Tokenization of MIDI Sequences for Transcription

Yosuke Amagasu, Florent Jacquemard, Masahiko Sakai

TOKENIZATION OF MIDI SEQUENCES FOR TRANSCRIPTION

Florent Jacquemard
INRIA/CNAM
florent.jacquemard@inria.fr

Masahiko Sakai
Nagoya University
sakai@i.nagoya-u.ac.jp

Yosuke Amagasu
Nagoya University
amagasu@trs.css.i.nagoya-u.ac.jp

ABSTRACT

There generally exists no simple one-to-one relationship between the events of a MIDI sequence, such as note-on and note-off messages, and the corresponding music notation elements, such as notes, rests, chords, and ornaments.

We propose a method for building a formal correspondence between them through a notion of tokens in an input MIDI event sequence and an effective tokenization approach based on a hierarchical representation of music scores. Our tokenization procedure is integrated with an algorithm for music transcription based on parsing *wrt* a weighted tree grammar. Its effectiveness is shown on examples.

1. INTRODUCTION

In the context of compiling (lexical analysis) and in Natural Language Processing (NLP), tokenization is the act of dividing a character sequence into elementary subsequences called tokens. The primary purpose of tokenization is to identify, in the early steps of processing, meaningful subsequences of characters in preparation for the next steps. For instance, tokenization consists of finding keywords and identifiers with automata-based pattern-matching techniques in compiling. For NLP based on statistical learning, tokenization aims at building sequential encodings of character sequences into integer vectors, using a dictionary of tokens, for training language models.

In languages based on Latin alphabets (including programming languages), tokens are generally words or subwords. The tokenization into words, in particular, is eased by space separation. The case of written Japanese language is more complicated since there is no space separation (although some punctuation exists). Therefore, some real-time tokenization process is required when reading a text. In the case of hiragana characters (phonetic characters), ambiguity may arise in tokenization, leading to confusion.

Similar ambiguity problems may occur when processing symbolic music data in sequential form, in particular MIDI data, in order to extract more structured information. A MIDI flow (or file) is essentially an unstructured sequence of *messages*, each one representing an elementary *event* for the production of music with an electronic instrument. Two common events are the start of a note (*note-on*), corresponding to a key press, and the end of note (*note-off*), for a key release. On the other hand, higher-level musical elements of Common Western musical notation, such as notes, rests, chords, grace notes, and other ornaments may correspond to several successive MIDI events, and conversely. Therefore, extracting such notational elements from a MIDI flow, in *e.g.*, a task like MIDI-to-score transcription, requires a processing similar to tokenization.

Let us illustrate this point of view with sneak observations on the short MIDI sequence of Figure 1, and some of its possible denotations as a 4/4 measure in a music score. We use a representation similar to a piano-roll, with note pitches on the vertical axis, time on the horizontal axis, and note-on events depicted as black dots and note-off as white dots. Several grouping of these events into *tokens* are considered in cases (a) to (h) of Figure 1 and correspond to different notations. Note that the piano-roll is the same in all these cases. In each case, the tokens, called T_0 to T_n (n depending on the case) are represented with blue boxes. Intuitively, all events in a token T_i represent one or several music notation elements occurring at the same time position τ_i in a score. The time position τ_i associated with the token T_i is indicated on the axis under the corresponding blue box. The notation elements represented may be *e.g.*, one note, one chord, some grace notes and one note, *etc*. In other terms, the groupings defined by the tokens can be seen as a re-alignment of MIDI events, like with a rhythm quantization functionality of a Digital Audio Workstation (DAW), except that the alignment points τ_i are not evenly distributed in a grid. One strategy for defining the alignment points τ_i according to hierarchical structures is proposed in Section 4.

Going back to Figure 1, in the first tokenization (a), and corresponding notation (a)', the two first note-on events (D4 and A4) belong both to the first token T_0, and their matching note-off events belong to the next tokens, resp. T_1 and T_2. This situation corresponds to two notes starting simultaneously on the first beat of the measure (time position $\tau_0 = 0$), and ending on different later beats: beat 2 (time position $\tau_1 = \frac{1}{4}$) for D4, and beat 3 (time position $\tau_2 = \frac{1}{2}$) for A4. It is denoted as an interval tied to a quarter note.

In the tokenization in Figure1(f) however, both the note-on and note-off events of D4 belong to the first token T_0, aligned at time position $\tau_0 = 0$ (beat 1). It follows that this note is a grace-note located at τ_0. On the opposite, the A4 has its note-on in T_0 and its note-off in T_1, aligned at time position $\tau_1 = \frac{1}{2}$ (beat 3). Hence, this note is a half note, with the D4 as an ornament, see Figure 1(f)'.

The interpretation of the other tokens of Fig. 1 is similar.

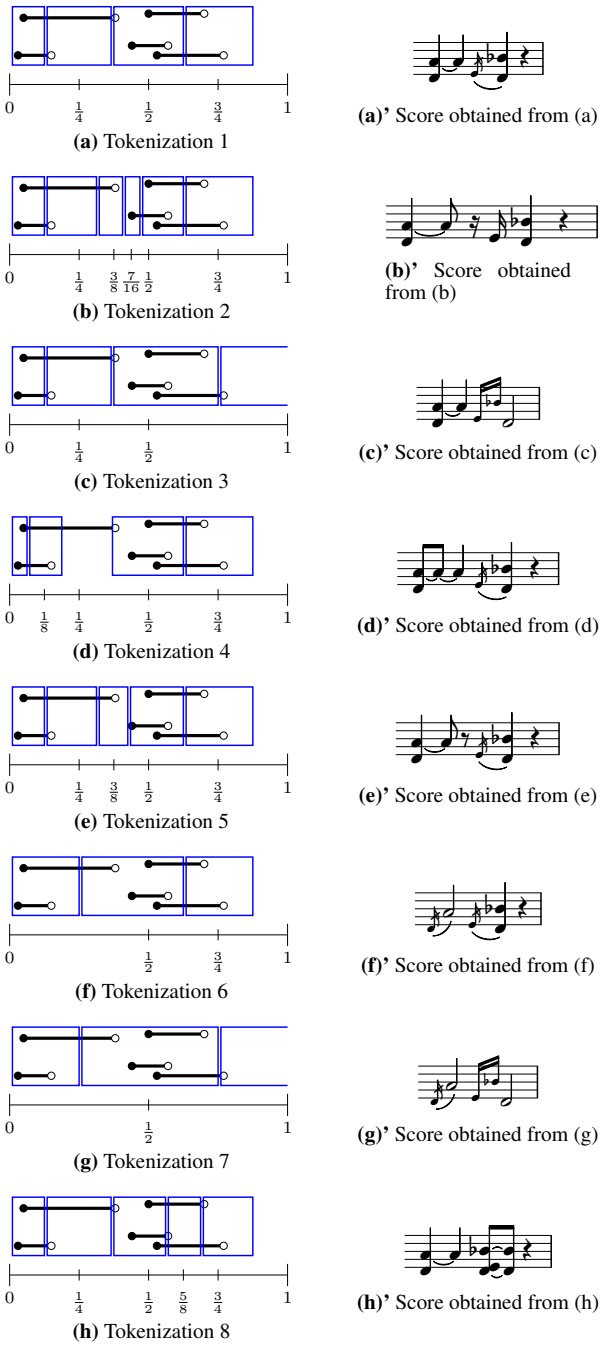


Figure 1: Examples of MIDI tokenizations and transcriptions.

