様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

令和　元 年　 ６ 月　 ７ 日現在

機関番号：１２６１２
研究種目：基盤研究(C)（一般）
研究期間：2015～2018
課題番号：１５Ｋ００９１

研究課題名（和文）デバッグ手法とツールの評価のためのベンチマーク環境の構築

研究課題名（英文）Benchmark set of buggy programs for the evaluation of debug techniques and tools

研究代表者
寺田　実（Terada, Minoru）

電気通信大学・大学院情報理工学研究科・准教授

研究者番号：８０１６３９２１

交付決定額（研究期間全体）：（直接経費）　　1,900,000円

研究成果の概要（和文）：デバッグ手法やツールの評価を目的として、バグを含むプログラムからなるベンチマークセットの構築を目指して研究を行った。
正しいプログラムにランダムに変異を加え、それらをテストセットの結果で選別する手法を採用し、ソーティング、リスト処理、再帰的なプログラムを対象として実験を行い、ほとんど正しい、しかも意外性のあるバグを含むプログラムを自動生成することができた。こうして作成したセットはデバッグ教育にも適用可能だと考える。

研究成果の学術的意義や社会的意義
提案した手法により、手続き型のプログラムを対象として、アプリケーションの特性や環境に依存しないバグ入りプログラムのセットを自動生成することができた。今回は「正答率の高さ」を選択尺度としたが、尺度の設定により多様なバグが生成できる。
これらは当初の目的であるデバッグ手法の評価に有用であるばかりでなく、プログラミング教育におけるデバッグ技術の習得、プログラム理解支援、視覚化の題材としても利用可能である。

研究成果の概要（英文）：The research aims at the creation of a benchmark set of "buggy" programs to evaluate various debug techniques and tools.
I apply random mutations to a correct program to get a code, which is filtered by the score for test inputs.I tried this method for various programs, including sorting, list processing and recursive program. The resulting codes behave correctly for most inputs but still include subtle bugs. The set may be usable for education purposes.

研究分野：情報工学

キーワード：デバッグ

様　式　Ｃ－１９、Ｆ－１９－１、Ｚ－１９、ＣＫ－１９（共通）

１．研究開始当初の背景

デバッグはソフトウェア作成において避けて通れない重要な技術である．プログラムの証明や自動生成といった先進的な研究も行われているが，適用可能範囲はまだ広いとは言えない．デバッグ技術も進歩しつつあり，プリント文の挿入やステップ実行などの古くからの技術もある一方，最近では計算機の向上した処理能力を活用した新しい方式の提案も盛んに行われている．たとえば，誤動作を発生させる入力を差分的に特定する技術(デルタデバッギング) や，プログラムの状態を保存して再実行や逆実行を可能にするものなどがある．また，オープンソースソフトウェアを対象としてバグ修正のパターンを分類した研究も行われている．

筆者もこれまで，Java プログラムにおける逆実行の実現とその応用としての reverse watchpoint の提案，実行トレースをもちいたプログラム理解システム，GUI プログラムの再現によるデバッギングなど，デバッグに関連する研究を行ってきた．

このようにデバッグ手法やツールについて多くの提案がなされているが，これまでそれらを統一的に評価する方法がなかった．それぞれの論文では研究ごとに独自のサンプルセッションを例示し有効性を主張しており，他のエラーへの有効性が判断できないし，相互比較も困難である．筆者にとっても，過去の研究において有効性の主張に困難を感じてきた．

２．研究の目的

本研究では上記の問題に対処するために，典型的なエラーを含むプログラムのセット(ベンチマークプログラム) を用意する．それらをサンプルとしてデバッグセッションを実行して操作系列を得て，その結果により手法やツールを評価しようというものである．

解決しようとする問題は，以下のように整理できる:

