Vol.2014
2015/2/10

ソースコード変更時の影響範囲の可視化

晴澤陽太[†1]　猪股俊光[†1]
新井義和[†1]　今井信太郎[†1]

ソフトウェアの開発工程などではソースコードの一部に変更を加える場面が多々あるが，その際に変更を加えていない箇所にも影響を及ぼす可能性が高い．ソースコードの変更による影響範囲を正確に知ることができれば，変更後のソースコードへのテスト範囲を限定することができ，工数の削減に大きな効果をもたらす．そこで，本研究では変数の変更が影響を与える範囲を可視化する静的解析ツールの実装，評価を行った．

Visualizations of Impact Range when Changing the Source Code

Yota Haresawa,[†1] Toshimitu Inomata,[†1]
Yoshikazu Arai[†1] and Shintaro Imai[†1]

In the software development process, there is a lot of opportunity to make changes to the part of the source code. However, some changes, is often influence place not changes. If an impact range can be analyzed correctly, to be able to limit the range to be tested, leads to the reduction of man-hours. Therefore, by this reseach, we were the implementation and evaluation of the static analysis tool what visualize the impact range in which change of a variable in the source code.

1. はじめに

ソフトウェアの開発工程や保守工程などでは，ソフトウェアの改良・修正のためにソースコードの一部に変更を加える場面が多々存在する．しかし，その変更が変更を加えていない部分にも影響を与える可能性は大きく，その影響はソースコード全体に波及する可能性もある．そのため，変更後のソフトウェアの品質を保証するためには，ソースコード全体を対象としたテスト[1)]が必要となる．しかし，ソースコードを変更した影響がどの範囲まで及ぶのかを正確に知ることができれば，テストを行う必要のある範囲を限定することができ，工数の削減に大きな効果をもたらす．そのため，本研究室では変更の影響範囲を解析するための表現法について研究を進めている[2)3)]．

本研究ではソースコード全体に影響を与える可能性が高いと考えられる構文要素の 1 つである変数に着目し，変数の変更が影響を与える範囲を可視化する静的解析ツール（影響範囲可視化ツール）を実装した．また，そのツールの出力する情報が実際に影響範囲の特定に有効であるかどうかを評価した．

†1 岩手県立大学 ソフトウェア情報学部
Faculty of Software and Information Science, Iwate Prefectural University

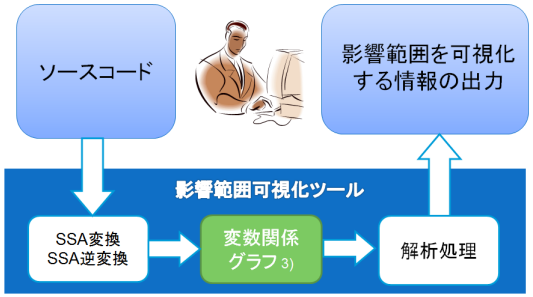


図 1　影響範囲可視化ツールの処理過程
Fig. 1　Process of impact range visualization tool.

2. 影響範囲可視化ツール

2.1 概　　要

本研究で開発した影響範囲可視化ツールの処理過程を，**図 1** に示す．本ツールは，ソースコードを入力とし，入力されたソースコードに対して SSA 変換・SSA 逆変換を行った後，変数どうしの関係を表す変数関係グラフ[3)] に変換する．変更の対象とする変数が選択されると，変数関係グラフに対して解析を行い，その影響範囲に関する情報を可視化して出力する．

2.2 対象とするソースコード

対象とするソースコードは**図 2**(a) のように C 言語で書かれていて，解析する単位は 1 つ

```
f() {
  a = 1;
  b = 2;
  if (a < b) {
    b = a;
  }
  a = b;
}
```

(a) 入力対象のソースコード

```
f() {
  a_1 = 1;
  b_1 = 2;
  if (a_1 < b_1) {
    b_2 = a_1;
  }
  b_3 = φ(b_1, b_2);
  a_1 = b_3;
}
```

(b) SSA 変換後のソースコード

```
f() {
  a_1 = 1;
  b_1 = 2;
  if (!(a_1 < b_1)) goto L1;
    b_2 = a_1;
    b_3 = b_2;
    goto L2;
  L1:
    b_3 = b_1;
  L2:
    a_2 = b_3;
}
```

(c) SSA 逆変換後のソースコード

図 2 SSA 変換と SSA 逆変換の例
Fig. 2 Example of SSA translation and SSA reverse translation.

