The 33rd Annual Conference of the Japanese Society for Artificial Intelligence, 2019

プログラムの静的特性を楽曲で表現する可聴化システムの開発とプログラミング支援への応用

Development of Auralization System to Represent Program by Music and its Application to Programming Support

渡邉 彰吾 *1　　六沢 一昭 *2
Syogo Watanabe　　Kazuaki Rokusawa

*2千葉工業大学 情報科学部 情報工学科
Department of Computer Science, Chiba Institute of Technology

*1千葉工業大学大学院 情報科学研究科 情報科学専攻
Graduate School of Information and Computer Science, Chiba Institute of Technology

This paper describes development of auralization system to represent static characteristics of program by music and its application to programming support. The system generates rhythms, chords, tunes and harmony aiming for comfortable music. The music also represents violation of coding standards by a dissonance.

1. はじめに

ソフトウェア開発ではソースコードの保守性や信頼性などが重視されている. これらを向上する手法として静的解析がある. 静的解析では命名規則や書式, コーディング規約違反などの検出が期待される. 静的解析ツールは用途に応じて多数存在している [桑原 2015]. これらのツールで得られた解析結果は画面に出力される. 一方, 本研究では静的解析結果を音に出力する. 音に出力することで, ながら作業が可能となる. 本稿では, 静的解析ツールとして静的解析結果可聴化システムを開発する.

可聴化における様々な研究がされてきた [P. Vickers 2003] [岩田 2013]. その中で, 音の羅列で可聴化したシステムは単調で飽きやすいといった問題があった [佐藤 2011]. そこで本研究では, 自然な楽曲による可聴化を試みる. 自然な楽曲にすることで, 飽きにくさが期待できる. また, プログラム変更による楽曲の変化により, 作業が楽しく感じることも期待できる.

2. 目的

本研究では, ソースコードを解析し, 得られた情報から自然な楽曲の生成を行なう. 規約に基づいてコードを解析し, 規約違反箇所を不協和音で表現する. 生成された楽曲を聴き, 不協和音の響きによって規約違反箇所を発見できるか検証する.

3. 自然な楽曲の生成

[渡邉 2018] の手法を用いて自然な楽曲を生成する.

3.1 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. ハ長調のダイアトニックコードを表 1 に示す. 表中の T, D, S はコードの機能を示し, 以下の特徴がある.

表 1: ダイアトニックコード (ハ長調)

コード	C	Dm	Em	F	G	Am	$Bm^{\flat 5}$
和音機能	T	S	T	S	D	T	D

トニック **(T)** – 曲のはじめと終わりによく使われる.
ドミナント **(D)** – トニックへ進もうとする.
サブドミナント **(S)** – トニックかドミナントへ進もうとする.

3.2 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. カデンツには T-S-T, T-S-D-T, T-D-T の 3 つの型がある.

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

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.

4. 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. 制御構造から伴奏を決定する.

連絡先: 六沢一昭, 千葉工業大学 情報科学部 情報工学科,
275-0016 千葉県習志野市津田沼 2-17-1,
rokusawa.kazuaki@it-chiba.ac.jp

4.1 コードの決定

コードの決定方式として以下の二つを用いる.

方式 A

表 2 から表 4 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.

表 2: 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 3: 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 4: 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

ネストの深さに応じてコードを決定する.

表 5: ネストの深さとコード (ハ長調)

ネストの深さ	0	1	2	3	4	5	6	7
コード	C	G	Am	Dm	Em	F	C	G

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. 方式 A では, ネストの変化が 0 の時を省いた状態で変化パターンに当てはまるかを考える.

コードの決定例を図 1 に示す.

行	ネスト	変化量	コード	
1	0	0	C	`if ( i % 2 == 0 ) {`
2	1	+1	F	`    printf( "#" );`
3	0	–1	G	`}`
4	0	0	G	`else {`
5	1	+1	Em	`    printf( "*" );`
6	0	–1	Am	`}`

図 1: コード決定例

2 行目から 6 行目はネストの変化が +1, –1, 0, +1,–1 である. 0 を省くと, 王道進行 (表 3) と同じネストの変化パターンとなる. このことから, 2 行目から 6 行目は王道進行が適用されていることがわかる. 適用された行において, ネストの変化が 0 の行は一つ前の行に割り当てられたコードと同じコードを割り当てる. これにより, 割り当てられたコードは F, G, G, Em, Am となっている.