(A) ベンチマークとして適切なプログラムとはどのような条件を満たすものか．

(B) 手法やツールへのベンチマークの適用はどのように行うべきか．

(C) 得られたベンチマーク結果から，手法やツールをどう評価するのが適切か．

(A) については，プログラミング入門レベルのエラーから，マルチスレッドプログラミングに見られるような再現性のないエラーにいたるまで，カバーすべき範囲は多岐にわたる．また，対象とするプログラミング言語についても，強い型づけをもった「安全な言語」から，C 言語のような自由度の高い「危険な言語」まで幅がある．本研究では，安全な言語による初等的なデバッグをスタートラインとして，より現実的なものへ対象を広げていこうとする．

(B) については，デバッグ特有の問題点がある．まず，デバッグは人間が主体となって行う過程であって，デバッガやツールはその補助にすぎない．つまりベンチマークの実行には人間が必要となる．さらに，本来デバッグは現象をてがかりに未知のエラーを発見する過程であるから，与えられた既知のエラーを含むベンチマークプログラムでは正当な評価が難しい．つまり，「このエラーが未知であったとしたらこのような操作を行うであろう」といった，一種の作業者モデルが必要になると考えている．

(C) については，最終的には人間による知的作業のサポートが目的であるため，そのようなベンチマークと作業者モデルが構成できるか，またそれによって種々のデバッグ手法やツールが正しく評価できるかを解明することが目標となる．

本研究の意義はふたつにわけられる.ひとつは，新しいデバッグ手法やツールの開発においては，標準的なベンチマークを利用することで有用性の主張に基盤をあたえ，当該分野の研究発展に貢献できる．また，既存デバッグ手法の特徴を明らかにすることによって，適切な手法の選択が可能になる．

つぎに，作成したベンチマークプログラムを教育分野で利用することが可能である．プログラミング教育の分野では，自分の作ったプログラムがなぜ動かないのかを理解させるデバッグの経験は重要である．こうした状況に対して，ベンチマークプログラムを一種の練習課題として利用することが可能で，ソフトウェア技術者の技能向上に役立てることができる．また，ソフトウェア開発においてデバッグ側からみたエラーの事例集として利用も可能である．

３．研究の方法

(1) バグ収集手法の検討

バグ入りプログラムを揃えるにはいろいろな方法がある．

ひとつは実際のコードから収集する方法である．現実性を持つコードを確保できる点でのぞましいが，いろいろ問題点がある

- 収集が容易ではない

個人の集められる件数はそう多くはなく，多くの人が集めたバグを共有する仕組みが必要となる．バグトラッキングシステムの利用が一つの解決策であるが，次項で述べる問題点もある．

- バグが単離できていない

現実のソフトウェアは大規模であるので，個々のバグには大きな関連情報が付随しているのが普通である．そうしたバグを学習やベンチマークで利用するには自己完結したバグとして単離（いわゆる Short，Self Contained，Correct Example）しておく必要がある．

二番目は経験をもとに作成する方法である．よくあるバグを再現するコードを人手で作成してセットとする．たとえば，繰り返しの構造において回数を一回だけ多く/少なく行ってしまうバグ（いわゆる Off-by-one error）などを念頭に置いている．学習用に使うのであれば，こう

したセットを標準の問題集として利用するのはよいと思う．ただ，実際に直面するバグのうちにはそうしたイディオムでは解決の難しいものが存在し，そうしたものは人手で作るのは難しい．意外性が必要なのである．

三番目の方法が自動生成である．上記の「意外性のあるバグ」を作る方法として，本研究ではこれに着目した．

(2) 自動生成の概要

本研究では，正解のプログラムを一つ用意し，それに変異を加えることでバグを作り出す．変異はランダムに行い，多数の試行を繰り返すことで望ましいバグをバリエーションを持って生成する．

この手法は遺伝的プログラミング（Genetic Programming）と類似している．そこでは，単純で正しくないコードを出発点とし，変異を繰り返すことで仕様を満たすコードを生成しようとする．またソフトウェアテストの文脈では，正しいプログラムに変異を加えてバグを導入し，その変異のうちのどれだけをテストが検出できるかによってテストセットの完成度を評価する手法として，Mutation Testing と呼ばれている．これは本研究とアプローチに共通点が多いが，本研究ではバグ入りプログラムの生成に目標を置いている点で異なっている．ソフトウェアテストにおけるもう一つの手法として Fuzzing がある．これは誤りを含むデータを生成してプログラムに与えて動作を観察するもので，近年ソフトウェアの脆弱性検出などに利用されてきている．

