8+8=4: Formalizing Time Units to Handle Symbolic Music Durations

Emmanouil Karystinaios[1], Francesco Foscarin[1], Florent Jacquemard[2], Masahiko Sakai[3], Satoshi Tojo[4], and Gerhard Widmer[1,5]

[1] Institute of Computational Perception, Johannes Kepler University Linz, Austria
[2] CNAM Paris, France
[3] Nagoya University, Japan
[4] Asia University, Japan
[5] LIT AI Lab, Linz Institute of Technology, Austria
emmanouil.karystinaios@jku.at
francesco.foscarin@jku.at

Abstract. This paper focuses on the nominal durations of musical events (notes and rests) in a symbolic musical score, and on how to conveniently handle these in computer applications. We propose the usage of a temporal unit that is directly related to the graphical symbols in musical scores, and pair this with a set of operations that cover typical computations in music applications. We formalise this time unit and the more commonly used approach in a single mathematical framework, as semirings, algebraic structures that enable an abstract description of algorithms / processing pipelines. We then discuss some practical use cases and highlight when our system can improve such pipelines by making them more efficient in terms of data type used and the number of computations.

Keywords: symbolic music; musical score; duration encoding.

1 Introduction

In a musical score, the duration of musical events (i.e., notes and rests) is defined by a finite set of symbols, and their temporal position by summing the duration of the previous musical events. Computer applications that deal with musical scores typically work with *Relative Symbolic Duration* (RSD) units, i.e., they choose a reference note duration and model all temporal information as ratios of that reference. For example, for the first four notes of the upper voice in Figure 1, one can choose a quarter note ♩ as a reference and represent the durations in the first two beats as the sequence $[0.5, 0.25, 0.25, 1]$. This kind of encoding shows its limits for certain durations, typically those produced by irregular groupings (also called tuplets). The 5th note in the top voice in the figure would have a duration of $2/3$, which is a periodic number not representable as a floating point value in computer applications, thus requiring a truncation. This introduces an

error that propagates to all subsequent musical events and creates a number of problems for applications that require exact matching of temporal positions.

Two main approaches have been proposed to solve this problem. The first is the *fraction approach*, implemented, for example, by the Python library Music21 [CA10]. It involves the representation of durations with specific Python objects made to mimic a fraction. This eliminates the rounding problems, but the fraction object is inefficient to handle with respect to native Python types and is not supported by libraries for heavy computations such as Numba, Pytorch, or TensorFlow. The second method, the *common divisor* approach, consists of setting the aforementioned reference duration to a value that is a common divisor of all durations appearing in a given piece or set of pieces. All temporal information can then be expressed with natural numbers, enabling very efficient computations. This solution is adopted by the Python library Partitura [Can+22], in some musical score storage formats such as MIDI, MEI, MusicXML, and in other computer music frameworks (e.g., [Fos+19]). However, this solution is still problematic for real-time scenarios when we do not know all duration in advance or when the piece can be modified. When a new duration is added that is not a multiple of the reference, the reference must be recomputed and all values updated.
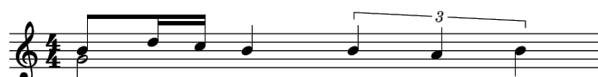

Fig. 1. A musical score example with two problematic configurations: a tuplet and an incomplete bottom voice.

Example 1. Let us consider a toy application on the score of Figure 1. We are interested in importing it from an MEI file, splitting the third note (the C) in the top voice into two, and producing a pianoroll representation. The notes in the top voice have durations, in RSD units (with a quarter note as reference), of $[\frac{1}{2}, \frac{1}{4}, \frac{1}{4}, 1, \frac{2}{3}, \frac{2}{3}, \frac{2}{3}]$. In the common division approach, we first need to compute as reference value δ a common divisor of all absolute note durations, the largest (i.e., the greatest common divisor GCD) if we want to optimize memory usage. In this case, this is $\frac{1}{12}$ of a quarter note. We then express each duration as a multiple of the reference, i.e., $[6, 3, 3, 12, 8, 8, 8]$. If we want to split the third note, we need to recalculate the value of δ as $1/24$, update the durations to $[12, 6, 6, 24, 16, 16, 16]$, and finally split the third note in two notes with duration 3. We can produce the pianorolls of the two voices independently and then perform an element-wise sum to obtain the score pianoroll. However, the second voice is logically incomplete in the score, missing an explicit half-note rest.[6] Thus, we first need to compute the maximum between the total duration of the two voices and insert the missing rests in the second voice. We compute the onset of each note by summing the durations of all previous notes in the same voice. ◊

