JAIST Repository
https://dspace.jaist.ac.jp/

Title	Melodic Morphing Algorithm in Formalism
Author(s)	Hirata, Keiji; Tojo, Satoshi; Hamanaka, Masatoshi
Citation	Lecture Notes in Computer Science, 6726/2011: 338-341
Issue Date	2011-06-18
Type	Journal Article
Text version	author
URL	http://hdl.handle.net/10119/10301
Rights	This is the author-created version of Springer, Keiji Hirata, Satoshi Tojo and Masatoshi Hamanaka, Lecture Notes in Computer Science, 6726/2011, 2011, 338-341. The original publication is available at www.springerlink.com, http://dx.doi.org/10.1007/978-3-642-21590-2_28
Description	

Japan Advanced Institute of Science and Technology

Melodic Morphing Algorithm in Formalism

Keiji Hirata[†] Satoshi Tojo[‡] Masatoshi Hamanaka[§]

[†] Nippon Telegraph and Telephone Corporation
[‡] Japan Advanced Institute of Science and Technology
[§] University of Tsukuba

Abstract. Up to now, experiments run with many subjects are almost the only way for demonstrating the validity of a musical algorithm. In contrast, we employ formalism to verify that a melodic morphing algorithm correctly works as specified. The way we present in the paper is formal, theoretical, and viable. We provide a full-fledged feature structure for representing a time-span tree of "A Generative Theory of Tonal Music" (GTTM). The key idea here is that the GTTM reduction is identified with the subsumption or the "is-a" relation, which is the most fundamental relation in knowledge representation. Since we obtain the domain in which a partial order is defined, we introduce the *join* and *meet* operations based on unification. In such an algebraic framework, as preliminaries of the proof of a melodic morphing algorithm, we introduce the following notions: reduction path, melodic complexity, similarity, and interpolation. These notions are defined using the subsumption relation, *join* and *meet*, in a formal way. Then we take the melodic morphing algorithm proposed by Hamanaka et al. (2008) because it is constructed using the *join* and *meet* operations. Finally we prove the theorem that the melodies generated by the algorithm are the interpolations of two given melodies.

Keywords: GTTM, time-span reduction, feature structure, subsumption, join, meet, renderability, similarity, interpolation, formalism

1 Introduction

Today, many people including both professionals and non-professionals enjoy creating musical pieces or adapting existing works to construct new ones. Such created music is circulated within a creator-consumer community. These works are well-known as user generated content (UGC) or consumer generated media (CGM). For those involved in UGC and CGM, however, it remains still challenging to properly express their thoughts and emotions through musical works or melodies. To create music, we describe it in a language, format, or structure with supporting tools. We think the main issues we face are the trade-off between descriptive power and simplicity[1] and symbol grounding.

[1] Besides descriptive power vs simplicity, we may also say expressiveness vs tractability [8] or generality vs efficiency [2].

Firstly, since descriptive power and simplicity are not compatible, creators have to cope with the trade-off between them. Descriptive power is the capability of a language and tools to precisely express creator's thoughts and emotions, or the degree of reproducing original music from the description in a language. Simplicity may be measured by the description length of the music. In general, the more abstract a description is, the shorter it is, and the less the product of writing and reading costs is. For instance, a Standard MIDI File has high descriptive power yet low simplicity, while the chord symbol is the opposite. The trade-off can also be considered a problem of controllability in expressing music. We argue that the key for being compatible is separating the basic operations whose meanings are well-understood from one's intention for realizing a complicated target task to combine the basic operations[2]. The basic operations include the ones like addition, subtraction, intersection, and union. We are led to an algebraic framework, in which a creator assembles basic operations into a calculation process for a target task as a creator intends.

Secondly, in the supporting tools for UGC and CGM, no matter how carefully and logically algorithms are designed and implemented, mostly the only way for verifying that they have been correctly designed and implemented is running experiments with many subjects[3]. It is mainly because of incomplete, inconsistent symbol grounding [5]. We argue that we rely on a solid music theory to establish the firm symbol grounding so that the symbols and relations are properly grounded onto the real world objects and relevancies.

The aim of the paper is exploiting the power of the algebraic framework in music. Thus, we employ formalism to verify that a complicated musical task correctly works as specified by proving the theorem of the task, not running experiments. In the paper, for symbol grounding, we equate the subsumption or an "is-a" relation of a feature structure with the reduction process of a time-span tree in music theory "A Generative Theory of Tonal Music" (GTTM). Since we can assume a collection of melodies, for any two of which a partial order is defined, we construct a domain of melodies, on which *join* (union) and *meet* (intersection) operations are defined, that is, the domain makes a lattice in terms of the GTTM reduction. Then we pay attention to morphing as a target task, which takes two different melodies and generates an inbetween melody. The morphing algorithm we take here is described by using the subsumption relation and the *join* and *meet* operations in the algebraic framework, and we formally prove that a morphed melody is properly located in the lattice.

We briefly mention the related work concerning music representation method. Marsden proposed a representational framework for polyphony, employing not only ternary but also n-ary ($n \geq 4$) relations for an elaboration vocabulary [9]. Marsden began with conventional tree representations and allowed joining of branches in the limited circumstances with preserving the directed acyclic

[2] This parallels Norman's simplicity design axiom: "the complexity of the information appliance is that of the task, not the tool. The technology is invisible." [10]

[3] Wiggins and Smaill call the people who participate in experiments "wetware" in the extreme [14].

graph (DAG) property for expressing information dependency. As a result, high expressiveness was achieved, while it was difficult to define consistent similarity between melodies. Also, to correctly interpret the various relations, an ontology for the vocabulary was required.

