



HAL
open science

Advanced synchronization of audio or symbolic musical patterns

Florent Berthaut, David Janin, Benjamin Martin

► **To cite this version:**

Florent Berthaut, David Janin, Benjamin Martin. Advanced synchronization of audio or symbolic musical patterns. Sixth IEEE International Conference on Semantic Computing, Sep 2012, Palermo, Italy. pp.202-209, 10.1109/ICSC.2012.11 . hal-00698963v1

HAL Id: hal-00698963

<https://hal.science/hal-00698963v1>

Submitted on 18 May 2012 (v1), last revised 25 Feb 2013 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



LaBRI, CNRS UMR 5800
Laboratoire Bordelais de Recherche en Informatique

Rapport de recherche RR-1461-12

Advanced synchronization of audio or symbolic musical pattern

May 18, 2012

Florent Berthaut, David Janin, Benjamin Martin,
Université de Bordeaux, LaBRI UMR 5800,
351, cours de la libération,
F-33405 Talence
berthaut | janin | bmartin@labri.fr

Contents

1	Introduction	3
2	The advanced synchronization model	4
2.1	Basic pattern model	4
2.2	Towards an extended model	5
2.3	Realization and synchronization windows	6
2.4	Sequential composition	7
2.5	Resynchronization operator	9
3	Derived and additional operators	10
3.1	Context patterns and left and right shifts	10
3.2	Fork and join	11
3.3	Expansion/contraction operator	11
3.4	Parallel and extended parallel products	12
4	Application to audio patterns: advanced recomposition	13
4.1	Audio advanced synchronization	13
4.2	Application to audio recomposition	15
5	Application to musical performance: Advanced Live-Looping	16
5.1	Model of control live-looping	17
5.2	Recalage: an advanced control live-looper	18
6	Conclusion	20

Abstract

We define a generic model for finite audio or symbolic musical patterns that structurally encode a rich and abstract synchronization mechanism. This is achieved by distinguishing for each pattern a *realization window*, describing what the pattern is, from a *synchronization window*, describing how the pattern can be used. The sequential composition of patterns is defined and studied. An algebra of musical patterns is introduced in a mathematically well-founded approach. We propose several high level operators that can be used either in audio processing or in musical analysis and composition. Practical uses and experiments conducted in both fields are finally described.

1 Introduction

The last decades have seen the development of various software programs for Computer Assisted Music either used on stage for live performances or integral to multimedia applications for rich interactive audio supports. These softwares range from low level sound synthesis and control tools such as *Faust* [7] or *Max/MSP* [4], to high level composition assistants such as *Elody* [13] or *OpenMusic* [1] to name but a few.

In some rough dichotomy, low level (audio) softwares manage the *nature* of sounds: at any time quantum, they provide its *value*. On the opposite side, high level (music) softwares manage the *structure* of music: it is defined as some combination of notes, motives, movements, *etc.*, each with a specific duration, dynamics, usage, *etc.* As there is an increasing need of mixed usage of both software types, there is also an increasing need of structuring sounds in such a way that high-level compositions of musical patterns can be translated into low-level partial superpositions of audio patterns. This becomes especially crucial when interactive musical pieces are to be defined and performed [5, 9].

In this paper, we consider the problems of synchronizing musical patterns, either in the audio case or in the symbolic case. By discriminating between the definition of the pattern (what music is to be played) and its usage (when music is to be played), we define an advanced synchronization mechanism. It essentially consists in distinguishing, for every pattern, a *realization window* from a *synchronization window*. Sequentially combining two audio or symbolic patterns then amounts to sequentially combining their synchronization windows. As a result, realization windows may overlap. The handling of overlapping patterns, which is critical for a practical use of advanced synchronization, is defined according to the application.

In other words, we introduce a kind of musical pattern algebra, deeply well-founded from a mathematical point of view. Once presented this algebra (Sections 2 and 3), we illustrate its relevance for musical applications by means of two case studies: automatic audio recomposition (Section 4) and live looping performance (Section 5).

2 The advanced synchronization model

In this section, we define our generic model of musical patterns with advanced synchronization information.

2.1 Basic pattern model

In both signal processing or computational music, an audio or symbolic pattern can be abstracted as the mapping

$$S : [0, d] \rightarrow A \cup \{\perp\}$$

such that $[0, d] \subseteq \mathbb{R}$ is the time interval on which S is defined, A is the (finite) set of values this pattern may take at a given time and \perp corresponds to an undefined value.

For convenience, we represent S on a relative time scale, hence starting at date 0 and lasting $d \in \mathbb{R}$. When modeling digital audio patterns, A is the set of possible sample values, depending on the audio quantization used. When modeling symbolic music, A is the set of all sets of control events that may be played at the same time.

One may remark that in both audio and symbolic cases, the definition domain $dom(S)$ is, a priori, a finite subset of $[0, d]$ either defined by the sample rate of the audio pattern, or defined by the event dates of the control pattern. In other words, a pattern S is a partial function from $[0, d]$ to A . To make S total on $[0, d]$, we introduce the additional undefined value \perp that is assigned for every $t \in [0, d]$ where $S(t)$ is undefined.

This simple definition enables considering basic operators acting on such patterns. For instance, two musical patterns S_1 and S_2 with domains $dom(S_1) = [0, d_1]$ and $dom(S_2) = [0, d_2]$ can be combined one after the other, in a simple sequential fashion, into a pattern $S_1.S_2$ with domain $dom(S_1.S_2) = [0, d_1 + d_2]$ by taking, for all $t \in [0, d_1 + d_2]$,

$$(S_1.S_2)(t) = \begin{cases} S_1(t) & \text{when } 0 \leq t < d_1 \\ S_2(t - d_1) & \text{when } d_1 \leq t \leq d_1 + d_2 \end{cases}$$

where, for convenience, we assume that $S_1(d_1) = \perp$.

