E-027

プログラムからの楽曲生成
Translate Programs into Music

渡邉 彰吾 †　　六沢 一昭 ‡
WATANABE Syogo　　ROKUSAWA Kazuaki

1. はじめに

本稿は, 聴いて楽しむことのできる楽曲をプログラムから生成するシステムについて述べる. プログラムから受ける印象を, プログラムの構造を反映した楽曲で表現することがこのシステムの目標である.

プログラム開発支援やプログラミング学習支援を主な目的としたプログラム可聴化システムがこれまで数多く開発されてきた [1][2][3][4][5][6]. これらは主にプログラムの実行状況の可聴化を行なうものであった. また, 可聴化として生成されるものは, 楽曲というよりは音の集合であり, 聴いて楽しむというエンターテインメント性の要素はほとんどなかった.

本システムは Perl 言語で記述されたプログラムを対象とする. ネスト構造や制御構造など, プログラムを見た時の印象に影響する情報を解析し, それらを反映した楽曲を生成する. 聴いて楽しむことができ, 元のプログラムについても何らかの想像ができるような楽曲を生み出すことが最終的な目標である.

2. 自然な楽曲の生成

自然な楽曲を生成するために着目したコードと規則を以下に示す.

2.1 ダイアトニックコード (Diatonic Chord)

ダイアトニックコードとは, スケール上の音で構成される三和音 (Triad) または四和音 (Tetrad) のことである. またスケールとは音階のことで, オークターブ内の音の配列である. 本研究では, 三和音と長調 (メジャースケール) を扱う. 使用するコードの一覧を表 1 に示す.

2.2 カデンツ (Kadenz)

カデンツとは, 終止形和音進行のことである. 文章で言う所の起承転結のようなものである. コード進行はカデンツに則って構成する. これにより, 自然な楽曲の生成を試みる. カデンツを組み立てる上での和音機能は 3 種類存在する.

表 1: 各調のダイアトニックコード (三和音)

調＼度数表記	I	$II m$	$III m$	IV	V	$VI m$	$VII m^{\flat 5}$
ハ長調	C	Dm	Em	F	G	Am	$Bm^{\flat 5}$
嬰ハ長調	$C\#$	$D\#m$	Fm	$F\#$	$G\#$	$A\#m$	$Cm^{\flat 5}$
ニ長調	D	Em	$F\#m$	G	A	Bm	$C\#m^{\flat 5}$
嬰ニ長調	$D\#$	Fm	Gm	$G\#$	$A\#$	Cm	$Dm^{\flat 5}$
ホ長調	E	$F\#m$	$G\#m$	A	B	$C\#m$	$D\#m^{\flat 5}$
ヘ長調	F	Gm	Am	$A\#$	C	Dm	$Em^{\flat 5}$
嬰ヘ長調	$F\#$	$G\#m$	$A\#m$	B	$C\#$	$D\#m$	$Fm^{\flat 5}$
ト長調	G	Am	Bm	C	D	Em	$F\#m^{\flat 5}$
嬰ト長調	$G\#$	$A\#m$	Cm	$C\#$	$D\#$	Fm	$Gm^{\flat 5}$
イ長調	A	Bm	$C\#m$	D	E	$F\#m$	$G\#m^{\flat 5}$
嬰イ長調	$A\#$	Cm	Dm	$D\#$	F	Gm	$Am^{\flat 5}$
ロ長調	B	$C\#m$	$D\#m$	E	$F\#$	$G\#m$	$A\#m^{\flat 5}$

トニック **(T)** – 最も安定した響きを持ち, 曲のはじめと終わりによく使われる.

ドミナント **(D)** – 単独では不安定な響きを持ち, トニックへ進もうとする.

サブドミナント **(S)** – 単独では不安定な響きを持ち, トニックかドミナントへ進もうとする.

また, カデンツには以下の 3 種類の型がある.

カデンツ第 **1** 型 **(K1)** – T-S-T

カデンツ第 **2** 型 **(K2)** – T-S-D-T

カデンツ第 **3** 型 **(K3)** – T-D-T

各 3 和音の機能別分類を表 2 に示す.

表 2: 3 和音の機能別分類

和音機能	該当する 3 和音
トニック (T)	I,$III m$,$VI m$
サブドミナント (S)	$II m$,IV
ドミナント (T)	V,$VII m^{\flat 5}$

これまで, 様々なコード進行が考えられてきた. J-POP によく使われる代表的なコード進行に以下がある.