Valero proposed a representation method dedicated to a similarity comparison task, called metrical tree [13]. Valero used a binary tree representing the metrical hierarchy of music and avoided the necessity of explicitly encoding onsets and duration; only pitches need to be encoded. For ease of a similarity comparison task, Valero also gave several procedures propagating pitch labels from the leaves upward to label the internal nodes in a metrical tree. As a measure to compare metrical trees, Valero basically adopted the tree edit distance. Metrical tree was intuitive, easy to automatize, and had a wide range of applicability. When Valero ran evaluation experiments, he however needed to set up many parameters, such as the cost for each edit operation, the label-propagation procedure, and the pruning level for controlling the metrical resolution. The parameter setup was justified by the best performance in experiments, not from the music theory point of view.

2 Representing Time-Span Tree in Feature Structure

A melody is considered a sequence of pitch events in temporal order[4]; pitch events include a single note and a chord. Time-span reduction assigns to the pitch events of the piece a hierarchy of structural importance with respect to their position in grouping and metrical structure [7]. The structural importance is derived from the pitch and temporal proximities between the pitch events and the regular alternation patterns of strong and weak beats. Thus, a time-span tree expressing the hierarchy of structural importance is a binary tree constructed in bottom-up and top-down manners by comparing the structural importances of adjacent pitch events at a number of hierarchical levels.

2.1 Time-Span Tree and Reduction

The music theory called “A Generative Theory of Tonal Music” (GTTM), proposed by Lerdahl and Jackendoff in 1983 [7], is one of the few music theories that involve the concept of reduction. Reduction, in general, connects an original element and its ornamented one (complicated one).

Figure 1 shows an excerpt from Lerdahl and Jackendoff’s GTTM book and demonstrates the reduction concept. The proper way to understand Figure 1 is to hear the successive levels in the same tempo[5]. If the reduction is done

[4] Following Lerdahl and Jackendoff [7], a melody means a homophony in the paper.

[5] Once a melody is reduced, each note with onset and duration properties becomes a virtual note that is just a pitch event dominating a corresponding time-span, omitting onset and duration. Therefore, to listen to a reduced melody, we assume that it can be rendered by regarding a time-span as a real note with such onset timing and length.

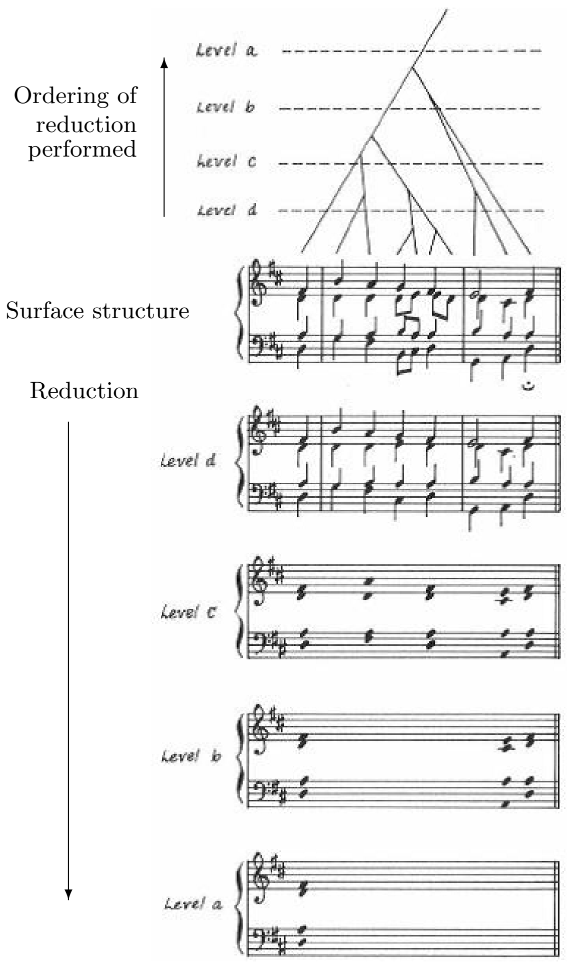


Fig. 1. Time-span reduction in GTTM (Lerdahl and Jackendoff [7, page 115])

satisfactorily, each level should sound like a natural simplification of the previous level. The alternative omission of notes must make the successive levels sound less like the original. Hence, reduction can be regarded as rewriting an expression to an equivalent, simpler one; it often has the same meaning as abstraction. Since GTTM reduction is designed based on Gestalt grouping, the reduction successfully associates a melody with another one that sounds quite similar.

We argue that the proper way of taking into account deep musical structures is to adopt the concept of reduction [12]. The key idea of our framework is that GTTM reduction is identified with the subsumption or the "is-a" relation, which is the most fundamental relation in knowledge representation. For example, in Figure 1, Level *d* melody is reduced to Level *c* melody, and similarly *c* to *b* to *a*. For another example, consider pitch C5 and portcullis C. Naturally, we say that "the C5 note is reduced to a C note," which can be interpreted as "a C5 note is a C" as well, because by instantiating pitchclass C, we can obtain instances of C4, C5, C6, and so on.

2.2 Feature Structure and Subsumption Relation

Feature structure has been mainly studied for applications to linguistic formalism based on unification and constraint, such as Head-driven Phrase Structure Grammar (HPSG)[11]. Feature structure can be considered a natural generalization of first-order terms to represent partial information and class hierarchy [1].