---

[6] Ideally, both voices will have the same duration, but in real scores, this is often not the case; see [Fos+20; FRT21] for a discussion about score quality.

Music Symbol	𝅝	𝅗𝅥	𝅘𝅥	𝅘𝅥³	𝅘𝅥𝅮	𝅘𝅥𝅮³	𝅘𝅥𝅯
Relative Symbolic Duration (1 = 𝅘𝅥)	4	2	1	$0.\bar{6}$	0.5	$0.\bar{3}$	0.25
Absolute Symbolic Duration		1	2	4	6	8	12	16

Table 1. Examples of music symbols and corresponding durations in RSD and ASD units. Notes with "3" on top are notes that are part of a triplet.

This paper discusses an alternative approach to handling durations: the use of *Absolute Symbolic Duration* (ASD) units. The core idea is to consider the integers implied by the names of the graphical symbols. For example, a quarter note 𝅘𝅥 as 4, an eight note 𝅘𝅥𝅮 as 8, a 16th note 𝅘𝅥𝅯 as 16, and so on. Durations produced from irregular groupings are also expressed as integers (see Table 1). ASD units are already used by the Humdrum **kern file format, and (in a mixed representation with the divs approach) by MEI and MusicXML. However, they are only used to encode single note/rest durations. The typical pipeline procedure is to translate this duration format into relative symbolic durations, as a preprocessing step before any other operation.

On the contrary, we explore the usage of ASD, as "standalone" units to manipulate musical score durations. To make this practicable, we define two operations that cover typical use cases and we prove that, like RSD, ASD units form a *semiring*, an algebraic structure that enables a more abstract general description of processing pipelines. The actual computations can later be performed in ASD or RSD (or a mixture of the two), depending on the situation. This is enabled by an isomorphism that we provide to translate between the two units. Finally, we discuss some practical cases where one unit is to be preferred over the other to make the pipeline more efficient in terms of the number of operations and data types that are considered. We implement some algorithms that use ASD units in the Python library Partitura [Can+22].

2 Definitions

In this section, we first introduce the semiring; then we formally define *ASD* and *RSD* units and a morphism between them. Our goal with the introduction of this formalism is to give a general, abstract way of describing algorithms on music durations which is valid for both ASD and RSD units. Such algorithms can practically be performed in one unit or the other (or a mix of the two) depending on the specific application (see Section 3).

2.1 Semiring

Formally, a *semiring* $\mathcal{S} = \langle \mathbb{S}, \oplus, \otimes \rangle$ is an algebraic structure that consists of a domain $\mathbb{S} = dom(\mathcal{S})$, and two associative binary operators $\oplus$ and $\otimes$. Some properties must be verified: $\oplus$ is commutative, and $\otimes$ distributes over $\oplus$, i.e., $\forall x, y, z \in \mathbb{S}$, $x \otimes (y \oplus z) = (x \otimes y) \oplus (x \otimes z)$.

Note that there is no complete agreement in the literature over the exact definition of a semiring. Other research (e.g., [Pin98]) defines the two operations of a semiring

with a neutral element ($\mathbb{0}$ and $\mathbb{1}$ respectively), such that $\mathbb{0}$ is absorbing for $\otimes$: $\forall x \in \mathbb{S}$, $\mathbb{0} \otimes x = x \otimes \mathbb{0} = \mathbb{0}$. Then the semiring without neutral elements is called *hemiring*. However, similarly to [DSA10; SMS21], we just use the term semiring, without including neutral elements (and in particular the absorbing propriety of $\mathbb{0}$). The motivation is that verifying the absorbing propriety requires changes that would take our framework further away from its use for practically useful operations on music duration (more on this in Section 2.3). Components of any semiring $\mathcal{S}$ may be superscripted by $\mathcal{S}$ when needed. By abuse of notation, we write $x \in \mathcal{S}$ to denote $x \in \mathbb{S}$.