†千葉工業大学 大学院 情報科学研究科 情報科学専攻
‡千葉工業大学 情報科学部 情報工学科

カノン進行 – $|C|G|Am|Em|F|C|F|G|$
王道進行 – $|F|G|E7|Am|$
小室進行 – $|Am|F|G|C|$

本システムではこれらの進行を用い, より楽しめる楽曲生成を試みる.

3. プログラムから得られる印象

プログラムを見た時, 以下から印象が作られるのは自然であると思われる.

3.1 ネストの深さの変化

ネストの深さは分岐やループがあると変化する. ネストが一定の部分からは単調な印象を受ける. これに対して, ネストが深くなる/戻る部分からは動きを感じる.

3.2 制御構造の種類

while, for, if (then, else) といった分岐やループの制御構造を見ると, それぞれに応じたプログラムの振る舞いを予測する.

3.3 各行の複雑さ

閉じ括弧のみの行や, 変数に定数を代入するだけの行は単純に感じる. 一方, 多くの変数や演算子, 関数呼び出しを含んだ計算式や条件式などからは複雑な印象を受ける.

4. プログラムから楽曲へ

4.1 楽曲生成の流れ

プログラムの 1 行を楽曲の 1 小節に対応させて楽曲を生成する.

まず, プログラムのネストの深さ (の変化) から各小節のコード (の変化) を決定する. そして, 対応するプログラムの行の内容を考慮してリズムを決定し, コードをもとにメロディ生成する. 同様にハーモニーも決定する. 各小節のメロディを構成する音は基本的にコードの構成音のみとする. この「まずコードを決め, コードの構成音からメロディを決定する」方式は [7][8] で用いたものである.

4.2 コードの決定

プログラムのネストの深さの変化をプログラムの進行と捉え, これをコード進行 (コードの変化) で表現する.

コード決定方式として以下の二つを用いる.

方式 A

表 3 から表 5 に基づき, 4 回あるいは 8 回のネストの深さの変化を, 前述した 3 種類のコード進行 (カノン, 王道, 小室) に対応させて 4 小節あるいは 8 小節のコードを決定する. その際, ネストの深さの等しい連続する行はまとめて 1 行と考える.

表 3: ネストとコード進行 (カノン進行)

ネストの深さの変化	+1	+1	+1	+1	–1	–1	–1	–1
コード進行	C	G	Am	Em	F	C	F	G

表 4: ネストとコード進行 (王道進行)

ネストの深さの変化	+1	–1	+1	–1
コード進行	F	G	$E7$	Am

表 5: ネストとコード進行 (小室進行)

ネストの深さの変化	+1	+1	–1	–1
コード進行	Am	F	G	C

方式 B

ひとつのネストの深さの変化と, 安定したコード進行 (図 1) からひとつの小節のコードを決定する.

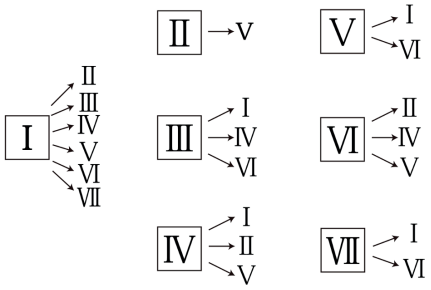

図 1: 和音進行表

まず方式 A によるコード決定を試みる. これによりコードが決定できなかった場合は方式 B によりコードを決定する.

コードの決定例を図 2, 図 3 に示す.

ネスト	コード
0	C
1	Am
2	F
1	G
0	C

```
for ( $i = 0 ; $i <= 0 ; $i++ ) {
    if ( $i % 2 == 0 ) {
        print "$i";
    }
}
```

図 2: コードの遷移例 1

ネスト	コード	
0	C	`if ( $i % 2 == 0 ) {`
1	F	`  print "#";`
0	G	`}`
0		`else {`
1	$E7$	`  print "*";`
0	Am	`}`

図 3: コードの遷移例 2

図 2 はネストの変化が +1,+1,–1,–1 であるため, 小室進行 (表 5) が適用された. 同様に, 図 3 はネストの変化が +1,–1,+1,–1 であるため, 王道進行 (表 4) が適用された. また,3 行目と 4 行目はネストの深さが等しいため, まとめてコードが割り当てられた.

4.3 リズムの決定

プログラムの各行について複雑さを考える. 行が複雑であるほど, 対応する小節のリズムを激しくするのは自然であろう.

プログラムの各行をパーツ (構成要素) に分ける. 例を図 4 に示す. このパーツの数が (プログラムの) 行の複雑さを表わすものとした.

