An Algebra for Tree-Based Music Generation

Frank Drewes and Johanna Högberg

Department of Computing Science, Umeå University
S–90187 Umeå, Sweden
{drewes,johanna}@cs.umu.se

Abstract. We present an algebra whose operations act on musical pieces, and show how this algebra can be used to generate music in a tree-based fashion. Starting from input which is either generated by a regular tree grammar or provided by the user via a digital keyboard, a sequence of top-down tree transducers is applied to generate a tree over the operations provided by the music algebra. The evaluation of this tree yields the musical piece generated.

1 Introduction

The purpose of this paper is to show that certain musical structures can be generated in a grammatical manner by using a general method known as tree-based generation. This method has successfully been applied to the areas of graph and picture generation [1, 2]. A tree-based generator consists of a tree generator and a Σ-algebra, where Σ is an alphabet of abstract operation symbols. The tree generator is any formal device (e.g., a grammar), that generates a set of formal expressions over Σ (called trees). The algebra is then used to evaluate the generated trees. Thus, the type of objects being generated depends on the domain of the algebra. If it, for instance, interprets the symbols in Σ as operations on pictures, a tree-based picture generator is obtained [2].

Here, we use the same idea to generate music. To be precise, we use the term music as a shorthand for "sound structures that adhere to certain basic rules of composition". Rather than trying to imitate human composers, we are interested in finding out how and how far the formal structures found in music can be captured using the devices of formal language theory and, in particular, tree-based generators. A precursor of the present paper is [3], where the system Willow is presented. In this system, the tree generator is composed of a regular tree grammar g and a sequence $td_1, \ldots, td_n$ of top-down tree transducers (td transducers, see, e.g. [4, 5]). Intuitively, g generates a (tree representing a) coarse rhythmical structure. This tree is then passed through $td_1, \ldots, td_n$. Each of them enriches the tree so as to model a certain musical attribute, e.g. tempo, chord progression, or melodic arc. Finally, the output tree of td_n represents the musical piece generated.

Thus, Willow is tree based in the sense that it uses tree generators to generate music. However, the generated trees are interpreted in an ad hoc way rather than using a formally defined algebra. In this paper, we present such an algebra and show how it can be used to generate music in a fully tree-based manner. Moreover, we now allow the user to input themes by means of a digital keyboard.

Formally, if user input is used, it is translated into a tree that becomes the input of $td_1, \ldots, td_n$, thus replacing the initial tree generated by the regular tree grammar g. However, whereas g generates trees over a finite output signature, user input gives rise to trees containing symbols which are unknown at the time the td transducers $td_1, \ldots, td_n$ are designed. Such a situation cannot be handled by ordinary td transducers; td_1 would simply reject any tree containing an unknown symbol. In contrast, the appropriate behaviour would be to "tolerate" them, i.e. copy them to the output without doing anything. Therefore, we propose a proper generalisation of the td transducer called tolerant td transducer (ttd transducer). A ttd transducer has, as a subset of its set of states, a set of so-called tolerant states. Whenever an unknown input symbol is encountered in a tolerant state, this symbol is simply copied to the output and the computation continues on its subtrees in the same state. In all other situations, a ttd transducer behaves like an ordinary td transducer.

To the best of our knowledge, no tree-based approach (in the sense described above) to the generation of music has been proposed earlier (except for the first attempt made in [3]). However, there are some publications, often with a somewhat different focus, that propose other approaches to music generation using techniques from formal language theory.

In [6], Jurish gives a characterisation of generic musical structure within the framework of formal language theory. Tojo and Oka [7] present an analysis system for chord progressions based on head-driven phrase structure grammars. In [8], Prusinkiewicz explores the generation of musical scores by means of L-systems and the so-called turtle geometry. This approach is further developed by Worth and Stepney, who describe their search for simultaneously 'pleasing' graphical and musical renderings of languages generated by L-systems in [9]. Another device that has been used for the generation of music is the cellular automaton. A survey of this type of work is found in [10].

Related approaches can also be found in the enclosing field of algorithmic composition. Markov models appear frequently in music theory, where they are used both as a compositional device [11, 12], and for attribute classification [13, 14]. Genetic algorithms have also become increasingly popular [15, 16], as have approaches to deriving music from fractals and chaotic systems [17]. Synthesis and analysis based on neural networks and learning systems are described in [18–20] and [21]. However, most established is probably the linguistic approach. Two representatives of this line of research are Baroni [22] and Moorer [23].

The structure of this paper is as follows. In the next section, we compile the terminology around trees and tree generation required, including the new concept of tolerant td transducers. In Section 3, the music algebra is introduced. An example is discussed in Section 4, and Section 5 concludes the paper.

2 Tree Generation

Throughout this paper, we denote the natural numbers (including 0) by $\mathbb{N}$, and $\mathbb{Z}$ and $\mathbb{R}^+$ denote the integers and the positive reals, resp. The set $\mathbb{R}^+ \cup \{0\}$

of nonnegative reals is denoted by $\mathbb{R}_0^+$. We denote the powerset of a set S by $\wp(S)$ and the set of all finite strings over S by S^*; the empty string is λ. The transitive and reflexive closure of a binary relation $\to$ is denoted by $\to^*$.