の関数とする．また，次の構文は対象外とする．

- 配列，構造体，共用体，ポインタなどのデータ構造
- 関数呼出し

ソースコードは SSA[4] 変換，SSA 逆変換される．ソースコードに SSA 変換と SSA 逆変換を行うと同じ変数でも異なる値が代入されている場合には異なる変数名に変換される．よって変数の値を変更した際にその影響がどこまで及ぶのかを分析することが容易になる．また，while 文や for 文などの制御構造は，if 文，goto 文，label だけを用いた形に書き換えられる．図 2(a) のソースコードに対して，SSA 変換，SSA 逆変換をした結果を同図 (b) と (c) にそれぞれ示す．

2.3 変数関係グラフ

2.3.1 形式的定義

変数関係グラフはソースコード中の変数間の関係を表現したグラフであり，影響範囲可視化ツールが出力する情報の 1 つである．変数関係グラフの形式的定義を以下に示す．

プログラム P に含まれる変数，演算子，式の集合をそれぞれ V，O，E とする．

$P = (V, O, E)$

$V =$ 変数の集合 $\cup$ 定数の集合 $\cup$ 仮引数の集合

$O = O_a \cup O_r \cup O_o$

O_a：算術演算子の集合 $\{(+, -, *, /)\}$

O_r：関係演算子の集合 $\{(==, >, \geq, <, \leq, <>)\}$

O_o：単項演算子の集合 $\{(!, -)\}$

$E = \{$ `@if`, `@=`, `@return` $\}$

P における，ノードの集合，アークの集合，ラベル関数をそれぞれ N，A，L とする．

$N = V \cup O \cup E$

$A = (V \times O) \cup (O \times V) \cup (O \times O) \cup (O \times E) \cup$
$(E \times O) \cup (V \times E)$

$L = A \rightarrow \{\mathtt{T}, \mathtt{F}, \mathtt{ref}, \mathtt{null}\}$

以上のことからプログラム P の変数関係グラフ G_p を次式とする．

$G_p = (N, A, L)$

2.3.2 変数関係グラフの構築手順

影響範囲可視化ツールは，SSA 変換，SSA 逆変換後のソースコードの変数どうしの関係を元に隣接行列を作成する．さらに，この隣接行列を元に，グラフ描画ソフトである Graphviz[5] を用いて変数関係グラフを出力する．

Graphviz は，dot 言語[6] のスクリプトで示されたグラフを描画する．よって，まず隣接行列から dot 言語のスクリプトを作成し，それを Graphviz に入力することで描画されたグラフを，ツール上で表示する．図 2(c) に示したソースコードから作成した dot 言語のスクリプトを，付録 A.1 に示す．