The simplicity of this basic composition operator makes it appealing for modeling purpose. It can be seen as a timed extension of the well-known concatenation product of strings, and its formal study already led to the rich theory of timed languages [2].

2.2 Towards an extended model

It may be argued that this concatenation product is not usable by itself for practical musical applications. In audio processing, for instance, signal continuity is crucial for combining patterns, in order to avoid unwanted artifacts; thus, overlapping patterns to produce smooth cross-fading transitions is highly desirable. In symbolic music modeling, time signatures and associated notions of weak and strong beats (or related notions in non-Western music) may induce constraints on the occurrence dates of musical events. Hence, the arbitrary sequential composition of musical patterns is very likely to lack of musical consistency.

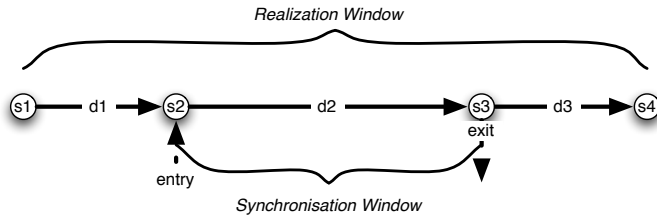
Therefore, in both audio and symbolic cases, the sequential composition of patterns generally requires additional parameters that accurately encode how such patterns should be combined one to the other. The basic pattern model introduced so far seems *incomplete*. It does indicate *what* musical patterns are but it does not indicate *how* such patterns can be used.

One may observe that, in music writing, this problem is solved by adding bars. Indeed, in musical scores, musical patterns are not only described as sequences of notes. Bars are added to describe how these musical sequences are to be played (or synchronized) one with the other. As an example, the notion of musical anacrusis refers to a few notes that are to be played before the first logical beat of a musical sequence.

Inspired by this synchronization mechanism encoded in music scores, we propose to enhance the aforementioned basic composition of musical patterns in order to allow our model to be practically meaningful. A former study of rhythm structures and rhythm compositions [11] suggests that one may distinguish in every rhythmic pattern a *realization window* where the pattern is defined from a *synchronization interval* that describes the usage of that pattern. This model, adapted to musical patterns, is presented here.

2.3 Realization and synchronization windows

In a first approach, the time structure of a musical pattern can be described by the following schema. The musical pattern starts at a given date s_1 and ends at a given date s_4 . Some subinterval described by two other dates s_2 and s_3 indicates, when involved in sequential composition, how this pattern must be combined with others.



Formally, abstracting from the date the pattern is actually fired, the *time structure* of a pattern S is modeled as a triple of durations

$$W(S) = (d_1, d_2, d_3) \in \mathbb{R} \times \mathbb{R}^+ \times \mathbb{R}$$

respectively describing durations of what can be called, following the presentation given in [11], the *introduction*, *development* and *conclusion* of pattern S .

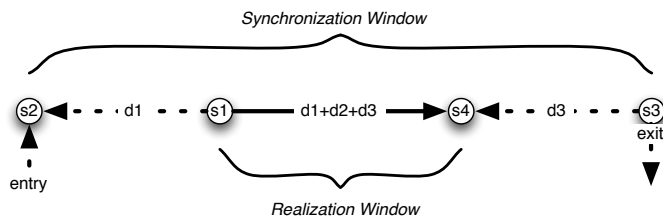
In this model, as soon as the above dates s_1 , s_2 , s_3 and s_4 are given, the *pattern durations* d_1, d_2, d_3 are defined such that:

- (1) $d_1 = s_2 - s_1$, with $[s_1, s_2[$ defining the *introduction* section,
- (2) $d_2 = s_3 - s_2$, with $[s_2, s_3]$ defining the *development* section,
- (3) $d_3 = s_4 - s_3$, with $]s_3, s_4]$ defining the *conclusion* section.

The triple comprising these three specific durations is called *time structure* of pattern S , and denoted by $W(S)$. $[s_1, s_4]$ is the pattern's *realization window*, and $[s_2, s_3]$ is the pattern's *synchronization window*.

The *effective duration* $d(S)$ of (the realization window of) a musical pattern S is defined by $d(S) = s_4 - s_1$ or, equivalently, $d(S) = d_1 + d_2 + d_3$. It must not be confused with the length of the synchronization window itself, the *synchronization duration* $s(S)$, defined by $s(S) = d_2$.

The model still makes sense when d_1 and/or d_3 are negative. For instance, when both d_1 and d_3 are negative, introduction and conclusion of S actually correspond to silent sections. In such a case, the induced dates of S are such that $s_2 \leq s_1 \leq s_4 \leq s_3$, and the resulting pattern only produces sound from s_1 to s_4 . The induced time structure can be illustrated by the following schema:



with the associated constraint that $d(S) = d_1 + d_2 + d_3 \geq 0$, i.e. the duration of the realization window remains positive. In that case, we have $s(S) \geq d(S)$.

In all cases, the beginning of the synchronization window, at date s_2 , is called *entry point* of pattern S , and the end of the synchronization window, at date s_3 , is called *exit point* of S . Moreover, from now on we define the pattern S with its associated time structure $W(S) = (d_1, d_2, d_3)$ on the domain

$$\text{dom}(S) = [-d_1, d_2 + d_3]$$

where the date 0 always stands for the date of the entry point. Such a definition turns out to be convenient for combining patterns.

