Panasonic Technical Journal Vol. 56 No. 3 Oct. 2010

高効率開発を実現する自動ソフトウェア検査技法と文書トレーサビリティ管理技法

Automatic Software Checking Technique and Document Traceability Management Technique for Efficient Software Development

青　島　武　伸*
Takenobu Aoshima

本稿では，ソフトウェアの誤りを自動的に検出する技術としてメモリーリーク自動検出技術，またソフトウェア開発の関連文書を管理する技法として文書トレーサビリティ管理技法について述べる。メモリーリーク検出技術では，境界モデル検査技術を利用し大規模なソフトウェアに対する高精度の検査を実現する。文書トレーサビリティ管理技法では，開発過程で変更されていく要求や，それに対応する設計，実装などの各要素についての文書変更作業，および文書同士のトレーサビリティの維持作業の効率化を実現する。

In this paper, a memory leak detection technique and a document traceability management technique are introduced. The memory leak detection utilizes a bounded model checking technique to check large-scale software with high accuracy. The document traceability management technique realizes an efficient environment to maintain documents when some elements of documents such as requirements, the corresponding design, and implementation are changed in the software development process.

1.　メモリーリーク自動検出技術

家電ソフトウェアの開発規模，および複雑さが増大するなか，開発効率に重大な影響を及ぼす課題として，メモリーリークに代表されるリソース管理の不具合の修正がある。

メモリーリークは，プログラム実行中，特定の処理を行うために確保したメモリーを，解放しないことによって起こる。これによって，使用しているメモリーが増加していった場合，正常な動作の続行が困難になり，たとえばプログラムが停止し，操作不能の状態に陥るなど，重大な品質問題が引き起こされる。

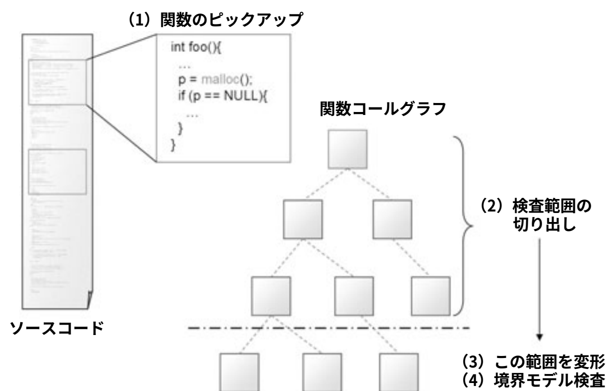
また，メモリーリークの原因箇所の発見は困難な場合が多い。これは，リーク発生直後において，システムは正常な動作を続けるためである。数度のメモリーリークが発生した直後では，まだメモリーに余裕がある場合が多く，直ちに不具合を引き起こすことは少ない。このため，リークの原因箇所と不具合発見時の実行箇所が異なることが多く，テストによる発見が困難になる。本稿で述べるメモリーリーク検出技術では，テストによる検査にかえて，ソースコードからメモリーリークが起こりうる形を抽出することで，網羅性の高い検査を実現する。

1.1　検出手法

メモリーリーク検出は，下記手順で行う。ここで，入力はC言語のソースコードとする：

＊ パナソニックR&Dセンターベトナム
パナソニックハノイ研究所
Panasonic Hanoi Lab., Panasonic R&D Center Vietnam

(1) メモリー確保関数を呼び出す関数のピックアップ
(2) 1の関数それぞれを起点とし，一定の深さの関数呼び出しまでの関数を，検査の対象範囲として切り出す
(3) 検査の対象範囲のソースコードをリーク検出のための形式に変形し，境界モデル検査器[1]に入力
(4) 境界モデル検査器の出力を解析し，結果を出力



第1図　メモリーリーク検出手順

Fig. 1 Procedure for memory leak detection

(3) の変形では引数の値が偽であるときエラーを出力するassert文を用い，メモリーリークが発生する形であればassert文の引数が偽になるようにコードを変形する（**第2図**）。関数addref，removerefはそれぞれ変数が格納されているアドレスを引数にとり，メモリーの確保および解放を記録，また関数の出口に正しくメモリーが解放されていれば真を返す関数checkrefをassertと共に置く。また，メモリーを補完するコードなどコードを部分的に切り取ることにより発生する不必要なエラーを避ける変形を別途加える。

```
int func(){
    char* p;
    p = malloc(sizeof(char));
    if(somefunc() == ERR){
        return -1;
    }
    doSome(p);
    free(p);
    return 0;
}
```

```
int func(){
    char* p;
    p = malloc(sizeof(char));
    addref(&p);
    if(somefunc() == ERR){
        assert(checkref());
        return -1;
    }
    doSome(p);
    free(p);
    removeref(&p);
    assert(checkref());
    return 0;
}
```