2.3.3 変数関係グラフの出力例

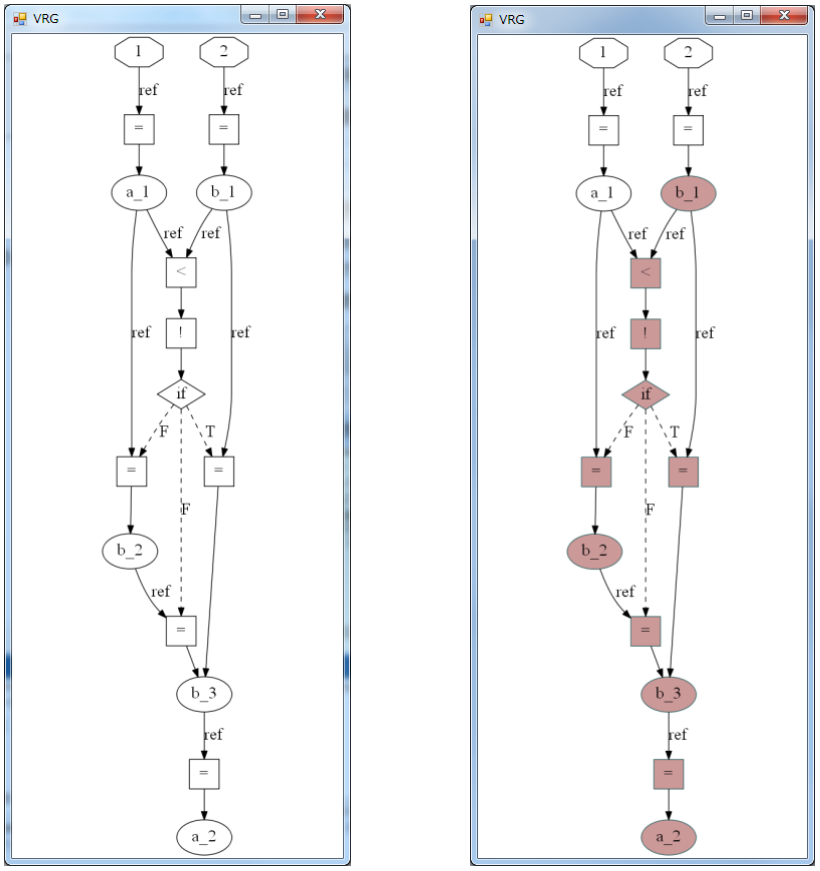
影響範囲可視化ツールの出力する変数関係グラフでは，**表 1** のようにノードの種類別にその形状を定めた．図 2(c) に示したソースコードを入力としたとき，出力された変数関係グラフを**図 3** に示す．また，調査対象とする変数を選択することで，その変数ノードから辿ることのできるノードを着色表示した変数関係グラフを出力する．図 3(b) は，変数 `b_1` を調査対象の変数とした場合の変数関係グラフである．

2.4 影響範囲の可視化情報

影響範囲可視化ツールでは，変数の変更による影響範囲に関する次の情報が出力される．

- 調査対象の影響を受けている変数の箇所と個数
- 調査対象の影響範囲の行番号
- 関数外への影響波及の有無
- 変数関係グラフ

ここで，影響を受けている変数とは，変更対象の変数の値の変化に応じて値が変化する可



(a) 変数関係グラフ　　(b) 変数選択時の変数関係グラフ

図 3　変数関係グラフの出力例
Fig. 3　Sample outputs of variable relationship graph.

表 1　変数関係グラフのノードの形状
Table 1　Form of nodes of variable relationship graph.

ノード		形　状
V	変数・仮引数	楕円形
	定数	八角形
O	演算子	四角形
E	if (条件式)	菱形
	= (代入式)	四角形
	return 文	六角形

能性がある変数を指し，影響の受け方によって，直接的な影響を受ける変数と間接的な影響を受ける変数の 2 種類に分類される．直接的な影響を受ける変数は，変更対象の変数が代入されている変数を指す．間接的な影響を受ける変数は，変更対象の変数が条件式に含まれていて，その条件式の真偽によって値が変化する可能性がある変数のことである．

例えば，図 2(c) のソースコードで変数 `b_1` を変更対象とした場合，直接的な影響を受ける変数は，`b_1` を値とする `b_3` と，その `b_3` が代入される `a_2` がである．間接的な影響を受ける変数は，`b_1` が含まれた条件式の真偽に依存して値が定まる，`b_2` と `b_3` である．図 3(b) の変数関係グラフを用いると，`a_1` からの路が存在する `b_3` と `a_2` が直接的な影響を受ける変数である．また，if(条件式) ノードからの路が存在する `b_2` と `b_3` が間接的な影響を受ける変数である．

