The T-Calculus : towards a structured programing of (musical) time and space

David Janin, Florent Berthaud, Myriam Desainte-Catherine, Yann Orlarey, Sylvain Salvati

LaBRI, CNRS UMR 5800
Laboratoire Bordelais de Recherche en Informatique

Rapport de recherche RR-1466-13

The T-Calculus : towards a structured programing of (musical) time and space

February 16, 2013

David Janin[1], Florent Berthaut[1], Myriam DeSainte-Catherine[1], Yann Orlarey[2], Sylvain Salvati[1]

[1] Université de Bordeaux
LaBRI UMR 5800
351, cours de la libération
F-33405 Talence, FRANCE

[2] GRAME
Centre Nat. de Création Musicale
11, Cours de Verdun
F-69002 Lyon, FRANCE

Contents

The T-Calculus : towards a structured programing of (musical) time and space

David Janin[1]*, Florent Berthaut[1], Myriam DeSainte-Catherine[1], Yann Orlarey[2], Sylvain Salvati[1]

[1] Université de Bordeaux
LaBRI UMR 5800
351, cours de la libération
F-33405 Talence, FRANCE

[2] GRAME
Centre Nat. de Création Musicale
11, Cours de Verdun
F-69002 Lyon, FRANCE

February 16, 2013

Abstract

The T-calculus is a typed functional extension of the tiled signal algebra previously defined for handling, with advanced synchronization feature, audio or musical finite signals. Primarily designed for programing interactive music systems, the T-calculus can also be seen as a multi-purpose programing language proposal that addresses the difficult problem of globally asynchronous and locally synchronous (GALS) programing. Indeed, our proposal integrates both *synchronous programing* features for real-time audio or music signals processing and *asynchronous (or event based) programing* features for interactive positioning of these synthesized signals in time. Handling both spatial features (multi-channel programing) and timed features (signals positioning in time) the T-calculus can also be seen as a general purpose programing language proposal for a structured programing of time and space via a single new programing paradigm : *tiled programing*.

1 Introduction

Designing an interactive real-time music system in computational music is probably as difficult as designing a general control-command system for a plane, a car or a robot.

Indeed, one is immediately confronted to the inherent difficulties related with the modeling of the underlying musical theory one aims at preserving in the designed systems. Many known proposals have been made during the past

*partially funded by the Project CONTINT 2012 - ANR 12 CORD 009 02 - INEDIT

years to cope with it, either via automatic learning of real musicians idioms (see [18, 8] to name but a few), or by developing various notions of augmented scores that allow composers to described some computarized musical events at some dates in the score that will be triggered when these dates are reached (see e.g. [4, 1]).

In every case, beyond the modeling of the musical theory itself, another major source of difficulties is the necessity to describe both the synchronous programatic features induced by, say, low level signal processing, and the asynchronous programatic features induced by the positioning in time of these signals. This necessary multi-scale design, known in computer system design as the Globally Asynchronous Locally Synchronous (GALS) design style, is the source of many difficulties [19] in most application fields.

Such a gap between known classical programing paradigms and the intrinsic complexity of designing interactive music systems is especially well illustrated by the specialized programing languages available for computational music. In that field, most classical programing paradigms have been successfully lifted to music application oriented languages. This is true for functional style programing [17, 9] or for synchronous style programing [7] to mention but a few among many others. The resulting languages can be used to synthesize music scores or audio signals in a rather quite efficient and elegant way. However, extending these languages for them to cope with interactive event handling remains a difficult chalenge.

Paradoxically, the language of music - in a wide sense - that has been developed for centuries, integrates, at least empirically, both the synchronous handling of low level signals, beats or notes, and the asynchronous handling of higher level musical events. Thus one may expect that the formalization of some existing musical features can still be used to cope, at the programatic level, with such a difficulty, both for musical application but also in the various other fields.

In this paper, we aim at illustrating such a fact by showing that, as soon as signals are enriched with what can just be seen as music bars, one manages to define a high level versatile programing language: the T-calculus, that integrates both synchronous and asynchronous programatic features.

2 The T-calculus : an overview

The T-calculus is based on the notion of *tiled signals* [2]. Before giving a formal description of the T-calculus in the following sections, let us briefly review its basic features.

2.1 Overlapping tiles

The basic meta-objects that are handled in the T-calculus - the *tiled signal* calculus - are *overlapping tiles*, simply called *tiles* in the sequel. Though cumbersome, overlapping tiles can truly be seen as *roof tiles*.

