7H-05

変数の変更履歴を用いたデバッグ支援

下山　真明*　　寺田　実‡　　丸山　一貴†

明星大学　　情報学部*†　　電気通信大学　　情報理工学部‡

概要

C 言語でデバッグ困難なものに，バッファオーバーフローによるメモリ破壊のバグがある．通常はデバッガのウォッチポイントを用いるが，ループ制御変数のように正当な代入が多い場合には破壊を発見するのは困難である．メモリ破壊のバグは規則的な変化を壊しているので、変数の変更履歴を記録して可視化することで、破壊的な代入を捕捉することができる．

本論文では Intel CPU と Linux の上での C プログラムを対象として，変更履歴の記録に GDB の record 機能を利用する．変更履歴はグラフとして表示し，グラフ上の点を選択することで該当する代入までデバッギの実行を遡らせる．本論文では実装の詳細と典型的な適用事例について述べる．

1　背景と目的

作成したプログラムにはバグは必ずあり，熟練したプログラマであれば様々な手法を用いてデバッグするが、そうでない場合うまくバグを発見できず，多くの時間を無駄にしてしまう．見つけにくいバグとして例えば，バッファオーバーフローによるメモリ破壊のバグでは、その破壊された変数に対しウォッチポイントを用いるが，ループ制御変数のように正当な代入が多い場合には破壊を発見するのは困難であると考える．

本研究ではデバッギの変数変更履歴を記録し記録からグラフを作成することで，破壊的な代入を捕捉し，デバッギの状態を再現することでデバッグを支援すること目的とする．

Visualization of history of variables helps programmers to locate bugs

*Masaaki Shimoyama, School of Information Science, Meisei University

‡Minoru Terada, Faculty of Informatics and Engineering, The University of Electro-Communications

†Kazutaka Maruyama, School of Information Science, Meisei University

この目的を実現するために record/replay 機能を持つデバッガである GDB を用いる．GDB とは GNU ソフトウェアシステムで動く標準のデバッガで，基本的なデバッガの機能のほかに逆実行のための保存機能も備えている．

2　先行研究

柏村ら[1]は実行トレースシステム(ETV)と開発環境(Eclipce)を組み合わせて，お互いの弱点を補ったプログラミング支援を提案している．プログラムの記録を観察することを特徴に述べているが，本研究では記録する部分を任意で決定できるためファイルの大きさを減らすことができると考えている．

Fabian ら[2]は変数の変更履歴をウォッチポイントの機能を利用し記録する．その記録から変数をグラフ化する方法がある．本研究は GDB を用いることでウォッチポイントだけではデバッグの難しい再現性のないプログラム(入出力，マルチスレッド)にも対応できると考えている．

Cheng ら[3]はプログラムスライシングを用いて成功したケースと失敗したケースの差分からブレークポイントを自動的に設定する．本研究では実行制御の目標地点を変数の値の時系列グラフから求めている．

3　提案手法

今回提案する手法についてのシステムの流れについては以下の通りである．

①ユーザーはデバッギをコンパイルし既存の GDB を用いて実行し，変数変更履歴を記録する．システムは記録情報をファイルに書き出す．

②システムは手に入れた記録を元にプログラムを用いて値の変化をグラフとして可視化する．ユーザーは可視化されたグラフの点を選択し状態再現に必要な命令のアドレス(プログラムカウンタ)と書き換わりの回数を入手する．

③ユーザーは手に入れた情報とファイルから再度 GDB を用いて状態再現を行う．システムの構成図は図１で示す．

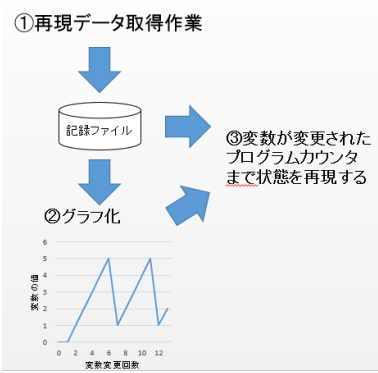

図 1 システムの構成図

4 実装方法

デバッギのメモリ空間の全変更履歴をファイルに保存する，GDB の save 機能を変更しグラフ化用のデータ(変数の内容とプログラムカウンタ)を,追加で出力するようにした．このためにソースコード 2860 行を読み 14 行を追加した．

記録したデータはプログラムで処理し，グラフ化する．グラフの横軸を変更回数，縦軸を変数の内容とし，それぞれの座標には変数を書き換えた命令のアドレス(プログラムカウンタ)の情報を紐づける．

save により保存した情報を restore して，グラフ経由で指定された状態まで，ブレークポイントを利用して移動する．ブレークポイントに，該当アドレスが履歴に出現する回数だけヒットしたら，指定の状態に到達したと判断する．

5 利用例

簡単なサンプルプログラムとしてループ文の制御変数を対象にグラフ化を行う．

例えば配列を初期化しようと考えた場合配列の大きさを格納する size の値が a[i]の配列のサイズを超えた場合配列以外の変数に 1 を代入してしまう．もしそうなれば無限ループが起こるのである程度の地点で実行を停止しグラフ化する．

実験用のソースコードを図 2 に示す．グラフ化したものの一部を図 3 に示す．

```
int a[10], i=0, j=1;
while(i<=size){
    a[i]=j;
    i++;
  }
  return 0;
}
```

図 2 実験用サンプルコード

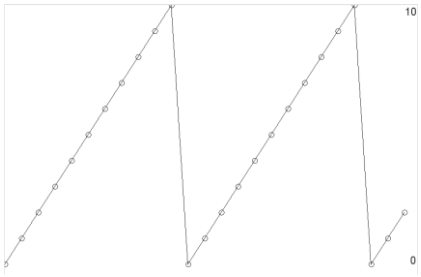

図 3 制御変数のグラフ

このグラフで値が増加していたのに突然減少した箇所が想定されていない代入と考えられるため，ここを選択し，状態を再現する．

6 考察

現状のグラフの提示方法では数値型の変数を対象にしているがどのようなプログラムがデバッグできるのかまた数値型の変数以外のデバッグ支援方法をどう提示するのか検討する必要がある．例えば配列であれば，また配列の要素の添え字をグラフ化することで，定常的なアクセスか否かを判断できる可能性がある他，配列のサイズとそのアドレスの増減を利用すれば書き換わりの方向やアクセスする場所を表現することができると考えている。

7 まとめと課題

本研究では GDB を用いて変数の変更履歴を記録しグラフ化することで，プログラムのデバッグを提案した．

今後の課題としてローカル変数のように，異なる変数や関数呼び出しが同じアドレスに割り当てられる場合の処理が必要である．これは履歴に含まれる call 命令等からコールスタックをシミュレートすることで可能と考えられる．

参考文献

[1] The GNU Project Debugger, <http://www.gnu.org/software/gdb/>, [accessed January 6th,2015].

[2] 柏村 俊太朗,寺田 実:実行トレース視覚化システムの Eclipse Plugin としての実装,第48回プログラミング・シンポジウム vol.48, pp. 213-216,2007.

[3] Fabian Beck, Fabrice Hollerich, Stephan Diehl and Daniel Weiskopf: Visual Monitoring of Numeric Variables Embedded in Source Code,IEEE VISSOFT,pp.1-4,2013.

[4] Cheng Zhang , Juyuan Yang, Dacong Yan, Shengqian Yang: Automated Breakpoint Generation for Debugging, JOURNAL OF SOFTWARE, VOL. 8, NO. 3, pp .603-616, 2013.