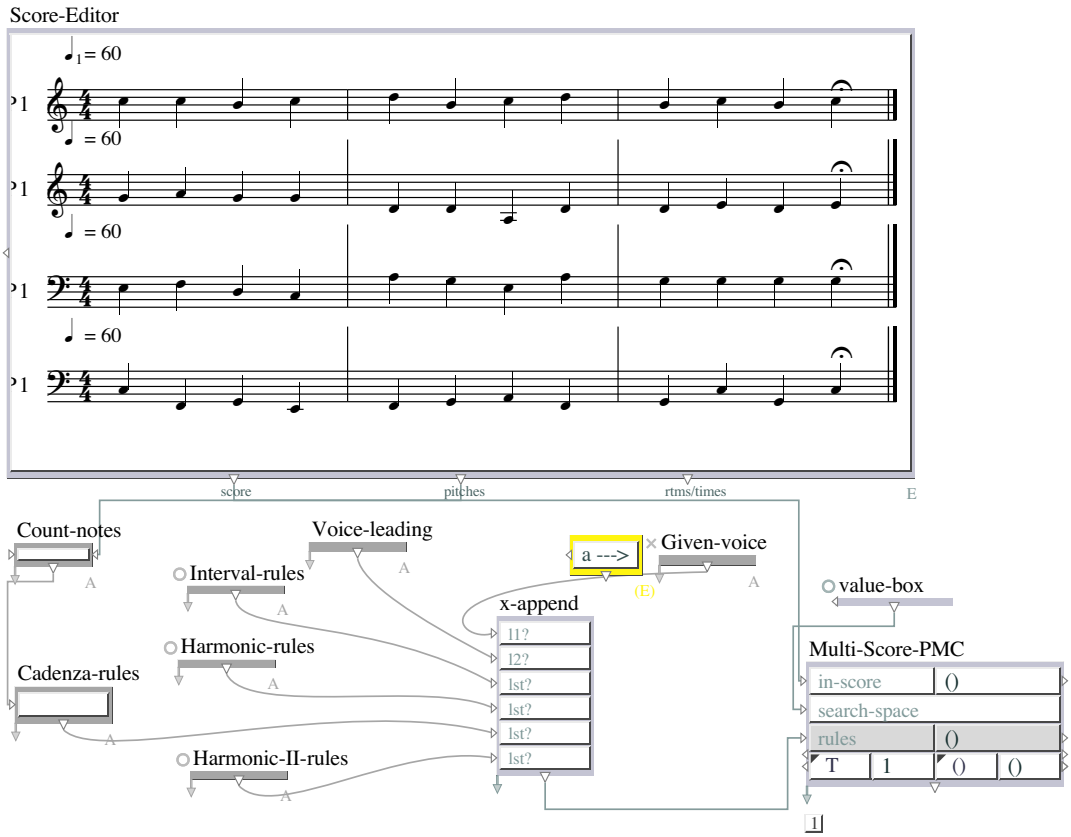

Figure 153: Counterpoint-09

5.2.11 Counterpoint-10

— COUNTERPOINT-10 —
Just a very last part. As you see I added some arbitrary slur in the SCORE-EDITOR. Open the 'Interval-rules' abstraction [a]. Now I define that when there is a slur [1], the intervals have to be smaller or equal than a perfect fourth.

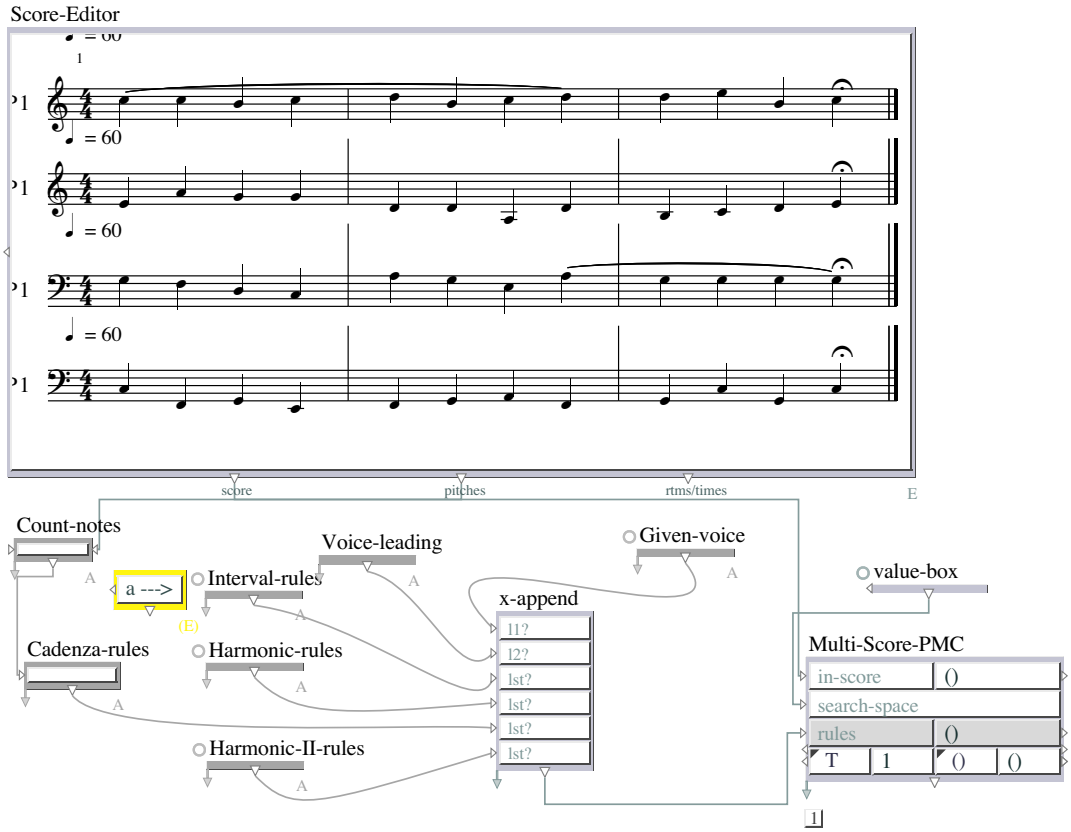

Figure 154: Counterpoint-10

5.3 03-Special-Combinations

5.3.1 Always-3-Given-Elements-01

— ALWAYS-3-GIVEN-ELEMENTS-01 —
The aim of this little series of patches is to show you how to use several Multi-PMCs in order to generate solutions always including some given values.
This first patch is just a reminder of what you should have seen in the Multi-PMC rules tutorials.
The three INDEX-RULES [1] force the solution to have, in a given place [a], the value defined in [b]. So in this case the values 2, 7 and 5 will always be found in the solution in the first, third, and fifth places.