Note that in case (c), and (g), two pairs of matching note-on and note-off events are embedded in the third token. It follows that the last note D4 has two grace notes E and B♭.

Therefore, we see in Figure 1 many possible denotations of the same MIDI sequence. How were chosen the tokens in this picture, in order to be associated a notation? Which denotation shall be considered the best?

In this paper, we propose a formal definition of the notion of token of MIDI events, as an attempt to clarify the correspondence between MIDI events and music notation elements. We characterize in particular what notation can be associated with a token, defining several *types* of tokens, corresponding to different notations. We focus in particular on notation elements which are not the most studied in the literature on music notation and symbolic music processing in general, namely the *rests* and the *ornaments*.

The main motivation for this study is the processing of MIDI data, in particular its transcription into Common Western music notation, a task sometimes referred to as MIDI-to-score Automatic Music Transcription (M2S-AMT). We propose an efficient procedure for tokenization and its integration into a M2S-AMT framework. The approach is based on *parsing* [1], in its usual computational sense: the inferrence of some (hierarchical) structure from unstructured (sequential) data. In these settings, we consider tree structures for defining hierarchically the time boundaries of tokens. A weighted tree grammar generates those trees, and the estimation of the best tree (*wrt* weight) in the grammar's language is performed by a dynamic programming algorithm. This method is implemented in a framework for MIDI-to-score transcription in development.

Related Work. Compiling (textual) programming languages generally involves steps of lexical analysis and syntactic analysis (parsing) [1]. In some sense, we follow a similar approach in our transcription approach, briefly described above. However, a significant difference is that, in our case, several tokenizations of the MIDI input are possible, and we must explore possibilities to choose the one giving the best notational result.

In the literature on music generation, several works propose sequential encodings of MIDI data in order to train models like the Transformer [2, 3, 4, 5, 6, 7, 8, 9, 10]. Some of these encodings, often referred to as *MIDI tokenization*, are implemented in the library MIDItok [11]. Their purpose is to format MIDI data as a sequence of tokens, where, roughly, each token represents one elementary component of a MIDI event (position, duration, pitch value, *etc*). This notion of elementary token is very different from the one we present here, where every token is made of several MIDI events (considered atomic) and conveys higher-level musical information about the notation. Some strategies for grouping elementary tokens are proposed *e.g.*, in [7], in order to reduce training sequences. Another critical difference is that the purpose of the above tokenization approaches is the generation of MIDI data by trained model, whereas our purpose is the construction of music notation from MIDI data.

The sequential input and structured output of M2S-AMT are of very different natures, and establishing a relation-

ship between input MIDI events and output score elements is not a trivial task, considering the richness of music notation. For instance, many existing M2S transcription tools, including general public ones, like MuseScore, fall short of detecting ornaments and rests. As observed formarly [12] problems often come from mismatches between input and output. Several transcription tools ignore note-off events to cope with the mismatches. In other works, note-on and note-off events are treated in different passes [15].

We believe that the notions of *tokens* and *token types* proposed here can help in making progress in the production of music notation, such as in M2S-AMT, in particular regarding the treatment of rests and ornaments in preprocessing. However, it should be noted that in the present work, we only consider homophonic input (monophonic voices including chords), whereas [15, 16] deal with the transcription piano input, which is a much harder problem.

The main contributions of this paper are the following:

- the definition of *tokens* as sequences of MIDI events corresponding to notation elements (Section 2),
- the proposition of an efficient procedure using trees for dividing a MIDI sequence into tokens (Section 3),
- a dynamic programming algorithm for parsing a MIDI sequence into structured music notation, joined with the above tokenization procedure (Section 4).

2. MIDI EVENTS, TOKENS AND SCORE ELEMENTS

In this section, we define a correspondence between music notation elements and sequences of MIDI events called *tokens*, as described informally in Figure 1. Each token is given a token type, a note, rest, or chord, with or without an ornament, corresponding to the score element. On the other hand, a MIDI event in a token may have an individual role, which we call an *event role in the token*.

2.1 MIDI Events

We consider a typical representation, in the form of so-called timestamped MIDI events, of the note-on and note-off messages in a MIDI file, often depicted as a piano-roll. A MIDI event e (or *event* for short) is made of the following components:

- a *real-time value* $\mathsf{ts}(e) \in \mathbb{Q}$, expressed in seconds,
- an integral *pitch value* $\mathsf{pitch}(e) \in \{0, \ldots, 128\}$,
- an integral *velocity value* $\mathsf{vel}(e) \in \{0, \ldots, 128\}$,
- an attribute $\mathsf{flag}(e)$ which is either on when e is a *note-on* message, or off for a *note-off* message.

A MIDI *sequence* $E = \langle e_1, e_2, \ldots, e_n \rangle$ is a finite sequence of MIDI events with increasing timestamps, *i.e.*, such that $\mathsf{ts}(e_1) \leq \mathsf{ts}(e_2) \leq \cdots \leq \mathsf{ts}(e_n)$. The length of E is denoted by $|E| = n$ and the *concatenation* of two sequences E and E' is denoted by $E\,E'$. We use the set-like-notation $e \in E$ to express that the event e belongs to the MIDI sequence E. The subset of note-on (resp. note-off) events in a sequence E of events is denoted by $\mathsf{on}(E) = \{e \in E \mid \mathsf{flag}(e) = \mathsf{on}\}$ (resp. $\mathsf{off}(E) = \{e \in E \mid \mathsf{flag}(e) = \mathsf{off}\}$).

The event *matching* e in a sequence E, denoted by e^{-1} is the first note-off after e with the same pitch when e is a note-on, and the note-on event d such that $d^{-1} = e$ when e is a note-off. Formally, for all event e,

- $\mathsf{pitch}(e^{-1}) = \mathsf{pitch}(e)$ and $\mathsf{flag}(e^{-1}) \neq \mathsf{flag}(e)$,
- if $\mathsf{flag}(e) = \mathsf{on}$, then $\mathsf{ts}(e) < \mathsf{ts}(e^{-1})$ and for all e' such that $\mathsf{ts}(e) < \mathsf{ts}(e') < \mathsf{ts}(e^{-1})$, it holds that $\mathsf{pitch}(e') \neq \mathsf{pitch}(e)$, and
- if $\mathsf{flag}(e) = \mathsf{off}$, then $e^{-1} = d$ such that $d^{-1} = e$.

