Vol.2016-SE-191 No.9
2016/3/14

# 組込みプログラムへのパターン違反検出技術の適用

清水 隆也[1,a)]　吉田 則裕[1,b)]　高田 広章[1,c)]

**概要**：パターン違反とは，コーディングパターンの一部が欠落したコード片を指す．パターン違反は，欠陥修正やリファクタリングの候補になりうるため，パターン違反検出技術が提案されている．本研究では，組込みプログラムにパターン違反検出技術を適用することで，修正候補となるコード片の特定を試みた．

# Applying Pattern Violation Detection to Embedded Programs

Takaya Shimizu[1,a)]　Norihiro Yoshida[1,b)]　Hiroaki Takada[1,c)]

## 1. はじめに

自然言語におけるイディオムと同じく，プログラミング言語においてもコーディングパターンが存在する [1], [2], [3], [4], [5], [6], [7], [8]．例えば，C 言語ではファイルを開くために fopen 関数を呼び出した後，ファイルの読み書きを行う処理を記述し，最後にファイルを閉じるため fclose 関数を呼び出す．

自然言語においてイディオムが誤用されるように，プログラミング言語においてもコーディングパターンの誤用が行われ，コーディングパターンの誤用を含むコード片はパターン違反と呼ばれる [1], [5], [9]．パターン違反は，ソフトウェアの不具合や保守性の低下の原因となる．例えば，fopen() 関数の後に fclose() 関数を呼出し忘れると不具合の原因になりうる．また，パターン違反を含むプログラムは一貫性が欠如しているため，理解しづらく保守性が低いと言える [7].

組込みプログラムには高い信頼性や保守性が求められることが多いため，パターン違反を除去することが強く望まれる．しかし，組込みプログラムではコーディングパターンが多く作りこまれる傾向にある．その理由は，組込みプログラムは，信頼性とともに応答性も求められるため，Java や C#のようなモジュール性の高い言語ではなく C 言語で記述されることが多いからである．コーディングパターンが多く作りこまれるということは，パターン違反が作りこまれる可能性が高まる．これまでに，組込みプログラムからコーディングパターンを検出する手法 [8] や，C 言語で記述されたプログラムを対象としてパターン違反を検出する手法 [1], [3], [5] が提案されてきたが，組込みプログラムからパターン違反を検出した研究は著者らの知る限り存在しない．

本研究では，TOPPERS/ATK2 及びスロットマシン用組込みプログラムを対象にパターン違反の検出を行った．まず，各関数に対して，関数呼出し文，マクロ関数の呼出し文，ラベル，ジャンプ文を構成要素とする順序列を抽出した．次に，抽出した順序列の集合に対して，PrefixSpan アルゴリズム [10] を用いた系列パターンマイニングを行った．最後に，相関ルールマイニング [11] を用いて，パターン違反の検出を行った．

本稿では，まず 2 章においてパターン違反検出技術の概要を述べたあと，3 章において組込みプログラムにパターン違反検出技術を適用した結果について述べる．最後に 4 章において，まとめを述べる．

## 2. パターン違反検出技術

パターン違反とは，プログラム言語において，コーディングパターンの誤用が行われることで発生するコーディングパターンの誤用を含むコード片のことである [1], [5], [9].

[1] 名古屋大学
Nagoya University, Nagoya, Aichi 464–8601, Japan
[a)] shimizu@ertl.jp
[b)] yoshida@ertl.jp
[c)] hiro@ertl.jp

**表 1** 系列パターンの例

| 系列 ID | 系列パターン | サポート値 |
|---|---|---|
| 1 | A → B | 1000 |
| 2 | A → B → C | 999 |
| 3 | A → B → D | 500 |

パターン違反は，ソフトウェアの不具合や保守性の低下の原因となる．

以下で例を交えながらパターン違反検出方法について説明する．まず，パターン違反であることを判断する閾値は 0.90(=90%) とする．閾値 $\alpha$ とは，系列パターンの出現確率が $\alpha$ を超えている時，その系列パターンにパターン違反が存在していることを識別する閾値となっている．例えば，閾値 $\alpha$ が 95%の時に出現確率 98%の系列パターンが存在するとする．その場合，その系列パターンにはパターン違反があると判断する．

