Appendix A

Rhythm-Box

A.1 Tutorial: searching rhythms

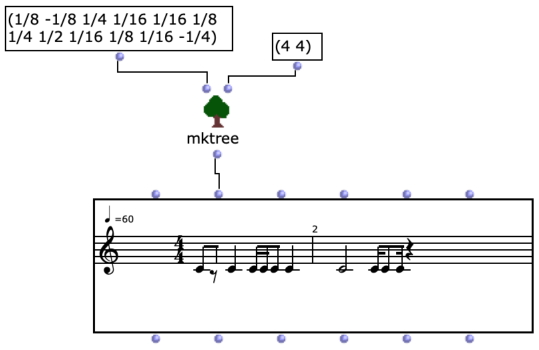
This section shows how to use the Rhythm-Box tool to generate rhythms from an input rhythm. The first steps are to install Gecode[1] and OpenMusic[2] and then download and install the GiL and Rhythm-Box OpenMusic libraries from `https://github.com/blapiere/GiL` and `https://github.com/blapiere/Rhythm-Box`[3]. Pay attention, currently only computers that can support dylib files can use GiL.

Once this is done, open OpenMusic and create a new patch. Create a voice object (eihter double click on the background of the patch and write "voice" or select it from the menu “Classes > Score > VOICE”) and add a rhythm tree. If you are not familiar with the rhythm trees, you can use the the OpenMusic function “MKTREE” to generate one from a list of ratios and a list of time signatures.

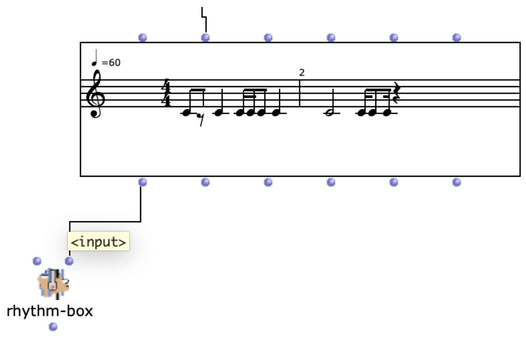
[1]`https://www.gecode.org/download.html`

[2]`https://openmusic-project.github.io/openmusic/`

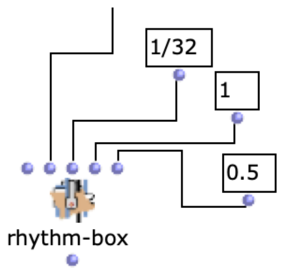
[3]To add libraries to OpenMusic: `http://support.ircam.fr/docs/om/om6-manual/co/UserLibraries.html`



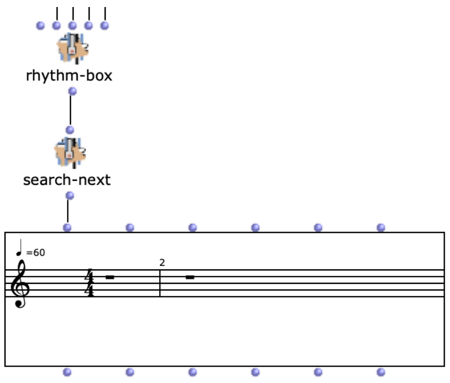
The next step is to create the rhythm-box. Double-click on the patch background and write "rhythm-box". The method will appear, with initially two inlets: the constraints and the input. Link the *self* or *tree* outlet of the voice to the *input* inlet of the rhythm-box.



