UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

Interface and application of constraint programming for musical composition

Dissertation presented by
Geoffroy ZOETARDT

for obtaining the Master's degree in
Computer Science

Supervisor(s)
Peter VAN ROY

Jean BRESSON, Sascha VAN CAUWELAERT

Academic year 2015-2016

Acknowledgements

I would like to thank people that helped and supported me all along this master's thesis. First, my supervisor Mr Peter Van Roy for allowing me to work on this subject, for all his help and interest. Then, I thank Jean Bresson as well as Gustavo Gutierrez that were always ready to help me in spite of the distance. I also thank Sascha Van Cauwelaert for his explanations about his work and Torsten Anders for his interest in my work and for the documentation and the information they provided me about the state of the art in this topic. And last but not least, my family and friends for their supports and their advices.

Contents

Chapter 1

Introduction

1.1 Foreword

Music has always been impacted by technological changes and mainly during the last centuries. It's logical that the computer sciences technologies and music have met and started a journey together. From the writing of scores to musical creations without the need of human, the computer sciences have found their place into the music world. A software named MEDALmusic system[1] has been developed in the end of the 90's by René-Louis Baron[2] to create musical compositions by itself that sound like human realisation. By the way, this has proved that the musical creation is not only subjective and abstract but has logical foundations transposable into mathematics form and, as a logical continuation, computer sciences world.
This evolution offers a wide world of new possibilities that passionate and motivate dreamers, musicians, searchers and scientists, such as the IRCAM's members[3] and myself.

Idea

« What if computers can reduce the distance between the abstract ideas of a compositor and the effective writing score of those ideas? ». This master thesis aims to provide a base solution to that question. Starting from an existing software that offers tools and environment to help compositors to generate musical score, this project intends to establish a foundation for the development of a high-level solution.
Artificial Intelligence techniques will allow the musical composer to focus on the main ideas and to leave the composition and music writing to the computer. Let's take a host software as OpenMusic and a constraint programming toolkit as Gecode ... and the journey can begin.

[1] System that generates automaticly musical creation[14]

[2] French inventor, autor, compositor and performer that has invented and developed the MEDALmusic system in 1998[13]

[3] IRCAM is an institution of research about music, a short presentation is given in section 2.2.1

1.2 Thesis' goals

The aims of this master's thesis is to interface functionalities of constraint programming, create a user's interface easy to use and provide mechanism to make it easy to improve. All of this has to be integrated into an existing programming language specialized in musical composition, OpenMusic. [4]. This contribution will help solving complex musical problems such as the construction of musical canons. Problems that usually take a lot of time and are complex to resolve will become easier and take only some second.

As the target of this integration is specified to music domain and more particularly musical composition, the interface and its functionalities must be oriented to this domain. The users of OpenMusic are principally compositors not familiar with constraint programming, that's why the user's interface to the constraint programming part should be easy to use, as understandable as possible and should only contain useful objects while the effective calculation would be realized in the background.

Another important point is the improvement of this system. As the duration of a master's thesis is only one year, the system must provide a way for further and continuous improvement. It should be easy to create new constraints to expand the possibilities offered to the users. And it also has to be easy for a developer to understand the interface and to add functionalities.

[4] Visual programming language designed and developed by the IRCAM Music Representation search group [9]. More explanations at the section 2.2.1

Chapter 2

Presentation of notions and tools

In this chapter, you will be introduced to the main notions and tools that are necessary for a good understanding of this project.

2.1 Notions

2.1.1 Musical composition

This section will provide a quick definition of musical composition and introduce some useful notions for the understanding of this paper.

Musical composition is the act of creating a song, a piece of music. To do so, a compositor creates pieces of music that express the sound of his thought. This step covers the whole process from the birth of a musical idea to the writing of a musical score.
This process manipulates some notions such as the notes, the chords and the musical score [1]. They can be represented by a set of musical concepts. Let's focus on some of them that will be useful for the comprehension of this paper:

- Pitch: It is the value representing the frequency of a note. For example, the A4 is the most common A with a frequency of 440 Hz and is currently represented by the number 69 in the MIDI notation[2].
- Duration: It can express the tempo, the beat, the metre[3] or a rhythm. For us, the notion of duration will be limited to a short part of the rhythm: the lifetime of a note.
- Velocity or dynamic: it represents the loudness or the softness with which a note is played.
- Color of the sound: It regroups all that defines the sound. It includes the type of instrument used, the musical effects applied, the sound quality, ...

A note is a unique instance of these concepts while a chord is composed of notes and a score can be seen as a sequence of chords. This last precision will be important for the

[1] All along this paper the term *"musical score"* will be shortened by *"score"*.
[2] MIDI or Musical Instrument Digital Interface is a technical standard describing a computer protocol representing musical value.[8]
[3] The way beats are grouped, also call the time signature of a score.

rest of the analysis.
Then, the musical composition also uses the notion of musical rules. Music has rules in the same way as a language has a grammar. They help to structure, with flexibility, a musical composition. The most known example is the rules of harmony. It is a set of rules that help to harmonize a composition and make it pleasant to the ear. For example a minor third is a music interval that express an harmonic rules where the harmonic distance between two note is of a tone and a semitone.
Those notions will lead a large part in the logic of the following.

2.1.2 Constraint programming

In computer sciences, one of the paradigm is the constraint programming. Part of the artificial intelligence, this technology allows to search a solution to problems in a wide range of possibilities in a relatively short time. This approach consists in two activities: modelling the problem and solving it.

The first part is the modelling. There, we declare decision variables with a finite domain representing the set of possibilities and on which the search can be executed. Then the constraints are created to specify conditions and relations on a single variable or between different variables. When all those constraints have been specified, we talk about propagators. The propagators represent the execution of the constraints on the decision variables and, through that, the modification of variables' domain to only keep the solutions that satisfy all the constraints.
The second part is the search. Once all constraints and decision variables have been defined, the search can start. In constraint programming, the search can be seen as a reversed tree: at the root we can find the basic state where no variable has an assigned value. From this root state, each possible assignation creates a new branch in the tree that leads to a new state where at least one additional variable has been assigned. Thereby, after each branching, the domains of the variables are decreased and lead to a state where all the variable are bound to one unique value, if it exists.

2.1.3 Example of Music Constraint Satisfaction Problem (MCSP)

This section presents an example of musical problem resolved with constraint programming. This example is the "all-intervals" problem. It has been defined by Torsten Anders in his paper "Constraint Programming Systems for Modeling Music Theories and Composition"[4].

An interval is the difference between two pitches. These intervals might be horizontal ("melodic") when describing the difference between two notes successively played, or vertical ("harmonic") to mark the difference between two notes played at the same time, as shown in the figure below.

[4]This definition is provided at the page 30:5 of [21] and Sascha Van Cauwelaert has also developed this problem in his master's thesis [23]

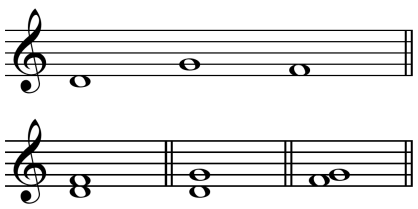


Figure 2.1: Melodic and harmonic intervals.[5]

The definition of an all-interval series given in the paper of Torsten Anders is :

> *"In this technique, the composer organizes the pitches in a composition with the help of a tone row (also known as twelve-tone series). A tone row is a sequence of 12 tones that arranges the twelve chromatic pitch classes (tone names) in a particular order, where each pitch class occurs exactly once. A row can be transformed in many ways. The most common transformations are transposition, retrograde (reversal in time), and inversion (mirrored along a pitch axis). Composers use tone rows as a device to achieve coherency in music, even if the music abandons conventional harmony. [...] In an all-interval series, not only the 12 pitch classes but also the eleven intervals between them are pairwise distinct (i.e., each interval occurs only once)"*

To solve this problem in Gecode, two decision variables and five constraints are necessary. The two decision variables are the following:

- *pitchClasses*: It is an array of integer variables that stands between 0 and $n-1$ where n is the size of the list.
- *intervals*: array of integer variables of size $n-1$ and where the domain is between 1 and $n-1$. The n used is still the size of pitchClasses.

And the involved constraints are:

- The first constraints forces the value of the i^{th} element of *intervals* to an inversional equivalent interval between the i^{th} element of pitchClasses and the $i+1^{\text{th}}$.

$$\bigwedge_{1}^{n-1} inversionalEquivalentInterval(pitchClasses_i, pitchClasses_{i+1}, Intervals_i)$$

- The second and the third constraints force the values of each list, pitchClasses and intervals, to be distinct.

$$distinct(pitchClasses) \wedge distinct(intervals)$$

- The last two constraints impose the first element of pitchClasses to be equal to 0 and the last to be equal to the half of the size of the list, $n/2$.

$$pitchClasses_0 = 0 \wedge pitchClasses_{n-1} = n/2$$

This example will illustrate the different notions all along this master's thesis.

[5]Figure source [5].

2.2 Tools

2.2.1 OpenMusic

OpenMusic[9] is a visual programming language oriented to music composition and sound manipulation developed by the IRCAM. It is based on Lisp[6], one of the oldest functional language. The particularity of the visual programs is that, instead of writing lines of code, the user simply drags and drops icons representing elements of the language such as function or data structures. With that way, OpenMusic aims to be opened to people who have no knowledge in programming or coding and provides a new vision of musical composition through programming and computer sciences.

It provides a wide number of modules to generate or process sound. Those modules can be pre-defined or user-defined, they allow to realize a lot of algorithmic computations over structures to express musical ideas. The result of an OpenMusic program can be displayed in common notation such as the MIDI notation or in an audio format. OpenMusic and its users also provide extensions for processing music with other functionalities such as the application of fractals to music, sound synthesis and constraint programming.

The figure 2.2 illustrates an example of a program developed in OpenMusic. This program computes an interpolation from one chord to another. Both boxes at the top of the figures represent the starting chord (on the left) and the final chord (on the right). They are passed to the function "interpolation" that takes two others parameters. The first parameter represents the desired number of steps between the beginning and the end point and the second one indicates a specific parameter, the curve number.

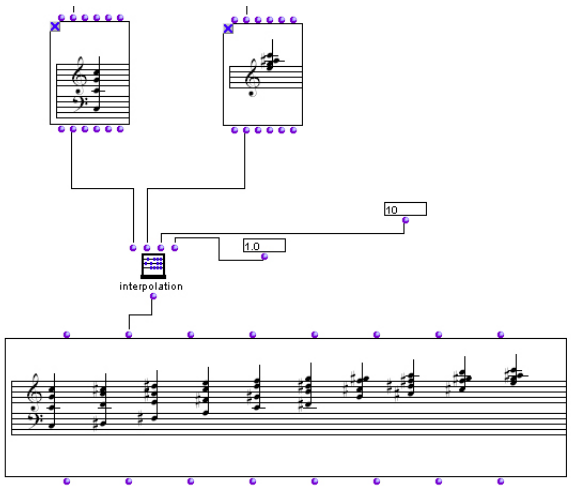


Figure 2.2: Realization of an interpolation between two chords in OpenMusic

This example shows the system of visual programming. By connecting object boxes and functions together, it creates a visual solution that can be listened.
OpenMusic provides the creation of complex and high level programs to express the ideas of the compositors.

[6]Lisp is a programming language based on list processing. This language is imperative and functional [6]

2.2.2 Gecode

Gecode[4] is an open source C++ toolkit created in 2005, it is a software library oriented on the resolution of constraint satisfaction problems. This toolkit provides a solver for constraint programming, it is easily modular and extensible. It has been designed to be « open, comprehensive, efficient, documented, free, portable, parallel and tested ».
Besides theoretical notions of constraint programming such as propagators, branching, variables,... Geocode also provides the notion of Space. This concept is important throughout this thesis because it will not only be part of a coding solution but space will also be included into the musical aspect of the solution. The given definition of a space is, *« A space is a home to the variables, propagators (implementations of constraints), branchers (implementations of branching, describing the search tree's shape, also known as labellings), and – possibly – an order determining a best solution during search »* [7]. The space can be compared to an object with the particularity that it is an object specific to the constraint programming. It always represents a complete model and several functions, specific to the Space type, can be executed in order to perform the current search or to explore new possibilities and new branching in the search tree.

2.2.3 SWIG

SWIG[16] is a software generator of wrapper and interfaces to bind, principally, C and C++ to other languages such as Python, Java, Ruby, and a lot of other languages among which Lisp. This software uses the C/C++ header files to generate wrappers that allow C/C++ function to be reachable from other languages. SWIG eases the interfacing without extra layer of interface languages, the only thing that is needed is the program that you want to wrap and a simple interface script to be able to use functions of C/C++ into another language.

SWIG interfaces a program wrote in C/C++ to a targeted language. Thanks to that, this language can execute C/C++ functions and classes to write its own program. The advantage of this technique is to provide the possibility to the targeted language to use C/C++ functionalities and to execute them with C/C++. Even if the function's call is made in an other language, the calculation of this function is done in C/C++.
The interest looking to this master's thesis, is to allow OpenMusic, through Lisp, to use the functions and the mechanisms of calculation of Gecode. Establishing a constraint programming system into Lisp.

2.3 Previous Works

In every project, one of the first thing to do is looking at the state of the art of the topic on which you are going to work, in our case, the adaptation of constraint programming to a visual programming language based on Lisp and OpenMusic.
Projects and theses have already analyzed this topic. Here are some of them:

- In 2001, Charlote Truchet, Carlos Agon and Gérard Assayag proposed an environment for musical constraint solving inside OpenMusic in a published paper[33] for the IRCAM. They brought to light the utility of coupling constraint programming and musical composition into OpenMusic.

[7] The definition comes from the user manual of Gecode [31], at page 13

- Between 2006 and 2008, Camilo Rueda[8] and Mauricio Toro Bermúdez[9] have worked on a first version of an interface between Gecode and Lisp. That's how GeLisp is born. GeLisp is a C++ library that interfaces Gecode functions. These ones will be called from Lisp thanks to SWIG and CFFI[10]

- After that, in 2010, Sascha Van Cauwelaert[11] also worked on the updating and improvement of GeLisp during his master thesis and his doctorate (during which he published some papers on the topic that will be presented later). Furthermore, he has proved the utility of using relational variables[12] to represent musical variables. His work is the most recent and the most documented on GeLisp today. It has been a basis for the notions as well as for the coding part of this master's thesis.

My project should be considered as a continuing process to improve this topic.

[8]Professor at the faculty of engineering at Universidad Javeriana-Cali in Colombia[11]. He worked several time with IRCAM and the music representation team.

[9]Researcher at Universidad Javeriana-Cali in Colombia[12], he worked on GeLisp under the supervision of Camilo Rueda.

[10]CFFI or Common Foreign Function Interface *"is a mechanism by which a program written in one programming language can call routines or make use of services written in another."*[3]

[11]Researcher at the Catholic University of Louvain-la-Neuve[19]. He realized his master's thesis on an improvement of GeLisp [23] supervised by Peter Van Roy. He published some papers[24, 25, 26] on the topic during his PhD.

[12]He did this during his master's thesis [23].

Chapter 3

Goals

Now that some notions and tools have been introduced. Let's clarify some points related to the goal of this project.

3.1 Why adapting constraint programming to OpenMusic

Constraint programming facilitates the search of a solution to a particular problem. Most of the time, the reason of its use is either the complexity of problems or the need of focusing on the definition of the problem more than on the calculation of its solution. Both are useful in the case of constraint programming.

Automation of computing a solution

The goal of OpenMusic is to provide one of the best tool to musician to compose musical creation. As explained before, the musical composition aims to design a score from ideas. This process can be divided in two parts: the modelling of the ideas and the computation of them. Adding the Gecode constraint programming system to OpenMusic reduces the task of the compositor for the first part of the process. It enables him to focus on the conception of the ideas and leaves the computation phase to the software.
For example, if the user wants to create a series of chords where each chords is composed of, at least, one note of the previous chord[1]. It won't be difficult for him to calculate a solution for a series of 10 chords. However, to apply this process to 100 of chords could take him a lot of time. In this case, the constraint programming can simplify his search by computing a solution in a few seconds.

Higher level calculation

Furthermore, when he wants to manipulate a lot of data, the complexity to provide a solution without the constraint programming will quickly be time consuming. While using this system will take significantly less time.
An example of that problem is the composition of a rhythmic canon on several tracks[2]. With a number of tracks playing simultaneously, we want no notes belonging to different tracks to be played at the same time. Finding a solution for that kind of problem might be

[1]This problem has been developed at page 12 of reference [33]
[2]This problem has been developed at page 13 in reference [33]

difficult for small amount of tracks but if the user works with a lot of tracks, the problem becomes much more complex.

3.2 State of the art

This section focus on the state of the art of integrating constraint programming into OpenMusic. A broader vision of the progress on the use of constraint programming for musical composition is given into the papers of Torsten Anders[3].

Using relational variables

Sascha Van Cauwelaert has demonstrated in his master's thesis, the utility of using relations to represent musical objects as well as relational variables for musical variables in musical constraint programming. As musical object such as note, chord, score are composed of parameters defining them, the relational variables allow to keep all those informations into a bundle while using classical variables do not provide this consistency. However, it is this consistency that provides a high level of informations to the objects. Thus, musical objects can now be represented as a tuple of musical parameters that might look like this:

$$\text{Tuple} = (\text{pitch,duration,velocity,tone color, instrument, ...})$$

It offers a view closer to the reality of the music. And it simplifies the modelization of problems in constraint programming. Because instead of using a lot of variables that aren't linked together, the user has only one variable to constraint. Furthermore, it gives results that are closer to the reality.

Work in peer

At this stage, the modelling of musical constraint problem was only possible by importing into OpenMusic a Lisp script, containing the whole definition of the problem. Thus a compositor had to either know Lisp and constraint programming or ask a software developer to model his problem. Then, modify, at least, the values of the decision variables implied to directly modify the code.
In the figure3.1a, we can see the realization of the all-intervals problem from the user interface. In it, there are no possibilities to manipulate the input to the problem. Furthermore, the script returns a unique solution to the problem.

3.3 Objectives and activities

Before going further into the analysis, let's clarify the objectives I had and the activities I performed, looking to the tools and the previous works.

Updates

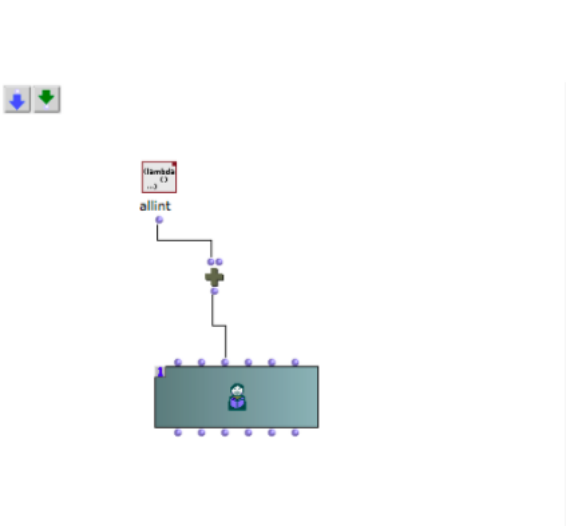
At the beginning of my work, GeLisp was using an outdated version of Gecode. The first objective was then to update the GeLisp's version. The first activities I have done were then to update the GeLisp's version because it was using an outdated version of Gecode.

[3] In his thesis [20] and his paper [21].

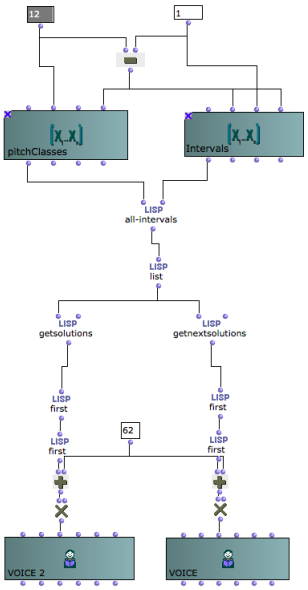
Once it was done, I then added new constraint programming functionalities in the GeLisp library, allowing to obtain more results and to use a greater variety of techniques.

Interface

As shown in the figure 3.1a, the graphic interface for the constraint programming system into OpenMusic is not easy to manipulate for a basic user. Thus, I have worked to create a simple user interface allowing to manipulate the variables and the constraints to model a musical problem. This interface also had to include the new functionalities added to GeLisp. Furthermore, the integration of constraint programming implies some aspect that must be taking into account. For example, the time for searching the solution of a complex problem may be long. Thus, during this search, the user must be able to use the software without any freeze of it. Otherwise, the possibility for the user to compare and obtain different solutions to his problem is also important and must be easily reachable by him.



(a) Old interface



(b) New interface

Figure 3.1: Comparison between the old and new user's interface.

In the new interface, the definition of variables and creation of constraints appear clearly for the user. Each part belonging to this interface will be detailed in the next sections.

Application of relational constraint programming into OpenMusic's interface

Beside classical variable types provides by Gecode, a new one has emerged, the relational variable. As explained in the last section, Sascha has shown the utility of using relational constraint programming. In this thesis, I try to use relational system to create global musical variables expressing musical object such as the notes, the chords. These variables would be consistent with OpenMusic's concepts and easy to manipulate. Furthermore,

the set of constraints applicable to relational variable is not the same as the set of general constraint programming[4], then they also had to be imported.

User involvement

As the user will be able to manipulate and create variables and constraints, he will be able to model problem and to add new constraints. Thus, by making the creation of new constraint as simple as possible, it will encourage users to create new ones. Then, with their help, it will be possible to develop a sharing library of musical constraint reachable by any user and that could be improve by any of them. This could lead to a great tool for compositor.

[4]This one using integers, boolean and sets as variables.

Chapter 4

Theoretical Aspects

4.1 Constraint Programming and its contribution

4.1.1 Links between constraint programming and musical composition

A parallelism between musical composition and constraint programming can easily be done because they have common characteristics: The process of modelling through harmonic, rhythmic, formal or instrumental rules found in music composition is based on the same idea as the constraint programming notion. Both constraint an object. This one takes different forms. In constraint programming it is a variable while in composition it is a musical object, more precisely a score. They can contain a set of values or notes lying between an empty set or a set with all possibilities. To represent an idea or finding a solution, the user puts together rules or constraints until they match at best what they want.
In this part, I will explain the similarities and the complementarities that exist between both concepts.

Similarities

There are principally two similarities between the concepts of constraint programming and musical composition:

Correlation between variables and musical objects: a musical object for the compositor is the same as a variable for the developer, in the way that each of them is defined by a domain, composed on one hand of notes, chords, rhythmic,... and on the other hand of integers or boolean, most of the time. At the beginning, the variable as the partition might be everything and nothing at the same time, the values will only be known after a process of composition for the musician or a process of computation in constraint programming.

Programming constraints and musical rules: The concept of constraints, that is the principal pillar of the constraint programming thought pattern, expresses the same idea as the rules gouverning the musical composition. They both realize a restriction on the domain of their own variables. While the compositor restricts the partition to a desired rhythmic style or a particular set of harmonics, the developer restrict its variables to fit particular constraints aiming to solve a problem.

Complementarities

The constraints programming (also called CP) is a way of thinking that has been created to solve real life problem. It isn't linked to a particular field of activity. It aims to represent usual problems and try to find solutions to them in the shortest time possible. On the opposite, musical composition is attached to the musical domain. While a clear contribution of the constraint programming to the musical domain gradually comes by, the musical domain can also help constraint programming to improve.

- The first direction between these concepts is from CP to musical composition. By modelling musical problems with constraints, compositors can automate computational process to find solutions and work with a higher number o f data to realize more complex musical problem.
- On the other side, application of CP to musical composition will lead to rethink the computation phase. Indeed, the search of a musical result is not the same as search for mathematical problem. Then the ways of searching a solution will be different in order to achieve more efficient solution. These new calculation approaches could lead to new constraint programming concepts that could open new perspectives.

4.2 Variable's Representation

As explained in the previous section, OpenMusic has musical objects that must be translated into constraint programming objects, using variables available from the constraint programming. To set the musical objects, we use the definition of the musical objects provided by OpenMusic.
There are several types of variables in constraint programming: the variables provided by Gecode that we will call the classic variables and those coming from the extension of Gecode to the relational constraint programming developed by Gustavo Gutierrez.

4.2.1 Classic variables and their representation

he classic variables from Gecode that can be used for OpenMusic are the integer, the integer list, the boolean and the boolean list . For example, an integer list variable can represent the list of desired notes that we want or, in the case of the all-intervals problem, they can be intervals or pitch classes. They are efficient but a bit low-level to represent musical variables under all their aspects, the disadvantage they have against the relational variable is that they do not express a relation between data. They can only represent an object with a unique information while the relational variables provide a system of tuples offering more possibilities. Both of them are used in this project, however we will see that the second ones are more efficient to represent musical variables.

4.2.2 Relations and variables

This subsection summarizes the theory of relations and relational variables based on the work of Gustavo Gutierrez[1].

[1]Searcher on the topic of relational constraint programming [27]

The definition of a relation is: *"A relation is a set of tuples. A tuple t is a bijection from a set of attributes A to a set V of values. $t(A_i) \in V$ is the value of the attribute A_i. All the tuples in a relation have the same set of attributes. Every attribute A_i has an associated domain with the set of possible values that the attribute can take."*[2]. From this point, he created a variable corresponding, called the RelVar. This new variable is initialized with two relations, a lower bound relation and an upper bound (mostly of the time, the lower bound is the empty relation, it is equal to the schema of the relation, and the upper bound is the relation itself with all the tuples that populate it). Furthermore, he extended Gecode to include this new variable. This last element is also one of the reasons to use relation.

4.2.3 Musical Representation through relation

The idea of representing musical objects through mathematical relations can be found in the master's thesis of Sascha Van Cauwelaert and before him, in the works of Torsten Anders[3]. Musical object are composed of several attributes. The idea through the relation is to group and link all these attributes in one object. Then instead to separate them, they are all linked into a relation as tuples. A tuple allows to bind the value of an attribute with the values of the other. For example, a music chord that has three notes is represented by a 3-tuples where each one binds a pitch with its velocity, its duration,...

4.2.4 Musical variables

Now that the relation, relation variable and the representation of music objects have been introduced, we can look for the effective instantiation of RelVar to generate musical variables.
During a part of his doctorat's thesis, Sascha has continued to developed the idea of a musical variables and created two kind of them, *Music Bundle* and *Music Bundle Set*[4].
The first one represents relational variables with, most of the time, two attributes, pitch and onset. The second variable is simply a set of the first one. The latter is efficient enough to define a score.

Based on this background, I started to think about more global musical variables that would fit to musical object used in OpenMusic. This thesis focuses on harmonic object and would not deal with an application of relational variables to rhythmic objects. This choice is made for some reasons. The first one is that those variables are closed to the notion of score. And another important reason concerns the OpenMusic's representation of rhythmic objects that offers a highly flexible notation that would be challenging to get in a constraint problem.
The principal harmonic objects are the note, the chord and the sequence of chords. Their equivalent variables are :

note variable $\langle$*midic-dom, velocity-dom, duration-dom*$\rangle$
chord variable $\langle$*midic-dom, velocity-dom, offset-dom, duration-dom*$\rangle$

[2]Definition of a relation from the doctorat thesis of Gustavo Gutierrez: *"Constraint programming on relations: Theory, practice and implementation"*, page 14

[3]Professor of music technology at University of Bedfordshire (England), he studied composition and is also a searcher on the computational modelling of composition and music theories[17].

[4]A more detailed description of his concept and implementation can be found in his papers[26]

chords-sequence variable ⟨midic-dom, onset-dom, velocity-dom, offset-dom, duration-dom⟩

Each parameter corresponds to the domain of these attributes. These variables extend the RelVar object and are subject to the same constraints.

4.3 Possibilities given by this thesis

Besides the systems of classical and relational constraint programming. This project aims to provide useful functionalities and ideas in order to improve the the use of constraint programming into OpenMusic.

4.3.1 Reflection about musical constraints

In the last sub-section, I introduced a little explanation about the constraints and their use. The principal set of constraints that exists for musical variables is the same as the one for relational variables. This one does not provide constraints for musical problems except those present in the examples. The reason for that is the intention to first develop a system of constraints programming that works in OpenMusic and then, to initiate a constraint sharing between the compositors.

Creation of new constraints

Objective is to provide a tool with which the user can create new constraints for solving musical problems starting from basic constraints. This creation can be done at different layers:

- A first layer is the basic OpenMusic interface, in which you can use constraints and bundle them to create a patch solving a musical rules. This stage is quite basic and can only use existing object into OpenMusic.
- A second layer involves some knowledge about Lisp. The user can create, by himself, new functions that constraint variable(s) and import them into OpenMusic. This approach allows to define more complex problem that would be difficult to develop in OpenMusic directly.
- The last layer is the developer layer. For this stage, the user needs Lisp and C++/Gecode/Relational knowledge to implement new constraints applicable for musical variables or widely for RelVar. He can also work on the development of new branching to fit to musical domain. And, in a wide sense, develop new concept(s) or new variable(s) based on Gecode before to translate them into GeLisp and finally into OpenMusic.

Library for constraints

According to the use and the development of constraints programming and by using this community of users, it would be interesting for OpenMusic to provide a library of constraints that could be filled in and reachable by any of its users. In this way, this library would be sharing a lot of musical constraint satisfaction problem solutions, each one following various approaches. To achieve this, the system of constraint programming underlying this library has to always be functional and on stable basis.

4.3.2 Search oriented to musical solutions

Before to look after search oriented to musical solutions, let's focus on "how the search is made in constraint programming" and "what are the possibility of it".

This project brings new concepts for searching a solution. Constraint programming uses several tools to orient the search of a solution like consistency level for constraints, branching functions and solvers. Each combination of them conducts the search through a different way to the solution. This subsection talks about these particularities and how they are expressed into the OpenMusic's interface.

Consistency level of a constraint

Each constraint use a propagator type. A propagator is a mechanism that propagates the effect of a variable assignment. For example, in the constraint *distinct*, when a variable is bound to a value, the propagator removes this value from the other variables to ensure that all variables have distinct values. The goal of consistency levels is to control which propagator is chosen by the constraint. Gecode provides 4 levels of consistency. They are in the table B.1[5]

Branching functions

In constraint programming, the search of a solution can be seen as a reverse tree. It starts from a root state and grows by branching to other possible states. At each branching, two selections are made, one on the next variable to assign and the second for the next value to select. Gecode provides several branching factors. The variable selection is composed of 26 possibilities and the value selection is composed of 15 possibilities [6]. Each possible combination leads to a different behaviour and then to a different order of the solutions. Today, this project only provides one combination of branching factorC. The reason of this unique possibility is that the first goal of this project is to create a functional system before improving it[7].

Solvers

A solver is a mechanism that searches the solutions according to a specific approach. Gecode provides two kinds of solvers, one based on the depth-first search (DFS) algorithm and the other based on the branch-and-bound search (BAB)[8]. The previous version of GeLisp provided only the DFS, now both are available. Furthermore, they can be executed on the same problem in parallel[9]. They can be used with the function *getDFSSolutions*, *getNextDFSSolutions* and *getBABSolutions*, *getNextBABSolutions* present in the OpenMusic's interface.

[5] More details about the different levels can be found at the page 56 of the manual [31]

[6] The different possibilities can be found in the figure 8.1 (page 117) and the figure 8.2 (page 118) of [31].

[7] A way to add branching factor can be found in the API description in the annex C

[8] A description of these tools can be found in [31]

[9] In this case "in parallel" means that the user has not to redefine the problem to get the solutions of the other solver. It does not reference the parallel execution of threads.

Musical search

The solver is the part of the constraints programming where all calculations to fit a maximum of variables and to return an efficient solution take part.
The solver used by Gecode has a mathematical approach of the solutions. For example, when defining a problem, you can state which approach you want: if you want the minimum of a certain variable or the maximum, if you want a random solution,... There exists several techniques that orient the solution in the desired way. But for now, Gecode doesn't provide any musical approach. To provide a solution that matches with the musical domain, the solver must have an approach that can be significant for a compositor. We have thought about different aspects that might be interesting to have for a compositor:

- A solver oriented for creation of melody.
- Another solver led by the desire of creating an accompaniment score and providing a series of chords.
- Another approach would be the diversity of the score, by never using the same pitch twice in a row.

A lot of ideas can come out but, unlike the constraints, the development of new solver is not easy and can not be done by anyone without a knowledge in constraint programming and relational algebra and variables.

Today, as the realization of this functionality involves time and reflection to correctly define and to develop the solution, it has not been possible to develop such a solver in addition to the functionalities already implemented.

4.3.3 Musical constraint programming inside a software

The search of a solution amongst the variables and the application of a constraint program involves some particularities that have to be taken into account in order to ensure a comfortable use of the host software, in our case OpenMusic.

Execution of the solver

As we have seen in the last sub-section, the solver is the part of the constraint programming that realizes all the calculation to find a solution. This step can take a huge amount of time if it has to deal with a lot of data.
But, meanwhile, OpenMusic must be usable and cannot crash because of the constraint programming part. Thus, to ensure that, the solver is executed in another thread than the one used by OpenMusic.

The use of a thread devoted to the solver has also advantages for the users. It provides them the possibility to choose when they want the search to stop. Hence, if they want to spend time to find the best solution, they can let the solver run until it stops by itself. But if they want a quick solution that doesn't have to be the best but that respects the constraints, they can stop it themselves at any time.

Chapter 5

Musical Aspect

This chapter explains the musical aspect of the project. It develops what is a musical concept and why it is relevant in this adaptation of constraint programming to the domain of music.

Let's remember some things, this thesis aims to help compositors to easily create music score. It tries to reduce the distance between the abstract ideas of the compositor and the effective writing of these ideas into a set of notes and chords. These abstract ideas are the starting point of the musical composition. By combining them together, the compositor creates a score that fit all or, at least, some of them. Before, we have seen the notion of music rules that fits the constraint notion in constraint programming, now there is the notion of musical concept. It can be seen as a set of music rules. The abstract ideas are themselfves composed of concepts.

5.1 Musical concepts

5.1.1 Some basic concepts

Let's present in more details some basic concepts of music and their representation into OpenMusic.
Those concepts are harmonic objects: the note, the chords and the sequence of chords (this latter is the closest to the notion of score). They are composed of parameters. Some of them have been defined in the section 2.1.1:the pitch, the duration and the velocity. In addition to these:

- Offset: The term of offset is not particular to the musical domain. In its common sense it expresses a difference between two or more things. In our case, it describes a delay in a chord between the first note and the others. Its values is represented in milliseconds.
- Onset: In the same way, it is not a term close to the musical domain. The onset represents the difference between the beginning of the score and the position of the note/chord. For example, if the onset of a note is set to the value 1000, it means that this note will be played 1000 milliseconds after the beginning of the score.

Now, let's see how OpenMusic represents them:

- Note:

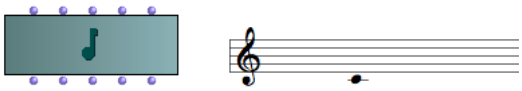


Figure 5.1: Note object box and its representation.

In OpenMusic, a note object is composed of 5 attributes: self[1], pitch, duration, velocity and MIDI channel.

- Chord:

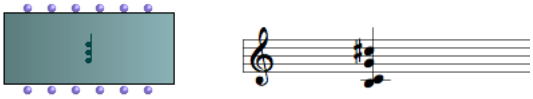


Figure 5.2: Chord object box and its representation.

A chord object is composed of 6 attributes: self, list of pitches, list of velocities, list of offsets, list of durations and list of MIDI channels.

- Sequence of chords (also known as Chord-Seq):

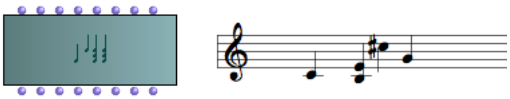


Figure 5.3: Chord sequence object box and its representation.

A chord sequence object is composed of 7 attributes: self, list of pitches, list of onsets, list of durations, list of velocities, list of offsets and list of MIDI channels. To initialize a chord sequence the attribute self can take a list of chords or even a unique chord. The values of these chords will be passed to the chord-seq.

5.1.2 Correlation between basic concepts and variables

As explained in the last chapter, to keep a logic between the musical objects of OpenMusic and the variables, each object presented before has its double as variable.

- NoteVar.

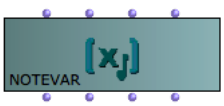


Figure 5.4: Box representing a variable note.

[1]The self attribute represent the object itself. As input, it could be bound to another object of the same type and as output, it allows to export the object to function or into another object.

Its is composed of a domain for each of its attributes: pitch domain, duration domain and velocity domain.

- ChordVar

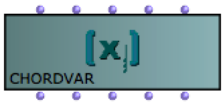


Figure 5.5: Box representing a variable chord.

Its is composed of a domain for each of its attributes: pitch domain, duration domain, velocity domain and offset domain.

- ChordSeqVar

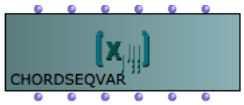


Figure 5.6: Box representing a variable chord sequence.

Its is composed of domain for each of its attributes: pitch domain, duration domain, velocity domain, offset domain and onset domain.

Today, there is no MIDI channel domain.

5.2 Musical rules versus constraints

5.2.1 Rules and their influences

To compose a musical score, a compositor applies musical rules to give a color, to spread an ambiance or an emotion through his score. For example, studies[2] have shown that a music played in major scales provide a feeling of happiness or brightness while minor scales tend to produce more sadness or nostalgic feeling. That's what musical rules are about, shape the music to express ideas and feelings.

The example "All-intervals" use the rule of interval. The main intention of this rule is to express the relations between the notes. For example a major scale has a sequence of intervals that is expressed like: tone, tone, semitone, tone, tone, tone, semitone. These relations are represented in the figure below where all red curves represent an interval of a tone and a red broken line a semitone. The notion of rules in music and the notions of constraint in constraints programming are quite the same but expressed for different domains.