Definition 1 (Feature structure). *A feature structure is a list of feature-value pairs where a value may be replaced by another feature structure recursively.*

Below is a feature structure in attribute-value matrix (AVM) notation where capital σ_i is a structure, lower-case f_i is a feature label, and v_i is its value:

$$\sigma_1 = \begin{bmatrix} f_1 \begin{bmatrix} f_3 \; v_3 \\ f_4 \; v_4 \end{bmatrix} \\ f_2 \; \sigma_7 \end{bmatrix} \quad .$$

Such a structure is also called a labeled directed acyclic graph (DAG), and needless to say, we can regard the structure as equivalent to a tree, where each node corresponds to a feature and a leaf to a value.

When feature structure σ properly includes the other from the same top node or is the hyper-structure of the other, σ subsumes the other.

Definition 2 (Subsumption Relation). *Let σ_1 and σ_2 be feature structures. We define subsumption relations by the following rules:*

$$\begin{array}{lll} \sigma_1 \sqsubseteq_H \sigma_2 & \leftarrow & \forall(f \;\; v_1) \in \sigma_1 \; \exists(f \;\; v_2) \in \sigma_2 \quad v_1 \sqsubseteq_H v_2 \\ \sigma_1 \sqsubseteq_S \sigma_2 & \leftarrow & \forall(f \;\; v_2) \in \sigma_2 \; \exists(f \;\; v_1) \in \sigma_1 \quad v_1 \sqsubseteq_S v_2 \quad . \end{array}$$

Here, $\sqsubseteq_H$ stands for the so-called Hoare order of sets, and $\sqsubseteq_S$ Smyth order. Hoare order is appropriate, if there are conjunctive relations between the elements of a set, and the Smyth order if disjunctive. For instance, we have $\{b, d\} \sqsubseteq_H \{a, b, c, d\}$ and $\{a, b, c, d\} \sqsubseteq_S \{b, d\}$. Since a feature structure is considered the conjunctive set of feature-value pairs, we adopt Hoare order and use notation '$\sqsubseteq$' hereafter.

For example, by assuming $v_4 \sqsubseteq [f_5 \;\; v_5]$, σ_1 is subsumed both by the following σ_2 and σ_3, and we write $\sigma_1 \sqsubseteq \sigma_2$ and $\sigma_1 \sqsubseteq \sigma_3$:

$$\sigma_2 = \begin{bmatrix} f_1 \begin{bmatrix} f_3 \; v_3 \\ f_4 \; [f_5 \;\; v_5] \end{bmatrix} \\ f_2 \; \sigma_7 \end{bmatrix} , \qquad \sigma_3 = \begin{bmatrix} f_1 \begin{bmatrix} f_3 \; v_3 \\ f_4 \; v_4 \end{bmatrix} \\ f_2 \; \sigma_7 \\ f_6 \; v_6 \end{bmatrix} \quad .$$

For the above example, although both σ_2 and σ_3 are elaborations of σ_1, the two hyper-structures are differently elaborated. Hence, ordering $\sqsubseteq$ is a partial order, not a total order. unlike integers and real numbers. Equivalence $a = b$ is defined as $a \sqsubseteq b \wedge b \sqsubseteq a$.

To denote value v of feature f in structure σ, we write $\sigma.f = v$. Thus, $\sigma_1.f_2 = \sigma_7$, and $\sigma_1.f_4$ is undefined. $\sigma_1.f_1.f_4 = v_4$. As long as no ambiguity is present, we omit long access $f_1.\cdots.f_n$ as $f_1..f_n$, e.g., $\sigma_2.f_1..f_5 = v_5$.

2.3 Time-Span Trees in Feature Structures

We design the feature structure for representing a time-span tree so that the reduction relation between the time-span trees is properly mapped to the subsumption relation between feature structures '$\sqsubseteq$'. In Figure 2, we show the feature structures representing the node of a time-span tree and a pitch event. A

$$\begin{bmatrix} {}^{\sim}\mathit{tree} & & \\ \mathit{head}\ \{\{\boxed{\mathrm{i}},\boxed{\mathrm{j}}\}, [{}^{\sim}\mathit{event}]\} & & \\ \mathit{dtrs}\ \{ \begin{bmatrix} \mathit{left} & \begin{bmatrix} {}^{\sim}\mathit{tree} \\ \mathit{head}\ \begin{bmatrix}\boxed{\mathrm{i}}\ {}^{\sim}\mathit{event}\end{bmatrix} \\ \mathit{dtrs}\ \{\cdots\} \end{bmatrix} \\ \mathit{right} & \begin{bmatrix} {}^{\sim}\mathit{tree} \\ \mathit{head}\ \begin{bmatrix}\boxed{\mathrm{j}}\ {}^{\sim}\mathit{event}\end{bmatrix} \\ \mathit{dtrs}\ \{\cdots\} \end{bmatrix} \end{bmatrix} , \bot\} & & \end{bmatrix}$$

$$\begin{bmatrix} {}^{\sim}\mathit{event} & \\ \mathit{pitch} & \mathit{Pitch} \\ \mathit{pos} & \begin{bmatrix} \mathit{bar} & \mathit{Integer} \\ \mathit{meter} & \mathit{Length} \end{bmatrix} \\ \mathit{duration} & \mathit{Length} \end{bmatrix}$$

Fig. 2. Feature structures for representing a time-span tree (upper) and a pitch event (lower)

feature structure is given a *type*, which is shown headed by '˜' (tilde) and is declared at the beginning of feature-value pairs, The set notation $\{x, y\}$ means the choice either of x or y, and $\bot$ is empty. Let σ be a feature structure of type ˜*tree*. A feature structure of $\sigma.dtrs = \bot$ corresponds to a leaf of a time-span tree, and at the same time, $\sigma.head$ is a pitch event. Here, *dtrs* means daughters.