The matching operator is idempotent: $(e^{-1})^{-1} = e$. We shall consider below only well-formed MIDI sequences E, such that e^{-1} exists in E for all $e \in E$.

Example 1. *Figure 2 presents the events $e_1, \ldots, e_{10}$ of the MIDI sequence in Figure 1. This sequence is well-formed, and ${e_1}^{-1} = e_3$, ${e_2}^{-1} = e_4$, ${e_3}^{-1} = e_1$, ${e_4}^{-1} = e_2$, ${e_5}^{-1} = e_8$, …, ${e_{10}}^{-1} = e_7$.*

2.2 Tokens

In order to be converted into music notation, the MIDI events of an input sequence can be aligned to some salient time points, like the *grid points* in DAWs or in earlier rhythm quantization algorithms [17, 18]. It may happen that several events, neighbours in a MIDI sequence E are aligned to the same time point. The notion of token aims at capturing the subsequences of events of E made simultaneous after alignment. More precisely, a *token* of E is a non-empty subsequence $T = \langle e_i, \ldots, e_j \rangle$ of E with $1 \leq i \leq j \leq n$, that is moreover closed *wrt* time equality: every $e \in E$ such that $\mathsf{ts}(e) = \mathsf{ts}(e_k)$ for some $i \leq k \leq j$ must be in T.

We shall use below the number $ns_E(T)$ of notes sounding just after the timestamp ℓ of the last event of a token T. Formally, it is defined as the following cardinality:

$$ns_E(T) = \left|\left\{e \in E \mid \mathsf{flag}(e) = \mathsf{on}, \mathsf{ts}(e) \leq \ell < \mathsf{ts}(e^{-1})\right\}\right|$$

where $\ell = \mathsf{ts}(e)$ for the last event $e \in T$ *i.e.*, for all $e' \in T$, $\mathsf{ts}(e') \leq \mathsf{ts}(e)$.

Intuitively, a token aims at containing some MIDI events representing simultaneous score elements, *i.e.*, elements occurring at the same musical date (beat) in the score. This can be the case of several notes involved in a chord, or of one note with one or several grace notes, or another specific ornament (*mordent*, *gruppetto*, *trill*...). Note that we are talking here about the theoretical simultaneity of notes in the score, not of simultaneity in a performance. In the score, all the notes of an ornament occur at the same theoretical date, in beats, as the note they decorate. However, during a performance, they will occur shortly before or after the decorated note.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
ts	0.03	0.05	0.15	0.38	0.44	0.50	0.53	0.57	0.70	0.77
flag	on	on	off	off	on	on	on	off	off	off
pitch	D4	A4	D4	A4	E4	B♭4	D4	E4	B♭4	D4
vel	110	100	0	0	45	90	110	0	0	0

Here, MIDI pitches are presented by note name and octave; D4 = 62, E4 = 64, A4 = 69, and B♭4 = 70.

Figure 2: MIDI event sequence for Figure 1.

	$e^{-1} \in T$	$e^{-1} \notin T$
$e \in \mathsf{on}(T)$	*grace-note*	*note*
$e \in \mathsf{off}(T)$	*goff*	*noff*

Figure 3: Role of event e in token T.

2.3 Event Role in a Token

Only some combinations of MIDI events will appropriately fit into one case, corresponding to a notation, like in Figure 1. Having too many events in a token may make no sense. In order to capture tokens that can be considered valid, *i.e.*, that can be transcribed into simultaneous notation elements, we introduce the notion of *role* of a MIDI event e in a token T. It is defined according to the flag of e (on or off-note) and to the presence or not in T of the matching event e^{-1}, as presented in Figure 3.

Intuitively, the role *grace note* corresponds to notes with a theoretical duration of 0 in a score. It can be an *appogiatura*, an *accaciatura*, or a part of another ornament, which expresses the fact that both the start (note-on e) and the end (note-off e^{-1}) of the corresponding note belong to the same token T.

On the opposite, an event of role *note* corresponds to a note starting in a token T ($e \in \mathsf{on}(T)$), but ending in a subsequent token ($e^{-1} \notin T$).

A note-off event matching an event in the token will have role *goff*. Otherwise, it shall have role *noff*. The event role *goff* is important in the context of MIDI processing. Indeed, events with this role correspond to *micro-rests* that may occur *e.g.*, when a key of a MIDI keyboard is released before the key for the next note is pressed (*i.e.*, the play is not enough *legato*). In general, one does not want such artifacts to be transcribed literally, and in several transcription procedures, micro-rests are discarded in a pre-processing step relying on parameters to be set, like the maximal duration of a micro-rest. In our approach, micro-rests are detected as a special case of event role *noff* or *goff* in a token, and may be discarded at will.

Example 2. *The roles of events in the tokens represented in (a) and (b) of Figure 1 are displayed in Figure 4.*

2.4 Token Types and Validity

We consider a finite set $\mathcal{K}$ of type names that can be assigned a token T in a MIDI sequence E as follows, according to the respective roles of the events in T:

T has type *chord* with n (> 0) notes and an *ornament* of size p (≥ 0), denoted by $\mathsf{ch}_{n,p}$, if and only if

it contains n events of role *note*,

it contains p events of role *grace-note*,

each *grace note* occurs before any *note* in T, and $ns_E(T) = n$ (the n notes sound after the token).

T has type *rest* denoted by r, if and only if $T = \mathsf{off}(T)(\neq \emptyset)$ and $ns_E(T) = 0$,

T has type *partial continuation*, denoted by pc, if and only if $T = \mathsf{off}(T)(\neq \emptyset)$, and $ns_E(T) > 0$.

The cases of a single note or an interval correspond respectively to the types *chord* with 1 note and *chord* with 2 notes. We use the name *chord* for these types of abuse in order to shorten the definition of token types. As explained above, an ornament of size p corresponds to a sequence of events occurring at the same musical date (beat) in a score. In this description, we use, for simplicity, the generic term of "*ornament of size* p", but we could distinguish between different kinds of ornaments, such as *mordent*, *gruppetto* or *trill*, or also a *tremolo* (or *roll* in the case of drums), by analyzing the respective MIDI pitch values of the events involved.

A *rest* corresponds to the case where some note ends during the token T (a note-off $e \in T$). No other note or grace note shall start in the same token (the condition $T = \mathsf{off}(T)$ implies that $\mathsf{on}(T) = \emptyset$), and every note started before ends in T (condition $ns_E(T) = 0$). Moreover, T contains no grace note (another consequence of $\mathsf{on}(T) = \emptyset$), since in CW music notation, no grace notes or ornaments may be attached to rests. The latter case may, however, correspond to a note played very shortly, and wrongly interpreted as a grace note. In our framework, we consider an option where such a grace note is transcribed into a note with an articulation *staccato* or *staccatissimo*.

