

(19) 日本国特許庁(JP)

(12) 特 許 公 報(B2)

(11) 特許番号

特許第6428853号
(P6428853)

(45) 発行日 平成30年11月28日(2018.11.28)

(24) 登録日 平成30年11月9日(2018.11.9)

(51) Int.Cl. F I
G 1 O H 1/00 (2006.01) G 1 O H 1/00 1 O 2 Z
G 1 O H 1/38 (2006.01) G 1 O H 1/38 Z

請求項の数 6 (全 47 頁)

(21) 出願番号 特願2017-114112 (P2017-114112)
(22) 出願日 平成29年6月9日(2017.6.9)
(62) 分割の表示 特願2014-235233 (P2014-235233)
の分割
原出願日 平成26年11月20日(2014.11.20)
(65) 公開番号 特開2017-182089 (P2017-182089A)
(43) 公開日 平成29年10月5日(2017.10.5)
審査請求日 平成29年11月15日(2017.11.15)

(73) 特許権者 000001443
カシオ計算機株式会社
東京都渋谷区本町1丁目6番2号
(72) 発明者 南高 純一
東京都羽村市栄町3丁目2番1号 カシオ
計算機株式会社羽村技術センター内
審査官 山下 剛史

最終頁に続く

(54) 【発明の名称】 自動作曲装置、方法、およびプログラム

(57) 【特許請求の範囲】

【請求項1】

入力されたモチーフに含まれる複数のノート夫々に対応する複数のノートタイプを、コード進行を示すコード進行データごとに取得するノートタイプ取得処理と、

連続するノートのノートタイプと前記連続するノートの音高の差を示す隣接音程との組み合わせを評価するノート接続ルールにおける前記評価に基づいて、前記コード進行データが前記モチーフにどの程度適合しているかを示す適合度を前記コード進行データごとに算出する適合度算出処理と、

前記適合度算出処理により算出された前記適合度に基づいて、コード進行データを選択するコード進行データ選択処理と、

前記入力されたモチーフに基づいてメロディデータを生成するメロディデータ生成処理と、

前記コード進行データ選択処理により選択されたコード進行データに基づいて伴奏データを生成する伴奏データ生成処理と、

を実行する制御部を備える自動作曲装置。

【請求項2】

前記制御部は、

前記メロディデータ生成処理により生成されたメロディデータと、前記コード進行データ選択処理により選択されたコード進行データに基づく伴奏データと、に基づく楽音を再生する楽音再生処理、

10

20

を実行する請求項 1 に記載の自動作曲装置。

【請求項 3】

前記制御部は、

前記メロディデータ生成処理により生成されたメロディデータに基づく楽譜を表示する楽譜表示処理、

を実行する請求項 1 又は 2 に記載の自動作曲装置。

【請求項 4】

前記メロディデータ生成処理は、

前記コード進行データ選択処理により選択されたコード進行データに対応する曲構造データに含まれる複数のフレーズのうちのいずれかのフレーズの先頭 2 小節のメロディを、前記入力されたモチーフに基づいて生成することを特徴とする請求項 1 乃至 3 のいずれかに記載の自動作曲装置。

【請求項 5】

自動作曲装置が、

入力されたモチーフに含まれる複数のノート夫々に対応する複数のノートタイプを、コード進行を示すコード進行データごとに取得し、

連続するノートのノートタイプと前記連続するノートの音高の差を示す隣接音程との組み合わせを評価するノート接続ルールにおける前記評価に基づいて、前記コード進行データが前記モチーフにどの程度適合しているかを示す適合度を前記コード進行データごとに算出し、

前記算出された前記適合度に基づいて、コード進行データを選択し、

前記入力されたモチーフに基づいてメロディデータを生成し、

前記選択されたコード進行データに基づいて伴奏データを生成する、

自動作曲方法。

【請求項 6】

入力されたモチーフに含まれる複数のノート夫々に対応する複数のノートタイプを、コード進行を示すコード進行データごとに取得するステップと、

連続するノートのノートタイプと前記連続するノートの音高の差を示す隣接音程との組み合わせを評価するノート接続ルールにおける前記評価に基づいて、前記コード進行データが前記モチーフにどの程度適合しているかを示す適合度を前記コード進行データごとに算出するステップと、

前記算出された前記適合度に基づいて、コード進行データを選択するステップと、

前記入力されたモチーフに基づいてメロディデータを生成するステップと、

前記選択されたコード進行データに基づいて伴奏データを生成するステップと、

をコンピュータに実行させるプログラム。

【発明の詳細な説明】

【技術分野】

【0001】

本発明は、自動作曲装置、方法、およびプログラムに関する。

【背景技術】

【0002】

複数のノートデータから構成されるモチーフメロディに基づいて、自動作曲を行う技術が知られている。例えば、次のような従来技術が知られている（例えば特許文献 1 に記載の技術）。特定調のコード進行を記憶したデータベースから所定のコード進行を選択し、モチーフを所定の調で入力すると、入力モチーフからモチーフ調が検出される。検出されたモチーフ調に基づきコード進行データがモチーフ調に移調され、メロディ生成段において、入力モチーフ及びモチーフ調に移調後のコード進行に基づき、モチーフ調におけるメロディが生成される。また、検出されたモチーフ調に基づきモチーフが特定調に移調され、特定調のコード進行及び移調後モチーフに基づいて特定調のメロディが生成され、その後、モチーフ調のメロディに移調される。

【 0 0 0 3 】

また、次のような従来技術も知られている（例えば特許文献 2 に記載の技術）。楽曲データのカラオケ演奏用データやガイドメロディデータから 4 分音符以上の長さの音符を抽出し、その音名（C ～ B）の出現度数の分布を集計する。この度数分布と長調判定スケールおよび短調判定スケールとを比較し、最も分布形状が一致するところを主音（音階音）とする調であると判定し、この調判定結果とガイドメロディデータとに基づいてハーモニーデータを生成し、このハーモニーデータに基づいてハーモニー音声信号を形成する。

【 先行技術文献 】

【 特許文献 】

【 0 0 0 4 】

【 特許文献 1 】 特許公開 2 0 0 2 - 3 2 0 8 0 号公報

【 特許文献 2 】 特許公開平 1 0 - 1 0 5 1 6 9 号公報

【 発明の概要 】

【 発明が解決しようとする課題 】

【 0 0 0 5 】

しかし、上記従来技術では、モチーフメロディは、ある調に特定されるという暗黙の前提たっており、モチーフ内での転調やモード的なメロディや無調のモチーフなどに対応できなかった。また、調の判定は音高の分布に基づいた方法であるため、正確な対応関係が得られない場合がある。たとえば、「レドシラソ」、「ソレラシド」は、同じ音高分布だが、それぞれ、ト長調、ハ長調とみなされるべきである。

【 0 0 0 6 】

そこで、本発明は、適切なコード進行データを選択可能として、自然な楽曲生成を実現することを目的とする。

【 課題を解決するための手段 】

【 0 0 0 7 】

態様の一例では、入力されたモチーフに含まれる複数のノート夫々に対応する複数のノートタイプを、コード進行を示すコード進行データごとに取得するノートタイプ取得処理と、連続するノートのノートタイプと前記連続するノートの音高の差を示す隣接音程との組み合わせを評価するノート接続ルールにおける前記評価に基づいて、前記コード進行データが前記モチーフにどの程度適合しているかを示す適合度を前記コード進行データごとに算出する適合度算出処理と、前記適合度算出処理により算出された前記適合度に基づいて、コード進行データを選択するコード進行データ選択処理と、前記入力されたモチーフに基づいてメロディデータを生成するメロディデータ生成処理と、前記コード進行データ選択処理により選択されたコード進行データに基づいて伴奏データを生成する伴奏データ生成処理と、を実行する制御部を備える。

【 発明の効果 】

【 0 0 0 8 】

本発明によれば、適切なコード進行データを選択可能として、自然な楽曲生成を実現することが可能となる。

【 図面の簡単な説明 】

【 0 0 0 9 】

【 図 1 】 自動作曲装置の実施形態のブロック図である。

【 図 2 】 本実施形態において自動作曲される楽曲の構造例を示す図である。

【 図 3 】 入力モチーフ 1 0 8 とコード進行データの適合動作例

【 図 4 】 入力モチーフのデータ構成例を示す図である。

【 図 5 】 伴奏・コード進行 D B のデータ構成例を示す図である。

【 図 6 】 1 レコード中の曲構造データのデータ構成例を示す図である。

【 図 7 】 標準ピッチクラスセットテーブルのデータ構成例を示す図である。

【 図 8 】 ノートタイプ、隣接音程、およびノートタイプと隣接音程の配列変数データにつ

10

20

30

40

50

いての説明図である。

【図 9】ノート接続ルールのデータ構成例を示す図である。

【図 10】コード進行選択部 102 の動作説明図である。

【図 11】フレーズセット DB のデータ構成例を示す図である。

【図 12】メロディ変形処理およびメロディ最適化処理の動作説明図である。

【図 13】メロディ最適化処理の詳細動作説明図である。

【図 14】自動作曲装置のハードウェア構成例を示す図である。

【図 15 A】各種変数データ、配列変数データ、および定数データのリストを示す図（その 1）である。

【図 15 B】各種変数データ、配列変数データ、および定数データのリストを示す図（その 2）である。 10

【図 16】自動作曲処理の例を示すフローチャートである。

【図 17】コード進行選択処理の詳細例を示すフローチャートである。

【図 18】コードデザインデータ作成処理の詳細例を示すフローチャートである。

【図 19】入力モチーフとコード進行の適合度チェック処理の詳細例を示すフローチャートである。

【図 20】チェック処理の詳細例を示すフローチャートである。

【図 21】入力モチーフの現在のノートのタイミングに対応するコード情報の取得処理の詳細例を示す図である。

【図 22】ノートタイプ取得処理の詳細例を示す図である。 20

【図 23】ノート接続性チェック処理の詳細例を示す図である。

【図 24】メロディ生成処理の詳細例を示す図である。

【図 25】メロディ生成 1 処理の詳細例を示す図である。

【図 26】フレーズセット DB 検索処理の詳細例を示す図ある。

【図 27】メロディ変形処理の詳細例を示す図である。

【図 28】メロディ最適化処理の詳細例を示す図である。

【図 29】メロディ生成 2 処理の詳細例を示す図である。

【発明を実施するための形態】

【0010】

以下、本発明を実施するための形態について図面を参照しながら詳細に説明する。図 1 は、自動作曲装置 100 の実施形態のブロック図である。この自動作曲装置 100 は、モチーフ入力部 101、コード進行選択部 102、伴奏・コード進行データベース（以下、「データベース」を「DB」と称する）103、ルール DB 104、メロディ生成部 105、フレーズセット DB 106、および出力部 107 を備える。 30

【0011】

モチーフ入力部 101 は、いわゆる A メロ、B メロ、C メロ（サビメロ）などの、曲調を決定付ける特徴的なメロディ部分のいずれかを、入力モチーフ 108 としてユーザに入力させる。入力モチーフ 108 は、A メロ部分のモチーフであるモチーフ A、B メロ部分のモチーフであるモチーフ B、あるいは C メロ（サビメロ）部分のモチーフであるモチーフ C のいずれかであり、例えば各メロディ部分の先頭の 2 小節の長さを有する。モチーフ入力部 101 は例えば、ユーザが鍵盤によりメロディを入力する鍵盤入力部 101 - 1、ユーザがマイクから歌声によりメロディを入力する音声入力部 101 - 2、ユーザがメロディを構成する音符のデータをキーボード等から入力する音符入力部 101 - 3 のいずれか一つ以上の手段を備える。また入力部 101 は、A メロ、B メロ、C メロ（サビメロ）というモチーフの種別を入力する、独立した操作子等を有する。 40

【0012】

コード進行選択部 102 は、伴奏・コード進行 DB 103 に記憶されている複数のコード進行データごとに、ルール DB 104 を参照しながら、そのコード進行データがモチーフ入力部 101 から入力された入力モチーフ 108 にどの程度適合しているかを示す適合度を算出し、適合度が高かった例えば上位 3 個のコード進行データをそれぞれ指し示す # 50

0、# 1、# 2のコード進行候補指示データ（図1中では「コード進行候補」と表示）109を出力する。

【0013】

メロディ生成部105は、例えばユーザに、コード進行選択部102が出力した# 0、# 1、# 2のコード進行候補指示データ109に対応する3つのコード進行候補のうちの1つを選択させる。あるいは、メロディ生成部105は、# 0、# 1、# 2のコード進行候補指示データ109のいずれかに対応するコード進行候補を自動的に順番に選択するようにしてもよい。この結果、メロディ生成部105は、選択されたコード進行候補に対応する曲構造データを、伴奏・コード進行DB103から読み込む。メロディ生成部105は、この曲構造データによって示される小節のフレーズごとに、入力モチーフ108とフレーズセットDB106に登録されているフレーズセット、およびルールDB104を参照しながら、そのフレーズのメロディを自動生成する。メロディ生成部105は、楽曲全体の小節にわたってメロディの自動生成処理を実行し、自動生成されたメロディ110を出力する。

【0014】

出力部107は、メロディ生成部105が自動生成したメロディデータ110に基づいてメロディの楽譜を表示する楽譜表示部107-1と、メロディデータ110および伴奏・コード進行DB103から取得した伴奏用MIDI（Musical Instrument Digital Interface）データとに基づいて、メロディおよび伴奏の再生を実行する楽音再生部107-2とを備える。

【0015】

次に、図1の機能構成を有する自動作曲装置100の動作の概略について説明する。図2は、本実施形態において自動作曲される楽曲の構造例を示す図である。楽曲は通常、イントロ、Aメロ、Bメロ、間奏、Cメロ（サビメロ）、エンディングなどのフレーズから構成される。イントロは、メロディが開始する前の伴奏のみからなる前奏部分である。Aメロは、通常、イントロの次に出てくるフレーズをいい、曲の中で一般には落ち着いたメロディが奏でられる。Bメロは、Aメロの次に出てくるフレーズをいい、Aメロより少し盛り上がった曲調になることが多い。Cメロは、Bメロの次に出てくるフレーズの場合が多く、日本の曲だとCメロが曲で一番盛り上がるサビメロになる場合が多い。エンディングは、イントロの逆で、曲の終わりのフレーズをいう。間奏は、例えば1曲目と2曲目の間のメロディの存在しない楽器演奏のみのフレーズである。図2に示される楽曲の構造例では、イントロ、Aメロ、Bメロ、Aメロ、間奏、Aメロ、Bメロ、Cメロ、エンディングの順に楽曲が構成されている。

【0016】

本実施形態では、ユーザは例えば、楽曲中で最初に現れるAメロの例えば先頭2小節のメロディを、モチーフ入力部101（図1参照）から、図2（a）のモチーフAとして（図1の入力モチーフ108の一例）として入力することができる。または、ユーザは例えば、楽曲中で最初に現れるBメロの例えば先頭2小節のメロディを、モチーフ入力部101（図1参照）から、図2（b）のモチーフB（図1の入力モチーフ108の他の一例）として入力モチーフ108として入力することができる。あるいは、ユーザは例えば、楽曲中で最初に現れるCメロ（サビメロ）の例えば先頭2小節のメロディを、モチーフ入力部101（図1参照）から、図2（c）のモチーフC（図1の入力モチーフ108のさらに他の一例）として入力することができる。

【0017】

図3（a）は、上述のように入力される入力モチーフ108の音符例を示す図である。このように、入力モチーフ108としては、例えば2小節分のメロディが指定される。

【0018】

このような入力に対して、コード進行選択部102（図1参照）が、伴奏・コード進行DB103に登録されているコード進行データの中から、例えば上位3位まで適合するコードとキー、スケールとからなるコード進行データを抽出する。コード進行データを構成

10

20

30

40

50

するコードおよびキー、スケールは、図2 (f) および (g) に示されるように、楽曲全体にわたって設定されている。

【 0 0 1 9 】

図3 (b) は、上位3位までのコード進行データによって表されるコード進行 (コードおよびキー、スケール) # 0、# 1、# 2の例を示す図である。

【 0 0 2 0 】

図1のメロディ生成部105は、これらの情報に基づいて、入力モチーフ108が入力された図2 (a)、(b)、または (c) のいずれかのフレーズ部分以外の図2 (d) に示されるフレーズ部分に対応するメロディを自動生成し、入力モチーフ108のメロディとともにメロディ110として出力する。そして、図1の出力部107が、自動生成されたメロディ110に対応する楽譜表示または放音を行う。なお、伴奏については、伴奏・コード進行DB103において最終的に選択されたコード進行に対応して登録されている伴奏用MIDIデータが順次読み出されて、そのデータに基づいて図2 (e) に示されるように楽曲全体にわたり伴奏が行われる。

【 0 0 2 1 】

図4は、図1のモチーフ入力部101において、ユーザ入力に基づいて生成される入力モチーフ108のデータ構成例を示す図である。図4 (a) に示されるように、入力モチーフ108は、# 0、# 1、・・・という複数のノートデータによって構成され、最後に終端コードが記憶される。各ノートデータは、例えば図3 (a) に例示される入力モチーフ108を構成する例えば2小節分の音符のそれぞれに対応し、モチーフとなるメロディ音の発音を指示するデータである。図4 (b) に示されるように、1つのノートデータは、そのノートデータに対応する音符の発音タイミングを例えば入力モチーフ108の先頭からの経過時間として示す「時間」データと、音符の長さを示す「長さ」データと、音符の強さを示す「強さ」データと、音符の音高を示す「ピッチ」データとから構成される。これらのデータによって、図3 (a) に例示されるような2小節分の入力モチーフ108中の1つの音符が表現される。

【 0 0 2 2 】

図5は、図1の伴奏・コード進行DB103のデータ構成例を示す図である。図5 (a) に示されるように、コード進行DBには、1つのレコード (図5 (a) の1行) がコード進行データ、伴奏用MIDIデータ、および曲構造データとからなる、# 0、# 1、・・・という複数レコードが記憶され、最後に終端コードが記憶される。

【 0 0 2 3 】

1レコード中のコード進行データは、楽曲の1曲分のコード進行を示している。図5 (a) に示されるコード進行DBには例えば、50レコード = 50曲分のコード進行データが記憶されている。1レコード中 (= 1曲分) のコード進行データは、図5 (b) に示されるように、# 0、# 1、・・・という複数のコードデータから構成され、最後に終端コードが記憶される。コードデータには、あるタイミングにおけるキーおよびスケールを指定するデータ (図5 (c)) と、あるタイミングにおけるコードを指定するデータ (図5 (d)) とがある (図3 (b) 参照)。キーおよびスケールを指定するデータは、図5 (c) に示されるように、そのキーおよびスケールが始まるタイミングを示す「時間」データと、「キー」データと、「スケール」データとから構成される。コードを指定するデータは、図5 (d) に示されるように、そのコードが始まるタイミングを示す「時間」データと、コードの根音 (ルート) を示す「ルート」データ、およびコードのタイプ (種類) を示す「タイプ」データとから構成される。コード進行データは例えば、MIDI規格のメタデータとして記憶される。

【 0 0 2 4 】

図5 (a) に示される伴奏・コード進行DB103の1レコード中 (= 1曲分) の曲構造データは、図6に示されるデータ構成例を有する。この曲構造データは、1曲中の小節ごとに1レコード (図6の1行) を形成する。曲構造データ中の1レコードには、その小節に対応するフレーズの種別およびそのフレーズにメロディが存在するか否かを示す情報

10

20

30

40

50

が記憶される。

【 0 0 2 5 】

図 6 に示される曲構造データにおいて、「Measure」項目には、各レコードのデータが楽曲中の何小節目であることを示す値が登録される。以降、「Measure」項目の値が M であるレコードを第 M レコード、そのレコードが示す小節を第 M + 1 小節とする。例えば「Measure」項目の値が 0 であるときそのレコードは第 0 レコード / 第 1 小節、その値が 1 であるときそのレコードは第 1 レコード / 第 2 小節である。

【 0 0 2 6 】

図 6 に示される曲構造データにおいて、「PartName[M]」項目および「iPartID[M]」項目（「M」は「Measure」項目の値）にはそれぞれ、第 M レコード / 第 M + 1 小節のフレーズの種別およびその種別に対応する識別値を示すデータが登録される。例えば、第 0 レコード（第 1 小節）の「PartName[M]」項目および「iPartID[M]」項目の値「Null」および「0」は、その小節が無音であることを示している。第 1、2 レコード（第 2、3 小節）の「PartName[M]」項目および「iPartID[M]」項目の値「Intro」および「1」は、その小節がイントロフレーズであることを示している。第 3 ~ 10、28 ~ 34 レコード（第 4 ~ 11、29 ~ 35 小節）の「PartName[M]」項目および「iPartID[M]」項目の値「A」および「11」は、その小節が A メロのフレーズであることを示している。第 11 ~ 18 レコード（第 12 ~ 19 小節）の「PartName[M]」項目および「iPartID[M]」項目の値「B」および「12」は、その小節が B メロのフレーズであることを示している。第 19 ~ 27 レコード（第 20 ~ 28 小節）の「PartName[M]」項目および「iPartID[M]」項目の値「C」および「13」は、その小節が C メロ（またはサビメロディ）のフレーズであることを示している。第 35 レコード（第 36 小節）の「PartName[M]」項目および「iPartID[M]」項目の値「Ending」および「3」は、その小節がエンディングのフレーズであることを示している。