図 1 のような系列パターンが，あるトランザクションにおいて存在するとする．ここで関数は左から右へと順序情報を持っているとし，行の末尾の値はサポート値である．つまり，1 の系列はアイテム A の後には，アイテム B が出現する系列がトランザクションに 1000 回出現していることを表している．

同様に 2 の系列は，系列 A → B の後にさらにアイテム C が出現する系列がトランザクションに 999 回出現していることを表す．この時，系列 A → B の後にアイテム C が出現する確率は 1 の系列 1000 回のうち，999 回出現しているため 999/1000(=99.9%) と計算できる．つまり，系列 A → B の後にアイテム C が出現しない系列はパターン違反であると考えられる．

次に，3 の系列についても検証する．3 の系列の場合，系列 A → B の後にアイテム D が出現する確率は 500/1000(=50.0%) となり，パターン違反ではないと考えられる．

以上の方法でパターン違反を検出する．閾値 $\alpha$ の値によって，系列パターンから検出されるパターン違反の数が変わる．閾値 $\alpha$ を高く設定してしまうと，検出されるパターン違反は少なくなる，一方，閾値 $\alpha$ を低く設定してしまうと，パターン違反が大量に検出され，さらにはパターン違反である信頼性の小さなものが検出されてしまう．適切な閾値 $\alpha$ の設定が必要となる．

## 3. 適用結果

組込み 0S である TOPPERS/ATK2 の 8 つのプロダクト及びスロットマシン用組込みプログラムを対象にした．TOPPERS/ATK2 の 8 つのプロダクトに関して，それぞれの機能とプログラムの規模を表 2 に，スロットマシン用プログラムの規模を表 3 に示す．プログラムの規模は関数および関数型マクロ定義数と，抽出された構成要素の種類をその指標とした．

本研究では，各関数に対して，関数呼出し文，マクロ関数の呼出し文，ラベル，ジャンプ文を構成要素とする順序列を抽出した．次に，抽出した順序列の集合に対して，PrefixSpan アルゴリズム [10] を用いた系列パターンマイニングを行った．最後に，相関ルールマイニング [11] を用いて，パターン違反の検出を行った．

### 3.1 TOPPERS/ATK2

パターン違反の閾値 $\alpha$ の値を 0.90 から 0.99 まで 0.01 ごとに変更し，検証を行った．系列パターンの出現確率が 1 となる場合は，プログラム内で常にその系列パターンに従った系列パターンが現れていることを示しているので，パターン違反は起こりえない．従って，今回は閾値 $\alpha$ が 1 のときは検証していない．

各プロダクトのパターン違反検出数を，表 4 にまとめた．閾値 $\alpha$ を高く設定するほど，パターン違反の検出数は減少している．これは閾値 $\alpha$ が高くなれば，その閾値を超える系列の出現確率も同時に高まる．出現確率が高くなることは，結局系列内での違反が少なくなっているということであるため，検出されるパターン違反は減少したと考えられる．

プロダクト SC1-MC や SC3-MC は，パターン違反検出数が比較的大きくなった．これは両プロダクトの規模が他のプロダクトより大きいため当然のことといえる．違反検出のための系列パターンが多く作り込まれることで，パターン違反も作り込まれる可能性が高まっているのである．

また，閾値 $\alpha$ が 0.98 を超えるとパターン違反は検出されなくなった．これは，パターン違反を含む系列のサポート値が小さいためであると考えられる．例えば，ある系列でサポート値が 100 の場合，1 つパターン違反紛れていた時，その系列の出現確率は 99/100(=99%) となる．しかし，サポート値が 20 の場合，パターン違反が 1 つだけしか紛れていないときでも，系列の出現確率は 19/20(=95%) となってしまう．各プロダクトの最大サポート数は表のようになった．いずれのプロダクトもそれほどサポート値が大きいということはなく，またパターン違反が常にサポート値の大きなものから検出されるとは限らないため，閾値 $\alpha$ が大きな場合に検出されなかったと考えられる．従って，パターン違反を検出する際にはプログラムに適した閾値 $\alpha$ を設定して検出を行う必要がある．

図 1 に，TOPPERS/ATK2 から検出されたパターン違反の例を示す．この例では，赤字で示すラベルとジャンプ文の順序が入れ替わっていることで，パターン違反として検出された．