Let B be the set consisting of the three special symbols '[', ']', and ','. The set of all *trees* is the smallest set $\mathcal{T}$ of strings such that, for every symbol $f \notin B$ and all trees $t_1, \ldots, t_k \in \mathcal{T}$ ($k \in \mathbb{N}$), the string $f[t_1, \ldots, t_k]$ is in $\mathcal{T}$ as well. As usual, the set of all subtrees of $f[t_1, \ldots, t_k]$ contains the tree itself and all subtrees of its *direct subtrees* $t_1, \ldots, t_k$. As a notational simplification, we identify the tree $f[]$ with f. In this sense, every single symbol is a tree. As another notational simplification, a tree of the form $g[t_1]$ (i.e., having only one direct subtree) may be denoted by $g\,t_1$.

For a tree $t = f[t_1, \ldots, t_k]$, the set $\mathrm{nod}(t)$ of *nodes of* t is the subset of $\mathbb{N}^*$ given by $\mathrm{nod}(t) = \{\lambda\} \cup \{iv \mid i \in [k] \text{ and } v \in \mathrm{nod}(t_i)\}$. For a node $u \in \mathrm{nod}(t)$, t/u denotes the subtree of t rooted at u, and $t[\![u \leftarrow s]\!]$ denotes the tree obtained from t by replacing t/u with $s \in \mathcal{T}$. Thus,

$$t/u = \begin{cases} t & \text{if } u = \lambda \\ t_i/v & \text{if } u = iv,\ i \in [k] \end{cases}$$

and

$$t[\![u \leftarrow s]\!] = \begin{cases} s & \text{if } u = \lambda \\ f[t_1, \ldots, t_{i-1}, t_i[\![v \leftarrow s]\!], t_{i+1}, \ldots, t_k] & \text{if } u = iv,\ i \in [k]. \end{cases}$$

Given a (not necessarily finite) alphabet Σ and a set T of trees, $\Sigma(T)$ denotes the set of all trees $f[t_1, \ldots, t_k]$ such that $f \in \Sigma$ and $t_1, \ldots, t_k \in T$ for some $k \in \mathbb{N}$. The set $\mathcal{T}_\Sigma(T)$ of all trees over Σ with subtrees in T is the smallest set of trees such that $T \cup \Sigma(\mathcal{T}_\Sigma(T)) \subseteq \mathcal{T}_\Sigma(T)$. The set $\mathcal{T}_\Sigma(\emptyset)$ of trees over Σ is briefly denoted by $\mathcal{T}_\Sigma$.

A *ranked alphabet* is an alphabet Σ which is given as a (not necessarily disjoint) union $\Sigma = \bigcup_{k \in \mathbb{N}} \Sigma^{(k)}$. A symbol in $\Sigma^{(k)}$ is said to be of rank k and may be denoted by $f^{(k)}$ to indicate its rank. A ranked alphabet is called finite if only finitely many of the $\Sigma^{(k)}$ are nonempty, and each of them is finite. If symbols from ranked alphabets are used to build trees, we apply the additional restriction that each occurrence of $f^{(k)}$ must have exactly k subtrees. Thus, in this case, $\Sigma(T)$ is the set of all trees $f[t_1, \ldots, t_k]$ such that $f^{(k)} \in \Sigma$ for some $k \in \mathbb{N}$, and $t_1, \ldots, t_k \in T$.

Throughout this paper, let $\mathcal{U}$ be a universe of symbols, where $\mathcal{U} \cap B = \emptyset$. The intuition behind $\mathcal{U}$ is that it is our supply of "ordinary" symbols. Symbols that do not belong to $\mathcal{U}$ have an auxiliary character. Variables provide an example of the latter: we let $X = \{x_1, x_2, \ldots\}$ be a countably infinite alphabet of pairwise distinct variables $x_i \notin \mathcal{U}$. The set $\{x_1, \ldots, x_k\}$ is denoted by X_k. In trees, variables will only appear as leaves, i.e., they will be considered as symbols of rank 0. A mapping $\sigma\colon X_k \to \mathcal{T}$ (where $k \in \mathbb{N}$) is a *substitution*. For a tree t, $t\sigma$ denotes the tree obtained from t by simultaneously replacing every occurrence of x_i ($i \in [k]$) with $\sigma(x_i)$. As a recursive definition, $x_i\sigma = \sigma(x_i)$ for all $i \in [k]$, and if $t = f[t_1, \ldots, t_l]$ with $l > 0$ or $f \notin \{x_1, \ldots, x_k\}$, then $t\sigma = f[t_1\sigma, \ldots, t_l\sigma]$.

A *term rewrite system* is a set R of rules of the form $l \to r$, where $l, r \in \mathcal{T}$. For trees s, t, there is a *rewrite step* $s \to_R t$ if there are a rule $l \to r$ in R, a node $v \in \mathrm{nod}(s) \cap \mathrm{nod}(t)$, and a substitution σ, such that $s/v = l\sigma$ and $t = s[\![v \leftarrow r\sigma]\!]$.

