# Practical Uses of Constraint Programming in Music using Relation Domains

Sascha Van Cauwelaert
Institute of
Information and Communication Technologies,
Electronics and Applied Mathematics,
Université Catholique de Louvain, Belgium
sascha.vancauwelaert [at] gmail.com

Gustavo Gutiérrez
Institute of
Information and Communication Technologies,
Electronics and Applied Mathematics,
Université Catholique de Louvain, Belgium
gustavo.ggutierrez [at] gmail.com

Peter Van Roy
Institute of
Information and Communication Technologies,
Electronics and Applied Mathematics,
Université Catholique de Louvain, Belgium
peter.vanroy [at] uclouvain.be

The use of Constraint Programming in Music is not new and is still very active today. This paper presents some advances and examples in the use of a completely new approach, constraint programming in music using relation domains. It defines a new high-level constraint, the filter constraint. This constraint is used in the modeling of a musical problem, the harmonized round problem. The paper presents the modeling of this problem and of some variants. In addition, another section describes a new concept, the constrained rewriting system. Its use is promising for some future practices in computer-aided composition. Rewriting systems have indeed been used in music for a long time and this new approach allows constraining those structures: production results and production rules can be constrained in order to ensure some desired properties. Some preliminary results are given regarding this new concept.

The general idea of constraint programming (CP) is to build a model for a problem to solve. Information is represented by means of decision variables, that is, variables with a set of possible values that they can take. Properties of the problem are expressed as constraints on variables of the model. Automated techniques are then used to find a value for every variable of the problem that satisfies all the constraints. This approach has been widely used to tackle problems in areas ranging from scheduling of activities to biology.

Constraint Programming is used for several decades in music. Several abstraction systems (Anders / Miranda 2011) have been built on top of CP and have led to several music compositions (Agon / Assayag / Bresson 2006; Bresson / Agon / Assayag 2008). A book describing CP in Music has also been recently published (Truchet / Assayag 2011).

Recently, a new approach of constraint programming in music has been presented (Van Cauwelaert / Gutiérrez / Van Roy 2012). This approach considers decision variables with relation domains, and related constraints. It allows representation and constraints of complex structures. For instance, it is possible to represent a whole score with only one decision variable. It is also possible to represent musical transformations and musical links with such variables. This means that they can be constrained too. The constraints can be very global about the music but also local.

In this paper, we present some advances in this field. We first provide the needed foundations to understand the presented content. Then, we explain a new high-level constraint: the filter constraint. After that, we describe how to solve a real musical problem: the harmonized round problem. Some variants are also provided. As this new approach allows tackling problems involving complex structures, we explain in the next section how to model *Constrained Rewriting Systems.* Rewriting systems have indeed been used for music for a long time, in particular Lindenmayer-Systems (Prusinkiewicz 1986; McCormack 1996; Worth / Stepney 2005). Our representation of rewriting systems is however more general than L-Systems. A substantial advantage is that it is now also possible to constrain those rewriting systems (including production rules, axioms and intermediate productions). In the last section, we provide some preliminary results in music regarding this new technique.

## Constraint programming in music with relations

In this section, we first give an introduction to Constraint Programming on Relations. We then explain briefly how it can be used to model musical problems.

### Constraint programming on relations

Constraint Programming on Relations (CPRel) is about using relation decision variables along with constraints that enforce properties on these variables. It also provides search abstractions and predefined generic heuristic strategies for solving models with this kind of variables.

A relation decision variable (relation variable for short) represents a relation out of a set of possible relations. A relation is a set of tuples of the same arity. A tuple is an element of the Cartesian product of the finite set of integers: $U=\{x:0\leq x\leq k\}$. The arity of a tuple is the number of elements that belong to it. For instance, tuples of arity *n* are elements of the set: $U\times\ldots\times U$. This set is also represented as $U^n$.

The domain of a relation variable *X*, denoted $D_X$, is the set of possible relations that *X* can be assigned to. A constraint *C* on a set of relation variables $\{X_1,\ldots,X_n\}$ will ensure that, in a given solution, the domain of every variable $X_i$ contains at least one relation that satisfies it. This is also called constraint inference. A variable is considered determined or assigned when its domain contains one and only one relation.

The constraints that can be used on relation variables come from two fields: Set Theory and Relational Algebra (Date / Darwen 1998). As a relation variable represents a relation it does make sense to express properties like: $X\cap Y=Z$ and $X=\overline{Y}$, where *X*, *Y* and *Z* are relation variables. Properties from the Relational Algebra include for instance:

$$Z = X \underset{C}{\bowtie} Z$$

which states that the join (in our case *eq-join*) of the two relations *X* and *Y* on the components (also called columns) in *C*, must be the relation *Z*.

The notion of tuple *concatenation* is used for the definitions of the constraints. Given two tuples $r=\langle r_1,\ldots,r_n\rangle$and $s=\langle s_1,\ldots,s_m\rangle$, we represent its concatenation by $r{+}{+}s=\langle r_1,\ldots r_n,s_1,\ldots,s_m\rangle$. The arity of the resulting tuple $|r{+}{+}s|=n+m$.

The notion of tuple *permutation* is also used. We say that a tuple *t* is a permutation of a tuple *s* under a permutation function *f*, denoted *permutation(t,s,f)* if $|t|=|s|$ and $\forall i\in\{1,\ldots,n\}:t_i=r_{f(i)}$.