A semiring $\mathcal{S}$ is *commutative* if $\otimes$ is commutative. It is *idempotent* if for all $x \in \mathcal{S}$, $x \oplus x = x$. It is *monotonic* w.r.t. a partial ordering $\leq$ iff for all x, y, z, $x \leq y$ implies $x \oplus z \leq y \oplus z$, $x \otimes z \leq y \otimes z$ and $z \otimes x \leq z \otimes y$. Every idempotent semiring $\mathcal{S}$ induces a partial ordering $\leq_{\mathcal{S}}$ called the *natural ordering* of $\mathcal{S}$ and defined by: for all x and y, $x \leq_{\mathcal{S}} y$ iff $x \oplus y = x$. It holds then that $\mathcal{S}$ is monotonic w.r.t. $\leq_{\mathcal{S}}$. $\mathcal{S}$ is called *total* if it is idempotent and $\leq_{\mathcal{S}}$ is total, i.e., when for all x and y, either $x \oplus y = x$ or $x \oplus y = y$.

Given the particular algebraic properties above, semirings can be used as a weight domain for optimization problems such as the search for shortest paths in weighted graphs or hypergraphs [Moh02; Hua08]. Indeed, the theory of semirings and in particular the min-plus and max-plus Tropical Algebras [GP97] is commonly applied in decision theory and operational research, performance evaluation and control of dynamic systems, and also formal language theory, for quantitative extensions of formal computation models [DK09] (weighted automata and grammars). They have also been recently used for the formalization of musical elements, e.g., harmonic/melodic intervals by Albini [AB19], and to describe algorithms for musical tasks, e.g., music transcription by n-best parsing [Fos+19], and melodic distance computation [GJ22].

This work focuses on formalisations of musical duration that form idempotent and commutative semirings. Intuitively, in the applications presented in this paper, $\oplus$ selects the longest duration and $\otimes$ aggregates two durations in a single one.

2.2 Absolute Symbolic Durations

Let us define the semiring of Absolute Music Duration units $\mathcal{A} = \langle \mathbb{Q}^+ \cup \{\infty\}, \oplus, \otimes \rangle$ by detailing its domain and the two operations.

The domain

The domain $dom(\mathcal{A}) = \mathbb{Q}^+ \cup \{\infty\}$ of $\mathcal{A}$ contains (but is not limited to) non-null integers implied by the graphical symbol of notes and rests, e.g., quarter notes, eight notes, 16th notes, 32th notes, etc. Intuitively, larger values correspond to shorter notes. The limiting case is the null musical duration (used, for example, for grace notes), which is denoted by ∞. $dom(\mathcal{A})$ also includes other values that can result from the use of *duration modifiers* in the musical score, such as dots and tuplets, and will be described later in this section. We define $\prec^{\mathcal{A}}$ to be the strict order of absolute musical durations on the domain of $\mathcal{A}$. Elements of $\mathcal{A}$ are defined such that $\forall a, b \in dom(\mathcal{A}), a \prec^{\mathcal{A}} b \iff a > b$.

Operations

We are interested in two operations: a *selection* operation to find the longest duration,

and a *concatenation* to combine two or more musical durations. We define $\oplus^{\mathcal{A}}$ such that $a \oplus^{\mathcal{A}} b = \min(a, b)$, as the selection operation. Practically, this operation can be used to select the longest voice within a measure, when their durations do not correspond, like in Example 1.