4.2 音楽の三要素

リズム、メロディ、ハーモニーの対応を以下に示す。

リズム – パーツ数
メロディ – パーツの ASCII 値
ハーモニー – 制御構造

5. コーディング規約

本研究では, コーディング規約として MISRA-C:2012 を採用した. MISRA-C とは, MISRA[*1] が制定した自動車関連ソフトウェア向けの C 言語の規約である. 現在では, 組み込み開発分野で広く利用されている規約であるため, C 言語の規約として最適だと考えた.

6. 規約違反の楽曲への織込み

コーディング規約違反箇所を不協和音で表現する. 不協和音とは, 全体が調和せず不安定な印象を与える和音である. 本研究では, 不協和音として減三和音 (Diminished-Triad) を用いる. この減三和音は非常に不安定な和音として知られている. このコードを用い, 音楽を大幅に逸脱することなく規約違反箇所を表現する.

7. システムの構成

楽曲生成対象プログラムは indent コマンドによって整形し, 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[*2] で記述される. その後, MIDI データに変換する. システムの構成を図 2 に示す.

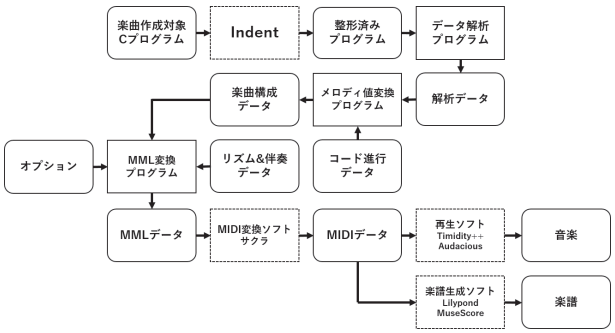

図 2: システム構成図

8. 楽曲生成例

図 3 のプログラムから得られた情報を表 6 に, その情報から生成された楽曲を図 4 に示す.

*1 MISRA(Motor Industry Software Reliability Association): 自動車メーカーや部品サプライヤー, エンジニアリングコンサルタントが連携したヨーロッパの団体

*2 MML(Music Macro Language): 楽曲記述言語の一つ

```
 1  #include <stdio.h>
 2  #define NUM 100
 3  int main( void )
 4  {
 5      int i;
 6      for ( i = 1; i <= NUM; i++ ) {
 7          if ( i % 3 == 0 ) {
 8              printf( "%d:fizz", i );
 9              if ( i % 5 == 0 )
10                  printf( "buzz" );
11          }
12          else if ( i % 5 == 0 ) {
13              printf( "%d:buzz", i );
14          }
15          printf( "\n" );
16      }
17      return ( 0 );
18  }
```

図 3: 整形済み楽曲作成対象 C プログラム

表 6: 図 3 の各行の解析データ

行	字下げ	コード	パーツ数	制御構造	規約チェック
1	0	C	2	other	ok
2	0	C	3	other	ok
3	0	C	2	other	ok
4	0	C	1	other	ok
5	1	G	3	other	ok
6	1	G	10	for	ok
7	2	Am	7	if	ok
8	3	Dm	4	if	ok
9	3	$Gdim$	6	if	err
10	4	Em	2	if	ok
11	2	Am	1	if	ok
12	2	Am	8	else if	ok
13	3	Dm	4	else if	ok
14	2	Am	1	else if	ok
15	2	$Ddim$	2	for	err
16	1	G	1	other	ok
17	1	G	3	other	ok
18	0	C	1	other	ok

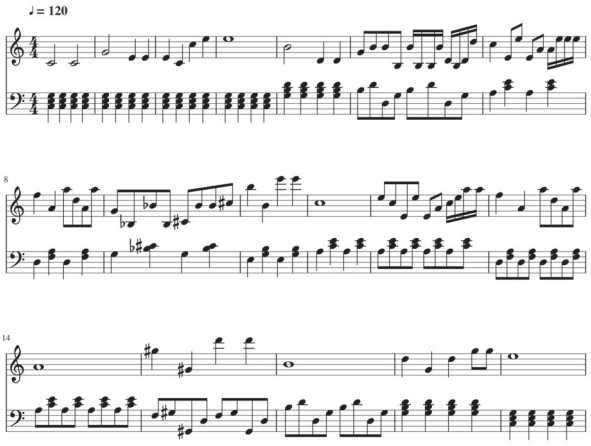

図 4: 表 6 のデータから生成された楽曲

表 6 より, 違反チェックにおいて 9 行目と 15 行目は err となっている.

