C-024

組込みソフトウェア向けコーディング規約チェッカのためのカスタマイズの一方式
A method of customize for embedded software coding rule checker

福原 和哉† 高橋 耶真人†† 猪股 俊光††† 新井 義和††† 今井 信太郎†††
Kazuya Fukuhara† Yamato Takahashi†† Yoshimitsu Inomata††† Yoshikazu Arai††† Shintarou Imai†††

1. はじめに

品質を確保するための既存の手法であるコードレビューはソフトウェア開発工程で見過ごされた誤りを検出・修正するためにソースコードの体系的な検査を行う作業であり，定式化されていない不具合が発生する箇所の発見や，技術者の経験や勘に基づく改善点などの発見によるソフトウェア品質の改善に有効である[1]. しかし，一方では担当者がコードレビューにより一定時間拘束されるため，作業の遅延が発生するなどの問題があるほか，不具合の定式化が困難な不具合については検出作業を自動化ができないという問題点もある[1]. また，現在広く使われている自動検査ツール[2] [3]はコードレビューよりも不具合発見の生産性が高いが，あらかじめ用意された分析モデルと検査項目のみに対応し，前述の技術者の経験や勘などを根拠としているような定式化ができないような不具合の検査には適用が困難であるという問題点がある.

このような問題点を解決するため，我々はコーディング規約チェッカのルールや分析モデルを拡張するための一手法を考案し，ANSI-C 言語で記述された組み込みソフトウェア向けの検査ツールとして実装し評価を行った.

その結果，提案手法では不具合検査を検査パターンとしての記述・検出が実際に可能であるという結果が得られた.

1.1 概要

本研究の提案するカスタマイズ方式の概念図を図 1 に示す．この本式は検査ルールの柔軟なカスタマイズ方式に対応させるために以下のような特徴を持つ.

- 解析器と検査器を分離
- コード解析器はソースコードから曖昧さを排した中間表現を生成
- モデル生成器は中間表現から必要とするモデルを生成
- 検査器はモデルと検査記述を読み込み検査を実行
- 検査記述は検査器毎に固有のものを与えることが可能

2. モデル

単一のモデルで様々な検査を行うことは困難であると同時にモデルの複雑化を招くため，モデル生成器が生成するモデルは検査の手法ごとに独立したモデルとする．また，コード解析器と検査器を分割することにより既存のモデルを利用した新しい検査の追加が容易となる.

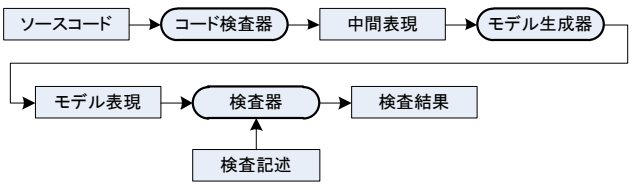

図 1　カスタマイズ方式の概念図

本研究ではコーディング規約の検査を対象として，モデルとして構文木に着目した構文木モデルを用いることとした．Switch 文をソースコードとしたときの構文木モデルの例を図 2 に示す．構文木モデルはコンパイル可能なソースコードから得られた木構造で，参照の解決，省略されている構文要素の補完等を行い，曖昧さを排除したものである.

3. 検査記述

検査すべき内容の記述（以下，検査記述）はモデル表現に依存する．たとえば，オートマトンを用いる検査[4]では，存在の否定などの記述が困難であるほかバックトラックが多発するため検査時間が長引く場合がある．他方 XPath 技術を用いる検査[5][6]ではツール本体を拡張しないかぎり記述できる検査は単純なもののみである.

本研究の構文木モデルを用いた検査手法では構文木の節の集合と親子・兄弟との位置関係を元にした検査記述である．これは検査を行う人間の思考方法に近いものであり，存在の否定などの記述も可能とである．また，記述の柔軟性と性能を実現するため検査記述として”検査マクロ”と，高度な検査の記述に対応できる”検査スクリプト”を用意した.

ソースコードの中の「default 文を持たない switch 文」の出現の有無を検査する例を考える．これはオートマトンを用いる検査ツールでは記述が困難な不具合であるが，考案した方式では以下のように簡単な集合演算で記述できる.

S：ソースコード
A：S中の switch 文を示す節の集合
B：S中の default 文を示す節の集合
$F(x)$：集合x内の要素それぞれから最も近い親要素である switch 文の集合を求める関数
C：S中の default 文を持つ Switch 文を示す節の集合
D：S中の default 文を持たない Switch 文を示す節の集合

$$C = F(B),\ D = A - C$$

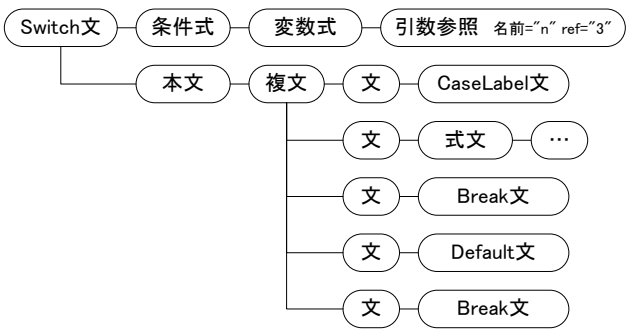

図 2　構文木モデルの例

```
VAR A = NODE "//Switch 文" FROM S
VAR B = NODE "//Default 文" FROM S
VAR C = First-Ancestor-Of  "switch 文"  FROM B
VAR D = A - C
```