A *continuation* represents the prolongation of an event (either a note or a chord), with a tie or a dot. A continuation is *partial* when some but not all chord notes are prolongated. Remark that no token exists with type (full) continuation because we defined an empty set, which corresponds to a continuation, is not a token.

2.5 Token Validity

Tokens of the above types will be considered *valid* or not for transcription, according to the case of input considered.

2.5.1 Monophonic case

When the input is considered as strictly monophonic, at most one note shall sound at a time. Hence, a token considered valid in this case is a note, *i.e.*, a chord with 1 note, with an ornament of size $p \geq 0$ or a rest. This case is appropriate *e.g.*, for the M2S transcription of a single voice, of a monophonic instrument like winds or one piano voice in strict counterpoint.

	Figure 1(a)										Figure 1(b)									
event	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8	e_9	e_{10}
role	n	n	no	no	gn	n	n	go	no	no	n	n	no	no	n	n	n	no	no	no
token	T_0	T_0	T_1	T_2	T_2	T_2	T_2	T_2	T_3	T_3	T_0	T_0	T_1	T_2	T_3	T_4	T_4	T_4	T_5	T_5

Figure 4: Event roles in some tokens of Figure 1.

2.5.2 Homophonic case

This case generalizes the previous one by allowing some chords in a monophonic voice. Then, tokens of all three above types are considered valid, and the others are not valid (discarded in the transcription procedure).

This case is appropriate for the M2S transcription of monophonic instruments like strings (where some chords may occur) or one voice of the piano.

2.5.3 Case of Drums

In the case of MIDI files recorded with electronic drumkits, see *e.g.*, [13, 14], the MIDI pitches encode the drumkit parts (snare drum, kick, toms, hi-hat, and different kinds of cymbals) and the play mode (*e.g.*, on the head or rim of snare drum or toms, on the ride, edge, or bow of cymbals, and so on) Moreover, in such input, the note-off events are not significant (for instance, they are added at a fixed duration, *e.g.*, 20ms, after the matching note-on event) and can be safely ignored. Concretely, for this case, we change the definition of event role such that every $e \in \mathsf{off}(T)$ has an extra role *ignored*, which excludes the rests from the transcription output. Actually, in a post-processing step, some rests are re-introduced by conversion of some continuations, when appropriate for the readability of the score (see [14] for details).

The definition of token validity follows additional constraints for drums. First, only ornaments of size 1, called *flams*, are allowed, and only for some parts of the drumkit. Second, chords are allowed (the drum is a polyphonic instrument) but with some restrictions: sticks cannot play more than two notes, and two notes played by pedals involved in the same chord. Finally, some particular effects like *cross-stick*, *rim-shot* or *buzz-roll* (see descriptions in [14]) require a particular processing of tokens. Some possible errors of capitation by MIDI sensors also need to be handled. We leave the details about these particular cases out of the scope of this paper.

2.5.4 Polyphonic case

This paper does not cover the polyphonic case, which requires a step of voice separation. To directly cope with polyphonic MIDI sequences, tokens are not just a partition of the input sequence. In fact, multiple tokens shall correspond to interleaved events in a MIDI-sequence, which is a complex extension to handle.

We call *tokenizer* a procedure that takes in input a MIDI sequence E and returns a sequence of valid tokens $T_1, \ldots, T_k$ that form a partition of E.

3. TREE-BASED TOKENIZATION PROCEDURE

In this section, we propose a tokenizer based on a tree structure. Intuitively, the idea is that the salient time points defining the tokens (by alignment of MIDI events) are characterized by a recursive subdivision of time intervals to reflect a metric organization of musical events [19]. This point of view is more involved (and more realistic) than considering regular space points in a grid (*e.g.*, one point every 16th note). It corresponds to a tree-based representation of music notation.

3.1 Real and Musical Time Scales

Here, we consider input MIDI sequences corresponding to performances. Their MIDI events are timestamped in a *real time* scale, whose units are seconds. By contrast, the durations in a music score are expressed in a *musical time* scale, whose units are beats or bars (given a time signature). The correspondence between these two time-scales is ensured by a tempo function, converting real-time values to musical-time ones. For the sake of presentation, we assume here a coincidence of these two-time scales, or equivalently, a constant tempo. We especially use a bar as a unit of musical time. The extension of the approach presented here to varying tempo functions is out of the scope of this paper.

3.2 Time Intervals

A *time interval* $I = [\tau_0, \tau_1)$ is defined by a pair of time points $\tau_0 \in \mathbb{Q}$ and $\tau_1 \in \mathbb{Q} \cup \{+\infty\}$: I starts at time τ_0 and ends just before τ_1, *i.e.*, $[\tau_0, \tau_1) = \{\tau \mid \tau_0 \leq \tau < \tau_1\}$. In the sequel, we use the notation $start(I)$ for τ_0 and $end(I)$ for τ_1. The interval $div_n(I, i)$, for $1 \leq i \leq n$, is the ith sub-interval obtained by splitting I into n parts of equal duration $\Delta = \dfrac{\tau_1 - \tau_0}{n}$ when $I = [\tau_0, \tau_1)$,

$$div_n(I, i) = [\tau_0 + (i-1) \cdot \Delta,\ \tau_0 + i \cdot \Delta).$$

If $end(I) = +\infty$, then $div_n(I, i)$ is undefined. Moreover, for $I = [\tau_0, \tau_1)$ with $\tau_1 > \tau_0 + 1$ (including the case $\tau_1 = +\infty$), we define $unit(I)$ as the subinterval $[\tau_0, \tau_0 + 1)$ of duration one bar, and $rem(I)$ for the remaining interval $[\tau_0 + 1, \tau_1)$, *i.e.*, $rem(I) = I \setminus unit(I)$. The latter is undefined when $\tau_1 < \tau_0 + 1$.

3.3 Trees

The trees defined in this section represent hierarchies of nested time intervals, defining salient points for alignment in tokens. Intuitively, the higher the points are in the hierarchy (in the tree), the strongest the corresponding beat is, from a meter point of view. The trees are labeled with two kinds of symbols:

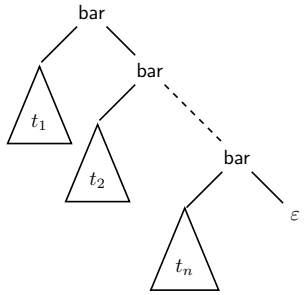


Figure 5: The form of trees

- every inner node is labeled with a function symbol from a fixed finite set $\mathcal{F}$,
- every leaf is a node labeled with the empty symbol ε or a token type in $\mathcal{K}$, as defined in Section 2.4.

Every function symbol of $\mathcal{F}$ is associated with an operation on time intervals. We consider here two kinds of such operations:

div_n, for $n > 0$, divides an interval I into n sub-intervals $div_n(I,1), \ldots, div_n(I,n)$ of the same duration;

bar (*bar split*), divides an interval I in two parts: $unit(I)$ and $rem(I)$.