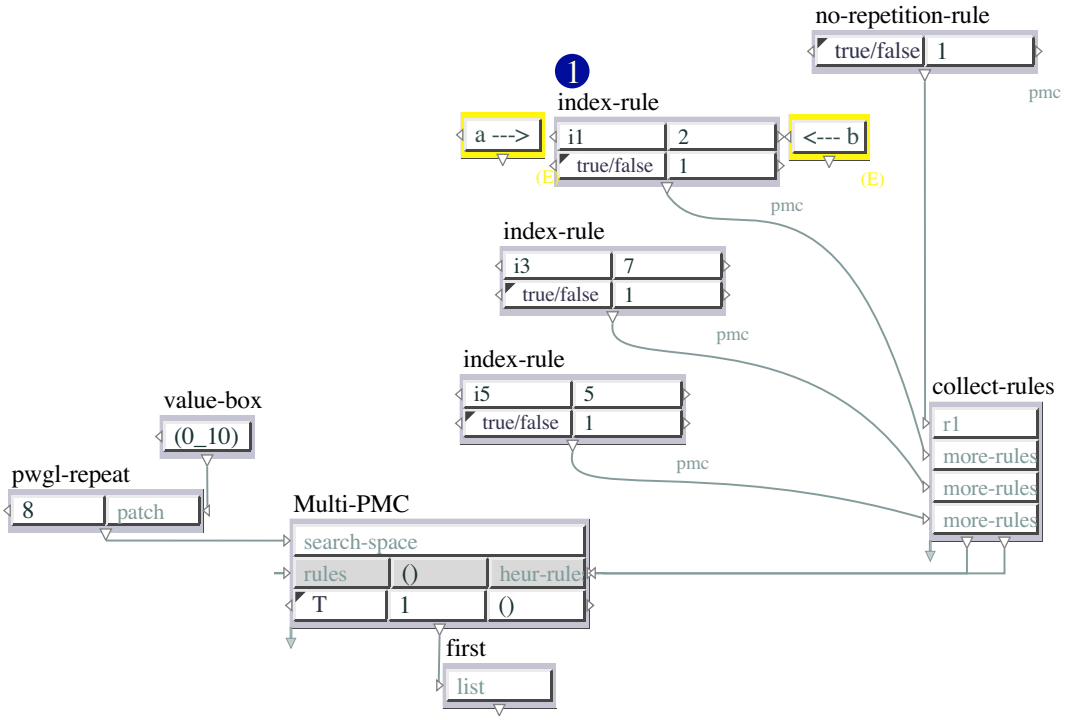

Figure 155: Always-3-given-elements-01

5.3.2 Always-3-Given-Elements-02

— ALWAYS-3-GIVEN-ELEMENTS-02 —
This second patch is little bit tougher. Its goal is to generate a solution in which a set of values, here (1 2 3) [a] will always be found in this order but with possible other values between them.
In this patch we use two Multi-PMCs in order to create such a solution.
The first one [1] generates a solution from candidates belonging to an arithmetic series with a length identical to the list we want to create [b] The length of the result, which has no repetition allowed [c], comes itself from the length of the set of values we want in the final solution [d].
This result is sorted, here in ascending order [e], and then goes to the first input of the ”’create-i?’ abstraction [2]. Please look inside.
Here the PWGL-MAP creates as much INDEX-RULES as there is values, coming from the second input [f], in the desired set [a]. The MAKE-I1 function [g] creates random but ordered indexes for the rules. Note that a COLLECT-RULES is needed here in order to prevent parentheses problems further in the patch.

The X-APPEND [h] recovers the generated rules and adds another NO-REPETITION-RULE [i], in order to avoid duplicates of the values we are interested in.
Please evaluate the second Multi-PMC [3] and see that the values 1, 2 and 3 can always be found somewhere in the solution, each time in the same ascending order.

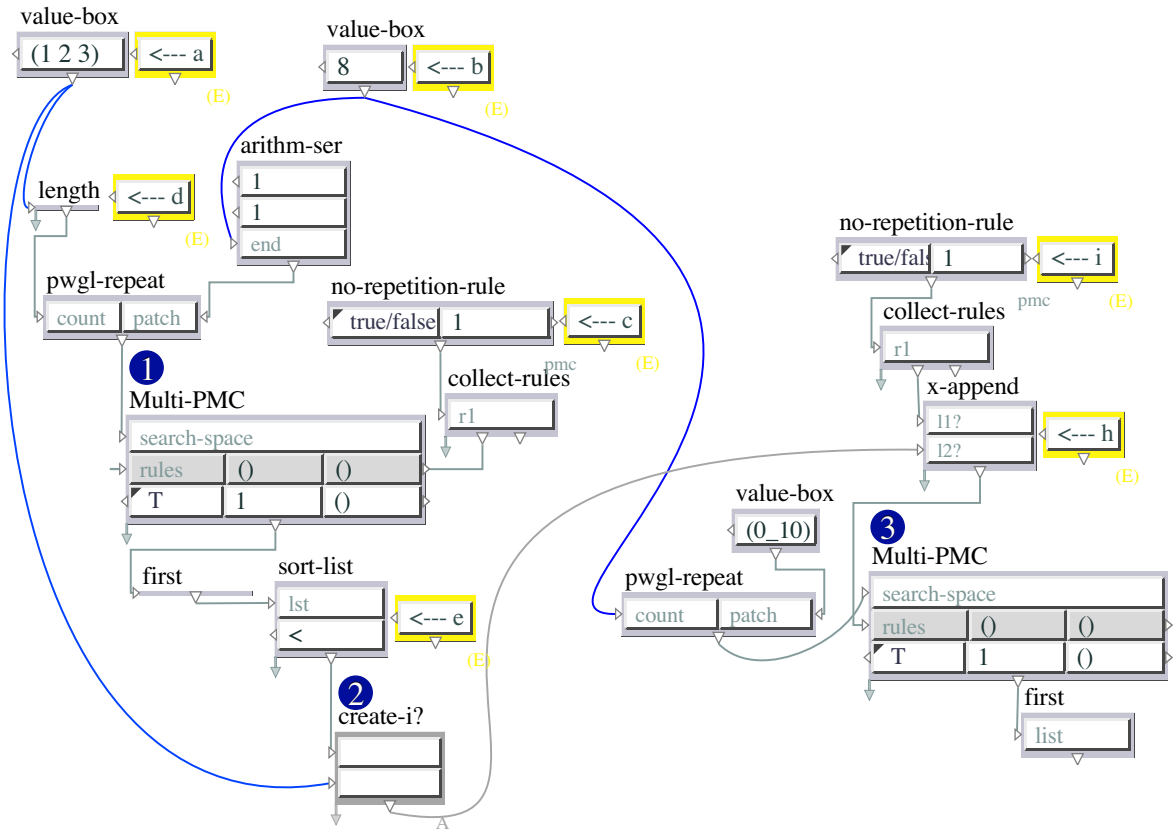

Figure 156: Always-3-given-elements-02

5.3.3 Always-3-Given-Elements-03

— ALWAYS-3-GIVEN-ELEMENTS-03 —
This third patch aims to create a solution in which a given set of values, here (1 2 3), can always be found in this form. In other words, the generated list will always include 1, 2 and 3 in this order, and without any values between them.
Like in the previous tutorial, the left part of this patch prepares a series of indexes that will be used to generate several INDEX-RULES in the 'create-i?' abstraction [1].
The first thing done here is generating, with a NTH-RANDOM function [d], a random value selected from an arithmetic series. This function takes its 'begin' and 'end' values from both length of the desired set [a] and the final solution [b]. Please look further in order to understand this little trick.
The same random value is then sent, thanks to the CONST-VALUE, to both a G- [e] and

the MAKE-I1-FROM-TO function [f]. This one uses the random value as the last value of a list of [c] indexes.
The G- calculates the first value of the same list from the random value itself and the 1- function [g] that prevents the first index of the list to be lower than i1.
The 'create-i?' abstraction works pretty much like in the previous patch. The loop generates a given number of INDEX-RULES according to the list of indexes and the desired set of values.
The X-APPEND [h] recovers the generated INDEX-RULES and also a NO-REPETITION-RULE, in order to avoid duplicates of the values we are interested with.
Please evaluate the second Multi-PMC [2] and see that the values 1, 2 and 3 can always be found somewhere in the solution, each time stuck together and in the same order.

