3L-8

プログラム実行過程を表す有向グラフを直接編集するデバッギングシステムの研究

日吉俊勝† 太田剛‡

静岡大学大学院情報学研究科† 静岡大学情報学部‡

１．研究の背景

デバッグを行うときには関連する三つの要素を考える必要がある．それはプログラマが期待する動作のイメージ，そのイメージを元に実際にプログラマが書いたプログラムソース，そのソースをコンパイル・実行した結果である実際のプログラムの動作である．このとき，プログラマが期待した動作と，実際のプログラムの動作とでは当初は異なっているのが普通である．

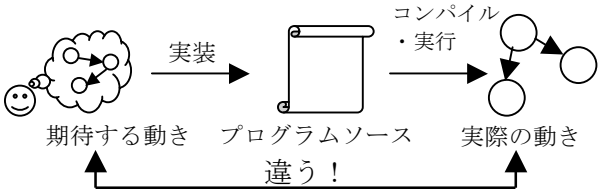

図１　デバッグの三要素

デバッグを行ううえでは，これらの要素の対応を取り，違う箇所を見つける必要がある．まず，期待した動きと実際の動きを正確に知り，理想と実際の動作間の違いを調べる（プログラムの動的構造と動的構造の理解・比較）．次に期待した動きになるようにソースを書き換える（動的構造から静的構造への変換）．このデバッグのステップにはそれぞれ難しさがある．プログラマには，プログラムが「こう動くはずだ」という思い込みがあり，そもそもプログラムの動くイメージを正確にプログラムテキストに変換できないことがバグを生む原因である．また，プログラムがどう動いたかを正確に知るのが難しい．さらに，プログラムの実際の動きのどこが期待と違うのかわかっても，ソース上でその違いがどこに，どのように対応し，どう書き換えるべきか理解するのも難しい．特にこうした作業は，プログラム初心者にとっては難しいことである．

既存のツールを検討する．フローチャート等を自動生成・編集するプログラム図示化ツールでは，ソースコードを抽象化したプログラムの静的構造について扱うだけであり，プログラムの実際の動作を扱うことはできない．シンボリックデバッガはプログラムの実際の動きを理解するのには役立つが，プログラム修正のためにはその動きの理解を元にソースを直接書き換える必要がある．統合開発環境では単純な時間軸上でのプログラムの動きを示すことができる．しかし，実際に人間がプログラムを考える上ではこうした単純な時間系列の動きだけではなく，データ間の依存関係が重要な役割を担っていると思われるため，不十分である．このように既存の開発・デバッグツールでは，動作やソースという要素間の結びつけが不足している．

そこで本稿で扱うデバッギングシステムでは，実際の実行動作を表わす有向グラフを画面上に表示し，これをユーザが GUI 上で編集することで，期待する動作をシステムに教え，その差分からもとのソースを自動変更することを目指す．

２．デバッギングシステム概要

システムの構成と動作の流れを述べる（図２）．まずインタープリタ部に C 言語によるデバッグ対象のプログラムソースを投入し実行する．実行履歴取得部で実行情報をもとに実行履歴を作成する．グラフ作成部では実行履歴により有向グラフデータを生成する．グラフ表示操作部はユーザに対してグラフを表示し，グラフを編集する操作を提供する．つまりグラフ表示操作部はユーザインターフェースを提供する．プログラム変換部では，ユーザによるグラフの編集をもとにプログラムソースを修正する．なお，グラフ生成部およびグラフ表示操作部では，グラフの取り扱いのためにオープンソースグラフツールセットの GraphViz[1]を用いる．

このシステムにおいてグラフはプログラム実行時の変数値の依存関係を表す．ノードはソース中の式を表わす．エッジは変数に値が設定されたノードから，値を参照したノードへと引か

A study on a debugging system which allows user a direct editing of dynamic program behavior