【 0 0 2 7 】

また、図 6 に示される曲構造データにおいて、「ExistMelody[M]」項目（「M」は「Measure」項目の値）には、第 M レコード（第 M + 1 小節）のフレーズにメロディが存在するか否かを示す値が登録される。メロディが存在するならば値「1」が、存在しないならば値「0」が登録される。例えば、M = 0、1、2、または 35（第 0、1、2、35 レコード（第 1、2、3、36 小節））である「PartName[M]」項目が「Null」、「Intro」、または「Ending」の各フレーズの「ExistMelody[M]」項目には値「0」が登録されて、メロディが存在しないことが示される。PartName[M] = 「Null」の場合は無音で、PartName[M] = 「Intro」、または「Ending」の場合は伴奏のみが存在する。

【 0 0 2 8 】

また、図 6 に示される曲構造データにおいて、「iPartTime[M]」項目（「M」は「Measure」項目の値）には、第 M レコードに対応する第 M + 1 小節の小節開始時間データが登録される。図 6 中では空欄になっているが、各レコードに実際の時間値が格納される。

以上の図 6 に示される曲構造データは例えば、MIDI 規格のメタデータとして記憶される。

【 0 0 2 9 】

図 2 で前述したように、ユーザは例えば、図 6 の曲構造データで最初に現れる A メロの例えば先頭 2 小節である第 3、4 レコード（第 4、5 小節）のメロディを、モチーフ A（図 2（a）参照）として、モチーフ入力部 101（図 1 参照）から入力できる。または、ユーザは例えば、図 6 の曲構造データで最初に現れる B メロの例えば先頭 2 小節である第 11、12 レコード（第 12、13 小節）のメロディを、モチーフ B（図 2（b）参照）として、モチーフ入力部 101 から入力できる。あるいは、ユーザは例えば、図 6 の曲構

10

20

30

40

50

造データで最初に現れるＣメロ（サビメロ）の例えば先頭２小節である第１９、２０レコード（第２０、２１小節）のメロディを、モチーフＣ（図２（ｃ）参照）として、モチーフ入力部１０１から入力できる。

【００３０】

コード進行選択部１０２は、伴奏・コード進行ＤＢ１０３に記憶されているコード進行データごと（以下「評価対象のコード進行データ」と記載する）に、その評価対象のコード進行データがモチーフ入力部１０１から入力された入力モチーフ１０８にどの程度適合しているかを示す適合度を算出する。

【００３１】

本実施形態では、入力モチーフ１０８に対する評価対象のコード進行データの適合度を、音楽理論におけるアヴェイラブルノートスケールの概念を使って算出する。アヴェイラブルノートスケールは、コード進行が与えられたときに、メロディに使うことが可能な音を音階として表したものである。アヴェイラブルノートスケールを構成するノートの種類（以下、「ノートタイプ」と呼ぶ）としては、例えば、コードトーン、アヴェイラブルノート、スケールノート、テンションノート、アヴォイドノートがある。コードトーンは、スケールの元となるコードの構成音であって、メロディとして１音は用いることが望ましいノートタイプである。アヴェイラブルノートは、メロディに一般的に使用可能なノートタイプである。スケールノートは、スケールの構成音であり、その音を長い音などで加えると、元々のコードサウンドとぶつかってしまうので、取り扱いに注意を要するノートタイプである。テンションノートは、コード音にかぶせられる、コードのテンションで用いられている音で、高次のテンションほどサウンドの緊張感が増したり色彩豊かなサウンドになるノートタイプである。アヴォイドノートは、コードと不協和な音で、使用を避けるか、短い音符で用いることが望ましいとされるノートタイプである。本実施形態では、入力モチーフ１０８を構成する各ノート（図３（ａ）の各音符）について、そのノートの発音タイミングに対応する評価対象のコード進行データ中のキーおよびスケールとコードの根音およびコードタイプとに基づいて、そのノートの当該コード進行上でのノートタイプが算出される。

【００３２】

上述した、入力モチーフ１０８を構成する各ノート（図３（ａ）の各音符）のノートタイプを取得するために、本実施形態では、標準ピッチクラスセットテーブルが使用される。図７は、標準ピッチクラスセットテーブルのデータ構成例を示す図である。標準ピッチクラスセットテーブルはコード進行選択部１０２内のメモリ領域（例えば後述する図４のＲＯＭ１４０２内）に置かれる。標準ピッチクラステーブルは、図７（ａ）に例示されるコードトーンテーブル、図７（ｂ）に例示されるテンションノートテーブル、および図７（ｃ）に例示されるスケールノートテーブルから構成される。

【００３３】

図７（ａ）、（ｂ）、または（ｃ）のテーブルにおいて、その１行に対応する１組のピッチクラスセットは、コードまたはスケールの根音を第０音（第０ビット目）の音階構成音としたときの１オクターブ分の半音階を構成する第０音（第０ビット目）（図中の行の右端）から第１１音（第１１ビット目）（図中の行の左端）の音階構成音のそれぞれに対して、「０」または「１」の値が与えられる、合計１２ビットのデータで構成される。１組のピッチクラスセットにおいて、値「１」が与えられた音階構成音はそれがピッチクラスセットの構成要素に含まれ、値「０」が与えられた音階構成音はそれがピッチクラスセットの構成要素に含まれないことを示す。

【００３４】

図７（ａ）のコードトーンテーブル内の各行に対応するピッチクラスセット（以下、「コードトーンピッチクラスセット」と呼ぶ）は、その右端に記載されているコードタイプについて、そのコード根音が第０音（第０ビット目）の音階構成音として与えられたときに、どの音階構成音がそのコードタイプのコード構成音であるかを記憶する。例えば、図７（ａ）に例示されるコードトーンテーブルの１行目において、コードトーンピッチクラ

スセット「000010010001」は、第0音（第0ビット目）、第4音（第4ビット目）、および第7音（第7ビット目）の各音階構成音がコードタイプ「MAJ」のコード構成音であることを表わしている。

【0035】

図1のコード進行選択部102は、入力モチーフ108を構成するノートごと（以下、このノートを「現在ノート」と呼ぶ）に、その現在ノートのピッチが、その現在ノートの発音タイミングに対応する評価対象のコード進行データ中のコード根音に対して、どの音程（以下、これを「コード音程」と呼ぶ）を有するかを算出する。このとき、コード進行選択部102は、現在ノートのピッチを、その現在ノートの発音タイミングに対応する評価対象のコード進行データ中のコード根音を第0音の音階構成音としたときの、第0音から第11音までの1オクターブ内の音階構成音のいずれかに写像させる演算を行い、その写像位置の音（第0音から第11音のいずれか）を、上記コード音程として算出する。そして、コード進行選択部102は、上記発音タイミングにおける評価対象のコード進行データ中のコードタイプに対応する図7（a）に例示されるコードトーンテーブル上のコードトーンピッチクラスセットのコード構成音に、上記算出されたコード音程が含まれるか否かを判定する。

10

【0036】

図7（b）のテンションノートテーブル内の各行に対応するピッチクラスセット（以下、「テンションノートピッチクラスセット」と呼ぶ）は、その右端に記載されているコードタイプについて、そのコード根音が第0音（第0ビット目）の音階構成音として与えられたときに、どの音階構成音がそのコードタイプに対するテンションであることを記憶する。例えば、図7（b）に例示されるテンションノートテーブルの1行目において、テンションノートピッチクラスセット「001001000100」は、第2音（第2ビット目）、第6音（第6ビット目）、および第9音（第9ビット目）がコードタイプ「MAJ」（コード根音＝C）に対するテンションであることを表わしている。

20

【0037】

図1のコード進行選択部102は、現在ノートの発音タイミングにおける評価対象のコード進行データ中のコードタイプに対応する図7（b）に例示されるテンションノートテーブル上のテンションノートピッチクラスセットのテンションノートに、前述した現在ノートのピッチのコード根音に対するコード音程が含まれるか否かを判定する。

30

【0038】

図7（c）のスケールノートテーブル内の各行に対応するピッチクラスセット（以下、「スケールノートピッチクラスセット」と呼ぶ）は、その右端に記載されているスケールについて、そのスケールの根音が第0音（第0ビット目）の音階構成音として与えられたときに、どの音階構成音がそのスケールに対応するスケール構成音であることを記憶する。例えば、図7（c）に例示されるスケールノートテーブルの1行目において、スケールノートピッチクラスセット「101010110101」は、第0音（第0ビット目）、第2音（第2ビット目）、第4音（第4ビット目）、第5音（第5ビット目）、第7音（第7ビット目）、第9音（第9ビット目）、および第11音（第11ビット目）がスケール「ダイアトニック」のスケール構成音であることを表している。

40

【0039】

図1のコード進行選択部102は、現在ノートのピッチが、その現在ノートの発音タイミングに対応する評価対象のコード進行データ中のキーに対して、どの音程（以下、これを「キー音程」と呼ぶ）を有するかを算出する。このとき、コード進行選択部102は、コード音程の算出の場合と同様に、現在ノートのピッチを、その現在ノートの発音タイミングに対応する評価対象のコード進行データ中のキーを第0音の音階構成音としたときの、第0音から第11音までの1オクターブ内の音階構成音のいずれかに写像させる演算を行い、その写像位置の音を、上記キー音程として算出する。そして、コード進行選択部102は、上記発音タイミングにおける評価対象のコード進行データ中のスケールに対応する図7（c）に例示されるスケールノートテーブル上のスケールノートピッチクラスセッ

50

トのスケール構成音に、上記算出されたキー音程が含まれるか否かを判定する。

【 0 0 4 0 】

以上のようにして、コード進行選択部 1 0 2 は、入力モチーフ 1 0 8 の現在ノートの発音タイミングにおける評価対象のコード進行データ中のコードタイプに対応する図 7 (a) に例示されるコードトーンテーブル上のコードトーンピッチクラスセットのコード構成音にコード音程が含まれるか否かを判定する。また、コード進行選択部 1 0 2 は、上記コードタイプに対応する図 7 (b) に例示されるテンションノートテーブル上のテンションノートピッチクラスセットのテンションノートにコード音程が含まれるか否かを判定する。さらに、コード進行選択部 1 0 2 は、評価対象のコード進行データ中のスケールに対応する図 7 (c) に例示されるスケールノートテーブル上のスケールノートピッチクラスセットのスケール構成音にキー音程が含まれるか否かを判定する。そして、コード進行選択部 1 0 2 は、これらの判定に基づいて、現在ノートが、コードトーン、アヴェイラブルノート、スケールノート、テンションノート、またはアヴォイドノートのいずれに該当するか、すなわちノートタイプの情報を取得する。ノートタイプ取得処理の詳細については、図 2 2 の説明において詳述する。

【 0 0 4 1 】

図 8 (a) は、図 3 (a) に例示される入力モチーフ 1 0 8 の各ノートのピッチ (図 8 (a) 中の灰色の部分) ごとに、図 1 の伴奏・コード進行 D B 1 0 3 から読み出される図 3 (b) に例示される # 0、# 1、# 2 の 3 つの評価対象のコード進行データの例のそれぞれに対して、コード進行選択部 1 0 2 が取得するノートタイプの例を示す図である。図 8 (a) において、「C」はコードトーン、「A」はアヴェイラブルノート、「S」はスケールノート、「V」はアヴォイドノートの、ノートタイプをそれぞれ示す値である。また、図示していないが、「T」はテンションノートのノートタイプを示す値である。なお、この図では、表記の簡略化のために、各ノートタイプを示す値をアルファベット 1 文字で表しているが、実際のメモリに記憶される各ノートタイプの値としては例えば、コードトーンを示す定数値として `ci_ChordTone` (表記「C」と等価)、アヴェイラブルノートを示す定数値として `ci_AvailableNote` (表記「A」と等価)、スケールノートを示す定数値として `ci_ScaleNote` (表記「S」と等価)、テンションノートを示す定数値として `ci_TensionNote` (表記「T」と等価)、アヴォイドノートを示す定数値として `ci_AvoidNote` (表記「V」と等価) が用いられる (後述する図 1 5 A 参照)。

【 0 0 4 2 】

次に、コード進行選択部 1 0 2 は、入力モチーフ 1 0 8 の各ノートのピッチごとに、隣接するピッチ間の半音単位の音程 (以下、「隣接音程」と呼ぶ) を算出する。図 8 (b) の「隣接音程」は、入力モチーフ 1 0 8 各ノートのピッチ (図 8 (b) 中の灰色の部分) 間の音程の算出結果の例を示す図である。

【 0 0 4 3 】

コード進行選択部 1 0 2 は、評価対象のコード進行データに対して、上述のように算出したノートタイプと隣接音程が交互に格納された配列変数データ (以下、この配列変数データを「`incon[i]`」(「i」は配列番号)と記載する)を生成する。図 8 (c) は、図 1 の伴奏・コード進行 D B 1 0 3 から読み出される図 3 (b) に例示される # 0、# 1、# 2 の 3 つの評価対象のコード進行データの例のそれぞれに対して算出された配列変数データ `incon[i]` の例を示す図である。図 8 (c) のコード進行 # 0、# 1、# 2 のそれぞれの配列変数データ `incon[i]` において、偶数番目の配列番号 $i = 0, 2, 4, 6, 8, 10, 12, 14, 16, 18$ の各要素には、図 8 (a) のコード進行 # 0、# 1、# 2 のそれぞれのノートタイプが先頭から順次コピーされる。また、コード進行 # 0、# 1、# 2 のそれぞれの配列変数データ `incon[i]` において、奇数番目の配列番号 $i = 1, 3, 5, 7, 9, 11, 13, 15, 17$ の各要素にはともに、図 8 (b) の隣接音程が先頭から順次コピーされる。

【 0 0 4 4 】

次に、コード進行選択部 102 は、現在の評価対象のコード進行データに対して上述のように算出した入力モチーフ 108 の各ノートのノートタイプと隣接音程を格納した配列変数データ `incon[i]` ($i = 0, 1, 2, 3, \dots$) において、配列番号 0 から順に例えば 4 組ずつ、ノートタイプと隣接音程の組合せの規則（以下、この規則を「ノート接続ルール」と呼ぶ）を評価するノート接続性チェック処理を実行する。このノート接続性チェック処理において、コード進行選択部 102 は、図 1 のルール DB 104 に記憶されているノート接続ルールを参照する。

【0045】

図 9 は、ルール DB 104 に記憶されるノート接続ルールのデータ構成例を示す図である。ノート接続ルールには、3 音のルールと 4 音のルールがあり、説明の便宜上、それぞれに例えば、「コードトーン」、「刺繍音」、「経過音」、「倚音」、「逸音」などの名称を付けてある。また、各ノート接続ルールには、メロディを形成する上でどの程度適合しているかを評価するための評価点が付与されている。さらに、本実施形態では、ノート接続ルールを示す変数として、`ci_NoteConnect[j][2k]` ($0 \leq k < 3$) および `ci_NoteConnect[j][2k+1]` ($0 \leq k < 2$) という配列変数データを用いる。ここで、変数データ「j」は、ルール DB 104 における j 番目（図 9 中では j 行目）のノート接続ルールのデータを指す。また、変数データ「k」は、0 から 3 までの値をとる。そして、`ci_NoteConnect[j][2k] = ci_NoteConnect[j][0]`、`ci_NoteConnect[j][2] = ci_NoteConnect[j][4]`、`ci_NoteConnect[j][6]` にはそれぞれ、j 番目のノート接続ルールにおける 1 ノート目（ノートタイプ # 0）、2 ノート目（ノートタイプ # 1）、3 ノート目（ノートタイプ # 2）、および 4 ノート目（ノートタイプ # 3）の各ノートタイプが格納される。なお、4 ノート目（ノートタイプ # 3）が「`ci_NullNoteType`」となっている $j = 0$ から $j = 8$ までのノート接続ルールは、4 ノート目のノートタイプは無いことを示しており、実質的に 3 音からなるノート接続ルールであることを示している。また、`ci_NoteConnect[j][2k+1] = ci_NoteConnect[j][1]`、`ci_NoteConnect[j][3] = ci_NoteConnect[j][5]` にはそれぞれ、j 番目のノート接続ルールにおける 1 ノート目（# 0）と 2 ノート目（# 1）の隣接音程、2 ノート目（# 1）と 3 ノート目（# 2）の隣接音程、および 3 ノート目（# 2）と 4 ノート目（# 3）の隣接音程が格納される。隣接音程の数値は、半音単位の音程を示し、プラス値は音程が上がることを示し、マイナス値は音程が下がることを示す。また、値「99」は、音程がどの値でもよいことを示し、値「0」は音程が変化しないことを示す。なお、4 ノート目（ノートタイプ # 3）が「`ci_NullNoteType`」となっている $j = 0$ から $j = 8$ までのノート接続ルールは、前述したように 4 ノート目のノートタイプは無い（値が「`ci_NullNoteType`」である）ため、3 ノート目（# 2）と 4 ノート目（# 3）の隣接音程が格納される `ci_NoteConnect[j][5]` の値は「0」とされる。最後の、`ci_NoteConnect[j][7]` には、j 番目のノート接続ルールの評価点が格納される。

【0046】

以上のようなデータ構成を有するノート接続ルールとして、図 9 に例示されるように $j = 0$ から $j = 17$ までの 18 ルールが、図 1 のルール DB 104 に予め登録されている。

【0047】

コード進行選択部 102 は、上記構成を有するノート接続ルールを用いて、ノート接続性チェック処理を実行する。コード進行選択部 102 は、図 10 (a) に例示される 2 小節分の入力モチーフ 108 の先頭のノートから順に、図 10 (b) の $i = 0 \sim 6$ に示されるように 4 ノートずつ、各ノートに対応して配列変数データ `incon[i]` に格納されているノートタイプと隣接音程の組と、 $j = 0$ から $j = 17$ までのノート接続ルールにより $j = 0$ から順に選択した 1 組のノート接続ルールのノートタイプと隣接音程の組とが一致するか否かを比較する。

10

20

30

40

50

【 0 0 4 8 】

例えば、コード進行選択部 1 0 2 は、図 1 0 (b) の $i = 0$ では、 $i = 0$ の右横の矢印で示されるように、入力モチーフ 1 0 8 の第 1、2、3、4 ノート目 (図中では 1 音、2 音、3 音、4 音目) のノートタイプおよび隣接音程の各組が、図 9 に例示される $j = 0$ 、1、2、3、・・・の各ノート接続ルールの 4 組のノートタイプおよび隣接音程の組と一致するか否かを比較する。

【 0 0 4 9 】

まず、図 9 に例示される $j = 0$ のノート接続ルールでは、# 0、# 1、および # 2 のノートタイプがともにコードトーン (`ci_ChordTone`) となる。これに対して、例えば評価対象のコード進行データが図 3 (b) に例示される # 0 のコード進行である場合には、図 3 (a) に対応する図 1 0 (a) の入力モチーフ 1 0 8 に対応するノートタイプと隣接音程の配列変数データ `inconn[i]` は、図 8 の説明で前述したように、図 1 0 (c) のコード進行 # 0 の右横に示されるデータとなる。従って、入力モチーフ 1 0 8 の第 1、2、3、4 ノート目のノートタイプは、コードトーン (C)、アヴェイラブルノート (A)、コードトーン (C) となって、 $j = 0$ のノート接続ルールとは一致しない。この場合には、 $j = 0$ のノート接続ルールの評価点は加算されない。

【 0 0 5 0 】

次に、図 9 に例示される $j = 1$ のノート接続ルールでは、# 0、# 1、および # 2 のノートタイプが、コードトーン (`ci_ChordTone`)、アヴェイラブルノート (`ci_AvailableNote`)、コードトーン (`ci_ChordTone`) となる。これに対して、例えば評価対象のコード進行データが図 3 (c) に例示される # 0 のコード進行である場合には、図 1 0 (c) のコード進行 # 0 の右横に示されるノートタイプと隣接音程の配列変数データ `inconn[i]` より得られる、入力モチーフ 1 0 8 の第 1、2、3、4 ノート目のノートタイプと一致する。しかし、 $j = 1$ のノート接続ルールにおける第 1 音 (# 0) と第 2 音 (# 1) の隣接音程は「 - 1」、第 2 音 (# 1) と第 3 音 (# 2) の隣接音程は「 1」であり、これは、図 1 0 (c) のコード進行 # 0 の右横に示されるノートタイプと隣接音程の配列変数データ `inconn[i]` より得られる、入力モチーフ 1 0 8 の第 1 音と第 2 音間の隣接音程「 - 2」および第 2 音と第 3 音間の隣接音程「 2」と一致しない。従って、 $j = 1$ の場合も $j = 0$ の場合と同様に、ノート接続ルールの評価点は加算されない。