Here, trees are in the form of Figure 5. Each left child of bar represents the content of one measure. We restrict the second argument of the bottom bar to ε for termination.

The set of trees labeled with symbols of $\mathcal{F}$ and $\mathcal{K}$ is denoted by $\mathcal{T}(\mathcal{F},\mathcal{K})$. We denote by $yield(t)$ the sequence of token type names of $\mathcal{K}$ labelling the leaves of a tree $t \in \mathcal{T}(\mathcal{F},\mathcal{K})$. Note that $yield(t)$ does not contain an empty leaf symbol ε.

Example 3. *Figure 6 depicts the trees corresponding to the scores in cases (a) to (d) of Figure 1. Each tree, rooted with* bar*, represents one measure whose content is the left subtree.*

3.4 Tree-based Tokenizer

Let $t \in \mathcal{T}(\mathcal{F},\mathcal{K})$ and let I be a time interval associated with the root node of t. We can associate a sub-interval of I to every other node of t, following the above interpretation of the symbols of $\mathcal{F}$.

Example 4. *In Figure 6, the time intervals associated with each node of trees are represented in red. They are computed from the interval assigned to the root node by recursive application of the operators labeling the nodes.*

Formally, the interval assignment is defined as follows, where $itv(t,I)$ is the sequence of the intervals labelling the leaves of t.

$$\begin{aligned} itv\big(\mathsf{div}_n(t_1,\ldots,t_n),I\big) &= itv(t_1, div_n(I,1)) \ldots \\ &\quad itv(t_n, div_n(I,n)) \qquad \text{if } n>0 \text{ and } end(I)\in\mathbb{Q}, \\ itv\big(\mathsf{bar}(t_1,t_2),I\big) &= itv(t_1,unit(I))\, itv(t_2,rem(I)) \\ &\qquad\qquad \text{if } \ end(I)=+\infty, \\ itv\big(\theta,I\big) &= I \qquad\qquad \text{for } \theta\in\mathcal{K}\cup\{\varepsilon\}. \end{aligned}$$

It can be observed that, when it is defined, $itv(t,I)$ forms a partition of I. It is always defined when $I=[\tau_0,+\infty)$ and t has the form of a right combination.

The sequence of time points induced by the partition $itv(t,I)$ is called the *grid* of t of carrier I, denoted by $grid(t,I)$. Formally, if $itv(t,I)$ is a partition of the form $[\tau_0,\tau_1),[\tau_1,\tau_2),\ldots,[\tau_{k-1},\tau_k)$ with $\tau_0 = start(I)$ and $\tau_k = end(I)$, for some $k \geq 0$ (τ_k might be $+\infty$), then:

$$grid(t,I) = \langle \tau_0,\ldots,\tau_k\rangle$$

To every time point τ_i in the grid, we can associate a set T_i of events in E containing the MIDI events closer to τ_i than to its neighbours τ_{i-1} and τ_{i+1} in the grid. For a formal definition, we introduce a notation $cp(\tau_1,\tau_2)$ to the present center point of τ_1 and τ_2:

$$cp(\tau_1,\tau_2) = \tau_1 + \frac{\tau_2-\tau_1}{2} = \frac{\tau_1+\tau_2}{2}$$

$$\begin{aligned} T_0 &= \big\{e\in E \mid \tau_0 \leq \mathsf{ts}(e) < cp(\tau_0,\tau_1)\big\}, \\ T_i &= \big\{e\in E \,\big|\, \ cp(\tau_{i-1},\tau_i) \leq \mathsf{ts}(e) < cp(\tau_i,\tau_{i+1}) \ \big\} \\ &\qquad\qquad\qquad\qquad \text{for all } i, 0<i<k. \end{aligned}$$

Based on $grid(t,I)$ and the above sequence $\langle T_0,\ldots,T_{k-1}\rangle$, we define

$$tokenize(t,I) = \langle (T_{i_0},\tau_{i_0}),\ldots,(T_{i_p},\tau_{i_p})\rangle$$

where $0 \leq i_0 < \ldots < i_p \leq k-1$ is the sequence of indices $0 \leq j \leq p$ such that $T_{i_j} \neq \emptyset$. In the sequel, we use a renumbered tokenized sequence as

$$tokenize(t,I) = \langle (T_0,\tau_0),\ldots,(T_p,\tau_p)\rangle.$$

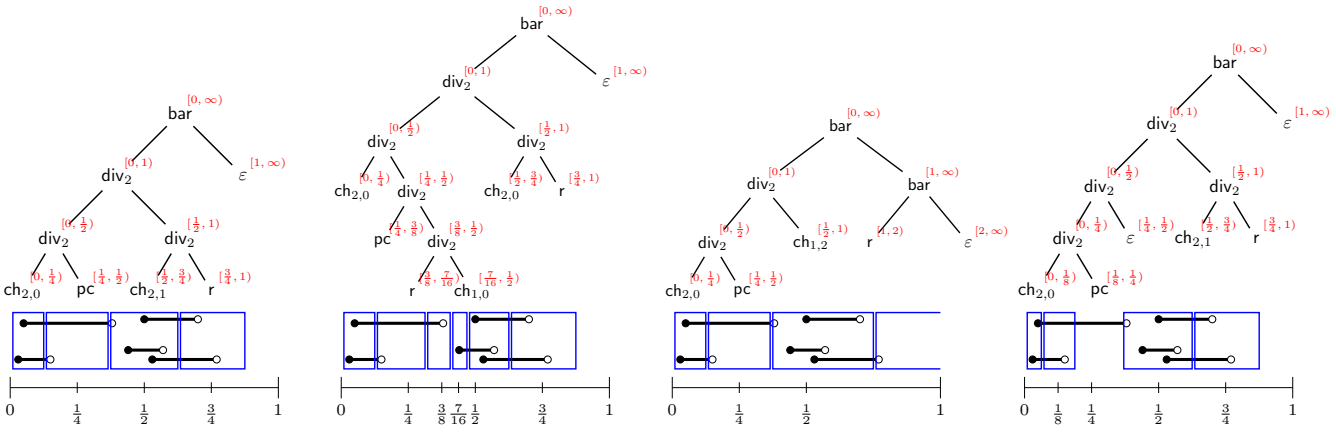
Example 5. *For the trees of Figure 6 it holds that:*

$$\begin{aligned} grid\big(t_1,[0,\infty)\big) &= \langle 0,\tfrac14,\tfrac12,\tfrac34,1,\infty\rangle, \\ tokenize\big(t_1,[0,\infty)\big) &= \langle (T_0,0),(T_1,\tfrac14),(T_2,\tfrac12),(T_3,\tfrac34)\rangle, \\ grid\big(t_2,[0,\infty)\big) &= \langle 0,\tfrac14,\tfrac38,\tfrac{7}{16},\tfrac12,\tfrac34,1,\infty\rangle, \\ tokenize\big(t_2,[0,\infty)\big) &= \\ &\quad \langle (T_0,0),(T_1,\tfrac14),(T_2,\tfrac38),(T_3,\tfrac{7}{16}),(T_4,\tfrac12),(T_5,\tfrac34)\rangle, \\ grid\big(t_3,[0,\infty)\big) &= \langle 0,\tfrac14,\tfrac12,1,2,+\infty\rangle \\ tokenize\big(t_3,[0,\infty)\big) &= \langle (T_0,0),(T_1,\tfrac14),(T_2,\tfrac12),(T_3,1)\rangle \\ grid\big(t_4,[0,\infty)\big) &= \langle 0,\tfrac18,\tfrac14,\tfrac12,\tfrac34,1,+\infty\rangle, \\ tokenize\big(t_4,[0,\infty)\big) &= \langle (T_0,0),(T_1,\tfrac18),(T_2,\tfrac12),(T_3,\tfrac34)\rangle, \end{aligned}$$

where $T_0, T_1, \ldots$ are the displayed tokens from left to right in the corresponding figures.