Structure sharing is indicated by tags such as $\boxed{\mathrm{i}}$ or $\boxed{\mathrm{j}}$. The feature structures of type ˜*event* occur at positions $\sigma.dtrs.left.head$ and $\sigma.dtrs.right.head$. By tags $\boxed{\mathrm{i}}$ and $\boxed{\mathrm{j}}$, value $\sigma.head$ is assigned to $\sigma.dtrs.left.head$ and $\sigma.dtrs.right.head$, respectively. If $\sigma.head = \sigma.dtrs.left.head$, the node has the right-hand elaboration of shape $\wedge$, and if $\sigma.head = \sigma.dtrs.right.head$, the left-hand elaboration

⋀ . As such, the *head* value at the parent level is recursively taken from either the left-hand or right-hand branch.

Within type ˜*event*, instances of type *Pitch* include $C4$, $B\flat 6$, $F\sharp 3$, and so on. Feature *pos* stands for the onset timing, and its value is of the form n-th bar from the beginning and the m-th beat position within a bar. In our paper, we assume that the duration of a quarter note is a temporal unit, which is of type *Length*, and feature *duration* also has a *Length* value. Using the representation method and the subsumption relation, in Figure 1, we can successfully write Level c melody $\sqsubseteq$ Level d melody, Level b melody $\sqsubseteq$ Level c melody, and Level a melody $\sqsubseteq$ Level b melody.

2.4 Full-Fledged Feature Structure

The reduction process in Figure 1 is linear (total order) with three steps in which every step is taken to remove several notes at one time. On the other hand, we merely remove a note or a feature-value pair to obtain a reduced feature structure. For removing a note, the reduction step becomes finer, and in general, more than one note can be removed at a time. For removing a feature-value pair, the reduction step becomes even finer, and a feature structure may represent either more or less information than is contained in a feature structure, just enough to be reproduced from the feature structure to a real melody (rendering). Now, we need the notion of a full-fledged feature structure.

Definition 3 (Full-Fledged Feature Structure). *There are intrinsic features to define a typed feature structure. When all the intrinsic features accompany a typed feature structure, the feature structure is said to be full-fledged.*

If there are missing features for a typed feature structure, those features are regarded to be valued $\perp$.

For example, ˜*tree* type requires the feature set of *head*, $dtrs.left$, and $dtrs.right$, and ˜*event* type *pitch*, *pos.bar*, *pos.meter*, and *duration*.

3 Calculus in Melody Lattice

We denote melody A as a time-span tree in feature structure T_A. Although more than one time-span tree may correspond to a single melody in general, we identify feature structure T_A with melody A as long as there is no ambiguity.

3.1 Unification, *Join* and *Meet*

Intuitively, unification is a process of information conjunction. To formalize it, we consider the sequence of features that we introduced for accessing the value of the feature above $f_1.f_2.\cdots.f_n$ called a feature path. Then we introduce the set notation of a feature structure using the set of feature-path-value pairs $\{f_1.\cdots.f_n\ v\}$. Notice that the set notation of the feature structure is equivalent to Definition 1. A feature structure is consistent, if the unique-value restriction on the features is satisfied.

Definition 4 (Unification). *Unification is the consistent union of feature structures in the set notation. Unification only fails when it is applied to feature structures that, when taken together, provide inconsistent information.*

For the domain in which a partial order is defined, *join* and *meet* are well-known as basic operations. *Join* corresponds to a union of sets or a consistent overlay, and *meet* corresponds to intersection or the common part.

Definition 5 (*Join*). *Let A and B be full-fledged feature structures representing the time-span trees of melodies A and B, respectively. If we can fix the least upper bound of A and B, that is, the least y such that $A \sqsubseteq y$ and $B \sqsubseteq y$ is unique, we call such y the join of A and B, denoted as $A \sqcup B$.*

Theorem 3.13 in Carpenter [1] provides that the unification of feature structures A and B is the least upper bound of A and B. Therefore, we adopt unification as *join* in the paper.

For the previous examples, we have

$$\sigma_2 \sqcup \sigma_3 = \begin{bmatrix} f_1 \begin{bmatrix} f_3\ v_3 \\ f_4\ [f_5 \quad v_5] \end{bmatrix} \\ f_2\ \sigma_7 \\ f_6\ v_6 \end{bmatrix}.$$

Definition 6 (*Meet*). *Let A and B be full-fledged feature structures representing the time-span trees of melodies A and B, respectively. If we can fix the greatest lower bound of A and B, that is, the greatest x such that $x \sqsubseteq A$ and $x \sqsubseteq B$ is unique, we call such x the meet of A and B, denoted as $A \sqcap B$.*

Similarly, we adopt the intersection of the unifiable feature structures as *meet*. For the previous examples, we have $\sigma_1 = \sigma_2 \sqcap \sigma_3$. We show another musical example in Figure 3. The notes in melody C also occur in both melodies A and B. They are the common parts of A and B, which are underlined in the figure.