【 0 0 5 1 】

次に、図 9 に例示される $j = 2$ のノート接続ルールでは、# 0、# 1、および # 2 のノートタイプが、コードトーン (`ci_ChordTone`)、アヴェイラブルノート (`ci_AvailableNote`)、コードトーン (`ci_ChordTone`) となる。これに対して、例えば評価対象のコード進行データが図 3 (c) に例示される # 0 のコード進行である場合には、図 1 0 (c) のコード進行 # 0 の右横に示されるノートタイプと隣接音程の配列変数データ `inconn[i]` より得られる、入力モチーフ 1 0 8 の第 1、2、3、4 ノート目のノートタイプと一致する。また、 $j = 1$ のノート接続ルールにおける第 1 音 (# 0) と第 2 音 (# 1) の隣接音程は「 - 2」、第 2 音 (# 1) と第 3 音 (# 2) の隣接音程は「 2」であり、これは、図 1 0 (c) のコード進行 # 0 の右横に示されるノートタイプと隣接音程の配列変数データ `inconn[i]` より得られる、入力モチーフ 1 0 8 の第 1 音と第 2 音間の隣接音程および第 2 音と第 3 音間の隣接音程と一致する。さらに、 $j = 2$ のノート接続ルールの 4 ノート目 (ノートタイプ # 3) は、ノートタイプが無いことを示す値「`ci_NullNoteType`」であるため、入力モチーフ 1 0 8 の 4 ノート目は比較しなくてよい。以上より、評価対象のコード進行データが # 0 である場合の入力モチーフ 1 0 8 の第 1、2、3 音が、図 9 の $j = 2$ のノート接続ルールと適合することがわかり、 $j = 2$ のノート接続ルールの評価点 (`ci_NoteConnect[2][7]`) = 9 0 点が、評価対象のコード進行データ # 0 に対応する総合評価点に加算される。図 1 0 (c) のコード進行 # 0 に記載されている「 < - No 2 : 9 0 - > 」の表示が、その加算処理に対応する。

10

20

30

40

50

【 0 0 5 2 】

以上のようにして、ノート接続ルールが見つかり、そのノート接続ルール以降のノート接続ルールについては、図 1 0 (b) の $i = 0$ の入力モチーフ 1 0 8 の第 1、2、3、4 ノートのノートタイプおよび隣接音程の組に対しての評価は実施されない。

【 0 0 5 3 】

図 1 0 (b) の $i = 0$ の入力モチーフ 1 0 8 の第 1、2、3、4 ノートのノートタイプおよび隣接音程の組に対する評価が終了すると、入力モチーフ 1 0 8 上の評価対象のノートが 1 つ進められ、図 1 0 (b) の $i = 1$ の状態になって、 $i = 1$ の右横の矢印で示されるように、入力モチーフ 1 0 8 の第 2、3、4、5 ノート目のノートタイプおよび隣接音程の各組が、図 9 に例示される $j = 0、1、2、3、\dots$ の各ノート接続ルールの 4 組のノートタイプおよび隣接音程の組と一致するか否かが比較される。この結果、図 1 0 (c) の評価対象のコード進行データ # 0 に対応する入力モチーフ 1 0 8 の第 2、3、4、5 ノート目のノートタイプおよび隣接音程の各組については、全てのノート接続ルールと一致せず、図 1 0 (b) の $i = 1$ の入力モチーフ 1 0 8 の第 2、3、4、5 ノートのノートタイプおよび隣接音程の組に対する評価点は 0 点となって、評価対象のコード進行データ # 0 に対応する総合評価点への加算は行われない。

10

【 0 0 5 4 】

図 1 0 (b) の $i = 1$ の入力モチーフ 1 0 8 の第 2、3、4、5 ノートのノートタイプおよび隣接音程の組に対する評価が終了すると、入力モチーフ 1 0 8 上の評価対象のノートがさらに 1 つ進められ、図 1 0 (b) の $i = 2$ の状態になって、 $i = 2$ の右横の矢印で示されるように、入力モチーフ 1 0 8 の第 3、4、5、6 ノート目のノートタイプおよび隣接音程の各組が、図 9 に例示される $j = 0、1、2、3、\dots$ の各ノート接続ルールの 4 組のノートタイプおよび隣接音程の組と一致するか否かが比較される。この結果、図 1 0 (c) の評価対象のコード進行データ # 0 に対応する入力モチーフ 1 0 8 の第 3、4、5、6 ノート目のノートタイプおよび隣接音程の各組については、図 9 の $j = 3$ のノート接続ルールが適合することがわかり、 $j = 3$ のノート接続ルールの評価点 ($c i_NoteConnect[3][7]$) = 8 0 点が、評価対象のコード進行データ # 0 に対応する総合評価点に加算される。図 1 0 (c) のコード進行 # 0 に記載されている「< -

20

$No3: 80 \quad - >$ 」の表示が、その加算処理に対応する。この結果、総合評価点は、9 0 点 + 8 0 点 = 1 7 0 点となる。

30

【 0 0 5 5 】

以降同様にして、図 1 0 (b) の $i = 7$ の入力モチーフ 1 0 8 の第 8、9、1 0 ノートのノートタイプおよび隣接音程の組に対する評価までが実行される。なお、本実施形態では評価は原則は 4 ノートずつ行われるが、最後の $i = 7$ の場合のみ、入力モチーフ 1 0 8 の 3 ノートに対して、図 9 の $j = 0$ から $j = 8$ までのノートタイプ # 3 が「 $c i_NullNoteType$ である」である 3 音のノート接続ルールが比較される。

【 0 0 5 6 】

以上のようにして、図 1 0 (c) の評価対象のコード進行データ # 0 に対応する入力モチーフ 1 0 8 の各ノートごとの評価処理が終了すると、その時点で評価対象のコード進行データ # 0 に対応して算出されている総合評価点が、その評価対象のコード進行データ # 0 の入力モチーフ 1 0 8 に対する適合度とされる。

40

【 0 0 5 7 】

例えば評価対象のコード進行データが図 3 (c) に例示される # 1 または # 2 の各コード進行である場合は、図 3 (a) に対応する図 1 0 (a) の入力モチーフ 1 0 8 に対応するノートタイプと隣接音程の配列変数データ $in_con[i]$ は、図 8 の説明で前述したように、図 1 0 (c) のコード進行 # 1 の右横に示されるデータまたは # 2 の右横に示されるデータとなる。これらの配列変数データ $in_con[i]$ についても上述したコード進行 # 0 の場合と同様の評価処理が実行される。例えば、コード進行 # 1 の場合は、図 1 0 (c) に示されるように、図 9 のノート接続ルールと適合する部分がないため、その総合評価点は 0 点となり、これがコード進行 # 1 の入力モチーフ 1 0 8 に対する適合度とな

50

る。また、コード進行#2の場合は、図10(c)に示されるように、入力モチーフ108の第5、6、7ノート目のノートタイプおよび隣接音程の各組について、図9のj=5のノート接続ルールが適合することがわかり、j=5のノート接続ルールの評価点(c i _ N o t e C o n n e c t [5] [7]) = 95点だが、評価対象のコード進行データ#2に対応する総合評価点に加算され、これがコード進行#2の入力モチーフ108に対する適合度となる。

【0058】

図1のコード進行選択部102は、以上の適合度の算出処理を、伴奏・コード進行DB103に記憶されている複数のコード進行データに対して実行し、適合度が高かった例えば上位3個のコード進行データをそれぞれ指し示す#0、#1、#2のコード進行候補指示データ109を出力する。なお、以上の処理において、入力モチーフ108と伴奏・コード進行DB103中の各コード進行データとは、キーが必ずしも一致しているとは限らないため、各コード進行データを1オクターブを構成する12段階にキーシフトさせたデータが、入力モチーフ108と比較される。

【0059】

次に、図1のメロディ生成部105の動作の概略について説明する。まず、図11は、図1のフレーズセットDB106のデータ構成例を示す図である。図11(a)に示されるように、フレーズセットDB106には、#0、#1、・・・という複数のフレーズセットデータのレコードが記憶され、最後に終端コードが記憶される。

【0060】

1レコード分のフレーズセットデータは、図11(b)に示されるように、Aメロデータ、Bメロデータ、Cメロ(サビメロディ)データ、エンディング1データ、エンディング2データの、複数のフレーズデータから構成される。

【0061】

図11(b)の各フレーズデータは、図11(c)に示されるように、#0、#1、・・・という複数のノートデータによって構成され、最後に終端コードが記憶される。各ノートデータは、各フレーズを構成する1小節分以上の音符のそれぞれに対応し、各フレーズのメロディ音の発音を指示するデータである。図11(d)に示されるように、1つのノートデータは、そのノートデータに対応する音符の発音タイミングを例えばフレーズの先頭からの経過時間として示す「時間」データと、音符の長さを示す「長さ」データと、音符の強さを示す「強さ」データと、音符の音高を示す「ピッチ」データとから構成される。これらのデータによって、フレーズを構成する各音符が表現される。

【0062】

図1のメロディ生成部105は、コード進行選択部102が出力した#0、#1、#2のコード進行候補指示データ109に対応する3つのコード進行候補のうちの1つから、ユーザ指定によりまたは自動的に選択されたコード進行候補に対応する曲構造データ(図6参照)を、伴奏・コード進行DB103から読み込む。メロディ生成部105は、この曲構造データによって示される小節のフレーズごとに、入力モチーフ108とフレーズセットDB106に登録されているフレーズセット(図11参照)、およびルールDB104(図9参照)を参照しながら、そのフレーズのメロディを自動生成する。

【0063】

この場合、メロディ生成部105は、曲構造データによって示される小節のフレーズが、入力モチーフ108が入力されたフレーズであるか否かを判定し、入力モチーフ108のフレーズである場合は、その入力モチーフ108のメロディをそのまま図1のメロディ110の一部として出力する。

【0064】

メロディ生成部105は、曲構造データによって示される小節のフレーズが、入力モチーフ108のフレーズでもなく、サビメロディの先頭フレーズでもない場合は、該当するフレーズのメロディが未だ生成されていなければ、フレーズセットDB106から入力モチーフ108に対応するフレーズセットを抽出し、そのフレーズセット内の該当するフレ

10

20

30

40

50

ーズのメロディをコピーし、生成済みであればその生成済みのフレーズからメロディをコピーする。そして、メロディ生成部105は、コピーしたメロディを変形する後述するメロディ変形処理と、さらにその変形したメロディを構成する各ノートのピッチを最適化する後述するメロディ最適化処理を実行して、曲構造データによって示される小節のフレーズのメロディを自動生成し、メロディ110の一部として出力する。既に生成済みのフレーズからメロディをコピーする処理の詳細については、図25の説明において後述する。

【0065】

メロディ生成部105は、曲構造データによって示される小節のフレーズが、サビメロディの先頭フレーズである場合は、該当するサビメロディの先頭フレーズが生成済みでなければ、フレーズセットDB106から入力モチーフ108に対応するフレーズセットを抽出し、そのフレーズセット内の該当するサビメロディ(Cメロ)の先頭フレーズのメロディをコピーし、そのメロディを構成する各ノートのピッチを最適化するメロディ最適化処理を実行して、サビメロディの先頭フレーズのメロディを自動生成し、メロディ110の一部として出力する。一方、該当するサビメロディの先頭フレーズが生成済みならば、その生成済みのフレーズからメロディをコピーし、メロディ110の一部として出力する。

【0066】

図12は、メロディ変形処理およびメロディ最適化処理の動作説明図である。予め生成されているメロディがあるときに、メロディ生成部105は、そのメロディをコピーして、例えば1201に示されるように、コピーしたメロディを構成する各ノートのピッチを、例えば2半音上にピッチシフトする処理を実行する。あるいは、メロディ生成部105は、例えば1202に示されるように、コピーしたメロディを構成する各ノートを、小節内で左右(再生順序)を反転させる処理を実行する。メロディ生成部105は、このようなメロディ変形処理を実行した小節のメロディに対して、さらに1203または1204として示されるメロディ最適化処理を実行して、最終的なメロディを自動生成する。

【0067】

図13は、メロディ最適化処理の詳細動作説明図である。いま、変数*iNoteCnt*には、メロディ変形処理を実行した小節のメロディを構成するノートの数が格納されており、配列データ*note[0] -> iPitch*、*note[1] -> iPitch*、*note[2] -> iPitch*、・・・、*note[iNoteCnt-2] -> iPitch*、*note[iNoteCnt-1] -> iPitch*には、上記各ノートのピッチデータが格納されているとする。メロディ生成部105はまず、各ノートのピッチデータ*note[i] -> iPitch*(0 ≤ *i* ≤ *iNoteCnt* - 1)をそれぞれ、*ipitd[0] = 0*、*ipitd[1] = 1*、*ipitd[2] = -1*、*ipitd[3] = 2*、*ipitd[4] = -2*という5段階の値だけピッチシフトさせ、合計5^{*iNoteCnt*}通りのピッチ列を生成する。そして、メロディ生成部105は、各ピッチ列ごとに、図7から図10を用いて前述したのと同様の処理によって、コード進行選択部102が抽出しているコード進行データの上記小節に対応する部分について、ノートタイプの取得と、隣接音程の計算を実行し、ノート接続性チェック処理を実行する。この結果、メロディ生成部105は、合計5^{*iNoteCnt*}通りのピッチ列に対して算出した適合度のうち、もっとも適合度が高いピッチ列を、その小節の各ノートのピッチデータ*note[i] -> iPitch*(0 ≤ *i* ≤ *iNoteCnt* - 1)として修正する。メロディ生成部105は、このようにして生成したピッチ列を含むその小節の各ノートのデータ*note[i]*(0 ≤ *i* ≤ *iNoteCnt* - 1)をメロディ110として出力する。

【0068】

上述した自動作曲装置100のさらに詳細な構成および動作について、以下に説明する。図14は、図1の自動作曲装置100のハードウェア構成例を示す図である。図14に例示される自動作曲装置100のハードウェア構成は、CPU(中央演算処理装置)1401、ROM(リードオンリーメモリ)1402、RAM(ランダムアクセスメモリ)1403、入力部1404、表示部1405、および音源部1406を備え、それらがシス

10

20

30

40

50

テムバス 1408 によって相互に接続された構成を有する。また、音源部 1406 の出力はサウンドシステム 1407 に入力する。

【0069】

CPU 1401 は、RAM 1403 をワークメモリとして使用しながら ROM 1402 に記憶された自動作曲制御プログラムを実行することにより、図 1 の 101 ~ 107 の各機能部分に対応する制御動作を実行する。

【0070】

ROM 1402 には、上記自動作曲制御プログラムのほか、図 1 の伴奏・コード進行 DB 103 (図 5、図 6 参照)、ルール DB 104 (図 9 参照)、フレーズセット DB 106 (図 11 参照)、および標準ピッチクラスセットテーブル (図 7 参照) が予め記憶される。

10

【0071】

RAM 1403 は、モチーフ入力部 101 から入力された入力モチーフ 108 (図 4 参照)、コード進行選択部 102 が出力するコード進行候補データ 109、メロディ生成部 105 が出力するメロディデータ 110 などを一時的に記憶する。このほか、RAM 1403 には、後述する各種変数データ等が一時的に記憶される。

【0072】

入力部 1404 は、図 1 のモチーフ入力部 101 の一部の機能に対応し、例えば、鍵盤入力部 101-1、音声入力部 101-2、または音符入力部 101-3 に対応する。入力部 1404 が鍵盤入力部 101-1 を備える場合には、演奏鍵盤と、当該演奏鍵盤の押鍵状態を検知しシステムバス 1408 を介して CPU 1401 に通知するキーマトリクス回路を備える。入力部 1404 が音声入力部 101-2 を備える場合には、歌声入力用のマイクと、当該マイクから入力された音声信号をデジタル信号に変換した後、歌声のピッチ情報を抽出しシステムバス 1408 を介して CPU 1401 に通知するデジタル信号処理回路を備える。なお、ピッチ情報の抽出は、CPU 1401 が実行してもよい。入力部 1404 が音符入力部 101-3 を備える場合には、音符入力用のキーボードと、当該キーボードの音符入力状態を検知しシステムバス 1408 を介して CPU 1401 に通知するキーマトリクス回路を備える。CPU 1401 は、図 1 のモチーフ入力部 101 の一部の機能に対応し、図 14 の入力部 1404 から入力した上記各種情報に基づいて、入力モチーフ 108 を検出して RAM 1403 に記憶する。

20

30

【0073】

表示部 1405 は、CPU 1401 による制御動作とともに、モチーフの入力を図 1 の出力部 107 が備える楽譜表示部 107-1 の機能を実現する。CPU 1401 は、自動作曲されたメロディデータ 110 に対応する楽譜データを生成し、その楽譜データの表示を表示部 1405 に指示する。表示部 1405 は、例えば液晶ディスプレイ装置である。

【0074】

音源部 1406 は、CPU 1401 による制御動作とともに、図 1 の楽音再生部 107-2 の機能を実現する。CPU 1401 は、自動生成されたメロディデータ 110 と伴奏・コード進行 DB 103 から読み出された伴奏用 MIDI データとに基づいて、メロディおよび伴奏を再生するための発音制御データを生成し、音源部 1406 に供給する。音源部 1406 は、この発音制御データに基づいて、メロディ音および伴奏音を生成し、サウンドシステム 1407 へ出力する。サウンドシステム 1407 は、音源部 1406 から入力したメロディ音および伴奏音のデジタル楽音データをアナログ楽音信号に変換した後、そのアナログ楽音信号を内蔵のアンプで増幅して内蔵のスピーカから放音する。

40

【0075】

図 15A および図 15B は、ROM 1402 または RAM 1403 に記憶される各種変数データ、配列変数データ、および定数データのリストを示す図である。これらのデータは、後述する各種処理で使用される。

【0076】

図 16 は、本実施形態における自動作曲処理の例を示すフローチャートである。この処

50

理は、自動作曲装置 100 の電源が投入されることにより、CPU 1401 が ROM 1402 に記憶されている自動作曲処理プログラムの実行を開始することによりスタートする。

【0077】

CPU 1401 はまず、RAM 1403 および音源部 1406 に対して初期化を行う（ステップ S1601）。その後、CPU 1401 は、ステップ S1602 から S1608 までの一連の処理を繰返し実行する。

【0078】

この繰返し処理において、CPU 1401 はまず、ユーザが特には図示しない電源スイッチを押下したことにより自動作曲処理の終了を指示したか否かを判定し（ステップ S1602）、終了を指示していなければ（ステップ S1602 の判定が NO）、繰返し処理を継続し、終了を指示したならば（ステップ S1602 の判定が YES）、図 16 のフローチャートで例示される自動作曲処理を終了する。

10

【0079】

ステップ S1602 の判定が NO の場合、CPU 1401 は、ユーザが入力部 1404 からモチーフ入力を指示したか否かを判定する（ステップ S1603）。ユーザがモチーフ入力を指示した場合（ステップ S1603 の判定が YES の場合）、CPU 1401 は、入力部 1404 からのユーザによるモチーフ入力を受け付け、その結果入力部 1404 から入力された入力モチーフ 108 を、例えば図 4 のデータ形式で RAM 1403 に記憶する（ステップ S1606）。その後、CPU 1401 は、ステップ S1602 の処理に戻る。

20

【0080】

ユーザがモチーフ入力を指示していない場合（ステップ S1603 の判定が NO の場合）、CPU 1401 は、ユーザが特には図示しないスイッチにより自動作曲を指示したか否かを判定する（ステップ S1604）。ユーザが自動作曲を指示した場合（ステップ S1604 の判定が YES の場合）、CPU 1401 は、コード進行選択処理（ステップ S1607）、続いてメロディ生成処理（ステップ S1608）を実行する。ステップ S1607 のコード進行選択処理は、図 1 のコード進行選択部 102 の機能を実現する。ステップ S1608 のメロディ生成処理は、図 1 のメロディ生成部 105 の機能を実現する。その後、CPU 1401 は、ステップ S1602 の処理に戻る。

30

【0081】

ユーザが自動作曲を指示していない場合（ステップ S1604 の判定が NO の場合）、CPU 1401 は、ユーザが特には図示しないスイッチにより自動作曲されたメロディ 110 の再生を指示したか否かを判定する（ステップ S1605）。ユーザがメロディ 110 の再生を指示した場合（ステップ S1605 の判定が YES の場合）、CPU 1401 は、再生処理（ステップ S1609）を実行する。この処理は、図 1 の出力部 107 内の楽譜表示部 107-1 および楽音再生部 107-2 の動作として前述した通りである。

【0082】

ユーザが自動作曲を指示していない場合（ステップ S1604 の判定が NO の場合）、CPU 1401 は、ステップ S1602 の処理に戻る。

40

【0083】

図 17 は、図 16 のステップ S1607 のコード進行選択処理の詳細例を示すフローチャートである。

【0084】

まず、CPU 1401 は、RAM 1403 上の変数データおよび配列変数データを初期化する（ステップ S1701）。

【0085】

次に、CPU 1401 は、伴奏・コード進行 DB 103 に記憶されている複数のコード進行データに対する繰返し処理を制御するための RAM 1403 上の変数 n を「0」に初期化する。その後、CPU 1401 は、ステップ S1714 で変数 n の値を +1 ずつイン

50

クリメントさせながら、ステップS 1 7 0 3で変数nの値がROM 1 4 0 2に記憶されている定数データMAX__CHORD__PROGの値よりも小さいと判定される間、ステップS 1 7 0 4からS 1 7 1 3までの一連の処理を実行する。定数データMAX__CHORD__PROGの値は、伴奏・コード進行DB 1 0 3に記憶されるコード進行データの数を示す定数データである。CPU 1 4 0 1は、図5に示される伴奏・コード進行DB 1 0 3のレコード数の分だけ、ステップS 1 7 0 4からS 1 7 1 3までの一連の処理を繰り返し実行することにより、適合度の算出処理を、伴奏・コード進行DB 1 0 3に記憶されている複数のコード進行データに対して実行し、入力モチーフ1 0 8との適合度が高かった例えば上位3個のコード進行データをそれぞれ指し示す# 0、# 1、# 2のコード進行候補指示データ1 0 9を出力する。

10

【0086】

ステップS 1 7 0 3からS 1 7 1 3の繰返し処理において、ステップSCPU 1 4 0 1はまず、変数nの値が定数データMAX__CHORD__PROGの値よりも小さいか否かを判定する(ステップS 1 7 0 3)。