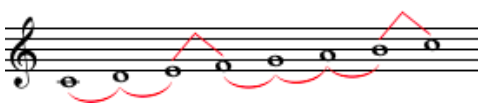


Figure 5.7: Pattern of a major scale.[3]

[2]More can be read about them in the references [10], [28], [29], [32] and [15].

The notion of rules in music and the notions of constraint in constraints programming are quite the same but expressed for different domains. This relation between rules and constraint is illustrated in the constraints of the "all-intervals" rule defined in the section 2.1.3. It shows that a rule can easily be transcribed into a set of constraints.

5.3 Integrated interface to OpenMusic

Now that the variables and the constraints has been introduced, let's take a look at a basic interface in its totality:

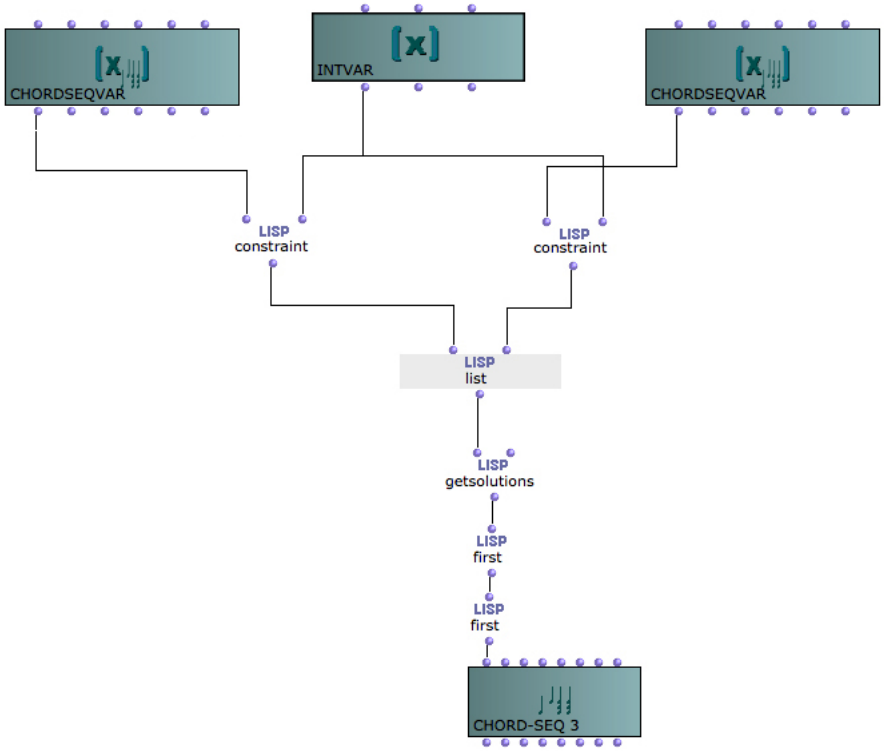


Figure 5.8: Classical user view of the new interface for constraint programming into OpenMusic.

This figure can be divided into 4 parts: the variables, the constraints, the calculation phase and the expression of the solution. In the next subsections, each of them will be quickly explained.

5.3.1 The variables

The variables have been widely explained previously. There are 7 kinds of variables available into the OpenMusic interface at this day: the IntVar, the IntVarList, the BoolVar, the BoolVarList, the NoteVar, the ChordVar and the ChordSeqVar.

[3]Figure source [7].

They allow to model decision variables and the solutions for each of them will be expressed in the result part.

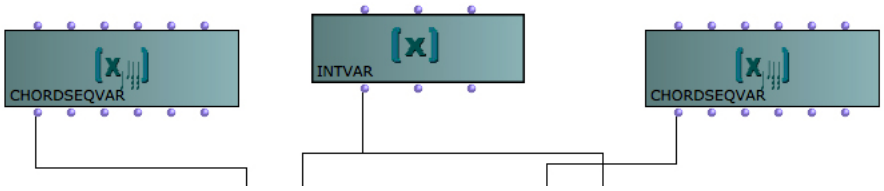


Figure 5.9: Variables boxes.

Each variable is linked to the constraints to which it is subject to.

5.3.2 The constraint functions

The constraints are represented in the form of functions returning a constraint object. A constraint object is a bundle containing the variables and a function that applies the constraint over the variables. A user can add as many constraints as he wants. Once they are all linked to their variables, they should be put into a list for the calculation phase.

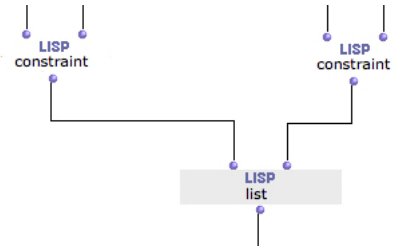


Figure 5.10: Constraint functions.

The constraints and their functionalities are more detailed in the section 5.4.

5.3.3 The computation phase

The function "getSolutions" is the place where the magic comes. This function receives the list of constraints initialized before and the number of desired solutions. Then it realizes all the computation and returns a list containing all solutions with, for each of them, the specific instantiation of each variable.

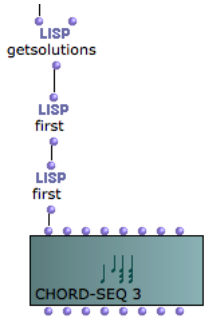


Figure 5.11: Calculation function and expression of the result.

Solutions appears in the non-variable form of the variables:

- Integers for IntVar or a list of intergers for IntVarList.
- Boolean for BoolVar or a list of boolean for BoolVarList.
- OpenMusic Note object for the NoteVar and the same logic for ChordVar and ChordSeqVar that are represented by Chord and Chord-Seq.

5.4 Composition of new concepts

An important point of this interface that has been introduced before is the constraints and the ability to create new constraints. A constraint is represented by a function that creates a constraint bundle as explained in section 5.3.2.
Constraint functions are structured in two pieces:

- A first part during which a lambda function[4] is created. This lambda function defines a musical rule by applying constraints on the variables passed in argument of the function.
- The second part creates the constraint object including the variables subject to the constraint and a function defining the musical constraint.

```
;;;;;;; constraint definition
(defmethod constraint ((CSself ChordSeqVar) (Iself IntVar))
   ;definition of the constraint.
        (setf rule
        (lambda (x y z) (gelisp-constraint x y z))
        )
  ;creation of the constraint object
        (make-instance 'Constraint-2 :var1 CSself :var2 Iself :
            constraint rule)
)
```

Any new constraint created by a user has to respect this format in order to be correctly executed. With all the variables and this shape of constraint, it is easy for users to implement their own musical rules through the constraints because they can create constraints that will be part of a constraint programming system without having to take care

[4]Also called anonymous function, it represent a function definition not bound to an identifier.

about other specific details of the constraint programming. This simplicity to create new constraints is a major interest of this interface.

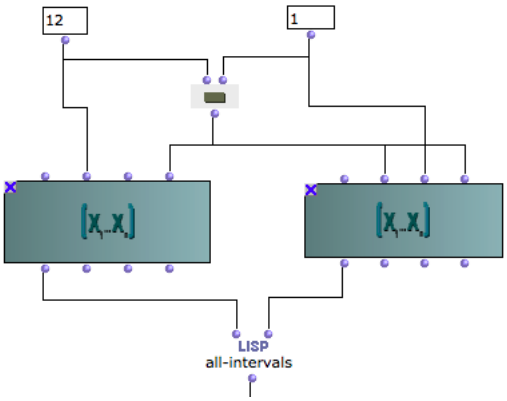


Figure 5.12: Interface for the all-intervals constraint into OpenMusic.

The example of "all-intervals" has been created in this way. This rule uses not only decision variables but also auxiliary variables and a structure that would be too complex to be directly represented into OpenMusic. For these reasons, it has been defined into a Lisp function that takes two arguments, the decision variables *pitchClasses* and *intervals*. The following figure illustrates the OpenMusic interface for the "all-intervals" problem. The Lisp code defining this rule can be found in the annex D.2.2.

Chapter 6

Program

This chapter will explain the underlying C++ interface that supports the functionalities. It will be divided in three parts: the new version of GeLisp due to the update, the implementation of the musical variables and a library providing a consistency to express the constraint programming results into OpenMusic.

6.1 Update of GeLisp

This first part will present the binding between Gecode and Lisp and the GeLisp library.

6.1.1 Binding and calculation

On one side, there is Gecode, an efficient constraints programming system, in which the solver will execute the research for a solution to a constraint satisfaction problem. And on the other side, Lisp, a functional language, used for implementing the OpenMusic visual-language oriented for music composition. For the purpose of integrating constraint programming functionality to OM, a binding allowing Lisp to call CP functions is needed. This one is called GeLisp.
With GeLisp, any Lisp code is able to execute a constraint programs modelled in Lisp. While the modelling takes place in Lisp, the calculation stay in the Gecode part. It generates the following type of sequence diagram:

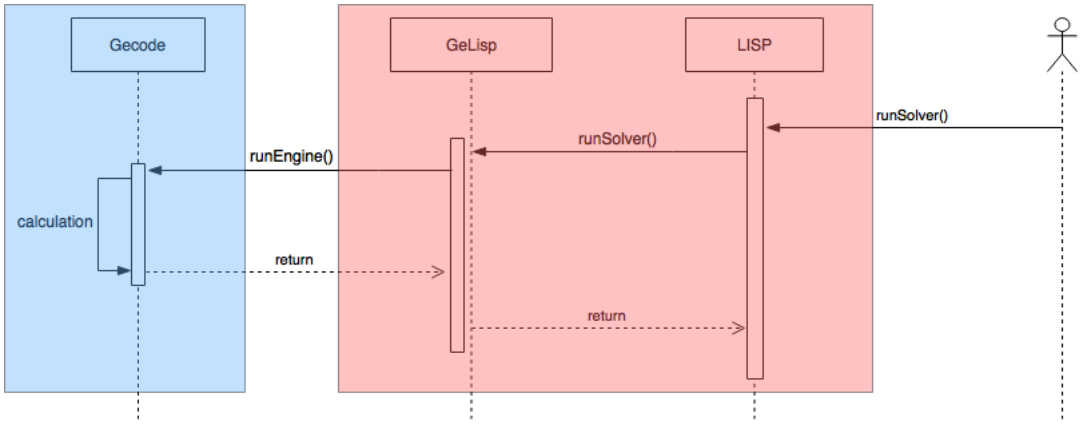


Figure 6.1: Sequence diagram of the call and the execution of the function "runSolver".

On this diagram, we can see the calculation phase (in blue) and the calling phase (in red). It illustrates that even if the modelling is in Lisp, the calculation is realized in Gecode. This separation is the strength of GeLisp, it allow a Lisp program to use the performances provide by a C++/Gecode program without having to code Gecode in Lisp.

6.1.2 GeLisp3

GeLisp is an interface created in 2006 by Camilo Rueda and Mauricio Toro Bermudez, updated by Sascha Van Cauwelaert in 2010. As Gecode is still quite young, versions of GeLisp suffer of Gecode's evolution. Since 2010 and the version of Sascha, Gecode basics functions are almost no more subject to implementation modifications.
I started from the last source of C++ GeLisp library, from Sascha.

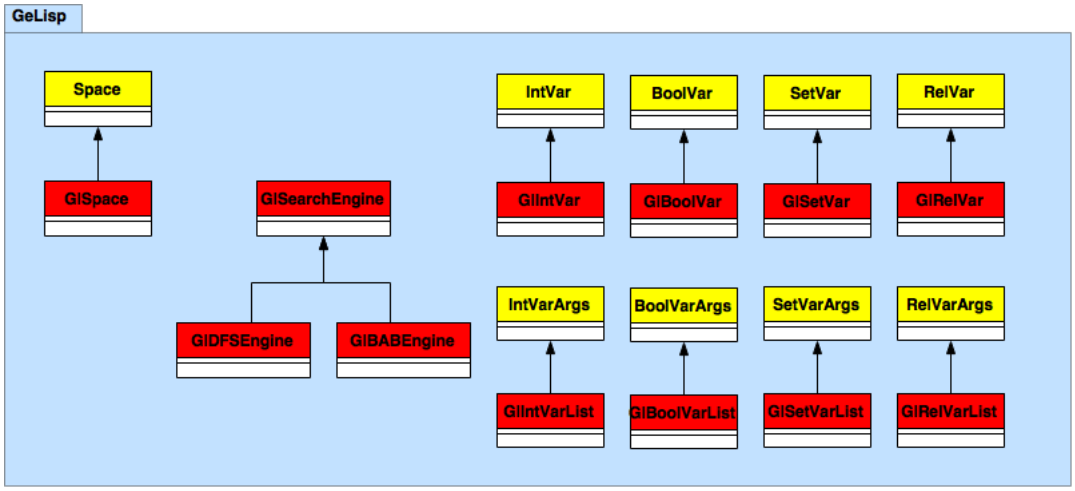


Figure 6.2: Classes diagram of the GeLisp3 library.

After updating the code to render it reusable, I inserted other functionalities to improve the possibilities of the library:

- A branch-and-bound solver. The previous version of GeLisp provided only a DFS[1] solver.
- A system of next solution. This allows the user to get the next solution of the problem if the first doesn't satisfy him. To realize this functionality, I added three classes: GlSearchEngine, GlDFSEngine and GlBABEngine. These are based on the first version of GeLisp that already provided them. Today, each space has a search engine from which it can calculate the next solution.

6.2 Implementation of Musical Variables

Variables oriented to music constraints problems are referenced into the GOM library. As the notion of musical variables has been explained in the previous chapter, this section quickly covers the implementation part of these objects.

[1] Depth-First-Search.

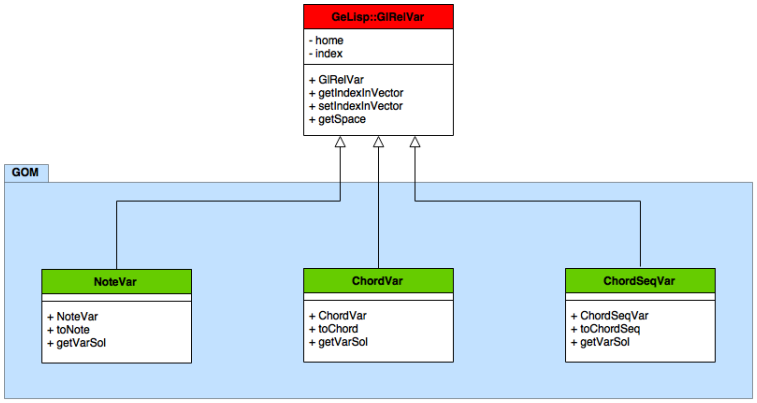


Figure 6.3: Classes diagram of the GOM library.

The musical variables extend *GlRelVar* from the GeLisp library. Each of them is composed of three functions:

- NoteVar, ChordVar, ChordSeqVar are constructors of musical variables. They create a relational variable with the relations initialize with the pitch, velocity, duration, onset and offset domain as explain previously.
- toNote, toChord, toChordSeq translate the variables into musical object implements in C++, those objects are explained in the next section. They are the first step to the representation of the result into OM object.
- getVarSol is the function that retrieves the solutions of a particular musical variables and returns a vector of the musical object(s) by using the toNote/toChord/toChordSeq function from the solution vector passed as parameter and calculated in GeLisp.

6.3 Dealing with C++ and OM

The binding between Gecode and Lisp represents a part of the conversion of objects between C++ and OM. Another part is the conversion between C++ objects and OM objects. Whereas GOM calculates the solutions for musical variable, it needs a conversion phase to express the result into OpenMusic. To ensure that, and generate a cohesion between C++ and OM, musical objects of this latter I have implemented into objects of the first.
While the binding is dealing with objects and basic types only, a conversion between other kinds of data is needed. Lisp is a language based on lists whereas C++ uses preferentially arrays and vectors. Then, in order to transform basic type structures from a language to another, a first part of the conversion phase is to ensure a conversion from Lisp lists to C++ arrays. It has been set up thanks to the CFFI library. Now that basic types' structures can be exchanged between language, let's see the library used for conversion from OM objects to C++ objects and vice versa.

6.3.1 OM++

This library represents musical object from OpenMusic as C++ object. The structure is barely the same as the one provided in OM.

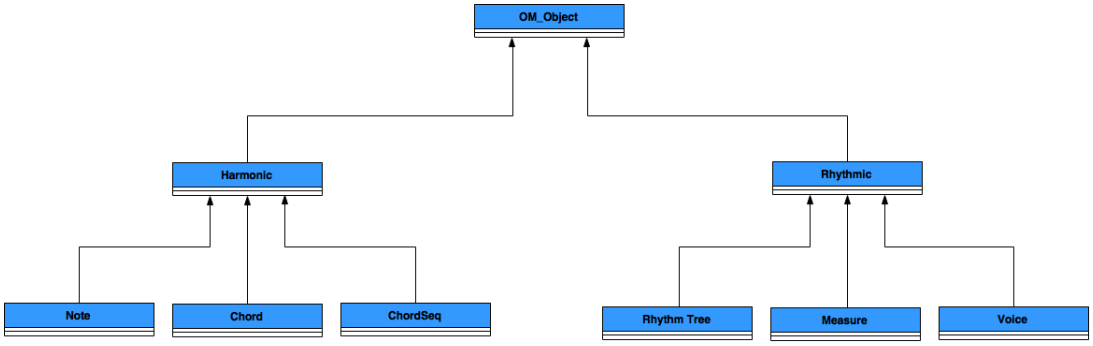


Figure 6.4: Classes diagram of the OM++ library.

This library is composed of a main class OM_ Object that is extended by the Harmonic and the Rhythmic classes. They both are extended by classes representing OpenMusic. These last ones are object classes composed of the same attribute than their equivalence in OpenMusic. It has been set up for two reasons. First, to ensure a cohesion between C++ and OpenMusic, because the result of the musical variables are expressed in the form of musical object implemented in this library. And secondly, to simplify the translation of a C++ musical object to an OpenMusic object because they are using the same attributes.

6.4 From C to Lisp

Since all the needed C++ library are available, they have to be passed in Lisp code to be usable into OpenMusic. To do so, I used SWIG that has been introduced in the presentation section[2]. An explanation of its use and what it provides in this particular topic is given in the reference [23].

6.5 Lisp code

With all the Gecode and other C++ library reachable from Lisp, it is time to focus on how Lisp and OpenMusic are dealing to make all this work together. There are 3 parts to the Lisp code: The first contains all functions allowing to pass data from C++ to Lisp and back out, the second implements the music variables and the last part is caring about the constraints.

6.5.1 Conversions

C++ and Lisp being different languages, it is normal to have some conversions that needs to be done between the data that are travelling from one to another. Two Lisp files are dealing to ensure a good conversion of the informations.

Utils

The first is utils.lisp. This one groups all the functions allowing to correctly convert Lisp list to C++ arrays.

[2]Section 2.2.3.

OM++

The second realizes a more particular conversion. It is focused on the conversion of musical objects. OpenMusic provides several musical object that have a corresponding object in C++. This Lisp file allows to create an OM object based on a C++ object such as Note, Chord, Chord-Seq and so on.

6.5.2 Music Variable

All of the variables provided by Gecode and the relational constraint programming find their match under the form of OpenMusic object in this part. Since the computation of solutions happens in C++, each object contains a pointer to it corresponding C++ object in order to be allow to manipulate data. In this file, there is also all the computation calls and the function reachable into OpenMusic that deal with the creation of a constraint script.

6.5.3 Constraints

Finally, the last part but not least is the one allowing to get and create the constraints. It is divided in two files, the first regrouping all basic constraints and the necessary functions to use them and create new ones. The second is devoted to the addition of new constraints. This latter is the one in which each user can add its own constraints and the basis of the library of constraints developed in previous chapter.

Chapter 7

Evaluation and applications

7.1 Evaluation

This section evaluates the result of the project today looking to the objectives.

7.1.1 Evaluation versus objectives

Creation of a simple user's interface

The main goal of this thesis was to create a user's interface to constraint programming into OpenMusic. The complexity of constraint programming has been hidden to only keep simple notions that a user without programming knowledge can manipulate. Furthermore new functionalities added to GeLisp, such as the "get next solution", provide a tool that can model a huge number of musical problems and even more solutions to these problems. From a constraint programming point of view, a lot of functionalities of Gecode have been interfaced. The different types of variables are reachable through GeLisp and the Integer variables and Boolean variables are represented into the user's interface. About the constraints, a lot of them are part of GeLisp and adding new one is not complex for a developer. This objective seems to be achieved.

User involvement

Another force of this project is to provide a tool that is easy to use and easy to improve by the user of OpenMusic. The user can easily develop problems and create new constraints. The definition of these constraints is not linked to the calculation phase. Thus, it is really easy to share new constraint between users. The section 8.2.1 develops some possibilities that can help this user involvement and improve the quality of the tool.

System possibilities

The system is functional and already provides a lot of functionalities. There is a wide range of possibilities from the constraint programming available in the OpenMusic interface. It goes from the creation of basic variables and constraints to the execution of different solvers on the same problem for all possible solutions. Furthermore, it can easily be improved by defining computation functions with more argument to orient the search of a solution[1].

[1] More can be read about the possible improvements in the section 8.2.1

Use of relational constraint programming

Concerning the use of relational constraint programming, it opens new ways to model musical constraint satisfaction problems and is also part of the user interface. This part is not functional today because there are still some problems to execute a search with relations through the musical variables into OpenMusic. The computation of problems using relational variable works well in C++ but causes trouble into Lisp and OpenMusic. The reason of this problem stands into the complexity of modelling relational problem for generic and simple variables. Some improvements has to be done about this part[2]. However, the other functions of this concept are functional. First, the representation of relational variables through global variables that are the NoteVar, ChordVar and ChordSeqVar. Secondly, the translation of music relational variables into OpenMusic's musical objects.

Solver threads

In the section 4.3.3, I explained the utility to insert the solver into a thread. Today this system does not work. The difficulty was to send back the solution from the thread while keeping a clear user's interface. However this is far away to be an insurmountable problem and with the new possibility to use different solvers for the same problem's model, the tool will provide high performance once the threads will be available. Several approaches can be taken to solve this, the synchronous way as well as the asynchronous and both are implementable in Lisp. Some lack of time are at the basis of this problem.

7.1.2 List of concepts and functions

During this project, concepts and functionalities have been developed. Here is the list of all achievements:

- The update of GeLisp to ensure the maintenance of parts created before the beginning of this project. The maintenance of this kind of project is important to achieve new possibilities because it allows to start on operational bases.
- Addition of constraint programming facilities through new functionalities such as the next solutions or the possibility of choosing which solver to use.
- Facilitate the manipulation for a basic user via new user interface. It goes by the implementation of new functionalities in GeLisp such as the next solutions or the possibility of choosing which solver to use. It also suggests ways to improve the use of solvers through the threads. A clear definition of variables and creation of constraints. The possibility to have several solutions to a problem and to compare them.
- The continuity in the application of relational constraint programming to musical composition through the searches to create global relational variables into OpenMusic is progressing.
- Giving the possibility to the user to create new constraints in an easy way.

[2]The parts that need of improvement are explained in the section 8.1.

7.2 Applications

As explained in the previous section, the computation of the relational part still causes trouble from the user's interface. The example developed in this section uses basic Gecode variables as the relational approach is not yet functional.

7.2.1 All-intervals

All along this paper, we have seen the all-intervals example. Now, here is the presentation of this problem and its solutions through the OpenMusic's new constraint programming interface.

First, let's remind the problem. The goals of this example is to provide a twelve-tone series in which intervals between two successive notes are distinct. To solve this problem, two decision variables and five constraints are necessary:

- pitchClasses: list of integer variables where $0 \leq x_i \leq n-1$ for all $0 \leq i < n$ with n the size of the list.
- intervals: list of integer variables where $1 \leq x_i \leq n-1$ for all $0 \leq i < n-1$ with n the size of pitchClasses.

And the 5 involved constraints:

- $\bigwedge_{1}^{n-1} inversionalEquivalentInterval(pitchClasses_i, pitchClasses_{i+1}, Intervals_i)$
- $distinct(pitchClasses)$
- $distinct(intervals)$
- $pitchClasses_0 = 0$
- $pitchClasses_{n-1} = n/2$

Through the new user's interface, the variables are represented as two boxes of type IntVarList. The arguments of both variables only depends on the size of pitchClasses, twelve in the case of dodecaphonism. This representation of the variables is illustrated in the figure 5.12.
Then, all constraints of the problems has been bundle in the *all-intervals* constraint function. It has been done through as the creation of a new constraint. The implementation of this function is given at the annex D.2.2. Finally the modelling of this problem into the user's interface is given below.

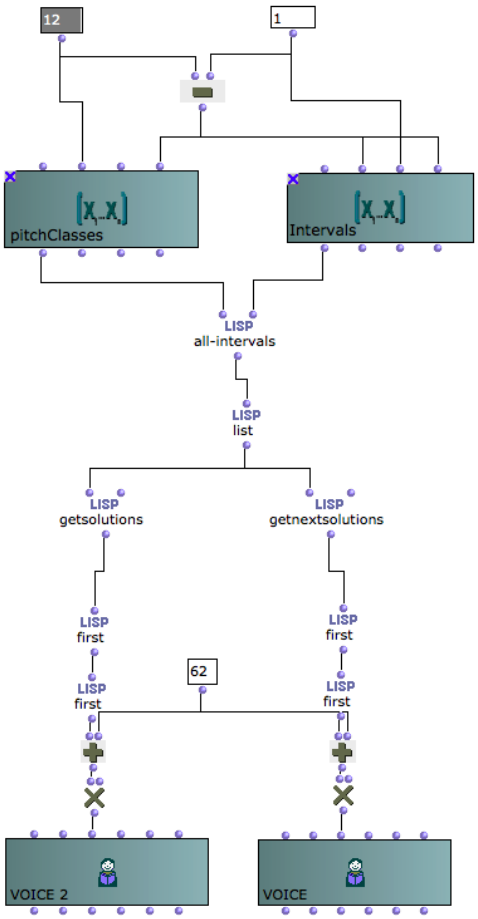


Figure 7.1: Resolution of the all-intervals problem for a dodecaphonic series through the user's interface of OpenMusic.

The output of the all-intervals function is put in a list. This list is passed as argument to the computation function getSolutions[3] that will find the first answer. The figure 7.2a illustrates the answer of this problem. If this result is not satisfying for the user, he can use the nextSolutions function that will return a new result. This new result is illustrated at the figure 7.2b. And the next one after until he gets the most valuable for him.

(a) First DFS solution.

(b) Next DFS solution.

Figure 7.2: Solutions to the all-intervals problem.

The solutions are expressed in a list of integer that can directly be reused. Finally, the solution is adapted to a pitch value around which the score is build.

[3]This function executes a DFS search, it is the basic function. There are two others functions getDFSSolutions and getBABSolutions that are made to be more specific.

Comparaison with previous version

The model presented above is more easy to manipulate for the user. From it, he can modify the problem to get the solutions for another series than the twelve-tone series. Another difference with the previous version is the possibility to get the following solutions. It provides to the user to obtain all existing solution to this problem.

Chapter 8

What's Next

This project has been made in one year. However, there is still so much that can be done for the constraint system of OpenMusic to be complete. This chapter covers several improvements that would expand the possibilities of the system. It is divided into two sections, one referencing to maintenance problems and the other one to technical improvements.

8.1 Maintenance

This first part concern the maintenance. There are mainly two points that should be covered by maintenance:

User's reviews

This project was quite long and there was almost no time for user's reviews and feedbacks. Thus, some issues can not be excluded. In addition, the user's point of view is really important in this kind of project to allow the tool to be improve and close to the user's need.

Tools' updates

As explained before, this project is based on several independent tools that are OpenMusic, Gecode, SWIG and the relations. They have all been subject to updates cause trouble into created libraries.
Since this project is contained into a static library, its functionalities will stay available into OpenMusic whatever the updates. But they might become outdated in terms of performances and new possibilities brought by tools' updates. This paper will not explain how to keep the libraries up-to-date but the textbook for developers is presenting the different updates that the the libraries might face.

8.2 Improvement

As this master's thesis comes as a proof of concept, some focus have been done on several aspects. This section introduces to notions that might be relevant but haven't

been explored yet. It is divided into two sub-sections, the first dealing with improvement on what already exists and the second with new functionalities.

8.2.1 Improvement of existing functionalities

Computation of relational constraint programming

The relational constraint programming take a large part in this paper. Unfortunately, it does not work. The computation still suffers of problem in OpenMusic while it works fine in C++. The problem seems to come from the definition of the relational space[1] but it has not been clearly identify. The use of relation can be a huge improvement for the constraint programming in music domain because the relations are the closest representation of musical object and in addition OpenMusic will be the unique software to use this technique.

Solver's thread

Another problem that has been developed in previous chapter should also be solved. It is the thread of the solvers. The solvers can take time to find a solution for complex problems. During this time, the user should not be force to wait and should be able to continue working with OpenMusic. Furthermore, the addition of threads could give the possibility to the user to choose himself when to stop the search. In this way, he could let the process run if he want the best solution or cut it if he is only looking for a solution even if it is not the best. This improvement is really important to ensure that the constraint programming part of OpenMusic is correctly integrated to the software.

Musical search rather than mathematics search

From a constraint point of view, developing new way of searching solution would be very interesting. For now, the search options used are mathematical but the results wanted for a musical search aren't the same as the mathematical ones. Then looking for options that focus the search on musical concepts might be very interesting. For example, a search that focus on creating a melody or on the opposite, that tries to create only chord. A lot of high level concepts could be explored.

Adding search parameters

The search of solutions could be more sharp by adding new parameters to the get solutions function. At this stage, there are three functions to get solutions. The first[2] is quite basic and only take the variables and constraints as arguments and the two others[3] are there to become more technical and to allow the user to choose the behaviour of the solver. This behaviour might be more detailed. Gecode offers a lot of options to conduct the search that has been introduced in the section 4.3.2. They can be used to improve the solutions.

[1] The relational space contains the attributes that initializes the relation and the possible value of these attributes.

[2] *getSolutions* and *getNextSolutions*

[3] *getDFSSolutions*, *getNextDFSSolutions* and *getBABSolutions*, *getNextBABSolutions*.

Musical relational search

Another interesting improvement is to develop a search focusing on musical aspects. With the help of Gustavo Gutierrez, it is possible to create new option for the solver that would lead to more musical and less mathematical solutions. Some examples have already been explained in the section about the musical interpretation of the solver[4].

8.2.2 New functionalities

Creation of new variables

The developed objects was all of harmonic type. It could be interesting to extend the variable of constraint programming to other types such as the rhythmic objects. This could lead to new applications and new concepts for both, music composition and constraint programming.
Higher level object such as polyphonic or multi-sequence objects (presents in OpenMusic) might be improve with constraint programming. It could be possible to generate an accompaniment score based on a melodic score. A lot of powerful functionalities could be created.

Library of constraints

As explained before, it is easy for a user to create new constraints. Thus, a shared library containing all created constraint that would be easy to update would be a real advantage to develop new musical concept.

Closer to OpenMusic's tools

Improve constraints and variables to use other musical objects. For example, the scale object would be very relevant to define the domain of variables or to constraint a score.

Artificial intelligence is more than constraint programming

In a far more advanced state, we could also imagine higher level possibilities.
For examples, some studies has shown musical correspondence between musics aiming to express the same emotion as the joy often made of major scale, the sadness or melancholy with a slower rhythm and more minor scales, ... There are common concepts that could be used to compose. They could be parameter to constraint a score.
Another artificial intelligence tool that could give good results is the data mining. It could analyze score and stand out concepts from a lot of score belonging to music style or even to a particular artist. In this way, those concepts could lead to new music rules and new ways of composing.

[4]Section 5.5 in the chapter about Musical Aspect

Chapter 9

Conclusion

All along this master's thesis, I have been working to improve the possibilities of constraint programming inside OpenMusic and to make this system more affordable for the users. Starting from previous work on this topic, I have offered a new layer on this project by creating a user's interface and by adding new concepts and functionalities. The interface allows the users to model musical problems without having trouble with writing code and scripts to perform constraint programming searches. It is simple and focuses on what is the most important to model, the variables and the constraints. Complexity of branching and computation are hidden. In addition, I have added new functions that widen the field of possible solutions. This allows the user to get all existing solutions for the problems that he models and to choose the one he prefers. In this way, the musical score resulting of modelling and calculation keeps a bit of subjectivity that is an integral part of music and art domains.

Previous works also shown the utility of using relations and relational constraint programming to create and represent musical objects. This idea being promising, I worked to create global musical variables based on relations in order to incorporate them into the user's interface. This allows the user to use a new kind of constraint based on the relational algebra. Unfortunately, at this time, this part does not work in the interface[1]. However the relational approach is very relevant and could bring a lot of new possibilities into this topic. And as far as I know, OpenMusic is the only tool delving into the relational constraint programming to compute musical scores.

One of the reasons for creating a user's interface to the constraint programming into OpenMusic, is to allow the user to use this technique of artificial intelligence by himself, without the need of a developer to model his problem. That is why I have focused on creating a system in which it will be easy to create new constraint. This facility comes by the possibility to use several constraints inside the interface and by the possibility to code new ones through simple Lisp functions.
Once a user models a problem, it is easy to share it with other users. These users can then use the model by adapting the variables to their own data or even re-manipulate the model to satisfy their problem. This sharing could lead to improvement of existing problem and by the way, improvement of OpenMusic.

[1] More explanations are given into the section 8.2.1

Bibliography

[1] Basic musical concepts -beat, rhythm, melody and harmony. https://www.didjshop.com/BasicMusicalHarmony.html.

[2] Curiculum vitae - serge lemouton. http://serge.lemouton.free.fr.

[3] Foreign function interface - wikipedia. https://en.wikipedia.org/wiki/Foreign_function_interface.

[4] Gecode - an open, free, efficient constraint solving toolkit. http://www.gecode.org.

[5] Interval (music) - wikipedia. https://en.wikipedia.org/wiki/Interval_(music).

[6] Lisp - wikipedia. https://en.wikipedia.org/wiki/Lisp_(programming_language).

[7] Major scale - wikipedia. https://en.wikipedia.org/wiki/Major_scale.

[8] Midi - wikipedia. https://en.wikipedia.org/wiki/MIDI.

[9] Openmusic:home. http://repmus.ircam.fr/openmusic/home.

[10] The philosophy of music (stanford encyclopedia of philosophy. http://plato.stanford.edu/entries/music/.

[11] Pontificia universidad javeriana - camilo rueda. http://cic.puj.edu.co/~crueda/camilo/Home.html.

[12] Pontificia universidad javeriana - mauricio toro. http://cic.puj.edu.co/~mauriciotoro/research.htm.

[13] Rene-louis baron - wikipedia. https://fr.wikipedia.org/wiki/René-Louis_Baron.

[14] Renee-louis baron - musical invention. http://www.realcomposer.com.

[15] Scales and emotions | ethan hein blog. http://www.ethanhein.com/wp/2010/scales-and-emotions/.

[16] Simplify wrapper and interface generator. http://www.swig.org.

[17] Torsten anders - biography. http://cmr.soc.plymouth.ac.uk/tanders/bio.htm.

[18] Understanding the rules of music. http://www.artsreformation.com/a001/ih-music-rules.html.

[19] Université catholique de louvain la neuve - sascha van cauwelaert. `http://www.uclouvain.be/sascha.vancauwelaert`.

[20] Torsten Anders. Composing music by composing rules: Design and usage of a generic music constraint system. 2007.

[21] Torsten Anders and Eduardo R. Miranda. Constraint programming systems for modeling music theories and composition. 2011.

[22] Mauricio Toro B., Camilo Rueda, Carlos Agón, and Gérard Assayag. Gelisp: a library to represent musical csps and search strategies. 2009.

[23] Sascha Van Cauwelaert. Application de la programmation par contraintes relationnelles à l'analyse et à la composition musicales, 2011.

[24] Sascha Van Cauwelaert, Gustavo Gutiérrez, and Peter Van Roy. Implementation of the relation domain for constraint programming. 2011.

[25] Sascha Van Cauwelaert, Gustavo Gutiérrez, and Peter Van Roy. A new approach for constraint programming in music using relation domains. 2012.

[26] Sascha Van Cauwelaert, Gustavo Gutiérrez, and Peter Van Roy. Practical uses of constraint programming in music using relation domains. 2012.

[27] Gustavo Gutiérrez. Constraint programming on relations: theory, practice and implementation.

[28] Patrick N. Juslin. *Music and emotion : theory and research.* Oxford university press, Oxford, 2002.

[29] Bernard Lechevalier. *Le cerveau musicien : neuropsychologie et psychologie cognitive de la perception musicale.* De Boeck, Bruxelles, 2006.

[30] Serge Lemouton. Huit problèmes musicaux résolus grâce á gecode. 2010.

[31] Christian Schulte, Guido Tack, and Mikael Z. Lagerkvist. *Modeling And Programming With Gecode*, 2015.

[32] William Forde Thompson. *Music, thought, and feeling : understanding the psychology of music.* Oxford university press, New York, 2009.

[33] Charlotte Truchet, Carlos Agon, and Gérard Assayag. Recherche adaptative et contraintes musicales. 2001.

[34] Geraint A. Wiggins. The use of constraint systems for musical composition.

Appendix A

Tutorial

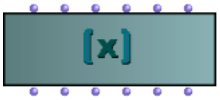
A.1 Introduction

A.2 Presentation of the objects and functions

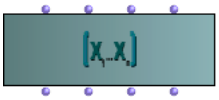
In this section, we introduce the different objects and functions to model a musical constraint satisfaction problem[1] into OpenMusic. There are 3 types of objects and functions, the variables, the constraint and finally the computation functions.