4. TREE LANGUAGES, PARSING AND M2S-TRANSCRIPTION

In this section, we propose to use weighted tree grammar in order to define languages of trees of Section 3.3. Then we reduce M2S-AMT into the problem of parsing an input MIDI sequence *wrt* such a grammar.



(a) Tree t_1 for Figure 1(a) **(b)** Tree t_2 for Figure 1(b) **(c)** Tree t_3 for Figure 1(c) **(d)** Tree t_4 for Figure 1(d)

Figure 6: Rhythm Trees for Figure 1

4.1 Tree Grammar and Cost of Readability

A weighted tree grammar is a tuple $\mathcal{G} = \langle \mathcal{N}, A_0, \mathcal{F}, \mathcal{K}, \mathcal{R} \rangle$ where $\mathcal{N}$ is a finite set of non-terminal symbols, $A_0 \in \mathcal{N}$ is the initial non-terminal, $\mathcal{F}$ and $\mathcal{K}$ are as in Section 3.3, and $\mathcal{R}$ is a finite set of *weighted production rules*. Every production rule of $\mathcal{R}$ has one of the following forms:

$A \xrightarrow{w} f(A_1, \ldots, A_n)$
where $A, A_1, \ldots, A_n \in \mathcal{N}$, $f \in \mathcal{F}$, and $w \in \mathbb{Q}$,
$A \xrightarrow{w} \theta \mid \varepsilon$
where $A \in \mathcal{N}$, $\theta \in \mathcal{K}$ and $w \in \mathbb{Q}$.

For each rule ρ of one of the two above kinds, $w \in \mathbb{Q}$ is called the weight of ρ, denoted by $weight(\rho) = w$. It is used to compute a value of tree complexity (called *cost of readability*). With the rules of the second kind, a readability cost value w is associated with every $\theta \in \mathcal{K}$ and non terminal A.
Let us extend $\mathcal{T}(\mathcal{F}, \mathcal{K} \cup \{\varepsilon\})$ to $\mathcal{T}(\mathcal{F}, \mathcal{K} \cup \{\varepsilon\} \cup \mathcal{N})$, the set of trees whose leaves can be labeled with type names of $\mathcal{K}$ or non-terminals of $\mathcal{N}$. A derivation D of $\mathcal{G}$ is a sequence of the form: $A_0 \xrightarrow{\rho_1} t_1 \xrightarrow{\rho_2} \ldots t_{k-1} \xrightarrow{\rho_k} t_k$, where, for all $0 < i \leq k$, $t_{i-1}, t_i \in \mathcal{T}(\mathcal{F}, \mathcal{K} \cup \{\varepsilon\} \cup \mathcal{N})$, $\rho_i = A_i \to u_i \in \mathcal{R}$, the leftmost innermost occurence of a non-terminal in t_{i-1} is A_i and t_i is obtained from t_{i-1} by replacing this occurence of A_i by u_i. We also write $A_0 \xrightarrow{D} t_k$ and abbreviate $D = \langle \rho_1, \ldots, \rho_k \rangle$. The above derivation is called *complete* if $t_k \in \mathcal{T}(\mathcal{F}, \mathcal{K} \cup \{\varepsilon\})$ (*i.e.*, is contains no more non-terminals).

The *weight* of D is defined by:

$$weight(D) = \sum_{i=1}^{k} weight(\rho_i).$$

The weight of a tree $t \in \mathcal{T}(\mathcal{F}, \mathcal{K})$ *wrt* $\mathcal{G}$, also called *cost of readability* of t is:

$$cr_{\mathcal{G}}(t) = \min_{A_0 \xrightarrow{D} t} weight(D).$$

The grammar $\mathcal{G}$ might be omitted when clear from the context. The purpose of tree grammars is to define a restricted prior language of trees that are an acceptable output of transcription.

4.2 Cost of Alignment

We assume that a *cost alignment function* $ca(\theta, T, \tau, \tau^{prev})$ is associated to a token T, a token type $\theta \in \mathcal{K}$, and two time points τ and τ^{prev} such that $\tau^{prev} < \tau$ or $\tau^{prev} = \perp$ with a undefined symbol $\perp$. This value is a summation of the time shifts induced by aligning all the MIDI events in the token T to the time point τ, if the token type is θ. Here, τ^{prev} is the time value to which the former token of T is aligned. If the type of T is not θ, then $ca(\theta, T, \tau, \tau^{prev}) = +\infty$. See Appenix A for the concrete defintion of a cost alignment function.

Let us consider a tree $t \in T(\mathcal{F}, \mathcal{K})$, a time interval I, and the associated sequence of leaves $yield(t) = \theta_0, \ldots, \theta_p$ and sequence of pairs of token and time point $tokenize(t, I) = \langle (T_0, \tau_0), \ldots, (T_p, \tau_p) \rangle$.

The *cost of alignment* of E to the tree t and the interval I, denoted by $ca_E(t, I)$, is defined as the sum of the $ca(\theta_i, T_i, \tau_i, \tau'_i)$ for $0 \leq i \leq p$ where $\tau'_i = \perp$ if $i = 0$; otherwise $\tau'_i = \tau_{i-1}$.

4.3 Transcription Objective

The problem of transcription can be defined using the above notions, by minimizing a combination of the two cost measures defined above: the cost of alignment and the cost of readability. More precisely, given an input MIDI sequence E and a grammar $\mathcal{G}$, we call transcription of E *wrt* $\mathcal{G}$, a tree $t \in \mathcal{T}(\mathcal{F}, \mathcal{K})$ minimizing the measure:

$$ca_E\big(t, [0, +\infty)\big) + cr_{\mathcal{G}}(t) \qquad (1)$$

We present in Section B an algorithm for solving the problem of computing such a tree.