【0087】

ステップS 1 7 0 3の判定がYESならば、CPU 1 4 0 1は、変数データnが示すn番目のコード進行データ# n(図5(a)参照)を、伴奏・コード進行DB 1 0 3からRAM 1 4 0 3内のコード進行データ領域に読み込む(ステップS 1 7 0 4)。このコード進行データ# nのデータ形式は、例えば図5の(b)、(c)、(d)で示されるフォーマットを有する。

20

【0088】

次に、CPU 1 4 0 1は、伴奏・コード進行DB 1 0 3からRAM 1 4 0 3内のコード進行データ# n用の配列変数データ要素iChordAttribute[n][0]に読み込まれた、コード進行データ# nの楽曲ジャンルを示す値が、予め特には図示しないスイッチによりユーザによって設定され、RAM 1 4 0 3内の変数データiJungleSelectに記憶されている楽曲ジャンルを示す値と等しいか否かを判定する(ステップS 1 7 0 5)。ステップS 1 7 0 5の判定がNOならば、そのコード進行データ# nは、ユーザが望む楽曲ジャンルに合わないため、選択せずに、ステップS 1 7 1 4に進む。

【0089】

ステップS 1 7 0 5の判定がYESならば、CPU 1 4 0 1は、伴奏・コード進行DB 1 0 3からRAM 1 4 0 3内のコード進行データ# n用の配列変数データ要素iChordAttribute[n][1]に読み込まれた、コード進行データ# nのコンセプトを示す値が、予め特には図示しないスイッチによりユーザによって設定され、RAM 1 4 0 3内の変数データiConceptSelectに記憶されている楽曲のコンセプトを示す値と等しいか否かを判定する(ステップS 1 7 0 6)。ステップS 1 7 0 6の判定がNOならば、そのコード進行データ# nは、ユーザが望む楽曲コンセプトに合わないため、選択せずに、ステップS 1 7 1 4に進む。

30

【0090】

ステップS 1 7 0 6の判定がYESならば、CPU 1 4 0 1は、コードデザインデータ作成処理を実行する(ステップS 1 7 0 7)。この処理において、CPU 1 4 0 1は、コード進行データ# nによって、時間経過に沿って順次指定されるコード進行の情報を、RAM 1 4 0 3上の配列変数データである後述するコードデザインデータcdesign[k]に格納する処理を実行する。

40

【0091】

次に、CPU 1 4 0 1は、RAM 1 4 0 3上の変数データiKeyShiftに初期値「0」を格納する(ステップS 1 7 0 8)。この変数データiKeyShiftは、1オクターブの半音階中で、初期値「0」からROM 1 4 0 2に記憶されている定数データPITCH__CLASS__Nより1小さい数までの範囲で、コード進行データ# nに対する半音単位のキーシフト値を指定する。定数データPITCH__CLASS__Nの値は、通常は1オクターブ内の半音数12である。

50

【 0 0 9 2 】

次に、CPU 1 4 0 1 は、変数データ *iKeyShift* の値が定数データ *PITCH__CLASS__N* の値よりも小さいか否かを判定する（ステップ S 1 7 0 9 ）。

【 0 0 9 3 】

ステップ S 1 7 0 9 の判定が Y E S ならば、変数データ *iKeyShift* が示すキーシフト値だけコード進行データ # *n* のキーをシフトさせた後、入力モチーフ 1 0 8 とコード進行 # *n* に対する適合度チェック処理を実行する（ステップ S 1 7 1 0 ）。この処理により、入力モチーフ 1 0 8 に対するコード進行 # *n* の適合度が RAM 1 4 0 3 上の変数データ *doValue* に得られる。

【 0 0 9 4 】

次に、CPU 1 4 0 1 は、変数データ *doValue* の値が、RAM 1 4 0 3 上の変数データ *doMaxValue* よりも大きいかなかを判定する（ステップ S 1 7 1 1 ）。変数データ *doMaxValue* は、現時点で最も高い適合度の値を格納する変数で、ステップ S 1 7 0 1 で値「0」に初期化されている。

【 0 0 9 5 】

ステップ S 1 7 1 1 の判定が Y E S ならば、CPU 1 4 0 1 は、変数データ *doMaxValue* の値を変数データ *doValue* の値で置き換える。また、CPU 1 4 0 1 は、RAM 1 4 0 3 内の配列変数データ *iBestKeyShift[iBestUpdate]* に、変数データ *iKeyShift* の現在値を格納する。また、CPU 1 4 0 1 は、RAM 1 4 0 3 内の配列変数データ *iBestChordProg[iBestUpdate]* に、伴奏・コード進行 DB 1 0 3 上のコード進行データを指し示す変数データ *n* の現在値を格納する。その後、CPU 1 4 0 1 は、RAM 1 4 0 3 内の変数データ *iBestUpdate* を + 1 インクリメントする（以上、ステップ S 1 7 1 2 ）。変数データ *iBestUpdate* は、ステップ S 1 7 0 1 で値「0」に初期化された後、現時点で適合度が最も高いコード進行データが見つかるごとにインクリメントされるデータであり、その値が大きいほど上位の適合度であることを示す。配列変数データ *iBestKeyShift[iBestUpdate]* は、変数データ *iBestUpdate* が示す順位におけるキーシフト値を保持する。配列変数データ *iBestChordProg[iBestUpdate]* は、変数データ *iBestUpdate* が示す順位における伴奏・コード進行 DB 1 0 3 上のコード進行の番号を保持する。

【 0 0 9 6 】

ステップ S 1 7 1 1 の判定が N O ならば、CPU 1 4 0 1 は、上記ステップ S 1 7 1 2 の処理はスキップして、今回のコード進行データ # *n* は入力モチーフ 1 0 8 に対する自動作曲用のコード進行データとしては選択しない。

【 0 0 9 7 】

その後、CPU 1 4 0 1 は、変数データ *iKeyShift* の値を + 1 インクリメントする（ステップ S 1 7 1 3 ）。その後、CPU 1 4 0 1 は、ステップ S 1 7 0 9 の処理に戻る。

【 0 0 9 8 】

CPU 1 4 0 1 は、変数データ *iKeyShift* の値をインクリメントしながらステップ S 1 7 0 9 から S 1 7 1 3 までの処理を繰り返し実行した後、1 オクターブ分のキーシフト値の指定が終了してステップ S 1 7 0 9 の判定が N O になると、ステップ S 1 7 1 4 に処理を進める。ステップ S 1 7 1 4 で、CPU 1 4 0 1 は、伴奏・コード進行 DB 1 0 3 上のコード進行データの選択用の変数データ *n* を + 1 インクリメントする。その後、CPU 1 4 0 1 は、ステップ S 1 7 0 3 の処理に戻る。

【 0 0 9 9 】

CPU 1 4 0 1 は、変数データ *n* の値をインクリメントしながらステップ S 1 7 0 3 から S 1 7 1 4 までの一連の処理を繰り返し実行した後、伴奏・コード進行 DB 1 0 3 内の全てのコード進行データに対する処理を終了してステップ S 1 7 0 3 の判定が N O になると、図 1 7 のフローチャートの処理すなわち図 1 6 のステップ S 1 6 0 7 のコード進行選

10

20

30

40

50

択処理を終了する。この結果、変数データ `iBestUpdate` の現在値よりも 1 だけ小さい値「`iBestUpdate - 1`」を要素番号とする配列変数データ `iBestKeyShift[iBestUpdate - 1]` および `iBestChordProg[iBestUpdate - 1]` に、入力モチーフ 108 に対して最も適合性が高いキーシフト値とコード進行データの番号が格納される。また、配列変数データ `iBestKeyShift[iBestUpdate - 2]` および `iBestChordProg[iBestUpdate - 2]` に、入力モチーフ 108 に対して 2 番目に適合性が高いキーシフト値とコード進行データの番号が格納される。さらに、配列変数データ `iBestKeyShift[iBestUpdate - 3]` および `iBestChordProg[iBestUpdate - 3]` に、入力モチーフ 108 に対して 3 番目に適合性が高いキーシフト値とコード進行データの番号が格納される。これらのデータセットが、上位から順に図 1 の # 0、# 1、および # 2 のコード進行候補指示データ 109 に対応する。

【0100】

図 18 は、図 17 のステップ S 1707 のコードデザインデータ作成処理の詳細例を示すフローチャートである。

【0101】

まず、CPU 1401 は、コード進行情報の番号を示す変数データ `iCDesignCnt` を初期値「0」に設定する（ステップ S 1801）。

【0102】

次に、CPU 1401 は、図 17 のステップ S 1704 で伴奏・コード進行 DB 103 から RAM 1403 に例えば図 5 (b)、(c)、(d) のデータ形式で読み込まれたコード進行データ # n の最初のメタイベント（図 5 (b) のコードデータ # 0 に対応）へのポインタを、RAM 1403 内のポインタ変数データ `mt` に格納する（ステップ S 1802）。

【0103】

次に、CPU 1401 は、ステップ S 1811 でポインタ変数データ `mt` に順次次のメタイベント（図 5 (b) のコードデータ # 1、# 2、・・・）へのポインタを格納しながら、ステップ S 1803 で終端（図 5 (b) の「終端」）に達したと判定するまで、ステップ S 1803 から S 1811 の一連の処理を、コード進行データ # n の各コードデータ（図 5 (b) 参照）に対して、繰り返し実行する。

【0104】

上記繰り返し処理において、CPU 1401 はまず、ポインタ変数データ `mt` が終端を指しているか否かを判定する（ステップ S 1803）。

【0105】

ステップ S 1803 の判定が NO ならば、CPU 1401 は、ポインタ変数データ `mt` が指すコードデータ（図 5 (b)）中のコード根音（ルート）とコードタイプ（図 5 (d) 参照）を抽出して、RAM 1403 内の変数データ `root` と `type` に格納することを試みる（ステップ S 1804）。そして、CPU 1401 は、ステップ S 1804 での格納処理に成功したか否かを判定する（ステップ S 1805）。

【0106】

ステップ S 1804 での格納処理に成功した場合（ステップ S 1805 の判定が YES の場合）、CPU 1401 は、ポインタ変数データ `mt` が指す記憶エリアの時間情報 `mt -> iTime`（図 5 (d) の「時間」データ）を、変数データ `iCDesignCnt` の現在値を要素番号とするコードデザインデータの時間項目 `cdesign[iCDesignCnt] -> iTime` に格納する。また、CPU 1401 は、ステップ S 1804 で変数データ `root` に格納されたコード根音情報を、変数データ `iCDesignCnt` の現在値を要素番号とするコードデザインデータのコード根音項目 `cdesign[iCDesignCnt] -> iRoot` に格納する。また、CPU 1401 は、ステップ S 1804 で変数データ `type` に格納されたコードタイプ情報を、変数データ `iCDesignCnt` の現在値を要素番号とするコードデザインデータのコードタイプ項目 `cdesign[iCDesignCnt] -> iType` に格納する。

`design[iCDesignCnt] -> iType`に格納する。さらに、変数データ*iCDesignCnt*の現在値を要素番号とするコードデザインデータのキー項目`design[iCDesignCnt] -> iKey`とスケール項目`design[iCDesignCnt] -> iScale`には、無効値「-1」を格納する（以上、ステップS1806）。その後、CPU1401は、ステップS1810の処理に移行して、変数データ*iCDesignCnt*の値を+1インクリメントする。

【0107】

ステップS1804での格納処理に成功しなかった場合（ステップS1805の判定がNOの場合）、CPU1401は、ポインタ変数データ*mt*が指すコードデータ（図5（b））中のスケールとキー（図5（c）参照）を抽出して、RAM1403内の変数データ*scale*と*key*に格納することを試みる（ステップS1807）。そして、CPU1401は、ステップS1807での格納処理に成功したか否かを判定する（ステップS1808）。

【0108】

ステップS1807での格納処理に成功した場合（ステップS1808の判定がYESの場合）、CPU1401は、ポインタ変数データ*mt*が指す記憶エリアの時間情報*mt -> iTime*（図5（c）の「時間」データ）を、変数データ*iCDesignCnt*の現在値を要素番号とするコードデザインデータの時間項目`design[iCDesignCnt] -> iTime`に格納する。また、CPU1401は、ステップS1807で変数データ*key*に格納されたキー情報を、変数データ*iCDesignCnt*の現在値を要素番号とするコードデザインデータのキー項目`design[iCDesignCnt] -> iKey`に格納する。また、CPU1401は、ステップS1807で変数データ*scale*に格納されたスケール情報を、変数データ*iCDesignCnt*の現在値を要素番号とするコードデザインデータのスケール項目`design[iCDesignCnt] -> iScale`に格納する。さらに、変数データ*iCDesignCnt*の現在値を要素番号とするコードデザインデータのコード根音項目`design[iCDesignCnt] -> iRoot`とコードタイプ項目`design[iCDesignCnt] -> iType`には、無効値「-1」を格納する（以上、ステップS1809）。その後、CPU1401は、ステップS1810の処理に移行して、変数データ*iCDesignCnt*の値を+1インクリメントする。

【0109】

CPU1401は、ステップS1810での変数データ*iCDesignCnt*の値のインクリメント処理の後、またはステップS1807での格納処理に成功しなかった場合（ステップS1808の判定がNOの場合）、ポインタ変数データ*mt*に次のメタイベント（図5（b）のコードデータ#1、#2、・・・）へのポインタを格納し（ステップS1811）、ステップS1803の判定処理に戻る。

【0110】

上記ステップS1803からS1811までの繰返し処理の結果、CPU1401は、現在のコード進行データ#*n*に対するコードデータを終端（図5（b）参照）まで読み込むと、ステップS1803の判定がYESとなって、図18のフローチャートで例示される処理すなわち図17のステップS1707のコードデザインデータ作成処理を終了する。この時点で、変数データ*iCDesignCnt*に現在のコード進行データ#*n*を構成するコード情報の数が得られ、コードデザインデータ`design[0]`から`design[iCDesignCnt - 1]`にそれぞれのコード情報が得られる。

【0111】

図19は、図17のステップS1710の入力モチーフ108とコード進行#*n*に対する適合度チェック処理の詳細例を示すフローチャートである。

【0112】

まず、CPU1401は、適合度を示す変数データ*dovalue*に初期値「0」をセットする（ステップS1901）。

【0113】

次に、CPU1401は、伴奏・コード進行DB103から、ステップS1704で読み込んだコード進行データ#nに対応する曲構造データ#n(図5(a)参照)を参照し、入力モチーフ108の入力時にユーザにより指定されたフレーズ種別と同じフレーズ種別が「PartName[M]」項目(図6参照)に指定されている先頭の小節のレコードに格納されている小節開始時間データiPartTime[M]を読み込み、RAM1403内の変数データsTimeに格納する(ステップS1902)。

【0114】

次に、CPU1401は、入力モチーフ108を構成するノートの順番を指す変数データiNoteCntの値を初期値「0」に設定する(ステップS1903)。

10

【0115】

次に、CPU1401は、図16のステップS1606でRAM1403に図4のデータ形式で入力された入力モチーフ108の最初のノートデータ(図4(a)のノートデータ#0に対応)へのポインタを、RAM1403内のポインタ変数データmeに格納する(ステップS1904)。

【0116】

次に、CPU1401は、ステップS1909でポインタ変数データmeに順次入力モチーフ108の次のノート(図4(a)のノートデータ#1、#2、...)へのポインタを格納しながら、ステップS1905で終端(図4(b)の「終端」)に達したと判定するまで、ステップS1905からS1909の一連の処理を、入力モチーフ108の各ノートデータ(図4(a)参照)に対して、繰り返し実行する。

20

【0117】

上記繰り返し処理において、CPU1401はまず、ポインタ変数データmeが終端を指しているか否かを判定する(ステップS1905)。

【0118】

ステップS1905の判定がNOならば、CPU1401は、ポインタ変数データmeが指すノートデータ(図4(b))中の「時間」データであるme->iTimeを参照し、これにステップS1902で得ている入力モチーフ108の該当小節の小節開始時間sTimeを加算し、その結果を新たにme->iTimeに上書きする(ステップS1906)。入力モチーフ108を構成する各ノートデータ中の「時間」データは、2小節からなる入力モチーフ108の先頭からの時間であるため、それを楽曲の先頭からの時間に変換するために、ステップS1902で曲構造データから得た入力モチーフ108の該当小節の小節開始時間sTimeが加算される。

30

【0119】

次に、CPU1401は、ポインタ変数データmeの値を、変数データiNoteCntの現在値を要素値とする配列変数データであるノートポインタ配列データnote[iNoteCnt]に格納する(ステップS1907)。

【0120】

その後、CPU1401は、変数データiNoteCntの値を+1インクリメントする(ステップS1908)。そして、CPU1401は、ポインタ変数データmeに入力モチーフ108中の次のノートデータ(図4(a)のノートデータ#1、#2、...)へのポインタを格納し(ステップS1909)、ステップS1905の判定処理に戻る。

40

【0121】

上記ステップS1905からS1909までの繰り返し処理の結果、CPU1401は、入力モチーフ108中のノートデータを終端(図4(a)参照)まで読み込むと、ステップS1905の判定がYESとなって、ステップS1910のチェック処理に進む。このチェック処理では、入力モチーフ108に対するコード進行#nの適合度を算出処理が実行され、この結果、適合度が変数データdovalueに得られる。その後、図19のフローチャートで例示される処理すなわち図17のステップS1710の入力モチーフ108とコード進行#nの適合度チェック処理を終了する。この時点で、変数データiNot

50

e C n tに入力モチーフ108を構成するノートの数(図3(a)の音符の数に対応)が得られ、ノートポインタ配列変数データnote[0]~note[iNoteCnt-1]にそれぞれのノートデータへのポインタが得られる。

【0122】

図20は、図19のステップS1910のチェック処理の詳細例を示すフローチャートである。

【0123】

まず、CPU1401は、入力モチーフ108のノート数をカウントするRAM1403内の変数iに初期値「0」を格納する(ステップS2001)。その後、CPU1401は、ステップS2008で変数iの値を+1ずつインクリメントさせながら、ステップS2002で変数iの値が図19の処理で最終的に得られた入力モチーフ108のノート数を示す変数データiNoteCntの値よりも小さいと判定される間、ステップS2002からS2008までの一連の処理を実行する。

【0124】

ステップS2002からS2008の繰返し処理において、ステップSCPU1401はまず、変数iの値が変数データiNoteCntの値よりも小さいか否かを判定する(ステップS2002)。

【0125】

ステップS2002の判定がYESならば、CPU1401は、変数データiによって指示されるi番目の処理対象ノートに対応するノートポインタ配列変数データnote[i]から、ピッチ項目値note[i]->iPit(図4(b)の「ピッチ」項目値を指す)を読み出し、それを変数データiの値を要素値とするRAM1403内のピッチ情報配列変数データipit[i]に格納する(ステップS2003)。

【0126】

次に、CPU1401は、入力モチーフ108の現在の処理対象ノートのタイミングに対応するコード情報の取得処理を実行する(ステップS2004)。この処理では、入力モチーフ108の現在の処理対象ノートの発音タイミングにおいて指定されるべきコードのコード根音、コードタイプ、スケール、およびキーが、変数データroot、type、scale、およびkeyに得られる。

【0127】

続いて、CPU1401は、ノートタイプの取得処理を実行する(ステップS2005)。この処理では、図8の説明で前述した、RAM1403内のノートタイプと隣接音程の配列変数データincon[i×2](偶数番目の要素)に、入力モチーフ108の現在のi番目の処理対象ノートの、ピッチipit[i]の現在の評価対象のコード進行データ#nに対するノートタイプが得られる。

【0128】

さらに、CPU1401は、変数iの値が0よりも大きいかな否か、すなわち処理対象ノートが先頭以外のノートであるかな否かを判定する(ステップS2006)。

【0129】

そして、ステップS2006の判定がYESのときに、CPU1401は、変数データiによって指示されるi番目の処理対象ノートに対応するピッチ情報ipit[i]から、i-1番目の処理対象ノートに対応するピッチ情報ipit[i-1]を減算することにより、ノートタイプと隣接音程の配列変数データincon[i×2-1](奇数番目の要素)に、図8の説明で前述した隣接音程を得る(ステップS2007)。

【0130】

ステップS2006の判定がNOのとき(先頭のノートのとき)には、CPU1401は、ステップS2007の処理はスキップする。

【0131】

その後、CPU1401は、変数iの値を+1インクリメントし(ステップS2008)、入力モチーフ108中の次のノートの処理に移行して、ステップS2002の判定処

10

20

30

40

50

理に戻る。

【0132】

CPU1401は、変数データ*i*の値をインクリメントしながらステップS2002からS2008までの一連の処理を繰り返し実行した後、入力モチーフ108を構成する全てのノートデータに対する処理を終了してステップS2002の判定がNOになると、ステップS2009のノート接続性チェック処理に進む。この時点で、配列変数データincon[i×2](0 ≤ i ≤ iNoteCnt-1)およびincon[i×2-1](1 ≤ i ≤ iNoteCnt-1)に、図8の説明等で前述したノートタイプと隣接音程の集合が得られる。そして、CPU1401は、このデータに基づいて、ステップS2009のノート接続性チェック処理により、評価対象のコード進行データ#*n*の入力モチーフ108に対する適合度を変数データdoValueに得る。その後、CPU1401は、図20のフローチャートで例示される処理すなわち図19のステップS1910のチェック処理を終了する。