Remark: When introducing the synchronization window of S , we assume that $s(S) = d_2 \geq 0$. This constraint ensures that the synchronization is defined in a past-to-future manner only. However, one might want to relax such a condition and allow negative synchronization windows. Such a consideration would be helpful from a mathematical point of view: it would result in manipulating a particular monoid structure, namely an inverse monoid [12] in which, by switching the entry and exit points of a given pattern S , one could associate the so-called pseudo-inverse of S in inverse monoid theory.

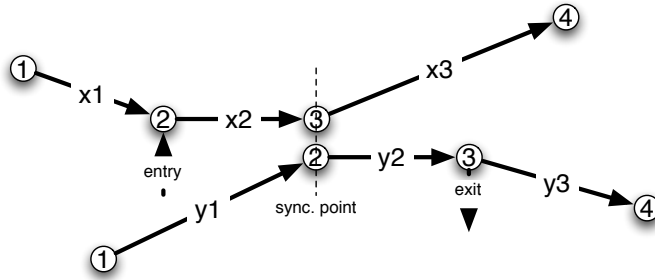
Nevertheless, negative synchronization windows do not appear to be relevant for application purpose. In all considered applications, synchronization mechanisms are practically performed from past to future. In the absence of negative synchronization windows, we still obtain concrete instances of monoid structures that have been recently investigated under the name of quasi-inverse monoids [10].

2.4 Sequential composition

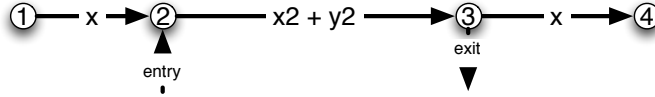
The purpose of distinguishing synchronization from realization is revealed when defining sequential compositions of musical patterns. This product is defined in two stages as a kind of concatenation of structured patterns. We first show how time

structures are combined. Then, from that combination, the effective composition of patterns, possibly with overlaps, is described in detail.

Let S_1 and S_2 be two musical patterns with respective time structures $W(S_1) = (x_1, x_2, x_3)$ and $W(S_2) = (y_1, y_2, y_3)$. The proposed sequential composition $S_1.S_2$ of patterns S_1 and S_2 consists in placing the synchronization window of S_2 right after the synchronization window of S_1 . As a result, the two musical patterns are positioned as described in the following schema:



The associated patterns are then assigned the resulting time structure:



with $x = \max(x_1, y_2 - x_2)$ and $y = \max(y_3, x_3 - y_2)$.

Formally, we define $W(S_1.S_2)$ by

$$W(S_1.S_2) = (\max(x_1, y_1 - x_2), x_2 + y_2, \max(y_3, x_3 - y_2))$$

In this composition, the resulting realization window of the product $S_1.S_2$ is defined as the union of realization windows of both S_1 and S_2 . It may be the case that one pattern is completely included into the other; thus, durations of introductory and concluding sections in the resulting time structure are computed by means of a \max .

Observe that the hereby defined product is *associative*, *i.e.* given three musical patterns S_1 , S_2 and S_3 , $(S_1.S_2).S_3 = S_1.(S_2.S_3)$. It should be clear that associativity is required in order to achieve a computationally robust formalism.

The value of the composition of patterns S_1 and S_2 with respective time structures (x_1, x_2, x_3) and (y_1, y_2, y_3) is defined as follows: for all $t \in \text{dom}(S_1.S_2)$,

$$(S_1.S_2)(t) = \begin{cases} S_1(t) \oplus S_2(t - x_2) & \text{when } t \in \text{dom}(S_1) \text{ and} \\ & t - x_2 \in \text{dom}(S_2), \\ S_1(t) & \text{when } t \in \text{dom}(S_1) \text{ and} \\ & t - x_2 \notin \text{dom}(S_2), \\ S_2(t - x_2) & \text{when } t \notin \text{dom}(S_1) \text{ and} \\ & t - x_2 \in \text{dom}(S_2), \\ \perp & \text{otherwise} \end{cases}$$

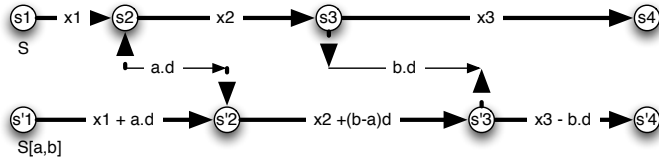
Note that the sum \oplus of pattern values depends on the type of data to be handled, and the precise semantics of that sum need to be adequately defined depending on the application.

2.5 Resynchronization operator

Given a pattern S with $W(S) = (x_1, x_2, x_3)$, given $d(S)$ and $s(S)$ the respective realization and synchronization durations of pattern S , and given two reals a and b such that $(a - b)d(S) \leq s(S)$, we define the *resynchronized* pattern $S[a, b]$ by $S[a, b](t) = S(t + a.d)$ for every $t \in \text{dom}(S[a, b])$.

Keeping 0 as the entry point of $S[a, b]$, we define $\text{dom}(S[a, b])$ from $\text{dom}(S)$ by a translation of $-a.d$. a and b are respectively called the *left offset* and the *right offset resynchronization ratios* of S . The underlying time structure is assigned as $W(S[a, b]) = (x_1 + a.d(S), x_2 + (b - a)d(S), x_3 - b.d(S))$. Doing so, we have $d(S[a, b]) = d(S)$.

This construction is illustrated by the following figure, where d stands for $d(S)$:



In other words, the resynchronization operator does not change the realization of the pattern it is applied to, it only modifies its synchronization offsets.