Note that if we apply the *join* and *meet* operations to melodies for a finite number of times, in general, *compound values*, which are ground terms containing '$\sqcup$' and '$\sqcap$', may occur at value positions in the resulting feature structure.

3.2 Renderability and Reduction Path

We define a property required for reproducing from a feature structure to a real melody.

Definition 7 (Renderability). *A full-fledged feature structure is renderable, if it possesses only concrete (scalar) values for all the intrinsic features, excluding compound values; that is, the feature structure can be uniquely mapped to a piece of melody.*

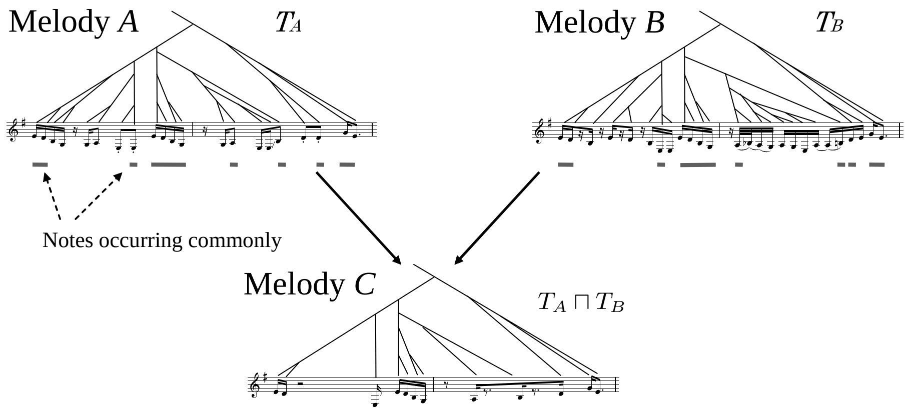


Fig. 3. *Meet* operation extracts common part of melodies A and B

Renderability means that a melody can be reproduced by neither more nor less information than is contained in a full-fledged feature structure. We hereafter restrict feature structures to be renderable ones.

For melodies A and B, $A \setminus B$ is the set of notes contained in A and not in $A \sqcap B$. Note that $A \setminus B$ is not a time-span tree but the set of notes. For example, in Figure 3, A is a melody of 21 notes, and C has 12 notes. The 12 notes of C are all contained in A and those in B, too. These 12 notes are underlined. We introduce a function of a set σ, returning the number of elements in set σ, $\#(\sigma)$: *Set* $\rightarrow$ *Integer*. For example, we have $\#(A \setminus C) = 21 - 12 = 9$.

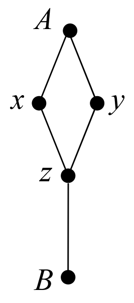
Definition 8 (Reduction Path). *For melodies A and B such that $T_B \sqsubseteq T_A$, a reduction path from A to B is defined as a sequence of melodies obtained by removing a note in $A \setminus B$ from A one-by-one, according to the algorithm, as follows:*

Step 1: $N := \#(T_A \setminus T_B)$, $i := 0$, and $T_0 := T_A$.
Step 2: Select note p of the minimum beat strength in $T_i \setminus T_B$.
Step 3: Reduce T_i to T_{i+1} by removing p.
Step 4: Iterate Steps 2 and 3 N times ($i = 0 \sim N - 1$).

The resulting sequence T_0, T_1, T_2, $\cdots$, T_N is the reduction path from A to B.

At Step 2, the algorithm collects the notes whose beat strengths are the minimum corresponding to the least important notes in a time-span tree. Since a note to be removed is nondeterministically selected from the collection, there is generally more than one reduction path. The selection is justified by the following time-span reduction preference rules: TSRPR1 (metrical position) and TSRPR5 (metrical stability) [7]. Hence the algorithm may automatically compute more than one reduction paths, and for every melody C on the reduction path from A to B, $B \sqsubsetneq C \sqsubsetneq A$ holds.

Furthermore, the long-step reduction in GTTM (Section 2.1) can always be embedded into a reduction path with preserving the ordering. We show an example of a reduction path from A to B in Figure 4:



Assume $\#(T_A \setminus T_B) = 3$, and we obtain reduction paths $A \to x \to z \to B$ and $A \to y \to z \to B$. Correspondingly, $B \sqsubseteq z \sqsubseteq x \sqsubseteq A$ and $B \sqsubseteq z \sqsubseteq y \sqsubseteq A$ hold.

Fig. 4. Reduction path

3.3 Melodic Complexity

Assume a function of melody A for measuring melodic complexity or the information contained in a melody, $|A|$: *Time-span tree* $\to$ *Real number*. We can use any function returning a real number such that $A \sqsubseteq B \Rightarrow |A| \leq |B|$. For example, the function may return the number of pitch events in A, the number of pitch evnets in a time-span tree, or the number of nodes weighted by the depth at which nodes occur. We adopt the number of pitch events in a melody as function $|A|$ for simplicity. We give the basic lemmas for $|A|$ below.