10

【0133】

図21は、図20のステップS2004の入力モチーフ108の現在のノートのタイミング*n*に対応するコード情報の取得処理の詳細例を示すフローチャートである。

【0134】

まず、CPU1401は、コードデザインデータの情報数をカウントするRAM1403内の変数*k*に初期値「0」を格納する(ステップS2101)。その後、CPU1401は、ステップS2107で変数*k*の値を+1ずつインクリメントさせながら、ステップS2102で変数*k*の値が図18の処理で最終的に得られた現在の評価対象のコード進行データ#*n*を構成するコード情報の数を示す変数データiCDesignCntの値よりも小さいと判定される間、ステップS2102からS2107までの一連の処理を実行する。

20

【0135】

ステップS2102からS2107の繰り返し処理において、ステップSCPU1401はまず、変数*k*の値が変数データiCDesignCntの値よりも小さいか否かを判定する(ステップS2102)。

【0136】

ステップS2102の判定がYESならば、CPU1401は、現在の処理対象のノートのノートポインタ配列データが指す時間項目値note[i]→iTimeが、変数データ*k*によって指示される*k*番目のコードデザインデータの時間項目cdesign[k]→iTimeの値よりも大きく、*k*+1番目のコードデザインデータの時間項目cdesign[k+1]→iTimeの値よりも小さく、かつ、*k*番目のコードデザインデータのキー項目cdesign[k]→iKeyとスケール項目cdesign[k]→iScaleの各値が0以上で有意な値が設定されているか否か(図18のステップS1806、S1808参照)を判定する(ステップS2103)。

30

【0137】

ステップS2103の判定がYESならば、入力モチーフ108の現在の処理対象のノートnote[i]の発音タイミングにおいて*k*番目のコードデザインデータcdesign[k]によるコード情報が指定されていると判定できる。そこで、CPU1401は、変数データkeyとscaleに、それぞれ*k*番目のコードデザインデータのキー項目cdesign[k]→iKeyとスケール項目cdesign[k]→iScaleの各値を格納する(ステップS2104)。

40

【0138】

ステップS2103の判定がNOならば、CPU1401は、ステップS2104の処理はスキップする。

【0139】

続いて、CPU1401は、現在の処理対象のノートのノートポインタ配列データが指す時間項目値note[i]→iTimeが、変数データ*k*によって指示される*k*番目

50

のコードデザインデータの時間項目 $c d e s i g n [k] - > i T i m e$ の値よりも大きく、 $k + 1$ 番目のコードデザインデータの時間項目 $c d e s i g n [k + 1] - > i T i m e$ の値よりも小さく、かつ、 k 番目のコードデザインデータのコード根音項目 $c d e s i g n [k] - > i R o o t$ とコードタイプ項目 $c d e s i g n [k] - > i T y p e$ の各値が 0 以上で有意な値が設定されているか否か（図 18 のステップ S 1806、S 1808 参照）を判定する（ステップ S 2105）。

【0140】

ステップ S 2105 の判定が YES ならば、入力モチーフ 108 の現在の処理対象のノート $n o t e [i]$ の発音タイミングにおいて k 番目のコードデザインデータ $c d e s i g n [k]$ によるコード情報が指定されていると判定できる。そこで、CPU 1401 は、変数データ $r o o t$ と $t y p e$ に、それぞれ k 番目のコードデザインデータのコード根音項目 $c d e s i g n [k] - > i R o o t$ とコードタイプ項目 $c d e s i g n [k] - > i T y p e$ の各値を格納する（ステップ S 2106）。

10

【0141】

ステップ S 2105 の判定が NO ならば、CPU 1401 は、ステップ S 2106 の処理はスキップする。

【0142】

以上の処理の後、CPU 1401 は、CPU 1401 は、変数 k の値を +1 インクリメントし（ステップ S 2107）、次のコードデザインデータ $c d e s i g n [k]$ の処理に移行して、ステップ S 2102 の判定処理に戻る。

20

【0143】

CPU 1401 は、変数データ k の値をインクリメントしながらステップ S 2102 から S 2107 までの一連の処理を繰り返し実行した後、全てのコードデザインデータに対する処理を終了してステップ S 2102 の判定が NO になると、図 21 のフローチャートで例示される処理すなわち図 20 のステップ S 2004 の処理を終了する。この結果、変数データ $r o o t$ と $t y p e$ および変数データ $s c a l e$ と $k e y$ に、入力モチーフ 108 の現在の処理対象ノートの発音タイミングに対応するコード情報が得られる。

【0144】

図 22 は、図 20 のステップ S 2005 のノートタイプ取得処理の詳細例を示すフローチャートである。この処理は、図 7 を用いて前述したように、図 20 のステップ S 2003 で設定されている入力モチーフ 108 の現在のノート $n o t e s [i]$ に対応するピッチ $i p i t [i]$ と、図 20 のステップ S 2004 で算出されている入力モチーフ 108 の現在のノート $n o t e s [i]$ の発音タイミングに対応するコード進行を構成するキー $k e y$ 、スケール $s c a l e$ 、コード根音 $r o o t$ 、およびコードタイプ $t y p e$ とに従って、入力モチーフ 108 の現在のノート $n o t e s [i]$ のノートタイプを取得する処理である。

30

【0145】

まず、CPU 1401 は、ROM 1402 に記憶されている標準ピッチクラスセットテーブル中の図 7 (a) に例示されるデータ構成を有するコードトーンテーブルから、図 20 のステップ S 2004 で算出されたコードタイプ $t y p e$ に対応するコードトーンピッチクラスセットを取得し、RAM 1403 上の変数データ $p c s 1$ に格納する（ステップ S 2201）。以下、この変数データ $p c s 1$ の値をコードトーンピッチクラスセット $p c s 1$ と呼ぶ。

40

【0146】

次に、CPU 1401 は、ROM 1402 に記憶されている標準ピッチクラスセットテーブル中の図 7 (b) に例示されるデータ構成を有するテンションノートテーブルから、上記コードタイプ $t y p e$ に対応するテンションノートピッチクラスセットを取得し、RAM 1403 上の変数データ $p c s 2$ に格納する（ステップ S 2202）。以下、この変数データ $p c s 2$ の値をテンションノートピッチクラスセット $p c s 2$ と呼ぶ。

【0147】

50

次に、CPU 1401は、ROM 1402に記憶されている標準ピッチクラスセットテーブル中の図7(c)に例示されるデータ構成を有するスケールノートテーブルから、図20のステップS2004で得られているスケールscaleに対応するスケールノートピッチクラスセットを取得し、RAM 1403上の変数データcs3に格納する(ステップS2203)。以下、この変数データpcs3の値をスケールノートピッチクラスセットpcs3と呼ぶ。

【0148】

続いて、CPU 1401は、入力モチーフ108の現在の処理対象のノートnotes[i]に対して図20のステップS2003で得られているピッチipit[i]を、コード根音rootを第0音の音階構成音としたときの第0音から第11音までの1オクターブ分の音階構成音のいずれかに写像させたときの、ピッチipit[i]のコード根音rootに対する音程を、次式により算出し、RAM 1403上の変数データpc1に格納する(ステップS2204)。以下、変数データpc1の値を、入力モチーフピッチクラスpc1と呼ぶ。

【0149】

$$pc1 = (ipit[i] - root + 12) \bmod 12 \quad \cdots (1)$$

【0150】

なお、「mod 12」は、その左側の括弧に対応する「値」を12で割ったときの余りである。

【0151】

同様に、CPU 1401は、入力モチーフ108の現在のノートnotes[i]に対して図20のステップS2004で得られているピッチipit[i]を、キーkeyを第0音の音階構成音としたときの第0音から第11音までの1オクターブ分の音階構成音のいずれかに写像させたときの、ピッチipit[i]のキーkeyに対する音程を、次式により算出し、RAM 1403上の変数データpc2に格納する(ステップS2205)。以下、変数データpc2の値を、入力モチーフピッチクラスpc2と呼ぶ。

【0152】

$$pc2 = (ipit[i] - key + 12) \bmod 12 \quad \cdots (2)$$

【0153】

次に、CPU 1401は、入力モチーフピッチクラスpc1がコードトーンピッチクラスセットpcs1に含まれるか否かを判定する(ステップS2206)。この判定演算処理は、2のpc1乗 = 2^{pc1} とpcs1(図7(a)参照)のビット毎の論理積をとりそれが 2^{pc1} と等しいか否かを比較する演算処理として実現される。

【0154】

CPU 1401は、ステップS2206の判定がYESならば、ノートタイプをコードトーンと決定し、ノートタイプと隣接音程の配列のノートタイプ要素の位置incon[i×2]に、ROM 1402からコードトーンを示す定数データci_ChordToneの値を読み出して格納する(ステップS2207)。その後、CPU 1401は、図22のフローチャートで例示される処理すなわち図20のステップS2005のノートタイプ取得処理を終了する。

【0155】

CPU 1401は、ステップS2206の判定がNOならば、入力モチーフピッチクラスpc1がテンションノートピッチクラスセットpcs2に含まれ、かつ入力モチーフピッチクラスpc2がスケールノートピッチクラスセットpcs3に含まれるか否かを判定する(ステップS2208)。この判定演算処理は、2のpc1乗 = 2^{pc1} とpcs2(図7(b)参照)のビット毎の論理積をとりそれが 2^{pc1} と等しく、かつ2のpc2乗 = 2^{pc2} とpcs3(図7(c)参照)のビット毎の論理積をとりそれが 2^{pc2} と等しいか否かを比較する演算処理として実現される。

【0156】

CPU 1401は、ステップS2208の判定がYESならば、ノートタイプをアヴェ

10

20

30

40

50

イラブルノートと決定し、ノートタイプと隣接音程の配列のノートタイプ要素の位置 $in_con[i \times 2]$ に、ROM 1402 からアヴェイラブルノートを示す定数データ $ci_AvailableNote$ の値を読み出して格納する（ステップ S 2209）。その後、CPU 1401 は、図 22 のフローチャートで例示される処理すなわち図 20 のステップ S 2005 のノートタイプ取得処理を終了する。

【0157】

CPU 1401 は、ステップ S 2208 の判定が NO ならば、入力モチーフピッチクラス $pc2$ がスケールノートピッチクラスセット $pcs3$ に含まれるか否かを判定する（ステップ S 2210）。この判定演算処理は、2 の $pc2$ 乗 = 2^{pc2} と $pcs3$ （図 7（c）参照）のビット毎の論理積をとりそれが 2^{pc2} と等しいか否かを比較する演算処理として実現される。

10

【0158】

CPU 1401 は、ステップ S 2210 の判定が YES ならば、ノートタイプをスケールノートと決定し、ノートタイプと隣接音程の配列のノートタイプ要素の位置 $in_con[i \times 2]$ に、ROM 1402 からスケールノートを示す定数データ $ci_ScaleNote$ の値を読み出して格納する（ステップ S 2211）。その後、CPU 1401 は、図 22 のフローチャートで例示される処理すなわち図 20 のステップ S 2005 のノートタイプ取得処理を終了する。

【0159】

CPU 1401 は、ステップ S 2210 の判定が NO ならば、入力モチーフピッチクラス $pc1$ がテンションノートピッチクラスセット $pcs2$ に含まれるか否かを判定する（ステップ S 2212）。この判定演算処理は、2 の $pc1$ 乗 = 2^{pc1} と $pcs2$ （図 7（b）参照）のビット毎の論理積をとりそれが 2^{pc1} と等しいか否かを比較する演算処理として実現される。

20

【0160】

CPU 1401 は、ステップ S 2212 の判定が YES ならば、ノートタイプをテンションノートと決定し、ノートタイプと隣接音程の配列のノートタイプ要素の位置 $in_con[i \times 2]$ に、ROM 1402 からテンションノートを示す定数データ $ci_TensionNote$ の値を読み出して格納する（ステップ S 2213）。その後、CPU 1401 は、図 22 のフローチャートで例示される処理すなわち図 20 のステップ S 2005

30

のノートタイプ取得処理を終了する。

【0161】

最後に、CPU 1401 は、ステップ S 2212 の判定も NO ならば、ノートタイプをアヴォイドノートと決定し、ノートタイプと隣接音程の配列のノートタイプ要素の位置 $in_con[i \times 2]$ に、ROM 1402 からアヴォイドノートを示す定数データ $ci_AvoidNote$ の値を読み出して格納する（ステップ S 2214）。その後、CPU 1401 は、図 22 のフローチャートで例示される処理すなわち図 20 のステップ S 2005 のノートタイプ取得処理を終了する。

【0162】

以上説明した図 22 のフローチャートで例示される図 20 のステップ S 2005 のノートタイプ取得処理により、入力モチーフ 108 の現在のノート $notes[i]$ のノートタイプが、ノートタイプと隣接音程の配列のノートタイプ要素の位置 $in_con[i \times 2]$ （図 7（b）参照）に取得される。

40

【0163】

図 23 は、図 20 のノート接続性チェック処理の詳細例を示すフローチャートである。この処理は、図 10 を用いて前述した処理を実現する。

【0164】

まず、CPU 1401 は、RAM 1403 内の変数データ $iTotalValue$ に初期値「0」を格納する（ステップ S 2301）。このデータは、現在の評価対象のコード進行データ # n （図 17 のステップ S 1704 参照）についての入力モチーフ 108 に対

50

する適合度を算出するための総合評価点を保持する。

【0165】

次に、CPU1401は、変数データ*i*について、ステップS2302で初期値「0」を格納した後、ステップS2321で+1ずつインクリメントしながら、ステップS2303の判定がYES、すなわち変数データ*i*の値が変数データ*i*NoteCntの値から2を減算した値よりも小さい値であると判定される間、ステップS2303からS2321までの一連の処理を繰返し実行する。この繰返し処理が、図10(b)の入力モチーフ108中のノートごとの*i* = 0から7までの繰返し処理に対応する。

【0166】

入力モチーフ108中の*i*番目のノートごとに実行されるステップS2304からS2320までの一連の処理において、CPU1401はまず、RAM1403内の変数データ*i*Valueに初期値「0」を格納する(ステップS2304)。続いて、CPU1401は、変数データ*j*について、ステップS2306で初期値「0」を格納した後、ステップS2318で+1ずつインクリメントしながら、ステップS2307の判定がYES、すなわち変数データ*j*の値が終端値に達するまでの間、ステップS2307からS2319までの一連の処理を繰返し実行する。この繰返し処理が、*i*番目のノートごとに、変数データ*j*の値で定まる図9の各ノート接続ルールをチェックする繰返し処理に対応する。

【0167】

入力モチーフ108中の*i*番目のノートごとに、*j*番目のノート接続ルールをチェックするステップS2308からS2316までの一連の処理において、CPU1401はRAM1403内の変数データ*k*について、ステップS2308で初期値「0」を格納した後、ステップS2315で+1ずつインクリメントしながら、ステップS2309からステップS2315の一連の処理を繰返し実行する。この繰返し処理により、入力モチーフ108中の*i*番目のノートから4つの連続するノートに対応する4つのノートタイプincon[*i*×2]、incon[*i*×2+2]、incon[*i*×2+4]、incon[*i*×2+6]のそれぞれと、図9に例示される*j*番目のノート接続ルール内の4つのノートタイプci__NoteConnect[*j*][0]、ci__NoteConnect[*j*][2]、ci__NoteConnect[*j*][4]、ci__NoteConnect[*j*][6]のそれぞれとの一致の有無が判定される。また、入力モチーフ108中の*i*番目のノートから4つの連続するノート間の3つの隣接音程incon[*i*×2+1]、incon[*i*×2+3]、incon[*i*×2+5]のそれぞれと、図9に例示される*j*番目のノート接続ルール内の3つの隣接音程ci__NoteConnect[*j*][1]、ci__NoteConnect[*j*][3]、ci__NoteConnect[*j*][5]のそれぞれとの一致の有無が判定される。

【0168】

入力モチーフ108中の*i*番目のノートから4つの連続するノートを図9の*j*番目のノート接続ルールと比較する処理として、変数データ*k*の値が0から3までインクリメントされながらステップS2309からステップS2315までの一連の処理が4回繰返し実行されるうちで、ステップS2310、S2312、またはS2314のいずれか1つでも条件が成立すると、現在の*j*番目のノート接続ルールは入力モチーフ108に対して不適合となって、ステップS2319に移行し、変数データ*j*の値がインクリメントされた次のノート接続ルールの適合評価に処理が移行する。

【0169】

具体的には、ステップS2310で、CPU1401は、入力モチーフ108の*i*+*k*番目のノートのノートタイプincon[*i*×2+*k*×2]と、*j*番目のノート接続ルールの*k*番目のノートタイプci__NoteConnect[*j*][*k*×2]とが不一致となったか否かを判定する。ステップS2310の判定がYESになると、CPU1401は、そのノート接続ルールの少なくとも1つのノートタイプが入力モチーフ108内の現在の処理対象(*i*番目)のノートから始まる4つのノートのノートタイプの少なくとも1

10

20

30

40

50

つと一致しないため、ステップS 2 3 1 9に移行する。

【0 1 7 0】

ステップS 2 3 1 0の判定がNOならば、ステップS 2 3 1 1およびステップS 2 3 1 2が実行されるが、これらについては後述する。ステップS 2 3 1 1およびS 2 3 1 2の判定がともにNOとなった後、CPU 1 4 0 1は、変数データkの値が3より小さい場合に、ステップS 2 3 1 3の判定がYESとなって、ステップS 2 3 1 4で隣接音程に関する判定処理を実行する。ステップS 2 3 1 3の判定が行われるのは、 $k = 3$ となる入力モチーフ1 0 8の4ノート目については、それ以降には隣接音程は存在しないため、変数データkの値が0から2までの範囲でのみ、隣接音程の判定処理を実行するためである。ステップS 2 3 1 4において、CPU 1 4 0 1は、入力モチーフ1 0 8の $i + k$ 番目のノートと $i + k + 1$ 番目のノートの間の隣接音程 $inc on [i \times 2 + k \times 2 + 1]$ と、j番目のノート接続ルール k 番目のノートタイプと $k + 1$ 番目のノートタイプの間の隣接音程 $ci_NoteConnect [j] [k \times 2 + 1]$ とが不一致であり、かつ、 $ci_NoteConnect [j] [k \times 2 + 1]$ の値が「99」と不一致であるか否かを判定する。隣接音程の値「99」は、その隣接音程がどの値でもよいことを示している。ステップS 2 3 1 4の判定がYESになると、CPU 1 4 0 1は、そのノート接続ルールの少なくとも1つの隣接音程が入力モチーフ1 0 8内の現在の処理対象(i 番目)のノートから始まる4つのノートの隣接ノート間の隣接音程の少なくとも1つと一致しないため、ステップS 2 3 1 9に移行する。

【0 1 7 1】

上記一連の処理で、ステップS 2 3 1 0において、入力モチーフ1 0 8の $i + k$ 番目のノートのノートタイプ $inc on [i \times 2 + k \times 2]$ と、j番目のノート接続ルール k 番目のノートタイプ $ci_NoteConnect [j] [k \times 2]$ の一致が検出されてステップS 2 3 1 0の判定がNOとなった後、CPU 1 4 0 1は、j番目のノート接続ルールの k 番目の次の $k + 1$ 番目のノートタイプ $ci_NoteConnect [j] [k \times 2 + 2]$ が $ci_NullNoteType$ であるか否かを判定する(ステップS 2 3 1 1)。

【0 1 7 2】

$ci_NullNoteType$ が設定されるのは、図9の $j = 0$ から8までのノート接続ルールにおける $k = 3$ の場合の $ci_NoteConnect [j] [6]$ に対してである。従って、ステップS 2 3 1 1の判定がYESとなるケースは、変数データjの値の範囲が0から8の間であって、変数データkの値が0、1、2の3音分についてノートタイプおよび隣接音程が一致して、 $k = 2$ となっている場合である。前述したように、 $j = 0 \sim 8$ の範囲のノート接続ルールは3音のルールであるため、4音目は $ci_NullNoteType$ となって評価をする必要がない。従って、ステップS 2 3 1 1の判定がYESとなる場合には、そのときのノート接続ルールは入力モチーフ1 0 8内の i 番目のノートから始まる3つのノートと適合する。このため、ステップS 2 3 1 1の判定がYESならば、CPU 1 4 0 1は、ステップS 2 3 1 6に移行して、変数データ $iValue$ に、そのノート接続ルールの評価点 $ci_NoteConnect [j] [7]$ (図9参照)を累算する。

【0 1 7 3】

一方、ステップS 2 3 1 1の判定がNOとなる場合は、ステップS 2 3 1 2およびS 2 3 1 3を経てステップS 2 3 1 4の隣接音程の評価処理に進む。ここで、CPU 1 4 0 1は、ステップS 2 3 1 1の判定がNOとなった直後のステップS 2 3 1 2で、変数データiの値が入力モチーフ1 0 8のノート数を示す変数データ $iNoteCnt$ の値から3を減算した値に等しく、かつ変数データkの値が2に等しいか否かを判定する。このケースでは、処理対象となる入力モチーフ1 0 8のノートは、 $i + k$ 番目、すなわち $iNoteCnt - 3 + 2 = iNoteCnt - 1$ 番目、つまり、入力モチーフ1 0 8中の最後のノートになる。この状態で、ステップS 2 3 1 1において、 $ci_NoteConnect [j] [k \times 2 + 2] = ci_NoteConnect [j] [6]$ の値が ci_Null