3 Derived and additional operators

In this section, we review additional properties of our model. Resynchronization and sequential composition, combined with a new expansion/contraction operator lead to the definition of many practically useful derived operators.

3.1 Context patterns and left and right shifts

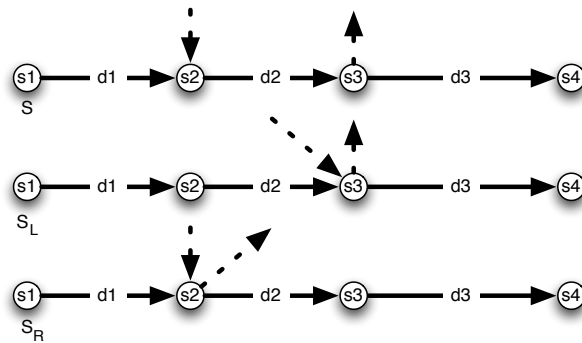
Patterns S such that $s(S) = 0$ are called *context patterns*. One can check that, provided that \oplus is commutative, then for every context patterns S and T we have $S.T = T.S$, i.e. sequential product on context patterns commute.

Observing that the resynchronization pattern $S[a, b]$ of pattern S is a context pattern when $a - b = s(S)/d(S)$, this leads us to define two special context patterns associated to S .

When $a = 0$ and $b = -s(S)/d(S)$, we call the context pattern $S[a, b]$ the *right shift* of S , and write $S_R = S[0, -s(S)/d(S)]$. Practically, S_R is obtained from S by shifting its exit point to its entry point, i.e. the sub-pattern in the sync window of S is *shifted to the right*;

When $a = s(S)/d(S)$ and $b = 0$, we call the context pattern $S[a, b]$ the *left shift* of S , and write $S_L = S[s(S)/d(S), 0]$. Practically, S_L is obtained from S by shifting its entry point to its exit point, i.e. the sub-pattern in the sync window of S is *shifted to the left*.

The synchronization structure of these new patterns is described in the following schema:



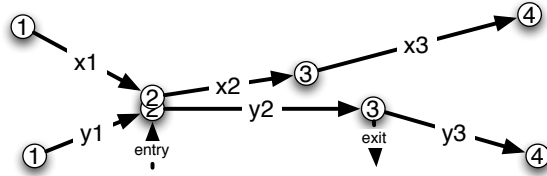
If we assume that the sum \oplus involved in the definition of the sequential composition is such that, for all pattern value v , $v \oplus v = v$, then the following properties are satisfied:

- (1) for all context patterns S and T , $S.S = S$ and $S.T = T.S$,
- (2) for all patterns S , $S.S_L = S_R.S = S$.

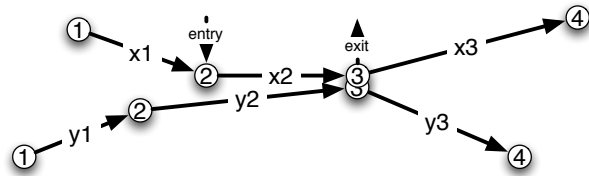
Many more properties are actually satisfied. In fact, the set of patterns equipped with such a sequential composition turns out to be a quasi-inverse monoid [10].

3.2 Fork and join

Let S and T be two patterns with respective time structures $W(S) = (x_1, x_2, x_3)$ and $W(T) = (y_1, y_2, y_3)$. The *fork composition* of S and T is defined as the sequential composition $S_R.T$. It consists in synchronizing S and T at their entry points, with the resulting synchronization window taken to be that of T . This construction can be depicted as follows:



Similarly, the *join composition* of S and T is defined as the sequential composition $S.T_L$. It consists in synchronizing S and T at their exit points, with the resulting synchronization windows taken to be that of S . This construction can be depicted as follows:



Observe that, in general, S and T do not have sync windows of same duration. Since $s(S_R.T) = s(T)$ and $s(S.T_L) = s(S)$, the join and fork operators defined here are not commutative in general.

3.3 Expansion/contraction operator

Given a pattern S and a positive real k such that $k \geq 1$ (respectively $k < 1$), we define the *expanded* pattern (resp. *contracted* pattern) kS as the pattern obtained from

S by realizing it k times faster (resp. slower). Formally, with $W(S) = (x_1, x_2, x_3)$, then $W(kS) = (k.x_1, k.x_2, k.x_3)$ and, for all $t \in \text{dom}(kS)$, $(kS)(t) = S(t/k)$.

Note that the practical use of this operator may require an additional application-dependent definition that describes how to modify the realization duration.

By combining resynchronization and expansion/contraction operators on a pattern S , we obtain an *extended resynchronization operator* (or *X-resync* for short) that modifies the realization of S but preserves the duration of its synchronization window. More precisely, for a pattern S with $W(S) = (x_1, x_2, x_3)$, all a and b such that $(a - b)d(S) < s(S)$ (for a non empty resulting sync window), the X-resync operator $S[[a, b]]$ is defined by

$$S[[a, b]] = \frac{s(S)}{s(S) - (a - b)d(S)} S[a, b]$$

By construction, $s(S) = s(S[[a, b]])$. This operator is particularly useful for musical performance applications, as described in Section 5.

3.4 Parallel and extended parallel products

When two patterns S and T have sync windows of equal size, i.e. when $s(S) = s(T)$, then not only both $S_R.T = T_R.S$ and $S.T_L = T.S_L$, i.e. fork and join arguments commute, but we also have $S_R.T = S.T_L$, i.e. fork and join coincide. This leads us to define, when $s(S) = s(T)$, the *parallel composition* of patterns S and T as:

$$S||T = S_R.T = S.T_L$$

Combined with expansion/contraction, this product can be generalized. Indeed, provided $s(T) \neq 0$ when $s(S) \neq 0$, we define the asymmetric parallel product $S[[T$ by

$$S[[T = S||kT$$

with $k = s(S)/s(T)$. In this case, T is expanded or contracted so that its sync window fits the size of the sync window of pattern S . In particular, $s(S[[T) = s(S)$.