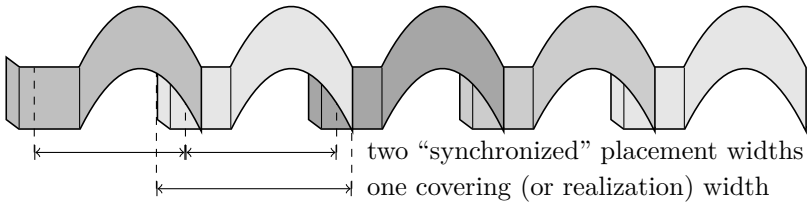

Figure 1: Roof tiles in a line

Indeed, in every tile line on a roof, one must distinguished for every tile the *covering width* of the tile, its real width, from the smaller *placement width* of the tile. Indeed, the *resulting covered width* by such a line is the sum of the placement widths of the placed tiles plus the remaining left and right covered widths of the left and right tiles.

The notion of *tiled signals* defined in the sequel essentially follows the roof tile intuition via the enrichment of the definition of signals based on such a tiling principle. Indeed, a tiled signal is just a signal with a given *realization interval*, the analogous of the covering width, that is enriched with a *synchronization interval*, the analogous of the placement width, that is included in the realization interval. This general intuition is illustrated in Figure 1

Saying so, one must pay attention to the fact that in both Mathematics or Computer Science, with some remarkable exceptions [15], the *classical usage* of the word tiles (or tilings) is generally to denote collections of objects that can be positioned one aside the other respecting *bounded* compatibility constraints.

Indeed, in most examples, compatibility constraints are typically defined by means of finitely many symbols (like in dominoes), or finitely many shapes (like in simple puzzles with repeated pieces). In these cases, the overlap that may occur is bounded and can thus be replaced by finitely many compatibility constraints. It follows that, in the underlying formal model of one dimensional tile, the synchronization interval and the realization interval coincide.

In the T-calculus, as in [15], tiles' overlaps are unbounded, distinguishing between realization and synchronization is thus needed. Despite such an increase in the complexity of the model, the combination of two tiled signals can still be seen as a single tile. It follows that handled objects can still be described inductively: this is an essential feature of classical data structure in programing languages.

2.2 Mixing signals vs concatenating tiled signals

Signal mixing, or more generally multi-signal processing, requires, to position in time, one relatively to the other, the signals to be processed before applying a given processing function.

Such an approach, perfectly valid, is a common practice for sound engineers in music studios. It can be abstractly described as an *external synchronization*

mechanism where the relative positioning of the signals to be processed depends on the combined analysis of these signals. This feature is depicted in Figure 2. Tiling signals arise when one wants to *internalize* such a synchronization spec-

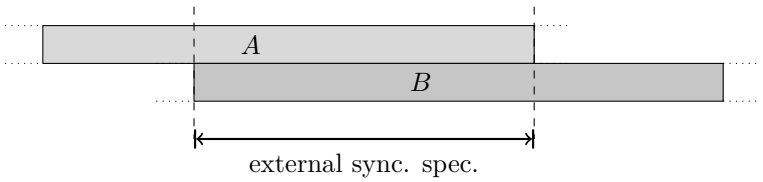

Figure 2: External specification of synchronization

ification by distinguishing for every involved signal, as in a tile, its realization interval (its cover) from its synchronization interval (its placement).

Doing so, computing the synchronization product $A; B$ of the resulting *tiled signals* just amounts to position the end of the synchronization interval of A at the same time than the beginning of the synchronization of B. Such a synchronization is illustrated in Figure 3. There, everything looks as if the original external synchronization specification has been *distributed* and *internalized* among the signals involved in the specification.

In other words, synchronization specifications are anticipated in every tiled signal by a *synchronization profile* that tells how the tiled signal will be placed with respect to the other, almost regardless of the realization interval of the underlying signal.

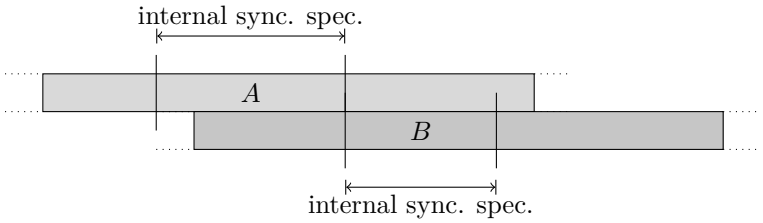

Figure 3: Internal specification of synchronization

2.3 Properties of tiled signals

Such an internalization of the synchronization specification into the synchronization profile of each tiled signals is actually one of the *key features* of our proposal.

From a semantical point of view, the synchronization product is a robust associative operator. Aiming at defining interactive signal handling, with signals

that are dynamically received, processed or synthesized, this is a much welcome property.