**Projection.** This is a binary constraint on variables *X* and *Y* of different arities. It takes a constant *i* as an extra parameter, the number of components on the right of *X* that are projected. Its semantics is:

$$\prod_i X = Y \equiv \forall t_1,\ldots,t_{|X|}: \\ \langle t_1,\ldots,t_{|X|-(i-1)},\ldots,t_{|X|}\rangle \in X \\ \iff \langle t_{|X|-(i-1)},\ldots,t_{|X|}\rangle \in Y$$

**Join.** This is a ternary constraint on relations. It takes also a constant *i*, which represents the number of components on the right of *X* and on the left of *Y* that are considered by the constraint. Its semantics is:

$$X \underset{i}{\bowtie} Y = Z \equiv \\ \forall r,s:\exists u:|u|=i \\ \wedge((r{+}{+}u\in X\wedge\ u{+}{+}s\in Y) \\ \iff (r{+}{+}u){+}{+}s\in Z)$$

**Compose.** This is a ternary constraint on relations. It takes also a constant *i*, which represents the number of components on the right of *X* and on the left of *Y* that are considered by the constraint. It matches the semantics of relation composition from the Relational Algebra (Date / Darwen 1998). Its semantics is:

$$X \underset{i}{\smile} Y = Z \equiv \\ \forall r,s:\exists u:|u|=i \\ \wedge((r{+}{+}u\in X\wedge\ u{+}{+}s\in Y) \\ \iff r{+}{+}s\in Z)$$

**Confluent join.** This is a special case of *join* with an additional confluence condition. The additional condition ensures that it collects only those combined steps in *Z* for which all possible first steps in *X* can always find a next step in *Y* that continues to the same result. Its semantics is:

$$X \overset{\forall}{\underset{i}{\bowtie}} Y = Z \equiv \\ \forall r,s:\forall u:|u|=i\wedge \\ (((r{+}{+}u\in X \implies u{+}{+}s\in Y) \\ \wedge(\exists t:|t|=i\ \wedge r{+}{+}t\in X\wedge\ t{+}{+}s\in Y)) \\ \iff (r{+}{+}u){+}{+}s\in Z)$$

**Permutation.** The permutation constraint allows relating columns in relations of the same arity. Given two relation variables *X* and *Y*, and a bijective function $p:i\to j$ where *i* and *j* are columns in *X* and *Y* resp., the permutation is defined as:

$$Permutation(X,p,Y)\equiv \\ \forall t\in X,\exists r\in Y: \\ permutation(t,r,p) \\ \wedge\#(X)=\#(Y)$$

If *p* is the identity function then this constraint enforces equality between *X* and *Y*. With this constraint we can represent the rearrangement of columns in relations. This allows imposing that two relations are equal modulo some permutation of their components. Using it, we can apply all existing constraints (Van Cauwelaert / Gutiérrez / Van Roy 2012) on components of relations without regarding their position. The permutation constraint will only be explicitly used in this paper when it is necessary for understanding.

**Music representation with constraint programming on relations**

We use the two concepts of Musical Bundle (MB) and Musical Bundle Sets (MBS) (Van Cauwelaert / Gutiérrez / Van Roy 2012) as a way of representing musical concepts in terms of tuples and relations, which are the basic concepts of our new constraint system.

**Musical bundle** is *a set of pairs where each pair combines a musical parameter with an integer value.* A musical parameter can be here very abstract (they are not restricted to be traditional parameters). An example of an MB is a note defined by several musical parameters, such as the following tuple:

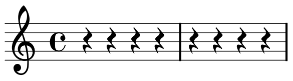
$$tuple = \langle pitch, onset, duration, ... \rangle$$

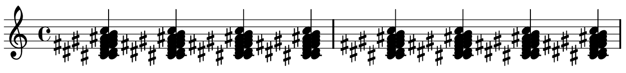
**Musical bundle set** is *a set of musical bundles.* Examples of MBSs are a chord, a bar, or a score. But an MBS can express more abstract musical concepts as well, for example a transformation between two scores can also be represented as an MBS. When modeling musical problems, we will use the terms MBS and relation variable interchangeably.

When we define a new musical problem in terms of relation variables, we need first to declare these variables. Providing the minimum and maximum MBS that the variable can assume does this. We then impose constraints on these relation variables. The constraint solver then uses a combination of propagation and heuristic search to find values of the relation variables that satisfy the constraints. For example, suppose that we are interested in finding a musical piece $M$ with some specific characteristics. The piece itself is represented by the relation variable $M$ with the following minimum and maximum MBSs:

$$M \in \left[ \{\ \} ... \{\{60, ..., 72\} \times \{0, ..., 7\}\} \right]$$

This notation says that the minimum MBS of $M$ is empty (i.e. 8 beats of silence) and the maximum MBS of $M$ is the musical piece that contains all the notes from middle C to C one octave higher, occurring on the first 8 beats (We assume here a signature 4/4 and quarter notes. For this example we only consider pitch and onset as the relevant parameters of the MBs.). The minimum and maximum MBSs of $M$ are represented by the following scores:





Additionally we want our score to respect some composition rules. Imposing constraints allows doing that.