The concatenation operation $\otimes^{\mathcal{A}}$ is defined as $a \otimes^{\mathcal{A}} b = \frac{ab}{a+b}$. This operation expresses mathematically the well-known musical rules about aggregating durations. For example, the concatenation of two eighth notes yields a quarter note, which in our framework can be written as $8 \otimes^{\mathcal{A}} 8 = 4$. A more advanced usage for ties and dots is also exemplified in Section 2.4. Readers who are not familiar with the semiring formalisms may find confusing that this concatenation operation, which looks very much like a sum, is denoted with the symbol $\otimes$, but this is what is commonly used and we keep it for consistency.

To prove that $\mathcal{A}$ is a semiring we need to prove that we have closure for both operations and that the multiplication distributes over additions. We go slightly further than proving closure and prove that both operations are commutative monoids (i.e. that they are commutative, associative, and there is an identity element) since this could be useful for further extension of our framework. Remember that for simplicity we write $x \in \mathcal{A}$ to denote $x \in dom(\mathcal{A})$.

Lemma 1. $\langle dom(\mathcal{A}), \oplus^{\mathcal{A}} \rangle$ *is a commutative monoid.*

Proof. Let $a, b, c \in \mathcal{A}$. By definition $a \oplus^{\mathcal{A}} b = \min(a, b)$.

Then from the commutativity and associativity properties of the min operation, $\oplus$ is also commutative and associative. The closure is trivial for min. The identity element is ∞, i.e., $\forall a \in \mathcal{A}, a \oplus \infty = a$.

Lemma 2. $\langle dom(\mathcal{A}), \otimes^{\mathcal{A}} \rangle$ *is a commutative monoid.*

Proof. Let $a, b, c \in \mathcal{A}$. By definition $a \otimes^{\mathcal{A}} b = \frac{ab}{a+b}$. By the commutativity of addition and multiplication, it follows that $\otimes^{\mathcal{A}}$ is also commutative and associative. Closure is also verified for the same reason. Let us investigate if the relationship also holds for the case of the null durations, i.e. ∞. We define $a \otimes^{\mathcal{A}} \infty$ as the limit $\lim_{b \to \infty} (a \otimes^{\mathcal{A}} b)$.

$$a \otimes^{\mathcal{A}} \infty = \lim_{b \to \infty} \left(a \otimes^{\mathcal{A}} b \right) = \lim_{b \to \infty} \left(\frac{ab}{a+b} \right) = a \lim_{b \to \infty} \left(\frac{b}{a+b} \right) = a \lim_{b \to \infty} \left(\frac{1}{\frac{a}{b}+1} \right) =$$
$$= a \left(\frac{\lim_{b \to \infty} 1}{\lim_{b \to \infty} \frac{a}{b} + 1} \right) = a \left(\frac{1}{\lim_{b \to \infty} \frac{a}{b} + 1} \right) = a \left(\frac{1}{0+1} \right) = a$$

Since $\otimes^{\mathcal{A}}$ is commutative, this also holds for the case $\infty \otimes^{\mathcal{A}} a$. We also proved that ∞ is the neutral element of $\otimes^{\mathcal{A}}$.

$\diamond$

We will now prove some Lemmas that will be useful for the proof of Theorem 1.

Lemma 3. *Let $a, b, c \in \mathcal{A}$, then $b < c \iff a \otimes^{\mathcal{A}} b < a \otimes^{\mathcal{A}} c$.*

Proof.

$$b < c \overset{a \geq 0}{\iff} ab < ac \overset{bc > 0}{\iff} ab + bc < ac + bc \iff b(a + c) < c(a + b) \overset{a \geq 0}{\iff}$$

$$ab(a + c) < ac(a + b) \overset{a+c>0,\ a+b>0}{\iff} \frac{ab}{a + b} < \frac{ac}{a + c} \equiv a \otimes^{\mathcal{A}} b < a \otimes^{\mathcal{A}} c$$

$\diamond$

Lemma 4. $\otimes^{\mathcal{A}}$ *is left and right distributive over* $\oplus^{\mathcal{A}}$.