10

20

30

40

50

1 N o t e T y p e にならない場合は、図 9 の j の値が 9 以上のノート接続ルールが処理されている場合である。つまり、ノート接続ルールは、4 音についてのものである。一方、この場合における入力モチーフ 108 中の処理対象のノートは、 $i = i N o t e C n t - 3$ から始まり最終ノートの $i = i N o t e C n t - 1$ までの 3 音である。従って、このケースでは、入力モチーフ 108 中の処理対象のノートの数とノート接続ルール中の音の数が合わないため、そのノート接続ルールは入力モチーフ 108 に適合することはない。従って、ステップ S 2312 の判定が Y E S となる場合は、C P U 1401 は、そのノート接続ルールに関する適合評価を行わずに、ステップ S 2319 に移行する。

【0174】

上述したステップ S 2310、S 2311、S 2312、および S 2314 のいずれの条件も成立せずに、ステップ S 2309 から S 2315 までの一連の処理が 4 回繰り返されてステップ S 2309 の判定が N O になると、入力モチーフ 108 中の i 番目のノートから 4 つの連続するノートに関して、ノートタイプと隣接音程が全て現在の j 番目のノート接続ルールのノートタイプおよび隣接音程と適合したことになる。この場合には、C P U 1401 は、ステップ S 2316 に移行して、変数データ $i V a l u e$ に、現在の j 番目のノート接続ルールの評価点 $c i_N o t e C o n n e c t [j] [7]$ (図 9 参照) を累算する。

【0175】

なお、1 つのノート接続ルールのみが入力モチーフ 108 に適合するとは限らず、例えば 3 音のノート接続ルールに適合しかつ 4 音のノート接続ルールにも適合する場合があります。

【0176】

そこで、C P U 1401 は、ステップ S 2319 で変数データ j の値がインクリメントされながらステップ S 2307 で全てのノート接続ルールに関する評価が完了するまで、ステップ S 2309 の判定が N O またはステップ S 2311 の判定が Y E S となってノート接続ルールが適合するごとに、ステップ S 2316 において、新たに適合したノート接続ルールの評価点 $c i_N o t e C o n n e c t [j] [7]$ が変数データ $i V a l u e$ に累算される。

【0177】

その後、C P U 1401 は、変数データ j の値を + 1 インクリメントして次のノート接続ルールの評価に移行し (ステップ S 2319)、ステップ S 2307 の判定処理に戻る。

【0178】

C P U 1401 は、全てのノート接続ルールに対する評価が完了してステップ S 2307 の判定が Y E S になると、現在のコード進行データ # n に対応する変数データ $i T o t a l V a l u e$ に、変数データ $i V a l u e$ に累算されている評価点を累算する (ステップ S 2320)。

【0179】

その後、C P U 1401 は、変数 i の値を + 1 インクリメントし (ステップ S 2321)、ステップ S 2303 の判定処理に戻って、入力モチーフ 108 中の次のノートに処理を移す (図 10 (b) 参照)。

【0180】

C P U 1401 は、入力モチーフ 108 中の全てのノートについて全てのノート接続ルールの適合評価の処理を終了すると、ステップ S 2303 の判定が N O となる。ここで、入力モチーフ 108 中の処理対象のノートの終了位置は、本来は入力モチーフ 108 中の最終ノートを含む 4 音手前のノートであり、それに対応する変数データ i の値は「 $(i N o t e C n t - 1) - 3 = i N o t e C n t - 4$ 」である。しかし、図 10 (b) の $i = 7$ として例示されるように、最後の処理は 3 音で行われるため、終了位置に対応する変数データ i の値は、「 $i N o t e C n t - 3$ 」となる。よって、ステップ S 2303 の終了判定は、「 $i < i N o t e C n t - 2$ 」が N O になる場合となる。

10

20

30

40

50

【0181】

ステップS2303の判定がNOになると、CPU1401は、変数データ*i*Total Valueの値を入力モチーフ108中の処理したノート数(*i*NoteCnt-2)で除算して正規化し、その除算結果をコード進行#*n*の入力モチーフ108に対する適度として、変数データ*do*Valueに格納する(ステップS2322)。その後、CPU1401は、図23のフローチャートすなわち図20のステップS2009のノート接続性チェック処理を終了する。

【0182】

図24は、図16の自動作曲処理において、ステップS1607のコード進行選択処理の次に実行されるステップS1608のメロディ生成処理の詳細例を示すフローチャートである。

10

【0183】

まず、CPU1401は、RAM1403の変数領域を初期化する(ステップS2401)。

【0184】

次に、CPU1401は、図16のステップS1607のコード進行選択処理によって選択され例えばユーザによって指示されたコード進行候補に対応する曲構造データ(図6参照)を、伴奏・コード進行DB103から読み込む(ステップS2402)。

【0185】

その後、CPU1401は、変数データ*i*の値を初期値「0」に設定した後(ステップS2403)、ステップS2409で*i*の値をインクリメントしながら、ステップS2404で曲構造データの終端に達したと判定するまで、変数データ*i*によって指示される曲構造データ上の小節のフレーズごとに、入力モチーフ108と、ROM1402に記憶されるフレーズセットDB106に登録されているフレーズセット(図11参照)、およびROM1402に記憶されるルールDB104(図9参照)を参照しながら、そのフレーズのメロディを自動生成する。変数データ*i*は、その値がステップS2409で0から+1ずつインクリメントされることにより、図6に例示される曲構造データの「Measure」項目の値を順次指定して、曲構造データ上の各レコードを指定してゆく。

20

【0186】

具体的には、まず、CPU1401は、曲構造データの終端に達したか否かを判定する(ステップS2404)。

30

【0187】

ステップS2404の判定がNOならば、CPU1401は、変数データ*i*によって指定される曲構造データの現在の小節が、入力モチーフ108が入力された小節と一致するか否かを判定する(ステップS2405)。

【0188】

ステップS2405の判定がYESならば、CPU1401は、その入力モチーフ108をそのままメロディ110(図1参照)の一部として、例えばRAM1403上の出力メロディ領域に出力する。

【0189】

ステップS2405の判定がNOならば、CPU1401は、現在の小節が、サビメロディの先頭小節であるか否かを判定する(ステップS2406)。

40

【0190】

ステップS2406の判定がNOならば、CPU1401は、メロディ生成1処理を実行する(ステップS2407)。

【0191】

一方、ステップS2406の判定がYESならば、CPU1401は、メロディ生成2処理を実行する(ステップS2408)。

【0192】

ステップS2407またはS2408の処理の後、CPU1401は、変数データ*i*を

50

+ 1 インクリメントする (ステップ S 2 4 0 9)。その後、CPU 1 4 0 1 は、ステップ S 2 4 0 4 の判定処理に戻る。

【 0 1 9 3 】

図 2 5 は、図 2 4 のステップ S 2 4 0 7 のメロディ生成 1 処理の詳細例を示すフローチャートである。

【 0 1 9 4 】

CPU 1 4 0 1 は、現在の小節が含まれるフレーズの種別が、入力モチーフ 1 0 8 のフレーズの種別と同じであるか否かを判定する (ステップ S 2 5 0 1)。現在の小節が含まれるフレーズの種別は、図 6 に例示される曲構造データ中で、変数データ *i* の値に対応する「Measure」項目を有するレコード中の「PartName[M]」項目または「iPartID[M]」項目を参照することにより、判定することができる。入力モチーフ 1 0 8 のフレーズの種別は、ユーザが入力モチーフ 1 0 8 を入力するときに指定する。

10

【 0 1 9 5 】

ステップ S 2 5 0 1 の判定が YES ならば、CPU 1 4 0 1 は、入力モチーフ 1 0 8 のメロディを現在の小節のメロディとして RAM 1 4 0 3 の所定領域にコピーする。その後、CPU 1 4 0 1 は、ステップ S 2 5 0 7 のメロディ変形処理に移行する。

【 0 1 9 6 】

ステップ S 2 5 0 1 の判定が NO ならば、CPU 1 4 0 1 は、現在の小節が含まれるフレーズの種別に対して、既にメロディが生成済みで、かつ小節の偶数 / 奇数が一致するか否かを判定する (ステップ S 2 5 0 3)。

20

【 0 1 9 7 】

ステップ S 2 5 0 3 の判定が YES ならば、CPU 1 4 0 1 は、生成済みのメロディを現在の小節のメロディとして RAM 1 4 0 3 の所定領域にコピーする (ステップ S 2 5 0 4)。その後、CPU 1 4 0 1 は、ステップ S 2 5 0 7 のメロディ変形処理に移行する。

【 0 1 9 8 】

該当するフレーズのメロディが未だ生成されていなければ (ステップ S 2 5 0 3 の判定が NO)、CPU 1 4 0 1 は、フレーズセット DB 検索処理を実行する (ステップ S 2 5 0 5)。フレーズセット DB 検索処理において、CPU 1 4 0 1 は、フレーズセット DB 1 0 6 から入力モチーフ 1 0 8 に対応するフレーズセットを抽出する。

30

【 0 1 9 9 】

CPU 1 4 0 1 は、ステップ S 2 5 0 5 で検索されたフレーズセット中の、現在の小節が含まれるフレーズの種別と同じ種別のフレーズのメロディを、RAM 1 4 0 3 の所定領域にコピーする (ステップ S 2 5 0 6)。その後、CPU 1 4 0 1 は、ステップ S 2 5 0 7 のメロディ変形処理に移行する。

【 0 2 0 0 】

ステップ S 2 5 0 2、S 2 5 0 4、または S 2 5 0 6 の処理の後、CPU 1 4 0 1 は、コピーしたメロディを変形するメロディ変形処理を実行する (ステップ S 2 5 0 7)。

【 0 2 0 1 】

さらに、CPU 1 4 0 1 は、ステップ S 2 5 0 7 で変形したメロディを構成する各ノートのピッチを最適化するメロディ最適化処理を実行する (ステップ S 2 5 0 8)。この結果、CPU 1 4 0 1 は、曲構造データによって示される小節のフレーズのメロディを自動生成し、RAM 1 4 0 3 の出力メロディ領域に出力する。

40

【 0 2 0 2 】

図 2 6 は、図 2 5 のステップ S 2 5 0 5 のフレーズセット DB 検索処理の詳細例を示すフローチャートである。

【 0 2 0 3 】

まず、CPU 1 4 0 1 は、入力モチーフ 1 0 8 のピッチ列を取り出し、RAM 1 4 0 3 内の配列変数データ *i* MelodyB[0] ~ *i* MelodyB[*i* LengthB - 1] に格納する。ここで、変数データ *i* LengthB には、入力モチーフ 1 0 8 のピッチ

50

列の長さが格納される。

【0204】

次に、CPU1401は、変数データkの値を初期値「0」に設定した後（ステップS2602）、ステップS2609でkの値をインクリメントしながら、ステップS2603でフレーズセットDB106の終端（図11（a）参照）に達したと判定するまで、変数データkによって指示されるフレーズセット（図11（a）参照）について、ステップS2603からS2609の一連の処理を繰り返し実行する。

【0205】

この一連の処理において、まず、CPU1401は、変数データkが示すk番目のフレーズセット内の、入力モチーフ108に対応するフレーズのピッチ列を取り出して、RAM1403内の配列変数データiMelodyA[0]～iMelodyA[iLengthA-1]に格納する（ステップS2604）。ここで、変数データiLengthAには、フレーズセットDB106内のフレーズのピッチ列の長さが格納される。

10

【0206】

次に、CPU1401は、ステップS2601で設定した入力モチーフ108のピッチ列の配列変数データiMelodyB[0]～iMelodyB[iLengthB-1]と、ステップS2604で設定したフレーズセットDB106内のk番目のフレーズセット内の該当フレーズのピッチ列の配列変数データiMelodyA[0]～iMelodyA[iLengthA-1]との間で、DP（Dynamic Programming：動的計画法）マッチング処理を実行し、その結果算出される両者間の距離評価値をRAM1403上の変数データdoDistanceに格納する。

20

【0207】

次に、CPU1401は、RAM1403上の変数データdoMinが示す最小距離評価値のほうが、ステップS2605のDPマッチング処理により新たに算出した距離評価値doDistanceよりも大きくなったか否かを判定する（ステップS2608）。

【0208】

ステップS2608の判定がYESならば、CPU1401は、変数データdoDistanceに格納されている新たな距離評価値を、変数データ変数データdoMinに格納する（ステップS2607）。

【0209】

また、CPU1401は、変数データkの値を、RAM1403上の変数データiBestMochiefに格納する（ステップS2608）。

30

【0210】

ステップS2608の判定がYESならば、CPU1401は、ステップS2607およびS2608の処理はスキップする。

【0211】

その後、CPU1401は、変数データkの値を+1インクリメントしてフレーズセットDB106内の次のフレーズセット（図11（a）参照）に対する処理に移行する。

【0212】

CPU1401は、フレーズセットDB106内の全てのフレーズセットに対する入力モチーフ108とのDPマッチング処理を終了し、ステップS2603の判定がYESになると、変数データiBestMochiefが示す番号のフレーズセットDB106内のフレーズセットを、RAM1403上の所定の領域に出力する（ステップS2610）。その後、CPU1401は、図26に例示されるフローチャートの処理すなわち図25のステップS2505のフレーズセットDB検索処理を終了する。

40

【0213】

図27は、図25のステップS2507のメロディ変形処理の詳細例を示すフローチャートである。この処理は、図12の説明で前述したピッチシフトまたは左右反転によるメロディ変形処理を実行する。

【0214】

50

まず、CPU 1401は、図25のコピー処理により得られたメロディのノート数をカウントするRAM 1403内の変数*i*に初期値「0」を格納する(ステップS2701)。その後、CPU 1401は、ステップS2709で変数*i*の値を+1ずつインクリメントさせながら、ステップS2702で変数*i*の値がメロディのノート数を示す変数データ*iNoteCnt*の値よりも小さいと判定される間、ステップS2702からS2709までの一連の処理を実行する。

【0215】

ステップS2702からS2709の繰返し処理において、ステップSCPU 1401はまず、変形タイプを取得する(ステップS2702)。変形タイプは、ピッチシフトまたは左右反転があり、ユーザが特には図示しないスイッチにより指定することができる。

10

【0216】

変形タイプがピッチシフトである場合には、CPU 1401は、配列変数データ*note[i]*の*iPitch*項目に得られているピッチデータ*note[i] -> iPitch*に、所定値を加算することにより、例えば図12の1201として説明したような例えば2半音上へのピッチシフトを実行する(ステップS2704)。

【0217】

変形タイプが左右反転である場合には、CPU 1401は、変数データ*i*の値が変数データ*iNoteCnt*の値を2で割った値よりも小さいか否かを判定する(ステップS2705)。

【0218】

20

ステップS2705の判定がYESの場合には、まず、CPU 1401は、配列変数データ*note[i]*の*iPitch*項目に得られているピッチデータ*note[i] -> iPitch*を、RAM 1403上の変数*ip*に退避させる(ステップS2706)。

【0219】

次に、CPU 1401は、(*iNoteCnt - i - 1*)番目の配列要素のピッチ項目*note[iNoteCnt - i - 1] -> iPitch*の値を、*i*番目の配列要素のピッチ項目*note[i] -> iPitch*に格納する(ステップS2707)。

【0220】

そして、CPU 1401は、変数データ*ip*に退避させていた元のピッチ項目値を、(*iNoteCnt - i - 1*)番目の配列要素のピッチ項目*note[iNoteCnt - i - 1] -> iPitch*に格納する(ステップS2708)。

30

【0221】

ステップS2705の判定がNOの場合には、CPU 1401は、ステップS2706、S2707、S2708の処理をスキップする。

【0222】

ステップS2704またはS2708の処理の後、あるいはステップS2705の判定がNOとなった後に、CPU 1401は、ステップS2709で、変数データ*i*の値を+1インクリメントし、次のノートに対する処理に移行してステップS2702の判定処理に戻る。

【0223】

40

以上の処理により、図12の1202で説明した左右反転処理が実現される。

【0224】

図28は、図25のステップS2508のメロディ最適化処理の詳細例を示すフローチャートである。この処理は、図13の説明で前述したピッチの最適化処理を実現する。

【0225】

まず、CPU 1401は、次式により、別ピッチ候補の全組合せ数を算出する(ステップS2801)。

【0226】

$$IWnum = MAX_NOTE_CANDIDATE \wedge iNoteCnt$$

【0227】

50

ここで、演算子「 \wedge 」は、べき乗演算を示す。また、ROM 1402上の定数データMAX__NOTE__CANDIDATEは、図13に示される1つのノートに対する別ピッチ候補ipitd[0]~ipitd[4]の候補数を示し、この例では5である。

【0228】

次に、CPU1401は、別ピッチ候補のカウント用の変数データiCntを初期値「0」に設定した後(ステップS2802)、ステップS2818で変数データiCntを+1ずつインクリメントしながら、ステップS2803で変数データiCntの値がステップS2801で算出した別ピッチ候補の全組合せ数より小さい範囲で、入力されたメロディのピッチを変更しながら、そのメロディの妥当性を評価する。

【0229】

CPU1401は、変数データiCntの値がインクリメントされるごとに、ステップS2805からS2817までの一連の処理を実行する。

【0230】

まず、CPU1401は、図25のコピー処理により得られたメロディのノート数をカウントするRAM1403内の変数iに初期値「0」を格納する(ステップS2805)。その後、CPU1401は、ステップS2813で変数iの値を+1ずつインクリメントさせながら、ステップS2806で変数iの値がメロディのノート数を示す変数データiNoteCntの値よりも小さいと判定される間、ステップS2806からS2813までの一連の処理を繰り返し実行する。この繰り返し処理によって、メロディの全てのノートに対して、ステップS2807、S2808、およびS2809によって、ピッチ修正が行われる。

【0231】

まず、CPU1401は、次式を演算することによって、ピッチ修正値をRAM1403上の変数データipitdevに得る(ステップS2807)。

【0232】

$$ipitdev = ipitd[(iCnt / MAX_NOTE_CANDIDATE \wedge i) \bmod MAX_NOTE_CANDIDATE]$$

ここで、「mod」は、剰余演算を示す。

【0233】

次に、CPU1401は、入力したメロディのピッチ項目値note[i] -> ipitに、ステップS2807で算出された変数データipitdevの値を加算し、その結果をピッチ情報列を示す配列変数データipit[i]に格納する(ステップS809)。

【0234】

次に、前述した図20のステップS2005~S2007と同様にして、ピッチ情報列を示す配列変数データipit[i]に対して、ノートタイプ取得処理(ステップS2810)と、隣接音程の算出処理(ステップS2811およびS2812)を実行する。

【0235】

CPU1401は、入力メロディを構成する全てのノートに対して、現在の変数データiCntの値に対応するピッチ修正が完了すると、ステップS2806の判定がNOなる。この結果、CPU1401は、ステップS2814において、ステップS2810~S2812で算出されたメロディを構成するノートごとのノートタイプおよび隣接音程に対して、前述した図23の処理と同じノート接続性チェック処理を実行する(ステップS2814)。なお、このとき、入力されたメロディの小節に該当するコード進行データ中のコード情報が抽出されて使用される。

【0236】

CPU1401は、ステップS2814のノート接続性チェック処理で変数データdoValueに新たに得られた適合度の値が、変数データiMaxValueに保持されている最良適合度の値よりも大きいか否かを判定する(ステップS2815)。

【0237】

10

20

30

40

50

ステップS 2 8 1 5の判定がYESならば、CPU 1 4 0 1は、変数データiMaxValueの値を変数データdoValueの値で置き換え(ステップS 2 8 1 6)、変数データiMaxCntの値を変数データiCntの値で置き換える(ステップS 2 8 1 7)。

【0 2 3 8】

その後、CPU 1 4 0 1は、変数データiCntの値を+ 1インクリメントし(ステップS 2 8 1 8)、ステップS 2 8 0 3の判定処理に戻る。

【0 2 3 9】

以上の動作が、順次インクリメントされる変数データiCntの値に対して繰返し実行された結果、別ピッチ候補の全ての組合せに対してノート接続性をチェックする処理が完了すると、ステップS 2 8 0 3の判定がNOとなる。

10

【0 2 4 0】

この結果、CPU 1 4 0 1は、変数iに初期値「0」を格納した後(ステップS 2 8 1 9)、ステップS 2 8 2 3で変数iの値を+ 1ずつインクリメントさせながら、ステップS 2 8 2 0で変数iの値がメロディのノート数を示す変数データiNoteCntの値よりも小さいと判定される間、ステップS 2 8 2 0からS 2 8 2 3までの一連の処理を繰返し実行する。この繰返し処理によって、メロディの全てのノートに対して、変数データiMaxCntに得られている最良値を用いて、ピッチの修正すなわち最適化が実行される。

【0 2 4 1】

20

具体的には、CPU 1 4 0 1は、ステップS 2 8 2 0の終了判定を行った後、次式を演算することによって、最適なピッチ修正値を、ピッチ情報列の配列変数データpit[i]に得る(ステップS 2 8 2 1)。

【0 2 4 2】

$$ipit[i] = note[i] -> i Pit + ipitd[(iMaxCnt / (MAX_NOTE_CANDIDATE^i) \bmod MAX_NOTE_CANDIDATE)]$$

【0 2 4 3】

そして、CPU 1 4 0 1は、ピッチ情報列の配列変数データpit[i]の値を、入力されたメロディのノートデータのピッチ項目値note[i] -> i Pitに上書きコピーする(ステップS 2 8 2 2)。