† Toshikatsu Hiyoshi・Graduate School of Informatics, Shizuoka University

‡ Tuyoshi Ohta・Faculty of Informatics, Shizuoka University

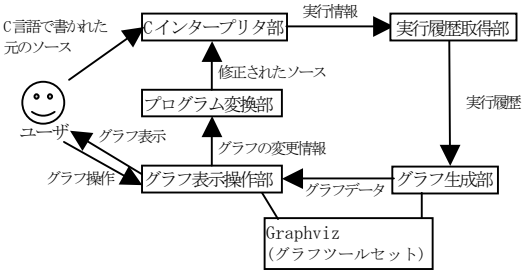

図２　システム構成図

れ，変数の依存関係を表す．反復や選択などのブロック構造は部分グラフとして表現する．表示上はブロックを枠線で囲む．反復の各実行過程はまとめずそのまま回数分が生成されるため，閉路ができず，エッジを順序関係とみなすと半順序集合となる．図３にグラフの例を示す．

```
1:#include <stdio.h>
2:
3:int main(void) {
4: int a = 0;
5: int b = a + 1;
6: int c = a + b;
7: return 0;
8:}
```

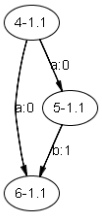

図３　グラフの例

３．プログラムソース変換部について

グラフ表示編集部において，グラフの修正は，ノードの追加・削除，ノードに対応する式の内容の修正，エッジの追加・削除，エッジのラベル（影響した変数）の変更，ブロックの追加，ブロックの種類の変更を行うことができる．

以上のグラフの修正をもとに，プログラムソース変換部ではソースを元のソースから変更する．変換部での処理の中心は式の並び替え，付随的なものは式の内容の変更・追加である．

式の並び替え処理はノードとエッジの追加・削除により元のグラフから変化した順序関係をもとにソースを修正する．すなわち，ノード間の順序関係通りにテキストを適切な位置へ移動する．基本的にはソースの式をグラフに基づきトポロジカルソートすることに相当する．ただし，ユーザが変換後のソースを理解するのを助けるために，できるだけ元のソースからの変更を少なくしたい．そのため，グラフのルートから幅優先探索を行い，ノードの順序関係と対応する式の位置関係を比べ，それが食い違っていたら式を入れ替える．この処理を行う前にユーザの編集によりグラフに閉路が作られていないかチェックする．

単純なエッジの順序関係のみにより式の位置の決定するのは問題が生じることがある．その例を図４に示す．

```
（略）
4: int x = 0;
5: int y = 0;
6: x = x + 1;
7: y = y + 1;
（略）
```

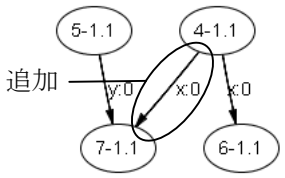

図４　単純な順序関係では問題が生じる例

この例では４行目で変数 x に値が設定され，エッジ追加により 7 行目でその値を参照しようとしている．ここで単純にエッジの依存関係のみを考えると式の順序を入れ替える必要がない．しかし，６行目で変数 x の値が更新されてしまうため，７行目の式では４行目で設定された値を読み出すことはできない．このような場合，間にあるノードも含めて式の順序を入れ替えるか，順序関係が決まってしまっている際には問題の変数 x の値をコピーする変数を追加する処置が必要になる．したがって，ノードの順序関係だけでなく，エッジのラベルも変換処理時には必要な情報となる．また，反復等のブロックでは条件式以外の内部での式の順序を変更することはできるが，ブロック内外をまたぐ順序の変更には問題が起こる場合がある．例えば，あるブロック内で変数 y を書き込んだあと，ブロック外で変数 y を読み込んだ上で更新し，さらに同じブロック内で更新された変数 y を読み込む，といったことはできない．

４．今後の課題

表示部ではグラフ表示レイアウトアルゴリズムに現在は Graphviz 組み込みの汎用のものを用いているが，このシステムのグラフに適した専用のアルゴリズムの考案・実装が今後の課題である．また，特定の変数に関連する部分のみ表示するなどの支援機能の実装も課題である．

グラフ編集部・変換部については while などといった反復構造に対して，反復回数の変更による条件式の変更や，ある実行回のみの変更に対して条件分岐などを追加することにより対応するといった機能が求められる．

参考文献

[1]Emden R. Gansner, Stephen C. North, An open graph visualization system and its applications to software engineering, SOFTWARE-PRACTICE AND EXPERIENCE, vol.30, pp.1203-1233, (Sept. 2000).