The trees of Section 3.3 are abstract descriptions of music scores. The elementary score elements (symbols) correspond to the names of token types (notes, rests, chords *etc* in $\mathcal{K}$) labelling leaves. The MIDI pitch of every note can be extracted from the input MIDI sequence using the

definition of tokens in Section 3.4. A pitch-spelling algorithm [20] is then necessary to cast these MIDI key values to note names. Moreover, the durations are encoded in t by the symbols of $\mathcal{F}$ labeling inner nodes. Additionally to the operations on time intervals, some info about the output score can be attached to the symbols of $\mathcal{F}$. For instance, we can express in a symbol div_n whether we want the notes below this symbol to be beamed or not.

4.4 Algorithm and Implementation

We designed and implemented an algorithm for the parsing problem defined in Section 4 based on ordinary tabulation technique, where we used the cost alignment function described in Appendix A.

Given in input a MIDI sequence E and a weighted tree grammar, it returns a tree t minimizing (1) in Section 4.3. Figure 8 presents the parsing algorithm designed based on ordinary Dynamic Programming. It assumes that the input midi-sequence E is played with a constant tempo and converted to musical time so that one bar is identical to one second.

The algorithm deals with a set of tabulated items, which is kept by the variable C, as candidates of a parsing result. each item contains

1. a current parse tree t (or a sequence of parse trees),
2. the weight $w = ca_E(t, I) + cr_G(t)$ of the parse tree t and its allignment cost (or the weight sum of the parse trees and their allignment costs),
3. the set $E|_{snd(I)}$ of unprocessed events in the interval I of the parse tree(s), and
4. the assigned time τ^{prev} of the last token in the parse tree(s).

See appendix B for the detail.

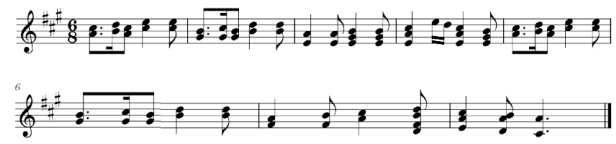
The source code of this implementation is found in URL [1] , and produces the command line utilities *monopase* for parsing monophonic/homophonic input in branch mono2 and *drumparse* for drums in branch beta. The current implementation recognizes the token type of partial continuation as an ordinary continuation.

4.5 Example

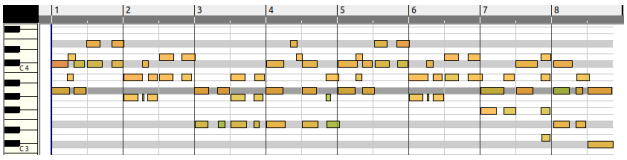
In addition to former examples of monophonic [2] and drum [3] transcription, we present an example of homophonic transcription in Figure 7. It is difficult to say that the result Figure 7(c) is a success, but it gives us valuable hints for future work.

1. There exist unnecessary printed accidentals.
2. The generation of scores in musicXML format has a bug that improperly transforms the continuations of chords. The following notes seem strange: the 2nd note in the 4th bar, the 2nd and 4th note in the 7th bar, and the 2nd note in the last bar.

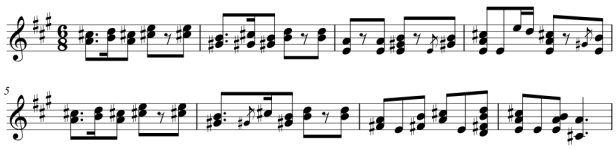
[1] https://gitlab.inria.fr/qparse/qparselib
[2] https://qparse.gitlabpages.inria.fr/docs/examples
[3] https://gitlab.inria.fr/transcription/gmdscores



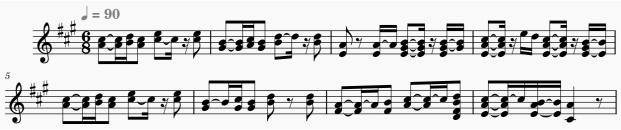
(a) Score: Sonate für Klavier Nr.11 A dur K.331 Mov.1



(b) The piano-roll of a human piano performance



(c) A transcription by our implementation



(d) A transcription by Logic

Figure 7: A transcription example

3. Some extremely short notes (see the piano roll in Figure 7(b)) are recognized as grace notes.

5. CONCLUDING REMARKS

We have proposed an approach for the tokenization of MIDI sequences which supports score elements such as rests and ornaments. Implemented as a Dynamic Programming algorithm, it is integrated into a framework of transcription by parsing.

The identification of grace-chords, as well as arpeggiated chords, shall be added to the cases of Section 2.4. Another future objective is the processing of polyphonic input, like piano MIDI files. When all voices are mixed into the same MIDI file, the application of a voice separation procedure is required. An important question is then whether voice separation should be performed before, after, or jointly to the parsing described in Section 4. Note that in the first case, voice separation would have to deal with unquantized MIDI input, and with quantized rhythms in the two latter cases.

Acknowledgments

This research was supported by JSPS Kaken 20H04302.

6. REFERENCES

[1] D. Grune and C. J. Jacobs, *Parsing Techniques*, ser. Monographs in Computer Science. Springer, 2008 2nd edition.

[2] M. Zeng, X. Tan, R. Wang, Z. Ju, T. Qin, and T.-Y. Liu, "MusicBERT: Symbolic music understanding with large-scale pre-training," in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*. Association for Computational Linguistics, 2021, pp. 791–800.

[3] Y. Ren, J. He, X. Tan, T. Qin, Z. Zhao, and T.-Y. Liu, "Popmag: Pop music accompaniment generation," in *Proceedings of the 28th ACM International Conference on Multimedia*, 2020, pp. 1198–1206.

[4] G. Hadjeres and L. Crestel, "The piano inpainting application," arXiv preprint arXiv:2107.05944, 2021.

[5] S. Oore, I. Simon, S. Dieleman, D. Eck, and K. Simonyan, "This time with feeling: Learning expressive musical performance," *Neural Computing and Applications*, vol. 32, pp. 955–967, 2020.

[6] Y.-S. Huang and Y.-H. Yang, "Pop music transformer: Beat-based modeling and generation of expressive pop piano compositions," in *Proceedings of the 28th ACM international conference on multimedia*, 2020, pp. 1180–1188.

[7] W.-Y. Hsiao, J.-Y. Liu, Y.-C. Yeh, and Y.-H. Yang, "Compound word transformer: Learning to compose full-song music over dynamic directed hypergraphs," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 1, 2021, pp. 178–186.

[8] J. Gardner, I. Simon, E. Manilow, C. Hawthorne, and J. Engel, "MT3: Multi-task multitrack music transcription," *arXiv preprint arXiv:2111.03017*, 2021.

[9] N. Fradet, J.-P. Briot, F. Chhel, A. E. F. Seghrouchni, and N. Gutowski, "Byte pair encoding for symbolic music," 2023. [Online]. Available: https://arxiv.org/abs/2301.11975

[10] J. Ens and P. Pasquier, "MMM : Exploring conditional multi-track music generation with the transformer," 2020.