From a programing point of view, the synchronized product $A; B$ of two tiled signals A and B can be understood as an event-based *asynchronous* composition of the form *"event" A followed by "event" B*. The same synchronized product $A; B$ is also an unambiguous description of the *synchronous* composition of the form *"signal" A synchronized with "signal" B* with possible overlaps. In some sense, every tiled signal can thus be seen both as an *asynchronous event* or as a *synchronous signal*.

From an algebraic point of view, the synchronization profile of an (abstract) tiled signal can be seen as triple of integers $(s, l, r) \in \mathbb{Z}\times\mathbb{N}\times\mathbb{N}$ with both $0 \leq l+s$ and $0 \leq s+r$. When $s \geq 0$ this can be depicted as in Figure 4. The integer s is

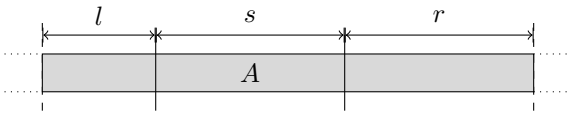

Figure 4: A tiled signal with profile (s, l, r)

the (relative) distance (in time units) from the beginning of the synchronization interval to its end, the integer l is the positive distance from the beginning of the realization interval to the beginning of the synchronization interval, and the integer r is the positive distance from the end of the synchronization interval to the end of the realization interval (integer r).

Then, the synchronization profile of the product of two abstract tiled signals with profiles (s, l, r) and (s', l', r') is directly defined as the product of their synchronization profiles

$$(s, l, r) \cdot (s', l', r') = (s + s', \max(l, l' - s), \max(r - s', r'))$$

It occurs that this product is associative with neutral element $1 = (0, 0, 0)$. The resulting algebraic structure, the sync profiles monoid, turns out to be well known in algebra. It is the free inverse monoid induced by a single generator [16]. As such, every element (s, l, r) has an inverse defined by $(s, l, r)^{-1} = (-s, l + s, s + r)$. Two special tiled signal transformations can also be derived: the left projection $L(A)$ and the right projection $R(A)$ of the tiled signal A as depicted in Figure 5.
Their resulting synchronization profiles are defined by $(s, l, r) \cdot (s, l, r)^{-1} = (0, l, s + r)$ for $L(A)$ and $(s, l, r)^{-1} \cdot (s, l, r) = (0, l + s, r)$ for $R(A)$.

In [2], it is shown that these specific operators considerably increase the expressive power of tiled signal expressions. Indeed, they allow to extend the default synchronization product by two other derived products, with either synchronization on the beginnings (see Figure 6) or on the ends (see Figure 7) of their synchronization intervals.

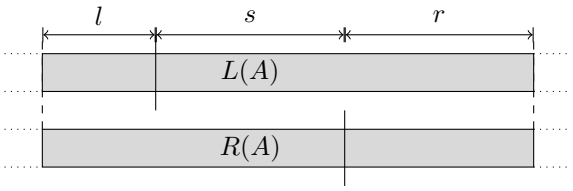

Figure 5: Left and right projection of a tiled signal with profile (l, s, r)

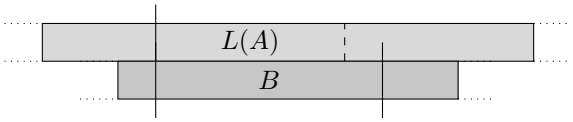

Figure 6: The Fork product $L(A) \cdot B$ of A and B

Extending the tiled signal algebra with additional typed operators that can be applied to the synchronized product of tiles, we show, in this paper, that this leads to the definition of rather subtle dynamic synchronization mechanisms.

2.4 Tiling signals : an old idea

As already mentioned, the notion of tiled signals goes back to early music writing. Indeed, this notion can just be seen as a generalization of the notion of bars in music.

When writing music, it is common practice for composers to write pieces of music with bars. Combining two such pieces of scores amounts then to synchronizing one bar in the first piece with one bar in the second piece. Some notes, especially before the first bar, i.e. appearing in a musical anacrusis, induce thus an overlaps between the two pieces of score.

In computer science, the notion of tiling signals is not new either. For instance, it already appeared more than twenty years ago in the music programing language LOCO [5] though in a rather *adhoc* formalism with PRE and POST synchronization indicators. Rediscovered recently in a study of traditional western music writing [11], a study oriented towards rhythm modeling for computer application, it was formalized in algebraic terms a little later [2] for being used either in static audio and symbolic music construction.

Though fairly simple, such an idea still seems both underexplored and underexploited in computer science. This paper is part of a more general research program that aims at studying and developing the potential great benefit of that idea.