図 3　検査マクロでの記述例

```
var A = S.Descendants("//Switch 文")
var B = S.Descendants("//Default 文")
var C = B.FirstAncestors((x) => x.Name == "Switch 文");
var D = A - C
```

図 4　検査スクリプトでの記述例

```
var funcref =
  "式/関数呼び出し式/呼び出し先/式/変数式/関数参照";
var call_lock = root.Descendants("式文").Where((x) =>
  x.Element(funcref).Attribute("名前").Value == "lock"
).Reverse();
var call_unlock = root.Descendants("式文").Where((x) =>
  x.Element(funcref).Attribute("名前").Value == "unlock"
);
IDictionary<XElement, XElement> unlock_dic =
  new Dictionary<XElement, XElement>();
call_lock.FirstSiblingWithSelf((x, y) => {
  var s = y.Element(funcref).Attribute("名前").Value;
  if ((s == "unlock") &&
      (unlock_dic.ContainsKey(y) == false)) {
        unlock_dic.Add(y, x);
        return true;
  } else {
    return false;
  }
}).Evaluation();
var unpair_unlock = call_unlock.Except(unlock_dic.Keys);
var unpair_lock = call_lock.Except(unlock_dic.Values);
return unpair_lock.Concat(unpair_unlock);
```

図 5　lock と unlock の対応検査記述

3.1 検査マクロ

検査マクロは検査記述で用いる集合演算や構文要素の選択などを容易に記述するために設計されたマクロ言語で制御構文等は持たない．検査マクロ記述は後述の検査スクリプトに翻訳され実行される．先程の例を検査マクロで記述したものを図 4 に示す．

3.2 検査スクリプト

検査マクロは記述の容易さを求めた反面，制御構文やデータコンテナ型などを持たない．そのため，複雑な検査を記述することができない．検査スクリプトは記述が困難な検査や，複雑な検査を行うためのより直接的な検査記述方式で C#言語を用いて記述する．先程の例を検査スクリプトで記述したものを図 5 に示す．

既存の検査ツールの一部[5][6]はソースコード自体に手を入れることによって検査の追加が可能である．しかし，そのためにはツール自体の再コンパイルが必要となるとともに，ツールを拡張するための知識も必要となるため検査の追加は容易ではない．本研究では.NET Framework の持つ CSharpCodeProvider 機能を用いた実現によってツール本体を再コンパイルすることなく検査ルールの追加が可能となり，実行速度の向上にもつながった．

表 1　MISRA-C:2004 での記述能力の評価

全ルール件数	141 件
コーディング規約件数	96 件
本手法で記述可能な規約	65 件
スクリプトによる記述	41 件
マクロによる記述	24 件

表 2　MISRA-C:2004 ルールの検査結果

検査ステップ数	32 ファイル/10231 行
実行時間	185 秒
挿入ヶ所の発見率	100%
既存の不具合の発見件数	3 件

4. 評価

4.1 MISRA -C:2004 を用いた評価

組込みソフトウェア向けコーディング規約である MISRA-C:2004 を記述対象とした規約検査の記述能力の評価を行った結果を表 1 に，既存の組込みソフトウェアソースコードに規約に対応する違反例を手作業で挿入したものを用い検出能力の評価を表 2 に示す．その結果，本手法では MISRA-C:2004 のルールの約 46%を記述・検出することが可能であることが確認できた．

4.2 検査記述能力の評価

組込みソフトウェアなどではよく用いられる，例えば「割り込み禁止と解除(lock と unlock)が同一ブロック内で一対一対応しているどうか」を検査するには括弧の入れ子の深さや出現位置を考慮に入れなければならない．しかし，これらは正規表現などを元にしたツールでは記述することができず，多くのコードレビューツールでは容易に記述することができないものであった．

本手法では検査スクリプトを用いて図 5 のように記述することが可能である．

5. まとめ

コーディング規約チェッカのルールや分析モデルを拡張するための一手法を提案し，ANSI-C 言語で記述された組み込みソフトウェア向けの検査ツールとして実装し評価を行った．その結果，本手法を用いることで自動検査器へのルールの追加とそれによる検査が可能であることを確認できた．今後はルールの拡充とともにフロー解析モデルなどを追加して検査可能範囲の拡大を図りたい．

参考文献

[1] Boris Beizer, "Software Testing Techniques", Van Nostrand Reinhold, 1990, 978-0442206727
[2] 東洋テクニカ，静的解析ツール QAC, http://www.toyo.co.jp/ss/qac/, 2012/07/02
[3] 富士通ソフトウェアテクノロジーズ，PGRelief, http://jp.fujitsu.com/group/fst/services/pgr/, 2012/07/02
[4] S. Paul and A. Prakash, "A Framework for Source Code Search Using Program Patterns," Software Engineering, vol. 20, pp. 463-475, 1994.
[5] InfoEther, PMD, http://pmd.sourceforge.net/pmd-5.0.0/, 2012/07/02
[6] 大須賀, 小林, 間瀬, 渥美, 山本, 鈴村, 阿草: CXChecker: C 言語プログラムのためのカスタマイズ可能なコーディングチェッカ, ソフトウェアエンジニアリング最前線 2009, 近代科学社, pp. 119-126, (2009).

†岩手県立大学いわてものづくり・ソフトウェア融合テクノロジーセンター　Iwate Monodukuri and Software Integration Technology Center
††岩手県立大学大学院ソフトウェア情報学研究科　Graduate School of Software and Information Science in Iwate Prefectural University
†††岩手県立大学ソフトウェア情報学部ソフトウェア情報学科
Faculty of Software and Information Science in Iwate Prefectural University