By symmetry, provided $s(S) \neq 0$ when $s(T) \neq 0$, we define the asymmetric parallel product $S]]T$ by

$$S]]T = kS||T$$

with $k = s(T)/s(S)$. In this case, S is expanded or contracted so to fit the synchronization window of T . In particular, $s(S]]T) = s(T)$.

These extended parallel products are particularly useful for audio pattern synchronization and reconstruction, as explained in Section 4.

4 Application to audio patterns: advanced re-composition

Audio synchronization usually refers to syncing audio patterns to metadata, such as musical score, lyrics, *etc.*, or to possibly multi-modal syncing, with audio or video material for instance. In this section, we focus on the audio-to-audio syncing problem, referred to as *audio pattern synchronization*. This particular problem has been of major concern over the last decades either for signal processing or music information researchers. Indeed, any system related to the arrangement and organization of several audio patterns must cope with pattern synchronization problems. Applications range from alignment of music data [16] to automatic mixing of audio playlists [8], for instance.

We propose to investigate the use of advanced synchronization in the case of audio pattern handling. Although most of previous works on audio synchronization focus on the accurate syncing of two audio patterns, the formal approach introduced in this paper enables the synchronization of any number of patterns using the aforementioned advanced operators. Moreover, the distinction between synchronization and realization windows turns out to have major practical benefits.

4.1 Audio advanced synchronization

Synchronizing audio patterns brings several specific issues that need to be addressed in order to enable a practical use.

First, re-synchronization used with contraction/expansion operators may change the duration of audio patterns. In such cases, most audio applications require expanding or contracting sounds while keeping the original pitch content. Thus, contrastingly to symbolic applications (Section 5), a particular time-stretching function has to be defined. Classical examples of such functions include time-domain signal processing such as Pitch Synchronous Overlap and Add (PSOLA) [15] inspired methods, or frequency domain processing such as advanced phase vocoder techniques (see [6] and references therein).

Another specificity of audio patterns lies in their digital representation, that restrict their codomain to some quantized, finite subset. Let A be this codomain. Due to the digital audio representation, the sum of digital audio patterns must be defined as follows: $\oplus_A : A \times A \rightarrow A$. Moreover, since considering audio patterns, the sum of any number of signals must be continuous in order to avoid any audio artifact. An adequate summing function respecting these two criteria may be obtained

by multiplying patterns to *amplitude masks*, and computing the mean of resulting signals. Formally, for an audio pattern S_1 with time structure $W(S_1) = (x_1, x_2, x_3)$ and for all $t \in \text{dom}(S_1)$, we define the *masked pattern* $M(S_1) : \text{dom}(S_1) \rightarrow A$ with the same time structure as follows:

$$M(S_1)(t) = \begin{cases} \frac{t + x_1}{x_1} S_1(t) & \text{when } -x_1 \leq t < 0 \\ S_1(t) & \text{when } 0 \leq t \leq x_2 \\ \frac{x_2 + x_3 - t}{x_2 + x_3} S_1(t) & \text{when } x_2 < t \leq x_2 + x_3 \end{cases}$$

We also define the inverse mask pattern $\overline{M}(S_1) : \text{dom}(S_1) \rightarrow A$ as the dual of $M(S_1)$: $\overline{M}(S_1) = S_1 - M(S_1)$.

The sum of two audio patterns is then defined as the arithmetic mean of masked patterns, *i.e.* for all $t \in \text{dom}(S_1) \cap \text{dom}(S_2)$,

$$S_1(t) \oplus_A S_2(t) = \frac{1}{2}(M(S_1)(t) + M(S_2)(t))$$

More generally, the sum of $k \in \mathbb{N}$ overlapping audio patterns is defined, for all $t \in \text{dom}(S_1) \cap \text{dom}(S_2) \cap \dots \cap \text{dom}(S_k)$, as

$$\bigoplus_{i=1}^k S_i(t) = \frac{1}{k} \sum_{i=1}^k M(S_i)(t)$$

It is worth noting that with such a definition, which ensures that signal continuity is respected, sequentially composing audio patterns may no longer be associative. In sequences where at most two patterns overlap, associativity is respected. However, if three or more patterns overlap each other at some point in their combination, the order in which they are summed may change the resulting pattern. This non-associative property makes real-time audio handling challenging with the present model, in which case patterns can be expected to be combined in an iterative manner. In that case, another definition should be considered. For offline applications, on the other hand, this associative property can be relaxed: the overall synchronization structure is established before composing patterns, and the composition may then be realized in an arbitrary order.

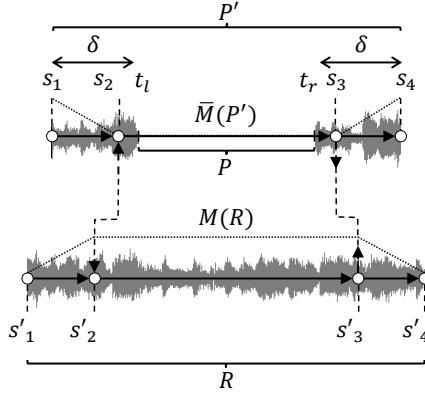


Figure 1: Synchronization structure for re-assigning a missing part \mathcal{P} with one pattern. Dashed lines correspond to synchronization points, while dotted lines show audio masks applied to the composition.