A.2.1 Variables

This first subsection presents the available object to represent the variables of the problem. Today, there are 4 functional objects: the integer variable, the list of integer variables, the boolean variable and the list of boolean variables. They are presented in OpenMusic with the following boxes.

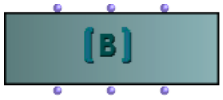


(a) IntVar

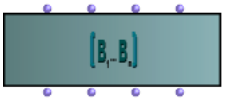


(b) IntVarList

Figure A.1: Integer variables types.



(a) BoolVar



(b) BoolVarList

Figure A.2: Boolean variables types.

The parameters for these objects are:

[1] Also called a MCSP.

Objects	**Parameter**	**Name**	**Value**
All	Lower bound	lb	int or boolean
All	Upper bound	ub	int or boolean
VarList	Size of the list	nsize	int ≥ 0

Figure A.3: Table of variables' parameters.

These objects are used to define decision variables[2]. It is only their solutions that the problem will return.

A.2.2 Constraints

The constraints are functions that will force the variables to respect some rules. Each one is represented by a Lisp function that takes as many inputs as it needs. These inputs can be of different forms but they always have one variable at least. Other possible inputs are integer as for the domain constraints[3] or constant value that can be of two types *IntRelType* or *IntConLevel*. These types are for specific use:

- *IntRelType* expresses a relation between some variables. These are called with "*rel-constraint*" functions. The values of IntRelType are constant and are detailed in the table A.4.
- *IntConLevel* expresses the level of consistency of the constraint. This point is a bit more complex. Since this version of the interface aims to stay simple, this type is not in the implemented constraints[4]. Nevertheless, the table B.1 summarizes the values of *IntConLevel* type.

Value	Relation
$:IRT\_EQ$	$=$
$:IRT\_NQ$	$\neq$
$:IRT\_LQ$	$\leq$
$:IRT\_LE$	$<$
$:IRT\_GQ$	$\geq$
$:IRT\_GR$	$>$

Figure A.4: IntRelType values.

[2]They are called like that because the goal of the problem is to decide which value to assign to each variable.

[3]Domain constraints force variables to restrict their domain to either a specific value or a set of values.

[4]However, if you want more information about this topic, I invite you to read the chapter 4.3 of the manual [31].

Value	Propagation type
$:ICL_VAL$	Value
$:ICL_BND$	Bound
$:ICL_DOM$	Domain
$:ICL_DEF$	Default

Figure A.5: IntConLevel values.

Let's get back to the constraints. As explained, they are Lisp functions and the implemented constraints are summarize at C. These functions return a constraint object containing the reference to their variables and a lambda function implementing the actions of the constraint. In the section A.4, you'll see that it is possible to create new constraints with a little bit of knowledge in Lisp.

List of existing constraint

A.2.3 Search of a solution

Once the problem has been modelled with the variables and the constraints, there are two functions allowing to get the solutions of the problem: *getSolutions* and *getNextSolutions*. They both take two parameters:

- *list-cstrt*, a list of constraint objects return by the constraint functions.
- *nsol*, an integer that indicates how many solutions are wanted.

While the first function, *getSolutions*, will always return the same solution for a problem, *getNextSolutions* will return a different solution at each execution. The functionality of this latter provides the user to look for his preferred solution amongst all of solutions that exists.

The result return by these two functions is a list of solutions. Each solution contains an assignation for each variable. Inside a solution, the assignation are ordered by constraint. Then, the first assignation is done in the first variable from the first constraint in the *list-cstrt*, the second is made in the second variable from the first variable and so on. It should be noted that even if a variable is used into more than one constraint, it will appear only once in the solution. Its place is the one of the first constraint in which it is involved. The example in the section A.3 will clarify this point.

A.3 Model a constraint problem

In this section, we will see a basic problem. It is not a MSCP but the aims is to show how to link variables and constraints and how to solve the problem.
This case is very simple, We want a list of integer to have all its element strictly greater than another variable. But this latter has to be lower or equal to a third variable. Let's mathematically define this problem:

- varlist A : That has a length of 10 and a lower bound of 0 and an upper bound of 100.
- var B : The second variable is an integer variable that stand between 5 and 25.
- var C : The third one is also an integer variable with a domain between 10 and 20.

And the constraint are defining in this form:

- $\bigwedge_{i=0}^{100} A_i > B$
- $B \leq C$
- And to force A to have only distinct values: $distinct(A)$

The modelling of this problem into OpenMusic is illustrated at the figure A.6. The list variable A is represented by the first variable, B by the second and C by the third. The *distinct-constraint* represent the third constraint below while the relation between A and B is expresses by the first *rel-constraint* function and between B and C by the second.

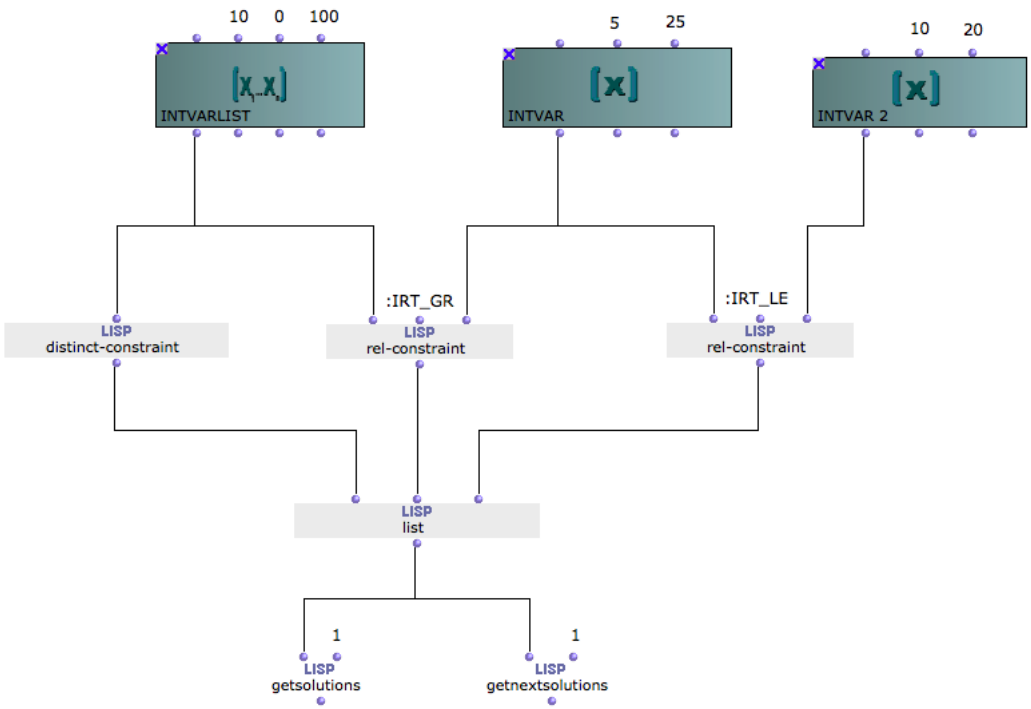
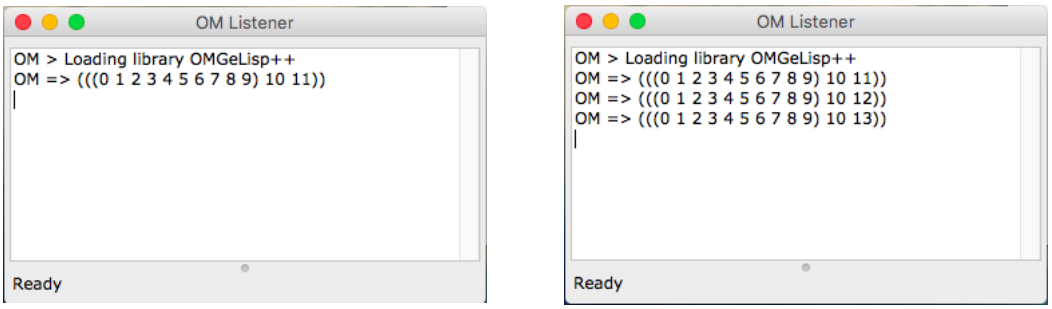


Figure A.6: Modelization of a constraint satisfaction problem.

This model clearly shows how to apply the variables to the constraints. Before the computation phase, all constraint objects return by the constraint functions are put in a

list and then the list is given to *getSolutions* and *getNextSolutions*. they will look for a solution to the problem if it exists, if not they will return the last solution or nil. The solutions returned by both functions are showned in the figure A.7. As each function returns only one result ($nsol = 1$), the solution is a list of all wanted solutions. Each solution contains a list with a solution for each variable. Before looking at the solutions, there are some rules that need to be known.

- Lock the variables. To be able to use the *getNextSolutions*, the variables have to be locked otherwise they will be reinitialize and they will be assigned to new references. These new references will cause errors because they did not get solve a first time by the *getSolutions* function.
- It goes without saying that executing *getNextSolutions* before *getSolutions* will cause the same errors.
- If you restarted the library, you will have to re-execute *getSolutions* first unless you have blocked the boxes.
- The search of a solution in constraint programming may take time if the problem is complex. Unfortunately, in this version of the interface, the search of a solution is execute in the same process than OpenMusic. Thus, in case of complex problem, the software may be freezing during until a solution has been return. An improvement of this will be an important part of the next version of the interface.



(a) Execution of *getSolutions*.

(b) Execution of two *getNextSolutions*.

Figure A.7: Solutions for the model.

A.4 Creation of new constraints

The system behind this interface allows the user to combine existing constraints but it also provides the possibility to create new constraints. To do so, knowledge of Lisp and some notions of constraint programming are required. The utility to create new constraint through Lisp while the user can do it through the OpenMusic's interface is to create more complex constraint that would be hard to do into the interface. Furthermore, in Lisp, more constraint programming tools are available than in the user's interface. Creating a new constraint is the same as creating a new Lisp method but following a certain format.

A.4.1 Format of a constraint function

The Lisp code for a constraint method is composed two part: The definition of the constraint and the creation of the constraint object.

```
;;;;;; constraint definition
(defmethod constraint-name ((v1 Var) (v2 Var) n)
   ;definition of the constraint.
        (setf constraint
        (lambda (space x y) (gelisp-constraint space y z n))
        )
  ;creation of the constraint object
        (make-instance 'Constraint-2 :var1 v1 :var2 v2 :constraint
           constraint)
)
```

First, the definition of the method:

```
(defmethod constraint-name ((v1 Var) (v2 Var) n)
```

This method can take several argument. These arguments can be variables but also integer or any other type of values that makes sense to the creation of your constraint. In this case, it takes two variables v1 and v2 and an integer n.

Secondly, there is the main part, the definition of the constraint. This definition is bundle into a lambda function in order to execute it at the right time. As we can see, the lambda function takes two arguments: the arguments on which the constraint will be applied and the function constraint. Here, the argument are *space*, x and y. The first one is mandatory and it has to stay the first one! It represent the search space to which the variables belong and in which the constraints will be executed[5]. These arguments are not OpenMusic object but are pointers to the GeLisp's variables corresponding to your OpenMusic's variables. Then, if you want to get attribute of your variables, you have to use the argument's name you gave in the definition of the constraint, v1 and v2 in this case. Note that the argument of the GeLisp's constraints are also pointers.

[5]See the notion of space of Gecode [31]

Appendix B

Advices for improvement

B.1 Improve the search

As explained in the section 4.3.2 of the paper, there are several ways to orient the search. It can be done on three parts: the constraints, the branching and the solver.

B.1.1 The constraints

In Gecode, all constraints have an optional argument named the *IntConLevel* or the consistency level. It indicates the kind of propagation that is used on the constraint. It can take 4 values:

Value	Propagation type
: ICL_VAL	Value propagation
: ICL_BND	Bound propagation
: ICL_DOM	Domain propagation
: ICL_DEF	Default propagation

Table B.1: IntConLevel values.

These values are already imported as constants from GeLisp. Their Lisp signature is given in the API chapter at the section C.1.6. To use them, they must be passed as last argument to a constraint function with option[1]. Every function has a default and an option signature given in the API at the section C.1.7. Higher about the consistency levels are provided in the Gecode's manual [31].

B.1.2 Branching function

The branching function is now quite basic, it is set to branch on the variable with the smallest size and to assign the smallest value first. There are a much more possibilities that would be interesting to provide into OpenMusic. Today, GeLisp has not the same branch function than in Gecode, it is simplify and does not take any argument instead of the Gecode's branch function. Here, gl_space is the space on which the branching is realized.

[1]They are named with _ *option* at the end of their name.

```
(branch gl_space)
```

Figure B.1: GeLisp branching function.

```
branch(home, x, INT_VAR_SIZE_MIN(), INT_VAL_MIN());
```

Figure B.2: Gecode branching function.

In this one, the home is the space and x is the variable on which the branching is applied and INT_VAR_SIZE_MIN() represents the variable selection and INT_VAL_MIN() the value selection of the branching.

INT_VAR_NONE()	first unassigned
INT_VAR_RND(r)	randomly
INT_VAR_MERIT_MIN(m,t*)	smallest value of merit function m
INT_VAR_MERIT_MAX(m,t*)	largest value of merit function m
INT_VAR_DEGREE_MIN(t*)	smallest degree
INT_VAR_DEGREE_MAX(t*)	largest degree
INT_VAR_AFC_MIN(afc$^+$,t*)	smallest accumulated failure count (AFC)
INT_VAR_AFC_MAX(afc$^+$,t*)	largest accumulated failure count (AFC)
INT_VAR_ACTIVITY_MIN(act$^+$,t*)	lowest activity
INT_VAR_ACTIVITY_MAX(act$^+$,t*)	highest activity
INT_VAR_MIN_MIN(t*)	smallest minimum value
INT_VAR_MIN_MAX(t*)	largest minimum value
INT_VAR_MAX_MIN(t*)	smallest maximum value
INT_VAR_MAX_MAX(t*)	largest maximum value
INT_VAR_SIZE_MIN(t*)	smallest domain size
INT_VAR_SIZE_MAX(t*)	largest domain size
INT_VAR_DEGREE_SIZE_MIN(t*)	smallest degree divided by domain size
INT_VAR_DEGREE_SIZE_MAX(t*)	largest degree by domain size
INT_VAR_AFC_SIZE_MIN(afc$^+$,t*)	smallest AFC by domain size
INT_VAR_AFC_SIZE_MAX(afc$^+$,t*)	largest AFC by domain size
INT_VAR_ACTIVITY_SIZE_MIN(act$^+$,t*)	smallest activity by domain size
INT_VAR_ACTIVITY_SIZE_MAX(act$^+$,t*)	largest activity by domain size
INT_VAR_REGRET_MIN_MIN(t*)	smallest minimum-regret
INT_VAR_REGRET_MIN_MAX(t*)	largest minimum-regret
INT_VAR_REGRET_MAX_MIN(t*)	smallest maximum-regret
INT_VAR_REGRET_MAX_MAX(t*)	largest maximum-regret

Figure B.3: Variable selection possibilities.

INT_VAL_RND(r)	random value
INT_VAL(v,c*)	defined by value function v and commit function c
INT_VAL_MIN()	smallest value
INT_VAL_MED()	greatest value not greater than the median
INT_VAL_MAX()	largest value
INT_VAL_SPLIT_MIN()	values not greater than mean of smallest and largest value
INT_VAL_SPLIT_MAX()	values greater than mean of smallest and largest value
INT_VAL_RANGE_MIN()	values from smallest range, if domain has several ranges; otherwise, values not greater than mean
INT_VAL_RANGE_MAX()	values from largest range, if domain has several ranges; otherwise, values greater than mean
INT_VALUES_MIN()	all values starting from smallest
INT_VALUES_MAX()	all values starting from largest
INT_VAL_NEAR_MIN(n)	values near to value in array n takes smaller value in case of ties
INT_VAL_NEAR_MAX(n)	values near to value in array n takes larger value in case of ties
INT_VAL_NEAR_INC(n)	values larger than value in array n first
INT_VAL_NEAR_DEC(n)	values smaller than value in array n first

Figure B.4: Value selection possibilities.

The figures[2] B.3 and B.4 show the range of possibilities that are provided by Gecode to set the branching variable or value selection.

How to improve the branching function

The first part to improve is the GeLisp's interface, for now the branching function implementation in GeLisp C++[3] part is:

[2]Both figure come from the page 117 and 118 of the Gecode's manual [31].
[3]More informations about this part are given into [23].

```
    void
GlSpace::branch(void) {
    std::cout << "will branch...." << std::endl;
    if (iv.size() > 0) {
        IntVarArgs ivArgs;
        for (int i =0; i < iv.size(); i++) {
            ivArgs<<iv[i];
        }
        Gecode::branch(*this,ivArgs,INT_VAR_SIZE_MIN(),
           INT_VAL_MIN());
    }
    if (bv.size() > 0) {
        BoolVarArgs bvArgs;
        for (int i =0; i < bv.size(); i++) {
            bvArgs<<bv[i];
        }
        Gecode::branch(*this,bvArgs,INT_VAR_SIZE_MIN(),
           INT_VAL_MIN());
    }
    if (sv.size() > 0) {
        SetVarArgs svArgs;
        for (int i =0; i < sv.size(); i++) {
            svArgs<<sv[i];
        }
        Gecode::branch(*this,svArgs,SET_VAR_NONE(),
           SET_VAL_MIN_INC());
    }
    if (rv.size() > 0) {
        for (int i =0; i < rv.size(); i++) {
            MPG::branch(*this, rv[i],REL_VAL_MIN());
        }
    }
}
```

Figure B.5: Implementation of the branching function.

To provide a way to use the variable and value selection, arguments has to be passed into the branching function above and the function has to be modified. Be careful that the relational branching function is not the same as the Gecode's one. More information has to be asked to Gustavo Gutierrez on this topic.

Once it is done, the second thing to do is to import the variable and value selection functions into GeLisp. All selection functions that has no argument are already in GeLisp[4] because all arguments needed to import new Gecode type into GeLisp. This is easy to do and has not been done principally for time reason.

[4]They are provided in the "constraint.hpp" file.

B.2 Solver's thread

The problem with the solver's thread is to return the values after that the thread has modified them without using the principal thread. To tackle this problem, there are two possibilities. The first one is to use a parallel threads and find an efficient and simple way for the user to return the right value. The second method is to use the asynchronous programming by using sockets for example.

Appendix C

API

C.1 Lisp functions

C.1.1 GeLisp

`(new_GlSpace) :pointer` [Function]

Creates a pointer to a new GlSpace

`(delete_GlSpace` *`gls`*`)` [Function]

Deconstructs a GlSpace. *gls* is a pointer to a GlSpace.

`(GlSpace_newIntVar_minmax`*`gls lb ub`*`) :pointer` [Function]

Returns a pointer to a new IntVar belonging to *gls*.

`(GlSpace_newIntVar_from Sets` *`gls set`*`) :pointer` [Function]

Returns a pointer to a new IntVar belonging to *gls*.

`(GlSpace_newBoolVar_minmax` *`gls lb ub`*`) :pointer` [Function]

Returns a pointer to a new BoolVar belonging to *gls*.

`(GlSpace_newSetVar_home` *`gls`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with greatest lower bound $\{glbMin, \ldots, glbMax\}$, least upper bound $\{lubMin, \ldots, lubMax\}$, and cardinality minimum cardMin and maximum cardMax.

`(GlSpace_newSetVar_minmax_minmax_2option` *`gls glbMin glbMax lubMin lubMax cardMin cardMax`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with greatest lower bound $\{glbMin, \ldots, glbMax\}$, least upper bound $\{lubMin, \ldots, lubMax\}$, and cardinality minimum cardMin and maximum cardMax.

`(GlSpace_newSetVar_minmax_minmax_1option` *`gls glbMin glbMax lubMin lubMax cardMin`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with greatest lower bound $\{glbMin, \ldots, glbMax\}$, least upper bound $\{lubMin, \ldots, lubMax\}$, and cardinality minimum cardMin.

`(GlSpace_newSetVar_minmax_minmax_default` *`gls glbMin glbMax lubMin lubMax`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with greatest lower bound $\{glbMin, \ldots, glbMax\}$, least upper bound $\{lubMin, \ldots, lubMax\}$.

`(GlSpace_newSetVar_set_minmax_2option` *`gls glbD lubMin lubMax cardMin cardMax`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with lower bound Sets to *glbD*, least upper bound $\{lubMin, \ldots, lubMax\}$, and cardinality minimum cardMin and maximum cardMax.

`(GlSpace_newSetVar_set_minmax_1option` *`gls glbD lubMin lubMax cardMin`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with lower bound Sets to *glbD*, least upper bound $\{lubMin, \ldots, lubMax\}$, and cardinality minimum cardMin.

`(GlSpace_newSetVar_set_minmax_default` *`gls glbD lubMin lubMax`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with lower bound Sets to *glbD*, least upper bound $\{lubMin, \ldots, lubMax\}$.

`(GlSpace_newSetVar_minmax_set_2option` *`gls glbMin glbMax lubD cardMin cardMax`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with greatest lower bound $\{glbMin, \ldots, glbMax\}$, upper bound Sets to *lubD*, and cardinality minimum cardMin and maximum cardMax.
.

`(GlSpace_newSetVar_minmax_set_1option` *`gls glbMin glbMax lubD cardMin`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with greatest lower bound $\{glbMin, \ldots, glbMax\}$, upper bound Sets to *lubD*, and cardinality minimum cardMin.

`(GlSpace_newSetVar_minmax_set_default` *`gls glbMin glbMax lubD`*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with greatest lower bound $\{glbMin, \ldots, glbMax\}$, upper bound Sets to *lubD*.

`(GlSpace_newSetVar_set_set_2option` *gls glbD lubD cardMin cardMax*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with lower bound Sets to *glbD*, upper bound Sets to *lubD*, and cardinality minimum cardMin and maximum cardMax.

`(GlSpace_newSetVar_set_set_1option` *gls glbD lubD cardMin*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with lower bound Sets to *glbD*, upper bound Sets to *lubD*, and cardinality minimum cardMin.

`(GlSpace_newSetVar_set_set_default` *gls glbD lubD*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*.The variable is created with lower bound Sets to *glbD*, upper bound Sets to *lubD*.

`(GlSpace_newRelVar` *gls l u*`) :pointer` [Function]

Returns a pointer to a new SetVar belonging to *gls*. *l* and *u* are respectively the lower and upper attributes.

`(GlSpace_runDFS` *gls nbSolutions solutionsVector*`) :pointer` [Function]

Runs the DFS solver of *gls* to find *nbSolutions* first solutions and which are integrated to *solutionsVector*.

`(GlSpace_runBAB` *gls nbSolutions solutionsVector*`) :pointer` [Function]

Runs the BAB solver of *gls* to find *nbSolutions* first solutions and which are integrated to *solutionsVector*.

`(GlSpace_nextDFSSolution` *gls nbSolutions solutionsVector*`) :pointer` [Function]

Runs the DFS solver of *gls* to find *nbSolutions* next solutions and which are integrated to *solutionsVector*.

`(GlSpace_nextBABSolution` *gls nbSolutions solutionsVector*`) :pointer` [Function]

Runs the BAB solver of *gls* to find *nbSolutions* next solutions and which are integrated to *solutionsVector*.

`(GlSpace_getIntVar` *gls indexInVector*`) :pointer` [Function]

Returns the pointer to the IntVar at the index *indexInVector* in *gls*.

`(GlSpace_getBoolVar` *gls indexInVector*`) :pointer` [Function]

Returns the pointer to the BoolVar at the index *indexInVector* in *gls*.

`(GlSpace_getSetVar` *gls indexInVector*`) :pointer` [Function]

Returns the pointer to the SetVar at the index *indexInVector* in *gls*.

`(GlSpace_getRelVar gls indexInVector) :pointer` [Function]

Returns the pointer to the RelVar at the index *indexInVector* in *gls*.

`(GlSpace_createGlSpaceVector gls) :pointer` [Function]

Creates a vector of GlSpace in which the solutions will be put on.

`(GlIntVar_getIndexInVector iv) :int` [Function]

Returns the index of the pointer to the GlIntVar *iv*.

`(GlIntVar_setIndexInVector iv newIndex) :int` [Function]

Sets the index of the pointer of the GlIntVar *iv* to *newIndex*.

`(GlIntVar_getIntVarSol iv solvect nsol)` [Function]

Returns the solutions for the pointer *iv* of the solution vector *solvect*.

`(delete_GlIntVar iv)` [Function]

Deletes the GlIntVar object references by *iv*.

`(new_GlIntVarList_minmax gls n min max) :pointer` [Function]

Creates a new GlIntVarList in *gls* where $min \leq v_i \leq max$ for all $0 \leq i < n$.

`(GlIntVarList_size ivl) :int` [Function]

Returns the size of the GlIntVarList references by *ivl*.

`(GlIntVarList_ Adds ivl x)` [Function]

Adds a GlIntVar to the GlIntVarList references by *ivl*.

`(GlIntVarList_get ivl) :pointer` [Function]

Returns the GlIntVarList references by *ivl*.

`(GlIntVarList_getVar ivl index) :pointer` [Function]

Returns the GlIntVar at the index *index* of the GlIntVarList references by *ivl*.

`(GlIntVarList_getIntVarListSol ivl solvect nsol) :pointer` [Function]

Returns the *nsol* last solutions of the GlIntVarList in *solvect*.

`(delete_GlIntVarList ivl)` [Function]

Deconstructs the GlIntVarList references by *ivl*.

`(GlBoolVar_getIndexInVector iv) :int` [Function]

Returns the index of the pointer to the GlBoolVar *iv*.

`(GlBoolVar_setIndexInVector` *iv newIndex*`) :int` [Function]

Sets the index of the pointer of the GlBoolVar *iv* to *newIndex*.

`(GlBoolVar_getIntVarSol` *iv solvect nsol*`)` [Function]

Returns the solutions for the pointer *iv* of the solution vector *solvect*.

`(delete_GlBoolVar` *iv*`)` [Function]

Deletes the GlBoolVar object references by *iv*.

`(new_GlBoolVarList_minmax` *gls n min max*`) :pointer` [Function]

Creates a new GlBoolVarList in *gls* where $min \leq v_i \leq max$ for all $0 \leq i < n$.

`(GlBoolVarList_size` *ivl*`) :int` [Function]

Returns the size of the GlBoolVarList references by *ivl*.

`(GlBoolVarList_ Adds` *ivl x*`)` [Function]

Adds a GlIntVar to the GlBoolVarList references by *ivl*.

`(GlBoolVarList_get` *ivl*`) :pointer` [Function]

Returns the GlBoolVarList references by *ivl*.

`(GlBoolVarList_getVar` *ivl index*`) :pointer` [Function]

Returns the GlIntVar at the index *index* of the GlBoolVarList references by *ivl*.

`(GlBoolVarList_getIntVarListSol` *ivl solvect nsol*`) :pointer` [Function]

Returns the *nsol* last solutions of the GlBoolVarList in *solvect*.

`(delete_GlBoolVarList` *ivl*`)` [Function]

Deconstructs the GlBoolVarList references by *ivl*.

C.1.2 OpenMusic musical variables

`(getSolutions` *list-cstrt nsol*`)` [Function]

Get the first solutions of a model. *nsol* is the number of solutions.

`(getNextSolutions` *list-cstrt nsol*`)` [Function]

Get the next solutions of a model. *nsol* is the number of solutions.

`(getDFSSolutions` *list-cstrt nsol*`)` [Function]

Get the first DFS solutions of a model. *nsol* is the number of solutions. This function aims to become more specific.

`(getNextDFSSolutions list-cstrt nsol)` [Function]

Get the next DFS solutions of a model. *nsol* is the number of solutions. This function aims to become more specific.

`(getBABSolutions list-cstrt nsol)` [Function]

Get the first BAB solutions of a model. *nsol* is the number of solutions. This function aims to become more specific.

`(getNextBABSolutions list-cstrt nsol)` [Function]

Get the next BAB solutions of a model. *nsol* is the number of solutions. This function aims to become more specific.

`(IntVar lb ub)` [Class]

Create a new IntVar v where $l \leq v \leq u$.

`(set-space (self IntVar) home)` [Function]

Set the GlSpace of *self* to *home*.

`(getSolForOneVar (solvect (self IntVar) nsol)` [Function]

Returns the solution of *self* from *solvect*. *solvect* is a pointer to a vector of spaces of solutions.

`(IntVarList n lb ub)` [Class]

Create a new IntVarList v where $l \leq v_i \leq u$ for all $0 \leq i < n$.

`(set-space (self IntVarList) home)` [Function]

Set the GlSpace of *self* to *home*.

`(getSolForOneVar (solvect (self IntVarList) nsol)` [Function]

Returns the solution of *self* from *solvect*. *solvect* is a pointer to a vector of spaces of solutions.

`(BoolVar lb ub)` [Class]

Create a new BoolVar v where $l \leq v \leq u$.

`(set-space (self BoolVar) home)` [Function]

Set the GlSpace of *self* to *home*.

`(getSolForOneVar (solvect (self BoolVar) nsol)` [Function]

Returns the solution of *self* from *solvect*. *solvect* is a pointer to a vector of spaces of solutions.

`(BoolVarList n lb ub)` [Class]

Create a new BoolVarList v where $l \leq v_i \leq u$ for all $0 \leq i < n$.

`(set-space (self BoolVarList) home)` [Function]

Set the GlSpace of *self* to *home*.

(getSolForOneVar *(solvect (self BoolVarList) nsol*) [Function]

Returns the solution of *self* from *solvect*. *solvect* is a pointer to a vector of spaces of solutions.

C.1.3 OpenMusic constraints

Basic constraints and constraint objects

(Constraint-1 *constraint var1*) [Class]

Create a constraint object with a *constraint* function and *var1* variable

(Constraint-2 *constraint var1 var2*) [Class]

Create a constraint object with a *constraint* function and *var1 var2* variables

(Constraint-3 *constraint var1 var2 var3*) [Class]

Create a constraint object with a *constraint* function and *var1 var2 var3* variables

(Constraint-4 *constraint var1 var2 var3 var4*) [Class]

Create a constraint object with a *constraint* function and *var1 var2 var3 var4* variables

(get-var *(self Constraint-X)*) [Function]

Returns a list with the variables of *self*

(set-cstrt-space *(self Constraint-X) home*) [Function]

Set the space of every variable of *self*, *home* is a pointer to a GlSpace

(apply-constraint *(self Constraint-X) home*) [Function]

Apply the constraint of *self* on its variables, *home* is a pointer to a GlSpace

(domain-constraint *(var1 IntVar) (var2 IntVar)*) [Function]

Constraints domain of *var1* according to domain of *var1*. *var1* and *var2* are OpenMusic variables

(distinct-constraint *(var1 IntVarList)*) [Function]

Post propagator for $var1_i \neq var1\_j$ for all $0 \leq i \neq j < |var1|$. *var1* is OpenMusic variable

(rel-constraint *(var1 IntVar) irt (var2 IntVar)*) [Function]

Post propagator for $var1 \sim_{irt} var2$. *var1* and *var2* are OpenMusic variables

(rel-constraint *(var1 IntVarList) irt (var2 IntVar)*) [Function]

Post propagator for $var1_i \sim_{irt} var2$ for all $0 \leq i < |var1|$. *var1* and *var2* are OpenMusic variables

C.1.4 Created constraints

`(all-intervals` *`(var1 IntVarList) (var2 IntVarList)`* `)` [Function]

Constraints *var1* and *var2* to the all-intervals problem. *var1* and *var2* are OpenMusic variables

C.1.5 Relation type

:IRT_EQ [Constant]

Equality (=)

:IRT_NQ [Constant]

Disequality ($\neq$).

:IRT_LQ [Constant]

Less or equal ($\leq$).

:IRT_LE [Constant]

Less ($<$).

:IRT_GQ [Constant]

Greater or equal ($\geq$).

:IRT_GR [Constant]

Greater ($>$).

C.1.6 Constency level

:ICL_VAL [Constant]

Value propagation or consistency (naive).

:ICL_BND [Constant]

Bounds propagation or consistency.

:ICL_DOM [Constant]

Domain propagation or consistency.

:ICL_DEF [Constant]

The default consistency for a constraint.

C.1.7 Constraints

Some clarifications about the next function signatures: the *home* argument is a pointer to a GlSpace, the *irt* argument is a constant of *IntRelType*, the *icl* argument is a constant of type *IntConLevel*, IntSet, IntSetArgs, IntArgs are pointers to the corresponding C++ type import by GeLisp. Furthemore the arguments of type X_ptr reference the ptr attribute of the Lisp object X. All other arguments are integers.
These constraints manipule C++ pointer, they can be used inside an OpenMusic constraint function.

`(dom_onIntVar_option` *home (x IntVar_ptr) n icl*`)` [Function]

Propagates $x = n$.

`(dom_onIntVar_default` *home (x IntVar_ptr) n*`)` [Function]

Propagates $x = n$.

`(dom_onIVa_option` *home (x IntVarList_ptr) n icl*`)` [Function]

Propagates $x_i = n$ for all $0 \leq i < |x|$.

`(dom_onIVa_default` *home (x IntVarList_ptr) n*`)` [Function]

Propagates $x_i = n$ for all $0 \leq i < |x|$.

`(dom_onIntVarWithBounds_option` *home (x IntVar_ptr) l m icl*`)` [Function]

Propagates $l \leq x \leq m$.

`(dom_onIntVarWithBounds_default` *home (x IntVar_ptr) l m*`)` [Function]

Propagates $l \leq x \leq m$.

`(dom_onIVAWithBounds_option` *home (x IntVarList_ptr) l m icl*`)` [Function]

Propagates $l \leq x_i \leq m$ for all $0 \leq i < |x|$.

`(dom_onIVAWithBounds_default` *home (x IntVarList_ptr) l m*`)` [Function]

Constrain domain of x according to domain of d.

`(dom_onIntVarFromIntVar_option` *home (x IntVar_ptr) (d IntVar_ptr) icl*`)` [Function]

Constrain domain of x according to domain of d.

`(dom_onIntVarFromIntVar_default` *home (x IntVar_ptr) (d IntVar_ptr)*`)` [Function]

Propagates $l \leq x \leq m$.

`(dom_onIntVarAndIntSet_option` *home (x IntVar_ptr) (s IntSet) icl*`)` [Function]

Propagates $x \in (sIntSet)$.

`(dom_onIntVarAndIntSet_default` *home (x IntVar_ptr) (s IntSet)*`)` [Function]

Propagates $x \in (sIntSet)$.

`(dom_onIVAAndIntSet_option home (x IntVarList_ptr) (s IntSet) icl)` [Function]

Propagates $x_i \in s$ for all $0 \leq i < |x|$.

`(dom_onIVAAndIntSet_default home (x IntVarList_ptr) (s IntSet))` [Function]

Propagates $x_i \in s$ for all $0 \leq i < |x|$.

`(rel_onIntVar_option home (x0 IntVar_ptr) irt (x1 IntVar_ptr) icl)` [Function]

Post propagator for $x_0 \sim_{irt} x_1$.

`(rel_onIntVar_default home (x0 IntVar_ptr) irt (x1 IntVar_ptr))` [Function]

Post propagator for $x_0 \sim_{irt} x_1$.

`(rel_onIntVarAndIVa_option home (x IntVarList_ptr) irt (y IntVar_ptr) icl)` [Function]

Post propagator for $x_i \sim_{irt} y$ for all $0 \leq i < |x|$.

`(rel_onIntVarAndIVa_default home (x IntVarList_ptr) irt (y IntVar_ptr))` [Function]

Post propagator for $x_i \sim_{irt} y$ for all $0 \leq i < |x|$.

`(rel_onIntVarAndInt_option home (x IntVar_ptr) irt c icl)` [Function]

Propagates $x \sim_{irt} c$.

`(rel_onIntVarAndInt_default home (x IntVar_ptr) irt c)` [Function]

Propagates $x \sim_{irt} c$.

`(rel_onIVAAndInt_option home (x IntVarList_ptr) irt c icl)` [Function]

Propagates $x_i \sim_{irt} c$ for all $0 \leq i < |x|$.

`(rel_onIVAAndInt_default home (x IntVarList_ptr) irt c)` [Function]

Propagates $x_i \sim_{irt} c$ for all $0 \leq i < |x|$.

`(rel_on2BoolVar_option home (x0 BoolVar_ptr) irt (x1 BoolVar_ptr) icl)` [Function]

Post domain consistent propagator for $x_0 \sim_{irt} x_1$.

`(rel_on2BoolVar_default home (x0 BoolVar_ptr) irt (x1 BoolVar_ptr))` [Function]

Post domain consistent propagator for $x_0 \sim_{irt} x_1$.

`(rel_onBVAAndBoolVar_option home (x BoolVarList_ptr) irt (y BoolVar_ptr) icl)` [Function]

Post domain consistent propagator for $x_i \sim_{irt} y$ for all $0 \leq i < |x|$.

`(rel_onBVAAndBoolVar_default` *home (x BoolVarList_ptr) irt (y BoolVar_ptr)* `)` [Function]

Post domain consistent propagator for $x_i \sim_{irt} y$ for all $0 \leq i < |x|$.

`(rel_onBoolVarAndInt_option` *home (x BoolVar_ptr) irt n icl*`)` [Function]

Propagates $x_i \sim_{irt} n$ for all $0 \leq i < |x|$.

`(rel_onBoolVarAndInt_default` *home (x BoolVar_ptr) irt n*`)` [Function]

Propagates $x_i \sim_{irt} n$ for all $0 \leq i < |x|$.

`(rel_on2BVa_option` *home (x BoolVarList_ptr) irt (y BoolVarList_ptr) icl*`)` [Function]

Post domain consistent propagator for relation between x and y.