Lemma 1. *Let A and B be renderable feature structures that are unifiable with each other. Then $|A \sqcup B| = |A| + |B| - |A \sqcap B|$.*

Proof. When we consider melodies A and B in the set notation, the result is obvious from the algebra of sets. □

Lemma 2. *Given A and $A \sqcap B$, we can compose a reduction path from A to $A \sqcap B$, $A \sqcap B \sqsubseteq a_1 \sqsubseteq a_2 \sqsubseteq \cdots \sqsubseteq a_{n-1} \sqsubseteq A$. Then we have $|a_i| = i + |A \sqcap B|$ and $|a_i \sqcup B| = i + |B|$.*

Proof. Since $a_i = (a_i \setminus (A \sqcap B)) \sqcup (A \sqcap B)$, $|a_i| = i + |A \sqcap B|$. From Lemma 1, we have $|a_i \sqcup B| = |a_i| + |B| - |a_i \sqcap B|$. Since $a_i \sqcap B = A \sqcap B$, $|a_i \sqcap B| = |A \sqcap B|$. Therefore, we obtain $|a_i \sqcup B| = i + |A \sqcap B| + |B| - |A \sqcap B| = i + |B|$. □

3.4 Similarity and Interpolation

Definition 9 (Similarity). *The similarity measure between melodies A and B is defined as follows:*

$$S(A, B) = \frac{|A \sqcap B|}{max(|A|, |B|)} \ . \tag{1}$$

We borrow the definition from Hirata et al. [6], since it is constructed using the *meet* operation. Obviously from the definition, we have $S(A, B) = S(B, A)$. If $A = B$, then $S(A, B) = 1$, while if $A \sqcap B = \bot$, then $S(A, B) = 0$

Definition 10 (Interpolation). *Let A and B be melodies. Melody μ is an interpolation of A and B, if μ satisfies $S(A, B) \leq S(A, \mu)$ and $S(A, B) \leq S(B, \mu)$ (Figure 5).*

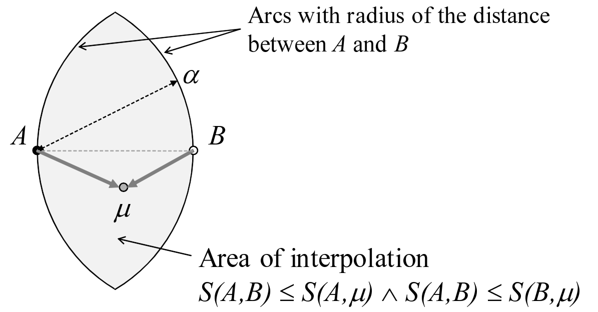

Arcs with radius of the distance between A and B
α
A
B
μ
Area of interpolation
S(A,B) ≤ S(A,μ) ∧ S(A,B) ≤ S(B,μ)

Fig. 5. Interpolation μ of melodies A and B

Figure 5 shows a schematic view of interpolation. Note that A, B, and μ are discrete feature structures representing melodies, although they appear to be in dense Euclidean space.

4 Proving Validity of Melodic Morphing Algorithm

4.1 Melodic Morphing Algorithm

We use an algorithm presented by Hamanaka et al. [3] to generate interpolating melody μ between A and B, because the algorithm is constructed using the *join* and *meet* operations. In the literature, to check if their algorithm works as planned, they ran an experiment, in which human subjects actually listened to and rated each sample melody generated by the algorithmic interpolation, given two melodies. After statistical processing, they concluded that the algorithm can generate the morphing melodies. In contrast, we employ formalism to check if the algorithm works as specified.

Definition 11 (Melodic Morphing Algorithm [3]). *The algorithm consists of the following steps (Figure 6):*

Step 1: Calculate $T_A \sqcap T_B$ (meet).
Step 2: Select melody T_C on the reduction path from T_A to $T_A \sqcap T_B$, and select T_D on the reduction path from T_B to $T_A \sqcap T_B$.
Step 3: Calculate $T_C \sqcup T_D$ (join), and the result is morphing melody μ.

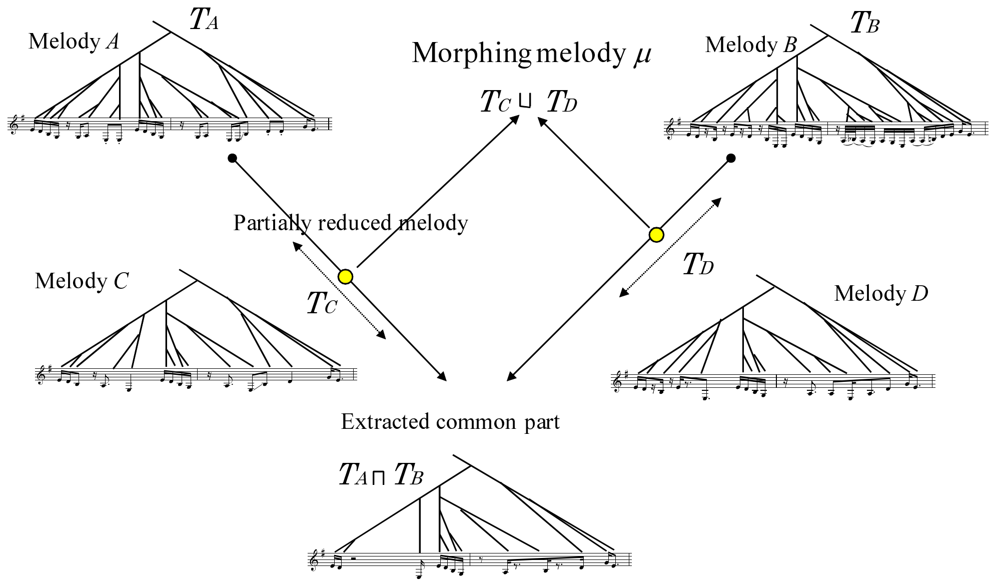


Fig. 6. Melodic morphing algorithm

If C close to T_A is chosen, the characters of melody A are reflected in output μ. On the other hand, If C close to $T_A \sqcap T_B$ is chosen, those of A are not so emphasized in μ. So is D similarly.