プログラムの行をパーツに分け, パーツ数を求め, 表 6 に基づきリズム (音符の数) を決定する.

表 6: パーツ数とリズム

パーツ	リズム	パーツ	リズム
1	𝅝	9	♩ ♫ ♫ 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯
2	𝅗𝅥 𝅗𝅥	10	♫ ♫ ♫ 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯
3	𝅗𝅥 ♩ ♩	11	♩ ♫ 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯
4	♩ ♩ ♩ ♩	12	♫ ♫ 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯
5	♩ ♩ ♩ ♫	13	♩ 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯
6	♩ ♩ ♫ ♫	14	♫ 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯
7	♩ ♫ ♫ ♫	15	♬ 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯
8	♫ ♫ ♫ ♫	16	𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯 𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯𝅘𝅥𝅯

```
for ( $i = 0 ; $i < 9 ; $i ++ ) {
    if ( $i % 2 == 0 ) {
        print "$i¥n" ;
    }
}
```

図 4: パーツ分解例

4.4 メロディの決定

ある小節のメロディの決定は, その小節に対応する (プログラムの) 行のパーツの数だけ構成音を生成することである. 前述したように, ある小節のメロディはその小節のコードの構成音から構成する.

各パーツを構成する文字/記号列は楽譜の音符に対応すると考えた. すると, メロディ構成音の生成は, パーツの文字/記号列の音符への変換となる. コードの構成音は高々3 つであり, また構成音の高さは 2 オクターブ程度に収まるのが適当であろう. このことから, パーツを構成する文字/記号のコードの合計値を 7 で割った余りを求め (これをメロディ値と呼ぶ), この値をもとにメロディ構成音を決定する. メロディ値とメロディ構成音の対応を表 7 と図 5 に示す.

表 7: コードの構成音 (ハ長調)

コード \ メロディ値	0	1	2	3	4	5	6	7	8
C	$C4$	$E4$	$G4$	$C5$	$E5$	$G5$	$C6$	$E6$	$G6$
Dm	$D4$	$F4$	$A4$	$D5$	$F5$	$A5$	$D6$	$F6$	$A6$
Em	$E4$	$G4$	$B4$	$E5$	$G5$	$B5$	$E6$	$G6$	$B6$
F	$F4$	$A4$	$C5$	$F5$	$A5$	$C6$	$F6$	$A6$	$C7$
G	$G3$	$B3$	$D4$	$G4$	$B4$	$D5$	$G5$	$B5$	$D6$
Am	$A3$	$C4$	$E4$	$A4$	$C5$	$E5$	$A5$	$C6$	$E6$
$Bm^{\flat 5}$	$B3$	$D4$	$F4$	$B4$	$D5$	$F5$	$B5$	$D6$	$F6$

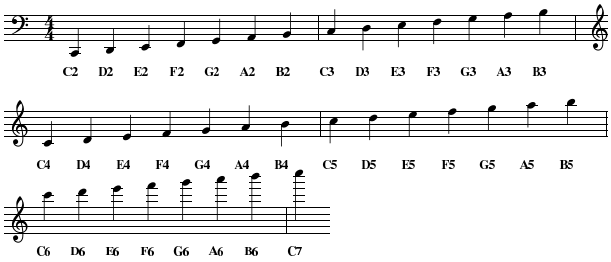

図 5: 構成音

4.5 ハーモニー (伴奏パターン) の決定

伴奏はメロディを支え引き立てるものである. 一方, 制御構造 (if や while など) はプログラムの骨格であり, 二つの間に共通点が見られる. このことから制御構造から伴奏パターンを決定することにした.

制御構造と伴奏パターンの関係を図 6 から図 9 に示す.

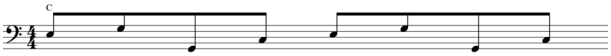

図 6: for の伴奏パターン

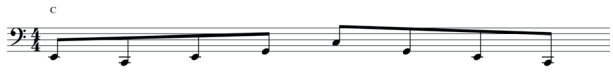

図 7: while の伴奏パターン

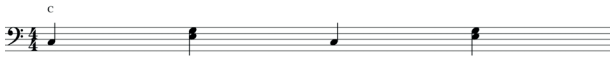

図 8: if の伴奏パターン

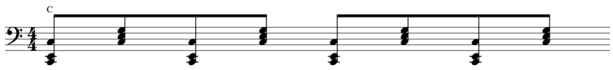

図 9: else の伴奏パターン

4.6 転調

ひとつのプログラムは, 一般に, ひとつの主プログラムと複数のサブルーチンからなる. プログラムリストを 1 行目から見て行くと, 主プログラムが終わり (一つ目の) サブルーチンが始まる. そして, ひとつのサブルーチンが終わると, 別のサブルーチンが始まる.