9 行目は規約 15.6 に違反している. 規約 15.6 とは, 「for 文や if 文, while 文などの本文に中括弧 ({}) を使用しなければならない.」といったものである. 9 行目の if 文には中括弧がないため, 規約違反と判定されている.

また, 15 行目は規約 15.7 に違反している. 規約 15.7 とは, 「全ての if...else if 構文は else 文で終了しなければならない.」といったものである. 12 行目に else if 文があるため, 15 行目には else 文が記述されなければならない. しかし, else 文が記述されることなく if...else if 構文を終了しているため, 規約違反と判定されている.

以上のことから, 9 行目と 15 行目はコードが減三和音 (dim) となっていることがわかる.

9. 評価実験

本システムの有効性を評価するために, 本学の情報工学科の学生 20 名 (C 言語のプログラミングは学習済み) に評価実験を行なった.

本システムによって規約違反箇所の検出及び改修が可能かを検証する. まず被験者は, 不協和音のある楽曲と無い楽曲を聴き比べ, 不協和音の確認をする. 確認後, 実際に本システムを利用し, 規約違反箇所の改修をした. 被験者が修正し終えたと感じたら実験終了とする. 実験終了後, 被験者には図 5 のアンケートに対する回答を紙面に記入して提出してもらった. Q1~Q6 は 4 択の選択式で, Q7~Q9 は記述式で回答してもらった. Q7 は, Q5 で「飽きやすかった」と回答した人にのみ回答してもらった.

実験対象プログラムは 70 行程度のコンパイルエラーのない C 言語プログラムである. その中に, 規約違反箇所が 6 箇所ある. 違反箇所一覧を表 7 に示す. また, 実験時間は無制限で行なってもらった. 実験時間は約 30 分から 1 時間程度であった.

Q1 音楽に自信がありますか?
Q2 不協和音の認識は難しかったですか?
Q3 コーディング規約違反箇所の検出は難しかったですか?
Q4 本システムにより検出は簡単になりましたか?
Q5 楽曲は飽きやすかったですか?
Q6 本システムによるソースコード改修は楽しかったですか?
Q7 飽きやすいと感じた理由をお願いします.
Q8 本システムを使わないと気づかなかった規約違反はどれですか?
Q9 本システムを使ってみた意見, 感想をお願いします.

図 5: アンケート設問

表 7: コーディング規約違反箇所一覧

問題番号	違反内容
(1)	for 文に中括弧がない
(2)	実行されない文がある
(3)	for 文に中括弧がない
(4)	if 文に中括弧がない
(5)	else if 文に中括弧がない
(6)	else 文が存在しない

10. 実験結果

図 5 の Q1~Q6 のアンケート結果を表 8 に示す. また, 問題毎の正答人数と Q8 のアンケート結果から導き出した本システムの利用率を表 9 に示す. 利用率とは, 正答者の中で, 本システムを用いたことによって正解に辿り着くことができた人の割合を表している. さらに, Q7 と Q9 のアンケート結果を図 6 に示す.

表 8: アンケート結果 1/3

	当てはまる	どちらかといえば当てはまる	どちらかといえば当てはまらない	当てはまらない
Q1	2	7	6	5
Q2	4	7	6	3
Q3	3	6	10	1
Q4	8	10	2	0
Q5	1	5	6	8
Q6	11	9	0	0

表 9: アンケート結果 2/3

問題番号	正答人数 (人)	本システムにより気付いた人数 (人)	利用率 (%)
(1)	17	10	58.8
(2)	12	6	50.0
(3)	14	9	64.3
(4)	18	11	61.1
(5)	18	10	55.6
(6)	8	4	50.0

Q7. 飽きやすいと感じた理由をお願いします.

- 4 分音符が続いてたから.
- リズムが単調だから.

Q9. 本システムを使ってみた意見, 感想をお願いします.(一部抜粋)

- 小学生などの子供向けに使えそう.
- 正しく修正されているかの判断が難しかった.
- 修正前後の変化が明確だったため, わかりやすかった.
- 音楽を取り入れることでプログラミング初心者が馴染みやすいと感じた.
- 何もなしでプログラムを修正するよりもわかりやすく飽きずに作業ができた.
- 不協和音がもっとわかりやすい方が良いと感じた.
- 音楽から規約違反箇所を探すことが楽しかった.
- プログラムができる人のためだけでなく, 初心者にも使えそうだと思った.
- 明らかに間違っていない箇所での音楽が不協和音に聴こえ, 自分の回答に不安を感じた.