(3) バグ生成の手順

本研究では以下の手順でバグ入りプログラムを生成する．まず，正解プログラムを用意する．つぎに，テストデータセットを用意する．そのあと，変異と実行の繰り返しを行う．

① 抽象構文木(AST) 生成と変異

正解プログラムを読み込み，AST をメモリ上に生成する．その AST を深さ優先でトラバースし，変異の候補となる節点に出会うたびに乱数を用いて一定確率で変異を加える．（したがって，プログラム全体での変異の発生数は一定しない．）変異候補節点は以下の三種とした．

- 変数参照

その時点での名前表を検索し，ローカル変数であったなら一定確率で型の一致する別のローカル変数と置き換える．

- 二項演算

演算子が比較演算子，条件演算子，算術演算子のいずれかのグループに属していたら，一定確率で同一グループの別のものと置き換える．

- ブロック

0 個以上の文の並びである．文が二つ以上あった場合，以下のいずれかを一定確率で行う．

-- 文の削除．並びのうちからランダムに一つの文を選び削除する
-- 文の複製．並びのうちからランダムに一つの文を選び，ランダムな位置に複製を挿入する
-- 文の交換．並びのうちからランダムにふたつの文を選び交換する

変異によって複製や移動のあった文についても再帰的に変異処理を加えていく．

変異は確率的であるため，試行の繰り返しの結果として同一のコードが生成されてしまう可能性がある．これを防止するために生成したコードのハッシュを保存しておき，重複を発見した場合にはそのあとの実行段階は行わない．

② テストセット実行

変異を加えたプログラムをファイルに書き出し，コンパイルしてテストデータを入力として実行する．出力を正解と比較して正解数を算出し記録する．あとからコードを吟味する必要があるがコードそのものを保存するのはデータ量に問題があるため，変異を与えるもととなった乱数のシードもあわせて記録することでコードの再現を可能としている．コンパイルの際にエラーになったもの，実行時に例外発生で異常終了したもの，規定時間内に実行が終了しなかったものは記録を残しておく．

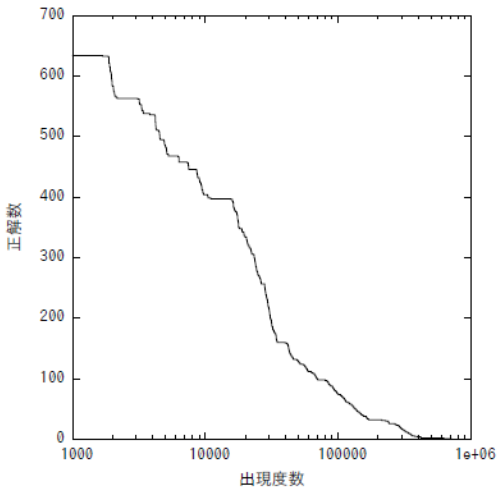
(4) 対象プログラム

ソーティング、リスト操作、N-Queens などとした．

４．研究成果

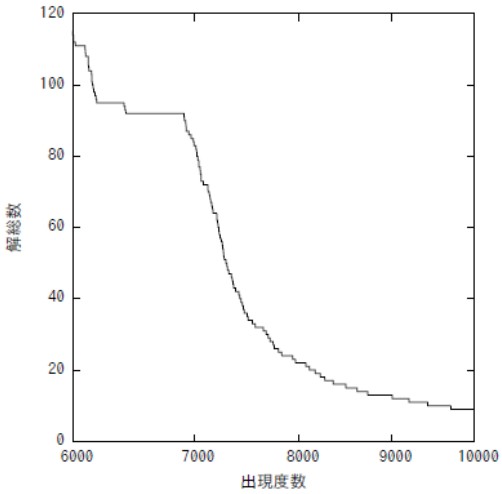
(1)定量的結果

変異を加えたプログラム群がテストにどの程度正解できるかを測定した．これは「よいバグ」つまりほとんど正答するプログラムが本当に生成できることを確認するためのものである．ヒープソートを例として、長さ 5 までの網羅的なテストデータ(総数 634 通り)に対しての正解数とその度数をプロットしたものを右図に示す．これをみるとある割合で「よいバグ」が生成できていることがわかる．実際、たった一つのテストだけに誤答するものを 175 種生成できている．