Proof. Let $a, b, c \in \mathcal{A}$: By induction on the order relation between b, c:

- $b = c$:
  Trivial.
- $b < c$:

$$a \otimes (b \oplus c) = \frac{a \min(b, c)}{a + \min(b, c)} \overset{\text{by IH}}{=} \frac{ab}{a + b} = a \otimes b \qquad (1)$$

$$(a \otimes b) \oplus (a \otimes c) = \min((a \otimes b),\ (a \otimes c)) \overset{\text{by Lemma 3 and IH}}{=\!=\!=\!=} a \otimes b \qquad (2)$$

  Then our proof is completed by using reflexivity on 1 and 2.

- $b > c$:
  Similar to $b < c$.

The right side distributivity follows by the commutativity properties of the operations.
$\diamond$

Theorem 1. $(dom(\mathcal{A}), \oplus^{\mathcal{A}}, \otimes^{\mathcal{A}})$ *is a semiring.*

Proof. We have all the elements to conclude the proof:

- $(dom(\mathcal{A}), \oplus^{\mathcal{A}})$ is associative and satisfies the closure property (by Lemma 1);
- $(dom(\mathcal{A}), \otimes^{\mathcal{A}})$ is associative and satisfies the closure property (by Lemma 2);
- Multiplication distributes over addition (by Lemma 4)

$\diamond$

When dealing with multiple equal durations in music, it is practical to extend the $\oplus$ operation to define a scalar multiplication. For a duration $a \in \mathcal{A}$ and a scalar $n \in \mathbb{Q}$, it is denoted by the function $\text{repeat}^{\mathcal{A}}(a, n) = a/n$.

2.3 Relative Symbolic Durations

We define the semiring of Relative Symbolic Duration units $\mathcal{R}^{\delta} = (\mathbb{Q}^{+} \cup \{0\}, \oplus, \otimes)$ relative to the reference duration δ.

The domain The domain $dom(\mathcal{R}^\delta) = \mathbb{Q}^+ \cup \{0\}$ of $\mathcal{R}$ contains durations measured relative to a reference duration value. Intuitively, smaller values correspond to shorter notes. The limiting case is the duration 0, which can be used, for example, for grace notes. We define $\prec^{\mathcal{R}}$ to be the strict order of absolute musical durations on the domain of $\mathcal{A}$. Elements of $\mathcal{R}$ are defined such that $\forall a, b \in dom(\mathcal{R}), a \prec^{\mathcal{R}} b \iff a < b$.

Operations Similarly to $\mathcal{A}$, $\oplus^{\mathcal{R}} \equiv \max$ is used to select the larger duration and $\otimes^{\mathcal{R}} \equiv +$ is used to add two durations together. The repeat operation can be defined as $\text{repeat}^{\mathcal{R}}(a, n) = a * n$.

   We skip the proof of $\mathcal{R}$ being a semiring for brevity. It can also be noted that the operations and domain we defined are equivalent to those of a tropical semiring [Pin98], so the proof for tropical semirings is also valid for our case. Differently from a tropical semiring, however, we don't have the absorption property of the $\otimes$ neutral elements $\mathbb{0}$, i.e., $\forall x \in \mathcal{R}, \mathbb{0} \otimes x = x \otimes \mathbb{0} = \mathbb{0}$. In order to verify this, we would need to swap the min with the max (and vice-versa) for the $\otimes$ in our two semirings, but this would make for a non-musically useful operation, violating the ultimate objective of this research.

2.4 A General Duration Framework

Table 2 summarises our formalization of ASD and RSD units. In the following, we introduce a morphing function to convert between these two units. Finally, we include in our framework the duration modifiers that are used in musical scores, i.e., ties, dots, and tuplets.