`(rel_on2BVa_default` *home (x BoolVarList_ptr) irt (y BoolVarList_ptr)* `)` [Function]

Post domain consistent propagator for relation between x and y.

`(rel_onBVa_option` *home (x BoolVarList_ptr) irt icl*`)` [Function]

Post domain consistent propagator for relation between elements in x.

`(rel_onBVa_default` *home (x BoolVarList_ptr) r*`)` [Function]

Post domain consistent propagator for relation between elements in x.

`(element_onIVAAnd2IntVar_option` *home (x IntVarList_ptr) (y0 IntVar_ptr) (y1 IntVar_ptr) icl*`)` [Function]

Post propagator for $x_{y\_0} = y\_1$.

`(element_onIVAAnd2IntVar_default` *home (x IntVarList_ptr) (y0 IntVar_ptr) (y1 IntVar_ptr)* `)` [Function]

Post propagator for $x_{y\_0} = y\_1$.

`(element_onIVAAndIntVarAndInt_option` *home (x IntVarList_ptr) (y0 IntVar_ptr) y1 icl*`)` [Function]

Post propagator for $x_{y\_0} = y\_1$.

`(element_onIVAAndIntVarAndInt_default` *home (x IntVarList_ptr) (y0 IntVar_ptr) y1*`)` [Function]

Post propagator for $x_{y\_0} = y\_1$.

`(element_onBVAAndIntVarAndBoolVar_option` *home (x BoolVarList_ptr) (y0 IntVar_ptr) (y1 BoolVar_ptr) icl*`)` [Function]

Post domain consistent propagator for $x_{y\_0} = y\_1$.

`(element_onBVAAndIntVarAndBoolVar_default` *home (x BoolVarList_ptr) (y0 IntVar_ptr) (y1 BoolVar_ptr)* `)` [Function]

Post domain consistent propagator for $x_{y\_0} = y\_1$.

`(element_onBVAAndIntVarAndInt_option` *home (x BoolVarList_ptr) (y0 IntVar_ptr) y1 icl* `)` [Function]

Post domain consistent propagator for $x_{y\_0} = y\_1$.

`(element_onBVAAndIntVarAndInt_default` *home (x BoolVarList_ptr) (y0 IntVar_ptr) y1* `)` [Function]

Post domain consistent propagator for $x_{y\_0} = y\_1$.

`(element_onIVAAnd3IntVarAnd2Int_option` *home (x IntVarList_ptr) (x IntVar_ptr) w (y IntVar_ptr) h (z IntVar_ptr) icl* `)` [Function]

Post propagator for $a_{x+w \cdot y} = z$.

`(element_onIVAAnd3IntVarAnd2Int_default` *home (x IntVarList_ptr) (x IntVar_ptr) w (y IntVar_ptr) h (z IntVar_ptr)* `)` [Function]

Post propagator for $a_{x+w \cdot y} = z$.

`(element_onBVAAnd2IntVarAnd2IntAndBoolVar_option` *home (x BoolVarList_ptr) (x IntVar_ptr) w (y IntVar_ptr) h (z BoolVar_ptr) icl* `)` [Function]

Post domain consistent propagator for $a_{x+w \cdot y} = z$.

`(element_onBVAAnd2IntVarAnd2IntAndBoolVar_default` *home (a BoolVarList_ptr) (x IntVar_ptr) w (y IntVar_ptr) h (z BoolVar_ptr)* `)` [Function]

Post domain consistent propagator for $a_{x+w \cdot y} = z$.

`(distinct_option` *home (x IntVarList_ptr) icl* `)` [Function]

Post propagator for $x_i \neq x_j$ for all $0 \leq i \neq j < |x|$.

`(distinct_default` *home (x IntVarList_ptr)* `)` [Function]

Post propagator for $x_i \neq x_j$ for all $0 \leq i \neq j < |x|$.

`(distinct_constant_added_to_var_option` *home n (x IntVarList_ptr) icl* `)` [Function]

Post propagator for $x_i + n\_i \neq x_j + n\_j$ for all $0 \leq i \neq j < |x|$.

`(distinct_constant_added_to_var_default` *home n (x IntVarList_ptr)* `)` [Function]

Post propagator for $x_i + n\_i \neq x_j + n\_j$ for all $0 \leq i \neq j < |x|$.

`(channel_onIVAAndIVa_option` *home (x IntVarList_ptr) (y IntVarList_ptr) icl* `)` [Function]

Post propagator for $x_i = j \leftrightarrow y\_j = i$ for all $0 \leq i < |x|$.

`(channel_onIVAAndIVa_default` *home (x IntVarList_ptr) (y IntVarList_ptr)* `)` [Function]

Post propagator for $x_i = j \leftrightarrow y\_j = i$ for all $0 \leq i < |x|$.

`(channel_on2IVAAnd2IntVar_option` *home (x IntVarList_ptr) xoff (y IntVarList_ptr) yoff icl* `)` [Function]

Post propagator for $x_i - \mathit{xoff} = j \leftrightarrow y\_j - \mathit{yoff} = i$ for all $0 \leq i < |x|$.

`(channel_on2IVAAnd2IntVar_default` *home (x IntVarList_ptr) xoff (y IntVarList_ptr) yoff* `)` [Function]

Post propagator for $x_i - \mathit{xoff} = j \leftrightarrow y\_j - \mathit{yoff} = i$ for all $0 \leq i < |x|$.

`(channel_onBoolVarAndIntVar_option` *home (x0 BoolVar_ptr) (x1 IntVar_ptr) icl* `)` [Function]

Post domain consistent propagator for channeling a Boolean and an integer variable $x_0 = x_1$.

`(channel_onBoolVarAndIntVar_default` *home (x0 BoolVar_ptr) (x1 IntVar_ptr)* `)` [Function]

Post domain consistent propagator for channeling a Boolean and an integer variable $x_0 = x_1$.

`(channel_onIntVarAndBoolVar_option` *home (x0 IntVar_ptr) (x1 BoolVar_ptr) icl* `)` [Function]

Post domain consistent propagator for channeling an integer and a Boolean variable $x_0 = x_1$.

`(channel_onIntVarAndBoolVar_default` *home (x0 IntVar_ptr) (x1 BoolVar_ptr)* `)` [Function]

Post domain consistent propagator for channeling an integer and a Boolean variable $x_0 = x_1$.

`(channel_onBVAAndIntVar_2option` *home (x BoolVarList_ptr) (y IntVar_ptr) o icl* `)` [Function]

Post domain consistent propagator for channeling Boolean and integer variables $x_i = 1 \leftrightarrow y = i + o$.

`(channel_onBVAAndIntVar_1option` *home (x BoolVarList_ptr) (y IntVar_ptr) o* `)` [Function]

Post domain consistent propagator for channeling Boolean and integer variables $x_i = 1 \leftrightarrow y = i + o$.

`(channel_onBVAAndIntVar_default` *home (x BoolVarList_ptr) (y IntVar_ptr)* `)` [Function]

Post domain consistent propagator for channeling Boolean and integer variables $x_i = 1 \leftrightarrow y = i + o$.

`(sorted_onIVAAndIVa_option` *home (x IntVarList_ptr) (y IntVarList_ptr) icl*`)` [Function]

Post propagator that y is x sorted in increasing order.

`(sorted_onIVAAndIVa_default` *home (x IntVarList_ptr) (y IntVarList_ptr)*`)` [Function]

Post propagator that y is x sorted in increasing order.

`(sorted_on3IVa_option` *home (x IntVarList_ptr) (y IntVarList_ptr) (z IntVarList_ptr) icl*`)` [Function]

Post propagator that y is x sorted in increasing order.

`(sorted_on3IVa_default` *home (x IntVarList_ptr) (y IntVarList_ptr) (z IntVarList_ptr)*`)` [Function]

Post propagator that y is x sorted in increasing order.

`(count_onIVAAnd2Int_option` *home (x IntVarList_ptr) n irt m icl*`)` [Function]

Post propagator for $\#i \in 0, \ldots, |x| - 1 \,|\, x_i = n \sim_{irt} m$.

`(count_onIVAAnd2Int_default` *home (x IntVarList_ptr) n irt m*`)` [Function]

Post propagator for $\#i \in 0, \ldots, |x| - 1 \,|\, x_i = n \sim_{irt} m$.

`(count_onIVAAndIntVarAndInt_option` *home (x IntVarList_ptr) (y IntVar_ptr) irt m icl*`)` [Function]

Post propagator for $\#i \in 0, \ldots, |x| - 1 \,|\, x_i = y \sim_{irt} m$.

`(count_onIVAAndIntVarAndInt_default` *home (x IntVarList_ptr) (y IntVar_ptr) irt m*`)` [Function]

Post propagator for $\#i \in 0, \ldots, |x| - 1 \,|\, x_i = y \sim_{irt} m$.

`(count_onIVAAndIntArgsAndInt_option` *home (x IntVarList_ptr) (y IntArgs) irt m icl*`)` [Function]

Post propagator for $\#i \in 0, \ldots, |x| - 1 \,|\, x_i = y\_i \sim_{irt} m$.

`(count_onIVAAndIntArgsAndInt_default` *home (x IntVarList_ptr) (y IntArgs_ptr) irt m*`)` [Function]

Post propagator for $\#i \in 0, \ldots, |x| - 1 \,|\, x_i = y\_i \sim_{irt} m$.

`(count_onIVAAndIntAndIntVar_option` *home (x IntVarList_ptr) n irt (z IntVar_ptr) icl*`)` [Function]

Post propagator for $\#i \in 0, \ldots, |x| - 1 \,|\, x_i = n \sim_{irt} z$.

`(count_onIVAAndIntAndIntVar_default` *home (x IntVarList_ptr) n irt (z IntVar_ptr)*`)` [Function]

Post propagator for $\#i \in 0,\ldots,|x|-1 \,|\, x_i = n \sim_{irt} z$.

`(count_onIVAAnd2IntVar_option` *`home (x IntVarList_ptr) (y IntVar_ptr) irt (z IntVar_ptr) icl`*`)` [Function]

Post propagator for $\#i \in 0,\ldots,|x|-1 \,|\, x_i \in y \sim_{irt} z$.

`(count_onIVAAnd2IntVar_default` *`home (x IntVarList_ptr) (y IntVar_ptr) irt (z IntVar_ptr)`*`)` [Function]

Post propagator for $\#i \in 0,\ldots,|x|-1 \,|\, x_i \in y \sim_{irt} z$.

`(count_onIVAAndIntArgsAndIntVar_option` *`home (x IntVarList_ptr) (y IntArgs) irt (z IntVar_ptr) icl`*`)` [Function]

Post propagator for $\#i \in 0,\ldots,|x|-1 \,|\, x_i = y \sim_{irt} z$.

`(count_onIVAAndIntArgsAndIntVar_default` *`home (x IntVarList_ptr) (y IntArgs) irt (z IntVar_ptr)`*`)` [Function]

Post propagator for $\#i \in 0,\ldots,|x|-1 \,|\, x_i = y \sim_{irt} z$.

`(count_on2IVa_option` *`home (x IntVarList_ptr) (c IntVarList_ptr) icl`*`)` [Function]

Posts a global count (cardinality) constraint.

`(count_on2IVa_default` *`home (x IntVarList_ptr) (c IntVarList_ptr)`*`)` [Function]

Posts a global count (cardinality) constraint.

`(count_on2IVAAndIntArgs_option` *`home (x IntVarList_ptr) (c IntVarList_ptr) (v IntArgs) icl`*`)` [Function]

Posts a global count (cardinality) constraint.

`(count_on2IVAAndIntArgs_default` *`home (x IntVarList_ptr) (c IntVarList_ptr) (v IntArgs)`*`)` [Function]

Posts a global count (cardinality) constraint.

`(count_onIVAAndISAIntArgs_option` *`home (x IntVarList_ptr) (c IntSetArgs) (v IntArgs) icl`*`)` [Function]

Posts a global count (cardinality) constraint.

`(count_onIVAAndISAIntArgs_default` *`home (x IntVarList_ptr) (c IntSetArgs) (v IntArgs)`*`)` [Function]

Posts a global count (cardinality) constraint.

`(count_onIVAAndIntSetIntArgs_option` *`home (x IntVarList_ptr) (c IntSet) (v IntArgs) icl`*`)` [Function]

Posts a global count (cardinality) constraint.

`(count_onIVAAndIntSetIntArgs_default` *`home (x IntVarList_ptr) (c IntSet) (v IntArgs)`*`)` [Function]

Posts a global count (cardinality) constraint.

`(sequence_IVa_option` *`home (x IntVarList_ptr) (s IntSet) q l u icl`*`)` [Function]

Post propagator for sequence (x, s, q, l, u).

`(sequence_IVa_default` *`home (x IntVarList_ptr) (s IntSet) q l u`*`)` [Function]

Post propagator for sequence (x, s, q, l, u).

`(sequence_BVa_option` *`home (x BoolVarList_ptr) (s IntSet) q l u icl`*`)` [Function]

Post propagator for sequence (x, s, q, l, u).

`(sequence_BVa_default` *`home (x BoolVarList_ptr) (s IntSet) q l u`*`)` [Function]

Post propagator for sequence (x, s, q, l, u).

`(min_on3IntVar_option` *`home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr) icl`*`)` [Function]

Post propagator for $\min x_0, x_1 = x_2$.

`(min_on3IntVar_default` *`home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr)`*`)` [Function]

Post propagator for $\min x_0, x_1 = x_2$.

`(min_onIVAANDIntVar_option` *`home (x IntVarList_ptr) (y IntVar_ptr) icl`*`)` [Function]

Post propagator for $\min x = y$.

`(min_onIVAANDIntVar_default` *`home (x IntVarList_ptr) (y IntVar_ptr)`*`)` [Function]

Post propagator for $\min x = y$.

`(max_on3IntVar_option` *`home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr) icl`*`)` [Function]

Post propagator for $\max x_0, x_1 = x_2$.

`(max_on3IntVar_default` *`home (x0 IntVar_ptr) (x1 IntVar_ptr) x2`*`)` [Function]

Post propagator for $\max x_0, x_1 = x_2$.

`(max_onIVAANDIntVar_option` *`home (x IntVarList_ptr) (y IntVar_ptr) icl`*`)` [Function]

Post propagator for $\max x = y$.

`(max_onIVAANDIntVar_default` *`home (x IntVarList_ptr) (y IntVar_ptr)`*`)` [Function]

Post propagator for $\max x = y$.

`(abs_option` *home (x0 IntVar_ptr) (x1 IntVar_ptr) icl*`)` [Function]

Post propagator for $|x_0| = x_1$.

`(abs_default` *home (x0 IntVar_ptr) (x1 IntVar_ptr)*`)` [Function]

Post propagator for $|x_0| = x_1$.

`(mult_option` *home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr) icl*`)` [Function]

Post propagator for $x_0 \cdot x_1 = x_2$.

`(mult_default` *home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr)* `)` [Function]

Post propagator for $x_0 \cdot x_1 = x_2$.

`(sqr_option` *home (x0 IntVar_ptr) (x1 IntVar_ptr) icl*`)` [Function]

Post propagator for $\lfloor\sqrt{x_0}\rfloor = x_1$.

`(sqr_default` *home (x0 IntVar_ptr) (x1 IntVar_ptr)*`)` [Function]

Post propagator for $\lfloor\sqrt{x_0}\rfloor = x_1$.

`(sqrt_option` *home (x0 IntVar_ptr) (x1 IntVar_ptr) icl*`)` [Function]

Post propagator for $x_0^n = x_1$.

`(sqrt_default` *home (x0 IntVar_ptr) (x1 IntVar_ptr)*`)` [Function]

Post propagator for $x_0^n = x_1$.

`(divmod_option` *home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr) (x3 IntVar_ptr) icl*`)` [Function]

Post propagator for $x_0 \mathrm{div} x_1 = x_2 \wedge x_0 \mathrm{mod} x_1 = x_3$.

`(divmod_default` *home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr) (x3 IntVar_ptr)*`)` [Function]

Post propagator for $x_0 \mathrm{div} x_1 = x_2 \wedge x_0 \mathrm{mod} x_1 = x_3$.

`(div_option` *home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr) icl*`)` [Function]

Post propagator for $x_0 \mathrm{div} x_1 = x_2$.

`(div_default` *home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr)* `)` [Function]

Post propagator for $x_0 \mathrm{div} x_1 = x_2$.

`(mod_option` *home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr) icl*`)` [Function]

Post propagator for $x_0 \mathrm{mod} x_1 = x_2$.

(mod_default *home (x0 IntVar_ptr) (x1 IntVar_ptr) (x2 IntVar_ptr)*) [Function]

Post propagator for $x_0^2 = x_1$.

(linear_onIVAAndInt_option *home (x IntVarList_ptr) irt c icl*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} x_i \sim_{irt} c$.

(linear_onIVAAndInt_default *home (x IntVarList_ptr) irt c*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} x_i \sim_{irt} c$.

(linear_onIVAAndIntVar_option *home (x IntVarList_ptr) irt (y IntVar_ptr) icl*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} x_i \sim_{irt} y$.

(linear_onIVAAndIntVar_default *home (x IntVarList_ptr) irt (y IntVar_ptr)*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} x_i \sim_{irt} y$.

(linear_onIntArgsAndIVAAndInt_option *home a (x IntVarList_ptr) irt c icl*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} a\_i \cdot x_i \sim_{irt} c$.

(linear_onIntArgsAndIVAAndInt_default *home a (x IntVarList_ptr) irt c*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} a\_i \cdot x_i \sim_{irt} c$.

(linear_onIntArgsAndIVAAndIntVar_option *home a (x IntVarList_ptr) irt (y IntVar_ptr) icl*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} a\_i \cdot x_i \sim_{irt} y$.

(linear_onIntArgsAndIVAAndIntVar_option *home a (x IntVarList_ptr) irt (y IntVar_ptr)*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} a\_i \cdot x_i \sim_{irt} y$.

(linear_onBVAAndInt_option *home (x BoolVarList_ptr) irt c icl*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} x_i \sim_{irt} c$.

(linear_onBVAAndInt_default *home (x BoolVarList_ptr) irt c*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} x_i \sim_{irt} c$.

(linear_onBVAAndIntVar_option *home (x BoolVarList_ptr) irt (y IntVar_ptr) icl*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} x_i \sim_{irt} y$.

(linear_onBVAAndIntVar_default *home (x BoolVarList_ptr) irt (y IntVar_ptr)*) [Function]

Post propagator for $\sum\_i = 0^{|x|-1} x_i \sim_{irt} y$.

`(linear_onIntArgsAndBVAAndInt_option` *`home a (x BoolVarList_ptr) irt c icl`*`)` [Function]

Post propagator for $\sum\_i = 0^{|x|-1} a\_i \cdot x_i \sim_{irt} c$.

`(linear_onIntArgsAndBVAAndInt_default` *`home a (x BoolVarList_ptr) irt c`*`)` [Function]

Post propagator for $\sum\_i = 0^{|x|-1} a\_i \cdot x_i \sim_{irt} c$.

`(linear_onIntArgsAndBVAAndIntVar_option` *`home a (x BoolVarList_ptr) irt (y IntVar_ptr) icl`*`)` [Function]

Post propagator for $\sum\_i = 0^{|x|-1} a\_i \cdot x_i \sim_{irt} y$.

`(linear_onIntArgsAndBVAAndIntVar_default` *`home a (x BoolVarList_ptr) irt (y IntVar_ptr)`*`)` [Function]

Post propagator for $\sum\_i = 0^{|x|-1} a\_i \cdot x_i \sim_{irt} y$.

Appendix D

Code

D.1 C++

D.1.1 GeLisp

```
1  /*
2  GeLisp is a library that interfaces Gecode's functions (from the Gecode's
       version 4.4.0) to the Lisp language using SWIG (version 3.0.7).
3  This library aims to adapt Gecode for OpenMusic, a software developed by
       the IRCAM.
4  More details on the use of SWIG and GeLisp are explained into these two
       documents:
5  https://www.info.ucl.ac.be/~pvr/MemoireSaschaVanCauwelaert.pdf
6  https://github.com/Zoetti/openmusicthesis2016.git
7
8  The following code has been principally coded by Sascha Van Cauwelaert,
       University Catholic of Louvain-la-neuve, during his master's thesis in
       2010.
9  An update and additional functionalities has been made by Geoffroy Zoetardt
       (geoffroy.zoetardt@gmail.com), University Catholic of Louvain-la-neuve
       , also during his master's thesis in 2016.
10 The section about the GlSearchEngine is an adaptation of the code of Camilo
       Rueda and Mauricio Toro Bermudez from Javeriana University.
11 */
12 #ifndef __GELISP_HPP__
13 #define __GELISP_HPP__
14
15 #include <rel.hh>
16 #include "Utils_OM_Object.hpp"
17 #include "OM++_Note.hpp"
18 #include "OM++_Chord.hpp"
19 #include "OM++_ChordSeq.hpp"
20
21 #include <iostream>
22 #include <vector>
23 #include <gecode/kernel.hh>
24 #include <gecode/int.hh>
25 #include <gecode/set.hh>
26 #include <gecode/search.hh>
27 #include <gecode/gist.hh>
28 #include <gecode/driver.hh>
29 #include <gecode/minimodel.hh>
30
```

```
31 namespace GeLisp {
32
33     using namespace std;
34     using namespace MPG;
35
36     class GlIntVar;
37     class GlBoolVar;
38     class GlSetVar;
39     class GlRelVar;
40
41     class GlSearchEngine;
42     class GlBABEngine;
43     class GlDFSEngine;
44
45
46
47     /*
           ************************************************************************
           */
48     /****************************************** GeLisp Space
           ******************************************/
49     /*
           ************************************************************************
           */
50
51     /***************************/
52     /*          GLSPACE        */
53     /***************************/
54     /*
55      * GlSpace is a class extending the notion of Gecode::Space.
56      * The implementation of this class can be found into gelisp3.cpp
57      */
58     class GlSpace : public Gecode::Space {
59         GlSpace(bool share, GlSpace&);
60     public:
61         /// All integer variables
62         std::vector<GlIntVar> iv;
63         /// All Boolean variables
64         std::vector<GlBoolVar> bv;
65         /// All set variables
66         std::vector<GlSetVar> sv;
67         /// All relation variables
68         std::vector<GlRelVar> rv;
69
70         ///engine at next dfs solution in the search tree
71         GlDFSEngine* eng_dfs;
72         ///engine at next bab solution in the search tree
73         GlBABEngine* eng_bab;
74
75         /// Construct an empty space
76         GlSpace(void);
77         /// Destructor
78         ~GlSpace(void);
79
80         /// Create a new integer variable with domain [lb,ub]
81         GlIntVar& newIntVar(int lb, int ub);
82         /// Create a new integer variable from IntSet is
83         GlIntVar& newIntVar(const Gecode::IntSet &d);
84
```

```
85          /// Create a new boolean variable with domain [lb,ub]
86          GlBoolVar& newBoolVar(int lb, int ub);
87
88          /// Create a new set variable from home
89          GlSetVar& newSetVar();
90
91          /// Create a new set variable with bounds and cardinality
92          GlSetVar& newSetVar(int glbMin,int glbMax,int lubMin,int lubMax,
93                              unsigned int cardMin = 0, unsigned int cardMax =
                                    Gecode::Set::Limits::card);
94
95          /// Create a new set variable with bounds(lower bound is an IntSet)
                and cardinality
96          GlSetVar& newSetVar(const Gecode::IntSet& glbD,int lubMin,int lubMax
                ,
97                              unsigned int cardMin = 0,
98                              unsigned int cardMax = Gecode::Set::Limits::card
                                );
99
100         /// Create a new set variable with bounds(upper bound is an IntSet)
                and cardinality
101         GlSetVar&
102         newSetVar(int glbMin,int glbMax,const Gecode::IntSet& lubD,
103                   unsigned int cardMin = 0,
104                   unsigned int cardMax = Gecode::Set::Limits::card);
105
106         /// Create a new set variable with bounds(lower and upper bound is
                an IntSet) and cardinality
107         GlSetVar&
108         newSetVar(const Gecode::IntSet& glbD,const Gecode::IntSet& lubD,
109                   unsigned int cardMin = 0,
110                   unsigned int cardMax = Gecode::Set::Limits::card);
111
112
113         GlRelVar&
114         newRelVar(const Relation& l, const Relation& u);
115
116         /// Perform propagation
117         const char* status(void);
118         /// Print space information
119         void print(std::ostream& os) const;
120         /// Debug
121         const char* debug(void) const;
122         /// Post the branchings
123         void branch(void);
124         /// Run the search for nbSolutions, returns the number of solutions.
                 if nbSolutions = 0, search on all the space.
125         unsigned int runDFSEngine(GlDFSEngine* eng, std::ostream& os, int
                nbSolutions, std::vector<GlSpace*>& solutionsVector);
126         /// Run the search for nbSolutions, returns the number of solutions.
                 if nbSolutions = 0, search on all the space.
127         unsigned int runBABEngine(GlBABEngine* eng, std::ostream& os, int
                nbSolutions, std::vector<GlSpace*>& solutionsVector);
128         /// Returns the number of solutions to the problem
129         const char* runDFS(int nbSolutions, std::vector<GlSpace*>&
                solutionsVector);
130
131         /// Returns the number of solutions to the problem
```

```
132         const char* runBAB(int nbSolutions, std::vector<GlSpace*>&
                solutionsVector);
133         // Returns the number of new solutions to the problem
134         const char* nextDFSSolution(int nbSolutions, std::vector<GlSpace*>&
                solutionsVector);
135         const char* nextBABSolution(int nbSolutions, std::vector<GlSpace*>&
                solutionsVector);
136
137         /// Copy function
138         virtual Gecode::Space* copy(bool share);
139
140         //get the variable reference from the index of iv
141         GlIntVar& getIntVar(int indexInVector);
142
143         //get the variable reference from the index of bv
144         GlBoolVar& getBoolVar(int indexInVector);
145
146         //get the variable reference from the index of sv
147         GlSetVar& getSetVar(int indexInVector);
148
149         //get the variable reference from the index of rv
150         GlRelVar& getRelVar(int indexInVector);
151
152         //get the Branch And Bound engine of the space
153         GlBABEngine* getBABEngine();
154
155         //get the Depth Search First engine of the space
156         GlDFSEngine* getDFSEngine();
157
158         //set the Branch And Bound engine of the space
159         void setBABEngine(GlBABEngine* eng);
160
161         //set the Depth Search First engine of the space
162         void setDFSEngine(GlDFSEngine* eng);
163
164         //create a vector of GlSpace and return a reference on it
165         std::vector<GlSpace*> createGlSpaceVector(void);
166
167
168     };
169
170     /*
            ************************************************************************
            */
171     /****************************************** GeLisp Search Engine
            ******************************************/
172     /*
            ************************************************************************
            */
173
174     /****************************/
175     /*      GLSEARCHENGINES     */
176     /****************************/
177     /*
178      * These classes implement the search objects.
179      * They allow a GlSpace to navigate among all the solutions thanks to
            the next and the hastNext methods.
180      */
181     class GlSearchEngine {
```

```
182     public:
183         virtual GlSpace* next() = 0;
184         virtual ~GlSearchEngine() {};
185         virtual bool hasNext(){
186             if (next()==0)
187                 return false;
188             return true;
189         }
190     };
191 
192     /*
193      * This class implement a search engine for a Depth First Search solver.
194      * More information about the Gecode::Search::Option can be found into
           gecode.org
195      */
196     class GlDFSEngine : public GlSearchEngine {
197     protected:
198         Gecode::DFS<GlSpace> * dfs ;
199     public:
200 
201         /* Construct a DFS engine.
202          * inputs: - spc is the space link to this engine.
203          *         - c_d is the number of commits realize before creation of
                      a clone
204          *         - c_a is the adaptive distance below which a clone is
                     created during the recomputation
205          *         - st is the time in millisecond after which the search is
                      stopped.
206          *
207          */
208         GlDFSEngine( GlSpace* spc, int c_d, int c_a, int st )
209         {
210             Gecode::Search::TimeStop * stopping;
211             if (st == 0)
212                 stopping = (Gecode::Search::TimeStop *)NULL;
213             else
214                 stopping = new Gecode::Search::TimeStop(st);
215 
216             Gecode::Search::Options *opt = new Gecode::Search::Options();
217             opt->c_d = c_d;
218             opt->a_d = c_a;
219             opt->stop = stopping;
220 
221             dfs = new Gecode::DFS<GlSpace>(spc, *opt);
222         }
223 
224         /* Construct a DFS engine.
225          * inputs: - spc is the space link to this engine.
226          *         - c_d is the number of commits realize before creation of
                      a clone
227          *         - c_a is the adaptive distance below which a clone is
                     created during the recomputation
228          *
229          */
230         GlDFSEngine( GlSpace* spc, int c_d, int c_a)
231         {
232             Gecode::Search::Options *opt = new Gecode::Search::Options();
233             opt->c_d = c_d;
234             opt->a_d = c_a;
```

```
235
236             dfs = new Gecode::DFS<GlSpace>(spc, *opt);
237         }
238
239         //Deconstructor
240         virtual ~GlDFSEngine() {delete dfs;};
241
242         //Return the next solution space.
243         virtual GlSpace* next() { return dfs->next(); }
244     };
245
246     /*
247      * This class implement a search engine for a Branch And Bound solver.
248      */
249     class GlBABEngine : public GlSearchEngine {
250     protected:
251         Gecode::BAB<GlSpace>* bab ;
252     public:
253         /* Construct a BAB engine.
254          * inputs: - spc is the space link to this engine.
255          *         - c_d is the number of commits realize before creation of
                        a clone
256          *         - c_a is the adaptive distance below which a clone is
                       created during the recomputation
257          *         - st is the time in millisecond after which the search is
                       stopped.
258          *
259          */
260         GlBABEngine( GlSpace* spc, int c_d, int c_a, int st )
261         {
262             Gecode::Search::TimeStop * stopping;
263             if (st == 0)
264                 stopping = (Gecode::Search::TimeStop *)NULL;
265             else
266                 stopping = new Gecode::Search::TimeStop(st);
267
268             Gecode::Search::Options *opt = new Gecode::Search::Options();
269             opt->c_d = c_d;
270             opt->a_d = c_a;
271             opt->stop = stopping;
272             bab = new Gecode::BAB<GlSpace>(spc, *opt);
273         }
274
275         /* Construct a BAB engine.
276          * inputs: - spc is the space link to this engine.
277          *         - c_d is the number of commits realize before creation of
                        a clone
278          *         - c_a is the adaptive distance below which a clone is
                       created during the recomputation
279          *
280          */
281         GlBABEngine( GlSpace* spc, int c_d, int c_a)
282         {
283             Gecode::Search::Options *opt = new Gecode::Search::Options();
284             opt->c_d = c_d;
285             opt->a_d = c_a;
286             bab = new Gecode::BAB<GlSpace>(spc, *opt);
287         }
288
```

```
289         //Deconstructor
290         virtual ~GlBABEngine() {delete bab;};
291 
292         //Return the next solution space.
293         virtual GlSpace* next() { return bab->next(); }
294     };
295 
296     /*
        ************************************************************************
        */
297     /****************************************** GeLisp Integer
        ******************************************/
298     /*
        ************************************************************************
        */
299 
300     /****************************/
301     /*          GLINTVAR        */
302     /****************************/
303 
304     //This class is used as equivalence to the Gecode::IntVar. "index"
        represent the index of the object in the iv vector of a GlSpace.
305     class GlIntVar: public Gecode::IntVar
306     {
307     private:
308         int index; //index of the variable in iv vector (used to get the
            values solution after search)
309     public:
310 
311         //Empty constructor.
312         GlIntVar(void): Gecode::IntVar(), index(-1){}
313 
314         //Min-max constructor.
315         //lb and ub are respectively the lowest and the highest value
            allowed.
316         GlIntVar(GlSpace& home, int lb, int ub):Gecode::IntVar(home, lb, ub)
            , index(-1){}
317 
318         //IntSet constructor.
319         GlIntVar(GlSpace& home, const Gecode::IntSet &d):Gecode::IntVar(home
            , d), index(-1){}
320 
321         //return the index of the object in the iv vector
322         int getIndexInVector(void){
323             return index;
324         }
325 
326         //set the index of the object in the iv vector
327         void setIndexInVector(int newIndex){
328             index = newIndex;
329         }
330 
331         /*
332          *convert the nsol solutions of the object into a vector of integers
                .
333          *inputs: - solvect is a vector of solution's space.
334          *        - nsol is the number of requested solutions.
335          */
336         vector<int> getIntVarSol(vector<GlSpace*> solvect, int nsol){
```

```
337             cout << endl;
338             cout << "getIntVarSol IN" << endl;
339             std::cout << "solvect size: " << solvect.size() <<endl;
340             cout << "nsol size: " << nsol <<endl;
341             cout << endl;
342             vector<int> out;
343             if (nsol > solvect.size()) {
344                 std::cout << "ERROR: getIntVarSol - number of solutions
                        given as parameter is higher than the possible number of
                         solution." << endl;
345
346             }
347             else{
348                 int realnsol =solvect.size()-nsol;
349                 for (int i = realnsol; i < solvect.size(); i++) {
350                     GlSpace* curr = solvect.at(i);
351                     out.push_back((curr->getIntVar(index)).val());
352                 }
353             }
354             return out;
355         }
356
357     };
358
359     /*******************************/
360     /*          GLINTVARLIST       */
361     /*******************************/
362
363     //This class is used as equivalence to the Gecode::IntVarArgs. The
            vector "index" represent all indexes of the IntVar objects of the
            list in the iv vector of a GlSpace.
364     class GlIntVarList {
365         Gecode::IntVarArgs va;
366         std::vector<int> vectorOfIndex; //will contain the indexes of the
                IntVar objects in the iv vector
367     public:
368
369         //Empty constructor.
370         GlIntVarList(void){}
371
372         //Min-Max constructor
373         //n is the size of the list.
374         //lb and ub are respectively the lowest and the highest value
                allowed.
375         GlIntVarList(GlSpace& home, int n, int min, int max){
376             GlIntVar* tempVar;
377             for (int i = 0; i < n; i++) {
378                 tempVar = &(home.newIntVar(min,max));
379                 va << *tempVar;
380                 vectorOfIndex.push_back(tempVar->getIndexInVector());
381             }
382         }
383
384         //IntSet constructor.
385         GlIntVarList(GlSpace& home, int n, const Gecode::IntSet &d){
386             for (int i = 0; i < n; i++) {
387                 va << home.newIntVar(d);
388             }
389         }
```

```
390
391         //return the size of the list.
392         unsigned int size(void) {
393             return va.size();
394         }
395
396         //add an IntVar to the list
397         void add(GlIntVar& x) {
398             va << x;
399             vectorOfIndex.push_back(x.getIndexInVector());
400         }
401
402         //get the IntVarArgs corresponding to the GlIntVarList
403         const Gecode::IntVarArgs& get(void) {
404             return va;
405         }
406
407         //get the variable at the index i
408         const GlIntVar& getVar(int i) {
409             assert(i >= 0 && i < va.size());
410             GlIntVar& toReturn = ((GlIntVar&) va[i]) ;
411             return toReturn;
412         }
413
414         //return the index in iv of an IntVar in va
415         const int getIndexOfAVarInVector(int i){
416             return vectorOfIndex[i];
417         }
418
419         /*
420          *convert the nsol solutions of the object into a vector of integers
                 ' vectors.
421          *inputs: - solvect is a vector of solution's space.
422          *        - nsol is the number of requested solutions.
423          */
424         vector< vector<int> > getIntVarListSol(vector<GlSpace*> solvect, int
                 nsol){
425             cout << endl;
426             cout << "getIntVarSol IN" << endl;
427             std::cout << "solvect size: " << solvect.size() <<endl;
428             cout << "nsol size: " << nsol <<endl;
429             cout << endl;
430             vector< vector<int> > out;
431             if (nsol > solvect.size()) {
432                 std::cout << "ERROR: getIntVarSol - number of solutions
                        given as parameter is higher than the possible number of
                         solution." << endl;
433
434             }
435             else{
436                 int realnsol =solvect.size()-nsol;
437                 cout << realnsol << endl;
438                 for (int i = realnsol; i < solvect.size(); i++) {
439                     GlSpace* curr = solvect.at(i);
440                     vector<int> vout;
441                     cout << vectorOfIndex.size() << endl;
442                     for (int j = 0; j < vectorOfIndex.size(); j++) {
443                         cout << "vec size: "<<vectorOfIndex.size()<< endl;;
444                         cout << "i: "<<i<<" j: "<<j <<endl;
```

```
445                         cout << "index: "<<vectorOfIndex.at(j)<<endl;
446                         vout.push_back((curr->getIntVar(vectorOfIndex.at(j))
                                ).val());
447                     }
448                     out.push_back(vout);
449                 }
450             }
451             return out;
452         }
453
454     };
455
456     /*****************************/
457     /*          GLINTARGS        */
458     /*****************************/
459     class GlIntList {
460
461     private:
462         Gecode::IntArgs va;
463     public:
464
465         GlIntList(int n):va(n){}
466
467         GlIntList(int n, int defaultValue):va(n)
468         {
469             for(int i = 0; i < n; i++)
470             {
471                 va[i] = defaultValue;
472
473             }
474
475         }
476
477         GlIntList(int n,int start, int inc):va(n)
478         {
479             for(int i = 0; i < n; i++, start+=inc)
480             {
481                 va[i] = start;
482
483             }
484         }
485         const Gecode::IntArgs& get(void) {
486             return va;
487         }
488
489         unsigned int size(void) {
490             return va.size();
491         }
492
493         const int getVar(int i) {
494             assert(i >= 0 && i < va.size());
495             return va[i];
496
497         }
498         const void setVar(int i, int value) {
499             assert(i >= 0 && i < va.size());
500             va[i] = value ;
501         }
502
```

```
503         const char* print(){
504 
505             std::stringstream out;
506             out << "[ " ;
507             for(int i = 0; i < va.size() - 1; i++)
508             {
509                 out << va[i] << ", " ;
510             }
511             out << va[va.size() - 1] << "]" << std::endl;
512             return out.str().c_str();
513         }
514     };
515 
516     /*******************************/
517     /*          GLINTSETARGS       */
518     /*******************************/
519     class GlIntSetList {
520 
521     private:
522         Gecode::IntSetArgs isa;
523     public:
524         GlIntSetList(int n,int min, int max):isa(n)
525         {
526             for(int i = 0; i < n; i++)
527             {
528                 isa[i] = Gecode::IntSet(min,max);
529 
530             }
531         }
532         const Gecode::IntSetArgs& get(void) {
533             return isa;
534         }
535 
536         const Gecode::IntSet& getElement(int i) {
537             return isa[i];
538         }
539 
540     };
541 
542 
543 
544     /*
        ******************************************************************************
        */
545     /****************************************** GeLisp Boolean
        ******************************************/
546     /*
        ******************************************************************************
        */
547 
548     /*****************************/
549     /*          GLBOOLVAR        */
550     /*****************************/
551     //This class is used as equivalence to the Gecode::BoolVar. "index"
        represent the index of the object in the bv vector of a GlSpace.
552     class GlBoolVar: public Gecode::BoolVar
553     {
554         int index; //index of the variable in bv vector (used to get the
            values solution after search)
```

```
555     public:
556
557         //Empty constructor
558         GlBoolVar(void):Gecode::BoolVar(),index(-1){}
559
560         //Min-max constructor
561         GlBoolVar(GlSpace& home, int lb, int ub):Gecode::BoolVar(home, lb,
                ub), index(-1){}
562
563         //return the index of the variable
564         int getIndexInVector(void)
565         {
566             return index;
567         }
568
569         //set the index of the variable
570         void setIndexInVector(int newIndex)
571         {
572             index = newIndex;
573         }
574
575         /*
576          *convert the nsol solutions of the object into a vector of integers
                .
577          *inputs: - solvect is a vector of solution's space.
578          *        - nsol is the number of requested solutions.
579          */
580         vector<bool> getBoolVarSol(vector<GlSpace*> solvect, int nsol){
581             cout << endl;
582             cout << "getIntVarSol IN" << endl;
583             std::cout << "solvect size: " << solvect.size() <<endl;
584             cout << "nsol size: " << nsol <<endl;
585             cout << endl;
586             vector<bool> out;
587             if (nsol > solvect.size()) {
588                 std::cout << "ERROR: getBoolVarSol - number of solutions
                        given as parameter is higher than the possible number of
                         solution." << endl;
589
590             }
591             else{
592                 int realnsol = solvect.size()-nsol;
593                 for (int i = realnsol; i < solvect.size(); i++) {
594                     GlSpace* curr = solvect.at(i);
595                     out.push_back((curr->getBoolVar(index)).val());
596                 }
597             }
598             return out;
599         }
600
601     };
602
603     /*********************************/
604     /*          GLBOOLVARLIST        */
605     /*********************************/
606
607     //This class is used as equivalence to the Gecode::BoolVarArgs. The
            vector "index" represent all indexes of the BoolVar objects of the
            list in the bv vector of a GlSpace.
```

```
608     class GlBoolVarList {
609         Gecode::BoolVarArgs ba;
610         std::vector<int> vectorOfIndex; //will contain the indexes of the
                BoolVar objects in the bv vector
611     public:
612
613         //Empty constructor
614         GlBoolVarList(void)  {}
615
616         //Min-max constructor
617         GlBoolVarList(GlSpace& home, int n, int min, int max)  {
618             GlBoolVar* tempVar;
619             for (int i = 0; i < n; i++) {
620                 tempVar = &(home.newBoolVar(min,max));
621                 ba << *tempVar;
622                 vectorOfIndex.push_back(tempVar->getIndexInVector());
623             }
624         }
625
626         //Size of the list
627         unsigned int size(void) {
628             return ba.size();
629         }
630
631         //add a variable to the list
632         void add(GlBoolVar& x) {
633             ba << x;
634             vectorOfIndex.push_back(x.getIndexInVector());
635         }
636
637         //Get the Gecode::BoolVarArgs of the object
638         const Gecode::BoolVarArgs& get(void) {
639             return ba;
640         }
641
642         //Return the variable at the index i
643         const GlBoolVar& getVar(int i) {
644             assert(i >= 0 && i < ba.size());
645             GlBoolVar& toReturn = ((GlBoolVar&) ba[i]) ;
646             return toReturn;
647
648         }
649         /*
650          *convert the nsol solutions of the object into a vector of integers
                ' vectors.
651          *inputs: - solvect is a vector of solution's space.
652          *        - nsol is the number of requested solutions.
653          */
654         vector< vector<bool> > getBoolVarListSol(vector<GlSpace*> solvect,
                int nsol){
655             cout << endl;
656             cout << "getBoolVarSol IN" << endl;
657             std::cout << "solvect size: " << solvect.size() <<endl;
658             cout << "nsol size: " << nsol <<endl;
659             cout << endl;
660             vector< vector<bool> > out;
661             if (nsol > solvect.size()) {
662                 std::cout << "ERROR: getBoolVarListSol - number of solutions
                        given as parameter is higher than the possible number
```

```
                    of solution." << endl;
663
664             }
665             else{
666                 int realnsol =solvect.size()-nsol;
667                 cout << realnsol << endl;
668                 for (int i = realnsol; i < solvect.size(); i++) {
669                     GlSpace* curr = solvect.at(i);
670                     vector<bool> vout;
671                     cout << vectorOfIndex.size() << endl;
672                     for (int j = 0; j < vectorOfIndex.size(); j++) {
673                         cout << "vec size: "<<vectorOfIndex.size()<< endl;;
674                         cout << "i: "<<i<<" j: "<<j <<endl;
675                         cout << "index: "<<vectorOfIndex.at(j)<<endl;
676                         vout.push_back((curr->getBoolVar(vectorOfIndex.at(j)
                                ))).val());
677                     }
678                     out.push_back(vout);
679                 }
680             }
681             return out;
682         }
683     };
684
685
686     /*
            ********************************************************************************
            */
687     /****************************************** GeLisp Set
            ***********************************************/
688     /*
            ********************************************************************************
            */
689
690     /*****************************/
691     /*          GLSETVAR         */
692     /*****************************/
693     //classe utilisee a la place de SetVar car on a besoin de connaitre l'
            indice de l'objet dans le vecteur sv
694     class GlSetVar: public Gecode::SetVar
695     {
696         int index; //index of the variable in sv vector (used to get the
                values solution after search)
697     public:
698
699         /* CONSTRUCTORS BEGINNING */
700         GlSetVar(void):Gecode::SetVar()
701         {
702             index = -1;
703         }
704
705         GlSetVar(GlSpace& home):Gecode::SetVar(home)
706         {
707             index = -1;
708         }
709
710         GlSetVar(GlSpace& home, int glbMin,int glbMax,int lubMin,int lubMax,
                unsigned int cardMin = 0, unsigned int cardMax = Gecode::Set::
               Limits::card):Gecode::SetVar(home, glbMin, glbMax, lubMin,
```

```
            lubMax, cardMin, cardMax)
711         {
712             index = -1;
713         }
714
715         GlSetVar(GlSpace& home, const Gecode::IntSet& glbD,int lubMin,int
                lubMax, unsigned int cardMin = 0,
716                  unsigned int cardMax = Gecode::Set::Limits::card):Gecode::
                     SetVar(home, glbD, lubMin, lubMax, cardMin, cardMax)
717         {
718             index = -1;
719         }
720
721         GlSetVar(GlSpace& home, int glbMin,int glbMax,const Gecode::IntSet&
                lubD, unsigned int cardMin = 0,
722                  unsigned int cardMax = Gecode::Set::Limits::card):Gecode::
                     SetVar(home, glbMin, glbMax, lubD, cardMin, cardMax)
723         {
724             index = -1   ;
725         }
726
727         GlSetVar(GlSpace& home, const Gecode::IntSet& glbD,const Gecode::
                IntSet& lubD, unsigned int cardMin = 0,
728                  unsigned int cardMax = Gecode::Set::Limits::card):Gecode::
                     SetVar(home, glbD, lubD, cardMin, cardMax)
729         {
730             index = -1;
731         }
732         /* CONSTRUCTORS END */
733
734         //Get the index of the object
735         int getIndexInVector(void)
736         {
737             return index;
738         }
739
740         //Set the index of the object
741         void setIndexInVector(int newIndex)
742         {
743             index = newIndex;
744         }
745
746     };
747
748
749     /*******************************/
750     /*          GLSETVARLIST       */
751     /*******************************/
752
753     class GlSetVarList {
754         Gecode::SetVarArgs sa;
755         std::vector<int> vectorOfIndex; //will contain the indexes of the
                SetVar objects in the sv vector
756     public:
757
758         /* CONSTRUCTORS BEGINNING */
759
760         GlSetVarList(GlSpace& home, int n)
761         {
```

```
762             GlSetVar* tempVar;
763             for (int i = 0; i < n; i++)
764             {
765                 tempVar = &(home.newSetVar());
766                 sa << *tempVar;
767                 vectorOfIndex.push_back(tempVar->getIndexInVector());
768             }
769         }
770
771         GlSetVarList(GlSpace& home, int n, int glbMin,int glbMax,int lubMin,
                int lubMax,
772                      unsigned int cardMin = 0, unsigned int cardMax = Gecode
                         ::Set::Limits::card)
773         {
774             GlSetVar* tempVar;
775             for (int i = 0; i < n; i++)
776             {
777                 tempVar = &(home.newSetVar(glbMin, glbMax, lubMin, lubMax,
                        cardMin, cardMax));
778                 sa << *tempVar;
779                 vectorOfIndex.push_back(tempVar->getIndexInVector());
780             }
781         }
782
783         GlSetVarList(GlSpace& home, int n, const Gecode::IntSet& glbD, int
                lubMin,int lubMax,
784                      unsigned int cardMin = 0, unsigned int cardMax = Gecode
                         ::Set::Limits::card)
785         {
786             GlSetVar* tempVar;
787             for (int i = 0; i < n; i++)
788             {
789                 tempVar = &(home.newSetVar(glbD, lubMin, lubMax, cardMin,
                        cardMax));
790                 sa << *tempVar;
791                 vectorOfIndex.push_back(tempVar->getIndexInVector());
792             }
793         }
794
795         GlSetVarList(GlSpace& home, int n, int glbMin,int glbMax, const
                Gecode::IntSet& lubD,
796                      unsigned int cardMin = 0, unsigned int cardMax = Gecode
                         ::Set::Limits::card)
797         {
798             GlSetVar* tempVar;
799             for (int i = 0; i < n; i++)
800             {
801                 tempVar = &(home.newSetVar(glbMin, glbMax, lubD, cardMin,
                        cardMax));
802                 sa << *tempVar;
803                 vectorOfIndex.push_back(tempVar->getIndexInVector());
804             }
805         }
806
807         GlSetVarList(GlSpace& home, int n, const Gecode::IntSet& glbD, const
                 Gecode::IntSet& lubD,
808                      unsigned int cardMin = 0, unsigned int cardMax = Gecode
                         ::Set::Limits::card)
809         {
```

```
810             GlSetVar* tempVar;
811             for (int i = 0; i < n; i++)
812             {
813                 tempVar = &(home.newSetVar(glbD, lubD, cardMin, cardMax));
814                 sa << *tempVar;
815                 vectorOfIndex.push_back(tempVar->getIndexInVector());
816             }
817         }
818         /* CONSTRUCTOR END */
819 
820         //Return the size of the list
821         unsigned int size(void) {
822             return sa.size();
823         }
824 
825         //Add a GlSetVar x to the GlSetVarList
826         void add(GlSetVar& x) {
827             sa << x;
828             vectorOfIndex.push_back(x.getIndexInVector());
829         }
830 
831         //Return the list.
832         const Gecode::SetVarArgs& get(void) {
833             return sa;
834         }
835 
836         //Get the SetVar at the index i
837         const Gecode::SetVar& getVar(int i) {
838             assert(i >= 0 && i < sa.size());
839             GlSetVar& toReturn = ((GlSetVar&) sa[i]) ;
840             return toReturn;
841         }
842 
843         //return the index in sv of an IntVar in sa
844         const int getIndexOfAVarInVector(int i)
845         {
846             return vectorOfIndex[i];
847         }
848     };
849 
850 
851     /*
            ********************************************************************************
            */
852     /****************************************** GeLisp RelVar
            ********************************************/
853     /*
            ********************************************************************************
            */
854 
855 
856     //This class is used as equivalence to the RelVar. The RelVar are
            relational variables from the thesis of Gustavo Gutierrez.
857     //The vector "index" represent the indexes of the RelVar objects in the
            rv vector of a GlSpace.
858     class GlRelVar: public RelVar
859     {
860         int index;
861     public:
```

```
862
863         //Empty constructor.
864         GlRelVar(void):RelVar(),index(-1){}
865
866         //Classic constructor.
867         GlRelVar(GlSpace& home, const Relation& l, const Relation& u) :
                RelVar(home,l,u), index(-1){}
868
869         //return the index of the object
870         int getIndexInVector(void){
871             return index;
872         }
873
874         //set the index of the object
875         void setIndexInVector(int newIndex){
876             index = newIndex;
877         }
878     };
879
880     //This class is used as equivalence to the RelVarArgs.
881     //The vector "index" represent all indexes of the RelVar objects of the
            list in the rv vector of a GlSpace.
882     class GlRelVarList {
883         RelVarArgs cpa;
884         std::vector<int> vectorOfIndex; //will contain the indexes of the
                IntVar objects in the iv vector
885     public:
886
887         //classic constructor
888         GlRelVarList(GlSpace& home, int n, const Relation& l, const Relation
                & u)
889         {
890             GlRelVar* tempVar;
891             for (int i = 0; i < n; i++) {
892                 tempVar = &(home.newRelVar(l,u));
893                 cpa << *tempVar;
894                 vectorOfIndex.push_back(tempVar->getIndexInVector());
895             }
896         }
897
898         //return the size of the list
899         unsigned int size(void) {
900             return cpa.size();
901         }
902
903         //add a RelVar to the list
904         void add(GlRelVar& x) {
905             cpa << x;
906             vectorOfIndex.push_back(x.getIndexInVector());
907         }
908
909         //Get the list
910         const RelVarArgs& get(void) {
911             return cpa;
912         }
913
914         //Get the RelVar at the index i
915         const GlRelVar& getVar(int i) {
916             assert(i >= 0 && i < cpa.size());
```

```
917             GlRelVar& toReturn = ((GlRelVar&) cpa[i]) ;
918             return toReturn;
919         }
920 
921         //return the index in rv of an RelVar in cpa
922         const int getIndexOfAVarInVector(int i)
923         {
924             return vectorOfIndex[i];
925         }
926 
927     };
```