Regular tree grammars are defined as usual. Thus, such a grammar is a system $g = (N, \Sigma, R, S)$ consisting of a ranked alphabet $N = N^{(0)}$ of nonterminals, where $N \cap \mathcal{U} = \emptyset$, a ranked alphabet $\Sigma \subseteq \mathcal{U}$ of terminals, a set R of rules, and an initial nonterminal S. The sets N, Σ, and R are required to be finite. Every rule in R is of the form $A \to t$, where $A \in N$ and $t \in \mathcal{T}_{\mathcal{U}}(N)$. The language generated by g, called a regular tree language, is given by $L(g) = \{t \in \mathcal{T}_\Sigma \mid S \to_R^* t\}$.

As mentioned in the introduction, our general approach for generating music is adopted from [3]: a tree denoting a musical piece is generated by starting with a very simple tree and then applying to it a sequence of td transducers each of which is responsible for adding a specific musical aspect. However, the ordinary top-down tree transducer (td transducer) is not the ideal device for this purpose, because a td transducer has a finite input alphabet and cannot process input trees that contains other symbols. Instead, we would like the transformation to "step over" such symbols, at least if they are encountered in certain states. For these reasons, we now introduce the so-called tolerant td transducer.

Definition 1 (tolerant td transducer). *A* tolerant top-down tree transducer *(ttd transducer, for short) is a system* $td^{\mathrm{t}} = (\Sigma, \Sigma', Q, Q^{\mathrm{t}}, R, q_0)$*, where*

- $\Sigma, \Sigma' \subseteq \mathcal{U}$ *are finite ranked alphabets, the* input *and* output alphabets*, resp.,*
- $Q = Q^{(1)}$ *with* $Q \cap \mathcal{U} = \emptyset$ *is a ranked alphabet of* states*,*
- $Q^{\mathrm{t}} \subseteq Q$ *is the set of* tolerant states*,*
- R *is a finite set of* rules *of the form* $q\, f[x_1, \ldots, x_k] \to r$ *with* $q \in Q$*,* $f^{(k)} \in \Sigma$ *for some* $k \in \mathbb{N}$*, and* $r \in \mathcal{T}_{\Sigma'}(Q(X_k))$*, and*
- $q_0 \in Q$ *is the* initial state*.*

Given an input tree $t \in \mathcal{T}_{\mathcal{U}}$, a computation of td^{t} starts with $q_0\, t$ and applies the term rewrite rules in R until a tree in $\mathcal{T}_{\mathcal{U}}$ is reached. Whenever a symbol not in Σ is reached in a tolerant state, this symbol is simply copied to the output.

Definition 2 (computed tree transduction). *Let* $td^{\mathrm{t}} = (\Sigma, \Sigma', Q, Q^{\mathrm{t}}, R, q_0)$ *be a tolerant td transducer. For trees* $s, t \in \mathcal{T}_{\mathcal{U}}(Q(\mathcal{T}_{\mathcal{U}}))$*, there is a* computation step $s \to_{td^{\mathrm{t}}} t$ *if* $s \to_{R'} t$*, where*

$$R' = R \cup \{q\, f[x_1, \ldots, x_k] \to f[q\, x_1, \ldots, q\, x_k] \mid q \in Q^{\mathrm{t}},\ f \in \mathcal{U},\ \text{and}\ f^{(k)} \notin \Sigma\}.$$

The tree transduction computed by td^{t}*, called a* tolerant td transduction*, is the mapping* $td^{\mathrm{t}} \colon \mathcal{T}_{\mathcal{U}} \to \wp(\mathcal{T}_{\mathcal{U}})$ *such that*

$$td^{\mathrm{t}}(s) = \{t \in \mathcal{T}_{\mathcal{U}} \mid q_0\, s \to_{td^{\mathrm{t}}}^* t\}$$

for all $s \in \mathcal{T}_{\mathcal{U}}$*. For a set* $T \subseteq \mathcal{T}_{\mathcal{U}}$*, we let* $td^{\mathrm{t}}(T) = \bigcup\{td^{\mathrm{t}}(t) \mid t \in T\}$*.*

As usual, the tree transduction computed by td^{t} is considered to be a partial function $td^{\mathrm{t}} \colon \mathcal{T}_{\mathcal{U}} \to \mathcal{T}_{\mathcal{U}}$ if $|td^{\mathrm{t}}(t)| \leq 1$ for all $t \in \mathcal{T}_{\mathcal{U}}$.

Obviously, td transducers are a special case of ttd transducers, because Definition 2 reduces to the one for ordinary td transducers if $Q^{\mathrm{t}} = \emptyset$.

3 An Algebra for Music

In this section, we introduce the main contribution of the paper, the so-called music algebra. The operations of this algebra can be used to assemble a musical piece in a stepwise manner. Let us first summarise a few basics regarding music that are needed for a better understanding of the elements of the music algebra.