Morphing between time units
Given a reference duration value δ, we define the reciprocal function $f(x) = \delta/x$ that maps every element $x \in dom(\mathcal{A})$ to its correspondent in $\mathcal{R}^\delta$, and vice-versa. It is trivial to see that this function is isomorphic and order-preserving (it preserves the ordering in the respective source/target domains, even though the order in $\mathcal{A}$ is reversed with respect to $\mathcal{R}$); it follows that f is a *Homomorphism*, i.e. $\forall a, b \in dom(\mathcal{R}), \ f(a \otimes^{\mathcal{R}} b) = f(a) \otimes^{\mathcal{A}} f(b)$. The choice of δ has interesting practical implications. For example, by setting it to a beat duration (which depends on the time signature), we obtain units typically used in music research to reduce the dependency on the time signature. By setting it to a quarter note duration we obtain the so-called *quarter length* durations, commonly used for general applications since they do not depend on other score parameters.

	$\mathbb{S}$	$\oplus$	$\otimes$
ASD $\mathcal{A}$	$\mathbb{Q}^+ \cup \{\infty\}$	$\min(a, b)$	$\frac{ab}{a+b}$
RSD $\mathcal{R}$	$\mathbb{Q}^+ \cup \{0\}$	$\max(a, b)$	$a + b$

Table 2. Table comparing the semirings of Absolute and Relative Symbolic Durations.

Duration modifiers

In a musical score, there are some graphical symbols, i.e., ties, dots, and tuplet groupings, that modify the duration of the notes/rests they are assigned to. In this section, we will define ties, dots, and tuplets as functions applied to elements of either $\mathcal{A}$ or $\mathcal{R}$. We use the symbol $\mathcal{X}$ to refer to either of the structures $\mathcal{A}$ or $\mathcal{R}$.

First, let us consider the ties between notes. The total duration of two tied notes $a, b \in \mathcal{X}$ can be easily captured by the $\otimes$ operation.

Definition 1. *The total duration of two tied notes $a, b \in \mathcal{X}$ is given by function tie :* $\mathcal{X} \times \mathcal{X} \to \mathcal{X}$

$$\text{tie}(a, b) = a \otimes^{\mathcal{X}} b \tag{3}$$

Another musical concept that can prolong the duration of a musical note is the dot. A dotted note a can be seen as a function dot applied to the note a. This can be generalized for an arbitrary number of dots:

Definition 2. *The function dot : $\mathcal{X} \times \mathbb{N} \to \mathcal{X}$ applied to a note $a \in \mathcal{X}$ is inductively defined as follows:*

$$\text{dot}(a, 0) = a \tag{4}$$

$$\text{dot}(a, n+1) = \text{repeat}(a, \frac{1}{2^{n+1}}) \otimes^{\mathcal{X}} \text{dot}(a, n) \tag{5}$$

Another function that can be used to construct musical duration is the tuplet function. The duration of a note in a tuplet of total duration $a \in \mathcal{X}$ can be seen as a function with two parameters, the base note duration a and the type of tuple γ (in this case 3 for triplet).

Definition 3. *Let $a \in \mathcal{X}$, $\gamma \in \mathbb{N}_{>2}$. The tuplet function, $t : \mathcal{X} \times \mathbb{N}^* \to \mathcal{X}$, is defined as:*

$$t(a, \gamma) = \text{repeat}(a, \frac{2}{\gamma}) \tag{6}$$

Example 2. We use the formalisms introduced in this section on the problem of Example 1, where the goal was to import the score from an MEI file, split the third note (the C) in the top voice into two, and produce a pianoroll representation. This process can abstractly be described solved as: (1) read all durations $[d_1, d_2, \ldots, d_n]$ from the input MEI file; (2) compute the values of the notes under the triplet with Definition 3; (3) split the third note into two notes with duration $d_{\text{new}} = \text{repeat}(d_3, 1/2)$; (4) find each note onset and offset position by $[d_1 \otimes d_2 \otimes \cdots \otimes d_n]$; (5) for the last note offset of each voice, compute the maximum with the $\oplus$ operator; (6) output the pianoroll representations for the two voices, using the start times and durations thus calculated.

3   From Abstract Description to Algorithm Implementation

In the previous section, we introduced an abstract formalism to describe algorithms on music sequences. We now discuss cases where it is more efficient to perform such algorithms in ASD units or in RSD units.