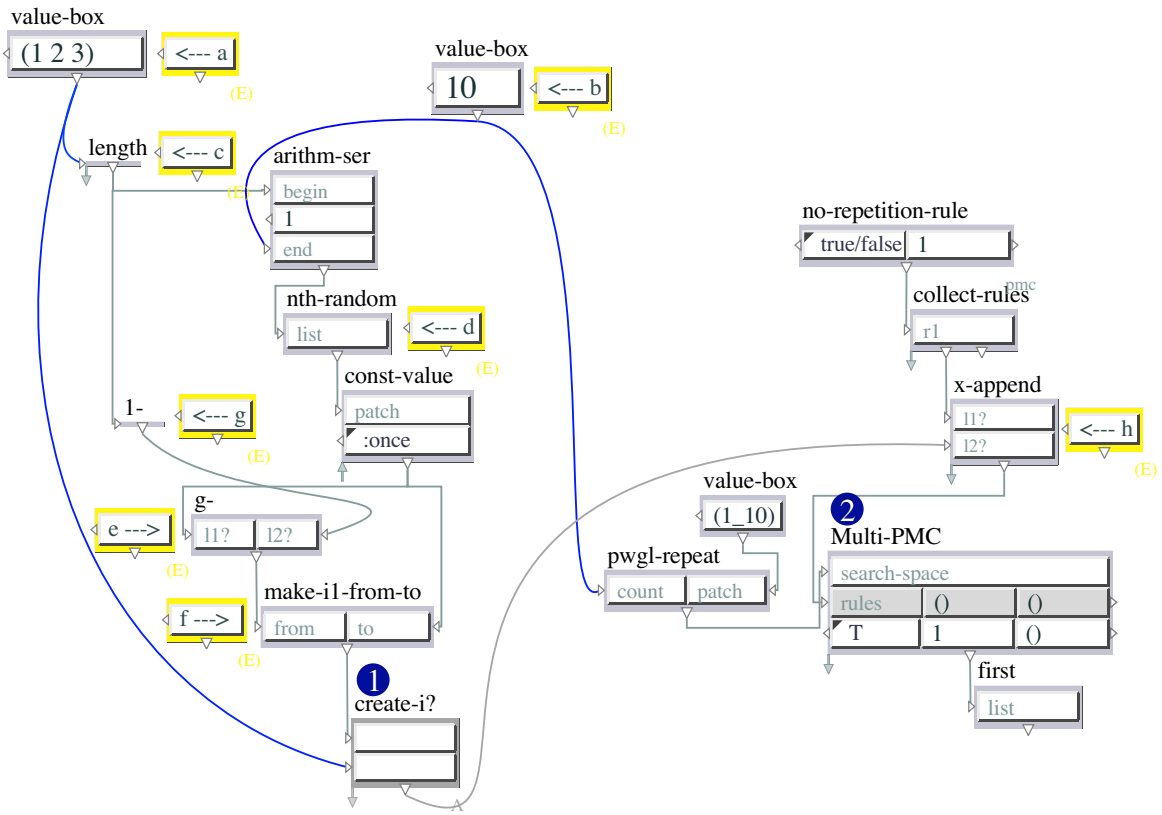

Figure 157: Always-3-given-elements-03

A Box Reference

allowed-chain-rule

arglist: `(element following mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges a given element, to be followed by those elements eneterd in following. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

allowed-distant-intervals-rule

arglist: `(distance intervals absolute? mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule allows a sequence of intervals to be equals with a given distance. The allowed intervals are put in INTERVALS. The distance has to be described giving the first and the last note of the distance. For instance, if in distance you put 1 and 3 it means that in a sequence each the interval between the first and the third note has to be a member of INTERVALS. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

allowed-intervals-rule

arglist: `(intervals absolute? mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule allows only the intervals indicated in 'intervals'. If the menu 'absolute?' is 'absolute, that means that intervals are intented in absolute mode. If this menu is 'up/down', that means that the intervals are divided into ascending and descending. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

allowed-pitch-class-rule

arglist: `(pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
Only the class (for instance minor triad) indicated in 'pitch' will be allowed in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule does not work because of SLEN...

allowed-pitch-class-sub-list-rule

arglist: `(pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
This function outputs a solution having only the class (for instance minor triad) indicated in 'pitch' will be allowed in any octave including also other notes. That means that is I'm looking for a minor triad in a 5 notes chord, the solution will look if a minor triad exists inside the 5 notes chord. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule does not work because of SLEN...

allowed-pitch-rule

arglist: `(pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
Only the pitches indicated in 'pitch' will be allowed in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

allowed-pitch-structure-rule

arglist: `(pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
Only the pitches indicated in 'pitch' will be allowed in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule does not work because of SLEN...

allowed-polarized-pitch-rule

arglist: `(pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
Only the pitches indicated in 'pitch' will be allowed in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

alternating-+/-first-elmt-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
This rule ceates a result of sub lists in which the fisrt element is, in an alterning way, before positive then negative an so on. ATTENTION : this rule works with list of lists.

ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

alternating-+/-last-elmt-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
This rule ceates a result of sub lists in which the last element is, in an alterning way, before positive then negative an so on. ATTENTION : this rule works with list of lists. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

alternating-positive-negative-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
This rule obliges positive number to be alternated with negative numbers. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

always-more-little-included-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Rules▶Pattern-Rules
When one element (as list of lists) is bigger than a second so the more little has to be included in the bigger. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

any-note-repeated-rule

arglist: `(times which? mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
Any notes has to be repeated (in modulo too) a less, exactly or more times as indicated in 'times'. N.B. BE CAREFULL: the menu which? defines less, equal or more. If $<$, the calculation is quite fast. If $=$, be sure to have a 'pari' number of candidates in the esarch space. If $>$, the calculation can be very slow. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

apply-interval-global-sum-rule

arglist: `(sum mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule outputs a solution having the the sum of all intervals equal to the value put in sum. First it makes the x->dx of all intervals and then it applies '+ to all. ATTENTION : in the mode true/false, does NOT work: because of SLEN

apply-interval-sum-rule

arglist: `(sum mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule outputs a solution having the the sum of all intervals equal to the value put in sum. First it makes the x->dx of all intervals and then it applies '+ to all. ATTENTION : in the mode true/false, does NOT work: because of SLEN

ascending-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule obliges all the value to be ascending. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

ascending-sub-group-no-repet-rule

arglist: `(nth-? mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule obliges the nth (put in nth-?) values of a list of lists to be ascending without any repetition. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

ascending-sub-group-with-repet-rule

arglist: `(nth-? mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule obliges the nth (put in nth-?) values of a list of lists to be ascending with repetitions. ATTENTION : The Heuristic implementation can not work.

ascending-without-repetition-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule obliges all the value to be ascending without any repetition. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

chain-common-element-lists-rule

arglist: `(chain-length? mode? weight)`
package: JBS-CONSTRAINTS
menu: Structure-Rules▶Matrix-Rules
This rule chains lists of lists following the criteria of adaptating figure: see CMI thoery. You have two lists of lists: in chain-lenngth?, the length of the last part of the corrent list (and automatically the length of the first part of the forword one. ATTENTION : IN THE HEURISTIC MODE IT DOES NOT WORK. WHY???