4.2 Application to audio recomposition

As a practical use of the synchronization theory, we propose to extend the automatic audio assignment method proposed in [14]. The solution proposed in that paper is to select one or several parts in an audio recording to reconstruct a missing section. The detection algorithm is based on string alignment methods in order to accurately detect musical repetitions. Although the algorithm is successful in identifying relevant parts, the accurate, possibly iterative, synchronization of reconstructed patterns was left as a perspective. This section is dedicated to using our synchronization model to explicit these advanced signal reconstruction operations.

Let S be an audio piece, and P its missing part. As explained in [14], the section that best fits P can be inferred by analysing local information around P . The pattern P' , composed as a development section containing P surrounded by an introduction and a conclusion, has to be synchronized with another pattern R within S that plays the role of replacement pattern. The durations of introductory and concluding parts of P' as well as the replacement pattern R are determined by the alignment algorithm [14]. With a properly defined synchronization between P' and R , one should be able to switch from P' to R in order to fill-in the missing part P .

Formally, let δ be the duration of local search contexts around P (see [14]). We denote by t_l the beginning time of P in S , and t_r the ending time of P in S . The beginning date of P' is denoted by $s_1 = t_l - \delta$. Similarly, the ending date of P' is denoted by $s_4 = t_r + \delta$. As explained above, the alignment algorithm identifies the

synchronization points between P' and R , denoted by $s_2 \in [s_1, t_l]$ and $s_3 \in [t_r, s_4]$ in P' and by s'_2 and s'_4 in R . Finally, the introduction and conclusion of pattern R are attributed the same durations as those of pattern P' , hence the realization times of R are defined as $s'_1 = s'_2 - (s_2 - s_1)$ and $s'_4 = s'_3 + (s_4 - s_3)$.

Figure 1 gives an overview of the synchronization structure involved. Synchronization and realization times are provided for both identified patterns P' and R . Masks are represented as dotted lines.

The overlapping introductions and conclusions in P' and R are intended to enable a seamless transition. Therefore, the parallel composition of these patterns must be realized between the inverse mask of P' (fading-out, then in), and the mask of R (fading-in, then out). Finally, the reconstructed pattern $r(P)$ is defined as follows:

$$r(P) = \overline{M}(P')[[M(R)$$

The synchronization operators defined in this paper enables us to bring the reconstruction model one step further. As suggested in [14], a major improvement of the reconstruction method would consist in combining a set of parts that locally fit the the missing data section P , especially when P is large. This problem can easily be addressed as a sequential composition of reconstruction sections. Let $k \in \mathbb{N}$ be the number of locally similar parts analysed by an alignment method, and denoted by $R_1, R_2 \dots R_k$. The reconstruction consists in sequentially combining the k distinct parts (adequately masked), and synchronizing the resulting pattern to P' . Figure 2 depicts the applied synchronization for $k = 3$.

The reconstructed pattern $r_k(P)$ is hence defined as follows:

$$r_k(P) = \overline{M}(P')[[M(R_1).M(R_2).\dots.M(R_k))$$

5 Application to musical performance: Advanced Live-Looping

Live-looping is a musical technique which consists in recording loops of data coming from a live musical input. These loops are then synchronized with a pulse, whose period often corresponds to the length of one of the loops.

When looped data is symbolic, consisting of notes for instance, we talk about *control live-looping*. Various live-looping interfaces exist, such as effects pedals, racks and software applications. New musical instruments such as Fijuu [17] or Drile [3] also rely on this technique and even improve it. For instance, in Drile, this technique

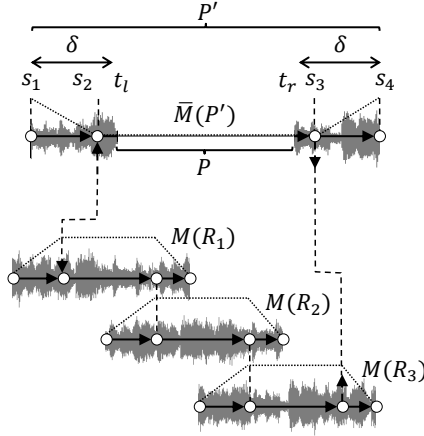


Figure 2: Synchronization structure for re-assigning a missing part \mathcal{P} with 3 patterns employed for reconstruction. Dashed lines correspond to synchronization points, while dotted lines show audio masks applied to the composition.

is expanded with the creation of live-looping trees, which provide new possibilities in terms of musical structures and loops manipulations.

5.1 Model of control live-looping

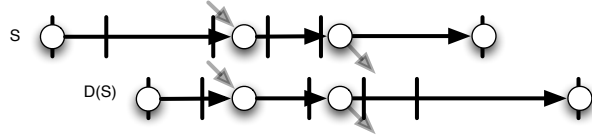
Live-looping can easily be conceptualized using the model and operators defined in previous sections. For this application, patterns are describing symbolic musical pieces. Each element of the codomain A of a pattern S thus describes a set of control events that have to be triggered simultaneously. The sum \oplus of two elements of A is then defined as the union of sets.

Modeling control live-looping amounts to defining, from each pattern S , the pattern $loop(S)$ that models the infinite sequential product of S with itself. In our formalism, this can be done as follows.

Given such a pattern S with non empty sync window, i.e. $s(S) > 0$, we first define the pattern $D(S)$ from pattern S by delaying S by the size $s(S)$ of its sync window. Formally:

$$D(S) = S[-s(S)/d(S), -s(S)/d(S)]$$

Figuring (sets of simultaneous) events by vertical bars, such a self-delay operator is illustrated as follows.