3.1 Advantages and Disadvantages

The use of ASD units can bring advantages in terms of data types because it can give a prevalence of integers over floating point (and periodic) values. For this to be the case, we need to deal with durations that span a maximum of a whole note. In a 4/4 piece, this will correspond to durations of one measure. This does not mean that algorithms implemented in ASD units cannot handle multiple measures, but rather that they should follow a "divide et impera" principle where every measure is handled independently. This is already quite common in file-parsing systems since scores are encoded measure by measure in file formats such as MusicXML and MEI.

In terms of the number of computations, ASD units are ideal for applications that concern the graphical symbols used in the score, for example, changing the pitch of a note, changing a duration, or segmenting a musical score. Such applications can skip the costly computation of common divisor, and conversion to RSD units altogether. Instead, when the measure is not specified (which could be the case, for example, in handling a MIDI file), or when we want to do operations that don't follow the measure segmentation (e.g., segmenting a score between measures), the usage of RSD units is preferred.

Example 3. Let us consider the problem of Example 1. By considering ASD units, we can parse the input score file simply by copying the values for the note graphical durations. The splitting of the third note of duration 16 in two parts yields two notes of duration $16 * 2 = 32$.

A big limit in the efficiency of ASD units is posed by time signatures where the beat is a dotted note, for example, 6/8. A dotted note will make the duration assume noninteger, or even periodic, values. A possible solution to this problem is given in the next section.

3.2 The lazy evaluation case

It is common for systems that deal with musical scores to have a generic import function, where the score file is converted to some internal representation. If in In this step, the user did not yet specify the set of operations they intend to perform, the choice of whether to use ASD or RSD cannot be performed made. In order to let the system choose between ASD and RSD to exploit the advantages described in the previous section, we suggest using a *lazy evaluation* parsing strategy. First, we propose to reduce the domain by considering only the *ASD* values $\{2^n \mid \forall n \in [0..7]\}$ (i.e. only single graphical note/rest symbols). Duration modifiers such as dots or tuplets are imported as functions dot or tuple without being computed. Only when the user specifies a task, will these functions be resolved to actual values, and the task is performed either in ASD or RSD units, depending on what would allow for the most efficient computation. From a functional programming perspective, this can be viewed as a Monad transformation [Wad92] of the parsed elements.

3.3 Implementation

We provide a proof of concept of the practical utility of the methods introduced in this paper, by implementing some functions in the Python library Partitura [Can+22]. The core of this library, i.e., the *Timeline* object, uses RSD units, in particular on the common divisor technique described in the introduction. However, some functions in the file parsing module are modular enough to make it possible to run them in ADS without the need of making major changes to the rest of the library. These are: (1) the functions to compute the common divisor for integer encoding of RSD durations, (2) the function to find the longest voice in a measure, and (3) the computation of the actual duration for a note inside a tuplet. We also implement an alternative (still partial) parser of **Kern files that leverages a lazy evaluation approach.

4 Conclusions and Discussion

In this paper, we proposed an alternative approach to handling the symbolic music durations from musical scores, that is based on absolute symbolic duration (ASD) units. We formalized ASD, and the (typically used) relative symbolic duration (RSD) units, in a single mathematical framework, and paired them with two operations. The result is two semirings: algebraic structures that enable an abstract description of algorithms on symbolic durations. We then moved to a more practical discussion and described some use cases where one unit is more efficient than the other, in terms of data types (integers vs floating point) and number of calculations. Finally, we advocated a functional parsing of symbolic music formats that can select the most efficient way of performing the various operations in an algorithm and enable considerable speed-up for common use cases.

It is clear that the proposals in this paper are mostly of theoretical interest, and belong to the research branch that formalizes musical elements with mathematical structures [AB19; Maz12; Pop+16]. However, our interest in this topic started from our practical experience with parsing and processing musical score files to use their information as input for music information retrieval (MIR) systems. While the improvement in efficiency that our methods may enable is negligible for a single score, large deep-learning models have to load thousands of scores, thus making each small optimization much more useful. For example, we will probably soon see some general tokenization techniques for musical scores (similar to the multiple ones that have been proposed for MIDI files [Fra+21]); in this context, a tokenization that focuses on the graphical symbols using only ASD, could enable major speedups in computing time.