30

【0 2 4 4】

最後に、CPU 1 4 0 1は、変数iの値をインクリメントし(ステップS 2 8 2 3)、その後ステップS 2 8 2 0の判定処理に戻る。

【0 2 4 5】

CPU 1 4 0 1は、入力されたメロディを構成する全てのノートデータに対する上記処理が完了すると、ステップS 2 8 2 0の判定がNOになって、図28のフローチャートで例示される処理すなわち図25のステップS 2 5 0 8のメロディ最適化処理を終了する。

【0 2 4 6】

図29は、図24のメロディ生成2処理(サビ先頭メロディ生成処理)の詳細例を示すフローチャートである。

40

【0 2 4 7】

まず、CPU 1 4 0 1は、サビ先頭メロディは生成済みか否かを判定する(ステップS 2 9 0 1)。

【0 2 4 8】

サビ先頭メロディはまだ生成されておらずステップS 2 9 0 1の判定がNOならば、CPU 1 4 0 1は、フレーズセットDB検索処理を実行する(ステップS 2 9 0 2)。この処理は、図25のステップS 2 5 0 5に対応する図26の処理と同じである。このフレーズセットDB検索処理により、CPU 1 4 0 1は、フレーズセットDB 1 0 6から入力モチーフ1 0 8に対応するフレーズセットを抽出する。

50

【0249】

次に、CPU1401は、ステップS2902で検索されたフレーズセット中の、サビ先頭（Ｃメロ）のフレーズのメロディを、RAM1403の所定領域にコピーする（ステップS2903）。

【0250】

続いて、CPU1401は、ステップS2903で得たメロディに対して、図25のステップS2508と同様の図28で示されるメロディ最適化処理を実行する（ステップS2904）。

【0251】

CPU1401は、ステップS2904で得られたピッチが最適化されたメロディデータを、メロディ110の一部として、RAM1403の出力メロディ領域に格納する。その後、CPU1401は、図29のフローチャートで例示される処理すなわち図24のメロディ生成2処理（サビ先頭メロディ生成処理）を終了する。

10

【0252】

サビ先頭メロディは生成されておりステップS2901の判定がYESならば、CPU1401は、生成済みのサビ先頭メロディを現在の小節のメロディとして、RAM1403の出力メロディ領域にコピーする（ステップS2905）。その後、CPU1401は、図29のフローチャートで例示される処理すなわち図24のメロディ生成2処理（サビ先頭メロディ生成処理）を終了する。

【0253】

20

以上説明した実施形態によれば、入力モチーフ108とコード進行データとの対応関係を適合度として数値化することが可能となり、その適合度に基づいて入力モチーフ108に適合するコード進行データを適切に選べるようになるため、自然な楽曲生成が可能になる。

【0254】

以上の実施形態に関して、更に以下の付記を開示する。

（付記1）

複数のノートデータを含むモチーフを入力するモチーフ入力部と、

連続するノートタイプの接続関係を規定する複数種のノート接続ルールを参照しながら、複数種のコード進行データそれぞれの前記モチーフに対する適合度を算出し、当該適合度の算出されたコード進行データと前記モチーフとに基づいてメロディを生成するメロディ生成部と、

30

を備えることを特徴とする自動作曲装置。

（付記2）

前記自動作曲装置はさらに、前記算出された適合度に基づいて、複数種のコード進行データの中から、コード進行データを選択するコード進行選択部を有する、付記1に記載の自動作曲装置。

（付記3）

前記ノート接続ルールは、複数の連続するノートタイプの接続関係を規定するとともに、隣接する当該ノートタイプ間の音程を規定し、

40

前記コード進行選択部は、前記複数種のコード進行データそれぞれに基づいて、前記モチーフを構成する各ノートデータについて、当該ノートデータの発音タイミングに対応する当該コード進行データ上でのノートタイプと、隣接する当該ノート間の音程とを算出するとともに、当該ノートタイプおよび音程を、前記ノート接続ルールを構成するノートタイプおよび音程と比較することにより、当該コード進行データの前記モチーフに対する適合度を算出する、付記2に記載の自動作曲装置。

（付記4）

前記コード進行選択部は、前記複数のコード進行データそれぞれに対してキーシフトをしたコード進行データごとに、当該コード進行データの前記モチーフに対する適合度を算出し、当該算出された適合度に基づいてコード進行データおよびキーシフト量を選択する

50

、付記 2 または 3 に記載の自動作曲装置。

(付記 5)

前記コード進行選択部は、前記算出された適合度が最も高くなるコード進行データおよびキーシフト量を選択する、付記 4 に記載の自動作曲装置。

(付記 6)

前記モチーフ入力部は、前記モチーフを、楽曲のメロディを構成する複数の種別のフレーズのいずれかに対応させて入力し、

前記メロディ生成部は、前記楽曲のメロディを構成する複数の種別のフレーズの組み合わせからなるフレーズセットを複数種記憶したフレーズデータベースと、前記複数のフレーズセットそれぞれについて、前記モチーフの種別と同じ種別のフレーズを当該モチーフと比較することにより、前記モチーフに類似するフレーズを有するフレーズセットを、前記フレーズデータベースから検索するフレーズセット検索部と、を有し、前記検索されたフレーズセットに含まれる各フレーズに基づいてメロディの生成を行う、付記 1 乃至 5 のいずれかに記載の自動作曲装置。

10

(付記 7)

前記メロディ生成部は、前記検索されたフレーズセットに含まれるフレーズを変形させ変形部を含む、付記 6 に記載の自動作曲装置。

(付記 8)

前記変形部は、前記フレーズを構成する各ノートデータに含まれるピッチを予め定められた値だけシフトする、付記 7 に記載の自動作曲装置。

20

(付記 9)

前記変形部は、前記フレーズを構成するノートデータの並び順を変更する、付記 7 に記載の自動作曲装置。

(付記 10)

前記自動作曲装置はさらに、前記複数種のコード進行データを記憶するコード進行データベースと、前記複数種のノート接続ルールを記憶するルールデータベースと、を有する付記 1 乃至 9 のいずれかに記載の自動作曲装置。

(付記 11)

前記自動作曲装置はさらに、前記メロディ生成部により生成されたメロディに基づいた楽曲を再生する再生部、及び当該楽曲を表す楽譜を表示する楽譜表示部の少なくとも一方を有する付記 1 乃至 10 のいずれかに記載の自動作曲装置。

30

(付記 12)

自動作曲装置が、

複数のノートデータを含むモチーフを入力し、

連続するノートタイプの接続関係を規定する複数種のノート接続ルールを参照しながら、複数種のコード進行データそれぞれの前記モチーフに対する適合度を算出し、

前記適合度の算出されたコード進行データと前記モチーフとに基づいてメロディを生成する、自動作曲方法。

(付記 13)

複数のノートデータを含むモチーフを入力するステップと、

連続するノートタイプの接続関係を規定する複数種のノート接続ルールを参照しながら、複数種のコード進行データそれぞれの前記モチーフに対する適合度を算出するステップと、

40

前記適合度の算出されたコード進行データと前記モチーフとに基づいてメロディを生成するステップと、

をコンピュータに実行させるプログラム。

【符号の説明】

【0255】

100 自動作曲装置

101 モチーフ入力部

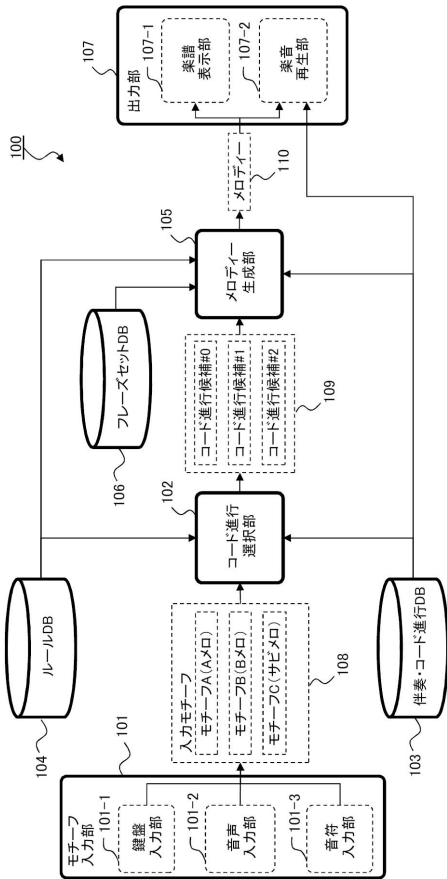
50

- 1 0 1 - 1 鍵盤入力部
- 1 0 1 - 2 音声入力部
- 1 0 1 - 3 音符入力部
- 1 0 2 コード進行選択部
- 1 0 3 伴奏・コード進行 D B
- 1 0 4 ルール D B
- 1 0 5 メロディ生成部
- 1 0 6 フレーズセット D B
- 1 0 7 出力部
 - 1 0 7 - 1 楽譜表示部
 - 1 0 7 - 2 楽音生成部
- 1 0 8 入力モチーフ
 - モチーフA (A×M)
 - モチーフB (B×M)
 - モチーフC (サビメロ)
- 1 0 9 コード進行候補
 - コード進行候補#0
 - コード進行候補#1
 - コード進行候補#2
- 1 1 0 メロディ
 - C P U
 - R O M
 - R A M
- 1 4 0 1 入力手段
- 1 4 0 2 表示手段
- 1 4 0 3 音源部
- 1 4 0 4 サウンドシステム

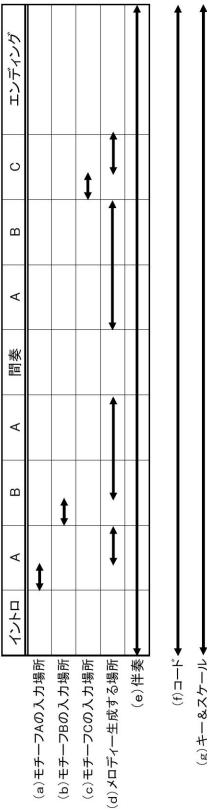
10

20

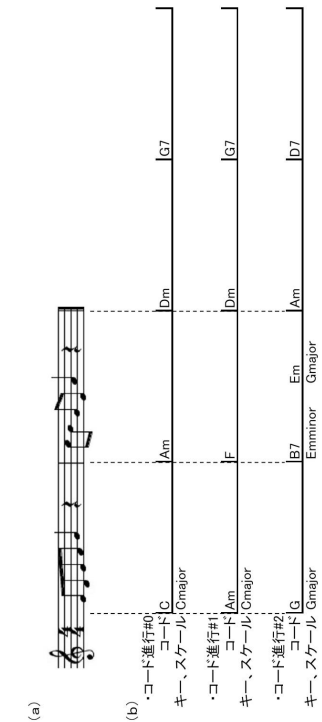
【 図 1 】



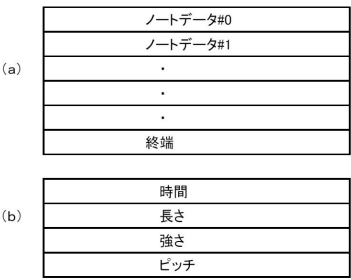
【 図 2 】



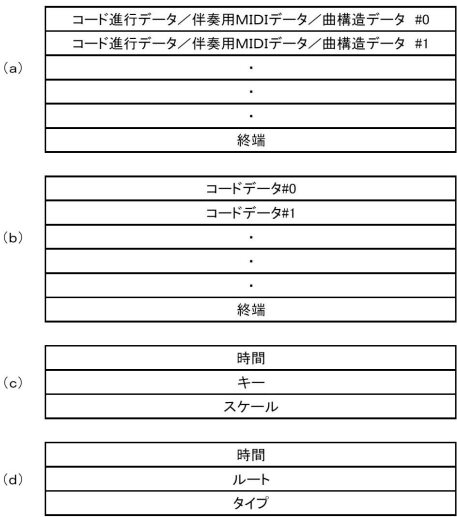
【図 3】



【図 4】



【図 5】



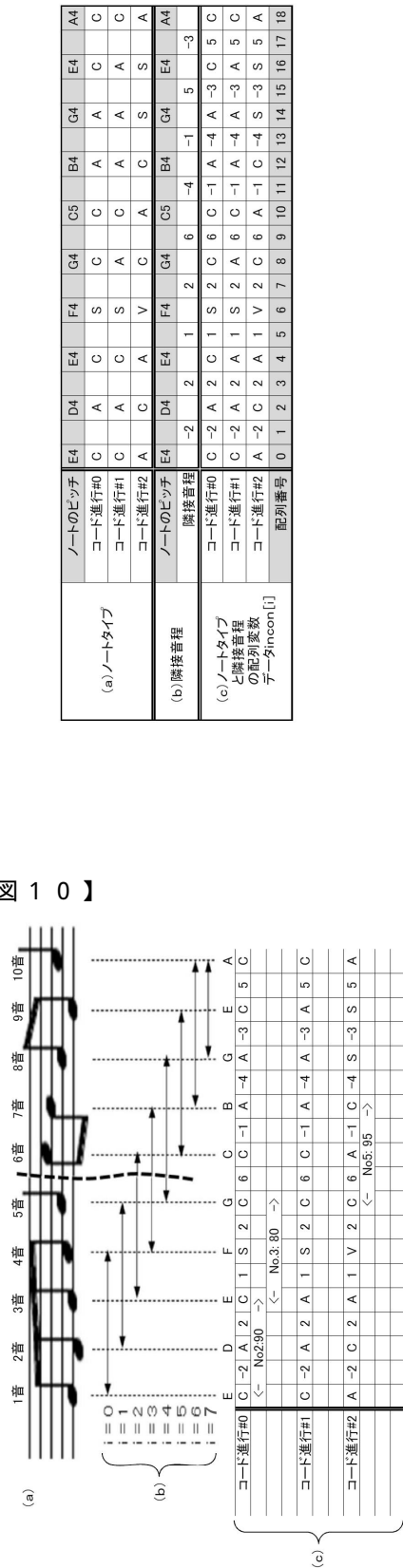
【図 6】

Measure	PartName[M]	iPartID[M]	ExistMelody[M]	iPartTime[M]
0	Null	0	0	
1	Intro	1	0	
2	Intro	1	0	
3	A	11	1	
4	A	11	1	
5	A	11	1	
6	A	11	1	
7	A	11	1	
8	A	11	1	
9	A	11	1	
10	A	11	1	
11	B	12	1	
12	B	12	1	
13	B	12	1	
14	B	12	1	
15	B	12	1	
16	B	12	1	
17	B	12	1	
18	B	12	1	
19	C	13	1	
20	C	13	1	
21	C	13	1	
22	C	13	1	
23	C	13	1	
24	C	13	1	
25	C	13	1	
26	C	13	1	
27	C	13	1	
28	A	11	1	
29	A	11	1	
30	A	11	1	
31	A	11	1	
32	A	11	1	
33	A	11	1	
34	A	11	1	
35	Ending	3	0	

【図 7】



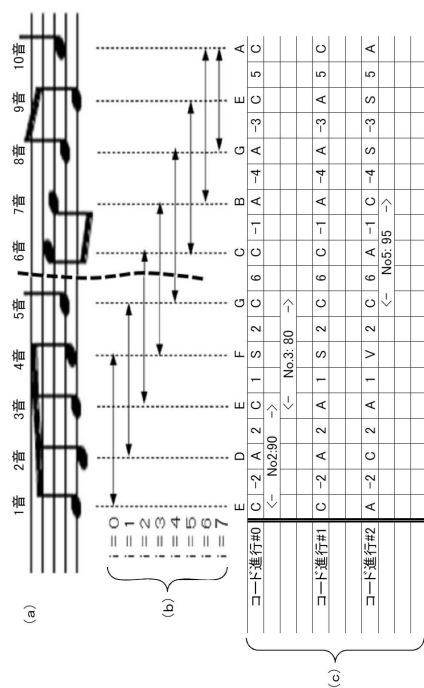
【 図 8 】



【 図 9 】

j	ノートタイプ#0	隣接音程#0	ノートタイプ#1	隣接音程#1	ノートタイプ#2	隣接音程#2	ノートタイプ#3	評価値
3音	コードトーン	0	ci, ChordTone	99	ci, ChordTone	99	ci, NullNote Type	100
	利鈍音	1	ci, ChordTone	-1	ci, AvailableNote	1	ci, ChordTone	0
	利鈍音	2	ci, ChordTone	-2	ci, AvailableNote	2	ci, NullNote Type	90
	経過音	3	ci, ChordTone	1	ci, ScaleNote	2	ci, ChordTone	0
	経過音	4	ci, ChordTone	2	ci, AvailableNote	2	ci, NullNote Type	80
	倚音	5	ci, ChordTone	0	ci, AvailableNote	-1	ci, ChordTone	0
4音	コードトーン	6	ci, ChordTone	99	ci, AvailableNote	-2	ci, NullNote Type	100
	倚音	7	ci, ChordTone	1	ci, AvailableNote	-2	ci, NullNote Type	95
	経過音	8	ci, ChordTone	2	ci, AvailableNote	99	ci, ChordTone	0
	経過音	9	ci, ChordTone	2	ci, AvailableNote	99	ci, NullNote Type	80
	コードトーン	10	ci, ChordTone	2	ci, AvailableNote	-3	ci, ChordTone	100
	利鈍音	11	ci, ChordTone	2	ci, AvailableNote	-3	ci, AvailableNote	1
4音	利鈍音	12	ci, ChordTone	2	ci, AvailableNote	-3	ci, ChordTone	80
	経過音	13	ci, ChordTone	2	ci, AvailableNote	2	ci, ChordTone	95
	経過音	14	ci, ChordTone	2	ci, AvailableNote	2	ci, ChordTone	95
	経過音	15	ci, ChordTone	-2	ci, AvailableNote	-2	ci, ChordTone	95
	経過音	16	ci, ChordTone	-2	ci, AvailableNote	-1	ci, ChordTone	95
	倚音	17	ci, ChordTone	99	ci, AvailableNote	-3	ci, AvailableNote	1
			↑	↑	↑	↑	↑	
			ci, NoteConnect[[0]	↑	↑	↑	↑	
			ci, NoteConnect[[1]	↑	↑	↑	↑	
			ci, NoteConnect[[2]	↑	↑	↑	↑	
			ci, NoteConnect[[3]	↑	↑	↑	↑	
			ci, NoteConnect[[4]	↑	↑	↑	↑	
			ci, NoteConnect[[5]	↑	↑	↑	↑	
			ci, NoteConnect[[6]	↑	↑	↑	↑	
			ci, NoteConnect[[7]	↑	↑	↑	↑	

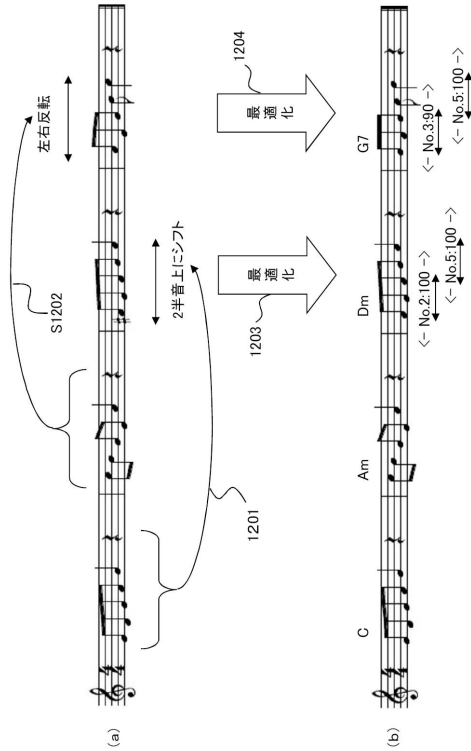
【 図 1 0 】



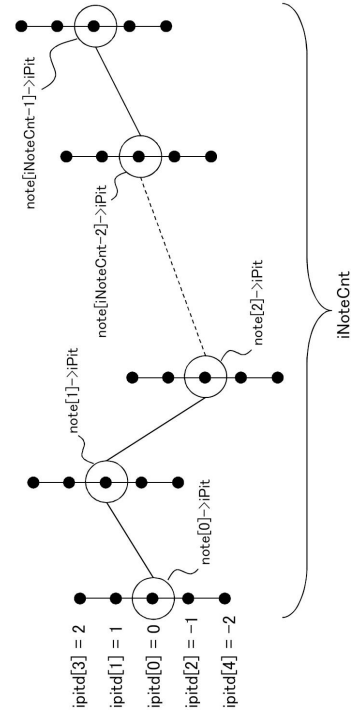
【 図 1 1 】

(a)	フレーズセット#0
	フレーズセット#1
	.
	.
	終端
(b)	Aメロデータ
	Bメロデータ
	Cメロ(サブメロディー)データ
	エンディング1データ
	エンディング2データ
(c)	ノートデータ#0
	ノートデータ#1
	.
	.
	.
	終端
(d)	時間
	長さ
	強さ
	音高