### 3.2 スロットマシン用組込みプログラム

図 2 に，スロットマシンから検出されたパターン違反の

表 2　TOPPERS/ATK2 の概要

| プロダクト | SC1 | SC1-MC | SC1-TP | SC2 |
|---|---|---|---|---|
| 機能 | 基本機能 | 基本機能+マルチコア | 基本機能+時間パーティション | 時間同期拡張 |
| 関数定義 | 129 | 212 | 145 | 145 |
| トランザクション | 1,129 | 2,341 | 1,230 | 1,307 |
| 構成要素の種類 | 199 | 324 | 224 | 228 |
| プロダクト | SC3 | SC3-MC | SC3-TP | SC4 |
| 機能 | メモリ保護拡張 | メモリ保護拡張+マルチコア | メモリ保護拡張+時間パーティション | メモリ保護拡張+時間同期拡張 |
| 関数定義 | 207 | 268 | 218 | 216 |
| トランザクション | 1,901 | 3,208 | 1,984 | 2,056 |
| 構成要素の種類 | 309 | 412 | 325 | 325 |

パターン違反例

```
GetActiveApplicationMode(void)
{
~~~
        appmode = appmodeid;

 exit_finish:
        LOG_GETAAM_LEAVE(appmode);
        return(appmode);

~~~
 exit_no_errorhook:
        appmode = INVALID_APPMODETYPE;
        goto exit_finish;
}
```

パターン適合例

```
GetAlarmBase(AlarmType AlarmID,
AlarmBaseRefType Info)
{
~~~
        Info->mincycle = p_cntcb-
>p_cntinib->mincyc;

 exit_no_errorhook:
        LOG_GETALB_LEAVE(ercd, Info);
        return(ercd);
}
```

図 1　TOPPERS/ATK2 から検出されたパターン違反

表 3　スロットマシンの概要

| プログラム | 5-3_a.c | 5-3_b.c | 5-3_c.c | 5-3_d.c |
|---|---|---|---|---|
| 関数定義 | 9 | 8 | 10 | 10 |
| トランザクション | 99 | 92 | 95 | 99 |
| 構成要素の種類 | 23 | 21 | 22 | 22 |

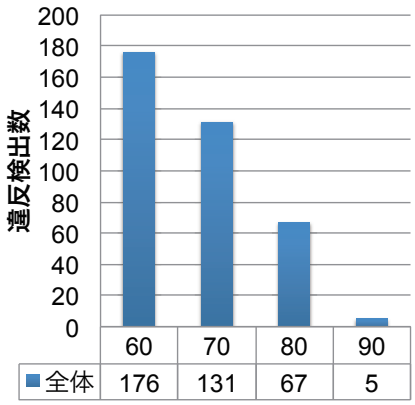


図 2　スロットマシンから検出されたパターン違反の概要

概要を示す．

図 3 の違反のない系列ではループの最後で関数 tslp_tsk() が呼び出されているが，図 4 違反のある系列では呼び出されていないことが分かった．なお関数 tslp_tsk() では，指定の時間をスリープする関数である．この関数によってスロットマシンのボタン押下の際に発生するチャタリングを防止している．ボタンを押下する処理の後には，必ず呼び出されるのが望ましいと考えられる．

4 種類のプログラムは各々開発者が異なっており，記述の方法に差が生じている．他のプログラムでは，ボタンの押下の際のチャタリング防止が実装されていたが，パターン違反の検出されたプログラムの開発者は，プログラムを通してチャタリング防止のための記述がされていなかった．よってプログラムにチャタリング防止のための記述する方が望ましいという提案が出来る．

## 4. まとめ

本研究では，TOPPERS/ATK2 及びスロットマシン用組込みプログラムを対象にパターン違反の検出を行った．まず，各関数に対して，関数呼出し文，マクロ関数の呼出し文，ラベル，ジャンプ文を構成要素とする順序列を抽出した．次に，抽出した順序列の集合に対して，PrefixSpan アルゴリズムを用いた系列パターンマイニングを行った．最後に，相関ルールマイニングを用いて，パターン違反の検出を行った．

TOPPERS/ATK2 およびスロットマシン用組込みプロ

**表 4** 各プロダクトのパターン違反検出数

| プロダクト | 0.90 | 0.91 | 0.92 | 0.93 | 0.94 | 0.95 | 0.96 | 0.97 | 0.98 | 0.99 |
|---|---|---|---|---|---|---|---|---|---|---|
| SC1 | 12 | 12 | 8 | 6 | 4 | 4 | 0 | 0 | 0 | 0 |
| SC1-MC | 115 | 106 | 80 | 67 | 57 | 48 | 21 | 2 | 0 | 0 |
| SC1-TP | 11 | 11 | 6 | 5 | 3 | 3 | 0 | 0 | 0 | 0 |
| SC2 | 19 | 18 | 13 | 11 | 8 | 6 | 4 | 0 | 0 | 0 |
| SC3 | 36 | 33 | 27 | 22 | 16 | 15 | 7 | 1 | 0 | 0 |
| SC3-MC | 95 | 95 | 95 | 87 | 80 | 61 | 51 | 19 | 0 | 0 |
| SC3-TP | 31 | 30 | 26 | 22 | 16 | 15 | 7 | 2 | 0 | 0 |
| SC4 | 36 | 33 | 28 | 23 | 17 | 16 | 8 | 3 | 0 | 0 |