第2図　境界モデル検査のためのコード変形

Fig. 2 Modification for bounded model checking

上記変形の後，境界モデル検査器に入力する。境界モデル検査器は，**第3図**のように論理式への変換を行い，論理式 $C \wedge \neg P$ を満たす変数の真偽の値が存在するか否かを判定する。論理式を満たす変数の真偽値とは，たとえば式が $(x \vee y) \wedge \neg z$ の場合，$(x, y, z) = (1, 0, 0)$ などの式全体の値を真とする変数値の組合せである。**第2図**で行ったリークが起こる場合に真となる条件を付加したコードを論理式に変換し，これを真にする値の組合せが存在するということは，すなわちリーク条件を真とする値の組合せが存在することを意味し，これによりリークする可能性の有無を判定できる。

```
if(x != 4){
    x+=y+2;
    y=x-2;
} else {
    x++;
}
assert(x<=y);
```

```
C:=(x1=x0+y0+2) ∧
    (y1=x1-2) ∧
    (x2=x0+1) ∧
    (x3=(x0!=4)?x1:x2) ∧
    (y2=(x0!=4)?y1:y0)

P:=(x3<=y2)
```

第3図　コードから論理式への変換

Fig. 3 Conversion from code to logical formulas

ここで境界モデル検査を用いる利点は，手順(2)のように一定深さで条件の考慮を打ち切った場合，論理式の制約は純粋に減るのみであるため，リークを見逃す可能性が増大しない点にある。たとえば，論理式 $f \wedge g \wedge j$ から f を真とする制約を取り除いた場合，式は $g \wedge j$ になる。このとき，式全体が偽になる可能性は f を取り除いたぶん減る。すなわち，リークと判定されない可能性が減るのみである。この性質を利用することにより，検出精度を損ねることなく検査を分割し，大規模なソースコードの検査を可能にしている。

（注）Microsoft および Excel は，米国Microsoft Corporation の米国およびその他の国における登録商標

1.2　従来技術との比較

従来のソースコード検査技法では，主にソースコードの振る舞いを計算によって算出し検査を行う，抽象実行と呼ばれる方法が用いられる。この方法では，さまざまな値の組合せの可能性を考慮しながら条件分岐の組合せをカバーしていくため，通常のテスト実行による検査に比べ，より多くの条件の組合せを検査することができる。しかし，プログラムの前から順を追って計算していく必要があり，1つの関数に多量の分岐がある場合や，アセンブラによる記述などサポートしていない形がある場合に，その計算が中断され，検査を中止してしまう問題がある。

これに比べ，本技法では論理式にすべて変換しているため，プログラムの上から順に計算をしていく必要はなく，リークが起こる条件から逆算していく形で不具合が起こるかどうかを検査することができる。また，サポートしていない形がある場合でも，計算結果を大きく損ねることなく，その部分の条件を無視することができる。これらにより実用的な検査を実現している。

第1表に市販のツールとの比較データを示す。本技術を利用したツールが検出精度で上回り，特に大規模なプロジェクトに有効であることを示している。

第1表　オープンソース4プロジェクトでの適用結果

Table 1 Benchmark on 4 open source projects

評価プロジェクト	コード行数	関数の数	リーク検出数	
			市販ツール	本技術
A	1 289 574	66 457	検査不能	10
B	826 928	23 339	検査不能	12
C	26 155	446	7	7
D	3 741	42	2	3

2.　文書トレーサビリティ管理技法

ソフトウェア開発に関する文書には要求，設計，テスト，課題管理などさまざまな種類があり，各種の文書の要素が別の種類の文書の幾つかの要素と関連するという構造をもつ。これら要素同士の関連の追跡可能性をトレーサビリティと呼ぶ。トレーサビリティを保持すれば，たとえば下記に挙げる，プロジェクト管理において重要な事項を容易に実践することができ，ソフトウェア開発を効率よく管理することができる。

- 要求などに変更があった場合の影響範囲の特定
- 適切な文書の構成管理
- テスト，レビューなどのカバー範囲の特定

しかし，従来のMicrosoft® (注) Wordや，Microsoft Excel® (注)を用いた文書作成手法では，変更が多発するソフトウェア開発において，トレーサビリティを管理する表の維持