2.5　GUI

GUI を**図 4** のような画面レイアウトを実装した．左側のウィンドウにソースコードを入力し，解析開始ボタンをクリックするとソースコードの SSA 変換・SSA 逆変換と，変数どうしの関係を表す変数関係グラフ[3] への変換が行われる．その後ソースコード上の変数をダブルクリックすることで変更対象とする変数を選択することができ，その影響範囲を可視化した情報が出力される．また，グラフ出力ボタンをクリックすると変数関係グラフが出力される．

3.　評　　価

3.1　評 価 方 法

本研究で実装したツールが出力する情報について，変数の影響範囲を正確に特定することができているかという点に着目した評価を行った．評価方法は，**表 2** に示す 8 個のテストプログラムに対して，ツールに入力することで得られた情報と，動的検査で得られた実際の

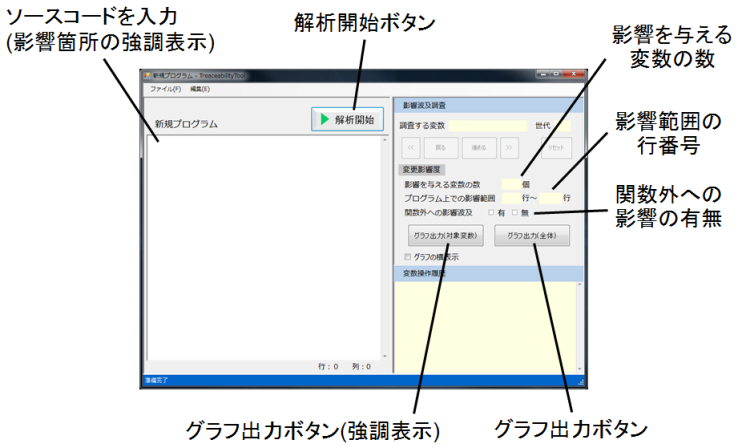


図 4　影響範囲可視化ツールの GUI
Fig. 4　GUI of impact range visualization tool.

影響箇所を比較することで，ツールが正しい情報を出力できているかどうかを評価した．

表 2　テストプログラム一覧
Table 2　Test program list.

プログラム名	テスト内容
directImpact.c	代入による直接的な影響
equal.c	== を含む条件式による間接的な影響
greaterThan.c	> を含む条件式による間接的な影響
greaterThanOrEqual.c	>= を含む条件式による間接的な影響
lessThan.c	< を含む条件式による間接的な影響
lessThanOrEqual.c	<= を含む条件式による間接的な影響
notEqual.c	! = を含む条件式による間接的な影響
return.c	return 文による関数外への影響の波及

動的検査による影響箇所の解析方法は，デバッガを用いてソースコードをトレースしながら代入文での値を検査し，次に変更対象とする変数の値に変更を加えた上で，再度代入文での値を検査する．変数の変更前と変更後で値に変化があった箇所を影響箇所とする．

3.2　評 価 結 果

テストプログラムを greaterThan.c，変更対象を変数 a_1 とした場合の，ツールの出力を**図 5** と**図 6** に示す．

a_1 の変更により影響を受ける変数は，図 5 より a_2，b_2，a_3 であり，影響範囲は 2 行目から 14 行目であることがわかる．また，関数外への影響波及はない．

表 3 は動的検査の結果であり，変更対象 a_1 の変更前と変更後で影響を受ける変数は，a_2 と b_2，a_3 の 3 つである．その影響範囲は変更を加えた 2 行目から，最後の影響箇所の 14 行目であることがわかる．また，この関数では retun 文がないため，関数外への影響の波及はない．

ツールの出力と，動的な調査結果から得られた情報を比較すると，同様の影響範囲を示していることが確かめられた．

表 2 の各テストプログラムに対して同様の作業を行った結果，ツールの出力と，動的検査により取得した影響箇所の情報に相違はなく，ツールの出力する情報は正確に影響範囲を可視化できた．