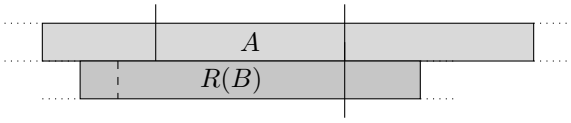

Figure 7: The Join product $A \cdot R(B)$ of A and B

2.5 A presentation based on program semantics

The purpose of this paper is to present and study the T-calculus : the typed functional extension of the tiled signal algebra [2]. Doing so, we obtain a fairly versatile programing language that conveys a new programing paradigm: *tiled programing*.

As there shall be no surprise in describing the syntax of our proposal (any typed functional programing language syntax can fit our needs) we essentially concentrate our presentation on the underlying objects and operations.

3 T-calculus semantics : typed tiles and related operators

In this section, we define the basic typed objects manipulated by the T-calculus and the related operators one can define on them.

3.1 Concrete and abstract tiled signals

A *concrete signal* is defined here as a mapping $S : \mathbb{Z} \to \mathcal{D}$ where $\mathbb{Z}$ is the set of absolute discrete time dates and $\mathcal{D}$ is a (presumably finite) set of signal values.

The set $\mathcal{D}$ is assumed to be equipped with an associative (and generally commutative) binary operation denoted by $+$ that admits a neutral element denoted by 0. It can also (and conveniently) be equipped with a product operator with element 0 acting as absorbant element. Additionally, any other heterogeneous operators can be used.

In a concrete signal, by opposition to an abstract signal defined below, $\mathbb{Z}$ is interpreted as an *absolute* time scale. A typical exemple of a concrete signal is the effective performance of a (framed) audio signal. In that case, every single frame value is sent to some audio processing devices at a specific absolute date.

A signal is said to be finite when there is an interval $[d_1, d_2[\subseteq \mathbb{Z}$ such that for all $t \notin [d_1, d_2[$ we have $S(t) = 0$. Such an interval is called the *realization interval* of signal S. Depending on the context of use, it can be automatically inferred from the signal values - as the shorter interval $[d_1, d_2[$ such that $S(t) = 0$ for every $t \notin [d_1, d_2[$ - or given as such.

In the sequel, we assume that every concrete signal S is associated to such a realization interval $r(S) = [s_1, d_2[$, be it implicitly or explicitly defined. The

duration of such a finite signal with realization interval is defined to be $d(S) = d_2 - d_1$.

A concrete signal $S : \mathbb{Z} \to \mathcal{D}$ with realization interval $r(S) = [d_1, d_2[$ is a *concrete tiled signal* when it is moreover equipped with a synchronization interval $s(S) = [d'_1, d'_2[\subseteq [d_1, d_2[= r(S)$. Since concrete tiled signals are already positioned in time, there may be no point to compute the synchronization product of two concrete tiled signals. However, a concrete tiled signal can be used in a product with an abstract tiled signal (see below) so that the abstract tiled signal is *concretized* in the product.

The *synchronization profile* of a concrete tiled signal S is defined to be the tuple $p(S) = (d, s, l, r) \in \mathbb{Z} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ where d is the concrete starting date of the synchronisation interval, and (s, l, r) is the related abstract synchronization profile defined in Section 2.3. The type $T(S)$ of a signal is defined to be (in relationship with) its domain $\mathcal{D}$.

In other words, when $p(S) = (d, s, l, r)$ then we have $r(S) = [d - l, d + s + r[$ and $s(S) = [d, d + s[$. Such an invariant relationship between synchronization profiles, realization interval and synchronization interval shall always be satisfied.

An *abstract tiled signal* is a collection of concrete signals equivalent under the *translation equivalence*. This equivalence is defined by $S \simeq S'$ when there exists $k \in \mathbb{R}$ such that $S'(t) = S(t - k)$ for every $t \in \mathbb{R}$ and if $p(S) = (d, s, l, r)$ then $p(S') = (d + k, s, l, r)$.

The synchronization profile $p([S])$ of the complete equivalence class $[S]$ induced by S is defined to be $p([S]) = (x, s, l, r) \in \mathcal{X} \times \mathbb{N} \times \mathbb{N} \times \mathbb{N}$ where the starting date x is now a variable symbol - taken from some set of variable names $\mathcal{X}$ - that may be evaluated (concretized) by any value in $\mathbb{Z}$. In general, abstract tiled signal may have additional constraints on x.

The distinction we are making from concrete (tiled) signal to abstract (tiled) signal makes a lot of sense in practical applications. Indeed, a (tiled) audio file can just be seen as a complete abstract (tiled) finite signal. Then, playing such a (tiled) audio file amounts to taking one of its concrete (tiled) representation: the one that starts when the file starts to be processed by the audio engine.