**MBS as a transformation of MBSs**

An MBS can be used to define a transformation between two other MBSs. For instance, each note $\langle pitch, onset \rangle$ in the original MBS can be transformed into $\langle pitch', onset' \rangle$ in the new MBS. How the new values for the attributes are computed depends on an MBS that represents the transformation itself. As a particular example, let us consider a score *score* defined as a set of MBs of the form

$$\langle pitch, onset \rangle$$

, and a relation $T$ to represent the intended musical transformation. Every element of $T$ has the form:

$$\langle pitch_{start}, onset_{start}, pitch_{end}, onset_{end} \rangle$$

To obtain the transformed score (represented by the binary relation $score_{trans}$ resulting from the transformation of *score* by $T$ we impose the compose constraint

$$score_{trans} = score \underset{2}{\smile} T$$

In this way we ensure that any MB of *score* that can be transformed by $T$ has one (or several) respective(s) transformed MB in $score_{trans}$.

Notice that other kinds of transformation could be modelled in a similar way. The use of MBSs as transformations of MBSs will be used in this paper to model production rules of rewriting systems.

## The filter constraint

This section presents a new high-level constraint that will be used in the next section. A common use case is to filter or classify the information in a relation based on some criteria. For instance, for harmony it is useful to filter the pitches that are not consonant. Filtering can be easily expressed thanks to the confluent join constraint.

The filter constraint is defined by (we do not write required permutation here):

$$filter(R, F, R_f) \equiv \prod_m R \underset{n}{\overset{\forall}{\bowtie}} F = R_f$$

In this definition, *R* (arity *m*, with *m* larger or equal to *n*) is the relation to be filtered and $R_f$ (arity *m*, with *m* larger or equal to *n*) is the result of the filtering. The relation F (arity *n+1*) represents the filter. The tuples of *F* have the form

$$\langle param_1, \ldots, param_n, id \rangle,$$

where $param_1, \ldots, param_n$ are parameters defined in *R* and *id* is a group identifier. All the *n*-ary sub-tuples sharing the same group identifier belong to the same group.

Intuitively, the filter constraint let be in the resulting relation $R_f$ only the tuples of *R* such that if they share the same information on the left sub-tuple (*m-n* components on the left) then "pass **together** through the filter", that is, their right sub-tuple (*n* components on the right) belong to a same group in *F.*

A simple but powerful example of the use of the filter constraint is the following: assume the relation *S* representing a score, for which each tuple has the form

$$\langle onset, pitch \rangle$$

. Let $F_{harmony}$ be a binary relation for which each tuple has the form

$$\langle pitch, groupID \rangle$$

where pitches with same group id belong to the same group. Those groups can for instance represent groups of consonant pitches. Imposing the constraint *filter(S,* $F_{harmony}$*,S)* allows to ensure that at a given onset, only consonant pitches will be heard together.

In practice, this constraint can also be used in order to get some information about the harmony from a given score and to use this information to constrain a new score we want to create.

## The harmonized round problem

Musical problems can be solved using constraint programming. Let us consider the problem of finding a round, with a leader voice $V_0$ and one follower voice $V_1$. $V_1$ has then to be played *o* beats after $V_0$. Additionally, we want some given harmony rules to be respected between the voices. The value *o* is called the offset between the onsets of $V_1$ and $V_0$.

**Resolution of the round problem**

To model the problem, we use a relation variable *score* to represent the final score. *score* is a binary relation in which every element has the form

$$\langle pitch, onset \rangle$$

. A tuple of that relation represents a note. The same representation is used for the two voices (relation variables $v_0$ and $v_1$). A unary relation *Offset* represents the offset. Any element in this relation will represent the offset performed by a voice. So, for the particular case of one follower voice, *Offset* contains only one element. We also use a relation value named *Plus,* that contains all the tuples of the form

$$\langle X, Y, X+Y \rangle$$

(Notice that this relation value is a constraint *per se* for finite domain variables). For the harmony constraint let us assume an input binary relation *Consonant* with elements of the form

$$\langle pitch, index \rangle$$

. This relation links pitches with indices. The semantic behind this relation is that all the pitches linked with a given index are all consonant with each other.

The round property is imposed by:

$$v_0^{off} = v_0 \underset{0}{\bowtie} offset$$

$$v_1 = v_0^{off} \underset{2}{\smile} Plus$$

$$score = v_0 \cup v_1$$

The harmony constraint is enforced by:

$$filter(score, Consonant, score)$$

One important thing in this model is that it allows expressing directly that three pitches *a*, *b* and *c* are not consonant altogether while they are consonant pairwise. And it generalizes to *n* pitches without any effort: pitches are consonant altogether if they are linked by a common index in the relation *Consonant*. This ease comes from the use of the *filter* constraint.

**Extending the round problem**

In this subsection, we explain how we have extended the round problem with other constraints.

**A more realistic harmony constraint.** Considering the harmony only for a given onset is actually a limitation for the composition process. It is more general to think about the harmony for a given time interval. We will talk here about a time window inside which all played notes must be consonant with each other. The model described in the previous subsection is actually a particular case for which the window size is equal to *1*.