We identify a musical piece with the corresponding sequence of notes. In general, a note is characterised by its *tone*, *length*, *accent*, and *timbre*. The tone of a note is the ratio between its frequency and that of some fixed reference note. The length of a note is also measured relative to the length of the reference note. In music, a note whose duration is equal to that of the reference note, is called a whole note, a note whose duration is but half of that of the reference note is called a half note, and so on. Similar to tone and length, accent is a relative property; a note is accented if it is played, for example, louder than any surrounding note. The timbre of a note is the subjective quality which lets us distinguish between instruments. For the sake of simplicity, we consider only the tone and the length of a note, disregarding its accent and timbre.

A *scale* is a set of tones, and just as there are an infinite number of tones, so are there an infinite number of conceivable scales. However, only some of these are actually used in practice. The most important of these is probably the chromatic scale, which is constructed as follows: First a reference tone is ordained, which would normally be the so-called *a above middle c*, an alias for 440 Hz (for convenience, we henceforth restrict ourselves to chromatic scales built around this tone). By doubling this pitch, the tone one *octave* higher is found, and by dividing it by two, the tone one octave lower. The whole of the audible interval is split into octaves, and every octave is divided into 12 tones, in such a way that the ratio between two consecutive tones amounts to $2^{1/12}$. The tones can thus be referred to by integers: 0 refers to the reference tone, and for every tone t, $t-1$ and $t+1$ refer to the next lower and higher tone, resp., in the chromatic scale. In other words, in our setting, tone t has a frequency of $2^{t/12} \cdot 440$ Hz.

The music algebra, which we denote by $\mathbf{M}$, is a many-sorted algebra whose data domains are the sets $\mathbb{Z}$, $\mathbb{R}^+$, and $\mathcal{P}$. While $\mathcal{P}$, to be defined below, denotes the set of all musical pieces, elements of $\mathbb{Z}$ are always interpreted as tones, and elements of $\mathbb{R}^+$ are always in one way or another related to time.

Definition 3 (note and musical piece). *A* note *is a pair* $n = (t, l) \in \mathbb{Z} \times \mathbb{R}^+$ *consisting of the tone* t *of length* l*. The set of all notes is denoted by* $\mathcal{N}$*.*

The set $\mathcal{P}$ *of* musical pieces *is the set of all pairs* $P = (N, L)$ *such that*

- $N \subseteq \mathcal{N} \times \mathbb{R}_0^+$ *is a finite set of* played notes*, where, for a played note* $(n, s) \in N$*,* s *is the point in time where* n *starts to be played;*
- $L \in \mathbb{R}^+$ *is the length of the entire piece, with* $L \geq s + l$ *for all* $((t, l), s) \in N$*.*

Note that there are no pieces whose length is 0, and that a piece cannot end before its last note has been played (owing to the requirement placed on L). In the following, we may denote the components of a note n by t_n and l_n, resp. Similarly, the components of a piece P may be denoted by N_P and L_P, resp.

We now define a number of operations on musical pieces. For this, an auxiliary notation turns out to be useful. Given a set N of played notes and a mapping f on played notes, we let $N[\![(n, s) \mapsto f(n, s)]\!] = \{f(n, s) \mid (n, s) \in N\}$. Similarly, if f is a mapping on notes, we let $N[\![n \mapsto f(n)]\!] = \{(f(n), s) \mid (n, s) \in N\}$. Now, consider pieces $P = (N, L)$ and $P' = (N', L')$.

- $\textsc{length}(P) = L$ returns the length of P.
- Let $q = \max\{s + l_n \mid (n, s) \in N\}$, i.e., q is the point in time where the last notes in P end (assuming that $N \neq \emptyset$). The highest tone at the end of P is returned by

$$\textsc{highest}(P) = \begin{cases} \max\{t_n \mid (n, s) \in N \text{ and } s + l_n = q\} & \text{if } N \neq \emptyset \\ 0 & \text{otherwise.} \end{cases}$$

  (In combination with the operations defined below, this gives access to the lowest tone at the end of P and the highest and lowest tones at the beginning of P as well.)
- For every factor $a \in \mathbb{R}^+$, $\textsc{scale}(P, a) = (N[\![(n, s) \mapsto ((t_n, a \cdot l_n), a \cdot s)]\!], a \cdot L)$ scales P by the factor a.
- $\textsc{inv}(P) = (N[\![n \mapsto (-t_n, l_n)]\!], L)$ inverts all notes in P.
- $\textsc{back}(P) = (N[\![(n, s) \mapsto (n, L - s - l_n)]\!], L)$ yields P played backwards.
- $\textsc{overlay}(P, P')$ yields the overlay of P and P', given by

$$\textsc{overlay}(P, P') = (N \cup N', \max(L, L')).$$

  For the sake of convenience, we extend overlay to any number of arguments, i.e. for pieces $P_1, \ldots, P_k$ $(k \geq 1)$,

$$\textsc{overlay}(P_1, \ldots, P_k) = (\bigcup_{i \in [k]} N_{P_i}, \max_{i \in [k]} L_{P_i}).$$