4.2 Proof of Morphing Theorem

The melodic morphing algorithm presented in the previous section can be shown in a diagram containing two reduction paths (Figure 7), which is called full morphing. Here, let us consider a simpler diagram containing a single reduction path, called half morphing in Figure 8. We split the proof process into two stages and first consider half morphing.

Lemma 3 (Similarity on side of reduction path). *For melodies A and B, we take x such that $(A \sqcap B) \sqsubseteq x \sqsubseteq A$. Then we have $S(A, B) \leq S(A, x \sqcup B)$ (Figure 8).*

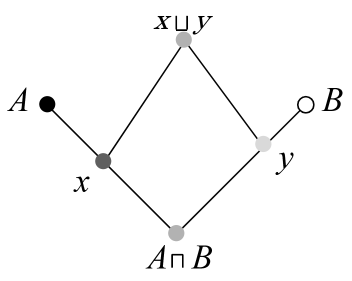


Fig. 7. Full morphing

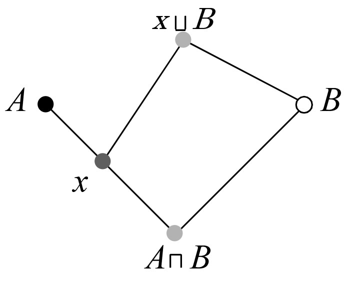


Fig. 8. Half morphing

Proof. We have

$$S(A, x \sqcup B) = \frac{|A \sqcap (x \sqcup B)|}{max(|A|, |x \sqcup B|)} = \frac{|x|}{max(|A|, |x \sqcup B|)} \ . \tag{2}$$

To compare the values of expressions 1 and 2, we consider the following two cases.
(C1) $|A| \leq |B|$. We get expression 1 = $|A \sqcap B|/|B|$, and expression 2 = $|x|/|x \sqcup B|$. For x in expression 2, we can chose i such that $x = a_i$. Lemma 2 leads to expression 2 = $|a_i|/|a_i \sqcup B| = (|A \sqcap B| + i)/(|B| + i)$, where $0 \leq i \leq n - 1$. Therefore, expression 1 $\leq$ expression 2.
(C2) $|B| \leq |A|$. If x is increased from $A \sqcap B$ to A, then $x \sqcup B$ is increased from B to $A \sqcup B$. Since $|A| \leq |A \sqcup B|$, we can assume that the value of $|x \sqcup B|$ equals $|A|$ at a certain point. That is, intermediate melody a_K exists such that $A \sqcap B \sqsubseteq a_K \sqsubseteq A$ and $|A| = |a_K \sqcup B|$. Now, we further consider two cases, depending on the denominator of expression 2.
(C2-1) $A \sqcap B \sqsubseteq x \sqsubseteq a_K$. We have expression 1 = $|A \sqcap B|/|A|$, and expression 2 = $|x|/|A|$. Since $A \sqcap B \sqsubseteq x \sqsubseteq A$, expression 1 $\leq$ expression 2.
(C2-2) $a_K \sqsubseteq x \sqsubseteq A$. For i such that $i \geq K$, we have expression 2 = $|a_i|/|a_i \sqcup B|$. From Lemma 2, we get expression 2 = $(|A \sqcap B| + i)/(|B| + i)$. Next, we calculate the difference between expressions 1 and 2. The numerator of (expression 2 − expression 1) is $|A \sqcap B| \cdot (|A| - |B|) + i \cdot (|A| - |A \sqcap B|)$, and its denominator is $|A| \cdot (|B| + i)$. Since both the numerator and denominator are positive, expression 2 − expression 1 ≥ 0.
Therefore, for all cases (C1)∼(C2-2), expression 1 $\leq$ expression 2. □

Lemma 4 (Similarity on diagonal side of reduction path). *Given melodies A and B, we have y such that $(A \sqcap B) \sqsubseteq y \sqsubseteq B$. Then $S(A, B) \leq S(A, y)$.*

Proof. We have

$$S(A, y) = \frac{|A \sqcap y|}{max(|A|, |y|)} \ . \tag{3}$$

To compare the values of expressions 1 and 3, let us consider the following three cases:

(C1) $|y| \le |B| \le |A|$. We have expression 1 = $|A \sqcap B|/|A|$, and expression 3 = $|y \sqcap A|/|A|$. Since $A \sqcap B \sqsubseteq y \sqsubseteq B$, $y \sqcap A = A \sqcap B$. Hence, expression 1 = expression 3.
(C2-1) $|A| \le |y| \le |B|$. We have expression 1 = $|A \sqcap B|/|B|$, and expression 3 = $|y \sqcap A|/|y|$. Since $y \sqcap A = A \sqcap B$ and $|y| \le |B|$, expression 1 $\le$ expression 3.
(C2-2) $|y| \le |A| \le |B|$. We have expression 1 = $|A \sqcap B|/|B|$, and expression 3 = $|y \sqcap A|/|A|$. Since $y \sqcap A = A \sqcap B$ and $|A| \le |B|$, expression 1 $\le$ expression 3. □

Theorem 1 (Morphing). *Full morphing satisfies the condition of interpolation.*

Proof. In the statement of Lemma 3, we replace B with y, and then $S(A, y) \le S(A, x \sqcup y)$ is obtained. Together with Lemma 4, we obtain equation $S(A, B) \le S(B, x \sqcup y)$. If the equation exchanges A and B for each other, then it holds, too. □

5 Conclusion

We have successfully proved the validity of the melodic morphing algorithm in a formal way and demonstrated the adequacy of the representation method for a time-span tree in feature structure and the power of an algebraic framework for designing and implementing musical tasks. We believe further room exists for extending descriptive power with preserving the simplicity in our framework. If creators could manipulate music more freely, they would enhance our capability to create music in the context of UGC and CGM.