This idea can be very easily modeled using constraint programming on relations. With only 3 constraints, we gather together all the notes that are inside the same window. We obtain then the relation variable *score<sub>Window</sub>*. We then simply impose the same harmony constraint than in the previous subsection but we replace *score* by *score<sub>Window</sub>* in the expressions. The relation *Window* is here the unary relation

$$\{\langle 0\rangle,\dots,\langle n-1\rangle\}$$

where *n* is the size of the time window.

$$permutation(Plus, Plus_{perm}, \{\{0,2\},\{0,1\}\})$$

$$Plus_{window} = Plus_{perm} \underset{1}{\smile} Window$$

$$score_{Window} = score \underset{1}{\smile} Plus_{Window}$$

**Vuza canon constraint,** Musical structures studied by the mathematician D. T. Vuza are the canons of maximal category. Those are canons in which two different voices never play at a same onset but for each onset there is always a voice that is playing. This structure has already been used in a composition (Bloch 2006). Using constraint programming on relations allows imposing this constraint in a concise way, even if we are working with several musical parameters. However, we will here only express the first part of the constraint.

First of all, we create the final score in a slightly different way than in the subsection presenting the harmonized round problem. The score (here represented by the variable *voices*) is made of tuples of the form

$$\langle voice_{id}, pitch, onset\rangle$$

. This allows keeping the voice membership information for all notes.

$$v_0^{off} = v_0 \underset{0}{\bowtie} offset$$

$$voices_{gen} = v_0^{off} \underset{2}{\bowtie} Plus$$

$$voices_{fol} = \prod_{3} voices_{gen}$$

$$v_0^{withnum} = \{\langle 0\rangle\} \underset{0}{\bowtie} v_0$$

$$voices = v_0^{withnum} \cup voices_{fol}$$

After that, we create a relation that links voices that contains notes played at common onset. To impose the constraint that two voices never play at a same onset, we simply enforce the fact that this relation is included in the relation *eq*. This relation only contains tuples of the form

$$\langle i,i\rangle : i \in \aleph$$

.

The constraint is defined by:

$$permutation(voices, voices_{perm}, \{\{1,2\},\{0,1\}\})$$

$$voices_{perm}^{proj} = \prod_{2} voices_{perm}$$

$$permutation(voices_{perm}^{proj},\ voices_{reperm}^{proj}, \{\{0,1\}\})$$

$$vv = voices_{reperm}^{proj} \underset{1}{\smile} voices_{perm}^{proj}$$

$$vv \subseteq eq$$

**Optimization of the number of voices,** Eventually, using a cardinality constraint on the number of tuples inside a relation, we can solve an optimization problem from the constraint satisfaction problem presented in this subsection. For instance, we can decide to optimize the number of voices. This can be done in a straightforward manner: optimize the cardinality of the unary relation *off* which contains all the offset values.

## Constrained rewriting systems

This section presents the concept of *constrained rewriting system*. This concept emerged from the idea of representing Lindenmayer-Systems (L-Systems) using CPRel, as they have been rewriting systems used in the musical field for a long time. Other rewriting systems exist and our approach does not add any a priori restriction regarding that. As this idea comes from L-Systems, we first introduce their use in music. Then, we explain how rewriting systems can be modeled in a Constraint Satisfaction Problem.

### Lindenmayer-systems for music composition

L-systems have been considered for long time in music composition/generation (Prusinkiewicz 1986; McCormack 1996; Worth / Stepney 2005). In this section, we briefly explain what they are and how they can be used for musical purposes. L-systems have been introduced by Lindenmayer (Lindenmayer 1968) to model the growth process of living organisms. L-systems are formal grammars, similar to Chomsky grammars, but with a main difference: productions are made in parallel. In our case, we will only consider Deterministic Context-Free L-systems (DOL-system). A DOL-system is a set $\{V, \omega, P\}$ where:

- *V* is an alphabet, i.e., the set of symbols of the L-system. *V** is the set of all words over *V*. $V^+$ is the set of all non-empty words over *V*.
- $\omega$ is a starting axiom. It is an element of $V^+$.
- *P* is a set of production rules.

Depending on the interpretation we give to the elements of *V*, we can use an L-system for different purposes. Originally, they were used to describe the growth of plants. In this case, we will give them a musical interpretation. There exist actually several ways of doing that, depending on which meaning we give to the symbols of *V*. One can directly interpret the symbols as symbolic music information, such as pitches (McCormack 1996). Other works show the use of L-system symbols to modify a current musical state, using a *turtle interpretation* (Prusinkiewicz 1986; Worth / Stepney 2005). Other interpretations could be given, such as symbols used to represent sound or gestural information. The approach taken in this paper does not make any a priori restriction on that regard.

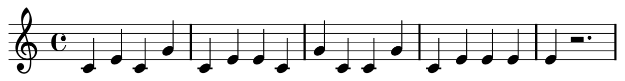
We will present here an example from McCormack (McCormack 1996) that uses symbols to represent pitches values. The definition of the (DO)L-system is the following :

- $V = \{C, E, G\}$
- $\omega = C$
- $p_1 : C \rightarrow E$
- $p_2 : E \rightarrow CGC$
- $p_3 : G \rightarrow \varepsilon$, where $\varepsilon$ is the empty word.