- For every tone $t \in \mathbb{Z}$, $\textsc{raise}(P, t) = (N[\![n \mapsto (t_n + t, l_n)]\!], L)$ raises every tone in P by t.
- $\textsc{mute}(P) = (\emptyset, L)$ returns a silent piece having the same length as P.
- The concatenation of P and P' is given by

$$\textsc{concat}(P, P') = (N \cup N'[\![(n, s) \mapsto (n, L + s)]\!], L + L').$$

  Similarly to overlay, concat is extended to any positive number of arguments $P_1, \ldots, P_k$, i.e.

$$\textsc{concat}(P_1, \ldots, P_k) = \textsc{concat}(P_1, \cdots \textsc{concat}(P_{k-1}, P_k) \cdots).$$

– Finally, let $S \subseteq \mathbb{Z}$ be a finite nonempty set of tones. Then $\text{SNAP}_S(P)$ adjusts all tones of notes in P to the nearest tone in S. Formally, for $t \in \mathbb{Z}$, let $\Delta(t) = \min_{s \in S} |t - s|$, and let $\alpha(t) \in S$ be given by

$$\alpha(t) = \begin{cases} t + \Delta(t) \text{ if } t + \Delta(t) \in S \\ t - \Delta(t) \text{ otherwise.} \end{cases}$$

(Thus, $\alpha(t)$ is the adjusted value of t, where we select the higher tone if both $t + \Delta(t)$ and $t - \Delta(t)$ belong to S.) Now, we let

$$\text{SNAP}_S(P) = (N[\![n \mapsto (\alpha(t_n), l_n)]\!], L).$$

The music algebra $\mathbf{M}$ contains all of the operations defined above. In addition, it contains the binary operations $+$ and $-$ (addition and subtraction, resp.) on $\mathbb{Z}$, as well as the binary operations $+$, $\cdot$, max, and min (addition, multiplication, maximum, and minimum, resp.) on $\mathbb{R}^+$. For the sake of better readability, whenever the binary operations $+$, $-$, and $\cdot$ occur in trees, we use the customary infix notation. For example, a tree of the form $+[t_1, t_2]$ is written as $t_1 + t_2$.

Let $\Sigma_{\mathbf{M}}$ denote the ranked alphabet consisting of

– the operations of $\mathbf{M}$, viewed as symbols of appropriate ranks, and
– all elements of $\mathbb{Z} \cup \mathbb{R}^+ \cup \mathcal{N}$, viewed as constants, i.e. symbols of rank 0. Here, every note $n \in \mathcal{N}$ is identified with the corresponding one-note piece $(\{(n, 0)\}, l_n)$.

For a well-typed tree $t \in \mathcal{T}_\Sigma$ (where well-typedness is defined in the obvious way), we let $\text{val}_{\mathbf{M}}(t)$ denote its value, obtained by recursively evaluating subtrees and applying the operation in the root of the tree to the results.

For the sake of convenience, our implementation extends $\mathbf{M}$ in such a way that every symbol $f^{(k)} \notin \Sigma_{\mathbf{M}}$ ($k \geq 1$) is interpreted as $\text{CONCAT}^{(k)}$, i.e. as k-ary concatenation of musical pieces. In particular, for $k = 1$, f is interpreted as the identity, which is very convenient as it allows us to use such symbols as markers in the generated trees, providing information for subsequent ttd transducers, without interfering with the evaluation process. Note, however, that this property of $\mathbf{M}$ is by no means essential for the power of the approach as it is straightforward to add a ttd transducer that removes such symbols.

Given any tree generator Γ, the pair $G = (\Gamma, \mathbf{M})$ is called a tree-based music generator. Here, a *tree generator* is any device Γ that defines a tree language $L(\Gamma) \subseteq \mathcal{T}_{\mathcal{U}}$; the set of musical pieces generated by G is then

$$L(G) = \{\text{val}_{\mathbf{M}}(t) \mid t \in L(\Gamma)\}.$$

In the example discussed in the next section, the tree generator Γ is composed of a regular tree grammar g and ttd transducers $td_1^{\text{t}}, \ldots, td_k^{\text{t}}$. In this case, we define $L(\Gamma) = td_k^{\text{t}}(\cdots td_1^{\text{t}}(L(g)) \cdots)$.

4 Variations and Canons

This section describes how simple *variations*[1] and *canons* can be generated. In a variation, the subject is introduced together with an answer – an imitation of the subject – and possibly a countersubject – a substantive figure that is meant to sound well when played parallel to the subject. If there are several voices available, then the subject appears at some point in all of them. The piece concludes after the subject (or answer) has appeared in the last voice. The subject can be explored and developed by performing it in *inversion* (upside-down), *retrograde* (back-to-front), *diminution* (with shorter note values) or *augmentation* (with longer note values). The subject can also appear in *stretto*, meaning that it is played as a canon, or as a *false entry*, in that it is fractioned or incomplete.