In the sequel, for simplicity, we drop the notation $[S]$ for describing the equivalence class that defines the abstract tiled signal associated to a concrete tiled signal S. This causes no ambiguity since the synchronization profile $p(S)$ of a tiled signal S tells us whether the tiled signal S is a concrete or and abstract tiled signal.

We extend possible profiles by allowing infinite left or right offsets so that even tiled signals built on infinite signals (such as constant below) can also be given a synchronization profile.

3.2 Typical examples of typed tiled signals

Audio and related tiles. The set of signal values is defined as a set $D \subseteq \mathbb{R}$ of sample's values equipped with (bounded) sum, (bounded) product and opposite.

As soon as a product is available, one may think about other types of tiled signals such as, for instance, amplitude signals with values in $[0, 1]$ used for defining dynamic mixing and cross-fading of audio signals.

Boolean tiles. These are signals with set of possible values $\mathcal{B} = \{0, 1\}$ with boolean disjunction as sum, boolean conjunction as product and 0 as neutral element for sum and absorbant element for product.

Boolean signals are especially interesting for modeling linked pairs of events of the form (On, Off) respectively occurring when the boolean signal shifts On or shifts Off.

Control states tiles. The set of values of control states signal is defined as a set of the form $\mathcal{P}(N)$ for some finite set of states N with union as sum and empty set as zero. Optionally, intersection can act as a product too.

For instance, when modeling interactive music, a state $n \in N$ can be used to model the fact that some keyboard key is being pressed. Following such a modeling intention, given a concrete signal $S : \mathbb{Z} \to \mathcal{P}(N)$, the value $S(t) \subseteq N$, defined at a given date $t \in \mathbb{Z}$, models the set of notes that are being pressed at date t.

Constant tiles. For every constant value $v \in \mathcal{D}$ the canonical abstract tiled signal associated to v, still denoted by v, is defined to be the constant signal v with profile $p(v) = (x, 0, \infty, \infty)$.

Another important class of constant boolean tiles is also defined from positive integers as follows. For every constant $n \in \mathbb{N}$, let $[n]$ be the abstract boolean tiled signal defined by (the equivalence class of) boolean signals that are true on intervals of the form $[x, x + n[$ and false everywhere else, with synchronization profile $p([n]) = (x, n, 0, 0)$.

3.3 Operators on tiled signals : general principle

We describe here a general principle that tells how to lift any operator defined on signal values to an operator defined on tiled signals.

Let $\mathcal{D}_1$, $\mathcal{D}_2$, $\mathcal{D}_3$ be three signal value domains. Let $op : \mathcal{D}_1 \times \mathcal{D}_2 \to \mathcal{D}_3$ be an operator defined between these domains. Operator op is lifted to a signal operator op that maps every signal $S_1 : \mathbb{Z} \to \mathcal{D}_1$ and $S_2 : \mathbb{Z} \to \mathcal{D}_2$ as a signal $S_1 \, op \, S_2 : \mathbb{Z} \to \mathcal{D}_3$ defined by $(S_1 \, op \, S_2)(t) = S_1(t) \, op \, S_2(t)$ for every $t \in \mathbb{R}$.

The operator op is lifted to a tiled signal operator as follows. Let $0_1 \in \mathcal{D}_1$, $0_2 \in \mathcal{D}_2$, $0_3 \in \mathcal{D}_3$ are the zeros of these domains. We say that operator op is a *max-operator* when, for every $x_1 \in \mathcal{D}_1$ and $x_2 \in \mathcal{D}_2$ we have $x_1 \, op \, x_2 = 0_3$ when $x_1 = 0_1$ *and* $x_2 = 0_2$. Similarly, we say that the operator op is a *min-operator* when, for every $x_1 \in \mathcal{D}_1$ and $x_2 \in \mathcal{D}_2$ we have $x_1 \, op \, x_2 = 0_3$ when $x_1 = 0_1$ *or* $x_2 = 0_2$.

Having said so, let S_1 and S_2 be two tiled signals with synchronization profiles $p(S_1) = (x_1, s_1, l_1, r_1)$ and $p(S_2) = (x_2, s_2, l_2, r_2)$. Assume that the *synchronization equation* $x_2 = x_1 + s_1$ has a solution, be it concrete (effective values for both x_1 and x_2) or abstract (inducing, in presence of variables, a solvable system of constraints). Then, the tiled product $S_1 \, op \, S_2$ is defined, for

the resulting signal, by $(S_1 \mathbin{op} S_2)(t) = S_1(t) \mathbin{op} S_2(t - s_1)$ for every $t \in \mathbb{Z}$, and, for the resulting synchronization profile, by

$$p(S_1 \mathbin{op} S_2) = (x_1, s_1 + s_2, \max(l_1, l_1 - s_2), \max(r_1 - s_1, r_2)$$

when *op* is a max-operator, and by

$$p(S_1 \mathbin{op} S_2) = (x_1, s_1 + s_2, \max(0, \min(l_1, l_1 - s_2)), \max(0, \min(r_1 - s_1, r_2)))$$

when *op* is a min-operator. Observe that the additional max operation added here is just to ensure that the left and right offsets remain positive. In both cases, the type $T(S_1 \mathbin{op} S_2) = \mathcal{D}_3$.