また，動的検査と比較し，開発したツールはソースコード上での影響箇所の強調表示や，変数関係グラフの出力などによる影響範囲を可視化している．影響箇所の強調表示は，影響を与える変数と影響を受ける変数の関係を直感的に読み取ることを支援し，変数関係グラフは，ノードを辿ることによって影響を追跡することを容易にする．

4.　お わ り に

本研究では，ソースコードの一部を変更した場合に，テストが必要となる範囲を特定すべく，変数を変更した場合の影響範囲の可視化を試みた．その結果，対象とするソースコードについては，影響範囲の可視化によりテストが必要な範囲を特定することができることを確認した．

今後の課題は，現在未対応としている C 言語の構文への対応と，可視化の範囲を関数間の影響にも拡大していくことである．

参　考　文　献

1) 高橋寿一：知識ゼロから学ぶ ソフトウェアテスト，翔泳社（2005）
2) 高橋耶真人，福原和哉，猪股俊光，新井義和，今井信太郎：プログラムの関数・変数関係の一表現法，電子情報通信学会技術研究報告，ソフトウェアサイエンス 113(269),

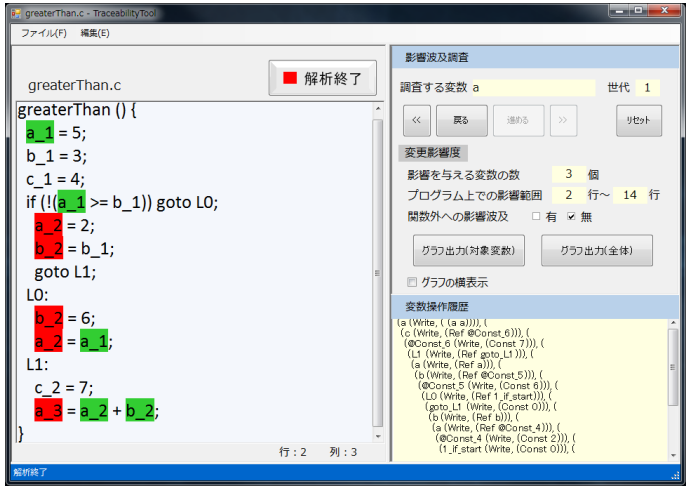


図 5 greaterThan.c 入力時のツールの出力結果
Fig. 5 Tools output result of when enter the greaterThan.c program

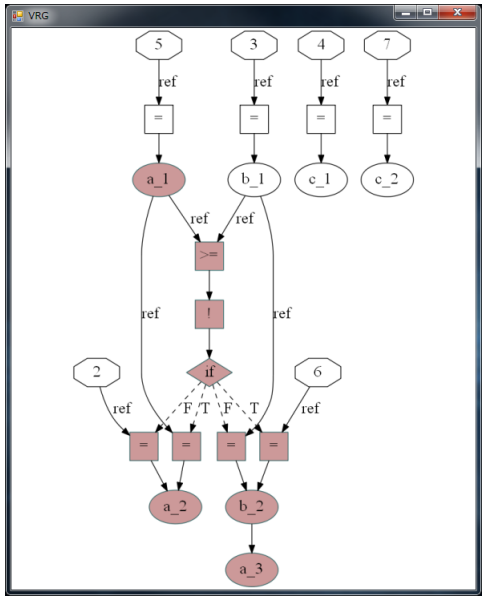


図 6 greaterThan.c 入力時の変数関係グラフ
Fig. 6 variable relationship graph of when enter the greaterThan.c program

表 3 greaterThan.c の動的な調査結果
Table 3 Dynamic survey result of the greaterThan.c program

行番号	変更前	変更後	影響の有無
2	a_1 = 5	a_1 = 1	○
3	b_1 = 3	b_1 = 3	
4	c_1 = 4	c_1 = 4	
5			
6	a_2 = 2		○
7	b_2 = 3		○
8			
9			
10		b_2 = 6	○
11		a_2 = 1	○
12			
13	c_2 = 7	c_2 = 7	
14	a_3 = 8	a_3 = 7	○