In our implementation, the subject is either generated by the regular tree grammar SUBJECT, or derived from a midi file. A suitable subject is a short sequence of notes that is easy to recognise, contained within an octave, and has a relatively simple rhythm. We do not make any assumptions about the time measure, that is to say, the input completely determines the rhythm of the generated piece, and is not modified to fit a standard meter. Since, in our algebra, the length of a note can be any positive real number, this is not a problem. However, as a consequence (mentioned in the introduction), our tree transducers must be able to tolerate input symbols that are not known a priori.

Before explaining how the actual generation process works, let us discuss a detail that illustrates the usefulness of ttd transducers. Suppose we want to use a subject provided by the user. In a first step, we turn the corresponding midi file into a tree t_{init} that evaluates to the subject. The symbols in this tree are $\text{CONCAT}^{(2)}$, $\text{OVERLAY}^{(2)}$, and $\text{MUTE}^{(1)}$ as internal nodes, and an unknown set of notes as leaves. Some of the subsequent ttd transducers work by descending down the input tree until a note is reached, which is then modified by applying some operation. However, there is a problem with this. Once a computation has reached such a note, the only thing that can be done is to tolerate the symbol, because it is unknown (unless the whole computation is aborted). Therefore, we need a preprocessing step that replaces every note n in t_{init} by $\text{NOTE}[n]$. Thus, NOTE is a marker signifying that a leaf will be reached in the next step. Interestingly, the preprocessing can be implemented by a ttd transducer. To see this, let $\text{PRE} = (\Sigma, \Sigma', \{q, q'\}, \{q'\}, R, q)$, where $\Sigma = \{\text{CONCAT}^{(2)}, \text{OVERLAY}^{(2)}, \text{MUTE}^{(1)}\}$, $\Sigma' = \Sigma \cup \{\text{NOTE}^{(1)}\}$, and R is

$$\{qf[x_1, \ldots, x_k] \to f[t_1, \ldots, t_k] \mid f^{(k)} \in \Sigma \text{ and } t_i \in \{q[x_i], \text{NOTE}[q'[x_i]]\} \text{ for all } i \in [k]\}.$$

If we disregard the case where t_{init} is a single note, there is exactly one successful computation for each input tree t_{init}. It descends down the tree in state q, guessing nondeterministically when it has reached a note. At that point, it adds the required occurrence of NOTE and continues in state q'. Since there are no rules at all for q', an incorrect guess means that the computation will fail. This also happens if a note is reached in state q, because q is not tolerant.

[1] We use this term to denote a toy fugue.

The modified subject t_{subj} obtained in this way is given as input to the ttd transducer THEMES that derives from it an answer, a countersubject, and an accompaniment. The answer is the inverted subject transposed by a fifth, so t_{ans} is given by $\text{INV}[\text{RAISE}[t_{\text{subj}}, 7]]$. The countersubject is the subject played in retrograde, i.e. t_{csubj} equals $\text{BACK}[t_{\text{subj}}]$. The accompaniment is obtained from the subject by lengthening some sections at the others' expense. The output from THEMES is the tree $\text{CONCAT}[t_{\text{subj}}, t_{\text{ans}}, t_{\text{csubj}}, t_{\text{acc}}]$.

Let us have a closer look at the declaration of THEMES in TREEBAG, which reads like this:

```
generators.ttdTransducer("Themes"):
(  {SUBJECT : 1, CONCAT : 2},
   {#include(../signature)},
   {S, P, A},
   {P, A},
   {  S[SUBJECT[x1]] → CONCAT[SUBJECT[P[x1]],
                              ANSWER[RAISE[INV[P[x1]], 7]],
                              CSUBJECT[BACK[P[x1]]],
                              ACCOMPANIMENT[A[x1]]],
      P[CONCAT[x1, x2]] → CONCAT[P[x1], P[x2]],
      A[CONCAT[x1, x2]] → CONCAT[A[x1], A[x2]] weight 2,
      A[CONCAT[x1, x2]] → SCALE[P[x1], 1 + LENGTH[P[x2]]/LENGTH[P[x1]]] },
   S ).
```

The first four components are: the input signature, the output signature, the set of states, and the set of tolerant states. We see that both P and A (which stands for *pass* and *accompaniment*, respectively) are tolerant whereas S is not. The fifth component is the set of rewrite rules, and since this particular transducer is only concerned with two input symbols, the set is quite small. The first rule turns the subject into three themes and an accompaniment. As mentioned earlier, we want to lengthen some of the notes in the accompaniment, but not all of them. When, during the generation of the accompaniment, a node of rank two labelled CONCAT is come across, we find that there are two applicable rules: one which leaves the local configuration untouched and simply proceeds downwards, and one which lengthens the notes found in the first subtree and discards those in the second. Using a feature of TREEBAG that allows to add weights to the rules, the first of these can be made twice as likely for application as the second. This decreases the risk of ending up with an accompaniment that is but a single long note. The sixth and last component is the initial state, in this case S.

SOPRANO			subject	countersubject
ALTO		answer	countersubject	
TENOR	subject	countersubject		
BARITONE				answer
BASS		accompaniment	accompaniment	accompaniment

Fig. 1. One possibility to weave a musical piece around a theme.