[11] N. Fradet, J.-P. Briot, F. Chhel, A. El Fallah Seghrouchni, and N. Gutowski, "MidiTok: A python package for MIDI file tokenization," in *22nd International Society for Music Information Retrieval Conference (ISMIR Late-Breaking Demo)*, 2021.

[12] F. Foscarin, F. Jacquemard, P. Rigaux, and M. Sakai, "A Parse-based Framework for Coupled Rhythm Quantization and Score Structuring," in *Mathematics and Computation in Music (MCM)*, ser. Lecture Notes in Artificial Intelligence, vol. 11502. Springer, 2019.

[13] F. Jacquemard and L. Rodriguez de la nava, "Symbolic weighted language models, quantitative parsing and automated music transcription," in *26th International Conference on Implementation and Application of Automata (CIAA)*, 2022.

[14] M. Digard, F. Jacquemard, and L. Rodriguez de la Nava, "Automated transcription of electronic drumkits," in *4th International Workshop on Reading Music Systems (WoRMS)*, 2022.

[15] K. Shibata, E. Nakamura, and K. Yoshii, "Non-local musical statistics as guides for audio-to-score piano transcription," *Information Sciences*, vol. 566, pp. 262–280, 2021.

[16] M. Suzuki, "Score transformer: Transcribing quantized midi into comprehensive musical score," in *Proceediongs of the International Soc. Music Information Retrieval Conference (ISMIR, Late-breaking Demo)*, 2021.

[17] C. Agon, G. Assayag, J. Fineberg, and C. Rueda, "Kant: a critique of pure quantification," in *International Computer Music Conference Proceedings (ICMC)*, 1994, pp. 52–59.

[18] A. T. Cemgil, B. Kappen, and P. Desain, "Rhythm quantization for transcription," *Computer Music Journal*, vol. 24, no. 2, pp. 60–76, Jul. 2000. [Online]. Available: http://dx.doi.org/10.1162/014892600559218

[19] J. Yust, *Organized Time*. Oxford University Press, 2018.

[20] D. Meredith, "The PS13 pitch spelling algorithm," *Journal of New Music Research*, vol. 35, no. 2, pp. 121–159, 2006.

A. COST ALIGHNMENT FUNCTION

Let E be an input MIDI sequence. For a token type $\theta \in \mathcal{K}$ and a token T of E, let $\theta(T)$ be defined by:

$\theta(T) = 1$	if T has type θ and θ is valid in the case of input considered (see Section 2.5),
$\theta(T) = +\infty$	otherwise,

The cost alignment function used in the implementation is as follows:

$$ca(T, \theta, \tau, \tau_p) = \frac{\theta(T)}{\Delta(\tau_p, \tau)} \sum_{e \in T} \alpha_\sigma\big(r(e,T)\big) \cdot |\mathsf{ts}(e) - \tau|$$

where $\Delta(\tau^{prev}, \tau) = \tau - \tau^{prev}$ if $\tau^{prev} \neq \perp$; otherwise a fixed constant value. σ is the sign of $\mathsf{ts}(e), \tau$ and α_+ and α_- associate appropriate constants to the role $r(e,T)$ of e in T. Typically, $\alpha_{\mathsf{offL}} > \alpha_{\mathsf{offR}}$ because note-off often occurs earlier than expected.

Input: Midi-sequence E, tree grammar $\mathcal{G} = \langle \mathcal{N}, A_0, \mathcal{F}, \mathcal{K}, \mathcal{R} \rangle$.

Output: The tree t obtained from $(t, w) := parse([0, \infty), \emptyset, \bot)$.

Functions:

$fst(I)$ (resp. $snd(I)$) denotes $div_2(I, 1)$ (resp. $div_2(I, 2)$).
$E|_I = \{e \in E \mid \mathsf{ts}(e) \in I)\}$ is the subsequence of events in an interval I.

$parse(I, F, \tau)$:

If $F \cup E|_I = \emptyset$ **then Return** $\langle \varepsilon, 0 \rangle$ **else**
$C := parseRec(unit(I), A_0, F, \tau)$
$C' := \{\ \langle \mathsf{bar}(t, t'), w + w' \rangle \mid \langle t, w, F', \tau' \rangle \in C, \langle t', w' \rangle := parse(rem(I), F', \tau')\ \}$
Return $\langle t, w \rangle$ with minimum weight w among $\langle t, w \rangle \in C'$.

$parseRec(I, A, F, \tau)$:

$C := \emptyset$ and $T := F \cup E|_{fst(I)}$.
For each rule $A \xrightarrow{w} u \in \mathcal{R}$, **do**
case $u = \varepsilon$**: If** $T = \emptyset$ **then** $C := C \cup \{\langle \varepsilon, w, E|_{snd(I)}, \tau \rangle\}$
case $u = \theta \in \mathcal{K}$**:**
$C := C \cup \{\langle \theta, w + ca(\theta, T, start(I), \tau), E|_{snd(I)}, start(I) \rangle\}$
case $u = \mathsf{div}_k(A_1 \cdots A_k)$**: Let** $C_0 = \{\langle \langle \rangle, w, F, \tau \rangle\}$.
For $i = 1, \ldots, k$, **do**

$$C_i := \min'\Big(\bigcup_{\substack{\langle \langle t_1, \ldots, t_{i-1} \rangle, w_{i-1}, F_{i-1}, \tau_{i-1} \rangle \in C_{i-1} \\ \langle t_i, w_i, F_i, \tau_i \rangle \in parseRec(div_k(I, i), A_i, F_{i-1}, \tau_{i-1})}} \langle \langle t_1, \ldots, t_i \rangle, w_{i-1} + w_i, F_i, \tau_i \rangle \Big)$$

$C := C \cup \{\langle \mathsf{div}_k(t_1, \ldots, t_k), w', F', \tau' \rangle \mid \langle \langle t_1, \ldots, t_k \rangle, w', F', \tau' \rangle \in C_k\}$.
Return C

$\min'(C)$: The set of $\langle S, w, F, \tau \rangle \in C$ having minimum w for each $\langle F, \tau \rangle$.

Figure 8: Parsing algorithm

B. ALGORITHM

This section gives details of the algorithm. Each function works as follows.

parse: Assuming the following arguments:

- an interval I of midi-sequence E to be parsed,
- a set F of events in prior to $start(I)$, and
- the time point τ aligned the last parsed token before $start(I)$,

it returns a pair of the best weighted tree t and its weight w for midi-sequence $F \cup E$ with alligning all events in F to $start(I)$.

parseRec: calculates possible best trees within one bar. Assuming the following arguments:

- an interval I of midi-sequence E to be parsed,
- a non-terminal symbol A of the grammar,
- a set F of events in prior to $start(I)$, and
- the time point τ aligned the last parsed token.

it returns a set of tuples $\langle t, w, F', \tau' \rangle$ consisting of the tree t with weight w and root nonterminal symbol A for each possble pair of F' and τ' determined in the calculation.