fonctions import directly from Gecode

```
1  //
2  //  constraints.hpp
3  //
4  // This file contains the set of functions that are direclty imported into
       GeLisp.
5  // It is inspired from gecheader.hpp from the previous version of GeLisp
       made by Sascha Van Cauwelaert.
6  //
7 
8  #ifndef constraints_hpp
9  #define constraints_hpp
10 
11 class IntSet {
12     public:
13     /// \name Constructors and initialization
14     //@{
15     /// Initialize as empty set
16     IntSet(void);
17     /** \brief Initialize as range with minimum \a n and maximum \a m
18      *
19      * Note that the set is empty if \a n is larger than \a m
20      */
21     IntSet(int n, int m);
22     /// Initialize with \a n integers from array \a r
23     IntSet(const int r[],    int n);
24     /** \brief Initialize with \a n ranges from array \a r
25      *
26      * For position \a i in the array \a r, the minimum is \a r[\a i][0]
27      * and the maximum is \a r[\a i][1].
28      */
29     IntSet(const int r[][2], int n);
30     /// Initialize with range iterator \a i
31     template<class I>
32     explicit IntSet(I& i);
33 #ifdef __INTEL_COMPILER
34     /// Initialize with integer set \a s
35     IntSet(const IntSet& s);
36     /// Initialize with integer set \a s
37     IntSet(IntSet& s);
38     /// Initialize with integers \a i
39     IntSet(const PrimArgArray<int>& i);
40     /// Initialize with integers \a i
41     IntSet(PrimArgArray<int>& i);
42 #endif
```

```
43      //@}
44
45      /// \name Range access
46      //@{
47      /// Return number of ranges of the specification
48      int ranges(void) const;
49      /// Return minimum of range at position \a i
50      int min(int i) const;
51      /// Return maximum of range at position \a i
52      int max(int i) const;
53      /// Return width of range at position \a i
54      unsigned int width(int i) const;
55      //@}
56
57      /// \name Entire set access
58      //@{
59      /// Return whether \a n is included in the set
60      bool in(int n) const;
61      /// Return size (cardinality) of set
62      unsigned int size(void) const;
63      /// Return width of set (distance between maximum and minimum)
64      unsigned int width(void) const;
65      /// Return minimum of entire set
66      int min(void) const;
67      /// Return maximum of entire set
68      int max(void) const;
69      //@}
70
71      /// \name Predefined value
72      //@{
73      /// Empty set
74      static const IntSet empty;
75      //@}
76  };
77
78  // Val.hh
79  inline IntValBranch
80  INT_VAL_MIN(void) {
81      return IntValBranch(IntValBranch::SEL_MIN);
82  }
83
84  inline IntValBranch
85  INT_VAL_MED(void) {
86      return IntValBranch(IntValBranch::SEL_MED);
87  }
88
89  inline IntValBranch
90  INT_VAL_MAX(void) {
91      return IntValBranch(IntValBranch::SEL_MAX);
92  }
93
94  inline IntValBranch
95  INT_VAL_RND(Rnd r) {
96      return IntValBranch(r);
97  }
98
99  inline IntValBranch
100 INT_VAL_SPLIT_MIN(void) {
101     return IntValBranch(IntValBranch::SEL_SPLIT_MIN);
```

```
102 }
103
104 inline IntValBranch
105 INT_VAL_SPLIT_MAX(void) {
106     return IntValBranch(IntValBranch::SEL_SPLIT_MAX);
107 }
108
109 inline IntValBranch
110 INT_VAL_RANGE_MIN(void) {
111     return IntValBranch(IntValBranch::SEL_RANGE_MIN);
112 }
113
114 inline IntValBranch
115 INT_VAL_RANGE_MAX(void) {
116     return IntValBranch(IntValBranch::SEL_RANGE_MAX);
117 }
118
119 inline IntValBranch
120 INT_VAL(IntBranchVal v, IntBranchCommit c) {
121     return IntValBranch(function_cast<VoidFunction>(v),
122                         function_cast<VoidFunction>(c));
123 }
124
125 inline IntValBranch
126 INT_VAL(BoolBranchVal v, BoolBranchCommit c) {
127     return IntValBranch(function_cast<VoidFunction>(v),
128                         function_cast<VoidFunction>(c));
129 }
130
131 inline IntValBranch
132 INT_VALUES_MIN(void) {
133     return IntValBranch(IntValBranch::SEL_VALUES_MIN);
134 }
135
136 inline IntValBranch
137 INT_VALUES_MAX(void) {
138     return IntValBranch(IntValBranch::SEL_VALUES_MAX);
139 }
140
141 inline IntValBranch
142 INT_VAL_NEAR_MIN(IntSharedArray n) {
143     return IntValBranch(IntValBranch::SEL_NEAR_MIN,n);
144 }
145
146 inline IntValBranch
147 INT_VAL_NEAR_MAX(IntSharedArray n) {
148     return IntValBranch(IntValBranch::SEL_NEAR_MAX,n);
149 }
150
151 inline IntValBranch
152 INT_VAL_NEAR_INC(IntSharedArray n) {
153     return IntValBranch(IntValBranch::SEL_NEAR_INC,n);
154 }
155
156 inline IntValBranch
157 INT_VAL_NEAR_DEC(IntSharedArray n) {
158     return IntValBranch(IntValBranch::SEL_NEAR_DEC,n);
159 }
160
```

```
161 // STATISTICS: int-branch
162
163 //var.hh
164
165 inline IntVarBranch
166 INT_VAR_NONE(void) {
167     return IntVarBranch(IntVarBranch::SEL_NONE,NULL);
168 }
169
170 inline IntVarBranch
171 INT_VAR_RND(Rnd r) {
172     return IntVarBranch(r);
173 }
174
175 inline IntVarBranch
176 INT_VAR_MERIT_MIN(IntBranchMerit bm, BranchTbl tbl) {
177     return IntVarBranch(IntVarBranch::SEL_MERIT_MIN,
178                         function_cast<VoidFunction>(bm),tbl);
179 }
180
181 inline IntVarBranch
182 INT_VAR_MERIT_MIN(BoolBranchMerit bm, BranchTbl tbl) {
183     return IntVarBranch(IntVarBranch::SEL_MERIT_MIN,
184                         function_cast<VoidFunction>(bm),tbl);
185 }
186
187 inline IntVarBranch
188 INT_VAR_MERIT_MAX(IntBranchMerit bm, BranchTbl tbl) {
189     return IntVarBranch(IntVarBranch::SEL_MERIT_MAX,
190                         function_cast<VoidFunction>(bm),tbl);
191 }
192
193 inline IntVarBranch
194 INT_VAR_MERIT_MAX(BoolBranchMerit bm, BranchTbl tbl) {
195     return IntVarBranch(IntVarBranch::SEL_MERIT_MAX,
196                         function_cast<VoidFunction>(bm),tbl);
197 }
198
199 inline IntVarBranch
200 INT_VAR_DEGREE_MIN(BranchTbl tbl) {
201     return IntVarBranch(IntVarBranch::SEL_DEGREE_MIN,tbl);
202 }
203
204 inline IntVarBranch
205 INT_VAR_DEGREE_MAX(BranchTbl tbl) {
206     return IntVarBranch(IntVarBranch::SEL_DEGREE_MAX,tbl);
207 }
208
209 inline IntVarBranch
210 INT_VAR_AFC_MIN(double d, BranchTbl tbl) {
211     return IntVarBranch(IntVarBranch::SEL_AFC_MIN,d,tbl);
212 }
213
214 inline IntVarBranch
215 INT_VAR_AFC_MIN(Gecode::IntAFC a, BranchTbl tbl) {
216     return IntVarBranch(IntVarBranch::SEL_AFC_MIN,a,tbl);
217 }
218
219 inline IntVarBranch
```

```
220 INT_VAR_AFC_MAX(double d, BranchTbl tbl) {
221     return IntVarBranch(IntVarBranch::SEL_AFC_MAX,d,tbl);
222 }
223
224 inline IntVarBranch
225 INT_VAR_AFC_MAX(Gecode::IntAFC a, BranchTbl tbl) {
226     return IntVarBranch(IntVarBranch::SEL_AFC_MAX,a,tbl);
227 }
228
229 inline IntVarBranch
230 INT_VAR_ACTIVITY_MIN(double d, BranchTbl tbl) {
231     return IntVarBranch(IntVarBranch::SEL_ACTIVITY_MIN,d,tbl);
232 }
233
234 inline IntVarBranch
235 INT_VAR_ACTIVITY_MIN(IntActivity a, BranchTbl tbl) {
236     return IntVarBranch(IntVarBranch::SEL_ACTIVITY_MIN,a,tbl);
237 }
238
239 inline IntVarBranch
240 INT_VAR_ACTIVITY_MAX(double d, BranchTbl tbl) {
241     return IntVarBranch(IntVarBranch::SEL_ACTIVITY_MAX,d,tbl);
242 }
243
244 inline IntVarBranch
245 INT_VAR_ACTIVITY_MAX(IntActivity a, BranchTbl tbl) {
246     return IntVarBranch(IntVarBranch::SEL_ACTIVITY_MAX,a,tbl);
247 }
248
249 inline IntVarBranch
250 INT_VAR_MIN_MIN(BranchTbl tbl) {
251     return IntVarBranch(IntVarBranch::SEL_MIN_MIN,tbl);
252 }
253
254 inline IntVarBranch
255 INT_VAR_MIN_MAX(BranchTbl tbl) {
256     return IntVarBranch(IntVarBranch::SEL_MIN_MAX,tbl);
257 }
258
259 inline IntVarBranch
260 INT_VAR_MAX_MIN(BranchTbl tbl) {
261     return IntVarBranch(IntVarBranch::SEL_MAX_MIN,tbl);
262 }
263
264 inline IntVarBranch
265 INT_VAR_MAX_MAX(BranchTbl tbl) {
266     return IntVarBranch(IntVarBranch::SEL_MAX_MAX,tbl);
267 }
268
269 inline IntVarBranch
270 INT_VAR_SIZE_MIN(BranchTbl tbl) {
271     return IntVarBranch(IntVarBranch::SEL_SIZE_MIN,tbl);
272 }
273
274 inline IntVarBranch
275 INT_VAR_SIZE_MAX(BranchTbl tbl) {
276     return IntVarBranch(IntVarBranch::SEL_SIZE_MAX,tbl);
277 }
278
```

```
279 inline IntVarBranch
280 INT_VAR_DEGREE_SIZE_MIN(BranchTbl tbl) {
281     return IntVarBranch(IntVarBranch::SEL_DEGREE_SIZE_MIN,tbl);
282 }
283
284 inline IntVarBranch
285 INT_VAR_DEGREE_SIZE_MAX(BranchTbl tbl) {
286     return IntVarBranch(IntVarBranch::SEL_DEGREE_SIZE_MAX,tbl);
287 }
288
289 inline IntVarBranch
290 INT_VAR_AFC_SIZE_MIN(double d, BranchTbl tbl) {
291     return IntVarBranch(IntVarBranch::SEL_AFC_SIZE_MIN,d,tbl);
292 }
293
294 inline IntVarBranch
295 INT_VAR_AFC_SIZE_MIN(Gecode::IntAFC a, BranchTbl tbl) {
296     return IntVarBranch(IntVarBranch::SEL_AFC_SIZE_MIN,a,tbl);
297 }
298
299 inline IntVarBranch
300 INT_VAR_AFC_SIZE_MAX(double d, BranchTbl tbl) {
301     return IntVarBranch(IntVarBranch::SEL_AFC_SIZE_MAX,d,tbl);
302 }
303
304 inline IntVarBranch
305 INT_VAR_AFC_SIZE_MAX(Gecode::IntAFC a, BranchTbl tbl) {
306     return IntVarBranch(IntVarBranch::SEL_AFC_SIZE_MAX,a,tbl);
307 }
308
309 inline IntVarBranch
310 INT_VAR_ACTIVITY_SIZE_MIN(double d, BranchTbl tbl) {
311     return IntVarBranch(IntVarBranch::SEL_ACTIVITY_SIZE_MIN,d,tbl);
312 }
313
314 inline IntVarBranch
315 INT_VAR_ACTIVITY_SIZE_MIN(IntActivity a, BranchTbl tbl) {
316     return IntVarBranch(IntVarBranch::SEL_ACTIVITY_SIZE_MIN,a,tbl);
317 }
318
319 inline IntVarBranch
320 INT_VAR_ACTIVITY_SIZE_MAX(double d, BranchTbl tbl) {
321     return IntVarBranch(IntVarBranch::SEL_ACTIVITY_SIZE_MAX,d,tbl);
322 }
323
324 inline IntVarBranch
325 INT_VAR_ACTIVITY_SIZE_MAX(IntActivity a, BranchTbl tbl) {
326     return IntVarBranch(IntVarBranch::SEL_ACTIVITY_SIZE_MAX,a,tbl);
327 }
328
329 inline IntVarBranch
330 INT_VAR_REGRET_MIN_MIN(BranchTbl tbl) {
331     return IntVarBranch(IntVarBranch::SEL_REGRET_MIN_MIN,tbl);
332 }
333
334 inline IntVarBranch
335 INT_VAR_REGRET_MIN_MAX(BranchTbl tbl) {
336     return IntVarBranch(IntVarBranch::SEL_REGRET_MIN_MAX,tbl);
337 }
```

```
338
339 inline IntVarBranch
340 INT_VAR_REGRET_MAX_MIN(BranchTbl tbl) {
341     return IntVarBranch(IntVarBranch::SEL_REGRET_MAX_MIN, tbl);
342 }
343
344 inline IntVarBranch
345 INT_VAR_REGRET_MAX_MAX(BranchTbl tbl) {
346     return IntVarBranch(IntVarBranch::SEL_REGRET_MAX_MAX, tbl);
347 }
348
349
350 // STATISTICS: int-branch
351
352 enum IntRelType {
353     IRT_EQ, ///< Equality (\f$=\f$)
354     IRT_NQ, ///< Disequality (\f$\neq\f$)
355     IRT_LQ, ///< Less or equal (\f$\leq\f$)
356     IRT_LE, ///< Less (\f$<\f$)
357     IRT_GQ, ///< Greater or equal (\f$\geq\f$)
358     IRT_GR  ///< Greater (\f$>\f$)
359 };
360
361 /**
362  * \brief Consistency levels for integer propagators
363  *
364  * The descriptions are meant to be suggestions. It is not
365  * required that a propagator achieves full domain consistency or
366  * full bounds consistency. It is more like: which level
367  * of consistency comes closest.
368  *
369  * If in the description of a constraint below no consistency level
370  * is mentioned, the propagator for the constraint implements
371  * domain consistency.
372  * \ingroup TaskModelInt
373  */
374 enum IntConLevel {
375     ICL_VAL, ///< Value propagation or consistency (naive)
376     ICL_BND, ///< Bounds propagation or consistency
377     ICL_DOM, ///< Domain propagation or consistency
378     ICL_DEF  ///< The default consistency for a constraint
379 };
380
381 /// Propagates \f$ x=n\f$
382 void dom(Space& home, IntVar x, int n, IntConLevel icl=ICL_DEF);
383
384 void dom(Space& home, const IntVarArgs& x, int n, IntConLevel icl=ICL_DEF);
385
386 /// Propagates \f$ l\leq x\leq m\f$
387 void dom(Space& home, IntVar x, int l, int m, IntConLevel icl=ICL_DEF);
388
389 /// Propagates \f$ l\leq x_i\leq m\f$ for all \f$0\leq i<|x|\f$
390 void dom(Space& home, const IntVarArgs& x, int l, int m, IntConLevel icl=
    ICL_DEF);
391
392 /// Propagates \f$ x\in s \f$
393 void dom(Space& home, IntVar x, const IntSet& s, IntConLevel icl=ICL_DEF);
394
395 /// Propagates \f$ x_i\in s\f$ for all \f$0\leq i<|x|\f$
```

```
396 void dom(Space& home, const IntVarArgs& x, const IntSet& s, IntConLevel icl=
        ICL_DEF);
397
398 /// Post propagator for \f$ (x=n) \Leftrightarrow b\f$
399 void dom(Space& home, IntVar x, int n, BoolVar b, IntConLevel icl=ICL_DEF);
400
401 /// Post propagator for \f$ (l\leq x \leq m) \Leftrightarrow b\f$
402 void dom(Space& home, IntVar x, int l, int m, BoolVar b, IntConLevel icl=
        ICL_DEF);
403 /// Post propagator for \f$ (x \in s) \Leftrightarrow b\f$
404 void dom(Space& home, IntVar x, const IntSet& s, BoolVar b, IntConLevel icl=
        ICL_DEF);
405
406
407 /**
408  * \defgroup TaskModelIntRelInt Simple relation constraints over integer
          variables
409  * \ingroup TaskModelInt
410  */
411 /** \brief Post propagator for \f$ x_0 \sim_r x_1\f$
412  *
413  * Supports both bounds (\a icl = ICL_BND) and
414  * domain consistency (\a icl = ICL_DOM, default).
415  * \ingroup TaskModelIntRelInt
416  */
417 void rel(Space& home, IntVar x0, IntRelType r, IntVar x1, IntConLevel icl=
        ICL_DEF);
418
419 /** \brief Post propagators for \f$ x_i \sim_r y \f$ for all \f$0\leq i</x/\
        f$
420  *
421  * Supports both bounds (\a icl = ICL_BND) and
422  * domain consistency (\a icl = ICL_DOM, default).
423  * \ingroup TaskModelIntRelInt
424  */
425 void rel(Space& home, const IntVarArgs& x, IntRelType r, IntVar y,
        IntConLevel icl=ICL_DEF);
426
427 /** \brief Propagates \f$ x \sim_r c\f$
428  * \ingroup TaskModelIntRelInt
429  */
430 void rel(Space&, IntVar x, IntRelType r, int c, IntConLevel icl=ICL_DEF);
431 /** \brief Propagates \f$ x_i \sim_r c \f$ for all \f$0\leq i</x/\f$
432  * \ingroup TaskModelIntRelInt
433  */
434 void rel(Space& home, const IntVarArgs& x, IntRelType r, int c, IntConLevel
        icl=ICL_DEF);
435
436 /** \brief Post propagator for \f$ (x_0 \sim_r x_1) \Leftrightarrow b\f$
437  *
438  * Supports both bounds (\a icl = ICL_BND) and
439  * domain consistency (\a icl = ICL_DOM, default).
440  * \ingroup TaskModelIntRelInt
441  */
442 void rel(Space& home, IntVar x0, IntRelType r, IntVar x1, BoolVar b,
        IntConLevel icl=ICL_DEF);
443 /** \brief Post propagator for \f$(x \sim_r c) \Leftrightarrow b\f$
444  *
445  * Supports both bounds (\a icl = ICL_BND) and
```

```
446  * domain consistency (\a icl = ICL_DOM, default).
447  * \ingroup TaskModelIntRelInt
448  */
449 void rel(Space& home, IntVar x, IntRelType r, int c, BoolVar b, IntConLevel
        icl=ICL_DEF);
450
451
452 void rel(Space& home, const IntVarArgs& x, IntRelType r, IntConLevel icl=
        ICL_DEF);
453
454 /** \brief Post propagator for relation between \a x and \a y.
455  *
456  * Note that for the inequality relations this corresponds to
457  * the lexical order between \a x and \a y.
458  *
459  * Supports both bounds (\a icl = ICL_BND) and
460  * domain consistency (\a icl = ICL_DOM, default).
461  *
462  * Throws an exception of type Int::ArgumentSizeMismatch, if
463  * \a x and \a y are of different size.
464  * \ingroup TaskModelIntRelInt
465  */
466
467 void rel(Space& home, const IntVarArgs& x, IntRelType r, const IntVarArgs& y
        , IntConLevel icl=ICL_DEF);
468
469
470 /**
471  * \defgroup TaskModelIntRelBool Simple relation constraints over Boolean
        variables
472  * \ingroup TaskModelInt
473  */
474 /** \brief Post propagator for \f$ x_0 \sim_r x_1\f$
475  * \ingroup TaskModelIntRelBool
476  */
477 void rel(Space& home, BoolVar x0, IntRelType r, BoolVar x1, IntConLevel icl=
        ICL_DEF);
478 /** \brief Post propagator for \f$(x_0 \sim_r x_1)\Leftrightarrow b\f$
479  * \ingroup TaskModelIntRelBool
480  */
481 void rel(Space& home, BoolVar x0, IntRelType r, BoolVar x1, BoolVar b,
        IntConLevel icl=ICL_DEF);
482 /** \brief Post propagator for \f$ x_i \sim_r y \f$ for all \f$0\leq i</x/|\
        f$
483  * \ingroup TaskModelIntRelBool
484  */
485 void rel(Space& home, const BoolVarArgs& x, IntRelType r, BoolVar y,
        IntConLevel icl=ICL_DEF);
486 /**
487  * \brief Propagates \f$ x \sim_r n\f$
488  *
489  * Throws an exception of type Int::NotZeroOne, if \a n is neither
490  * 0 or 1.
491  * \ingroup TaskModelIntRelBool
492  */
493 void rel(Space& home, BoolVar x, IntRelType r, int n, IntConLevel icl=
        ICL_DEF);
494 /**
495  * \brief Propagates \f$(x \sim_r n)\Leftrightarrow b\f$
```

```
496  *
497  * Throws an exception of type Int::NotZeroOne, if \a n is neither
498  * 0 or 1.
499  * \ingroup TaskModelIntRelBool
500  */
501 void rel(Space& home, BoolVar x, IntRelType r, int n, BoolVar b, IntConLevel
         icl=ICL_DEF);
502 /**
503  * \brief Propagates \f$ x_i \sim_r n \f$ for all \f$0\leq i</x/|\f$
504  *
505  * Throws an exception of type Int::NotZeroOne, if \a n is neither
506  * 0 or 1.
507  * \ingroup TaskModelIntRelBool
508  */
509 void rel(Space& home, const BoolVarArgs& x, IntRelType r, int n, IntConLevel
         icl=ICL_DEF);
510
511 /** \brief Post propagator for relation between \a x and \a y.
512  *
513  * Note that for the inequality relations this corresponds to
514  * the lexical order between \a x and \a y.
515  *
516  * Throws an exception of type Int::ArgumentSizeMismatch, if
517  * \a x and \a y are of different size.
518  * \ingroup TaskModelIntRelBool
519  */
520 void rel(Space& home, const BoolVarArgs& x, IntRelType r, const BoolVarArgs&
         y, IntConLevel icl=ICL_DEF);
521 /** \brief Post propagator for pairwise relation on \a x.
522  *
523  * States that the elements of \a x are in the following relation:
524  *  - if \a r = IRT_EQ, then all elements of \a x must be equal.
525  *  - if \a r = IRT_LE, \a r = IRT_LQ, \a r = IRT_GR, or \a r = IRT_GQ,
526  *    then the elements of \a x are ordered with respt to \a r.
527  *  - if \a r = IRT_NQ, then all elements of \a x must be pairwise
528  *    distinct (corresponds to the distinct constraint).
529  *
530  * \ingroup TaskModelIntRelBool
531  */
532 void rel(Space& home, const BoolVarArgs& x, IntRelType r, IntConLevel icl=
         ICL_DEF);
533 /** \brief Post propagator for Boolean operation on \a x0 and \a x1
534  *
535  * Posts propagator for \f$ x_0 \diamond_{\mathit{o}} x_1 = x_2\f$
536  * \ingroup TaskModelIntRelBool
537  */
538 void rel(Space& home, BoolVar x0, BoolOpType o, BoolVar x1, BoolVar x2,
         IntConLevel icl=ICL_DEF);
539 /** \brief Post propagator for Boolean operation on \a x0 and \a x1
540  *
541  * Posts propagator for \f$ x_0 \diamond_{\mathit{o}} x_1 = n\f$
542  *
543  * Throws an exception of type Int::NotZeroOne, if \a n is neither
544  * 0 or 1.
545  * \ingroup TaskModelIntRelBool
546  */
547 void rel(Space& home, BoolVar x0, BoolOpType o, BoolVar x1, int n,
         IntConLevel icl=ICL_DEF);
548 /** \brief Post propagator for Boolean operation on \a x
```

```
549  *
550  * Posts propagator for \f$ x_0 \diamond_{\mathit{o}} \cdots
551  * \diamond_{\mathit{o}} x_{|x|-1}= y\f$
552  *
553  * Throws an exception of type Int::TooFewArguments, if \f$|x|<2\f$
554  * and \a o is BOT_IMP, BOT_EQV, or BOT_XOR.
555  * \ingroup TaskModelIntRelBool
556  */
557 void rel(Space& home, BoolOpType o, const BoolVarArgs& x, BoolVar y,
        IntConLevel icl=ICL_DEF);
558 /** \brief Post propagator for Boolean operation on \a x
559  *
560  * Posts propagator for \f$ x_0 \diamond_{\mathit{o}} \cdots
561  * \diamond_{\mathit{o}} x_{|x|-1}= n\f$
562  *
563  * Throws an exception of type Int::NotZeroOne, if \a n is neither
564  * 0 or 1.
565  *
566  * Throws an exception of type Int::TooFewArguments, if \f$|x|<2\f$
567  * and \a o is BOT_IMP, BOT_EQV, or BOT_XOR.
568  * \ingroup TaskModelIntRelBool
569  */
570 void rel(Space& home, BoolOpType o, const BoolVarArgs& x, int n, IntConLevel
        icl=ICL_DEF);
571 /** \brief Post propagator for Boolean clause with positive variables \a x
        and negative variables \a y
572  *
573  * Posts propagator for \f$ x_0 \diamond_{\mathit{o}} \cdots
574  * \diamond_{\mathit{o}} x_{|x|-1} \diamond_{\mathit{o}} \neg y_0
575  * \diamond_{\mathit{o}} \cdots \diamond_{\mathit{o}} \neg y_{|y|-1}= z\f$
576  *
577  * Throws an exception of type Int::IllegalOperation, if \a o is different
578  * from BOT_AND or BOT_OR.
579  * \ingroup TaskModelIntRelBool
580  */
581 void clause(Space& home, BoolOpType o, const BoolVarArgs& x, const
        BoolVarArgs& y, BoolVar z, IntConLevel icl=ICL_DEF);
582 /** \brief Post propagator for Boolean clause with positive variables \a x
        and negative variables \a y
583  *
584  * Posts propagator for \f$ x_0 \diamond_{\mathit{o}} \cdots
585  * \diamond_{\mathit{o}} x_{|x|-1} \diamond_{\mathit{o}} \neg y_0
586  * \diamond_{\mathit{o}} \cdots \diamond_{\mathit{o}} \neg y_{|y|-1}= n\f$
587  *
588  * Throws an exception of type Int::NotZeroOne, if \a n is neither
589  * 0 or 1.
590  *
591  * Throws an exception of type Int::IllegalOperation, if \a o is different
592  * from BOT_AND or BOT_OR.
593  * \ingroup TaskModelIntRelBool
594  */
595 void clause(Space& home, BoolOpType o, const BoolVarArgs& x, const
        BoolVarArgs& y, int n, IntConLevel icl=ICL_DEF);
596
597
598 /**
599  * \defgroup TaskModelIntElement Element constraints
600  * \ingroup TaskModelInt
601  */
```

```
602
603 //@{
604 /// Arrays of integers that can be shared among several element constraints
605 //typedef SharedArray<int> IntSharedArray;
606 /** \brief Post propagator for \f$ n_{x_0}=x_1\f$
607  *
608  *  Throws an exception of type Int::OutOfLimits, if
609  *  the integers in \a n exceed the limits in Int::Limits.
610  */
611 //void element(Space& home, IntSharedArray n, IntVar x0, IntVar x1,
        IntConLevel icl=ICL_DEF);
612 /** \brief Post propagator for \f$ n_{x_0}=x_1\f$
613  *
614  *  Throws an exception of type Int::OutOfLimits, if
615  *  the integers in \a n exceed the limits in Int::Limits.
616  */
617 /* void
618  element(Space& home, IntSharedArray n, IntVar x0, BoolVar x1,
619  IntConLevel icl=ICL_DEF);*/
620 /** \brief Post propagator for \f$ n_{x_0}=x_1\f$
621  *
622  *  Throws an exception of type Int::OutOfLimits, if
623  *  the integers in \a n exceed the limits in Int::Limits.
624  */
625 //void element(Space& home, IntSharedArray n, IntVar x0, int x1,IntConLevel
        icl=ICL_DEF);
626
627 /** \brief Post propagator for \f$ x_{y_0}=y_1\f$
628  *
629  * Supports both bounds (\a icl = ICL_BND) and
630  * domain consistency (\a icl = ICL_DOM, default).
631  */
632 void element(Space& home, const IntVarArgs& x, IntVar y0, IntVar y1,
        IntConLevel icl=ICL_DEF);
633 /** \brief Post propagator for \f$ x_{y_0}=y_1\f$
634  *
635  * Supports both bounds (\a icl = ICL_BND) and
636  * domain consistency (\a icl = ICL_DOM, default).
637  */
638 void element(Space& home, const IntVarArgs& x, IntVar y0, int y1,
        IntConLevel icl=ICL_DEF);
639 /// Post propagator for \f$ x_{y_0}=y_1\f$
640 void element(Space& home, const BoolVarArgs& x, IntVar y0, BoolVar y1,
        IntConLevel icl=ICL_DEF);
641 /// Post propagator for \f$ x_{y_0}=y_1\f$
642 void element(Space& home, const BoolVarArgs& x, IntVar y0, int y1,
        IntConLevel icl=ICL_DEF);
643
644 /** \brief Post propagator for \f$ a_{x+w\cdot y}=z\f$
645  *
646  * If \a a is regarded as a two-dimensional array in row-major
647  * order of width \a w and height \a h, then \a z is constrained
648  * to be the element in column \a x and row \a y.
649  *
650  * Throws an exception of type Int::OutOfLimits, if
651  * the integers in \a n exceed the limits in Int::Limits.
652  *
653  * Throws an exception of type Int::ArgumentSizeMismatch, if
654  * \f$ w\cdot h\neq|a|\f$.
```

```
655  */
656 /*void
657  element(Space& home, IntSharedArray a,
658  IntVar x, int w, IntVar y, int h, IntVar z,
659  IntConLevel icl=ICL_DEF);*/
660 /** \brief Post propagator for \f$ a_{x+w\cdot y}=z\f$
661  *
662  * If \a a is regarded as a two-dimensional array in row-major
663  * order of width \a w and height \a h, then \a z is constrained
664  * to be the element in column \a x and row \a y.
665  *
666  * Throws an exception of type Int::OutOfLimits, if
667  * the integers in \a n exceed the limits in Int::Limits.
668  *
669  * Throws an exception of type Int::ArgumentSizeMismatch, if
670  * \f$ w\cdot h\neq|a|\f$.
671  */
672 /* void
673  element(Space& home, IntSharedArray a,
674  IntVar x, int w, IntVar y, int h, BoolVar z,
675  IntConLevel icl=ICL_DEF);*/
676 /** \brief Post propagator for \f$ a_{x+w\cdot y}=z\f$
677  *
678  * If \a a is regarded as a two-dimensional array in row-major
679  * order of width \a w and height \a h, then \a z is constrained
680  * to be the element in column \a x and row \a y.
681  *
682  * Supports both bounds (\a icl = ICL_BND) and
683  * domain consistency (\a icl = ICL_DOM, default).
684  *
685  * Throws an exception of type Int::OutOfLimits, if
686  * the integers in \a n exceed the limits in Int::Limits.
687  *
688  * Throws an exception of type Int::ArgumentSizeMismatch, if
689  * \f$ w\cdot h\neq|a|\f$.
690  */
691 void element(Space& home, const IntVarArgs& a, IntVar x, int w, IntVar y,
        int h, IntVar z, IntConLevel icl=ICL_DEF);
692 /** \brief Post propagator for \f$ a_{x+w\cdot y}=z\f$
693  *
694  * If \a a is regarded as a two-dimensional array in row-major
695  * order of width \a w and height \a h, then \a z is constrained
696  * to be the element in column \a x and row \a y.
697  *
698  * Throws an exception of type Int::OutOfLimits, if
699  * the integers in \a n exceed the limits in Int::Limits.
700  *
701  * Throws an exception of type Int::ArgumentSizeMismatch, if
702  * \f$ w\cdot h\neq|a|\f$.
703  */
704 void element(Space& home, const BoolVarArgs& a, IntVar x, int w, IntVar y,
        int h, BoolVar z, IntConLevel icl=ICL_DEF);
705 //@}
706
707
708
709
710 void distinct(Space& home, const IntVarArgs& x, IntConLevel icl=ICL_DEF);
711
```

```
712 void distinct(Space& home, const IntArgs& n, const IntVarArgs& x,
        IntConLevel icl=ICL_DEF);
713
714 /**
715  * \defgroup TaskModelIntChannel Channel constraints
716  * \ingroup TaskModelInt
717  */
718
719 //@{
720 /** \brief Post propagator for \f$ x_i = j \leftrightarrow y_j=i \f$ for all \
        f$0 \leq i</x/\f$
721  *
722  * \li Supports domain consistency (\a icl = ICL_DOM) and value
723  *     propagation (all other values for \a icl).
724  * \li Throws an exception of type Int::ArgumentSizeMismatch, if
725  *     \a x and \a y are of different size.
726  * \li Throws an exception of type Int::ArgumentSame, if \a x or
727  *     \a y contain the same unassigned variable multiply. Note that a
728  *     variable can occur in both \a x and \a y, but not more than
729  *     once in either \a x or \a y.
730  */
731 void channel(Space& home, const IntVarArgs& x, const IntVarArgs& y,
        IntConLevel icl=ICL_DEF);
732
733 /** \brief Post propagator for \f$ x_i - \mathit{xoff} = j \leftrightarrow
        y_j - \mathit{yoff} = i \f$ for all \f$0 \leq i</x/\f$
734  *
735  * \li Supports domain consistency (\a icl = ICL_DOM) and value
736  *     propagation (all other values for \a icl).
737  * \li Throws an exception of type Int::ArgumentSizeMismatch, if
738  *     \a x and \a y are of different size.
739  * \li Throws an exception of type Int::ArgumentSame, if \a x or
740  *     \a y contain the same unassigned variable multiply. Note that a
741  *     variable can occur in both \a x and \a y, but not more than
742  *     once in either \a x or \a y.
743  * \li Throws an exception of type Int::OutOfLimits, if \a xoff or
744  *     \a yoff are negative.
745  */
746 void channel(Space& home, const IntVarArgs& x, int xoff, const IntVarArgs& y
        , int yoff, IntConLevel icl=ICL_DEF);
747
748 /// Post propagator for channeling a Boolean and an integer variable \f$ x_0
        = x_1 \f$
749 void channel(Space& home, BoolVar x0, IntVar x1, IntConLevel icl=ICL_DEF);
750 /// Post propagator for channeling an integer and a Boolean variable \f$ x_0
        = x_1 \f$
751 /*forceinline*/ void channel(Space& home, IntVar x0, BoolVar x1, IntConLevel
        icl=ICL_DEF)
752 {
753     channel(home,x1,x0,icl);
754 }
755 /** \brief Post propagator for channeling Boolean and integer variables \f$
        x_i = 1 \leftrightarrow y=i+o \f$
756  *
757  * Throws an exception of type Int::ArgumentSame, if \a x
758  * contains the same unassigned variable multiply.
759  */
760 void channel(Space& home, const BoolVarArgs& x, IntVar y, int o=0,
        IntConLevel icl=ICL_DEF);
```

```
761 //@}
762
763 /**
764  * \defgroup TaskModelIntSorted Sorted constraints
765  *
766  * All sorted constraints support bounds consistency.
767  *
768  * \ingroup TaskModelInt
769  */
770 //@{
771 /**
772  * \brief Post propagator that \a y is \a x sorted in increasing order
773  *
774  * Might throw the following exceptions:
775  *  - Int::ArgumentSizeMismatch, if \a x and \a y differ in size.
776  *  - Int::ArgumentSame, if \a x or \a y contain
777  *              shared unassigned variables.
778  */
779 void sorted(Space& home, const IntVarArgs& x, const IntVarArgs& y,
    IntConLevel icl=ICL_DEF);
780
781 /**
782  * \brief Post propagator that \a y is \a x sorted in increasing order
783  *
784  * The values in \a z describe the sorting permutation, that is
785  * \f$\forall i\in\{0,\dots,|x|-1\}: x_i=y_{z_i} \f$.
786  *
787  * Might throw the following exceptions:
788  *  - Int::ArgumentSizeMismatch, if \a x and \a y differ in size.
789  *  - Int::ArgumentSame, if \a x or \a y contain
790  *              shared unassigned variables.
791  */
792 void sorted(Space& home, const IntVarArgs& x, const IntVarArgs& y, const
    IntVarArgs& z, IntConLevel icl=ICL_DEF);
793 //@}
794
795 /**
796  * \defgroup TaskModelIntCount Counting constraints
797  * \ingroup TaskModelInt
798  *
799  *  \note
800  *    Domain consistency on the extended cardinality variables of
801  *    the Global Cardinality Propagator is only obtained if they are bounds
802  *    consistent, otherwise the problem of enforcing domain consistency
803  *    on the cardinality variables is NP-complete as proved by
804  *    Qumiper et. al. in
805  *    ''Improved Algorithms for the Global Cardinality Constraint''.
806  */
807
808 //@{
809 /** \brief Post propagator for \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=n\}\
    sim_r m\f$
810  *
811  * Supports domain consistent propagation only.
812  */
813 void count(Space& home, const IntVarArgs& x, int n, IntRelType r, int m,
    IntConLevel icl=ICL_DEF);
814 /** \brief Post propagator for \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=y\}\
    sim_r m\f$
```

```
815  *
816  * Supports domain consistent propagation only.
817  */
818 void count(Space& home, const IntVarArgs& x, IntVar y, IntRelType r, int m,
        IntConLevel icl=ICL_DEF);
819 /** \brief Post propagator for \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=y_i\}\
        sim_r m\f$
820  *
821  * Supports domain consistent propagation only.
822  *
823  * Throws an exception of type Int::ArgumentSizeMismatch, if
824  *  \a x and \a y are of different size.
825  */
826 void count(Space& home, const IntVarArgs& x, const IntArgs& y, IntRelType r,
         int m, IntConLevel icl=ICL_DEF);
827 /** \brief Post propagator for \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=n\}\
        sim_r z\f$
828  *
829  * Supports domain consistent propagation only.
830  */
831 void count(Space& home, const IntVarArgs& x, int n, IntRelType r, IntVar z,
        IntConLevel icl=ICL_DEF);
832 /** \brief Post propagator for \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=y\}\
        sim_r z\f$
833  *
834  * Supports domain consistent propagation only.
835  */
836 void count(Space& home, const IntVarArgs& x, IntVar y, IntRelType r, IntVar
        z, IntConLevel icl=ICL_DEF);
837 /** \brief Post propagator for \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=y_i\}\
        sim_r z\f$
838  *
839  * Supports domain consistent propagation only.
840  *
841  * Throws an exception of type Int::ArgumentSizeMismatch, if
842  *  \a x and \a y are of different size.
843  */
844 void count(Space& home, const IntVarArgs& x, const IntArgs& y, IntRelType r,
         IntVar z, IntConLevel icl=ICL_DEF);
845
846 /** \brief Posts a global count (cardinality) constraint
847  *
848  * Posts the constraint that
849  * \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=j\}=c_j\f$ and
850  * \f$ \bigcup_i \{x_i\} \subseteq \{0,\ldots,|c|-1\}\f$
851  * (no other value occurs).
852  *
853  * Supports value (\a icl = ICL_VAL, default), bounds (\a icl = ICL_BND),
854  * and domain consistency (\a icl = ICL_DOM).
855  *
856  * Throws an exception of type Int::ArgumentSame, if \a x contains
857  * the same unassigned variable multiply.
858  */
859 void count(Space& home, const IntVarArgs& x, const IntVarArgs& c,
        IntConLevel icl=ICL_DEF);
860
861 /** \brief Posts a global count (cardinality) constraint
862  *
863  * Posts the constraint that
```

```
864  * \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=j\}\in c_j\f$ and
865  * \f$ \bigcup_i \{x_i\} \subseteq \{0,\ldots,|c|-1\}\f$
866  * (no other value occurs).
867  *
868  * Supports value (\a icl = ICL_VAL, default), bounds (\a icl = ICL_BND),
869  * and domain consistency (\a icl = ICL_DOM).
870  *
871  * Throws an exception of type Int::ArgumentSame, if \a x contains
872  * the same unassigned variable multiply.
873  */
874 void count(Space& home, const IntVarArgs& x, const IntSetArgs& c,
        IntConLevel icl=ICL_DEF);
875
876 /** \brief Posts a global count (cardinality) constraint
877  *
878  * Posts the constraint that
879  * \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=v_j\}=c_j\f$ and
880  * \f$ \bigcup_i \{x_i\} \subseteq \bigcup_j \{v_j\}\f$
881  * (no other value occurs).
882  *
883  * Supports value (\a icl = ICL_VAL, default), bounds (\a icl = ICL_BND),
884  * and domain consistency (\a icl = ICL_DOM).
885  *
886  * Throws an exception of type Int::ArgumentSame, if \a x contains
887  * the same unassigned variable multiply.
888  *
889  * Throws an exception of type Int::ArgumentSizeMismatch, if
890  *  \a c and \a v are of different size.
891  */
892 void count(Space& home, const IntVarArgs& x, const IntVarArgs& c, const
        IntArgs& v, IntConLevel icl=ICL_DEF);
893
894 /** \brief Posts a global count (cardinality) constraint
895  *
896  * Posts the constraint that
897  * \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=v_j\}\in c_j\f$ and
898  * \f$ \bigcup_i \{x_i\} \subseteq \bigcup_j \{v_j\}\f$
899  * (no other value occurs).
900  *
901  * Supports value (\a icl = ICL_VAL, default), bounds (\a icl = ICL_BND),
902  * and domain consistency (\a icl = ICL_DOM).
903  *
904  * Throws an exception of type Int::ArgumentSame, if \a x contains
905  * the same unassigned variable multiply.
906  *
907  * Throws an exception of type Int::ArgumentSizeMismatch, if
908  *  \a c and \a v are of different size.
909  */
910 void count(Space& home, const IntVarArgs& x, const IntSetArgs& c, const
        IntArgs& v, IntConLevel icl=ICL_DEF);
911
912 /** \brief Posts a global count (cardinality) constraint
913  *
914  * Posts the constraint that
915  * \f$\#\{i\in\{0,\ldots,|x|-1\};|;x_i=v_j\}\in c\f$ and
916  * \f$ \bigcup_i \{x_i\} \subseteq \bigcup_j \{v_j\}\f$
917  * (no other value occurs).
918  *
919  * Supports value (\a icl = ICL_VAL, default), bounds (\a icl = ICL_BND),
```

```
920  * and domain consistency (\a icl = ICL_DOM).
921  *
922  * Throws an exception of type Int::ArgumentSame, if \a x contains
923  * the same unassigned variable multiply.
924  *
925  * Throws an exception of type Int::ArgumentSizeMismatch, if
926  *  \a c and \a v are of different size.
927  */
928 void count(Space& home, const IntVarArgs& x, const IntSet& c, const IntArgs&
        v, IntConLevel icl=ICL_DEF);
929
930 //@}
931
932 /**
933  * \defgroup TaskModelIntSequence Sequence constraints
934  * \ingroup TaskModelInt
935  */
936
937 //@{
938 /** \brief Post propagator for \f$\operatorname{sequence}(x,s,q,l,u)\f$
939  *
940  * Posts a domain consistent propagator for the constraint
941  * \f$\bigwedge_{i=0}^{|x|-q}
942  *       \operatorname{among}(\langle x_i,\ldots,x_{i+q-1}\rangle,s,l,u)\f$
943  * where the among constraint is defined as
944  * \f$l\leq\#\{j\in\{i,\ldots,i+q-1\};|;x_j\in s\}\leq u\f$.
945  *
946  * Throws the following exceptions:
947  *  - Of type Int::TooFewArguments, if \f$|x|=0\f$.
948  *  - Of type Int::ArgumentSame, if \a x contains
949  *    the same unassigned variable multiply.
950  *  - Of type Exception, if \f$q < 1 \vee q > |x|\f$.
951  */
952 void sequence(Space& home, const IntVarArgs& x, const IntSet& s, int q, int
       l, int u, IntConLevel icl=ICL_DEF);
953
954 /** \brief Post propagator for \f$\operatorname{sequence}(x,s,q,l,u)\f$
955  *
956  * Posts a domain consistent propagator for the constraint
957  * \f$\bigwedge_{i=0}^{|x|-q}
958  *       \operatorname{among}(\langle x_i,\ldots,x_{i+q-1}\rangle,s,l,u)\f$
959  * where the among constraint is defined as
960  * \f$l\leq\#\{j\in\{i,\ldots,i+q-1\};|;x_j\in s\}\leq u\f$.
961  *
962  * Throws the following exceptions:
963  *  - Of type Int::TooFewArguments, if \f$|x|=0\f$.
964  *  - Of type Int::ArgumentSame, if \a x contains
965  *    the same unassigned variable multiply.
966  *  - Of type Exception, if \f$q < 1 \vee q > |x|\f$.
967  */
968 void sequence(Space& home, const BoolVarArgs& x, const IntSet& s, int q, int
        l, int u, IntConLevel icl=ICL_DEF);
969
970 //@}
971
972
973 //@{
974 /** \brief Post propagator for \f$ \min\{x_0,x_1\}=x_2\f$
975  *
```

```
976  * Supports both bounds consistency (\a icl = ICL_BND, default)
977  * and domain consistency (\a icl = ICL_DOM).
978  */
979 void min(Space& home, IntVar x0, IntVar x1, IntVar x2, IntConLevel icl=
         ICL_DEF);
980
981 /** \brief Post propagator for \f$ \min x=y\f$
982  *
983  * Supports both bounds consistency (\a icl = ICL_BND, default)
984  * and domain consistency (\a icl = ICL_DOM).
985  *
986  * If \a x is empty, an exception of type Int::TooFewArguments is thrown.
987  */
988 void min(Space& home, const IntVarArgs& x, IntVar y, IntConLevel icl=ICL_DEF
         );
989 /** \brief Post propagator for \f$ \max\{x_0,x_1\}=x_2\f$
990  *
991  * Supports both bounds consistency (\a icl = ICL_BND, default)
992  * and domain consistency (\a icl = ICL_DOM).
993  */
994 void max(Space& home, IntVar x0, IntVar x1, IntVar x2, IntConLevel icl=
         ICL_DEF);
995 /** \brief Post propagator for \f$ \max x=y\f$
996  *
997  * Supports both bounds consistency (\a icl = ICL_BND, default)
998  * and domain consistency (\a icl = ICL_DOM).
999  *
1000  * If \a x is empty, an exception of type Int::TooFewArguments is thrown.
1001  */
1002 void max(Space& home, const IntVarArgs& x, IntVar y, IntConLevel icl=ICL_DEF
         );
1003
1004 /** \brief Post propagator for \f$ |x_0|=x_1\f$
1005  *
1006  * Supports both bounds consistency (\a icl = ICL_BND, default)
1007  * and domain consistency (\a icl = ICL_DOM).
1008  */
1009 void abs(Space& home, IntVar x0, IntVar x1, IntConLevel icl=ICL_DEF);
1010
1011 /** \brief Post propagator for \f$x_0\cdot x_1=x_2\f$
1012  *
1013  * Supports both bounds consistency (\a icl = ICL_BND, default)
1014  * and domain consistency (\a icl = ICL_DOM).
1015  */
1016 void mult(Space& home, IntVar x0, IntVar x1, IntVar x2, IntConLevel icl=
         ICL_DEF);
1017
1018 /** \brief Post propagator for \f$x_0\cdot x_0=x_1\f$
1019  *
1020  * Supports both bounds consistency (\a icl = ICL_BND, default)
1021  * and domain consistency (\a icl = ICL_DOM).
1022  */
1023 void sqr(Space& home, IntVar x0, IntVar x1, IntConLevel icl=ICL_DEF);
1024
1025 /** \brief Post propagatorSpace& for \f$\lfloor\sqrt{x_0}\rfloor=x_1\f$
1026  *
1027  * Supports both bounds consistency (\a icl = ICL_BND, default)
1028  * and domain consistency (\a icl = ICL_DOM).
1029  */
```

```
1030 void sqrt(Space& home, IntVar x0, IntVar x1, IntConLevel icl=ICL_DEF);
1031
1032 /** \brief Post propagator for \f$x_0\ \mathrm{div}\ x_1=x_2 \land x_0\ \
     mathrm{mod}\ x_1 = x_3\f$
1033  *
1034  * Supports bounds consistency (\a icl = ICL_BND, default).
1035  */
1036 void divmod(Space& home, IntVar x0, IntVar x1, IntVar x2, IntVar x3,
     IntConLevel icl=ICL_DEF);
1037
1038 /** \brief Post propagator for \f$x_0\ \mathrm{div}\ x_1=x_2\f$
1039  *
1040  * Supports bounds consistency (\a icl = ICL_BND, default).
1041  */
1042 void div(Space& home, IntVar x0, IntVar x1, IntVar x2, IntConLevel icl=
     ICL_DEF);
1043
1044 /** \brief Post propagator for \f$x_0\ \mathrm{mod}\ x_1=x_2\f$
1045  *
1046  * Supports bounds consistency (\a icl = ICL_BND, default).
1047  */
1048 void mod(Space& home, IntVar x0, IntVar x1, IntVar x2, IntConLevel icl=
     ICL_DEF);
1049 //@}
1050
1051 /**
1052  * \defgroup TaskModelIntLI Linear constraints over integer variables
1053  * \ingroup TaskModelInt
1054  *
1055  * All variants for linear constraints over integer variables share
1056  * the following properties:
1057  *  - Bounds consistency (over the real numbers) is supported for
1058  *    all constraints (actually, for disequlities always domain consistency
1059  *    is used as it is cheaper). Domain consistency is supported for all
1060  *    non-reified constraint. As bounds consistency for inequalities
1061  *    coincides with domain consistency, the only
1062  *    real variation is for linear equations. Domain consistent
1063  *    linear equations have exponential complexity, so use with care!
1064  *  - Variables occurring multiply in the argument arrays are replaced
1065  *    by a single occurrence: for example, \f$ax+bx\f$ becomes
1066  *    \f$(a+b)x\f$.
1067  *  - If in the above simplification the value for \f$(a+b)\f$ (or for
1068  *    \f$a\f$ and \f$b\f$) exceeds the limits for integers as
1069  *    defined in Int::Limits, an exception of type
1070  *    Int::OutOfLimits is thrown.
1071  *  - Assume the constraint
1072  *    \f$\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r c\f$.
1073  *    If  \f$|c|+\sum_{i=0}^{|x|-1}a_i\cdot x_i\f$ exceeds the maximal
1074  *    available precision (at least \f$2^{48}\f$), an exception of
1075  *    type Int::OutOfLimits is thrown.
1076  *  - In all other cases, the created propagators are accurate (that
1077  *    is, they will not silently overflow during propagation).
1078  */
1079 /** \brief Post propagator for \f$\sum_{i=0}^{|x|-1}x_i\sim_r c\f$
1080  * \ingroup TaskModelIntLI
1081  */
1082 void linear(Space& home, const IntVarArgs& x, IntRelType r, int c,
     IntConLevel icl=ICL_DEF);
1083 /** \brief Post propagator for \f$\sum_{i=0}^{|x|-1}x_i\sim_r y\f$
```

```
1084  * \ingroup TaskModelIntLI
1085  */
1086 void linear(Space& home, const IntVarArgs& x, IntRelType r, IntVar y,
      IntConLevel icl=ICL_DEF);
1087 /** \brief Post propagator for \f$\left(\sum_{i=0}^{|x|-1}x_i\sim_r c\right)
      \Leftrightarrow b\f$
1088  * \ingroup TaskModelIntLI
1089  */
1090 void linear(Space& home, const IntVarArgs& x, IntRelType r, int c, BoolVar b
      , IntConLevel icl=ICL_DEF);
1091 /** \brief Post propagator for \f$\left(\sum_{i=0}^{|x|-1}x_i\sim_r y\right)
      \Leftrightarrow b\f$
1092  * \ingroup TaskModelIntLI
1093  */
1094 void linear(Space& home, const IntVarArgs& x, IntRelType r, IntVar y,
      BoolVar b, IntConLevel icl=ICL_DEF);
1095 /** \brief Post propagator for \f$\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r c\f$
1096  *
1097  *  Throws an exception of type Int::ArgumentSizeMismatch, if
1098  *  \a a and \a x are of different size.
1099  * \ingroup TaskModelIntLI
1100  */
1101 void linear(Space& home, const IntArgs& a, const IntVarArgs& x, IntRelType r
      , int c, IntConLevel icl=ICL_DEF);
1102 /** \brief Post propagator for \f$\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r y\f$
1103  *
1104  *  Throws an exception of type Int::ArgumentSizeMismatch, if
1105  *  \a a and \a x are of different size.
1106  * \ingroup TaskModelIntLI
1107  */
1108 void linear(Space& home, const IntArgs& a, const IntVarArgs& x, IntRelType r
      , IntVar y, IntConLevel icl=ICL_DEF);
1109 /** \brief Post propagator for \f$\left(\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r
       c\right)\Leftrightarrow b\f$
1110  *
1111  *  Throws an exception of type Int::ArgumentSizeMismatch, if
1112  *  \a a and \a x are of different size.
1113  * \ingroup TaskModelIntLI
1114  */
1115 void linear(Space& home, const IntArgs& a, const IntVarArgs& x, IntRelType r
      , int c, BoolVar b, IntConLevel icl=ICL_DEF);
1116 /** \brief Post propagator for \f$\left(\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r
       y\right)\Leftrightarrow b\f$
1117  *
1118  *  Throws an exception of type Int::ArgumentSizeMismatch, if
1119  *  \a a and \a x are of different size.
1120  * \ingroup TaskModelIntLI
1121  */
1122 void linear(Space& home, const IntArgs& a, const IntVarArgs& x, IntRelType r
      , IntVar y, BoolVar b, IntConLevel icl=ICL_DEF);
1123
1124
1125 /**
1126  * \defgroup TaskModelIntLB Linear constraints over Boolean variables
1127  * \ingroup TaskModelInt
1128  *
1129  * All variants for linear constraints over Boolean variables share
1130  * the following properties:
1131  *  - Bounds consistency (over the real numbers) is supported for
```

```
1132  *     all constraints (actually, for disequlities always domain consistency
1133  *     is used as it is cheaper).
1134  *  - Variables occurring multiply in the argument arrays are replaced
1135  *     by a single occurrence: for example, \f$ax+bx\f$ becomes
1136  *     \f$(a+b)x\f$.
1137  *  - If in the above simplification the value for \f$(a+b)\f$ (or for
1138  *     \f$a\f$ and \f$b\f$) exceeds the limits for integers as
1139  *     defined in Int::Limits, an exception of type
1140  *     Int::OutOfLimits is thrown.
1141  *  - Assume the constraint
1142  *     \f$\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r c\f$.
1143  *     If  \f$|c|+\sum_{i=0}^{|x|-1}a_i\cdot x_i\f$ exceeds the limits
1144  *     for integers as defined in Int::Limits, an exception of
1145  *     type Int::OutOfLimits is thrown.
1146  *  - In all other cases, the created propagators are accurate (that
1147  *     is, they will not silently overflow during propagation).
1148  */
1149 /** \brief Post propagator for \f$\sum_{i=0}^{|x|-1}x_i\sim_r c\f$
1150  * \ingroup TaskModelIntLB
1151  */
1152 void linear(Space& home, const BoolVarArgs& x, IntRelType r, int c,
        IntConLevel icl=ICL_DEF);
1153 /** \brief Post propagator for \f$\left(\sum_{i=0}^{|x|-1}x_i\sim_r c\right)
        \Leftrightarrow b\f$
1154  * \ingroup TaskModelIntLB
1155  */
1156 void linear(Space& home, const BoolVarArgs& x, IntRelType r, int c, BoolVar
        b, IntConLevel icl=ICL_DEF);
1157 /** \brief Post propagator for \f$\sum_{i=0}^{|x|-1}x_i\sim_r y\f$
1158  * \ingroup TaskModelIntLB
1159  */
1160 void linear(Space& home, const BoolVarArgs& x, IntRelType r, IntVar y,
        IntConLevel icl=ICL_DEF);
1161 /** \brief Post propagator for \f$\left(\sum_{i=0}^{|x|-1}x_i\sim_r y\right)
        \Leftrightarrow b\f$
1162  * \ingroup TaskModelIntLB
1163  */
1164 void linear(Space& home, const BoolVarArgs& x, IntRelType r, IntVar y,
        BoolVar b, IntConLevel icl=ICL_DEF);
1165 /** \brief Post propagator for \f$\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r c\f$
1166  *
1167  *  Throws an exception of type Int::ArgumentSizeMismatch, if
1168  *  \a a and \a x are of different size.
1169  * \ingroup TaskModelIntLB
1170  */
1171 void linear(Space& home, const IntArgs& a, const BoolVarArgs& x, IntRelType
        r, int c, IntConLevel icl=ICL_DEF);
1172 /** \brief Post propagator for \f$\left(\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r
         c\right)\Leftrightarrow b\f$
1173  *
1174  *  Throws an exception of type Int::ArgumentSizeMismatch, if
1175  *  \a a and \a x are of different size.
1176  * \ingroup TaskModelIntLB
1177  */
1178 void linear(Space& home, const IntArgs& a, const BoolVarArgs& x, IntRelType
        r, int c, BoolVar b, IntConLevel icl=ICL_DEF);
1179 /** \brief Post propagator for \f$\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r y\f$
1180  *
1181  *  Throws an exception of type Int::ArgumentSizeMismatch, if
```

```
1182  *   \a a and \a x are of different size.
1183  * \ingroup TaskModelIntLB
1184  */
1185 void linear(Space& home, const IntArgs& a, const BoolVarArgs& x, IntRelType
      r, IntVar y, IntConLevel icl=ICL_DEF);
1186 /** \brief Post propagator for \f$\left(\sum_{i=0}^{|x|-1}a_i\cdot x_i\sim_r
      y\right)\Leftrightarrow b\f$
1187  *
1188  *   Throws an exception of type Int::ArgumentSizeMismatch, if
1189  *   \a a and \a x are of different size.
1190  * \ingroup TaskModelIntLB
1191  */
1192 void linear(Space& home, const IntArgs& a, const BoolVarArgs& x, IntRelType
      r, IntVar y, BoolVar b, IntConLevel icl=ICL_DEF);
1193
1194 //}
1195 namespace MPG {
1196 /**
1197  * Posting function for the constraint
1198  * x = y
1199  */
1200     void equal(Gecode::Space& home, RelVar x, RelVar y);
1201 /**
1202  * Posting function for the constraint
1203  * x = y
1204  *
1205  * @warn @a x must be a unary relation
1206  */
1207 //   void equal(Space& home, RelVar x, SetVar y);
1208 /**
1209  * Posting function for the constraint
1210  * y = \complement{x}
1211  */
1212     void complement(Gecode::Space& home, RelVar x, RelVar y);
1213 /**
1214  * Posting function for the constraint
1215  * z = x \intersect y
1216  */
1217     void intersection(Gecode::Space& home, RelVar x, RelVar y, RelVar z);
1218 /**
1219  * Posting function for the constraint
1220  * z = z \union y
1221  */
1222     void Union(Gecode::Space& home, RelVar A, RelVar B, RelVar C);
1223 /**
1224  * Posting function for the constraint
1225  * x \intersect y = \emptyset
1226  */
1227     void disjoint(Gecode::Space& home, RelVar x, RelVar y);
1228 /**
1229  * Posting function for the constraint
1230  * x \subseteq y
1231  */
1232   void subset(Gecode::Space& home, RelVar x, RelVar y);
1233 /**
1234  * Posting function for the constraint
1235  * y = \project{a}{x}
1236  */
1237 //    void projection(Space& home, RelVar x, const Schema& a, RelVar y);
```

```
1238 /**
1239  * Posting function for the constraint
1240  * z = x \join_a y
1241  */
1242 void join(Space& home, RelVar x, RelVar y, RelVar z);
1243 /**
1244  * Posting function for the constraint
1245  * z = x \compose_a y
1246  *
1247  * The constraint is enforced by two constraints:
1248  * z' = \join{x}{y}
1249  * z = \proj{\attributes{z}}{z'}
1250  *
1251  * Where z' is a temporary variable on schema x.schema() \cup y.schema{}
1252  */
1253     void compose(Gecode::Space& home, RelVar x, RelVar y, RelVar z);
1254 /**
1255  * Posting function for the constraint
1256  * y =_{m} x
1257  */
1258 //    void rename(Gecode::Space& home, RelVar x, const AttributeMap& m,
      RelVar y);
1259 /**
1260  * Posting function for the constraint
1261  * y = |x|
1262  */
1263 void cardinality(Gecode::Space& home, RelVar x, IntVar y);
1264 }
1265 #endif /* constraints_hpp */
```