chain-more-little-included-common-lists-rule

arglist: `(first-n? last-n? mode? weight)`
package: JBS-CONSTRAINTS
menu: Structure-Rules▶Matrix-Rules
This rule chains lists of lists following the criteria of adaptating figure: see CMI thoery. You have two lists of lists: in chain-lenngth?, the length of the last part of the corrent list (and automatically the length of the first part of the forword one. ATTENTION : IN THE HEURISTIC MODE IT DOES NOT WORK. WHY???

collect-rules

arglist: `(r1 &rest more-rules)`
package: JBS-CONSTRAINTS
menu: Utilities
Collect the rules and seperate true/false from heuristic rules. Use the two outputs in order to connect to the pmc-engine

collect-script-rules

arglist: `(r1 &rest more-rules)`
package: JBS-CONSTRAINTS
menu: Utilities
Collect the rules for the enp-script. Use the two outputs in order to connect to the pmc-engine

count-any-element-rule

arglist: `(how-many mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obligesa solution to have any element repeated many times as indicated in 'how-many. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

count-common-elements-rule

arglist: `(how-many mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges sub lists to have the number of common elements put in 'how-many. ATTENTION : This rule works with a list of lists. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

count-negative-intervals-rule

arglist: `(number mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
The solution must have a number of negative intrevals as indicatd in number. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule DOES NOT WORK because of Slen...

count-negative-rule

arglist: `(number mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
The solution must have a number of negative valure as indicatd in number. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

count-positive-intervals-rule

arglist: `(number mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
The solution must have a number of positive intrevals as indicatd in number. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule DOES NOT WORK because of Slen...

count-positive-rule

arglist: `(number mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
The solution must have a number of positive valure as indicatd in number. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

count-this-element-rule

arglist: `(element how-many mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obligesa solution to have any element repeated many times as indicated in 'how-many. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

count-this-modulo-rule

arglist: `(note how-many mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
This rule obliges a solution to have a given note repeated many times as indicated in 'how-many, in any possible octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

count-this-note-rule

arglist: `(note how-many mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
This rule obliges a solution to have the given note repeated many times as indicated in 'how-many inthe exact octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

create-expression-for-beats

arglist: `(beats-numbers expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
This function adds expression not on main beat for the beats set in beats-numbers in any measure. In expression you pt the kind of expression you want. In parts if you put :all, the rule will be applied on all parts. If you put 1 only in the first, second only in the second...

create-expression-for-measures

arglist: `(measure-numbers expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
This function adds expression for the entire measures set in measure-numbers. In expression you pt the kind of expression you want. In parts if you put :all, the rule will be applied on all parts. If you put 1 only in the first, second only in the second...

create-expression-not-on-main-beat

arglist: `(expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
This function adds expression not on main beat. In expression you pt the kind of expression you want. In parts if you put :all, the rule will be applied on all parts. If you put 1 only in the first, second only in the second...

create-expression-on-chord-sequence

arglist: `(expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
This function adds expression on consecutive chords. In expression you pt the kind of expression you want. In parts if you put :all, the rule will be applied on all parts. If you put 1 only in the first, second only in the second...

create-expression-on-grace-note-sequence

arglist: `(expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
This function adds expression on consecutive grave-note sequence. In expression you pt the kind of expression you want. In parts if you put :all, the rule will be applied on all parts. If you put 1 only in the first, second only in the second...

create-expression-on-main-beat

arglist: `(expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
This function adds expression on main beat value. In expression you pt the kind of expression you want. In parts if you put :all, the rule will be applied on all parts. If you put 1 only in the first, second only in the second...

create-expression-on-note-sequence

arglist: `(expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
This function adds expression on consecutive single notes. In expression you pt the kind of expression you want. In parts if you put :all, the rule will be applied on all parts. if you put 1 only in the first, second only in the second...

create-face-value-expression

arglist: `(face-values? expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
In face value you put the rhythmical values on whom you want to apply an expression. In expression you put the kind of expression you want. In parts if you put :all, the rule will be applied on all parts. If you put 1 only in the first, second only in the second...

create-group-expression

arglist: `(indexes? expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
In indexes? you put a list of lists. The first value indicates from which index you want to start a group expression (like crescendo, or slur...), and the second indicates when you want the group to stop. In expression you pt the kind of expression you want. In parts if you put :all, the rule will be applied on all parts. if you put 1 only in the first, second only in the second... ATTENTION : index are in the cosntraints way: it means that 1 is for the first, 2 for the second... and not as in lisp... (O for the first, 1 for the second...)

create-individual-expression

arglist: `(index? expression parts)`
package: JBS-CONSTRAINTS
menu: Voice-Leading-Rules▶Create-Expressions-Tools
In index? you put the indexes on which you want to put some expression.. In parts if you put :all, the rule will be applied on all parts. if you put 1 only in the first, second only in the second... ATTENTION : index are in the cosntraints way: it means that 1 is for the first, 2 for the second... and not as in lisp... (O for the first, 1 for the second...)

descending-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS

menu: Pitch-Rules▶Shaping-Rules
This rule obliges all the value to be descending ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

descending-sub-group-no-repet-rule

arglist: `(nth-? mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule obliges the nth (put in nth-?) values of a list of lists to be descending without any repetition. ATTENTION : THE HEURISTIC MODE IT IS NOT WORKING

descending-sub-group-with-repet-rule

arglist: `(nth-? mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule obliges the nth (put in nth-?) values of a list of lists to be descending with repetitions. ATTENTION : The Heuristic implementation can not work.

descending-without-repetition-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule obliges all the value to be descending without any repetition. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

direct-analysis-rule

arglist: `(analysis mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule asks the engine to put out a solution identical to the one put in profile. As you understand, this rule is usefull only as heuristic! ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

distance-rule

arglist: `(pattern distance which? mode? weight)`
package: JBS-CONSTRAINTS
menu: Pattern-Rules▶Distance-Rules

This is a morphological rule. It asks to the engine those solutions who have a distance - given in 'distance - with the 'pattern. In which? you can chose if you want an equal distance '=, a more little distance '< or a bigger distance '>. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

do-not-reach-that-interval-rule

arglist: `(how-many interval mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
Does not reach a given interval in how-many notes

do-reach-that-interval-rule

arglist: `(how-many interval mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
Does reach a given interval in how-many notes

energy-profile-rule

arglist: `(energy-profile mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

find-apply-approx-absolute-sum-rule

arglist: `(sum approx mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
This rule finds out the candidates that summed together (in the absoulte mode) they give as a risult the same value put in sum. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

find-apply-approx-sum-rule

arglist: `(sum approx mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
This rule finds out the candidates that summed together they give as a risult the same value put in sum. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

find-apply-global-absolute-sum-rule

arglist: `(sum mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
This rule finds out the candidates that summed together (in an absolute mode) they give as a risult the same value put in sum. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

find-apply-global-sum-rule

arglist: `(sum mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
This rule finds out the candidates that summed together they give as a risult the same value put in sum. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

find-this-ptrn-n-times-rule

arglist: `(pattern repeat-ptrn mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Rules▶Pattern-Rules
This rule looks for solutions having patterns with a giveen length of element put in ptrn-length. In repeated-ptrn you have to put how many time do you want the pattern to be repeated. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

index-applied-sum-rule

arglist: `(index sum mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges the value (indicated as index) of the solution to have elements that summed together they give back the number put in sum. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible ATTENTION: HEURISTIC NOT YET IMPLEMENTED.