We would like to add two comments to our design of a morphing algorithm. First, we proved the soundness of the melodic morphing algorithm in the sense that all the results generated by it are interpolation. However, we conjecture that the melodic morphing algorithm is not complete because in general, there is more than one reduction path from B to $A \sqcap B$, as pointed out in Section 3.2. Thus, we are interested in designing a melodic morphing algorithm that is sound and complete. Second, besides the melodic morphing algorithm based on interpolation, we can consider one based on extrapolation [4]. We are interested in providing the formal definition of extrapolation and proving its properties.

Our future work includes the following. In Section 3.1, we introduced *join* and *meet* operations, which however only work correctly on renderable, full-fledged feature structures that represent time-span trees (Section 3.2). From a realistic point of view, the above condition seems too restrictive; when a creator arbitrarily gives two melodies to the *join* operation, they are not unifiable in many cases, hence *join* cannot be calculated (the result of *meet* becomes $\perp$). Conversely, let us think of a method to obtain two melodies that are unifiable. Let A and B be melodies such that $A \sqsubseteq B$, and there are two distinct reduction paths from B to A, P_1 and P_2. We choose one melody on P_1, C and the other on P_2, D. Since C and D are unifiable, we can calculate both $C \sqcup D$ and $C \sqcap D$ ($\neq \perp$). As we see in the process of making C and D, it is obvious that C and D are similar (sounds similarly) to each other. Therefore, to meet a more practical situation, we must improve the *join* and *meet* operations to handle melodies that are not similar to each other and/or the representation method for a time-span tree in feature structure.

Also, we need to revise the definition of similarity (Definition 9), because it includes the following problem. Suppose that three melodies A, B_1, and B_2 satisfy $(A \sqcap B_2) \sqsubseteq B_2 \sqsubseteq B_1$ and $|B_2| \leq |B_1| \leq |A|$. Then, according to expression 1, we have $S(A, B_1) = S(A, B_2)$ since $max(|A|, |B_i|) = |A|$ ($i = 1$ or 2). However, this result is contrary to our intuition, and $S(A, B_1) \leq S(A, B_2)$ seems more natural. One possible solution is to divide $|A \sqcap B_i|$ by $|A \sqcup B_i|$ for the normalization in terms of size. For the more precise definition of similarity, we may further take into account the length of reduction path, i.e., the number of reduction steps from A and B_i to $A \sqcap B_2$.

For these purposes, we have to examine in more depth the computational meaning of the time-span reduction, its tree representation, and the basic operations on time-span trees.

Acknowledgement. This work is partially supported by Grant for Scientific Research (B) 20300035 of the Ministry of Education, Culture, Sports, Science and Technology, Japan.

References

1. Carpenter, B.: The Logic of Typed Feature Structures. Cambridge University Press (1992)
2. Davenport, J.H.: Fast REDUCE: the trade-off between efficiency and generality. ACM SIGSAM, vol. 16, issue 1, pp. 8–11 (1982)
3. Hamanaka, M., Hirata, K., Tojo, S.: Melody Morphing Method Based on GTTM. In: Proc. of ICMC 2008, pp.155–158 (2008)
4. Hamanaka, M., Hirata, K., Tojo, S.: Melody Extrapolation in GTTM Approach. In: ICMC 2009, pp. 89–92 (2009)
5. Harnad, S.: The symbol grounding problem. Physica D 42, pp. 335–346 (1990)
6. Hirata, K., Matsuda, S.: Interactive Music Summarization based on Generative Theory of Tonal Music. J. New Music Research, vol. 32, no. 2, pp. 165–177 (2003)
7. Lerdahl, F., Jackendoff, R.: A Generative Theory of Tonal Music. The MIT Press, Cambridge (1983)
8. Levesque, H.J., Brachman, R.J.: Expressiveness and tractability in knowledge representation and reasoning. Computational intelligence, vol. 3, no. 2, pp. 78–93 (1987)
9. Marsden A.: Generative Structural Representation of Tonal Music. J. New Music Research, vol. 34, no. 4, pp. 409–428 (2005)
10. Norman, D.: The Invisible Computer: Why Good Products Can Fail, the Personal Computer Is So Complex, and Information Appliances Are the Solution. The MIT Press, Cambridge (1998)
11. Sag, I.A., Wasow, T.: Syntactic Theory: A Formal Introduction. CSLI Publications (1999)
12. Selfridge-Field, E.: Conceptual and representational issues in melodic comparison. Computing in Musicology vol. 11, pp. 3–64, The MIT Press, Cambridge (1998)

13. Valero, D.R.: Symbolic Music Comparison with Tree Data Structure. Ph.D. Thesis, Universitat d' Alacant, Departament de Llenguatges i Sistemes Informatics (2010)
14. Wiggins, G., Smaill, A.: Musical knowledge: What can artificail intelligence bring to the musician? In: Miranda, E.R. (Ed.), Readings in Music and Artificial Intelligence, pp. 29–46. Harwood Academic Publisheres, Amsterdam (2000)