D.1.2 GOM

```
1
2  using CuddAbstraction::Attribute;
3     using CuddAbstraction::AttributeDomain;
4     using CuddAbstraction::makeDomain;
5     using CuddAbstraction::DomainGroup;
6     using CuddAbstraction::insert;
7
8     static Attribute pitch = Attribute(MPG::Rel::getRelationSpace(),
          makeDomain(128), "pitch");
9     static Attribute vel = Attribute(MPG::Rel::getRelationSpace(),
          makeDomain(128), "velocity");
10    static Attribute dur = Attribute(MPG::Rel::getRelationSpace(),
          makeDomain(10001), "duration");
11    static Attribute offset = Attribute(MPG::Rel::getRelationSpace(),
          makeDomain(10001), "offset");
12    static Attribute onset = Attribute(MPG::Rel::getRelationSpace(),
          makeDomain(10001), "onset");
13    static const Relation chordseqrel = Relation(MPG::Rel::getRelationSpace
          (),{pitch,vel,dur,offset,onset});
14
15    class NoteVar : public GeLisp::GlRelVar{
16    public:
17
18        /* CONSTRUCTORS */
19        NoteVar(GlRelVar& rv):GlRelVar(rv){}
20
```

```
21          NoteVar(GlSpace& home, vector<int> pitchdom, vector<int> veldom,
                vector<int> durdom):GlRelVar(initNote(home,pitchdom,veldom,
                durdom)){}
22
23          GlRelVar initNote(GlSpace& home,vector<int> pd, vector<int> vd,
                vector<int> dd){
24              auto& relHome = MPG::Rel::getRelationSpace();
25              {
26                  DomainGroup dg;
27                  dg << pitch << vel << dur;
28                  //2.
29                  Relation pitchRel(relHome, {pitch});
30                  {
31                      for (vector<int>::iterator piter = pd.begin(); piter !=
                            pd.end(); ++piter)
32                          pitchRel.add(relHome,*piter);
33                  }
34                  Relation velRel(relHome, {vel});
35                  {
36                      for (vector<int>::iterator viter = vd.begin(); viter !=
                            vd.end(); ++viter)
37                          velRel.add(relHome,*viter);
38                  }
39                  Relation durRel(relHome, {dur});
40                  {
41                      for (vector<int>::iterator diter = dd.begin(); diter !=
                            dd.end(); ++diter)
42                          durRel.add(relHome,*diter);
43                  }
44                  Relation note = pitchRel.crossWith(relHome,velRel.crossWith(
                        relHome,durRel));
45
46                  GlRelVar notevar = home.newRelVar(empty(note.schema()),note)
                        ;
47                  GlIntVar* y = new GlIntVar(home,3,3);
48                  cardinality(home,notevar,*y);
49                  return notevar;
50              }
51          }
52          /*
53           Returns a vector with the nsol last solutions for NoteVar in
                solvect
54           */
55          vector<Note> getNoteVarAllSol(vector<GlSpace*> solvect, int nsol){
56              cout << "getNoteVarAllSol IN" << endl;
57              vector<Note> out;
58              int index = this->getIndexInVector();
59              for (int i = solvect.size()-nsol; i < solvect.size(); i++) {
60                  GlSpace* curr = solvect.at(i);
61                  NoteVar nv(curr->getRelVar(index));
62                  out.push_back(*nv.toNote(*curr,index));
63              }
64              return out;
65          }
66
67
68          /*
69           Convert a NoteVar to a Note and return the pointer to the note
                result
```

```
70          */
71          Note* toNote(GlSpace& home, int index)const{
72              cout << "index: " << index << endl;
73              Note* out = new Note();
74              const auto& tuples =  home.getRelVar(index).glb().asTuples(MPG::
                    Rel::getRelationSpace());
75              vector<int> vtuple;
76              for (const auto& v: tuples[0]) {
77                  vtuple.push_back(v);
78              }
79              out = new Note(vtuple[0],vtuple[1],vtuple[2],1);
80              return out;
81          }
82
83      };
84
85
86      class ChordVar : public GlRelVar{
87      public:
88
89          /*  CONSTRUCTORS  */
90          ChordVar(GlRelVar& rv):GlRelVar(rv){}
91
92          ChordVar(GlSpace& home, vector<int> pitchdom, vector<int> veldom,
                vector<int> durdom, vector<int> offdom):GlRelVar(initChord(home,
                pitchdom,veldom,durdom, offdom)){}
93
94          GlRelVar initChord(GlSpace& home,vector<int> pd, vector<int> vd,
                vector<int> dd, vector<int> ofd){
95              auto& relHome = MPG::Rel::getRelationSpace();
96              {
97                  DomainGroup dg;
98                  dg << pitch << vel << dur << offset;
99                  Relation pitchRel(relHome, {pitch});
100                 {
101                     for (vector<int>::iterator piter = pd.begin(); piter !=
                            pd.end(); ++piter)
102                         pitchRel.add(relHome,*piter);
103                 }
104                 Relation velRel(relHome, {vel});
105                 {
106                     for (vector<int>::iterator viter = vd.begin(); viter !=
                            vd.end(); ++viter)
107                         velRel.add(relHome,*viter);
108                 }
109                 Relation durRel(relHome, {dur});
110                 {
111                     for (vector<int>::iterator diter = dd.begin(); diter !=
                            dd.end(); ++diter)
112                         durRel.add(relHome,*diter);
113                 }
114                 Relation offRel(relHome, {offset});
115                 {
116                     for (vector<int>::iterator ofiter = ofd.begin(); ofiter
                            != ofd.end(); ++ofiter)
117                         offRel.add(relHome,*ofiter);
118                 }
119                 Relation chord = pitchRel.crossWith(relHome,velRel.crossWith
                        (relHome,durRel.crossWith(relHome,offRel)));
```

```
120
121                 GlRelVar chordVar = home.newRelVar(empty(chord.schema()),
                        chord);
122                 return chordVar;
123             }
124
125         }
126         /*
127          Returns a vector with the nsol last solutions for ChordSeq in
                solvect
128          */
129         vector<Chord> getChordVarAllSol(vector<GlSpace*> solvect, int nsol){
130             vector<Chord> out;
131             int index = this->getIndexInVector();
132             for (int i = solvect.size()-nsol; i < solvect.size(); i++) {
133                 GlSpace* curr = solvect.at(i);
134                 ChordVar cv(curr->getRelVar(index));
135                 out.push_back(*cv.toChord(*curr,index));
136             }
137             return out;
138         }
139
140         /*
141          Convert a ChordVar to a Chord and return the pointer to the Chord
                result
142          */
143         Chord* toChord(GlSpace& home, int index)const{
144             // int index = this->getIndexInVector();
145             cout << "index: " << index << endl;
146             Chord* out;
147             vector<double> lmidic;
148             vector<int> lvel;
149             vector<double> ldur;
150             vector<double> loff;
151             vector<int> lchan;
152             lchan.push_back(1);
153             const auto& tuples =  home.getRelVar(index).glb().asTuples(MPG::
                    Rel::getRelationSpace());
154             for (const auto& t : tuples) {
155                 lmidic.push_back(t[0]);
156                 lvel.push_back(t[1]);
157                 ldur.push_back(t[2]);
158                 loff.push_back(t[3]);
159             }
160             out = new Chord(lmidic,lvel,loff,ldur,lchan);
161             return out;
162         }
163
164     };
165
166
167     class ChordSeqVar : public GlRelVar{
168     public:
169         /*  CONSTRUCTORS  */
170         ChordSeqVar(void):GlRelVar(){}
171         ChordSeqVar(GlRelVar& rv):GlRelVar(rv){}
172
173         ChordSeqVar(GlSpace& home, vector<int> pitchdom, vector<int> veldom,
                vector<int> durdom, vector<int> offdom, vector<int> ondom):
```

```
        GlRelVar(initChordSeq(home,pitchdom,veldom,durdom, offdom, ondom
        )){cout<<"RelHome: "<<&MPG::Rel::getRelationSpace()<<endl;}
174
175     GlRelVar initChordSeq(GlSpace& home,vector<int> pd, vector<int> vd,
            vector<int> dd, vector<int> ofd, vector<int> ond){
176         auto& relHome = MPG::Rel::getRelationSpace();
177         {
178             DomainGroup dg;
179             dg << pitch << vel << dur << offset << onset;
180             //2.
181             Relation pitchRel(relHome, {pitch});
182             {
183                 for (vector<int>::iterator piter = pd.begin(); piter !=
                        pd.end(); ++piter)
184                     pitchRel.add(relHome,*piter);
185             }
186             Relation velRel(relHome, {vel});
187             {
188                 for (vector<int>::iterator viter = vd.begin(); viter !=
                        vd.end(); ++viter)
189                     velRel.add(relHome,*viter);
190             }
191             Relation durRel(relHome, {dur});
192             {
193                 for (vector<int>::iterator diter = dd.begin(); diter !=
                        dd.end(); ++diter)
194                     durRel.add(relHome,*diter);
195             }
196             Relation offRel(relHome, {offset});
197             {
198                 for (vector<int>::iterator ofiter = ofd.begin(); ofiter
                        != ofd.end(); ++ofiter)
199                     offRel.add(relHome,*ofiter);
200             }
201             Relation onRel(relHome, {onset});
202             {
203                 for (vector<int>::iterator oniter = ond.begin(); oniter
                        != ond.end(); ++oniter)
204                     onRel.add(relHome,*oniter);
205             }
206             Relation chordseq = pitchRel.crossWith(relHome,velRel.
                    crossWith(relHome,durRel.crossWith(relHome,offRel.
                    crossWith(relHome,onRel))));
207
208             GlRelVar chordSeqVar = home.newRelVar(empty(chordseq.schema
                    ()),chordseq);
209             return chordSeqVar;
210         }
211     }
212
213     /*
214      Returns a vector with the nsol last solutions for ChordSeqVar in
            solvect
215      */
216     vector<ChordSeq> getChordSeqVarAllSol(vector<GlSpace*> solvect, int
            nsol){
217         vector<ChordSeq> out;
218         int index = this->getIndexInVector();
219         for (int i = solvect.size()-nsol; i < solvect.size(); i++) {
```

```
220                 GlSpace* curr = solvect.at(i);
221                 ChordSeqVar csv(curr->getRelVar(index));
222                 out.push_back(*csv.toChordSeq(*curr,index));
223             }
224             return out;
225         }
226 
227         /*
228          Convert a ChordSeqVar to a ChordSeq and return the pointer to the
                 chordseq result
229          */
230         ChordSeq* toChordSeq(GlSpace& home, int index)const{
231             ChordSeq* out;
232             vector<vector<double> > lmidic;
233             vector<vector<int> > lvel;
234             vector<vector<double> > ldur;
235             vector<vector<double> > loff;
236             vector<int> lon;
237             vector<vector<int> > lchan;
238             vector<int>ulch(1,1);
239             lchan.push_back(ulch);
240             const auto& tuples = home.getRelVar(index).glb().asTuples(MPG::
                    Rel::getRelationSpace());
241             for (const auto& t : tuples) {
242                 cout << t[0] << "-"<< t[1] << "-"<< t[2] << "-"<< t[3] << "-
                        "<< t[4] << endl;
243                 vector<double> v1 (1,t[0]);
244                 lmidic.push_back(v1);
245                 vector<int> v2 (1,t[1]);
246                 lvel.push_back(v2);
247                 vector<double> v3 (1,t[2]);
248                 ldur.push_back(v3);
249                 vector<double> v4 (1,t[3]);
250                 loff.push_back(v4);
251                 lon.push_back(t[4]);
252             }
253             out = new ChordSeq(lmidic,lon,ldur,lvel,loff,lchan,0);
254             return out;
255         }
256 
257     };
258 }
```