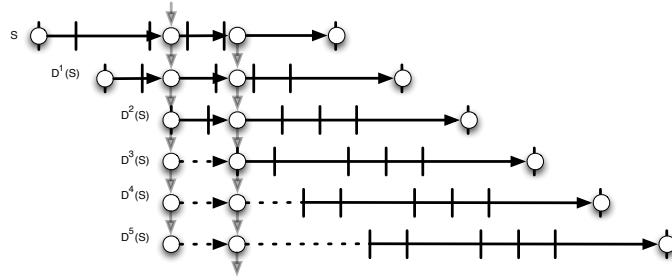


By iterating this construction, we define, for every k , $D^{k+1}(S) = D(D^k(S)) = S[-k.s(S)/d(S), -k.s(S)/d(S)]$. Since all delayed patterns have sync windows of same length, the parallel composition defined in Section 3.4 can be applied.

Formally, we then define the pattern $loop(S)$ resulting from a looping execution of pattern S as the infinite parallel composition:

$$loop(S) = S || D(S) || D^2(S) || \dots || D^k(S) || D^{k+1}(S) || \dots$$

This construction is illustrated in the following figure:

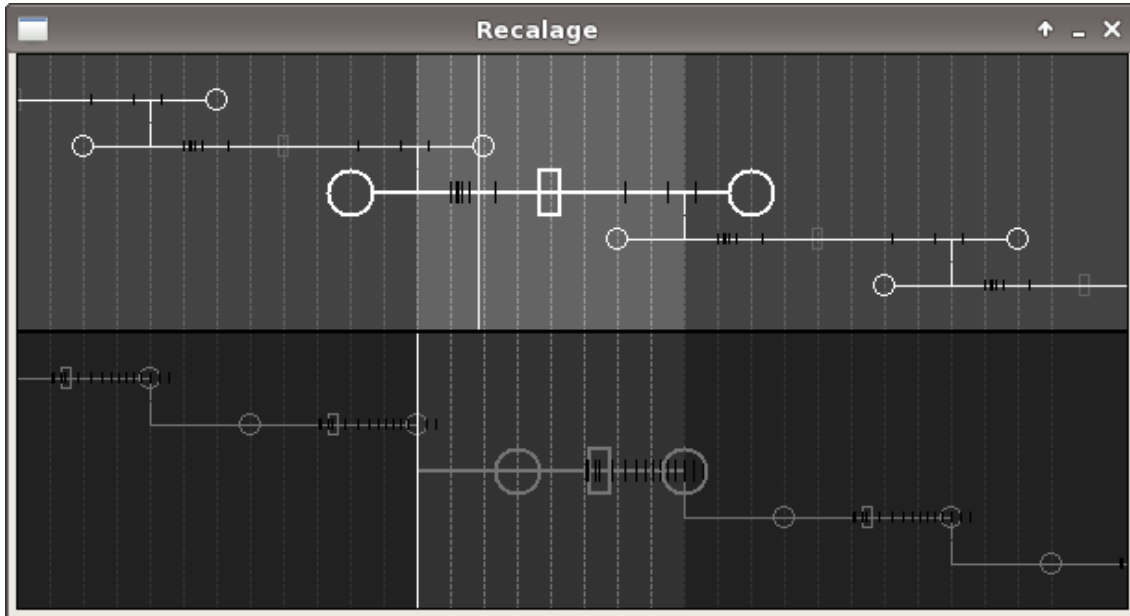


Note that in practice, since $d(S)$ is finite and $s(S) > 0$, $(loop(S))(t)$ is finitely defined since, at a given date t , only finitely many delayed copy of S are defined, i.e. $D^k(S)(t) \neq \perp$ for finitely many k .

5.2 Recalage: an advanced control live-looper

Combining the above loop operator with the extended resync operator defined in Section 3.3, we eventually define a versatile technique for musical performance which gives a way to generate new musical patterns from previously recorded ones. We implement these ideas in a instrument called *Recalage*.

Recalage allows one to record and play loops of midi events. When recording a loop, it is automatically synchronized with a multiple of the first previously recorded loop currently playing. *Recalage* provides controls for triggering loops, i.e. recording them if they are empty, toggling them otherwise, as well as for erasing them. *Recalage* is currently used with two loops playing at the same time, each represented by a specific widget, although stacking more loops only amounts to adding more loop widgets.



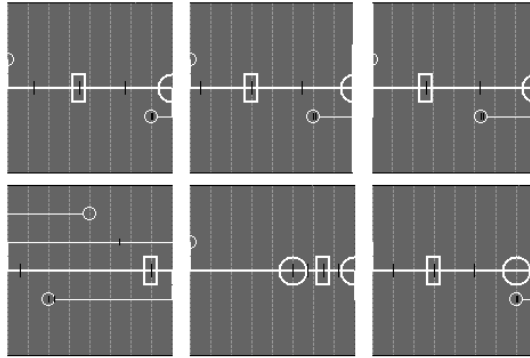
For each loop, the current instance of the pattern is drawn in the middle of the widget. The timeline is horizontal; thus, previous and next instances of a loop, which may overlap with the current one, are displayed respectively above and below the current instance. Beginning and end of the realization window are represented by white circles, and can be changed in real time. A white rectangle is added to the middle of the realization window, allowing users to simultaneously modify the begin and end offsets without changing the length of the realization window. For the current instance, the offset control elements (white circles and square) are larger and thicker and can be moved by clicking and dragging them. Recorded events are drawn as black lines. The beginning and end of the synchronization windows, for all instances, are represented by white vertical lines. The part with a lighter background emphasizes the fixed synchronization window of the current instance, played in loop. Finally, a vertical cursor provides a visual feedback on the playback of the loop.