index-higher-rule

arglist: `(index value mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges the value (indicated as index) of the solution to be higher than the value indicated in 'value'.. ATTENTION : in the mode true/false, the rule is perfectly

applied. In the mode heuristic, the rule is applied as much as possible ATTENTION: HEURISTIC NOT YET IMPLEMENTED.

index-interval-rule

arglist: `(index allowed absolute? mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule obliges a given interval indicated with 'index' to be a member of a list of possible intervals indicated in 'allowed'. If the menu 'absolute?' is 'absolute, that means that intervals are intented in absolute mode. If this menu is 'up/down', that means that the intervals are divided into ascending and descending. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

index-length-rule

arglist: `(index length mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges the value (indicated as index) of the solution to have the length put in 'length'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible ATTENTION: HEURISTIC NOT YET IMPLEMENTED.

index-lower-rule

arglist: `(index value mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges the value (indicated as index) of the solution to be lower than the value indicated in 'value'.. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible ATTENTION: HEURISTIC NOT YET IMPLEMENTED.

index-nth-rule

arglist: `(index nth? what? mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges the nth (indicate from 0 to n) of the index (indicated with i1, i2, i3...) to be the value put in what?. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible ATTENTION: HEURISTIC NOT YET IMPLEMENTED.

index-pitch-rule

arglist: `(index pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
For the give index (i1, i2, i3...) only the pitches indicated in 'pitch' will be allowed in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

index-rule

arglist: `(index value mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges the value (indicated as index) of the solution to be the value indicated in 'value'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

interval-structure-rule

arglist: `(interval-structure mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule obliges elements to have the given 'interval-structure'. N.B. BE CAREFULL: the number of intervals put in 'interval-stricture' has to be one element less than the number of candidates you put in the search space!. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

item-sub-group-member-rule

arglist: `(sub-group-length item-index allowed mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
Item indicated with 'item index' will be a member of 'allowed' elements in each sub group of length 'sub-group-length'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

jump-resolution-rule

arglist: `(interval resolution mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
If an interval is higher than the value put in interval, the next interval as to go in the opposite direction and it has to be smaller than the value put in resolution.

length-sub-group-applied-sum-rule

arglist: `(length? mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
This rule obliges each sub list of the solution ti have an applied sum equal to length?. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

length-sub-group-rule

arglist: `(curve-min curve-max steps lengths mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges the sub solutions to have a length accordingly to the list put in 'lengths'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

logic-or-condition

arglist: `(&rest rules)`
package: JBS-CONSTRAINTS
menu: Utilities
It is an exensible box. TO BE CONTINUED...

make-?1

arglist: `(list)`
package: JBS-CONSTRAINTS
menu: Utilities
It creates a list of constraints candidates like ?1 ?2 ?3...

make-?1-from-to

arglist: `(from to)`
package: JBS-CONSTRAINTS
menu: Utilities
It creates a list of constraints candidates like ?from ... ?to

make-i1

arglist: `(list)`
package: JBS-CONSTRAINTS
menu: Utilities
It creates a list of i1 i2 i3 accordingly with the list of number you put in list.

make-i1-from-to

arglist: `(from to)`
package: JBS-CONSTRAINTS
menu: Utilities
It creates a list of constraints candidates like ?from ... ?to

member-rule

arglist: `(list mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges any element of the solution to belong to the elements indicated in 'domain'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

mk-chain-candidates

arglist: `(list groups)`
package: JBS-CONSTRAINTS
menu: Utilities
FROM MIKAEL LAUSRON : it creates a list of candidates that share a high level of common elements. In list you put all the range of candidates. In groups how many sub group you want to create.

mk-fix-profile-rule

arglist: `(profile mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule asks the engine to put out a solution identical to the one put in profile. As you understand, this rule is usefull only as heuristic! ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

mk-latin-matrix-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Structure-Rules▶Matrix-Rules
This rule create a latin matrix like following :
1 3 2 4
2 1 4 3
3 4 1 2
4 2 3 1

ATTENTION : It works with list of lists. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

mk-linear-candidates

arglist: `(lista how-many)`
package: JBS-CONSTRAINTS
menu: Utilities
it creates a list of candidates having for each element of list, with a positive and negative step indicated in 'how-many. Be careful the increasing or decreasing step is always 1.

mk-pitch-candidates

arglist: `(pitch octave)`
package: JBS-CONSTRAINTS
menu: Utilities
It creates a list of pitches entered in 'pitch for number of octaves indicate in 'octave. If you put 1 you will obtain one octave lower and upper of the original one.

mk-pitch-candidates-not-symmetric

arglist: `(pitch down up)`
package: JBS-CONSTRAINTS
menu: Utilities
It creates a list of pitch candidates. In pitch you put the note yuo want to be repeated. In down and up you indicate how many octaves you want the pitch to be transponsed down and in up.

mk-profile-pitch-modulo-rule

arglist: `(pitches mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
This rule asks the engine to put out a solution of pitches having the same modulo 12 of the given profile. It is a OTTAVIATORE. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

mk-profile-pitch-rule

arglist: `(curve-min curve-max steps profile approx mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
This rule asks the engine to put out a solution identical to the one put in profile. As you understand, this rule is usefull only as heuristic! ATTENTION : in the mode true/false,

the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

mk-profile-rule

arglist: `(curve-min curve-max steps profile decimals mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule asks the engine to put out a solution identical to the one put in profile. As you understand, this rule is usefull only as heuristic! ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

mk-range-candidates

arglist: `(lista how-many step)`
package: JBS-CONSTRAINTS
menu: Utilities
it creates a list of candidates having for each element of list, with a positive and negative step indicated in 'step and in the range indicated in 'range.

mk-symbol-structure-rule

arglist: `(structure mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
This rule obliges the solution to be as indicated in structure. You understand that this rule has meaning only as heuristic. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

modulo-x-repetition-rule

arglist: `(modulo mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule allows any candidate having the same modulo given in modulo. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible. N.B. USEFULL IN QUANTIFICATION OF DURATIONS.

more-first-repeated-than-second-rule

arglist: `(element-1 element-2 mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Rules▶Pattern-Rules

This rule asks the engine to have a solution with the first element repeated more times than the second. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-absolute-repetition-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
It does not allow any repetition in abslute mode inside a solution. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-consecutive-equal-intervals-rule

arglist: `(how-many mode? absolute? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule does not allow any repetition of intervals for a length put in 'how-many'. If the menu 'absolute?' is 'absolute, that means that intervals are intented in absolute mode. If this menu is 'up/down', that means that the intervals are divided into ascending and descending. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-consecutive-pulses-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
Two positive values can not be consecutive. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-consecutive-rests-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
Two negative values can not be consecutive. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-interval-local-repetition-rule

arglist: `(absolute? mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules

This rule does not allowed any local repetition of intervals. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-interval-repetition-rule

arglist: `(mode? absolute? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule does not allow any repetition of intervals. If the menu 'absolute?' is 'absolute, that means that intervals are intented in absolute mode. If this menu is 'up/down', that means that the intervals are divided into ascending and descending. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-local-repetition-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule dos not allow any repetition. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-locally-repeated-given-interval-rule

arglist: `(interval absolute? mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule obliges a solution not to have a given 'interval' locally repeated. It is a sort of no-local-repetition but limited to the given interval. ATTENTION : in the mode true/-false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-repetition-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule dos not allow any repetition. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-resulting-interval-rule

arglist: `(interval mode? weight)`
package: JBS-CONSTRAINTS

menu: Generic-Rules▶Interval-Rules
This rule does not allow the existence of the given 'interval' even as the result of the verticality of all the elements of a solution. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

no-shape-pattern-lcl-repetition-rule

arglist: `(ptrn-length mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Rules▶Pattern-Rules
This rule does not allow two consecutive pattern described in the dirct-analys format. In Italiano : questa regola non ammette due pattern consecutivi la cui descrizione basata sul direct analys di morphologie: chiama Filippo se non ti ricordi...