プログラムを見た印象は, 新しいサブルーチンの始まりで大きく変化すると思われる. この印象の変化は転調で表現する. サブルーチンの開始時点で転調を行なう.

5. システムの構成

楽曲生成対象プログラムは Perltidy*によって整形され, 本システムによって楽曲が生成される. 生成された楽曲は MML**で記述される. システムの構成を図 10 に示す.

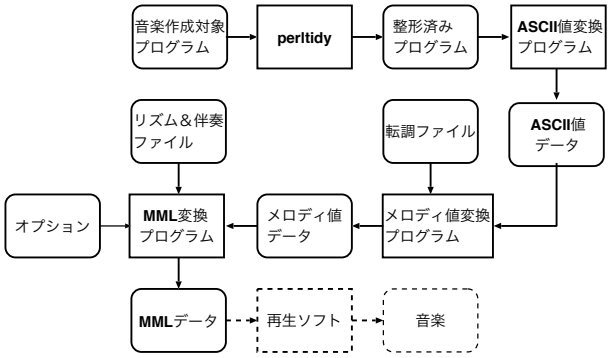

図 10: システム構成図

6. 生成例

図 11 のプログラムから得られた情報を表 8 に, その情報から生成された楽曲を図 12 に示す.

```
$fin = 15;
$sum = 0;
for ( $i = 0 ; $i <= $fin ; $i++ ) {
    if ( $i % 2 == 0 ) {
        print "#";
    }
    else {
        print "*";
        $sum += $i;
    }
    print "\n";
}
if ( $sum % 2 == 0 ) {
    print "$sum\n";
}
```

図 11: 整形済み楽曲生成対象 Perl プログラム 1

表 8: 図 11 の Perl プログラムの各行の情報

行	字下げ	コード	パーツ数	制御構造
1	0	C	4	other
2	0	F	4	other
3	0	G	10	for
4	1	Am	7	if
5	2	F	3	if
6 7	1	G	3	else
8 9	2	$E7$	6	else
10	1	Am	1	for
11	1	F	3	other
12	0	C	1	other
13	0	F	7	if
14	1	G	3	if
15	0	C	1	if

*Perl プログラムの整形ツール
**Music Macro Language, 楽曲記述言語の一つ

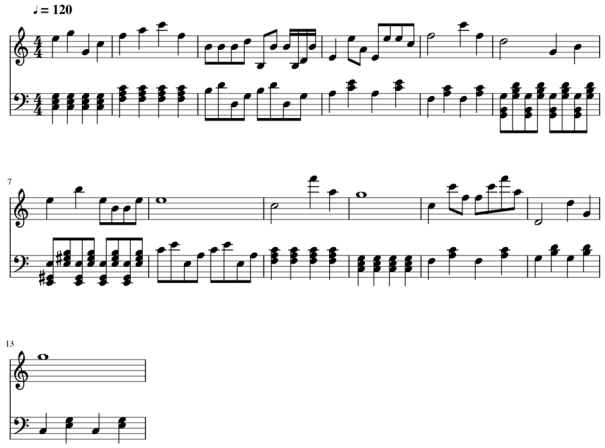

図 12: 図 11 の Perl プログラムから生成された楽曲

図 13 のプログラムから得られた情報を表 9 に, その情報から生成された楽曲を図 14 に示す.

```
my $n = <STDIN>;
chomp $n;
if ( $n > 0 ) {
    print "$n\n";
}
else {
    my $np = $n * $n;
    print "$np\n";
}

for ( my $i = 0 ; $i < 64 ; ++$i ) {
    if ( ( $i % 2 ) == 0 ) {
        print " $i";
    }
}
print "\n";

for ( my $k1 = 0 ; $k1 < 4 ; ++$k1 ) {
    for ( my $k2 = 0 ; $k2 < 4 ; ++$k2 ) {
        for ( my $k3 = 0 ; $k3 < 4 ; ++$k3 ) {
            my $sum = $k1 + $k2 + $k3 + 1;
            while ( $sum < 100 ) {
                print " $sum";
                $sum = $sum * 2;
            }
            print "\n";
        }
    }
    print "\n";
}
```

