情報処理学会論文誌　プログラミング　Vol.10　No.4　1–11 (July 2017)

変数の値の変化の可視化によるプログラム理解支援

小山 秀明[1,a)]　山田 俊行[1,b)]

受付日 2016年12月9日, 採録日 2017年2月17日

概要：プログラムを理解するためには，プログラム実行時の変数の値の変化を理解する必要がある．しかし，プログラミング初学者にとって，ソースコードと実行結果から変化を理解することは困難である．実行時の変数の値を知るためにデバッガを使うことは，初学者にとって難しい．そのため，プログラム実行時の変数の値を可視化するツールは有用である．既存のプログラム可視化ツールの多くが，ステップごとの可視化のみを行うため，プログラムを通して変数の値がどう変化したかが分からない．本論文では，プログラム全体を通した変化の理解を支援する可視化手法を提案する．1つの変数に着目したトレースとデータ依存を用いた可視化，ブロックごとの変数の値の変化の可視化，コード編集の前後の実行結果の比較，の3種類の可視化手法を提案し，理解支援ツールとして実現する．

キーワード：可視化，理解支援，初学者支援，データ依存，動作トレース

Visualizing Change of Variable's Value for Program Understanding

Hideaki Koyama[1,a)]　Toshiyuki Yamada[1,b)]

Received: December 9, 2016, Accepted: February 17, 2017

Abstract: Understanding how the value of each variable changes during the program execution is necessary for program understanding. However, novice programmers have difficulty to understand the change from a source code and the result of program execution. Using a debugger is one method to know the value of a variable during program execution. However, using a debugger is difficult for novice programmers. Therefore, a visualization tools for novice programmers are useful and a lot of such tools already exist. However, these tools visualize only change in each step. In this paper, we propose three visualization methods for understanding the change during the program execution. The first method is using a trace and data dependence. The second method is visualizing the change of a value of a variables in multiple steps. The third method is comparing the original program and the edited program. Moreover, we implement these methods as a tool.

Keywords: program visualization, comprehension support, novice support, data dependence, action trace

1. 背景

プログラミングにおいて，プログラムの動作を理解することは重要である．しかし，プログラミング初学者にとってソースコードとその実行結果からプログラムの動作を把握することは困難である．その主な要因は，プログラム実行時の変数の値の変化が分からないことである．

プログラム実行時の変数の値を知る方法に，デバッガを用いる方法がある．しかし，デバッガの操作を覚える必要があり，ブレイクポイントの設定が必要なため，デバッガは学習には不向きである．また，プログラム内に出力関数を組み込むことでも変数の値を調べられるが，可視化ツールを用いれば，学習者に余分な作業をさせずに理解を支援できる．そのため，初学者のためにプログラム実行中の変数の値を可視化するツールは有用であり，また，そのようなツールはすでに多く存在する．

ステップごとの変数の値を可視化することで，学習者は各ステップでの変数の値の変化が分かりやすくなる．しかし，プログラムの動作を理解するには，ステップごとの変化だけではなく，プログラム全体を通した変数の値の変化

[1] 三重大学大学院工学研究科
Graduate School of Engineering, Mie University, Tsu, Mie 514–8507, Japan
[a)] koyama@cs.info.mie-u.ac.jp
[b)] toshi@cs.info.mie-u.ac.jp

© 2017 Information Processing Society of Japan

を把握することも重要である．多くの可視化ツールはステップごとの可視化だけを行うため，大局的なプログラムの変化の理解につながらない．

本研究では，プログラムを通した変数の値の変化，コード編集時の変化，の2種類の可視化で理解支援を行う．変数がどのように使われるかを示すために，1つの変数の振舞いを抽出したトレースである「動作トレース」を提案する．動作トレースは，1つの変数が関わったコードとそのときの値から構成される．それぞれの変数の値の変化を動作トレースを用いて示し，変数間の関係を示すために，変数の代入と参照の依存関係を示すデータ依存を用いる．動作トレースとデータ依存を用いて，変数の値が求まる経路を示すことで，プログラムを通した変数の値の変化の理解を支援する．また，ソースコードを1行編集したときのプログラムの動作の変化を可視化し，編集の前後の動作を比較することで，そのコードの持つ意味の理解を支援する．そして，プログラムを通した変数の値の変化の可視化と，コード編集時の変化の可視化による理解支援ツールを実現する．

2. 関連研究

Javaプログラムの可視化ツールであるJeliot 3 [1]は，プログラムの1ステップごとの実行内容をアニメーションで表現する．処理の実行や変数の変化が初学者でも分かりやすいが，プログラミングに慣れてくると，値の取得や演算，変数への値の格納などの細かいアニメーションは冗長になる．1つの命令に対する動作は直感的に理解しやすいが，プログラム全体を通しての動作の理解にはつながりにくい．

Avis [2]はプログラム自動可視化ツールである．ソースコードを読み込み，フローチャート，実行時の振舞いを示す逐次型実行経路図，モジュール間のつながりを示すモジュール遷移型実行経路図を作成し表示する．実行経路を示すことにより，プログラムの動作の理解支援を行うが，学習者が実行時の変数の値の変化を把握する機能がない．

ETV [3]はプログラムを1度実行し，そのときの情報をトレースファイルとして記録したのち，そのトレースを基にプログラムの実行を再現し可視化するツールである．トレースを使うため，実行のステップを進めるだけでなく戻すこともできる．停止中の行をソースコード上で強調し，関数が呼び出された場合，呼び出された関数のコードを現在の関数のコードに積み重ねるように表示する．このように，関数のコールスタックをソースコードの積み重ねで示す．関数の呼び出しが分かりやすくなるが，関数内における変数の値の変化は可視化されていない．