no-spaced-repetition-rule

arglist: `(candidates mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This function does not allow any repetition for an element and another. The distance between the two elements is indicated with the numbers put in 'candidates'. For example if you put (1 4) it means that each element has to be different from the fourth after itself. If you put (1 7) that means that an element has to be different from the seventh after itself. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible.

not-allowed-chain-rule

arglist: `(element following mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges a given element, NOT to be followed by those elements eneterd in following. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-allowed-intervals-rule

arglist: `(intervals absolute? mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule does not allow the intervals indicated in 'intervals'. If the menu 'absolute?' is 'absolute, that means that intervals are intented in absolute mode. If this menu is 'up/down', that means that the intervals are divided into ascending and descending. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-allowed-pitch-class-rule

arglist: `(pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
Only the class (for instance a minor triad) indicated in 'pitch' will be allowed in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule does not work because of SLEN...

not-allowed-pitch-class-sub-list-rule

arglist: `(pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
This function outputs a solution having only the class (for instance minor triad) indicated in 'pitch' will NOT be allowed in any octave including also other notes. That means that is I'm looking for NOT HAVING a minor triad in a 5 notes chord, the solution will look if a minor triad exists inside the 5 notes chord. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule does not work because of SLEN...

not-allowed-pitch-rule

arglist: `(pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
Only the pitches indicated in 'pitch' will be allowed in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-allowed-pitch-structure-rule

arglist: `(pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
Only the pitches indicated in 'pitch' will be allowed in any octave and in any position. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule does not work because of SLEN...

not-bigger-interval-rule

arglist: `(limit sign? mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
First you have to define if you work on positif or negatif intrevals using the menu 'sign?'. If you chose '+', it means that this function does not allow interval higher than the one

entered in 'limit' only for positif interval. If you chose '-' it means that this function does not allow interval higher than the one entered in 'limit' only for negatif interval. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-complementary-interval-rule

arglist: `(interval mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule does not allow the existence of a given interval as the product of two consecutive intervals. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-consecutive-ascending-rule

arglist: `(how-many mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule does not allow more ascending value than as indicate in how-many. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-consecutive-descending-rule

arglist: `(how-many mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule does not allow more descending value than as indicate in how-many. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-consecutive-equal-length-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
The length of two consecutive groups has not to be equal. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-consecutive-equal-rule

arglist: `(how-many mode? weight)`
package: JBS-CONSTRAINTS

menu: Pmc-Rules▶Generic-Rules
This rule does not allow more equal values than as indicate in how-many. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-consecutive-number-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule does not allow numeric candidate to be followed by its consecutive. It means that if candidates are: 1 2 3 4 5 there will not be 1 follow by 2, but by 3, or 4 or 5. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-higher-than-rule

arglist: `(limit mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges any element of the solution not to belong to the elements indicated in 'domain'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-index-interval-rule

arglist: `(index allowed absolute? mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule obliges a given interval indicated with 'index' NOT to be a member of a list of possible intervals indicated in 'allowed'. If the menu 'absolute?' is 'absolute, that means that intervals are intented in absolute mode. If this menu is 'up/down', that means that the intervals are divided into ascending and descending. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-index-pitch-rule

arglist: `(index pitch mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
For the give index (i1, i2, i3...) only the pitches indicated in 'pitch' will NOT be allowed in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-index-rule

arglist: `(index value mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges the value (indicated as index) of the solution NOT to be the value indicated in 'value'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible ATTENTION: HEURISTIC NOT YET IMPLEMENTED.

not-interval-structure-rule

arglist: `(interval-structure mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule obliges elements not to have the given 'interval-structure'. N.B. BE CAREFULL: the number of intervals put in 'interval-stricture' has to be one element less than the number of candidates you put in the search space!. ATTENTION : in the mode true/-false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-length-repetition-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
No repetition of sub group equal length. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-lower-than-rule

arglist: `(limit mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges any element of the solution not to belong to the elements indicated in 'domain'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-member-rule

arglist: `(list mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule obliges any element of the solution not to belong to the elements indicated in 'domain'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-modulo-x-local-repetition-rule

arglist: `(modulo mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule does not allow any candidate having the same local modulo given in modulo. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible N.B. USEFULL IN QUANTIFICATION OF DURATIONS.

not-modulo-x-repetition-rule

arglist: `(modulo mode? weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule does not allow any candidate having the same modulo given in modulo. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-modulo12-local-repetition-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
This rule allows only solution without any LOCAL repetition in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-modulo12-repetition-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
This rule allows only solution without any repetition in any octave. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-obliged-interval-chain-rule

arglist: `(interval int-list mode? absolute? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule obliges an interval NOT to be followed by those put in int-list. If the menu 'absolute?' is 'absolute, that means that intervals are intented in absolute mode. If this menu is 'up/down', that means that the intervals are divided into ascending and

descending. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-ptrn-find-rule

arglist: `(ptrn-length repeat-ptrn mode?  weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Rules▶Pattern-Rules
This rule looks for solutions NOT having patterns with a giveen length of element put in ptrn-length. In repeated-ptrn you have to put how many time do you want the pattern to be repeated. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-repeated-element-sub-group-rule

arglist: `(sub-group-length mode?  weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule does not allow any repeated element into a given sub-group. The input sub-group-length indicates the length of the sub-group. ATTENTION : in the mode true/-false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-repeated-list-sub-group-rule

arglist: `(sub-group-length mode?  weight)`
package: JBS-CONSTRAINTS
menu: Pmc-Rules▶Generic-Rules
This rule does not allow any repeated list into a given sub-group. The input sub-group-length indicates the length of the sub-group. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-repeated-modulo12-sub-group-rule

arglist: `(sub-group-length mode?  weight)`
package: JBS-CONSTRAINTS
menu: Interval-Rules▶Pitch-Rules
There are no modulo 12 repetition inside each sub group indicated with 'sub-group-length'. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-rpt-elmts-in-lists-rule

arglist: `(mode?  weight)`
package: JBS-CONSTRAINTS

menu: Pmc-Rules▶Generic-Rules
This rule does not allow any element of one list to be repeated in the following list. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

not-smaller-interval-rule

arglist: `(limit sign? mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
First you have to define if you work on positif or negatif intrevals using the menu 'sign?'. If you chose '+', it means that this function does not allow interval lower than the one entered in 'limit' only for positif interval. If you chose '-' it means that this function does not allow interval lower than the one entered in 'limit' only for negatif interval. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

obliged-interval-chain-rule

arglist: `(interval int-list mode? absolute? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule obliges an interval to be followed by those put in int-list. If the menu 'absolute?' is 'absolute, that means that intervals are intented in absolute mode. If this menu is 'up/down', that means that the intervals are divided into ascending and descending. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

pitch-extract-from-score-editor

arglist: `(complex-list mensural-or-not?)`
package: JBS-CONSTRAINTS
menu: Utilities
This function extracts pitches from a Score-Editor in the chord format. That means that the output for a songle note will be like this (60) and for a chord like this (60 62 69)