If we concatenate the different strings we obtain at each iteration, we obtain the following string for 5 iterations:

CECGCEECGCCGCEEEE

This string can be interpreted as a score if each symbol represents the pitch (or pitch-class) value of a note. Durations are not represented here so we consider all of them to be quarter notes. The string can then be interpreted as the following score:



As we have seen, an L-system (or more generally, a rewriting system) can be used to generate music. However, we may not know how to define it exactly (e.g., the production rules to be used), and may want it to own some properties (e.g., two productions from different iterations are exactly equal). In the following section, we will describe how it is possible to define a rewriting system inside a Constraint Satisfaction Problem. This will allow us to constrain the rewriting system itself.

**Rewriting systems defined in a constraint satisfaction problem**

Let **G** = $\{V, \omega, P\}$ be a rewriting system (where *V*, $\omega$ and P have similar semantics to those used in an L-System) and let the elements of *V* be $note_1, \ldots, note_{n*m*q}$. They correspond to MBs of the form

$$\langle pitch, onset, duration \rangle$$

where *n*, *m* and *q* are respectively the number of different pitches, onsets and durations we represent.

We can represent the ith iteration of **G** by the MBS $M_i$, that will contain MBs of the form

$$\langle pitch, onset, duration \rangle$$

(this MBS can be used to represent a score for instance). An iteration produces then here a set of symbols instead of a sequence as it is the case with L-Systems.

As said before we can use an MBS to transform MBSs. This allows modeling the elements of *P*. For instance, assume the following rule to be part of *P*:

$$note_i \rightarrow note_j$$

This rule can be modeled using an MBS $Prod_{note}$ that contains the MB (*p*, *o* and *d* stands here respectively for *pitch*, *onset* and duration):

$$\left\langle p_{note_i}, o_{note_i}, d_{note_i}, p_{note_j}, o_{note_j}, d_{note_j} \right\rangle$$

To apply the rule on $M_k$ (iteration result *k* of the rewriting process), the *compose* constraint is used:

$$M_{k+1} = M_k \underset{3}{\smile} Prod_{note}$$

This allows getting the MBS $M_{k+1}$ of iteration *k+1.* Other rules to be applied can be added just by adding MBs to $Prod_{note}$.

Assume now we want to use the set of rules:

$$\langle p_a, o_1, d_1 \rangle \rightarrow \langle p_b, o_1, d_1 \rangle$$
$$\vdots$$
$$\langle p_a, o_n, d_q \rangle \rightarrow \langle p_b, o_n, d_q \rangle$$

This set of rules only needs one MB to be represented:

$$\langle p_a, p_b \rangle$$

Those rules can be applied all in once using the constraint:

$$M_{k+1} = M_k \underset{1}{\smile} Prod_{pitch}$$

where $Prod_{pitch}$ is the MBS that contains this MB. Intuitively, this means that for any note with a pitch $p_a$ in iteration $M_k$, the same note but with a pitch $p_b$ will be in iteration $M_{k+1}$. This idea can also be used for the onset and duration parameters.

In all cases, we may have to use the *permutation* constraint to apply the constraint on the right relation components.

Notice that we could also consider more exotic production rules, such as if there exists a note with a given pitch, then a given note is produced. This involves the *projection* constraint.

We are now able to model a rewriting system as a part of a Constraint Satisfaction Problem. We call this a *Constrained Rewriting System* (CRS).

## General constrained rewriting system

A CRS is a Rewriting System modeled as a part of a constraint satisfaction problem. Because the elements defining the rewriting system are modeled by relation variables, we can *constrain* them. This means that we are able to constrain the axiom of the rewriting system, but also production rules and results of given iterations. The production rules do not have to be fixed at modeling time; we may not know which production rules we have to use in order to respect some desired properties about the produced music and/or the rewriting system. In a sense, we model a set of potential rewriting systems and we try to find out of this set those that respect our desired constraints.

Let us define the $RS_{step}$ constraint (the permutation constraints needed to be imposed are not written here for readability):

$$RS_{step}(R, Pr_{set}, Q) \equiv Q = \bigcup_i (R \underset{f(Pr_{set}(i))}{\smile} Pr_{set}(i))$$

where $R$ and $Q$ are two relations of the same arity, $Pr_{set}$ is a set of relations (possibly of different arities) and $f(A) = arity(A)/2$. Each relation in $Pr_{set}$ describes a set of production rules of the same kind (e.g., rules regarding pitch only, such as those presented in the previous subsection). Notice that the elements of $Pr_{set}$ must have even arities; we only consider rules such as described in the previous subsection. This constraint is defined in terms of existing constraints and allows describing exactly one iteration of a CRS.

Assume we want to obtain $R_j$, the result of the $j^{th}$ iteration of a CRS, for which $R_i$, the product of the $i^{th}$ iteration, has a property ensured by the constraint $C$ (some musical constraints expressed using CPRel are provided in a previous work (Van Cauwelaert / Gutiérrez / Van Roy 2012)).

Let $R_0$ be a ternary relation for which all tuples have the form

$$\langle pitch, onset, duration \rangle$$

. $R_0$ describes the axiom of the CRS. Let *Rules* be the set of relations that describes the production rules. The CRS can now easily be modeled:

$$\begin{gathered} RS_{step}(R_0, Rules, R_1) \\ RS_{step}(R_1, Rules, R_2) \\ \vdots \\ RS_{step}(R_i, Rules, R_{i+1}) \\ \vdots \\ RS_{step}(R_j, Rules, R_{j+1}) \\ \vdots \end{gathered}$$

Imposing additional constraints is now straightforward. For instance, the constraint $C$ just needs to be imposed on $R_i$. The effect on the CRS (and so on $R_j$) will be directly seen.