The partial operator *op* extended to tiled signals is completed, in the case the synchronization equation cannot be satisfied, by saying that the resulting tiled signal is the undefined tiled signal $\bot_{\mathcal{D}_3}$ of type $\mathcal{D}_3$.

Several operations are allowed on concrete signals. There is no much point to propose here an exhaustive list since these operations strongly depend on the sets of signal values that are used. However, one must keep in mind that, a priori, we only allow operations that preserve finiteness. We give below a brief list of such a kind of operators.

3.4 Operator examples

Synchronization product. We formally define here the synchronization product presented in Section 2.2. Given any pair of tiled signals S_1 and S_2 with domains $\mathcal{D}_1$ and $\mathcal{D}_2$ one can define the tiled signal $S_1; S_2$ with domain $\mathcal{D}_1 \times \mathcal{D}_2$ that is defined by extending to tiled signals the pairing function $p : \mathcal{D}_1 \to \mathcal{D}_2 \to \mathcal{D}_1 \times \mathcal{D}_2$ that maps any $x_1 \in \mathcal{D}_1$ and $x_2 \in \mathcal{D}_2$ to the pair $(x_1, x_2) \in \mathcal{D}_1 \times \mathcal{D}_2$. This is a first, fundamental, max-operator.

Signal sum. Assume $\langle \mathcal{D}, +, 0 \rangle$ is a monoid. In that case, the operator $+$ in domain $\mathcal{D}$ is a typical example of max-operator. The sum, as defined above, of every two tiled signals with value domain $\mathcal{D}$ is a first generic example. The resulting tiled signal operator is still associative. Its neutral element, denoted by 00, being defined as the constant signal 0 tiled with the synchronization profile $(x, 0, 0, 0)$.

Signal product. Assume $\langle \mathcal{D}, \cdot, 1 \rangle$ is a monoid with absorbant element 0. In that case, the product $\cdot$ in domain $\mathcal{D}$ is a typical example of min-operator. The product, as defined above, of every two tiled signals with value domain $\mathcal{D}$ is a second generic example. The resulting (partial) tiled signal operator is still associative with the constant tiled signal 1 as neutral element. In that case, the tiled signal 00 is an absorbant element.

Observe that for every tiled signal S, the product $S \cdot 0 = 0 \cdot S$ equals the constant signal 0 tiled by the synchronization profile $p(S)$ of S. It follows that, in particular, we have $00 = 0 \cdot [0] = [0] \cdot 0$.

Audio signal filter. As a particular case of product, we can define tiled audio signal filters as follows. Lifting the mixed product of a boolean by a frame value,

given a boolean tiled signal B and an audio tiled signal A, the On-Off filter of A by B can be defined by $B \cdot A$. More generally, assuming that B is a amplitude tiled signal with values in $\mathbb{R}^+$, we can also define the filter of A by B similarly.

State change detection. Given a control tiled signal C with value domain $\mathcal{P}(N)$ for some set of states N, given some state $n \in N$, we define, lifting the membership operator $\in$: $(N \times \mathcal{P}(N)) \to \mathcal{B}$ to a tiled signal operator, the state change detection signal $(n \in C)$ that is set to true (resp. false) whenever the state n is activated (resp. deactivated) in the set of states C.

Similarly, given an audio tiled signal S on some frame value domain $\mathcal{D}$, given some positive sample value $v \in \mathcal{D}$, another possible event detection is defined by $(|S| > v)$ that is set to true whenever the absolute value of S is above v.

4 T-calculus semantics : dynamical tiling

Every operator defined above can be used on either concrete or abstract tiled signals (although possibly generating the undefined tiled signal). The main question is then how to dynamically tile the handled signals.

4.1 Tiling input signals

An input signal S of any of the above type, is, by definition, a concrete signal (positioned in the absolute time scale) which value $S(t)$ is moreover unknown till the absolute date t is passed.