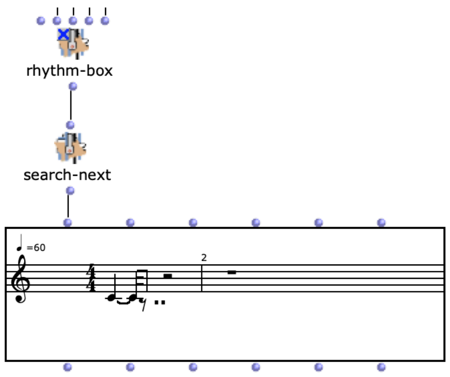
If you evaluate the rhythm-box now, a base CSP is created with data from the input. But let's modify this data. In our new rhythm, we want the minimum duration of an event to be 1/32. The default minimum duration is the one of the the input (here 1/16), so we have to specify it. Reveal the optional *s* inlet of the rhythm-box by selecting it and pressing ALT+→, and add a primitive 1/32 box. Finally, as the 1/32 shortest duration will allow the space to generate up to 64 events and since we don't want that many events, we will set the *hd* inlet, i.e. the density handle, to 0,5, which will reduce the the maximum number of events to its half.



In that state, the rhythm-box can already generate rhythms. Add a search-next box connect the output of the rhythm-box to its input. Then, add a new voice object and connect the output of the search-next box to its *self* inlet. You can evaluate the new voice (press V while selecting it) and generate rhythms.



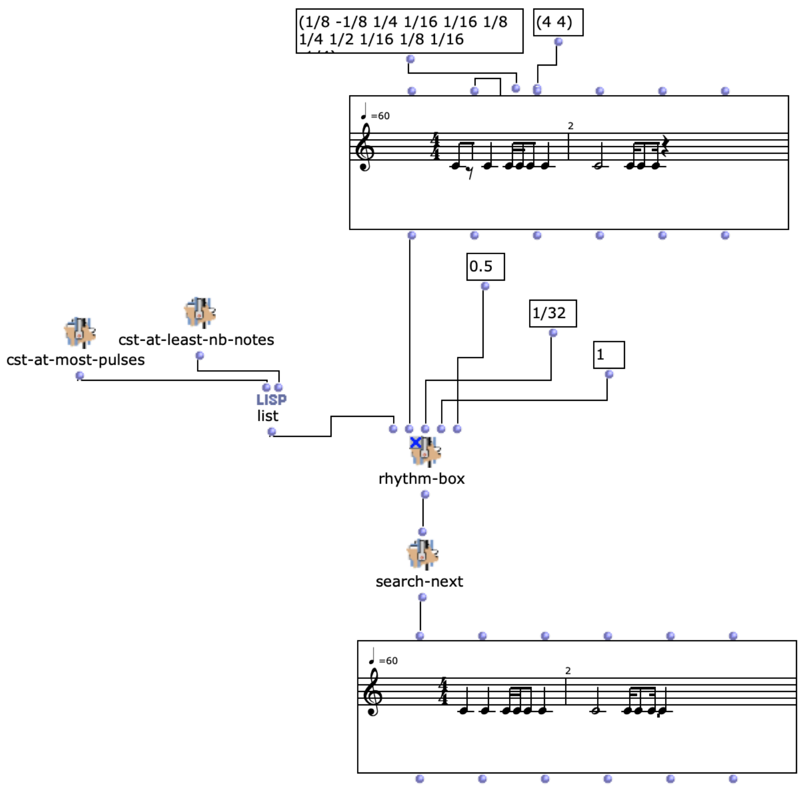
You will notice that the search always returns the same score without any note. The reason is simple: in this configuration, a silent rhythm is always the first generated solution. And each time the output score is evaluated, the evaluation process evaluates the rhythm-box as well, creating a new identical CSP with the same first solution at each evaluation. To solve this inconvenience, the evaluation of the rhythm-box must be blocked once it is first evaluated. Select the rhythm-box and press B. A small blue cross appears in its top-left corner, meaning its evaluation is blocked. You can now search for rhythms by evaluating the output voice.



The results are not so interesting for the moment. It is time to add constraints to our base CSP. Let's say we want a rhythm with the at least the same number of notes as the input, but with at most the same pulses (i.e. if there is a pulse in the output, there must be a pulse at this position in the input).

Create a Lisp list, select it and press ALT+$\rightarrow$ twice to reveal two inlet. Then, create a "cst-at-least-nb-notes" and a "cst-at-most-pulses" boxes, and link their output to the list inlets. You can now link the list outlet to the *csts* inlet of the rhythm-box.