The themes thus produced are then arranged by the ttd transducer EXPOSITION over the five voices, beginning as in Figure 1. The output tree t_{exp} is of the form $\text{CONCAT}[t_1, t_2, \ldots, t_n]$, where $t_i = \text{OVERLAY}[t_i^{\text{bas}}, t_i^{\text{bar}}, t_i^{\text{ten}}, t_i^{\text{alt}}, t_i^{\text{sop}}]$ for every $i \in [n]$. The abbreviations bas, bar, ten, alt, and sop stand for bass, baritone, tenor, alto, and soprano, respectively.

If t_{exp} is interpreted as a piece of music, i.e. $\text{val}_{\mathbf{M}}(t_{\text{exp}})$ is played, then it is very likely to contain dissonances, as the theme is played against itself both in retrograde and inversion. To clear these, and to add a sense of movement, we wish to label the tree with chords in such a way that when the assigned chords are read left-to-right, they appear in accordance with some common chord progression. If this progression can be expressed as a directed graph G, in which the individual chords are the nodes, then the labelling can be done by the ttd transducer PROGRESSION that operates along the following principle.

The states of the ttd transducer are tuples of the form $\langle c, c' \rangle$, where c and c' are chords. We choose $\langle c_s, c_e \rangle$ as the initial state if we wish the progression to start with c_s and end with c_e. The rules R of PROGRESSION can be divided into two types of rules, which develop and settle the progression, respectively. A rule

$$\langle c, c' \rangle[\text{CONCAT}[x_1, x_2]] \to \text{CONCAT}[\langle c, c'' \rangle[x_1], \langle c'', c' \rangle[x_2]]$$

is in R if there is a path from c to c' in G that passes through c''. A rule

$$\langle c, c' \rangle[\text{CONCAT}[x_1, x_2]] \to \text{SNAP}_{\hat{c}}[\text{CONCAT}[\mathsf{P}[x_1], \mathsf{P}[x_2]]]$$

is in R if the distance from c to c' in G is less or equal to one. Here, P is an auxiliary state that simply copies the subtree below it to output, and the only state that is tolerant with respect to OVERLAY, CONCAT and NOTE. Since the refinement of the progression cannot proceed below these symbols, we know that each chord is represented by at least one note, and that no two chords are played in parallel. If every state was tolerant, then this could not be attained.

In the second rule above, $\hat{c}$ is the closure of the notes in c under transposition by an octave.[2] This assures that the local notes belong to $\hat{c}$, and that the complete note sequence respects the chosen chord progression. For a more detailed discussion of this subject, see [3].

When we generate a canon, we begin as we did for variations: a subject is either derived by a regular tree grammar, or extracted from midi data. Copies of this subject are then arranged over four voices by the ttd transducer CANON, and this is done in such a way that there are frequent overlaps and many false entries. Because of the overlaps, it is now easier to generate one voice at a time, and then combine them using OVERLAY, rather than generating one measure at a time, and then concatenating the results. This can be done using an extended version of the rule

[2] In fact, strictly speaking, we cannot use $\hat{c}$ because it is infinite. Therefore, it is replaced with its restriction to the audible range. This has the additional advantage that no tones outside this range will be generated.

$$\mathsf{S}[\textsc{subject}[x_1]] \to \textsc{overlay}[\textsc{raise}[\textsc{concat}[\textsc{mute}[\mathsf{H}[x_1]], \mathsf{P}[x_1], \textsc{mute}[\mathsf{H}[x_1]], \textsc{mute}[\mathsf{T}[x_1]]], 12], \textsc{raise}[\textsc{concat}[\mathsf{P}[x_1], \mathsf{H}[x_1], \mathsf{P}[x_1]], 24], \textsc{raise}[\textsc{concat}[\textsc{mute}[\mathsf{H}[x_1]], \textsc{mute}[\mathsf{H}[x_1]], \mathsf{P}[x_1], \textsc{mute}[\mathsf{T}[x_1]]], 36]].$$

The two states H and T select the first and the second half, respectively, of the subtree below them. For this approach to yield a nice result, the tree t_{subj} must not be comb-like.

To make the generated piece more interesting to listen to, we end the generation process by adding various ornaments. An *ornament* is a musical embellishment that is not part of the overall melody, but rather an added decoration. An example is the mordent: a single rapid alteration between a note of the melodic line, and the note immediately above it. The ornaments are added by the ttd transducer Ornament, which forms a mordent using the rule

$$\mathsf{S}[\textsc{note}[x_1]] \to \textsc{scale}[\textsc{concat}[\mathsf{S}[x_1], \textsc{raise}[\mathsf{S}[x_1], 1], \mathsf{S}[x_1]], \tfrac{1}{3} \cdot \textsc{length}[\mathsf{S}[x_1]]].$$

The generated tree is interpreted by the algebra described in Section 3 as a piece of music, which can then be performed using the jMusic library [24].