Tiling such a signal can be done dynamically by saying that both realization and synchronization intervals coincide and, moreover, that this interval is the least interval outside which the signal value is zero. In other words, the synchronization profile of an external tiled signal S is of the form $p(S) = (t, s, 0, 0)$ with $r(S) = [t, s + t[= s(S)$ with, depending on the absolute execution time, either none, or t, or $s + t$ (henceforth s) known.

Induced dynamic conditionals. This implicit tiling leads to the definition of rather subtle scheduling features.

Indeed, assume that B is such a input boolean tiled signal with synchronization profile $p(B) = (t, s_1, 0, 0))$. Let S be any other (abstract) signal with a profile of the form $p(S) = (x, s_2, l, r)$. We can define the tiled signal expression

$$\textbf{if } B \textbf{ play } S \equiv L(B).0 + S$$

where $L(B)$ is the left reset operator defined in Section 2.3 applied to B. This piece of a T-program specifies that S is dynamically "played" *when* B becomes true. The resulting synchronization profile is (t, s_2, l, r). The abstract synchronization interval is inherited from the tiled signal S.

Of course, since the arrival of B is unpredictable, this expression is incoherent w.r.t. the time flow whenever $l > 0$. In that case, it is set to $\perp_{\mathcal{D}}$ where $\mathcal{D}$ is the type of S. Indeed, there is no way to anticipate by l time units the beginning of the tile B. It appears that detecting incoherent tiled expressions is a key issue

for defining and proving the *validity* of T-calculus program. In general, with a T-calculus that allows recursive calls, this is probably an undecidable problem. Investigating that decidability is however left to further studies.

Another event-guarded tiled signal expression worth being identified is given by

$$\textbf{play}\, S\, \textbf{when}\, B \equiv L(S) + B.0$$

This still denotes the dynamical firing of the tile S *when* B becomes true. For the same reason, this expression is incoherent when $l > 0$. However, in that case, the resulting synchronization profile is now defined by (t, s_1, l, r). The abstract synchronization interval is now inherited from the tiled signal B.

4.2 Recursion mechanism: (least) fixpoint equations

The above construction suggests that, using tile variables of the form $X \in \mathcal{S}$ from some set of tile variables $\mathcal{S}$, one can define a recursion mechanism by means of (least) fixpoint equations of the form $X \stackrel{def}{=} F(X_1, \cdots, X_n)$. Classical constructions apply to define the semantics of such operations.

Induced dynamical conditional loops. With the same tiled signals as above, a typical example is defined by the fixpoint equation

$$\textbf{while}\, B\, \textbf{play}\, S \equiv \left(X \stackrel{def}{=} (L(B).0 + S) + X\right)$$

From an execution point of view, this amounts to playing S repeatedly, with the overlaps that can occur in the synchronized sums $S + S$, until the boolean tiled signal B eventually becomes false. When S is a fixed abstract signals, e.g. an audio file, the resulting synchronization profile is of the form $(t, k * s_2, l, r)$ where k is the number of iterations that are performed.

In other words, even though the number of iterations depends on the realization interval of B, the synchronization interval of the resulting tiled signal is a multiple of the synchronization interval of the tiled signal S. Such a programatic feature is especially relevant (and confortable for the musician) when programing dance music where strong beats play a key role.

Another typical example is defined by

$$\textbf{play}\, S\, \textbf{while}\, B \equiv L(X) + 0.B$$

with X defined as above. The computations of X or Y are essentially the same up to the fact that, for Y, the resulting synchronization interval is now the one inherited from B; a feature most welcome for any musician who wants to keep control on his positioning in time.

In both case, when B is a tiled input boolean signal, we require that $l = 0$ otherwise the resulting tiled signal is set to $\perp_{\mathcal{D}}$.

4.3 Memory management features

Last, a feature that is especially crucial in signal processing languages is the capacity one may have to record some concrete computed signals in a buffer for it to be reused *later* many times.

This recording capacity is handled via an additional operator (**save**) used as follows. For every concrete tiled signal S with profile $p(S) = (t, s, l, r)$, equation

$$X \stackrel{def}{=} (\mathbf{save})\, S$$

converts the concrete tiled signal S into the corresponding abstract tiled signal X that contains S and with a profile $p(X)$ *approximated* by (x, s, l, r) with the additional constraint that $x \geq t+l$. Indeed, at a date $t' < t+l$ the value of $S(t')$ can be unknown when S results from the evaluation of a T-calculus program triggered by an input tiled signal.

For instance, a program of the form

$$X \stackrel{def}{=} (\mathbf{save})[p] \cdot L(B) \cdot f \cdot X$$

encodes the echo audio effect defined with period p and decreasing factor f. Of course, an additional approximating mechanism is needed to make the induced computation effective.