D.1.3 OM++

Utils_OM_Objects.hpp

This object contains the functions allowing the good format to pass an array to OM or to receive an pointer to an array from OM. The repetition of the functions with different names is quite ugly. I am aware that using the overriding would be quite better but, since these functions are using in C++ as in Lisp, these choice has been done that to be sure to use the good function. Using the overriding is part of a futur improvement of the code.

```
1 //
2 //  OM_Object.hpp
3 //  IOM
4 //
5 //  Created by Geoffroy Zoetardt on 8/04/16.
```

```
6  //  Copyright 2016 Geoffroy Zoetardt. All rights reserved.
7  //
8
9  #ifndef OM_Object_hpp
10 #define OM_Object_hpp
11
12 #include <iostream>
13 #include <stdio.h>
14 #include <vector>
15 #include <string>
16
17 using namespace std;
18
19 class OM_Object{
20 public:
21
22     static int getSizeVect(vector<OM_Object>arg);
23
24     /*  Return a vector initialize with an array  */
25     static vector<double> init_vect(double* arr_in);
26
27     static vector<int> init_vectI(int* arr_in);
28
29     /*  Return a vector2D initialize with an array2D  */
30     static vector< vector<double> > init_vect2D(double** arr_in);
31
32     static vector< vector<int> > init_vect2DI(int** arr_in);
33
34     /*  Return an array where the first element is the number of element in
           it  */
35     static double* put_length_first( vector<double> arg);
36
37
38
39     static int* put_length_firstI(vector<int> arg);
40
41     static int* put_length_firstB(vector<bool> arg);
42
43     /*  Return an array2D where the first element is a sub-array with the
           number of sub-array in the array2D.
44      Moreover, each sub-array has, as first element, the numbere of element
            in the sub-array.
45      example:
46
47      arg = {{1,2},{3,4,5},{6}} ----------> out = {{3},{2,1,2},{3,3,4,5},{1,6}}
48
49      */
50     static double** put_length_first2D( vector< vector<double> > arg) ;
51
52     static int** put_length_first2DI( vector< vector<int> > arg) ;
53
54     static int** put_length_first2DB(vector< vector<bool> > arg);
55
56
57     static string toString();
58
59 };
60
61 #endif /* OM_Object_hpp */
```

OM++_Harmonic.hpp

```
//
//  Harmonic.h
//  IOM
//
//  Created by Geoffroy Zoetardt on 8/04/16.
//  Copyright  2016 Geoffroy Zoetardt. All rights reserved.
//

#ifndef Harmonic_hpp
#define Harmonic_hpp

#include "Utils_OM_Object.hpp"

#include <stdio.h>
#include <string>
#include <vector>

using namespace std;

class Harmonic : public OM_Object{
public:
    static string toString();
};

#endif /* Harmonic_hpp */
```

OM++_Note.hpp

```
//
//  Note.h
//  IOM
//
//  Created by Geoffroy Zoetardt on 7/04/16.
//  Copyright 2016 Geoffroy Zoetardt. All rights reserved.
//

#ifndef Note_hpp
#define Note_hpp

#include "OM++_Harmonic.hpp"

class Note : protected Harmonic {
    /*
     A Note is the C++ representation of the OM Note object
     It is composed of:
     - one pitche (midic expressed as midicent (cent of a Midi unit).
     - one velocitie (vel) represents the dynamic of note.
     - one duration (dur) represents the duration of note.
     - one channel (chan) represents the midi channel used.
    */

private:
    int midic;
    double vel;
    double dur;
    double chan;
```

```
29
30 public:
31
32     /*  CONSTRUCTORS  */
33     Note(Note const &self);
34
35     Note(double midic=6000, int vel=0, double dur=1000, int chan=1);
36
37     /*  SETTERS  */
38     void set_midic(double midic);
39     void set_vel(int vel);
40     void set_dur(double dur);
41     void set_chan(int chan);
42
43     /*  GETTERS  */
44     double get_midic() const;
45     int get_vel() const;
46     double get_dur() const;
47     int get_chan() const;
48
49     static Note getNoteFromVectorNote (vector<Note> vnote, int index);
50     static int getSizeVect(vector<Note>arg);
51
52     string toString();
53     void print();
54 };
55
56 #endif /* Note_hpp */
```

OM++_Chord.hpp

```
1  //
2  //  Chord.h
3  //  GOM
4  //
5  //  Created by Geoffroy Zoetardt on 7/04/16.
6  //  Copyright  2016 Geoffroy Zoetardt. All rights reserved.
7  //
8
9  #ifndef Chord_hpp
10 #define Chord_hpp
11
12 #include "OM++_Note.hpp"
13 #include "OM++_Harmonic.hpp"
14
15 #include <vector>
16
17 using namespace std;
18
19 /*********************************/
20 /*              CHORD            */
21 /*********************************/
22 /*
23  A Chord is the C++ representation of the OM Chord object
24  It is composed of:
25  - a vector of pitches (lmidic) that represents all notes of the chord
        expressed as midicent (cent of a Midi unit).
26  - a vector of velocities (lvel) represents the dynamic of note(s).
```

```
27  - a vector of durations (ldur) represents the duration of note(s).
28  - a vector of channels (lchan) represents the midi channel used for each
        note.
29  - a vector of offsets (loffset) indicates the delay between the the first
        note and the following.
30  All those elements are expressed in more details in the documentation of
        OpenMusic.
31  - a vector of Notes (lnote) is a vector of Note (cf. Note.cpp) only used in
         C++ to facilitate manipulations
32
33  For a better understanding of the COM objects, see the chapter about Score
        Objects in OpenMusic.
34
35  References:
36  OM Score Objects: cf. http://support.ircam.fr/docs/om/om6-manual/co/
        ScoreObjects.html
37  OM Harmonic objects: cf. http://support.ircam.fr/docs/om/om6-manual/co/Note
        -Chord-Chord-seq.html
38  */
39 class Chord : protected Harmonic {
40 private:
41
42     vector<double> lmidic;      //Pitches
43     vector<int> lvel;         //Velocities
44     vector<double> ldur;        //Durations
45     vector<int> lchan;        //Channels
46     vector<double> loffset;     //Offsets
47     vector<Note> lnote;       //Notes
48
49     /*
50      Create a note based on other attribute. The created note correspond to
            the OM's definition of the Note.
51      (cf. Note.cpp )
52      */
53    // vector<Note> create_lnote(vector<double> lmidic, vector<int> lvel,
           vector<double> ldur, vector<int> lchan);
54
55     /*  Update the vector of Notes */
56  //    void update_lnote();
57
58
59 public:
60 /*  self Constructor */
61     Chord(void);
62     Chord(Chord const &self);
63
64 /*  Constructor (constructor for conversion between OM and C++)
65  @preconditions:
66  lmidic, lvel, loffset, ldur, lchan are composed of doubleeger >= 0
67  */
68     Chord(double* lmidic, int* lvel, double* loffset, double* ldur, int*
           lchan);
69
70 /*  Vector Constructor  */
71     Chord(vector<double> lmidic, vector<int> lvel, vector<double> loffset,
           vector<double> ldur, vector<int> lchan);
72
73 /*  Vector of notes Constructor */
74 //    Chord(vector<Note> lnote, vector<double> loffset);
```

```
75
76 /*  Setters  */
77     void set_lmidic(vector<double> lmidic);
78     void set_lvel(vector<int> lvel);
79     void set_ldur(vector<double> ldur);
80     void set_lchan(vector<int> lchan);
81     void set_loffset(vector<double> loffset);
82     void set_lnote(vector<Note> lnote);
83     void setup_lnote(vector<Note> lnote, vector<double> lmidic, vector<int>
           lvel, vector<double> ldur, vector<int> lchan);
84
85 /*  Vector Getter   */
86     vector<double> get_lmidic()const;
87     vector<int> get_lvel()const;
88     vector<double> get_ldur()const;
89     vector<int> get_lchan()const;
90     vector<double> get_loffset()const;
91     vector<Note> get_lnote()const;
92
93 /*  Array Getter (for doubleerfacing with OM)  */
94     double* OMget_lmidic()const;
95     int* OMget_lvel()const;
96     double* OMget_ldur()const;
97     int* OMget_lchan()const;
98     double* OMget_loffset()const;
99
100    static int getSizeVect(vector<Chord>arg);
101    static Chord getChordFromVectorChord (vector<Chord> vchord, int index);
102
103    vector<Note> create_lnote(vector<double> lmidic, vector<int> lvel,
           vector<double> ldur, vector<int> lchan);
104    void update_lnote();
105
106 /*  ToString  */
107    string toString();
108 };
109
110 #endif /* Chord_hpp */
```

OM++_ChordSeq.hpp

```
1 //
2 //  ChordSeq.hpp
3 //  IOM
4 //
5 //  Created by Geoffroy Zoetardt on 8/04/16.
6 //  Copyright  2016 Geoffroy Zoetardt. All rights reserved.
7 //
8
9 #ifndef ChordSeq_hpp
10 #define ChordSeq_hpp
11
12 #include "OM++_Chord.hpp"
13 #include "OM++_Harmonic.hpp"
14
15 #include <vector>
16
17 using namespace std;
```

```
18
19 class ChordSeq : protected Harmonic {
20     /*********************************/
21     /*              CHORDSEQ         */
22     /*********************************/
23     /*
24      A ChordSeq is the C++ representation of the OM ChordSeq object
25      It is composed of:
26      - a vector of pitches (lmidic) that represents all pitches of the chord
            -sequence expressed as midicent (cent of a Midi unit).
27      - a vector of vectors of velocities (lvel) represents the dynamic of
            note(s).
28      - a vector of vectors of durations (ldur) represents the duration of
            note(s).
29      - a vector of vectors of channels (lchan) represents the midi channel
            used for each note.
30      - a vector of vectors of offsets (loffset) indicates the delay between
            the the first note and the following.
31      - a vector of onsets (lonset) indicates the delay between the the first
             note and the following.
32      */
33 private:
34
35     vector< vector<double> > lmidic;
36     vector< vector<int> > lvel;
37     vector< vector<double> > ldur;
38     vector< vector<int> > lchan;
39     vector< vector<double> > loffset;
40     vector<Chord> lchord;
41     vector<int> lonset;
42     double legato;
43
44     vector<Chord> create_lchord (vector< vector<double> > lmidic, vector<
           vector<int> > lvel, vector< vector<double> > ldur, vector< vector<
           int> > lchan, vector< vector<double> > loffset);
45     void update_lchord();
46     void setup_lchord (vector<Chord> lchord, vector< vector<double> > lmidic
           , vector< vector<int> > lvel, vector< vector<double> > ldur, vector<
            vector<int> > lchan, vector< vector<double> > loffset);
47
48 public:
49
50     /* CONSTRUCTORS */
51     ChordSeq(ChordSeq const &self);
52     ChordSeq(vector<Chord> lchord, vector<int> lonset, double legato = 0);
53     ChordSeq(double** lmidic, int* lonset, double** ldur, int** lvel, double
           ** loffset, int** lchan, int legato);
54     ChordSeq(vector< vector<double> > lmidic, vector<int> lonset, vector<
           vector<double> > ldur, vector< vector<int> > lvel, vector< vector<
           double> > loffset, vector< vector<int> > lchan, int legato = 0);
55
56
57     /*  SETTERS  */
58     void set_lchord(vector<Chord> lchord);
59     void set_lonset(vector<int> lonset);
60     void set_legato(int legato);
61     void set_lmidic(vector< vector<double> > lmidic);
62     void set_lvel(vector< vector<int> > lvel);
63     void set_ldur(vector< vector<double> > ldur);
```

```
64      void set_lchan(vector< vector<int> > lchan);
65      void set_loffset(vector< vector<double> > loffset);
66
67      /*  GETTERS  */
68      vector< vector<double> > get_lmidic()const;
69      vector< vector<int> > get_lvel()const;
70      vector< vector<double> > get_ldur()const;
71      vector< vector<int> > get_lchan()const;
72      vector< vector<double> > get_loffset()const;
73      vector<Chord> get_lchord()const;
74      vector<int> get_lonset()const;
75      int get_legato()const;
76
77      /* OM GETTERS */
78      double** OMget_lmidic()const;
79      int** OMget_lvel()const;
80      double** OMget_ldur()const;
81      int** OMget_lchan()const;
82      double** OMget_loffset()const;
83      int* OMget_lonset()const;
84
85      static int getSizeVect(vector<ChordSeq>arg);
86      static ChordSeq getChordSeqFromVectorChordSeq (vector<ChordSeq> vchord,
            int index);
87      string toString();
88      void print();
89
90  };
91
92  #endif /* ChordSeq_hpp */
```

Rhythmic.hpp

```
1   //
2   //  Rhythmic.hpp
3   //  IOM
4   //
5   //  Created by Geoffroy Zoetardt on 8/04/16.
6   //  Copyright  2016 Geoffroy Zoetardt. All rights reserved.
7   //
8
9   #ifndef Rhythmic_hpp
10  #define Rhythmic_hpp
11
12  #include "OM_Object.hpp"
13
14  #include <stdio.h>
15  #include <string>
16  #include <vector>
17
18  using namespace std;
19
20  class Rhythmic : public OM_Object{
21  public:
22      static string toString();
23  };
24  #endif /* Rhythmic_hpp */
```

Measure.hpp

```
//
//  Measure.hpp
//  IOM
//
//  Created by Geoffroy Zoetardt on 12/04/16.
//  Copyright  2016 Geoffroy Zoetardt. All rights reserved.
//

#ifndef Measure_hpp
#define Measure_hpp

#include "Rhythmic.hpp"

#include <vector>
#include <stdio.h>
#include <iostream>

using namespace std;

class Measure : protected Rhythmic{
private:
    vector<int> time_sign;       //Time Signature
    vector<int> rhyt_prop;           //Rhythmic proportions

public:

    Measure(Measure const &self);

    Measure(int** measure);

    Measure(vector<int> time_sign, vector<int> r_p);

    void set_time_sign(vector<int> t_s);

    void set_rhyt_prop(vector<int>r_p);

    vector<int> get_time_sign() const;

    vector<int> get_rhyt_prop() const;

    int** OMget_measure();

    string toString();

};


#endif /* Measure_hpp */
```

Rhythm_tree.hpp

```
//
//  Rhythm_tree.hpp
//  IOM
//
//  Created by Geoffroy Zoetardt on 12/04/16.
```

```
//  Copyright  2016 Geoffroy Zoetardt. All rights reserved.
//

#ifndef Rhythm_tree_hpp
#define Rhythm_tree_hpp

#include "Rhythmic.hpp"
#include "Measure.hpp"

#include <stdio.h>
#include <iostream>
#include <vector>

using namespace std;


class Rhythm_tree : protected Rhythmic{
protected:
    int dur;
    vector<Measure> lmeasure;
public:
    Rhythm_tree(Rhythm_tree const &self);

    Rhythm_tree(int dur, vector<Measure> lmeasure);

    Rhythm_tree(int** rhythm_tree);

    void set_dur(int dur);

    void set_lmeasure(vector<Measure> lmeasure);

    int get_dur() const;

    vector<Measure> get_lmeasure() const;

    int** OMget_rhythm_tree()const;

    string toString() const;

};


#endif /* Rhythm_tree_hpp */
```

Voice.hpp

```
//
//  Voice.hpp
//  IOM
//
//  Created by Geoffroy Zoetardt on 12/04/16.
//  Copyright  2016 Geoffroy Zoetardt. All rights reserved.
//

#ifndef Voice_h
#define Voice_h

#include "OM_Object.hpp"
```

```
13 #include "Rhythmic.hpp"
14 #include "Rhythm_tree.hpp"
15 #include "ChordSeq.hpp"
16
17 class Voice : public Rhythmic{
18 private:
19     Rhythm_tree tree;
20     ChordSeq lchord;
21     int tempo;
22     int legato;
23     vector< vector<int> > ties;
24
25 public:
26     Voice(Voice const &self); //Voice object
27
28     Voice(Rhythm_tree tree, ChordSeq lchord, int tempo, int legato, vector<
          vector<int> > ties);  //ties vector & chordseq
29
30     Voice(Rhythm_tree tree, ChordSeq lchord, int tempo, int legato, int**
          ties);            //ties array & chordseq
31
32     Voice(int** rhythm_tree, int** lchord, int tempo, int legato, int** ties
          );
33
34     void set_rhythm_tree(Rhythm_tree tree);
35     void set_lchord(ChordSeq lchord);
36     void set_tempo(int tempo);
37     void set_legato(int legato);
38     void set_ties(vector< vector<int> >ties);
39
40     Rhythm_tree get_rhythm_tree()const;
41     ChordSeq get_lchord()const;
42     int get_tempo()const;
43     int get_legato()const;
44     vector< vector<int> > get_ties()const;
45
46     int** OMget_rhythm_tree()const;
47     int** OMget_lchord()const;
48     int** OMget_ties()const;
49
50     string toString()const;
51 };
52
53 #endif /* Voice_h */
```

D.2 Lisp

D.2.1 Musical-objects

musicvar_objects.lisp

```
1 ;(in-package "omglpp")
2 (in-package :om)
3
4 ;;;; This file defines the search method for using GeLisp into OpenMusic and
      the OpenMusic's variables
5 ;;;; For now, four are efficient: IntVar, IntVarList, BoolVar, BoolVarList
```

```
;;;; For more information on the project "interface of GeLisp into OpenMusic" see the documentations in https://github.com/Zoetti/openmusicthesis2016
;;;; Realized in 2016 by Geoffroy Zoetardt (geoffroy.zoetardt@gmail.com)

(defparameter *space-gl* nil)
(defparameter *space-glDFS* nil)
(defparameter *space-glBAB* nil)

(defparameter *solvect* nil)
(defparameter *solvectDFS* nil)
(defparameter *solvectBAB* nil)

;;;;Function of initialization and execution of the constraint programming part.
;;;;list-cstrt provides all constraint objects needed for the execution of the model.
;;;;nsol indicates the number of wanted solutions to be returned.
;;;;This function is basic one, the following (getDFSSolutions, getBABSolutions) are made to be optimized.

(defun getSolutions (list-cstrt nsol)
  ;;;;initialization phase
  ;initialization of the search space, set the space of each variable and update its ptr to the gecode variable, then apply the constraints
  (let* ((list-var '())(sv *solvect*))
    ;initialization of the space
    (defparameter space-gl (new_glspace))
    (loop for cstrt in list-cstrt do
          (progn
            (loop for x in (get-var cstrt) do
                  ;loop avoiding the reinitialization of already initialize variable.
                  ;Furthemore, it appends list-var with the variables then list-var will have all variables only once.
                  (if (equal (member x list-var) nil)
                      (progn
                        (set-space x space-gl)
                        (setf list-var (append list-var (list x)))
                        )
                    )
                  )
            ;Apply the constraint cstrt on its variables
            (apply-constraint cstrt space-gl)
            )
          )
  ;computation phase
    ;branching
    (glspace_branch space-gl)
    ;initialization of the vector of solutions
    (setf sv (glspace_createglspacevector space-gl))
    ;run the search
    (glspace_rundfs space-gl nsol sv)
    (setf *solvect* sv)
    ;reorganization of the solutions to look like ((var1_sol1 var2_sol1) (var1_sol2 var2_sol2))
    (reorganization-list-of-list-sol (loop for var in list-var collect (getSolForOneVar sv var nsol)))
    )
```

```
55    )
56
57  ;;;;Function of initialization and execution of the constraint programming
        part.
58  ;;;;list-cstrt provides all constraint objects needed for the execution of
        the model.
59  ;;;;nsol indicates the number of wanted solutions to be returned.
60  ;;;;This function can be optimized with new arguments to refine the search.
61  ;;;;It focuses on the DFS execution.
62  (defun getDFSSolutions (list-cstrt nsol)
63    ;initialization phase
64    (let* ((list-var '())(sv *solvect*))
65      (defparameter space-glDFS (new_glspace))
66      (loop for cstrt in list-cstrt do
67            (progn
68              (loop for x in (get-var cstrt) do
69                    (if (equal (member x list-var) nil)
70                        (progn
71                          (set-space x space-glDFS)
72                          (setf list-var (append list-var (list x)))
73                          )
74                      )
75                    )
76              (apply-constraint cstrt space-glDFS)
77              )
78            )
79    ;computation phase
80      (glspace_branch space-glDFS)
81      (setf sv (glspace_createglspacevector space-glDFS))
82      (glspace_rundfs space-glDFS nsol sv)
83      (setf *solvectDFS* sv)
84      (reorganization-list-of-list-sol (loop for var in list-var collect (
            getSolForOneVar sv var nsol)))
85      )
86    )
87
88  ;;;;Function of initialization and execution of the constraint programming
        part.
89  ;;;;list-cstrt provides all constraint objects needed for the execution of
        the model.
90  ;;;;nsol indicates the number of wanted solutions to be returned.
91  ;;;;This function can be optimized with new arguments to refine the search.
92  ;;;;It focuses on the BAB execution.
93  (defun getBABSolutions (list-cstrt nsol)
94    ;initialization phase
95    ;initializeation of the search space, set the space of each variable and
          update its ptr to the gecode variable, then apply constraints
96     (let* ((list-var '())(sv *solvectBAB*))
97      (defparameter space-glBAB (new_glspace))
98      (loop for cstrt in list-cstrt do
99            (progn
100             (loop for x in (get-var cstrt) do
101                   (if (equal (member x list-var) nil)
102                       (progn
103                         (set-space x space-glBAB)
104                         (setf list-var (append list-var (list x)))
105                         )
106                     )
107                   )
```

```
108             (apply-constraint cstrt space-glBAB)
109             )
110           )
111   ;computation phase
112     (glspace_branch space-glBAB)
113     (setf sv (glspace_createglspacevector space-glBAB))
114     (glspace_runbab space-glBAB nsol sv)
115     (setf *solvectBAB* sv)
116     (reorganization-list-of-list-sol (loop for var in list-var collect (
          getSolForOneVar sv var nsol)))
117     )
118   )
119
120
121 ;;;;Function of initialization and execution of the constraint programming
      part.
122 ;;;;list-cstrt provides all constraint objects needed for the execution of
      the model.
123 ;;;;nsol indicates the number of wanted solutions to be returned.
124 ;;;;This function get the next solutions, it MUST be executed after the
      getSolutions function or will cause error.
125 (defun getNextSolutions (list-cstrt nsol)
126   ;initialization phase
127   ;initializeation of the search space, set the space of each variable and
        update its ptr to the gecode variable, then apply constraints
128   (let* ((list-var '())(sv *solvect*))
129     (loop for cstrt in list-cstrt collect
130             (loop for x in (get-var cstrt) do
131                   (if (equal (member x list-var) nil)
132                         (setf list-var (append list-var (list x)))
133                     )
134                   )
135             )
136   ;computation phase
137     (glspace_nextdfssolution space-gl nsol sv)
138     (setf *solvect* sv)
139                                         ;retrieval phase
140     (reorganization-list-of-list-sol (loop for var in list-var collect (
          getSolForOneVar sv var nsol)))
141     )
142   )
143
144 ;;;;Function of initialization and execution of the constraint programming
      part.
145 ;;;;list-cstrt provides all constraint objects needed for the execution of
      the model.
146 ;;;;nsol indicates the number of wanted solutions to be returned.
147 ;;;;This function get the next DFS solutions, it MUST be executed after the
      getDFSSolutions function or will cause error.
148 (defun getNextDFSSolutions (list-cstrt nsol)
149   ;initialization phase
150   ;initializeation of the search space, set the space of each variable and
        update its ptr to the gecode variable, then apply constraints
151   (let* ((list-var '())(sv *solvectDFS*))
152     (loop for cstrt in list-cstrt collect
153             (loop for x in (get-var cstrt) do
154                   (if (equal (member x list-var) nil)
155                         (setf list-var (append list-var (list x)))
156                     )
```

```
                        )
                )
    ;computation phase
      (glspace_nextdfssolution space-glDFS nsol sv)
      (setf *solvectDFS* sv)
      (reorganization-list-of-list-sol (loop for var in list-var collect (getSolForOneVar sv var nsol)))
      )
    )

;;;;Function of initialization and execution of the constraint programming part.
;;;;list-cstrt provides all constraint objects needed for the execution of the model.
;;;;nsol indicates the number of wanted solutions to be returned.
;;;;This function get the next BAB solutions, it MUST be executed after the getBABSolutions function or will cause error.
(defun getNextBABSolutions (list-cstrt nsol)
    ;initialization phase
    ;initializeation of the search space, set the space of each variable and update its ptr to the gecode variable, then apply constraints
    (let* ((list-var '())(sv *solvectBAB*))
      (loop for cstrt in list-cstrt collect
                (loop for x in (get-var cstrt) do
                      (if (equal (member x list-var) nil)
                            (setf list-var (append list-var (list x)))
                        )
                      )
                )
    ;computation phase
      (glspace_nextbabsolution space-glBAB nsol sv)
      (setf *solvectBAB* sv)
      (reorganization-list-of-list-sol (loop for var in list-var collect (getSolForOneVar sv var nsol)))
      )
    )


;Reorganize a list of list like  ((var1_sol1 var1_sol2) (var2_sol1 var2_sol2)) to be like  ((var1_sol1 var2_sol1) (var1_sol2 var2_sol2))
(defun reorganization-list-of-list-sol (lofl-sol)
    (setf size-sublist (list-length (nth 0 lofl-sol)))
    (setf reorg-sol
          (loop for i from 0 to (- size-sublist 1) collect
                (setf subsol (loop for elem in lofl-sol collect (nth i elem)))
                )
          )
    )


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;                              INTVAR OBJECT                              ;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