5 Acknowledgements

References

[Wad92]   Philip Wadler. "The essence of functional programming". In: *Proceedings of the ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 1992, pp. 1–14.

[GP97]   Stéphane Gaubert and Max Plus. "Methods and applications of (max,+) linear algebra". In: *STACS 97: 14th Annual Symposium on Theoretical Aspects of Computer Science Lübeck, Germany February 27–March 1, 1997 Proceedings 14*. Springer. 1997, pp. 261–282.

[Pin98]   Jean-Eric Pin. *Tropical semirings*. Cambridge Univ. Press, Cambridge, 1998.

[Moh02]   Mehryar Mohri. "Semiring frameworks and algorithms for shortest-distance problems". In: *Journal of Automata, Languages and Combinatorics* 7.3 (2002), pp. 321–350.

[Hua08]   Liang Huang. "Advanced Dynamic Programming in Semiring and Hypergraph Frameworks". In: *Int. Committee on Computational Linguistics Conference (COLING)*. 2008.

[DK09]   Manfred Droste and Werner Kuich. "Semirings and formal power series". In: *Handbook of Weighted Automata*. Springer, 2009, pp. 3–28.

[CA10]   Michael Scott Cuthbert and Christopher Ariza. "Music21: A Toolkit for Computer-Aided Musicology and Symbolic Music Data." In: *Proocedings of the International Society for Music Information Retrieval Conference (ISMIR)*. 2010, pp. 637–642.

[DSA10]   Wieslaw A Dudek, Muhammad Shabir, and Rukhshanda Anjum. "Characterizations of hemirings by their h-ideals". In: *Computers & Mathematics with Applications* 59.9 (2010), pp. 3167–3179.

[Maz12]   Guerino Mazzola. *The topos of music: geometric logic of concepts, theory, and performance*. Birkhäuser, 2012.

[Pop+16]   Alexandre Popoff et al. "From K-nets to PK-nets: a categorical approach". In: *Perspectives of New Music* 54.2 (2016), pp. 5–63.

[AB19]   Giovanni Albini and Marco Paolo Bernardi. "Tropical Generalized Interval Systems". In: *Mathematics and Computation in Music*. Ed. by Mariana Montiel, Francisco Gomez-Martin, and Octavio A. Agustin-Aquino. Springer International Publishing, 2019, pp. 73–83.

[Fos+19]   Francesco Foscarin et al. "A parse-based framework for coupled rhythm quantization and score structuring". In: *Proceedings of the International Conference on Mathematics and Computation in Music (MCM)*. Springer. 2019, pp. 248–260.

[Fos+20]   Francesco Foscarin et al. "ASAP: a dataset of aligned scores and performances for piano transcription". In: *Proocedings of the International Society for Music Information Retrieval Conference (ISMIR)*. 2020, pp. 534–541.

[FRT21]   Francesco Foscarin, Philippe Rigaux, and Virginie Thion. "Data quality assessment in digital score libraries: The GioQoso Project". In: *International Journal on Digital Libraries* 22 (2021), pp. 159–173.

[Fra+21] Nathan Fradet et al. "MidiTok: A Python package for MIDI file tokenization". In: *Late-Breaking Demo Session of the International Society for Music Information Retrieval Conference*. 2021.

[SMS21] MK Sen, SK Maity, and KP Shum. "Some Aspects of Semirings". In: *Southeast Asian Bulletin of Mathematics* 45.6 (2021).

[Can+22] Carlos Eduardo Cancino-Chacón et al. "Partitura: A Python Package for Symbolic Music Processing". In: *Proceedings of the Music Encoding Conference (MEC2022)*. Halifax, Canada, 2022.

[GJ22] Mathieu Giraud and Florent Jacquemard. "Weighted Automata Computation of Edit Distances with Consolidations and Fragmentations". In: *Information and Computation* 282 (2022).