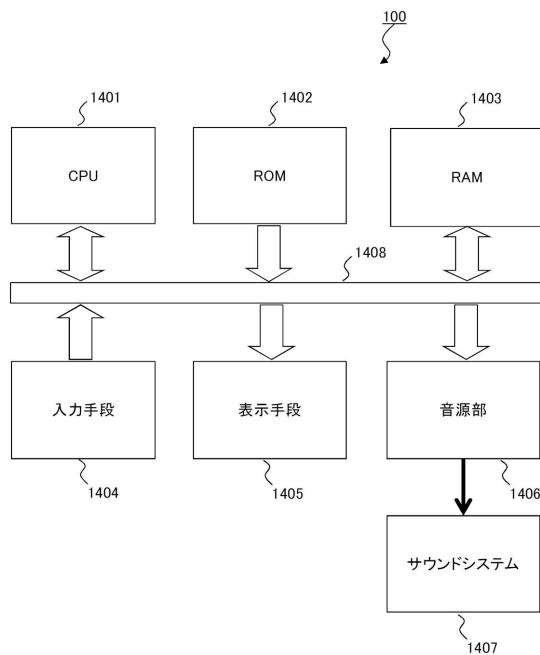
【図 12】



【図 13】



【図 14】



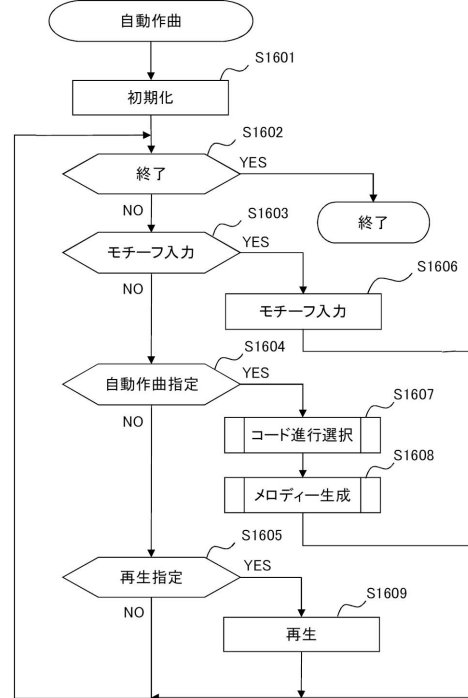
【図 15 A】

変数名	意味
n, i, j, k	繰返し処理制御用変数データ
MAX_CHORD_PROG	コード進行データの数を示す定数データ
iJunleSelect	楽曲ジャンル選択用変数データ
iChordAttribute[n][0]	コード進行#nの楽曲ジャンルを示す配列変数データ
iConceptSelect	楽曲コンセプト選択用変数データ
iChordAttribute[n][1]	コード進行#nの楽曲コンセプトを示す配列変数データ
iKeyShift	キーシフト値を示す変数データ
PITCH_CLASS_N	キーシフトの数を示す定数データ
doValue	適合度を示す変数データ
doMaxValue	適合度の最大値を示す変数データ
iBestUpdate	n番目の時点での最良コード進行を指す変数データ
iBestKeyShift[n]	n番目の時点での最良コード進行のキーシフト値
iBestChordProg[n]	n番目の時点での最良コード進行の番号
iCDesignCnt	コード進行内の情報番号を示す変数データ
cdesign[iCDesignCnt]	iCDesignCnt番目のコードデザインデータ
cdesign[iCDesignCnt]->iTime	コードデザインデータの時間情報
cdesign[iCDesignCnt]->iRoot	コードデザインデータのコード根音情報
cdesign[iCDesignCnt]->iType	コードデザインデータのコードタイプ情報
cdesign[iCDesignCnt]->iKey	コードデザインデータのキー情報
cdesign[iCDesignCnt]->iScale	コードデザインデータのスケール情報
mt	メタイベントを指すポインタ変数データ
root, type, scale, key	コード根音、コードタイプ、スケール、キーを示す変数データ
sTime	小節開始時間を示す変数データ
iNoteCnt	音列のノート番号を示す変数データ
me, me->iTime	ノートを指すポインタ変数データ、その時間項目
notes[iNoteCnt]	ノートポインタ配列変数データ
iPit	ノートのピッチ項目値
ipit[i]	ピッチ情報配列変数データ
incon[i × 2], incon[i × 2 - 1]	i番目のノートのノートタイプと隣接音程の配列変数データ
pcs1	コードトーンのピッチクラスセットを格納する変数データ
pcs2	テンションノートのピッチクラスセットを格納する変数データ
pcs3	スケールノートのピッチクラスセットを格納する変数データ
pc1, pc2	候補ピッチクラス1, 2を示す変数データ
ci ChordTone	コードトーンを示す定数データ
ci AvailableNote	アウェイラブルノートを示す定数データ
ci ScaleNote	スケールノートを示す定数データ
ci TensionNote	テンションノートを示す定数データ
ci AvoidNote	アヴォイドノートを示す定数データ
iTotalValue	総合評価値を示す変数データ
iValue	評価値を示す変数データ
iMaxValue	最大評価値を示す変数データ
ci NoteConnect[i][k × 2]	i番目のノート接続ルールのk番目の要素
ci NoteConnect[i][k × 2 - 1]	

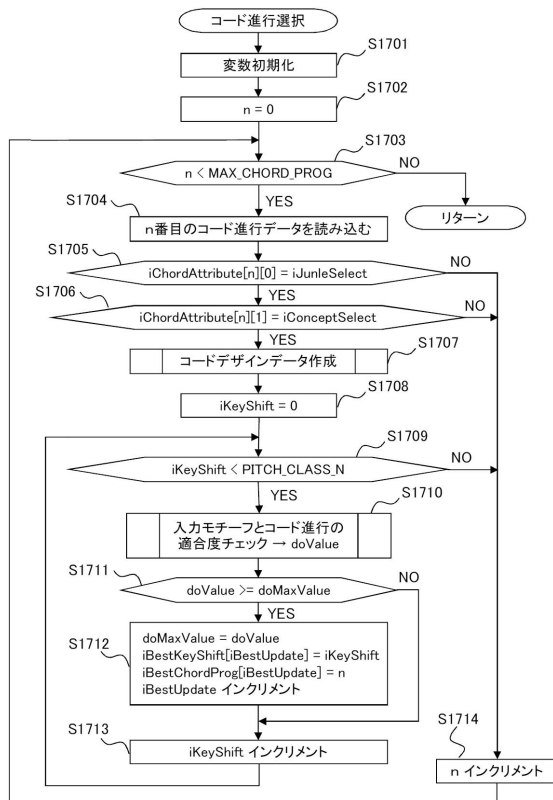
【図 15 B】

変数名	意味
iMelodyA[0]~iMelodyA[iLengthA-1]	モチーフDB上のフレーズのピッチ列配列変数データ
iMelodyB[0]~iMelodyB[iLengthB-1]	入力モチーフのピッチ列
iLengthA	モチーフDB上のフレーズのピッチ列の長さ変数データ
iLengthB	入力モチーフのピッチ列の長さ変数データ
doDistance	距離評価値を示す変数データ
doMin	最小距離評価値を示す変数データ
iBestMochief	最良のフレーズセットを示す変数データ
MAX NOTE CANDIDATE	あるノートに対する別ピッチ候補の数
iWnum	音列のノート数すべての別ピッチ候補の数
ipitd[]	或るノートに対する別ピッチ候補(差分)
ipitdev	ピッチ修正値を示す変数データ
iCnt	別ピッチ候補カウント用の変数データ
iMaxValue	最良適合度を示す変数データ
iMaxCnt	最良カウンタを示す変数データ

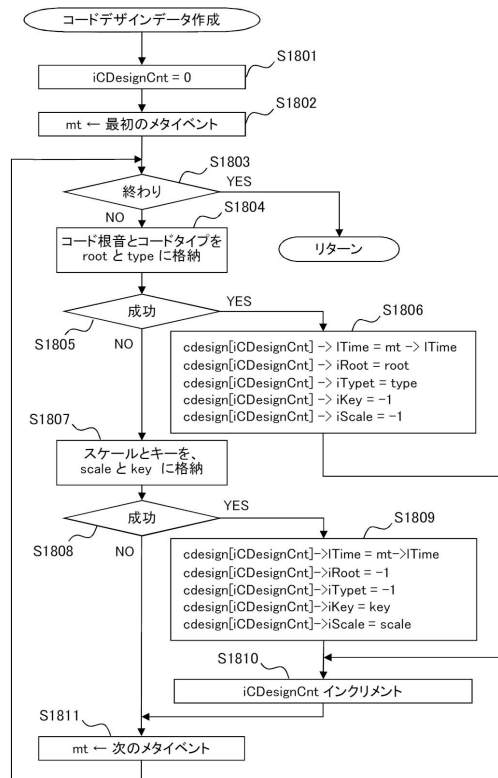
【図 16】



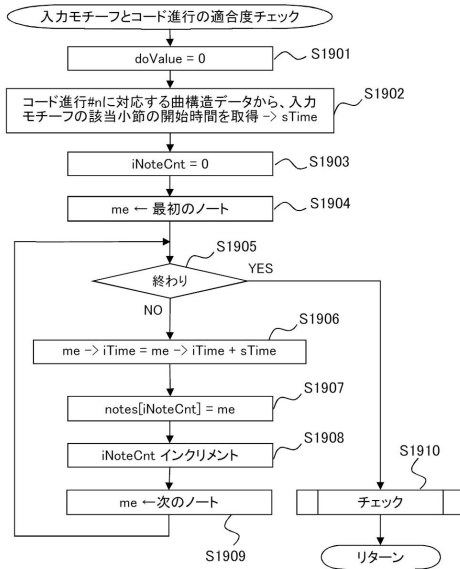
【図 17】



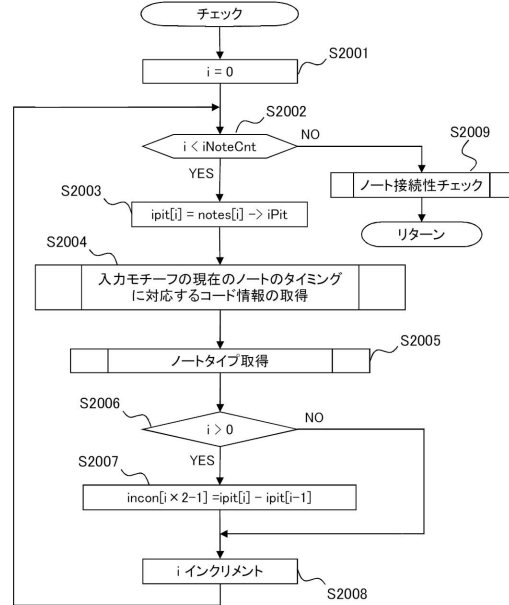
【図 18】



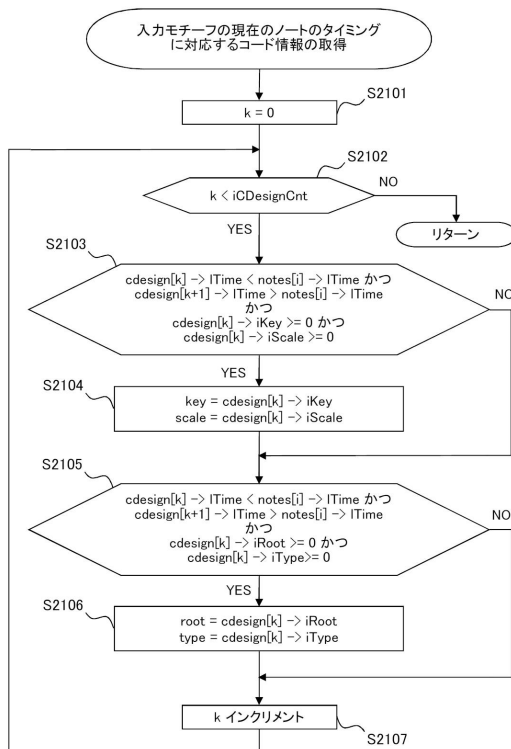
【図 19】



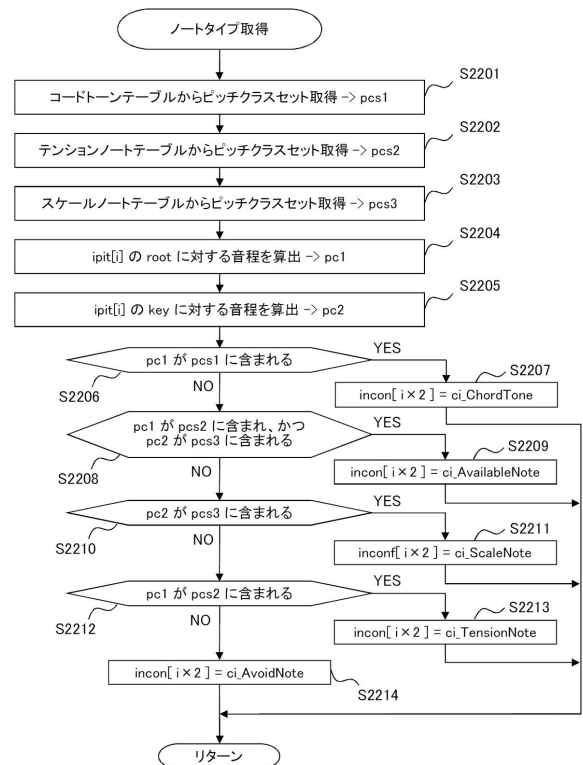
【図 20】



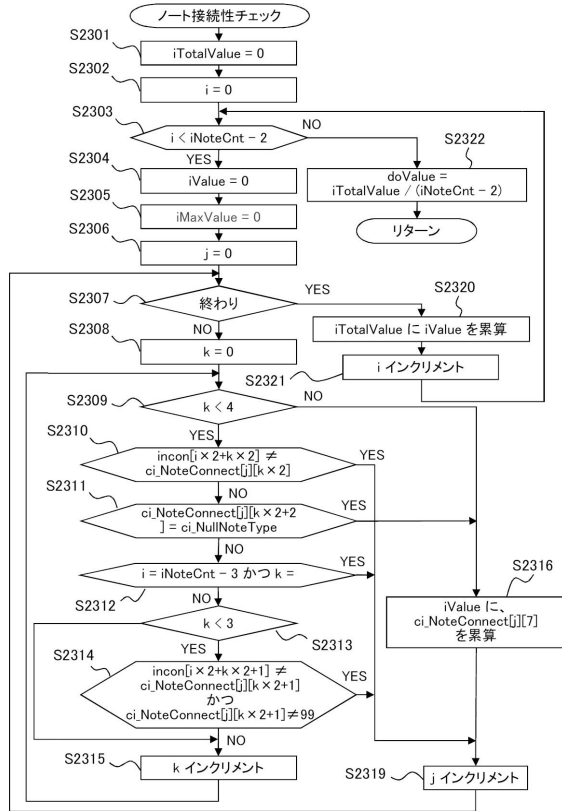
【図 21】



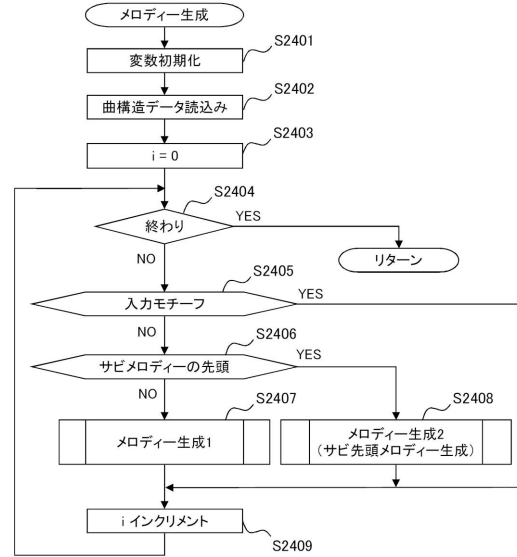
【図 22】



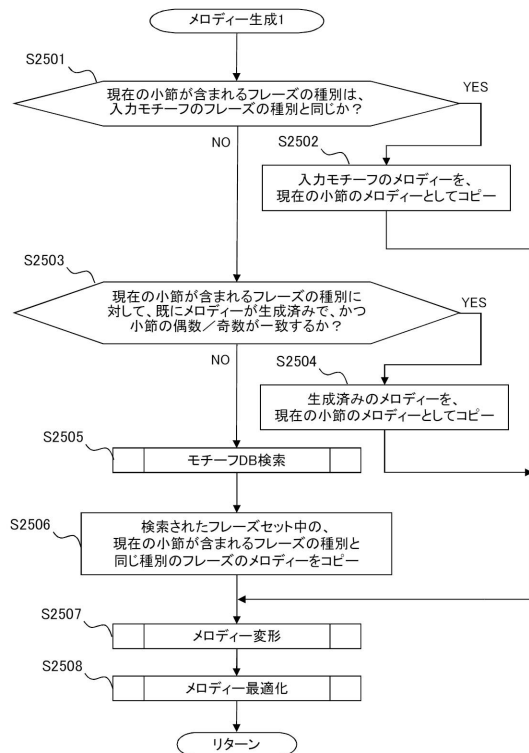
【図 23】



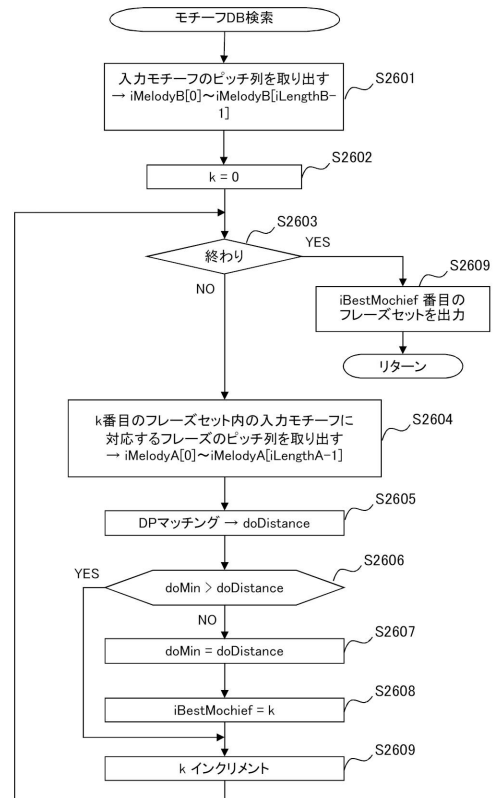
【図 24】



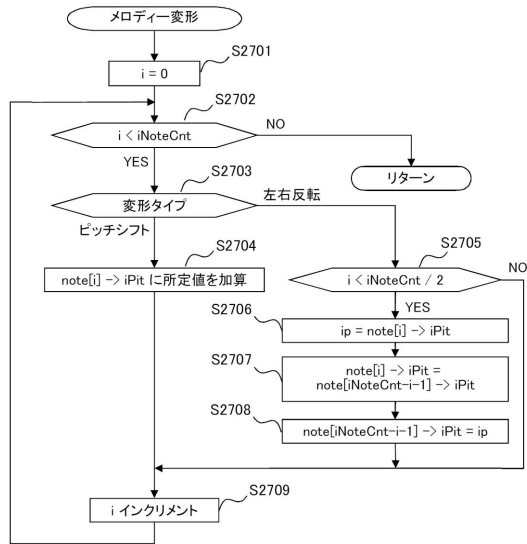
【図 25】



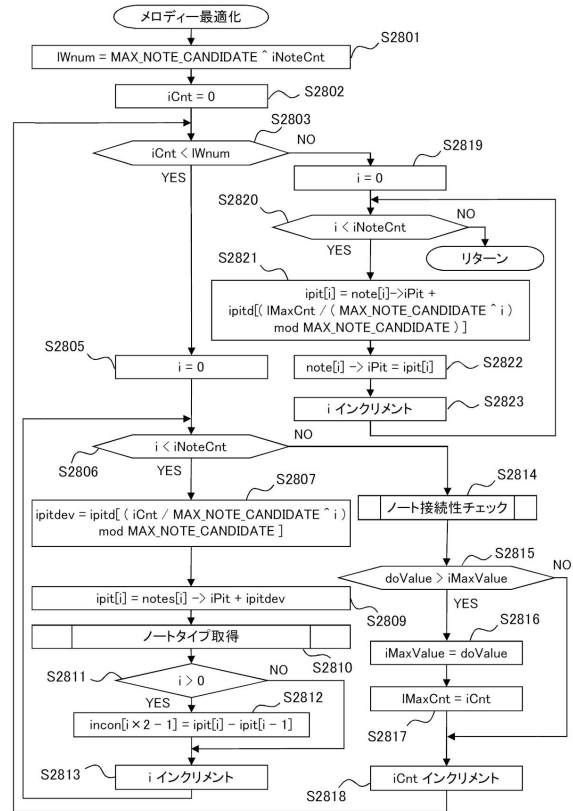
【図 26】



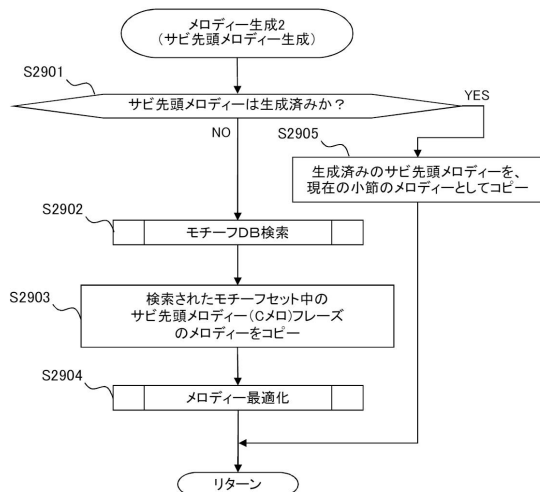
【図 27】



【図 28】



【図 29】



フロントページの続き

(56)参考文献 特開2002-32077(JP,A)
特開2002-32078(JP,A)
特開平1-167781(JP,A)
特開平4-110884(JP,A)
特開2005-173492(JP,A)
国際公開第96/24422(WO,A1)

(58)調査した分野(Int.Cl., DB名)

G10H	1/00 - 1/46
G10G	1/00 - 3/04