49-54(2013)
3) 大久保建男，猪股俊光，新井義和，今井信太郎：ソースコード変更時の影響波及解析のための一表現法，岩手県立大学ソフトウェア情報学部 卒業論文 (2015)
4) 中田育男，渡邊担，佐々政孝，滝本宗宏：コンパイラの基盤技術と実践 コンパイラ・インフラストラクチャCOINS を用いて，朝倉出版 (2008)
5) Graphviz - Graph Visualization Software，http://graphviz.org/
6) Emden Gansner，Eleftherios Koutsofios，Stephen North，とみながだいすけ (訳)：dot を使ったグラフ描画，http://www.cbrc.jp/tominaga/translations/graphviz/dotguide.pdf

付　　録

A.1 dot 言語のスクリプト

```
digraph {
  layout = dot;
  node [fontsize = 20];
  edge [arrowsize = 1, fontsize = 20];
  "2"[shape = octagon, fixedsize = true, width = 0.8, height = 0.5];
  "a_1"[shape = ellipse];
  "3"[shape = octagon, fixedsize = true, width = 0.8, height = 0.5];
  "b_1"[shape = ellipse];
  "4"[shape = octagon, fixedsize = true, width = 0.8, height = 0.5];
  "c_1"[shape = ellipse];
  "5"[shape = octagon, fixedsize = true, width = 0.8, height = 0.5];
  "a_2"[shape = ellipse];
  "b_2"[shape = ellipse];
  "6"[shape = octagon, fixedsize = true, width = 0.8, height = 0.5];
  "7"[shape = octagon, fixedsize = true, width = 0.8, height = 0.5];
  "a_3"[shape = ellipse];
  "=0"[shape = box, label = "=", fixedsize = true, width = 0.5, height = 0.5];
  "=1"[shape = box, label = "=", fixedsize = true, width = 0.5, height = 0.5];
  "=2"[shape = box, label = "=", fixedsize = true, width = 0.5, height = 0.5];
  "=3"[shape = box, label = "=", fixedsize = true, width = 0.5, height = 0.5];
  "=4"[shape = box, label = "=", fixedsize = true, width = 0.5, height = 0.5];
  "=5"[shape = box, label = "=", fixedsize = true, width = 0.5, height = 0.5];
  "=6"[shape = box, label = "=", fixedsize = true, width = 0.5, height = 0.5];
  "=7"[shape = box, label = "=", fixedsize = true, width = 0.5, height = 0.5];
  "if8"[shape = diamond, label = "if", fixedsize = true, width = 0.8, height = 0.5];
  "!9"[shape = box, label = "!", fixedsize = true, width = 0.5, height = 0.5];
  "==10"[shape = box, label = "==", fixedsize = true, width = 0.5, height = 0.5];
  "2" -> "=0"[label = ref];
  "a_1" -> "=6"[label = ref];
  "a_1" -> "==10"[label = ref];
  "3" -> "=1"[label = ref];
  "b_1" -> "=4"[label = ref];
  "b_1" -> "==10"[label = ref];
  "4" -> "=2"[label = ref];
  "5" -> "=3"[label = ref];
  "b_2" -> "a_3";
  "6" -> "=5"[label = ref];
  "7" -> "=7"[label = ref];
  "=0" -> "a_1";
  "=1" -> "b_1";
  "=2" -> "c_1";
  "=3" -> "a_2";
  "=4" -> "b_2";
  "=5" -> "b_2";
  "=6" -> "a_2";
  "=7" -> "c_1";
  "if8" -> "=3"[label = F][style = dashed];
  "if8" -> "=4"[label = F][style = dashed];
  "if8" -> "=5"[label = T][style = dashed];
  "if8" -> "=6"[label = T][style = dashed];
  "!9" -> "if8";
  "==10" -> "!9";
}
```