ptrn-find-rule

arglist: `(ptrn-length repeat-ptrn which? mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Rules▶Pattern-Rules
This rule looks for solutions having patterns with a giveen length of element put in ptrn-length. In repeated-ptrn you have to put how many time do you want the pattern to be repeated. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

repeat-interval-rule

arglist: `(interval which? times mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule obliges a solution to have a given interval repeated many times as indicated in time. The interval is considered ins the absolute mode. If the menu which? is set on $<$, it means that the given interval has to be repeated a number of time inferior to the one put in times. If the menu which? is set on $=$, it means that the given interval has to be repeated a number of time equal to the one put in times. If the menu which? is set on $>$, it means that the given interval has to be repeated a number of times bigger than the one put in times. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

repeat-resulting-interval-rule

arglist: `(resulting-interval which? times mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
This rule obliges a solution to have a given resulting-interval repeated many times as indicated in time. A resulting interval is an interval between a note with any possible other notes. In this sense, look at the function find-all-intervals (that you can call using the package jbs-constraints::find-all-intervals). This function gives all the interval between all notes of a sequence. So a resulting interval is an interval that is the result of the function find-all-intervals.
If the menu which? is set on $<$, it means that the given interval has to be repeated a number of time inferior to the one put in times. If the menu which? is set on $=$, it means that the given interval has to be repeated a number of time equal to the one put in times. If the menu which? is set on $>$, it means that the given interval has to be repeated a number of times bigger than the one put in times.
ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

repeated-pattern-rule

arglist: `(pattern times which? absolute? mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Rules▶Pattern-Rules
This rule is for pattern repetitions. In 'pattern you put what you want to be repeated. In 'times you put how many times you want teh pattern to be repeated. In 'which? you can chose among '$<$ it that means less times, '$=$ that means an exact number of times and '$>$ that means more times. In 'absolute? you can chose if you are looking for positive elements (absolute) or not. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

resulting-interval-rule

arglist: `(interval mode? weight)`
package: JBS-CONSTRAINTS
menu: Generic-Rules▶Interval-Rules
A resulting interval is an interval between a note of a sequence with any possible other notes of the same sequence. In this sense, look at the function find-all-intervals (that you can call using the package jbs-constraints::find-all-intervals). This function gives all the interval between all notes of a sequence. So a resulting interval is an interval that is the result of the function find-all-intervals. Resulting-interval-rule [1] obliges the solution to have, among all the intervals among all eh notes of the sequence, the defined interval. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

s-pmc-all-notes-included-on-beat-rule

arglist: `(beat-number all-notes mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
all the notes set in all-notes have to be on a given beat in every measure

s-pmc-all-notes-included-rule

arglist: `(all-notes mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
all notes of a chord of three notes into 4 parts

s-pmc-allowed-harm-int-rule

arglist: `(intervals mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
The given intervals set in INTERVALS have be inside the solution.

s-pmc-allowed-harm-rule

arglist: `(mainchords chord-subsets mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
allowed harmony even incomplete

s-pmc-allowed-harmony-in-given-measures-rule

arglist: `(harmony? measures mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
This rule forces one or a series of measures to have a specific harmony set in harmony? (in modulo 12)

s-pmc-allowed-harmony-on-beat-rule

arglist: `(harmony? beats? mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
This rule forces one or a series of beats to have a specific harmony set in harmony? (in modulo 12)

s-pmc-allowed-interval-rule

arglist: `(intervals absolute? parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Melodic-Rules▶Generic-Poly-Rules▶Interval-Poly-Rules
This rule outputs a solution having only the interval entered in allowed-intervals. With the menu intervall-mod you can chose if the intervals are in absolute value or not. In the menu mode? you can chose if the rule is true /faulse or heuristic. ATTENTION: in parts if :all it means that the rule is vfor all the parts. If you want specific part to be constrained you have to put: :parts1, or :parts2...

s-pmc-allowed-pitch-class-sub-group-rule

arglist: `(pitch parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Generic-Poly-Rules▶Interval-Poly-Rules▶Pitch-Poly-Rules
This function outputs a solution where the class (for instance minor triad) indicated in 'pitch' WILL be allowed in any octave including also other notes. That means that is I'm looking for NOT HAVING a minor triad in a 5 notes chord. ATTENTION IN HEURISTIC MODE NOT YET IMPLEMENTED CORRECTELY

s-pmc-allowed-pitch-rule

arglist: `(pitch parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Generic-Poly-Rules▶Interval-Poly-Rules▶Pitch-Poly-Rules
This rule outputs a solution having only the pitch (in modulo 12) entered in pitch. In the menu mode? you can chose if the rule is true /faulse or heuristic. ATTENTION: in parts if :all it means that the rule is vfor all the parts. If you want specific part to be constrained you have to put: :parts1, or :parts2...

s-pmc-bigger-int-between-2-parts-rule

arglist: `(part1 part2 interval mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
bigger vertical intervals between two parts

s-pmc-chords-succession-rule

arglist: `(database mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
data base oriented chords succession with pwgl value

s-pmc-forbid-harm-int-rule

arglist: `(intervals mode? weight)`
package: JBS-CONSTRAINTS
menu: Structure-Rules▶Matrix-Rules
The given intervals set in INTERVALS have be inside the solution.

s-pmc-forbidden-int-relation-between-2-parts-rule

arglist: `(part1 part2 interval mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
forbidden interval relation between two parts with a given interval (false relazioni do do#...do# do...)

s-pmc-forbidden-inversions-rule

arglist: `(database mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
forbidden inversions...no quarta e sesta

s-pmc-forbidden-succession-rule

arglist: `(interval1 interval2 mode? weight)`
package: JBS-CONSTRAINTS
menu: Harmonic-Rules▶Voice-Leading-Rules
no triton followed by a fifth...

s-pmc-given-voice-rule

arglist: `(given-voice parts mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Poly-Rules▶Resolutions-Poly-Rules▶Shaping-Poly-Rules
given voice

s-pmc-hidden-parallel-rule

arglist: `(intervals mode? weight)`
package: JBS-CONSTRAINTS
menu: Harmonic-Rules▶Voice-Leading-Rules
hidden parallel fifths/octs with stepwise upper voice

s-pmc-index-all-notes-included-rule

arglist: `(index all-notes mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
This rule forces a solution to have the number of different notes set in 'all-notes' on a given index.

s-pmc-index-allowed-harmony-rule

arglist: `(harmony? index mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
This rule allows a specific harmony set in harmony? on a giv en index.

s-pmc-index-higher-rule

arglist: `(index value parts mode? weight)`
package: JBS-CONSTRAINTS
menu: Score-Pmc-Rules▶Melodic-Rules▶Generic-Poly-Rules
Its an index rule obliging an index to be higher than 72 (in this example) for a given part.

s-pmc-index-lower-rule

arglist: `(index value parts mode? weight)`
package: JBS-CONSTRAINTS
menu: Score-Pmc-Rules▶Melodic-Rules▶Generic-Poly-Rules
Its an index rule obliging an index to be lower than 72 in this example for a given part.

s-pmc-index-not-allowed-harmony-rule

arglist: `(harmony? index mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
This rule allows a specific harmony set in harmony? on a giv en index.

s-pmc-index-rule

arglist: `(index value parts mode? weight)`
package: JBS-CONSTRAINTS
menu: Score-Pmc-Rules▶Melodic-Rules▶Generic-Poly-Rules
Its an index rule for a give part....