5 Conclusion

In this paper, we have continued the work that was started in [3]. In particular, we have presented an algebra for the tree-based generation of music. The motivation behind this work is to investigate how far typical structures that appear in musical pieces can be captured using the limited means of the tree-based formalism, as opposed to using Turing-complete formalisms for, e.g., imitating a particular human composer. Naturally, the discussion of the example in the previous section could not reveal much detail without becoming lengthy and repeating much of what has been said in [3]. Readers who want to explore the details are invited to download the system and the example from http://www.cs.umu.se/~johanna/algebra.

Clearly, more (and more sophisticated) examples are needed in order to understand whether the operations of the algebra proposed in this paper are really appropriate. In fact, it will probably turn out that different types of music require different algebras and maybe also different types of tree generators. This situation is well known in the area of picture generation, where each choice consisting of a class of tree generators and a class of picture algebras results in a specific type of picture generator.

Let us finally point out that the concept of tolerant td transducers may be of independent interest, as it may be useful in other applications in which unknown symbols can occur. Thus, it could be worthwhile to study the theoretical properties of ttd transducers by, e.g., comparing them with ordinary td transducers.

Acknowledgement We thank Albert Graef and Carl Rehnberg for providing us with references regarding computer-generated music.

References

1. Engelfriet, J.: Graph grammars and tree transducers. In Tison, S., ed.: Proc. CAAP 94. Volume 787 of LNCS. Springer (1994) 15–37
2. Drewes, F.: Grammatical Picture Generation – A Tree-Based Approach. Texts in Theoretical Computer Science. An EATCS Series. Springer (2006)
3. Högberg, J.: Wind in the willows – generating music by means of tree transducers. In Farré, J., Litovski, I., Schmitz, S., eds.: Proc. 19th Conf. on Implementations and Applications of Automata. Volume 3845 of LNCS., Springer (2005) 153–162
4. Gécseg, F., Steinby, M.: Tree languages. In Rozenberg, G., Salomaa, A., eds.: Handbook of Formal Languages. Vol. 3: Beyond Words. Springer (1997) 1–68
5. Fülöp, Z., Vogler, H.: Syntax-Directed Semantics: Formal Models Based on Tree Transducers. Monographs in Theoretical Computer Science. An EATCS Series. Springer (1998)
6. Jurish, B.: Music as a formal language. In Zimmer, F., ed.: Bang | Pure Data. Wolke Verlag (2006)
7. Tojo, S., Oka, Y., Nishida, M.: Analysis of chord progression by HPSG. In: AIA'06: Proc. 24th IASTED international conference on Artificial intelligence and applications, ACTA Press (2006) 305–310
8. Prusinkiewicz, P.: Score generation with L-systems. In Berg, P., ed.: Proc. ICMC. Volume 1., Royal Conservatory, The Hague, Netherlands (1986) 455–457
9. Worth, P., Stepney, S.: Growing music: Musical interpretations of L-systems. In: EvoWorkshops. (2005) 545–550
10. Burraston, D., Edmonds, E., Livingstone, D., Miranda, E.R.: Cellular automata in midi based computer music (2004)
11. Ames, C.: The Markov process as a compositional model: A survey and tutorial. Leonardo **22**(2) (1989) 175–187
12. Visell, Y.: Spontaneous organisation, pattern models, and music. Organised Sound (2004)
13. Shao, X., Xu, C., Kankanhalli, M.S.: Unsupervised classification of music genre using hidden Markov model. In: ICME. (2004) 2023–2026
14. Chai, W., Vercoe, B.: Folk music classification using hidden Markov models. In: Proc. Int. Conference on Artificial Intelligence. (2001)
15. Horner, A., Goldberg, D.E.: Genetic algorithms and computer-assisted music composition. In: Proc. Fourth Int. Conference on Genetic Algorithms, San Diego, CA. (1991) 437–441
16. Jacob, B.: Composing with genetic algorithms. In: Proc. ICMC. (1995) 452–455
17. Chapel, R.H.: Realtime Algorithmic Music Systems from Fractals and Chaotic Functions: Toward an Active Musical Instrument. PhD thesis, Univ. Pompeu Fabra, Barcelona (2003)
18. Bharucha, J.: Neural net modeling of music. In: Proc. First Workshop on Artificial Intelligence and Music, American Association for AI (1988) 173–182
19. Bresin, R., De Poli, G., Vidolin, A.: A neural networks based system for automatic performance of musical scores. In: Proc. 1993 Stockholm Music Acoustic Conference, Royal Swedish Academy of Music, Stockholm (1994) 74–78
20. Todd, P.: A connectionist approach to algorithmic composition. Computer Music Journal **13**(4) (1991) 27–43
21. Mozer, M.: Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints .... Connection-Science **6**(2) (1994) 247–280
22. Baroni, M.: The concept of musical grammar. Music Analysis **2**(2) (1983) 175–208
23. Moorer, J.A.: Music and computer composition. Commun. ACM **15**(2) (1972) 104–113
24. Sorensen, A., Brown, A.: Introduction to jMusic. Internet resource (2007) Available at `http://jmusic.ci.qut.edu.au/`. Accessed 27 Feb 2007.