To evaluate the addition of the new constraints, you have to first unlock the rhythm-box evaluation, then evaluate it, and then lock it again. You can now evaluate the output score and see the results.



This tutorial is over. A catalogue of available constraints can be found in the next section. Keep in mind the locking/unlocking of the rhythm-box when changing constraints or input.

A.2 Constraints catalogue

This section presents the exhaustive list of currently available constraints.

- **cst-rel-pulse** The specified positions must contain a note of duration with the specified relation to the specified value (e.g. equal to 1/4 or greater than 1/16).

- **cst-keep-pulses** The output rhythm must include at least the pulses contained in the input rhythm.

- **cst-at-most-pulses** If there is a pulse in the output, there must be a pulse at this position in the input.

- **cst-at-most-nb-notes** The output rhythm can not contain more notes than the input.

- **cst-at-least-nb-notes** The output rhythm can not contain less notes than the input.

- **cst-keep-note-drts** The output rhythm can not have events of different duration than the durations present in the input.

- **cst-keep-nb-pulses** The number of pulses in the output is exactly the same as the number of pulses in the input.

- **cst-at-least-notes** All the notes from the input are in the output (their positions can change, but not their durations).

- **cst-insert-pattern** The output rhythm is the same as the input, but with the specified pattern inserted in place of the specified pulses. The length of the inserted pattern is dependant of the duration of the pulse it replaces.

A.3 Tutorial: create a new constraint

This section shows how to create new constraints for the Rhythm-Box. The example will be constraining the output rhythm to have all its notes duration greater than the previous note duration in the rhythm, and to have a note at position 0 with duration equal to an input duration.

Open the Lisp Editor (menu "Windows > Lisp Editor"). The first step is to express our new rule in terms of constraint. In our case, it would be:

$\forall i \in [1, n_{max}[: \ drt_i > 0 \Leftrightarrow drt_i > drt_{i-1}$

Create a Lisp function that expresses this constraint using GiL. All the constraint functions must use the same arguments list containing all the input parameters of the CSP plus a list containing their own arguments. To ease the writing, begin by defining the function with only the needed parameters:

```
(in-package :om)

;<value> is the duration of the first event (e.g. 1/4)
;<sp> is search space of the problem
;<drt> is the list of references to the duration variables
;<s> is the shortest note duration (required to convert a ratio into
; a tick duration)
;<nmax> is the maximum number of events in the output
(defun my-cst (value sp drt s nmax)
    (gil::rel (first drt) gil::IRT_EQ (* s value))
    (loop for i from 1 below nmax do
        ;first create the boolean expressions.
        (let ((b1 (gil::add-bool-var-expr sp (nth i drt) gil::IRT_GR
            0))
              ;if an event duration is greater than 0 (i.e. it
                exists and is not a rest) ...

              (b2 ((gil::add-bool-var-expr sp (nth i drt)
                gil::IRT_GR (nth (- i 1) drt)))))
              ;... then the preceding event is of shorter duration.

            ;and then express the equivalence relation between them.
            (gil::g-op sp b1 gil::BOT_EQV b2))
    )
)
```

Now, you can create a function that calls that function and that takes all the needed parameters. Note that our specific parameter that defines the duration of the first note (“value” in the code above) is hidden in the “args-list” parameter and axtracted when the my-cst function is called.

```
(defun post-my-cst (sp pos drt drt* N Dn s l duration nmin nmax
   input args-list)
    (my-cst (first args-list) sp drt nmax)
)
```

The last step is to create an OpenMusic method box that takes a ratio in input and returns a constraint. All constraint in Rhythm-Box are instances of the class *constraint*, that hold a constraint function and its list of specific parameters.

```
(defmethod! cst-my-cst (value)
    :initvals '(1/4)
    :indoc '("a␣ratio")
    :icon 262
    :icon 262
    :doc "
This␣is␣my␣first␣constraint.
"
    (make-instance 'constraint
        :cstf #'post-my-cst
        :args (list value))
)
```

The new constraint is ready to be used in an OpenMusic program.