Moreover, some constraints involving elements of *Rules* can also be considered. The rewriting system can be adapted to our needs, thanks to the use of constraints. Going further, we can model the rewriting system without knowing exactly how it is, while forcing it to own some properties. A solution to the Constraint Satisfaction Problem provides also a rewriting system having these desired properties.

## Constrained rewriting system with musical motives

Consider now to work with musical structures, or motives, instead of going directly to the details of musical parameters. This can be done easily using CPRel. What is proposed here follows the approach of Anders to use L-systems to generate the motif structure and then use CP for local details (Anders 2007). However, because we model the rewriting system as a part of the constraint satisfaction problem, this approach is more general as the rewriting system may not be known at modeling time.

Let *Motives* be a relation used to represent all the musical motives we want. Every tuple of *Motives* has the form:

$$\langle motiveID, pitch, onset, duration, ..., param_n \rangle$$

*Motives* is a relation of arity *n+1* to be used to create a relation that represents an MBS (e.g., a score) with *n* parameters. The component *motiveID* is used to identify one motive in the set of motives. Notice that *Motives* can be variable.

To model the rewriting system working with motives, we will work with MBs represented by tuples of the form

$$\langle motiveID, offset \rangle$$

where *offset* is the offset in time according to the onset parameter. This provides the possibility to offset the used motives in time. An iteration result will then be a binary relation that contains elements with this semantic. The rewriting system is then defined in the same way as in the previous subsection.

There is an extra step in this approach: the generation of the final resulting MBSs (e.g., scores). Additional constraints can then be enforced on those MBSs and have a direct effect on the whole system. To show how to make this "translation", let us consider the $i^{th}$ result, represented by the relation $R_i$. In order to get the information relative to the different motives, we use the following constraint (permutation constraints will still be ignored in the following):

$$R_i^{Off} = R_i \underset{1}{\smile} Motives$$

We need to transform this relation by adding the offset value to the onset value, for each tuple. This is done directly by the following constraint:

$$Score_i = R_i^{Off} \underset{2}{\smile} Plus$$

where *Plus* is the ternary relation that contains all the tuples of the form

$$\langle X, Y, X+Y \rangle$$

(where the set of possible values for *X* and *Y* are bounded but large enough).

$Score_i$ represents now the score resulting from the $i^{th}$ iteration and can easily be constrained and interpreted.

Notice that it is also possible to offset other parameters than the onset, using a similar approach (Van Cauwelaert / Gutiérrez / Van Roy 2012).

## Preliminary results of constrained rewriting systems

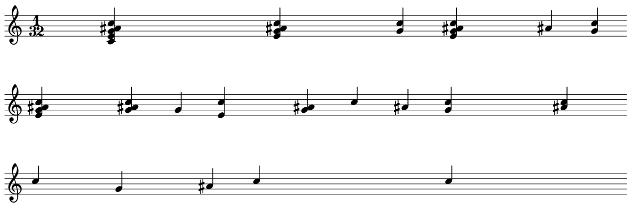
This section provides a few examples in the use of CRSs. The results are obtained using the relation domain implementation (Gutiérrez / Jaradin 2010), which is implemented on top of *Gecode* (Schulte / Lagerkvist / Tack 2006). *GeLiSo* (Van Cauwelaert / Gutiérrez 2011), an interface in development to integrate Gecode into *OpenMusic* (Agon / Assayag / Bresson 1998) is used in order to be able to interpret the results. The library "OMLily" (Haddad n.d.) has been used to generate the scores (in pdf format) from the OpenMusic musical objects. The values used in the presented models are MIDI values (e.g., $\langle 1, 60 \rangle$ represents middle C on onset *1*.). The rules presented are arbitrary and do not follow a special musical theory.

### Simple rewriting system

We only define here a rewriting system with the presented approach, without any additional constraints on it. The symbols of *V* are of the form $\langle onset, pitch \rangle$. The definition is (the notation is still borrowed to the definition of an L-System):

- $\omega : \{\langle 1, 60 \rangle\}$
- $Rules_{pitch} : \{\langle 60, 60 \rangle, \langle 60, 64 \rangle, \langle 60, 67 \rangle, \langle 64, 67 \rangle, \langle 64, 70 \rangle, \langle 67, 72 \rangle\}$
- $Rules_{note} : \{\langle i, 60+j, i+j, 60+j \rangle : i \in \{1,...,100\}, j \in \{0,...,24\}\}$

The result of the fifth iteration is the score:



### Rewriting system with motives

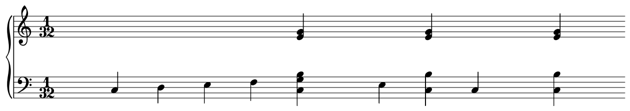
As presented before in this paper, we can model a rewriting system with motives as symbols. We show here one result regarding this. The tuples used to represent the notes are of the form

$$\langle pitch, duration, onset \rangle$$

(even if the results use indeed the duration parameter in OpenMusic, note durations are note explicitly represented in the shown results in this document). The three motives represented have been borrowed and translated from an example (Anders n.d.). The model is:

- $\omega : \{\langle 0,1\rangle\}$, i.e., motif 1 is played with an offset of 0 (i.e., it starts at the first onset)
- $$Patterns: \left\{\begin{array}{l}\langle 1,48,1,1\rangle,\langle 1,50,1,2\rangle,\langle 1,52,1,3\rangle,\\ \langle 1,53,1,4\rangle,\langle 2,48,3,1\rangle,\langle 2,59,3,1\rangle,\\ \langle 2,64,3,1\rangle,\langle 2,67,3,1\rangle,\langle 2,48,3,4\rangle,\\ \langle 2,59,3,4\rangle,\langle 2,64,3,4\rangle,\langle 2,67,3,4\rangle,\\ \langle 2,48,3,7\rangle,\langle 2,59,3,7\rangle,\langle 2,64,3,7\rangle,\\ \langle 2,67,3,7\rangle,\ \langle 3,55,2,1\rangle,\langle 3,52,2,3\rangle,\\ \langle 3,48,2,5\rangle\end{array}\right\}$$
- $Rules_{offset,motif} : \{\langle 0,\ 1,\ 4,\ 2\rangle\}$
- $Rules_{motif} : \{\langle 2,3\rangle,\langle 1,1\rangle,\langle 2,2\rangle,\langle 3,3\rangle\}$

The result of the second iteration is the score:



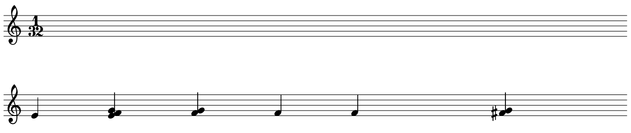
**Rewriting system with constraints**

As we showed in this paper, it is also possible to add other constraints to a rewriting system. We consider here the first rewriting system presented in this section, but we will not fix the axiom (its domain is only bounded). Additionally, we impose the constraint

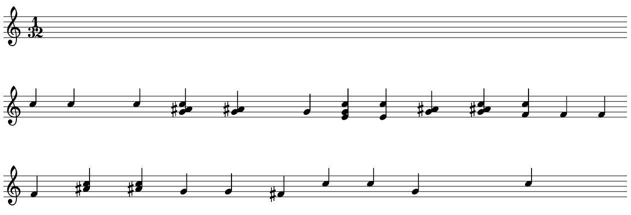
$$\{\langle 1,60\rangle\} \not\subset R_3$$

where $R_3$ is the result of the third iteration.

The found axiom is the score:

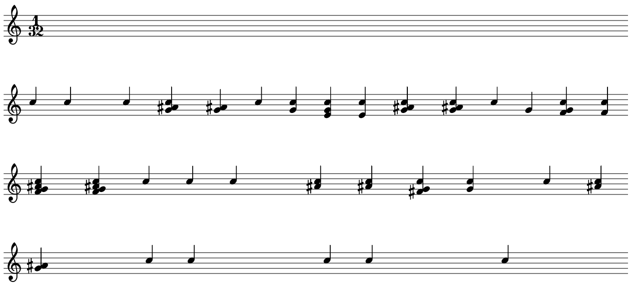


The found result for the third iteration is the score:



As we can see, the constraint is indeed respected.

The found result for the fifth iteration is the score:



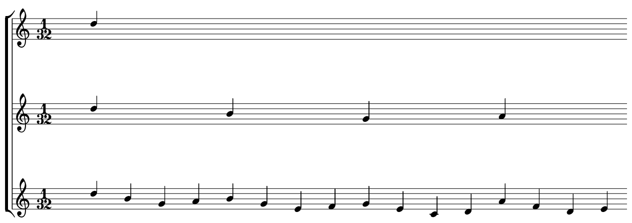
**Fractal music**

As a last example, we have considered an example presented by Henderson-Seller and Cooper in a work where they discuss about fractal music (Henderson-Sellers / Cooper 1993). As rewriting systems can produce fractal results, we model here one of the example provided in their work, using the presented approach. The symbols of *V* are of the form $\langle onset, pitch, duration\rangle$.

The definition is:

- $$\omega : \left\{\begin{array}{l}\langle 1,74,16\rangle,\langle 17,71,16\rangle,\\ \langle 33,67,16\rangle,\langle 49,69,16\rangle\end{array}\right\}$$
- $$\begin{array}{l}Rules_{note}:\\ \{\langle i,74,j,i,74,j/4\rangle :\\ i\in\{1,\ldots,100\}, j\in\{1,4,16\}\}\\ \cup\\ \{\langle i,74,j,i+j/4,71,j/4\rangle :\\ i\in\{1,\ldots,100\}, j\in\{1,4,16\}\}\\ \cup\\ \{\langle i,74,j,i+j/2,67,j/4\rangle :\\ i\in\{1,\ldots,100\}, j\in\{1,4,16\}\}\\ \cup\\ \{\langle i,74,j,i+3j/4,69,j/4\rangle :\\ i\in\{1,\ldots,100\}, j\in\{1,4,16\}\}\\ \cup\\ \vdots\end{array}$$

We have not written $Rules_{note}$ completely here, as it is a bit repetitive. If we play the axiom and the two first iterations together, we obtain the score (only the beginning of the score is represented here):



which is indeed equivalent to the one presented by Henderson-Seller and Cooper.