In this particular example, one loop (top) is expanded and playing and one (bottom) is contracted and stopped. Observe that at any time, one only need to visualize which events occur in a single sync window.

All the available controls on the graphical user interface can also receive MIDI Control Change and MIDI Note messages, so that all the looping and resynchronization operations can be done with an external hardware controller.

Thanks to *Recalage*, one may notably create complex rhythmic structures out

of much simpler ones by applying graphically straightforward transformations. The figure below depicts typical musical examples created using *Recalage* from a simple 4 notes pattern.



These examples of musical sequences are created from manipulating a simple 4/4 pattern S made of four quarter notes. From left to right, they can be described as follows.

On the first line, the left expansion $S[[-1/8, 0]]$ creates a slight shift in the basic 4/4 rhythm that is typically found as anticipation pattern in electronic music. The left expansion $S[[-1/4, 0]]$ echoes a typical salsa bass rhythm : the *Tumbao*. The left expansion $S[[-1/3, 0]]$ creates a triplet pattern with the formerly first and fourth beats played together. On the second line, $S[[-3/4, 1/2]]$ sounds like a rather unbalanced pulse. The left contraction $S[[5/8, 0]]$ emphasize the expectation of the next strong beat as all played notes are pushed to the right. Finally, the left shift $S[[-1/8, -1/8]]$ moves the whole pattern to the left by half a beat.

For every derived looping pattern, the underlying pulse remains the same since the length of sync windows remains equal to the original. This allows the superposition of various patterns to potentially create many musical contrasts. For instance, the 4/4 pattern S played with the triplet pattern $S[[-1/3, 0]]$ creates a typical 3 on 4 poly-rhythm.

6 Conclusion

In this paper, we propose a rich algebra for manipulating and synchronizing musical patterns. From a generic model for representing musical events, we distinguish the notions of realization and synchronization of a pattern. We then define several

meaningful advanced synchronization operators and emphasize their practical relevance. A case study for a concrete audio application is presented, in which the model is employed to explicit the re-assignment of missing audio parts. The practical use of our model is also presented in a particular application of symbolic pattern manipulation, namely control live-looping. The introduced live-looper *Recalage* enables to obtain rich rhythmic structures out of re-synchronizing simpler ones, and to combine them in a versatile graphical interface. For both symbolic and audio applications, our model is successful in formally describing and expliciting complex synchronization problems.

References

- [1] Bresson J. Agon C. and Assayag G. *The OM composer's Book, Vol.1 & Vol.2*. Collection Musique/Sciences. Ircam/Delatour, 2006.
- [2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theor. Comput. Sci.*, 126(2):183–235, 1994.
- [3] Florent Berthaut, Myriam Desainte-Catherine, and Martin Hachet. Drile : an immersive environment for hierarchical live-looping. In *Proceedings of New Interfaces for Musical Expression (NIME10)*, pages 192–197, Sydney, Australia, 2010.
- [4] Alessandro Cipriani and Maurizio Giri. *Electronic Music and Sound Design - Theory and Practice with Max/Msp*. Contemponet, 2010.
- [5] Arshia Cont. Antescofo: Anticipatory synchronization and control of interactive parameters in computer music. In *International Computer Music Conference (ICMC)*, 2008.
- [6] A.J.S. Ferreira. An odd-dft based approach to time-scale expansion of audio signals. *Speech and Audio Processing, IEEE Transactions on*, 7(4):441–453, 1999.
- [7] D. Fober, Y. Orlarey, and S. Letz. Faust architectures design and osc support. In *14th Int. Conference on Digital Audio Effects (DAFx-11)*, pages 231–216. IRCAM, 2011.

- [8] H. Ishizaki, K. Hoashi, and Y. Takishima. Full-automatic dj mixing system with optimal tempo adjustment based on measurement function of user discomfort. In *Proc. of ISMIR*, 2009.
- [9] David Janin. A lazy real-time system architecture for interactive music. In *Actes des Journées d'Informatique Musicales (JIM 2012)*, pages 133–139, Mons, Belgique, 2012.
- [10] David Janin. Quasi-inverse monoids. Technical Report RR-1459-12, LaBRI, Université de Bordeaux, February 2012.
- [11] David Janin. Vers une modélisation combinatoire des structures rythmiques simples de la musique. *Revue francophone d'informatique musicale*, 2:to appear, 2012.
- [12] Mark V. Lawson. *Inverse Semigroups : The theory of partial symmetries*. World Scientific, 1998.
- [13] S. Letz, Y. Orlarey, and D. Fober. Real-time composition in Elody. In *Proceedings of the International Computer Music Conference*, pages 336–339. ICMA, 2000.
- [14] Benjamin Martin, Pierre Hanna, Vinh-Thong Ta, Myriam Desainte-Catherine, and Pascal Ferraro. Exemplar-based assignment of large missing audio parts using string matching on tonal features. In *Proc. of ISMIR*, pages 507–512, 2011.
- [15] E. Moulines and F. Charpentier. Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones. *Speech communication*, 9(5):453–467, 1990.
- [16] M. Müller, H. Mattes, and F. Kurth. An efficient multiscale approach to audio synchronization. *Proc. of ISMIR*, pages 192–197, 2006.
- [17] Julian Oliver and Steven Pickles. Fijuu2: a game-based audio-visual performance and composition engine. In *NIME '07: Proceedings of the 7th international conference on New interfaces for musical expression*, pages 430–430, New York, NY, USA, 2007. ACM.