が容易ではない。これは，ときに数百から数千項目に及ぶ要求やその関連項目の数が膨大になること，およびバグ管理や課題管理など，目的に応じて多種多様なツールを用い，その情報が分散することが主な原因である。

また，たとえば要求の名称に変更があった場合，この変更に対応するすべての箇所を適切に変更する必要があるが，複数の文書，および箇所に記述が及ぶ場合が多く，文書全体の整合性を維持することは容易ではない。これは，スケジュール管理，タスク管理，要求管理および分析，アーキテクチャ設計，詳細設計，テスト管理などの文書を，それぞれの目的に応じて，別々に記述していくことが原因として挙げられる。すなわち開発プロジェクトを，さまざまな断面で切り取って表現するため，重複した記述が文書群の複数箇所に出現する。

2.1 管理技法

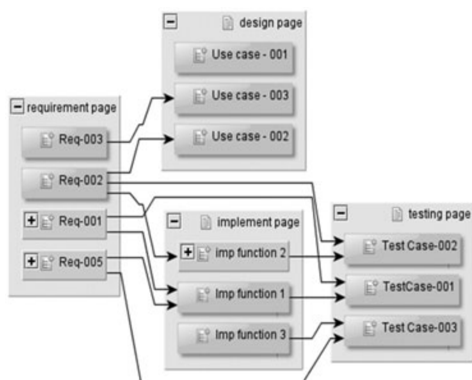
そこで本技法では，要求項目，設計項目，テスト項目などをそれぞれ一要素ごとに分けて記述する環境，またそれらの関連を記述する環境を用意する。タスク管理や要求管理などの目的別の文書は，これら要素の情報を基にし，ツールにより生成することで，同一の情報の重複記述を避け，また文書同士のトレーサビリティを容易に確保する環境を実現する。

2.2 管理技法の実装

文書トレーサビリティ管理ツールは，下記構成からなる。

(1) 要求，設計など要素ごとの編集機能
(2) 要求文書などテンプレート作成機能
(3) 要素同士の関連の可視化
(4) 外部ツールが生成する情報の取り込み

要素同士の関連の可視化では，**第4図**に示すように要素の関連を矢印によって図示する。ここでは，たとえば，Req-003のIDをもつ要求に関連づけられているテスト項目がないため，この要求に対するテストがないことを読み取ることができる。

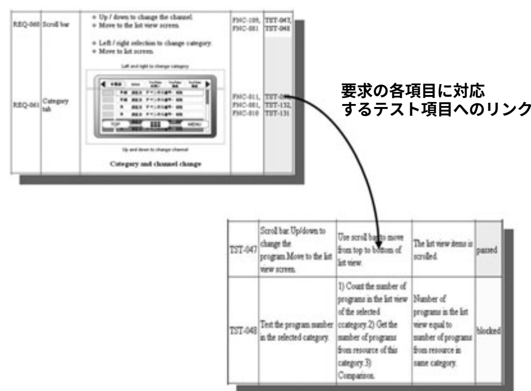


第4図 関連性の可視化

Fig. 4 Visualization for relation between elements

(4) の外部ツールの要素の取り込みでは，たとえばソースコード成形ツールが生成するソースコード上のコメント，メトリクス測定ツールが生成するメトリクス情報，テスト管理ツールが生成するテスト管理情報を取得，文書内の要求など関連要素との関連づけを行う。情報の取得には，各ツールが提供するXML（eXtended Markup Language）出力を主に利用している。

これらの結果得られる文書を，**第5図**に示す。HTML（HyperText Markup Language）で関連要素が互いにリンクされ，関連文書の参照を容易にするほか，情報共有において各種ツールを各自がセットアップする必要がなくなる。



第5図 出力の文書

Fig. 5 Output document of system

3. 今後の展望

メモリーリーク自動検出技術は，現在，C++言語へ対応するための実装，およびメモリーリーク以外の欠陥検出にも対応するための実装を進めており，より広範囲の品質向上を実現する。トレーサビリティ管理技法については，Microsoft WordやMicrosoft Excel，PDFなど広く一般に使われるフォーマットの文書への関連づけを実装し，利用範囲の拡大を実現していく予定である。

参考文献

1) A. Biere, et al. : Symbolic model checking without BDDs. LNCS1579, pp.193-207 (1999).