s-pmc-interval-bigger-rule

arglist: `(interval parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Melodic-Rules▶Generic-Poly-Rules▶Interval-Poly-Rules
intervals have to be bigger than....

s-pmc-interval-smaller-rule

arglist: `(interval parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Melodic-Rules▶Generic-Poly-Rules▶Interval-Poly-Rules
intervals have to bi bigger than....

s-pmc-intv-between-2-parts-rule

arglist: `(part1 part2 intervals mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
allowd vertical intervals between contralto and tenor

s-pmc-jump-resolution-rule

arglist: `(jump-size resolution parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Interval-Poly-Rules▶Pitch-Poly-Rules▶Resolutions-Poly-Rules
The PMC-JUMP-RESOLUTION-RULE defines that if there is a jump bigger than the one set in jump-size, the next interval has to be smaller than the one set in resolution and in the opposite direction than previous.

s-pmc-mk-profile-rule

arglist: `(curve-min curve-max steps profile parts mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Poly-Rules▶Resolutions-Poly-Rules▶Shaping-Poly-Rules
It is the mk-fix-profile-rule but for the score-pmc

s-pmc-n-ascending-rule

arglist: `(how-many parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Score-Pmc-Rules▶Melodic-Rules▶Generic-Poly-Rules
no more than Xnote ascending....

s-pmc-n-descending-rule

arglist: `(how-many parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Score-Pmc-Rules▶Melodic-Rules▶Generic-Poly-Rules
no more than Xnote descending

s-pmc-no-crossing-voice-rule

arglist: `(mode? weight)`
package: JBS-CONSTRAINTS
menu: Harmonic-Rules▶Voice-Leading-Rules
no part-crossings

s-pmc-no-lcl-repetition-rule

arglist: `(parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Score-Pmc-Rules▶Melodic-Rules▶Generic-Poly-Rules
no local repetition in the given voice....

s-pmc-no-open-parallel-rule

arglist: `(intervals mode? weight)`
package: JBS-CONSTRAINTS
menu: Harmonic-Rules▶Voice-Leading-Rules
no open parallel...

s-pmc-no-reached-intrv-rule

arglist: `(how-many interval parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Melodic-Rules▶Generic-Poly-Rules▶Interval-Poly-Rules
do not reach a given interval in three given notes...

s-pmc-not-allowed-harm-int-rule

arglist: `(intervals mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
The given intervals set in INTERVALS have be inside the solution.

s-pmc-not-allowed-harmony-in-given-measures-rule

arglist: `(harmony? measures mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
This rule forces one or a series of measures NOT to have a specific harmony set in harmony? (in modulo 12)

s-pmc-not-allowed-harmony-on-beat-rule

arglist: `(harmony? beats? mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
This rule forces one or a series of beats to NOT have a specific harmony set in harmony? (in modulo 12)

s-pmc-not-allowed-interval-rule

arglist: `(intervals absolute? parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Melodic-Rules▶Generic-Poly-Rules▶Interval-Poly-Rules
This rule outputs a solution having only the interval entered in allowed-intervals. With the menu intervall-mod you can chose if the intervals are in absolute value or not. In the menu mode? you can chose if the rule is true /faulse or heuristic. ATTENTION: in parts if :all it means that the rule is vfor all the parts. If you want specific part to be constrained you have to put: :parts1, or :parts2...

s-pmc-not-allowed-pitch-class-sub-group-rule

arglist: `(pitch parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS

menu: Generic-Poly-Rules▶Interval-Poly-Rules▶Pitch-Poly-Rules
This function outputs a solution where the class (for instance minor triad) indicated in 'pitch' will NOT be allowed in any octave including also other notes. That means that is I'm looking for NOT HAVING a minor triad in a 5 notes chord. ATTENTION IN HEURISTIC MODE NOT YET IMPLEMENTED CORRECTELY

s-pmc-not-allowed-pitch-rule

arglist: `(pitch parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Generic-Poly-Rules▶Interval-Poly-Rules▶Pitch-Poly-Rules
This rule outputs a solution having only the pitch (in modulo 12) entered in pitch. In the menu mode? you can chose if the rule is true /faulse or heuristic. ATTENTION: in parts if :all it means that the rule is vfor all the parts. If you want specific part to be constrained you have to put: :parts1, or :parts2...

s-pmc-not-higher-rule

arglist: `(limit parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Score-Pmc-Rules▶Melodic-Rules▶Generic-Poly-Rules
The S-PMC-NOT-HIGHER-RULE is a rule obliging any value to be higher than a given number

s-pmc-not-intv-between-2-parts-rule

arglist: `(part1 part2 intervals mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
allowd vertical intervals between contralto and tenor

s-pmc-not-lower-rule

arglist: `(limit parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Score-Pmc-Rules▶Melodic-Rules▶Generic-Poly-Rules
S-PMC-NOT-LOWER-RULE is a rule obliging any value to be lower than a given number

s-pmc-not-n-consecutive-harm-int-rule

arglist: `(part1 part2 repetition mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
in repetition you put how many equal intervals you do not allowed between two parts. The concerned parts are to be specified in part1 and part2.

s-pmc-not-n-same-directions-rule

arglist: `(part1 part2 repetition mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
This rule does not admit more than n repetitions of the same directions between two voices. In part1 and part2 you define which are the intersted parts. In repetition you set how many same directions do you admit.

s-pmc-not-tone-resolution-rule

arglist: `(sensibile resolution parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Interval-Poly-Rules▶Pitch-Poly-Rules▶Resolutions-Poly-Rules
the sensibile (in mod 12) has NOT to solve on resolution (in mod12)...

s-pmc-or-condition

arglist: `(rule1 rule2)`
package: JBS-CONSTRAINTS
menu: Utilities
:or condition from Laurson...

s-pmc-preferred-duplicate-rule

arglist: `(dups mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
prefers a given duplicates in mod12

s-pmc-smaller-int-between-2-parts-rule

arglist: `(part1 part2 interval mode? weight)`
package: JBS-CONSTRAINTS
menu: Shaping-Poly-Rules▶Harmonic-Rules
smaller vertical intervals between two parts

s-pmc-tone-resolution-rule

arglist: `(sensibile resolution parts mode? weight &optional expression)`
package: JBS-CONSTRAINTS
menu: Interval-Poly-Rules▶Pitch-Poly-Rules▶Resolutions-Poly-Rules
the sensibile (in mod 12) has to solve on resolution (in mod12)...

structured-order-sum-rule

arglist: `(candidates order sum mode? weight)`
package: JBS-CONSTRAINTS
menu: Distance-Rules▶Structure-Rules
In candidates you put in sconstraints symbol the candidates you are looking for (ex. ?1 ?2 ?3). Then in order you put a list of indexes that has to be applied to a posn-match for the candidates /////(ex. (0 1 0 2 2 2 2 0 1)). Then in sum you put the value that the 3 candidates you haveindicated, summmed together in the given order. ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

sub-group-mk-profile-rule

arglist: `(curve-min curve-max nth-? steps profile mode? weight)`
package: JBS-CONSTRAINTS
menu: Pitch-Rules▶Shaping-Rules
This rule asks the engine to put out a solution in which for each sub-groups the nth (put in nth-?) has to be identical to the one put in profile. As you understand, this rule is usefull only as heuristic!ATTENTION : in the mode true/false, the rule is perfectly applied. In the mode heuristic, the rule is applied as much as possible

Box Index