4.4 Expressive power

The expressive power of (fragments of) the T-calculus appears when showing how to recover other programing languages. For instance, restricting to bi-infinite constant tiled signals with synchronization profiles of the form $(x, 0, \infty, \infty)$ yields to defined yet another classical *typed functional langage*. Restricting to tiled signals with profiles of the form $(x, s, 0, \infty)$ and no *save* operator, yields to the definition of the superset of a typical *synchronous programing language*. Restricting to finite states tiled signals with synchronization profiles of the form $(x, y, 0, 0)$ and no *save* operator seems to lead to an *event based communicating automata modeling language*.

Of course, the expressive power of the T-calculus deserves to be studied in details. However, the above examples already illustrate both the universality and the robustness of our proposal. As an immediate consequence of universality, many validation problems are undecidability and this is not a surprise.

5 Conclusion

We have provided an overview of the T-calculus that allows both synchronous signal processing and asynchronous event handling. A concrete instance of the T-calculus is currently under development. The low level synchronous tiled signal algebra is being developed as a library: the *libTuile*, that integrates both the (so far static) audio processing library *libaudiostream*[6] and (compiled) signal

processing functions definable in the synchronous programing language *faust*[7]. Development of some related graphical editor, analyser and compiler/simulator of T-calculus programs are scheduled.

One can observe that sets of tiles can themselves be seen as (more abstract) tiles though with a more complex synchronization profile. From the language theoretical point of view, such a shift from single tiles to sets (or languages) of tiles, have already demonstrated quite of its mathematical robustness and simplicity [13, 10, 14, 12]. These successes in theoretical computer science tell that the T-calculus can probably be extended to a higher order calculus that permits hierarchical description of system behaviors, from the most abstract (event based) layers to the most concrete (signal based) layer extending thus abstraction/refinement methods such as, for instance, *event B* [3].

References

[1] A. Allombert, M. Desainte-Catherine, and G. Assayag. Iscore: a system for writing interaction. In *Third International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA 2008)*, pages 360–367. ACM, 2008.

[2] F. Berthaut, D. Janin, and B. Martin. Advanced synchronization of audio or symbolic musical patterns: an algebraic approach. *International Journal of Semantic Computing*, 6(4):1–19, 2012.

[3] D. Cansell and D. Méry. Foundations of the B method. *Computers and Informatics*, 22, 2003.

[4] Arshia Cont. Antescofo: Anticipatory synchronization and control of interactive parameters in computer music. In *International Computer Music Conference (ICMC)*, 2008.

[5] P. Desain and H. Honing. Loco: a composition microworld in logo. *Computer Music Journal*, 12(3):30–42, 1988.

[6] S. Letz et al. The LibAudioStream library, 2012. `http://libaudiostream.sourceforge.net/`.

[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] M. Chemillier G. Assayag, G. Bloch. Omax-ofon. In *Sound and Music Computing (SMC) 2006*, 2006.

[9] C. Agon J. Bresson and G. Assayag. Visual lisp/clos programming in openmusic. *Higher-Order and Symbolic Computation*, 22(1), 3 2009.

[10] D. Janin. Quasi-recognizable vs MSO definable languages of one-dimensional overlaping tiles. In *Mathematical Foundations of computer Science (MFCS)*, volume 7464 of *LNCS*, pages 516–528, 2012.

[11] D. Janin. Vers une modélisation combinatoire des structures rythmiques simples de la musique. *Revue Francophone d'Informatique Musicale (RFIM)*, 2, 2012.

[12] D. Janin. Algebras, automata and logic for languages of labeled birooted trees. Technical Report RR-1467-13, LaBRI, Université de Bordeaux, 2013.

[13] D. Janin. On languages of one-dimensional overlapping tiles. In *International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM)*, volume 7741 of *LNCS*, pages 244–256, 2013.

[14] D. Janin. Overlaping tile automata. In *8th International Computer Science Symposium in Russia (CSR)*, LNCS (to appear). Springer-Verlag, 2013.

[15] J. Kellendonk and M. V. Lawson. Universal groups for point-sets and tilings. *Journal of Algebra*, 276:462–492, 2004.

[16] M. V. Lawson. *Inverse Semigroups : The theory of partial symmetries*. World Scientific, 1998.

[17] 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.

[18] F. Pachet. The continuator: Musical interaction with style. In ICMA, editor, *Proceedings of ICMC*, pages 211–218, Göteborg, Sweden, September 2002. ICMA. best paper award.

[19] P. Teehan, M. R. Greenstreet, and G. G. Lemieux. A survey and taxonomy of GALS design styles. *IEEE Design & Test of Computers*, 24(5):418–428, 2007.