```

```lisp
;Intvar lisp object is the correspondance to the Gecode intvar.
;lb is the lower bound and ub is the upper bound.
;ptr is the C++ pointer to the real Gecode object.
(defclass! IntVar ()
          (
           (lb :initform 0 :initarg :lb :accessor lb :type integer)
           (ub :initform 0 :initarg :ub :accessor ub :type integer)
           (ptr :initform nil :ptr :accessor ptr)
           )
          (:icon 1010)
)

;Set the space of self with space-gl
(defmethod set-space ((self IntVar) space-gl)
  (setf (slot-value self 'ptr) (glspace_newintvar_minmax space-gl (lb self) (ub self)))
)

;method to return the solution of an intVar after that the script has been run.
;solvect is the c++ pointer to the vector of solutions
;nsol is the number of required solutions.
(defmethod getSolForOneVar (solvect (self IntVar) nsol)
 (if (equal (slot-value self 'ptr) nil)
     (print "IntVar is not initialized")
   (progn
     (let ((pointersol)
           (arraysol))
       (setf pointersol (glintvar_getintvarsol (slot-value self 'ptr) solvect nsol))
       (setf arraysol (om_object_put_length_firstI pointersol))
       (c-array-to-lisp-listI arraysol)
       )
     )
   )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;                     INTVARLIST OBJECT                     ;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;IntVarList lisp object is the correspondance to the Gecode intVarArgs.
;n is the size of the list, lb is the lower bound and ub is the upper bound.
;ptr is the C++ pointer to the real Gecode object.
(defclass! IntVarList ()
          (
           (nsize :initform 0 :initarg :nsize :accessor nsize :documentation "Size of the list")
           (lb :initform 0 :initarg :lb :accessor lb :documentation "Minimum value")
           (ub :initform 0 :initarg :ub :accessor ub :documentation "Maximum value")
           (ptr :initform nil :ptr :accessor ptr)
           )
```

```
          (:icon 1011)
)

;Set the space of self with space-gl
(defmethod set-space ((self IntVarList) space-gl)
  (setf (slot-value self 'ptr) (new_GlIntVarList_minmax space-gl (nsize self) (lb self) (ub self)))
)

;method to return the solution of an intVarList after that the script has been run.
;solvect is the c++ pointer to the vector of solutions
;nsol is the number of required solutions.
(defmethod getSolForOneVar (solvect (self IntVarList) nsol)
 (if (equal (slot-value self 'ptr) nil)
     (print "IntVarList is not initialized")
   (progn
     (let ((pointersol)
           (arraysol))
       (setf pointersol (glintvarlist_getintvarlistsol (slot-value self 'ptr) solvect nsol))
       (setf arraysol (om_object_put_length_first2DI pointersol))
       (setf arr (c-array2d-to-lisp-list2dI arraysol))
       arr
       )
     )
   )
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;                    BOOLVAR OBJECT                    ;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;Boolvar lisp object is the correspondance to the Gecode boolVar.
;lb is the lower bound and ub is the upper bound. 0<=lb<=1 and 0<=ub<=1
;ptr is the C++ pointer to the real Gecode object.
(defclass! BoolVar ()
          (
           (lb :initform 0 :initarg :lb :accessor lb)
           (ub :initform 0 :initarg :ub :accessor ub)
           (ptr :initform nil :ptr :accessor ptr)
           )
          (:icon 1012)
)

;Set the space of self with space-gl
(defmethod set-space ((self BoolVar) space-gl)
  (setf (slot-value self 'ptr) (glspace_newboolvar space-gl (lb self) (ub self)))
)

;method to return the solution of an boolVar after that the script has been run.
;solvect is the c++ pointer to the vector of solutions
```

```
;nsol is the number of required solutions.
(defmethod getSolForOneVar (solvect (self BoolVar) nsol)
 (if (equal (slot-value self 'ptr) nil)
     (print "BoolVar is not initialized")
   (progn
     (let ((pointersol)
           (arraysol))
       (setf pointersol (glboolvar_getboolvarsol (slot-value self 'ptr)
           solvect nsol))
       (setf arraysol (om_object_put_length_firstB pointersol))
       (c-array-to-lisp-listI arraysol)
       )
     )
   )
)

;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;                    BOOLVARLIST OBJECT
                                ;;;;;;;;;;;;;;;;;;;;;;
;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;BoolVarList lisp object is the correspondance to the Gecode boolVarArgs.
;n is the size of the list, lb is the lower bound and ub is the upper bound.
;ptr is the C++ pointer to the real Gecode object.
(defclass! BoolVarList ()
           (
            (n :initform 0 :initarg :n :accessor n)
            (lb :initform 0 :initarg :lb :accessor lb)
            (ub :initform 0 :initarg :ub :accessor ub)
            (ptr :initform nil :ptr :accessor ptr)
            )
           (:icon 1013)
           )

;Set the space of self with space-gl
(defmethod set-space ((self BoolVarList) space-gl)
  (setf (slot-value self 'ptr) (new_glBoolVarList space-gl (n self) (lb self
      ) (ub self)))
)

;method to return the solution of an boolVarList after that the script has
    been run.
;solvect is the c++ pointer to the vector of solutions
;nsol is the number of required solutions.
(defmethod getSolForOneVar (solvect (self BoolVarList) nsol)
 (if (equal (slot-value self 'ptr) nil)
     (print "BoolVarList is not initialized")
   (progn
     (let ((pointersol)
           (arraysol))
       (setf pointersol (glboolvarlist_getboolvarlistsol (slot-value self '
           ptr) solvect nsol))
       (setf arraysol (om_object_put_length_first2DB pointersol))
       (setf arr (c-array2d-to-lisp-list2dI arraysol))
       arr
```

```
          )
        )
    )
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;                    NOTEVAR OBJECT                    ;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Object that represent a variable note that can be used with a constraint program.
;It is initialized with a script to which it is linked, a pitch domain, a velocity domain and a duration domain;
;nvprt is the pointer to the real gecode object that represent the variable in c++.
(defclass! NoteVar ()
           (
            (pitchdom :initform (list 6000) :initarg :pitchdom :accessor pitchdom :type list)
            (veldom :initform (list 100) :initarg :veldom :accessor veldom :type list)
            (durdom :initform (list 1000) :initarg :durdom :accessor durdom :type list)
            (ptr :initform nil :ptr :accessor ptr)
            )
           (:icon 1001)
           )

;Set space and initialize the Gecode's pointer to the NoteVar object
(defmethod set-space ((self NoteVar) space-gl)
 (setf pid (lisp-list-to-c-arrayi (pitchdom self)))
 (setf ved (lisp-list-to-c-arrayi (veldom self)))
 (setf dud (lisp-list-to-c-arrayi (durdom self)))
 (setf pd (om_object_init_vecti pid))
 (setf vd (om_object_init_vecti ved))
 (setf dd (om_object_init_vecti dud))
 (setf (slot-value self 'ptr) (new_notevar space-gl pd vd dd))
)

;Method to get solutions for a specific NoteVar, it needs a vector of solution and the object notevar of which we want the solution.
(defmethod getSolForOneVar (solvect (self NoteVar) nsol)
 (setq pointersol (notevar_getnotevarallsol (slot-value self 'ptr) solvect nsol))
 (setq size_vn (note_getsizevect pointersol))
 (loop for i from 0 to (- size_vn 1) collect (cnote-to-omnote (new_cnote_self (note_getnotefromvectornote pointersol i))))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;                    CHORDVAR OBJECT                    ;;;;;;;;;;;;;;;;;;;;;
```

```lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Object that represent a variable chord that can be used with a constraint program.
;It is initialized with a script to which it is linked, a pitch domain, a velocity domain, a duration domain and an offset domain.
;cvprt is the pointer to the real gecode object that represent the variable in c++.

(defclass! ChordVar ()
          (
           (pitchdom :initform (list 6000) :initarg :pitchdom :accessor pitchdom :type list)
           (veldom :initform (list 100) :initarg :veldom :accessor veldom :type list)
           (durdom :initform (list 1000) :initarg :durdom :accessor durdom :type list)
           (offdom :initform (list 1000) :initarg :offdom :accessor offdom :type list)
           (ptr :initform nil :ptr :accessor ptr)
           )
          (:icon 1002)
)

;Set space and initialize the Gecode's pointer to the ChordVar object
(defmethod set-space ((self ChordVar) space-gl)
  (setf pid (lisp-list-to-c-arrayi (pitchdom self)))
  (setf ved (lisp-list-to-c-arrayi (veldom self)))
  (setf dud (lisp-list-to-c-arrayi (durdom self)))
  (setf offd (lisp-list-to-c-arrayi (offdom self)))

  (setf pd (om_object_init_vecti pid))
  (setf vd (om_object_init_vecti ved))
  (setf dd (om_object_init_vecti dud))
  (setf ofd (om_object_init_vecti offd))
  (setf (slot-value self 'ptr) (new_chordvar space-gl pd vd dd ofd))
)

;Method to get solutions for a specific ChordVar, it needs a vector of solution and the object chordvar of which we want the solution.
(defmethod getSolForOneVar (solvect (self ChordVar) nsol)
 (setq pointersol (chordvar_getchordvarallsol (slot-value self 'ptr) solvect nsol))
 (setq size_vc (chord_getsizevect pointersol))
 (loop for i from 0 to (- size_vc 1) collect (cchord-to-omchord (new_cchord_self (chord_getchordfromvectorchord pointersol i))))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;                    CHORDSEQVAR OBJECT                    ;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;Object that represent a variable chord-sequence that can be used with a constraint program.
```

```
434 ;It is initialized with a script to which it is linked, a pitch domain, a
      velocity domain, a duration domain, an offset domain and an onset domain
      .
435 ;csvprt is the pointer to the real gecode object that represent the variable
       in c++.
436
437 (defclass! ChordSeqVar ()
438           (
439            (pitchdom :initform (list 6000) :initarg :pitchdom :accessor
                 pitchdom :type list)
440            (veldom :initform (list 100) :initarg :veldom :accessor veldom :
                 type list)
441            (durdom :initform (list 1000) :initarg :durdom :accessor durdom :
                 type list)
442            (offdom :initform (list 1000) :initarg :offdom :accessor offdom :
                 type list)
443            (ondom :initform (list 1000) :initarg :ondom :accessor ondom :
                 type list)
444            (ptr :initform nil :ptr :accessor ptr)
445            )
446           (:icon 1003)
447 )
448
449 ;Set space and initialize the Gecode's pointer to the ChordSeqVar object
450 (defmethod set-space ((self ChordSeqVar) space-gl)
451  (setf pid (lisp-list-to-c-arrayi (pitchdom self)))
452  (setf ved (lisp-list-to-c-arrayi (veldom self)))
453  (setf dud (lisp-list-to-c-arrayi (durdom self)))
454  (setf offd (lisp-list-to-c-arrayi (offdom self)))
455  (setf ond (lisp-list-to-c-arrayi (ondom self)))
456
457  (setf pd (om_object_init_vecti pid))
458  (setf vd (om_object_init_vecti ved))
459  (setf dd (om_object_init_vecti dud))
460  (setf ofd (om_object_init_vecti offd))
461  (setf od (om_object_init_vecti ond))
462  (setf (slot-value self 'ptr) (new_chordseqvar space-gl pd vd dd ofd od))
463 )
464
465 ;Method to get solutions for a specific ChordSeqVar, it needs a vector of
      solution and the object chordseqvar of which we want the solution.
466 (defmethod getSolForOneVar (solvect (self ChordSeqVar) nsol)
467  (setq pointersol (chordseqvar_getchordseqvarallsol (slot-value self 'ptr)
       solvect nsol))
468  (setq size_vc (chordseq_getsizevect pointersol))
469  (loop for i from 0 to (- size_vc 1) collect
470        (cchordseq-to-omchordseq (new_cchordseq_self (
             chordseq_getchordseqfromvectorchordseq pointersol i))))
471 )
```

D.2.2 Constraints

constraints.lisp

```
1
2 ;;;; This file provides the implementation of constraint objects for the
      interface of OpenMusic and of some basic constraint function from GeLisp
      .
```

```lisp
;;;; For more information on the project "interface of GeLisp into OpenMusic" see the documentations in https://github.com/Zoetti/openmusicthesis2016
;;;; Realized in 2016 by Geoffroy Zoetardt (geoffroy.zoetardt@gmail.com)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;                              CONSTRAINTS OBJECTS                              ;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;(in-package "omglpp")
(in-package :om)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; Constraint with one argument and a constraint function ;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defclass! Constraint-1 ()
           ((constraint :initform nil :initarg :constraint :accessor constraint)
            (var1 :initform nil :initarg :var1 :accessor var1)
            )
)

;;;; method applying the constraint of the constraint object on its variable.
(defmethod apply-constraint ((self Constraint-1) space)
  (setf (symbol-function 'const) (constraint self))
  (const space (ptr (var1 self)))
)

;;;; initialize the space of the variable.
(defmethod set-cstrt-space ((self Constraint-1) space)
  (set-space (var1 self) space)
)

;;;; Return the variable of the object.
(defmethod get-var ((self Constraint-1))
  (setq list-var (list))
  (setq list-var (append list-var (list (var1 self))))
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;; Constraint with two arguments and a constraint function ;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(defclass! Constraint-2 ()
           ((constraint :initform nil :initarg :constraint :accessor constraint)
            (var1 :initform nil :initarg :var1 :accessor var1)
            (var2 :initform nil :initarg :var2 :accessor var2)
            )
)

;;;; method applying the constraint of the constraint object on its variables.
(defmethod apply-constraint ((self Constraint-2) space)
  (setf (symbol-function 'const) (constraint self))
```

```
51   (const space (slot-value (var1 self) 'ptr) (slot-value (var2 self) 'ptr))
52 )
53
54 ;;;; initialize the space of the variables.
55 (defmethod set-cstrt-space ((self Constraint-2) space)
56   (set-space (var1 self) space)
57   (set-space (var2 self) space)
58 )
59
60 ;;;; Return the variables of the object.
61 (defmethod get-var ((self Constraint-2))
62   (setq list-var '())
63   (setq list-var (append list-var (list (var1 self))))
64   (setq list-var (append list-var (list (var2 self))))
65   list-var
66 )
67
68 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
69 ;;;;;;; Constraint with three arguments and a constraint function ;;;;;;;
70 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
71 (defclass! Constraint-3 ()
72           ((constraint :initform nil :initarg :constraint :accessor
                constraint)
73            (var1 :initform nil :initarg :var1 :accessor var1)
74            (var2 :initform nil :initarg :var2 :accessor var2)
75            (var3 :initform nil :initarg :var3 :accessor var3)
76            )
77 )
78
79 ;;;; method applying the constraint of the constraint object on its
       variables.
80 (defmethod apply-constraint ((self Constraint-3) space)
81   (setf (symbol-function 'const) (constraint self))
82   (const space (ptr (var1 self)) (ptr (var2 self)) (ptr (var3 self)))
83 )
84
85 ;;;; initialize the space of the variables.
86 (defmethod set-cstrt-space ((self Constraint-3) space)
87   (set-space (var1 self) space)
88   (set-space (var2 self) space)
89   (set-space (var3 self) space)
90 )
91
92 ;;;; Return the variables of the object.
93 (defmethod get-var ((self Constraint-3))
94   (setq list-var (list))
95   (setq list-var (append list-var (list (var1 self))))
96   (setq list-var (append list-var (list (var2 self))))
97   (setq list-var (append list-var (list (var3 self))))
98 )
99
100
101 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
102 ;;;;;;; Constraint with four arguments and a constraint function ;;;;;;;
103 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
104 (defclass! Constraint-4 ()
105           ((constraint :initform nil :initarg :constraint :accessor
                constraint)
106            (var1 :initform nil :initarg :var1 :accessor var1)
```

```
            (var2 :initform nil :initarg :var2 :accessor var2)
            (var3 :initform nil :initarg :var3 :accessor var3)
            (var4 :initform nil :initarg :var4 :accessor var4)
            )
)

;;;; method applying the constraint of the constraint object on its
    variables.
(defmethod apply-constraint ((self Constraint-4) space)
  (setf (symbol-function 'const) (constraint self))
  (const space (ptr (var1 self)) (ptr (var2 self)) (ptr (var3 self)) (ptr (
      var4 self)))
)

;;;; initialize the space of the variables.
(defmethod set-cstrt-space ((self Constraint-4) space)
  (set-space (var1 self) space)
  (set-space (var2 self) space)
  (set-space (var3 self) space)
  (set-space (var4 self) space)
)

;;;; Return the variables of the object.
(defmethod get-var ((self Constraint-4))
  (setq list-var (list))
  (setq list-var (append list-var (list (var1 self))))
  (setq list-var (append list-var (list (var2 self))))
  (setq list-var (append list-var (list (var3 self))))
  (setq list-var (append list-var (list (var4 self))))
)

;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;;;;;;;;;;;;;;;;;;;;                           CONSTRAINTS
                                   ;;;;;;;;;;;;;;;;;;;;;;
;
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;


;;;; These constraint function are based on GeLisp and through that on
    Gecode's implementation of constraint functions.

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;          Domain constraints          ;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;; Mono-domain
;dom_onIntVar_default

;;;; On IntVarList
;dom_onIVA_default

;;;; On IntVar With bounds
;dom_onIntVarWithBounds_default

;;;; On IntVarList with bounds
;dom_onIVAWithBounds_default

```

```
;;;; On IntVar with int and boolvar
;dom_onIntVarAndIntAndBoolVar_default

;;;; On IntVar with bounds and boolvar
;dom_onIntVarAndBoolVarWithBounds_default

;;;; On IntVar from IntVar
(defmethod domain-constraint ((var1 IntVar) (var2 IntVar))
  ;creation of the constraint object
  (setq cstrt (make-instance 'Constraint-2 :var1 var1 :var2 var2))
  ;constraint phase
  (setf (constraint cstrt)
        (lambda (space x y) (dom_onIntVarFromIntVar_default space x y))
        )
  ;return
  cstrt
)


;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;          Distinct constraints          ;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

(defmethod distinct-constraint ((var1 IntVarList))
  ;creation of the constraint object
  (setq cstrt (make-instance 'Constraint-1 :var1 var1))
  ;constraint phase
  (setf (constraint cstrt)
        (lambda (space x) (distinct_default space x))
        )
  ;return
  cstrt
)

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;            Rel constraints             ;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

;;;; On IntVar default
(defmethod rel-constraint ((v1 IntVar) r (v2 IntVar))
  ;creation of the constraint object
  (setq cstrt (make-instance 'Constraint-2 :var1 v1 :var2 v2))
  ;constraint phase

  (setf (constraint cstrt)
        (lambda (space x y)(rel_onIntVar_default space x r y))
        )
  ;return
  cstrt
)

(defmethod rel-constraint ((v1 BoolVarList) r (v2 BoolVar))
  ;creation of the constraint object
  (setq cstrt (make-instance 'Constraint-2 :var1 v1 :var2 v2))
  ;constraint phase

  (setf (constraint cstrt)
        (lambda (space x y)(rel_onBVAAndBoolVar_default space (
           glBoolVarList_get x) r y))
```

```
216         )
217   ;return
218   cstrt
219 )
220
221 ;;;; On IntVar and IntVarList
222 (defmethod rel-constraint ((v1 IntVarList) r (v2 IntVar))
223   ;creation of the constraint object
224   (setq cstrt (make-instance 'Constraint-2 :var1 v1 :var2 v2))
225   ;constraint phase
226
227   (setf (constraint cstrt)
228         (lambda (space x y) (rel_onIntVarAndIVA_default space (
              glIntVarList_get x) r y))
229         )
230   ;return
231   cstrt
232 )
233
234 ;;;;rel_onIntVarAndInt_default
235
236 ;;;;rel_onIVAAndInt_default
237
238 ;;;;rel_on2IntVarAndBoolVar_default
239
240 ;;;;rel_onIntVarAndIntAndBoolVar_default
241
242 ;;;;rel_onIVA_default
243
244 ;;;;rel_onIVAAndIVA_default
245
246 ;;;;rel_on2BoolVar_default
247
248 ;;;;rel_on3BoolVar_default
249
250 ;;;;rel_onBVAAndBoolVar_default
251
252 ;;;;rel_onBoolVarAndInt_default
253
254 ;;;;rel_on2BoolVarAndInt_default
255
256 ;;;;rel_onBVAAndInt_default
257
258 ;;;;rel_on2BVA_default
259
260 ;;;;rel_onBVA_default
261
262
263 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
264 ;;;;;;;;;;          Element constraints          ;;;;;;;;;
265 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
266
267 ;;;;element_onIVAAnd2IntVar_default
268
269 ;;;;element_onIVAAndIntVarAndInt_default
270
271 ;;;;element_onIVAAnd3IntVarAnd2Int_default
272
273 ;;;;element_onBVAAndIntVarAndBoolVar_default
```

```
274
275 ;;;;element_onBVAAndIntVarAndInt_default
276
277 ;;;;element_onBVAAnd2IntVarAnd2IntAndBoolVar_default
278
279
280 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
281 ;;;;;;;;;;    Cardinality constraints          ;;;;;;;;;
282 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
283
284 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
285 ;;;;;;;;;;       Channel constraints          ;;;;;;;;;
286 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
287
288 ;;;;channel_onIVAAndIVA_default
289
290 ;;;;channel_on2IVAAnd2IntVar_default
291
292 ;;;;channel_onBoolVarAndIntVar_default
293
294 ;;;;channel_onIntVarAndBoolVar_default
295
296 ;;;;channel_onBVAAndIntVar_default
297
298 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
299 ;;;;;;;;;;        Sorted constraints           ;;;;;;;;;
300 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
301
302 ;;;;sorted_onIVAAndIVA_default
303
304 ;;;;sorted_on3IVA_default
305
306 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
307 ;;;;;;;;;;        Count constraints            ;;;;;;;;;
308 ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
309
310 ;;;;count_onIVAAnd2Int_default
311
312 ;;;;count_onIVAAndIntVarAndInt_default
313
314 ;;;;count_onIVAAndIntAndIntVar_default
315
316 ;;;;count_onIVAAnd2IntVar_default
317
318 ;;;;count_on2IVA_default
```

created-constraints.lisp

```
1
2 ;;;; This file provides not basic constraints and aims to be improved by
      every created constraint of the user.
3 ;;;; For more information on the project "interface of GeLisp into OpenMusic
      " see the documentations in https://github.com/Zoetti/
      openmusicthesis2016
4 ;;;; Realized in 2016 by Geoffroy Zoetardt (geoffroy.zoetardt@gmail.com)
5
6
```

```lisp
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;         CREATED CONSTRAINTS         ;;;;;;;;;;;;;;;;;;;;;;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(in-package :om)

;;;;;;;;;;;;;;;;;;;;;;;;
;;;; ALL-INTERVALS ;;;;
;;;;;;;;;;;;;;;;;;;;;;;;

(defmethod all-intervals ((v1 IntVarList) (v2 IntVarList))
  ;creation of the constraint object
  (setq cstrt (make-instance 'Constraint-2 :var1 v1 :var2 v2))
  ;constraint phase

  (setf (constraint cstrt)
        (lambda (sp pitchClasses intervals)
          (progn

            (setq nOfPitch (nsize v1))
    ;;;;;;;;;;;;;;;;;;;;;;;;;;; auxiliary variables ;;;;;;;;;;;;;;;;;;;;;;;;;;

    ; only used for mod divisor
            (setq divisor (GlSpace_newIntVar_minmax sp nOfPitch nOfPitch))
    ; only used to express the mod constraint. It will contains the true pitch differences.
            (setq diffpitch (new_GlIntVarList_minmax sp (- nOfPitch 1) (- 0 (- nOfPitch 1)) (- nOfPitch 1)))
    ;used because domain must be shifted because Gecode doesn't manage modulo with negative dividend
            (setq diffpitchshifted (new_GlIntVarList_minmax sp (- nOfPitch 1) 1 (+ nOfPitch (- nOfPitch 1))))

            ;;;;;;;;;;;;;;;;;;;;;;;;;;; constraints ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    ;coeff used to shift pitch differences
            (setq coeffListForShift (new_GlIntList_sizeAndDefaultValue 2 1))

    ;inversionalEquivalentInterval
            (loop for x from 0 to (- (nsize v2) 1) do

          ;establish needed coefficients
                  (setq coeffList (new_GlIntList_sizeAndDefaultValue nOfPitch 0))
                  (GlIntList_setVar coeffList x -1)
                  (GlIntList_setVar coeffList (+ x 1) 1)

          ;diffpitch(i) = pitchClasses(i+1) - pitchClasses(i)
                  (linear_onIntArgsAndIVAAndIntVar_default sp (GlIntList_get coeffList) (GlIntVarList_get pitchClasses) :IRT_EQ (GlIntVarList_getVar diffpitch x))

          ;diffpitchshifted(i) = diffpitch(i) + 12
                  (setq tempVarList (new_GlIntVarList))
```

```
53                  (GlIntVarList_add tempVarList (GlIntVarList_getVar
                        diffpitch x))
54                  (GlIntVarList_add tempVarList divisor)
55                  (linear_onIntArgsAndIVAAndIntVar_default sp (GlIntList_get
                        coeffListForShift) (GlIntVarList_get tempVarList) :
                        IRT_EQ (GlIntVarList_getVar diffpitchshifted x))
56
57          ;intervals(i) = diffpitchshifted(i) mod 12
58                  (mod_default sp (GlIntVarList_getVar diffpitchshifted x)
                        divisor (GlIntVarList_getVar intervals x))
59
60          ;additionnal constraint to have a symmetric serie : intervals(i) +
                intervals(11 - 1 -i) = 12
61                  (setq tempVarList (new_GlIntVarList))
62                  (GlIntVarList_add tempVarList (GlIntVarList_getVar
                        intervals x))
63                  (GlIntVarList_add tempVarList (GlIntVarList_getVar
                        intervals (- (- nOfPitch 2) x)))
64                  (linear_onIntArgsAndIVAAndIntVar_default sp (GlIntList_get
                        coeffListForShift) (GlIntVarList_get tempVarList) :
                        IRT_EQ divisor)
65
66                  )
67
68      ;distinctPitchClasses
69              (distinct_default sp (GlIntVarList_get pitchClasses))
70
71      ;distinctIntervals
72              (distinct_default sp (GlIntVarList_get intervals))
73
74      ;symmetry break (avoid transposition) : pitchClasses(0) = 0
75              (dom_onIntVar_default sp (GlIntVarList_getVar pitchClasses 0) 0)
76
77      ;redundant constraint (allows to prune more) :  pitchClasses(11) = 6
78              (dom_onIntVar_default sp (GlIntVarList_getVar pitchClasses (-
                    nOfPitch 1)) (/ nOfPitch 2))
79
80
81              ))
82          )
83    cstrt
84    )
```

D.2.3 OM++

OMplusplus.lisp

```
1  ;(in-package "omglpp")
2  (in-package :om)
3
4  ;;;; This file provides a conversion from C++ musical object from the OM++
       library to these of OpenMusic and vice versa
5  ;;;; For more information on the project "interface of GeLisp into OpenMusic
       " see the documentations in https://github.com/Zoetti/
       openmusicthesis2016
6  ;;;; Realized in 2016 by Geoffroy Zoetardt (geoffroy.zoetardt@gmail.com)
7
8  ;;;; Convert an OpenMusic note to a pointer of a c++ note.
```

```
(defun omnote-to-cnote (OMself)
  (new_cnote (coerce (midic OMself) 'double-float) (coerce (vel OMself) '
      integer) (coerce (dur OMself) 'double-float) (coerce (chan OMself) '
      integer))
)

;;;; Convert a c++ pointer referencing a note to an OpenMusic note.
(defun cnote-to-omnote (Cself)
  (make-instance 'note :midic (note_get_midic Cself)
                :vel (note_get_vel Cself)
                :dur (round (note_get_dur Cself))
                :chan (note_get_chan Cself))
)

;;;; Convert an OpenMusic chord to a pointer of a c++ chord.
(defun omchord-to-cchord (OMself)
  (new_cchord (lisp-list-to-c-array (lmidic OMself) :double) (
      lisp-list-to-c-array (lvel OMself) :int) (lisp-list-to-c-array (ldur
      OMself)) (lisp-list-to-c-array (loffset OMself)) (lisp-list-to-c-array
       (lchan OMself) :int))
)

;;;; Convert a c++ pointer referencing a chord to an OpenMusic chord.
(defun cchord-to-omchord (Cself)
  (make-instance 'chord :lmidic (coerce (c-array-to-lisp-list (
      chord_omget_lmidic Cself) :double) 'list)
                :lvel (coerce (c-array-to-lisp-list (chord_omget_lvel Cself
                    ) :int) 'list)
                :ldur (coerce (c-array-to-lisp-list (chord_omget_ldur Cself
                    ) :double) 'list)
                :loffset (coerce (c-array-to-lisp-list (chord_omget_loffset
                     Cself) :double) 'list)
                :lchan (coerce (c-array-to-lisp-list (chord_omget_lchan
                    Cself) :int) 'list)
                )
)

;;;; Convert an OpenMusic chord-seq to a pointer of a c++ chordseq.
(defun omchordseq-to-cchordseq (OMself)
  (new_cchordseq
   (lisp-list2d-to-c-array2d (lmidic OMself) :double)
   (lisp-list-to-c-array (lonset OMself) :int)
   (lisp-list2d-to-c-array2d (ldur OMself) :double)
   (lisp-list2d-to-c-array2d (lvel OMself) :int)
   (lisp-list2d-to-c-array2d (loffset OMself) :double)
   (lisp-list2d-to-c-array2d (lchan OMself) :int)
   (coerce (legato OMself) 'number)
  )
)

;;;; Convert a c++ pointer referencing a chordseq to an OpenMusic chord-seq.
(defun cchordseq-to-omchordseq (Cself)
  (make-instance 'chord-seq
                :lmidic (coerce (c-array2d-to-lisp-list2d (
                    chordseq_omget_lmidic Cself) :double) 'list)
                :lvel (coerce (c-array2d-to-lisp-list2di (
                    chordseq_omget_lvel Cself) :int) 'list)
                :ldur (coerce (c-array2d-to-lisp-list2d (
                    chordseq_omget_ldur Cself)) 'list)
```

```
                    :loffset (coerce (c-array2d-to-lisp-list2d (
                        chordseq_omget_loffset Cself) :double) 'list)
                    :lchan (coerce (c-array2d-to-lisp-list2di (
                        chordseq_omget_lchan Cself) :int) 'list)
                    :lonset (coerce (c-array-to-lisp-listi (
                        chordseq_omget_lonset Cself) :int) 'list)
                    :legato (coerce (chordseq_omget_legato Cself) 'number)
                    )
)

;;;; Convert an OpenMusic measure to a pointer of a c++ measure.
(defun ommeasure-to-cmeasure (OMself)
  (new_cmeasure (lisp-list2d-to-c-array2d (tree OMself)))
)

;;;; Convert a c++ pointer referencing a measure to an OpenMusic measure.
(defun cmeasure-to-ommeasure (Cself)
 (print (c-array2d-to-lisp-list2d (measure_omget_measure Cself) :int))
  (make-instance 'measure :tree (c-array2d-to-lisp-list2d (
      measure_omget_measure Cself) :int))
)

;(defun omvoice-to-cvoice (OMself)
 ; (print (tempo OMself))
  ;(new_cchordseq
   ;(lisp-list2d-to-c-array2d (tree OMself))
   ;(lisp-list-to-c-array (lonset OMself))
   ;(lisp-list2d-to-c-array2d (ldur OMself))
   ;(lisp-list2d-to-c-array2d (lvel OMself))
   ;(lisp-list2d-to-c-array2d (loffset OMself))
   ;(lisp-list2d-to-c-array2d (lchan OMself)) (legato OMself))
;)

;(defun cvoice-to-omvoice (Cself)
 ; (make-instance 'voice
  ;               :tree (list2d-to-rhythm-tree (c-array2d-to-lisp-list2d (
      voice_omget_rhythm_tree Cself)))
   ;              :chords (c-array2d-to-lisp-list2d (voice_omget_lchord
       Cself))
    ;             :tempo (voice_omget_tempo Cself)
     ;            :legato (voice_omget_legato Cself)
      ;           :ties (c-array2d-to-lisp-list2d (voice_omget_ties Cself))
       ;          )
;)
```

D.2.4 Utils

Utils.lisp

```
;(in-package "omglpp")
(in-package :om)

;;;; This file deals with the conversion from OM to C and vice versa
;;;; For more information on the project "interface of GeLisp into OpenMusic
    " see the documentations in https://github.com/Zoetti/
    openmusicthesis2016
;;;; Realized in 2016 by Geoffroy Zoetardt (geoffroy.zoetardt@gmail.com)

```

```
8  (defun number-to-double (i) (coerce i 'double-float))
9  (defun number-to-float (i) (coerce i 'float))
10
11 (defun coerce-list (liste type)
12   (case type
13     (:double (mapcar #'number-to-double liste))
14     (:float (mapcar #'number-to-float liste))
15     (:int (mapcar #'round liste))
16     (otherwise liste)))
17
18 ; Takes a list from OM and converts it into a C array.
19 ; The C-array is allocated here, but not freed
20 (defun lisp-list-to-c-array (liste &optional (type :double))
21   (push (length liste) liste)
22   (cffi::foreign-alloc type :initial-contents (coerce-list liste type)))
23
24 (defun lisp-list-to-c-arrayI (liste &optional (type :int))
25   (push (length liste) liste)
26   (cffi::foreign-alloc type :initial-contents (coerce-list liste type)))
27
28 ; Takes a list of lists from OM and converts it into a C array of arrays.
29 ; The C-array is allocated here, but not freed
30 (defun lisp-list2D-to-c-array2D (liste &optional (type :double))
31   (cffi::foreign-alloc :pointer :initial-contents  (flat_for_c liste type))
32 )
33
34 ; Takes a list of lists from OM and converts it into a C array of arrays.
35 ; The C-array is allocated here, but not freed
36 (defun lisp-list2D-to-c-array2DI (liste &optional (type :int))
37   (cffi::foreign-alloc :pointer :initial-contents  (flat_for_c liste type))
38 )
39
40 ; Takes C array and converts it into a lisp list.
41 ; The C-array is allocated here, but not freed
42 (defun c-array-to-lisp-listI (ptr  &optional (type :int))
43   (setq size (cffi::mem-aref ptr type 0))
44   (loop for i from 1 to size collect (cffi::mem-aref ptr type i)))
45
46 ; Takes C array and converts it into a lisp list.
47 ; The C-array is allocated here, but not freed
48 (defun c-array-to-lisp-list (ptr  &optional (type :double))
49   (setq size (cffi::mem-aref ptr type 0))
50   (loop for i from 1 to size collect (cffi::mem-aref ptr type i)))
51
52 ; Takes C array and converts it into a lisp list.
53 ; The C-array is allocated here, but not freed
54 (defun c-array-to-lisp-list-ptr (ptr size &optional (type :pointer))
55   (loop for i from 0 to (- size 1) collect (cffi::mem-aptr ptr type i)))
56
57 ; Flat a list to send it to C with all its information.
58 ; It principally consists to set the lenght of the list at the front to
     inform C of the list's size
59 (defun flat_for_c (liste type)
60 (setq lout nil)
61   (push (cffi::foreign-alloc type :initial-contents (coerce-list (list (
       length liste)) type)) lout)
62   (loop for x in liste
63         if (listp x)
```

```
64          do (progn (push (length x) x)(push (cffi::foreign-alloc type :
                initial-contents (coerce-list x type)) lout))
65          if (numberp x)
66          do (progn (setq lx (list x))(push (length lx) lx)(push (cffi::
                foreign-alloc type :initial-contents (coerce-list lx type)) lout
                ))
67          )
68    (nreverse lout)
69  )
70
71  ; Converts a rhythm-tree to a list of lists
72  (defun rhythm-tree-to-list2D (lin)
73    (setq lout (flattening lin nil))
74    (setf lout (nreverse lout))
75  )
76
77  ; flatten a list in and return the result into a list out
78  (defun flattening (lin lout)
79    (setq li nil)
80    (loop for x in lin
81          if (numberp x)
82          do (push x li)
83          if (listp x)
84          do (progn
85               (if (not (eql li nil))(progn(push li lout)(setf li nil)))
86               (setf lout (flattening x lout))
87               )
88          )
89    (if (not (eql li nil))(progn(push li lout)(setf li nil)))
90    lout
91  )
92
93  ;Converst a list2d to a rhythm-tree format
94  (defun list2d-to-rhythm-tree (l1)
95    (setq l nil)
96    (setq subl nil)
97    (loop for x in (cdr l1)
98          for y from 1 to (- (length l1) 1)
99          do (progn
100              (push x subl)
101              (if
102                  (eq (mod y 2) 0)
103                  (progn (setf subl (reverse subl))(push subl l)(setf subl
                         nil)
104                    )
105                )
106              )
107         )
108   (setf l (list (reverse l)))
109   (push (car (car l1)) l)
110
111 )
112
113 ; Takes a C-array of arrays and converts it into a lisp list of lists
114 ; The pointer is not freed here.
115 (defun c-array2D-to-lisp-list2D (ptr &optional (type :double))
116   (setq size (cffi::mem-aref (cffi::mem-aref ptr :pointer 0) type 0))
117   (loop for i from 1 to size collect (c-array-to-lisp-list (cffi::mem-aref
        ptr :pointer i)))
```

```
118 )
119
120 ; Takes a C-array of arrays and converts it into a lisp list of lists
121 ; The pointer is not freed here.
122 (defun c-array2D-to-lisp-list2DI (ptr &optional (type :int))
123   (setq size (cffi::mem-aref (cffi::mem-aref ptr :pointer 0) type 0))
124   (loop for i from 1 to size collect (c-array-to-lisp-listi (cffi::mem-aref
          ptr :pointer i)))
125 )
```

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**