```
void entry4(void)
{
    char LcdStr0[17] = "";
    char LcdStr1[17] = "";
    for (;;) {
        slp_tsk();
        judge4 = ON;
    ~~~~
        textOut(0, LcdStr0);
        textOut(1, LcdStr1);
        tslp_tsk(30);
    }
}
```

**図 3** チャタリング防止実装

```
void Entry4(void)
{
    int i=0;
    char LcdStr1[17];
    char LcdStr2[17];
    for(;;)
    {
    slp_tsk();
    if(flag_b1_c==0)
   ~~~~
    else
    {
        flag_b1_c=0;
        int0ic=INT0_OK;
    }
    /* tslp_tsk() is missing */
    }
}
```

**図 4** チャタリング防止未実装

グラムを対象としてパターン違反を検出したところ，修正の検討を要する関数を特定することができた．

**謝辞** 本研究にご協力くださった名古屋大学工学部電気電子・情報工学科 卒業生の古橋拓人氏に感謝いたします．本研究は JSPS 科研費 26730036 の助成を受けたものです．

## 参考文献

[1] Li, Z. and Zhou, Y.: PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code, *Proc. of ESEC/FSE '13*, pp. 306–315 (2005).
[2] Xie, T. and Pei, J.: MAPO: Mining API Usages from Open Source Repositories, *Proc. of MSR '06*, pp. 54–57 (2006).
[3] Kagdi, H., Collard, M. L. and Maletic, J. I.: Comparing Approaches to Mining Source Code for Call-Usage Patterns, *Proc. of MSR '07*, pp. 123–130 (2007).
[4] Acharya, M., Xie, T., Pei, J. and Xu, J.: Mining API Patterns As Partial Orders from Source Code: From Usage Scenarios to Specifications, *Proc. of ESEC-FSE '07*, pp. 25–34 (2007).
[5] Tan, L., Zhang, X., Ma, X., Xiong, W. and Zhou, Y.: AutoISES: Automatically Inferring Security Specifications and Detecting Violations, *Proc. of USENIX Security '08*, pp. 379–394 (2008).
[6] 石尾 隆，伊達浩典，三宅達也，井上克郎：シーケンシャルパターンマイニングを用いたコーディングパターン抽出，情報処理学会論文誌，Vol. 50, No. 2, pp. 860–871 (2009).
[7] Nguyen, T. T., Nguyen, H. A., Pham, N. H., Al-Kofahi, J. M. and Nguyen, T. N.: Graph-based Mining of Multiple Object Usage Patterns, *Proc. of ESEC/FSE '09*, pp. 383–392 (2009).
[8] 中村勇太，崔 恩瀞，吉田則裕，春名修介，井上克郎：パターンマイニング技術を用いた C 言語プログラムからのコーディングパターン抽出，電子情報通信学会技術研究報告，Vol. 115, No. 20, pp. 41–46 (2015).
[9] Wasylkowski, A., Zeller, A. and Lindig, C.: Detecting Object Usage Anomalies, *Proc. of ESEC-FSE '07*, pp. 35–44 (2007).
[10] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U. and Hsu, M.-C.: PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth, *Proc. ICSE 2001*, pp. 215–224 (2001).
[11] Agrawal, R. and Srikant, R.: Fast Algorithms for Mining Association Rules in Large Databases, *Proc. of VLDB '94*, pp. 487–499 (1994).