この傾向は異なった問題に対しても同様にあら

われる. 右図は 8-Queens の解総数を出力する問題（正解は 92）に対して得られた解総数のプロットである．正解の周辺にもプログラムが存在していることがわかる．



(2)生成したバグの内容
生成した多数のバグプログラムの一部を目視により解析分類してみた．その結果いくつかのパターンが見られた．
- 惜しいコード
わずかの変更によりごく一部の入力に対して誤りを発生するもので、本研究で目指した「よいバグ」の一つである．
- 変異を含んでいても全テスト正解
プログラムの動作がことなってしまっても正しく動くもので、場合によってはプログラムの最適化に利用できる可能性もある．
- 人間の行いやすい誤り
配列の添字や比較演算子の向きなどが考えられたが、網羅的なテストによる評価で選別したため、そうした誤りは極めて低いスコアとなってしまう．テスト正解数以外の選別尺度の必要性を示唆している．
- 人間の理解を越えたコード
非常に惜しい結果を出すにもかかわらず、人間がそれをうまく説明できないものも見られた．これは人間のプログラム理解のありように深く関連するものといえよう．
(3)まとめ
　正しいコードに確率的な変異を加えテストケース通過数という尺度でフィルタリングすることによって，たとえばデバッグの練習問題として使えるような「よいバグ入りプログラム」の作成を試みた．コードの自然さ，動的な振る舞いの類似など，異なる尺度によるフィルタリングも併用することでより有用な結果が作れるであろう．

５．主な発表論文等

〔学会発表〕（計８件）

① 寺田 実: "ランダムな変異を用いたバグ入りプログラムの生成", 情報処理学会 第 60 回プログラミング・シンポジウム報告集, pp. 103-110 (2019).

② 村松 啓寛, 寺田 実: "コード編集と Web 閲覧履歴の時系列による統合とその活用" (poster), 情報処理学会インタラクション 2019, 2B-19 (2019).

③ 佐々木 透, 寺田 実: "データベース学習のための関係代数アニメーション" (poster), 情報処理学会インタラクション 2019, 2B-42 (2019).

④ 三谷 将大, 寺田 実: "Web アプリケーションによるゲーミフィケーションを用いたプログラミング上達支援システム" (poster), 第 26 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(WISS2018), 3-A04 (2018).

⑤ 藤本 明優, 寺田 実: "並列プログラム理解支援のための細粒度プログラムアニメーション", 情報処理学会 夏のプログラミング・シンポジウム 2018 (2018).

⑥ 渡邊 裕貴, 寺田 実: "ソフトウェア依存関係ネットワークにおけるベイズ的コミュニティ抽出" (poster), 第 20 回情報論的学習理論ワークショップ(IBIS2017), D2-28 (2017).

⑦ 阿部 真之, 寺田 実: "相関ルールを利用したソースコードの識別子推薦手法", 情報処理学会第 58 回プログラミング・シンポジウム報告集, pp. 51-58 (2017).

⑧ 下山 真明, 寺田 実, 丸山 一貴: "変数の変更履歴を用いたデバッグ支援", 情報処理学会第 78 回全国大会講演論文集 7H-05 (2016)

６．研究組織

※科研費による研究は、研究者の自覚と責任において実施するものです。そのため、研究の実施や研究成果の公表等に

ついては、国の要請等に基づくものではなく、その研究成果に関する見解や責任は、研究者個人に帰属されます。