図 13: 整形済み楽曲生成対象 Perl プログラム 2

表 9: 図 13 の Perl プログラムの各行の情報

行	字下げ	コード	パーツ数	制御構造
1	0	C	5	other
2	0	F	3	other
3	0	C	5	if
4	1	F	3	if
5 6	0	G	3	if
7 8	1	$E7$	9	else
9	0	Am	1	else
10	0	G	10	for
11	1	Am	7	if
12	2	F	4	if
13	1	G	1	if
14	0	C	1	for
15	0	F	3	other
16	0	G	10	for
17	1	C	10	for
18	2	G	10	for
19 20	3	Am	16	for
21 22	4	Em	9	while
23 24	3	F	5	while
25	2	C	1	for
26 27	1	F	4	for
28	0	G	1	for

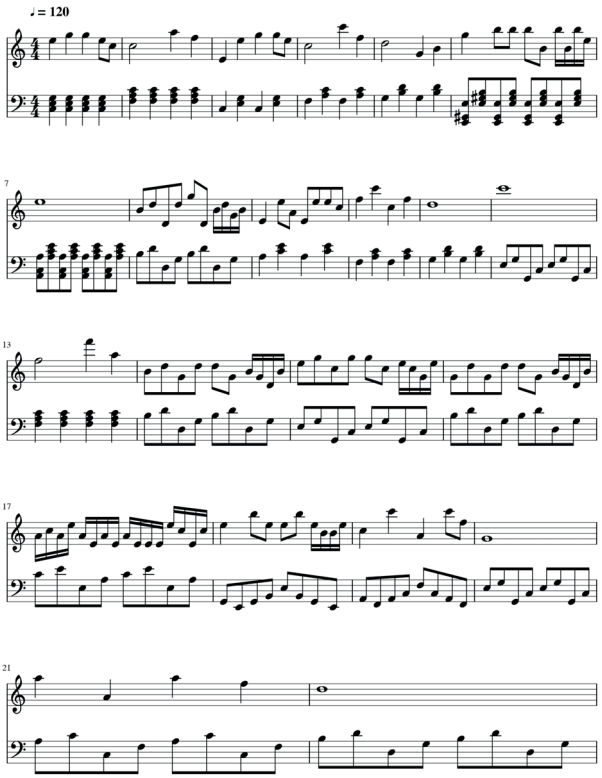

図 14: 図 13 の Perl プログラムから生成された楽曲

どちらのプログラムもネストの変化が多い. これより, コードの変化が多く, 楽曲が複雑になっていることがわかる.

また, ネストの深さの変化に応じて表 3 から表 5 に基づき, コード進行が適用されていることがわかる.

さらに, 図 13 のプログラムは表 3 から表 5 の全てが適用されている. これより, 制御構造とネストの変化が楽曲で表現できていることがわかる.

7. まとめ

Perl プログラムから楽曲を生成した. コード進行の検討の下, メロディや伴奏を決定し, 自然な楽曲生成を目指した.

参考文献

[1] 佐藤 和哉, 丸山 一貫, 寺田 実 : “CodeMusician:プログラム実行可聴化の試み”, 第 3 回エンターテイメントと認知科学シンポジウム (2009).

[2] 佐藤 和哉, 平井 重行, 丸山 一貴, 寺田 実 : “CodeDrummer: リズムに着目したプログラム可聴化システム,” インタラクション 2011, 2SCL-14, 情報処理学会 (2011).

[3] 大澤 範高 : “継承関係の可聴化,” 日本ソフトウェア科学会第 8 回インタラクティブシステムとソフトウェアに関するワークショップ, 近代科学社, (2000).

[4] D. B. Boardman : *et al.*, “LISTEN: A Tool to Investigate the Use of Sound for the Analysis of Program Behavior,” COMPSAC’95, pp.184–189 (1995).

[5] P. Vickers and J. Alty : “Siren songs and swan songs: Debugging with music,” *Communications of the ACM*, Vol.46, No.7, pp.86–93 (2003).

[6] Andreas Stefik, *et al.* : “Layered Program Auralization: Using Music to Increase Runtime Program Comprehension and Debugging Effectiveness,” *ICPC’06*, pp.89–93, (2006).

[7] 杉原 大和, 六沢 一昭 : “プログラムからの楽曲生成の試み”, 第 5 回エンターテイメントと認知科学シンポジウム (2011).

[8] 渡邉 彰吾, 六沢 一昭 : “プログラムからの楽曲生成”, 情報処理学会 研究報告音楽情報科学, Vol.2017-MUS-115, No.25 (2017).

[9] 島岡 譲 他 10 名 : “和声 理論と実習”, 株式会社音楽之友社 (1964).

[10] テキスト音楽「サクラ」, https://sakuramml.com (2018).