## Conclusion

This paper presents some advances in the use of Constraint Programming in Music using Relation Domains. We first define a high level constraint that allows representing the fact that a structure is the result of the filtering of a second structure by a third one. We also describe how to model the harmonized round problem and some of its variants. The concept of *Constrained Rewriting System* is then defined. It allows representing and constraining Rewriting Systems inside Constraint Satisfaction Problems. In our case, those rewriting systems are used to create music. We can then constrain the creation process by constraining the rewriting systems. In the last section, some preliminary results regarding this approach are provided.

In the future, we would like to extend the study of what is presented in this paper, in order to provide more concrete results and see to which extent it can be used in practice. For instance, we would like to consider big size problems that could not be solved using the usual musical constraint programming systems.

## References

Agon, C. / Assayag, G. / Bresson, J. (1998). Retrieved from OpenMusic: *http://repmus.ircam.fr/openmusic/home*

Agon, C. / Assayag, G. / Bresson, J. (2006). *The OM Composer's Book, Vol. 2*. Paris: Editions Delatour France / IRCAM Centre Pompidou.

Anders, T. (2007). *Composing Music by Composing Rules: Design and Usage of a Generic Music Constraint System.* Ph.D. Dissertation, Queen's University.

Anders, T. (n.d.). *Harmonised Lindenmayer-System*. Retrieved from Strasheela: *http://strasheela.sourceforge.net/strasheela/doc/Example-HarmonisedLindenmayerSystem.html*

Anders, T. / Miranda, E. (2011). Constraint programming systems for modeling music theories and composition. *ACM Computing Surveys (CSUR)* 43/4: 30:1-38.

Bloch, G. (2006). Vuza canons into the museum. *The OM composer's Book, Vol. 1*, edited by C. Agon, G. Assayag and J. Bresson. Paris: Editions Delatour France / Ircam-Centre Pompidou.

Bresson, J. / Agon, C. / Assayag, G. (2008). *The OM Composer's Book,* Vol. 2. Paris: Editions Delatour France / IRCAM Centre Pompidou.

Date, C. J. / Darwen, H. (1998). *Foundation for object/relational databases: The third manifesto : a detailed study of the impact of objects and type theory on the relational model of data including a comprehensive proposal for type inheritance.* Reading, Massachusetts: Addison-Wesley.

Gutiérrez, G. / Jaradin, Y. (2010). Retrieved from CPRelLib: A Relation Domain Constraint Library: *http://ggutierrez.github.com/cprelmaster*

Haddad, K. (n.d.). Retrieved from OMLily: *http://karim.haddad.free. fr/pages/downloads.html*

Henderson-Sellers, B. / Cooper, D. (1993). Has classical music a fractal nature? – A reanalysis. *Computers and the Humanities* 27: 277-284.

Lindenmayer, A. (1968). Mathematical models for cellular interactions in development. *Journal of theoretical biology* 18: 280-315.

McCormack, J. (1996). Grammar based music composition. *Complex Systems* 96: 321-326.

Prusinkiewicz, P. (1986). Score generation with L-systems. *International Computer Music Conference.*

Schulte, C. / Lagerkvist, M. / Tack, G. (2006). Retrieved from Gecode: Generic Constraint Development Environment: *http://www.gecode.org*

Truchet, Ch. / Assayag, G. (2011). *Constraint Programming in Music.* London: ISTE; Hoboken, N.J.: Wiley.

Van Cauwelaert, S. / Gutiérrez, G. (2011). Retrieved from GeLiSo: Gecode in common Lisp using Sockets: *https://github.com/svancauw/GeLiSo*.

Van Cauwelaert, S. / Gutiérrez, G. / Van Roy, P. (2012). A new approach for constraint programming in music using relation domains. *International Computer Music Conference.*

Worth, P. / Stepney, S. (2005). Growing music: musical interpretations of L-systems. *Applications of Evolutionary Computing*.

**[Abstract in Korean | 국문 요약]**

**관계 영역을 사용한 음악에서의 제약 프로그래밍의 실제적 사용**

**사샤 판 카우벨라어르트 / 구스타보 구티에레즈 / 페테르 판 로이**

제약 프로그래밍constraint programming은 오늘날에도 여전히 활발히 사용되고 있으며, 음악 분야에서 이를 사용하는 것은 새로운 일이 아니다.. 이 글은 관계 영역을 사용하는 음악에서의 제약 프로그래밍이라는 완전히 새로운 접근법의 이점과 사용 사례를 보여 준다. 이는 새로운 높은 수준의 제약, 즉 필터 제약을 정의한다. 이 제약은 음악적 문제 중 하나인 화음 매듭 문제harmonized round problem의 모형화에 사용된다. 이 글은 이 문제와 함께 이 문제에 관한 몇몇 다른 형태의 모형화에 대해 설명한다. 이어지는 부분에서는 제한 재기록 체계constrained rewriting system라는 새로운 개념에 대해 논한다. 이 체계의 사용은 컴퓨터의 도움을 받는 작곡에서 모종의 미래적 행위가 될 가능성이 있다. 실제로 재기록 시스템은 오래 동안 음악에서 사용되어 왔으며 이러한 새로운 접근 방식은 그러한 구조를 제한하는 것을 가능하게 하는바, 원하는 특성을 확보하기 위하여 생성물과 그 과정에 대한 조건을 제약할 수 있다. 이 개념에 대한 몇 가지 초기 결과들이 이 글에서 예시된다.