図 6: アンケート結果 3/3

表 8 より, Q4 は「当てはまる, どちらかといえば当てはまる」と回答した人が大多数であった. これは, 本システムがコーディング規約違反箇所の検出を容易にしていたということがわかる. このことから, 本システムはプログラム改修の一助となっていたといえる. また表 9 より, 全ての問題に対して利用率が 50%以上であった. つまり, 正答者の半数以上は本システムを利用してコーディング違反箇所を検出したということがわかる. これらのことから, 本システムがコーディング規約違反箇所検出の一助となっていたといえる. 以上のことから, 本システムはソースコードの静的解析ツールとして有効であったと考えられる. また, 本システムはソースコードの静的解析結果の可聴化システムである. 本システムの有効性を示したため, ソースコードの静的解析結果は画面に出力する可視化だけでなく, 音に出力する可聴化でも有効であることがわかった.

従来の可聴化システムでは単なる音の羅列による可聴化のため, 飽きやすいといった問題があった. 本システムは従来可聴化システムの飽きやすさを改善できたかどうかを考える. 表 8 より, Q5 は「当てはまる」と回答した人が 1 人, 「どちらかといえば当てはまる」と回答した人が 5 人であった. このことから, 本システムでは未だ不十分であると考えられる. しかし, 大多数は飽きにくいと感じており, 楽曲による可聴化は飽きやすさの改善に効果的であったと考えられる. また図 6 より, 飽きやすいと感じた理由として, 「4 分音符が続いていたから」「リズムが単調だから」があった. これらはメロディのリズムではなく, 伴奏のリズムに対してである. 制御構造に応じて伴奏を決定しているため, 制御文を使用している箇所が少なければ自ずと単調になってしまう. このため, 伴奏の決定方法を改善する必要がある. 以上のことから, 本システムは飽きやすさを完全に改善できたとはいえない. しかし, 楽曲による可聴化は飽きやすさの改善に効果的であったと考えられる. 飽きやすさを感じた人への改善策として, 伴奏の決定方法の変更が考えられる.

本研究では, コーディング規約違反箇所を不協和音で表現した. 表 8 より, Q2 は「当てはまる, どちらかといえば当てはまる」と回答した人が半数以上であった. また図 6 より, 不協和音の認識が難しいといった意見が多数あった. これらのことから, 不協和音は認識が難しく, 規約違反箇所の表現としては不十分であったと考えられる.

表 8 より, Q6 は全ての被験者が「当てはまる, どちらかといえば当てはまる」と回答していた. このことから, 本システムによりソースコードの改修作業が楽しくなったといえる.

11. おわりに

本研究では, 静的解析ツールとして, 静的解析結果可聴化システムを開発した. また, 可聴化システムにおける飽きやすさを改善すべく, 自然な楽曲による可聴化を提案した.

評価の結果, 静的解析結果の可聴化はソースコードの静的解析ツールとして有効であることを示した. また, 飽きやすいといった従来システムの問題は完全に解決できたとはいえない. しかし, 楽曲による可聴化は飽きやすさの改善に効果的であった.

参考文献

[桑原 2015] 桑原 寛明, 渥美 紀寿, “ソースコードの静的検査における警告の版間追跡ツール,” ソフトウェアエンジニアリングシンポジウム 2015, 情報処理学会 (2015).

[P. Vickers 2003] 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).

[佐藤 2011] 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).

[岩田 2013] 岩田 まどか, 隼田 尚彦, 向田 茂, 斉藤 一, 安田 光孝, 大島 直樹, “ソースコードの可視化・可聴化によるメタ認知的学習支援手法の研究–メディアアートの要素を取り入れたアプリケーション開発–,” エンターテインメントコンピューティングシンポジウム 2013, 情報処理学会 (2013).

[渡邉 2018] 渡邉 彰吾, 六沢 一昭, “プログラムからの楽曲生成,” 第 17 回情報科学技術フォーラム (FIT2018), E-027, 電気電子情報通信学会, 情報処理学会 (2018).