AZUR [4]はCプログラムのブロック構造の可視化による学習支援環境である．分岐や繰返しなどの制御文の範囲が明確になるように字下げに沿って線を表示し，ボールが線の上を動くアニメーションを用いて逐次実行を示す．ステップごとの変化やブロック構造を可視化しているが，ブロックごとの変数値の変化の理解はサポートされていない．

VILLE [5]は特定のプログラムパラダイムに束縛されないプログラム可視化ツールである．プログラムの実行を同時に2種類の言語で見せることにより，特定の言語におけるプログラムの書き方ではなく，プログラムの基本概念の学習を支援する．また，プログラム実行中にそのプログラムに関する問題を解かせることで，学習者の考える機会を作り，理解を深めさせる．VILLEはプログラミング言語に共通する考え方の学習を目的としており，実行時の変数値の変化の理解は支援されていない．

pgtracer [6]はプログラムとトレース表の穴埋め問題を生成することで，プログラミング教育を支援する．問題に用いるプログラムと穴埋め箇所のパターンの定義を分離し，複数の難易度の穴埋めパターンを用意することで，学生の理解度に合わせた問題を出題する．自動採点機能があり，授業と自習の両方で活用できる．また，学習履歴の分析機能があり，学生の理解度の把握や教育の改善に役立つ．しかし，教育支援が目的であり，理解支援のための可視化は行われていない．

以上で述べたように，ステップごとの可視化や，プログラム全体を静的な図として可視化するツールは存在するが，プログラムを通した変化の理解の支援が不十分である．また，コードを編集したときのプログラムの動作の変化に関する支援は行われていない．

3. 提案手法

本研究では，プログラムを通した変数の値の変化を理解するための3つの手法を提案する．

(1) 動作トレースとデータ依存を用いた可視化
(2) ブロックごとの変化の可視化
(3) コード編集による変化の比較

(1)は，調べたい変数の値の変化にのみ注目でき，変数間での値のやりとりが見えるため，その変数がプログラムを通してどう使われているかが分かりやすくなる．

(2)は，ifやforなどのプログラムのブロック単位での変数の変化が分かるため，ループ1回ごとの変化や，ループが複数回実行されたときの変化など，複数ステップ間の変数の変化が分かりやすくなる．

(3)は，(2)の可視化を用いてソースコードを編集したときの動作の変化を可視化することで，編集部分のコードがどのような意味を持つかの理解につなげる．

これらの手法とステップごとの可視化を実装することで，1ステップごと，複数ステップ間，コード編集の前後，の3種類の変化を可視化する理解支援ツールを実現する．

4. 実行時の変数の値の変化

4.1 ステップごとの変化の可視化

ステップごとの変化を示すために，変数とその値を表

© 2017 Information Processing Society of Japan

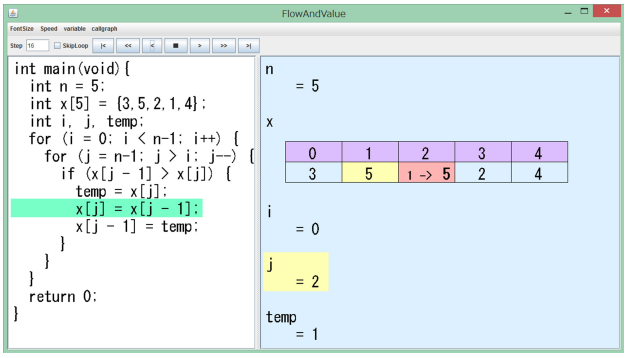

図 1　ステップごとの可視化
Fig. 1　Visualizing the change of each step.

示し，参照された変数と代入された変数を強調表示する．ツールは 1 ステップずつ実行を進めながら，そのときの変数の値を表示し，参照・代入された変数を強調する．ステップを戻せるため，一部の処理を繰り返し確認することもできる．

ツールでバブルソートを実行したときの画面を**図 1** に示す．左側にソースコードを表示し，停止中の行の背景色を強調する．右側にその行で有効な変数とその値を表示し，参照・代入された変数をそれぞれ別の色で強調する．変数を強調することにより，その行に関わる変数が一目で分かる．図 1 では，背景色で強調されている `x[j] = x[j - 1]` が実行され，`x[1]` と `j` が参照され，`x[2]` の値が 1 から 5 に変わることを示す．

4.2　動作トレースとデータ依存を用いた可視化

各変数の使われ方の理解を支援するために「動作トレース」を提案し，そのトレースとデータ依存を表示することで，値を求める経路を示し，変数間の関わりの理解を支援する．

4.2.1　動作トレースとは

本研究では，1 つの変数の動作を抽出したトレースである「動作トレース」を提案する．動作トレースは，1 つの変数に着目し，プログラム実行時にその変数がどのように使用されたかを抽出したトレースである．動作トレースは，着目した変数が参照・代入された処理と，そのときの変数の値から構成される．

表形式により動作トレースを示すことで，その変数が関わった処理を一覧でき，プログラム実行時にその変数がどのように使用され変化したかを分かりやすくする．**図 2** にツールが表示する表形式の動作トレースを示す．表は「参照」,「値」,「代入」の 3 列からなる．それぞれ，その変数が参照された処理，そのときの値，その変数に値が代入された処理を示す．これにより，その変数の値がどのように参照され，代入されたかが見やすくなる．着目した変数が配列の場合，参照されている要素，代入された要素，それ以外の要素をそれぞれ別の色で表示する．

j		
参照	値	代入
for(j=n-1;j>i;j--){	4	for(j=n-1;j>i;j--){
if(x[j-1]>x[j]){	4	
for(j=n-1;j>i;j--){	3	for(j=n-1;j>i;j--){
if(x[j-1]>x[j]){	3	
temp=x[j];	3	
x[j]=x[j-1];	3	

図 2　動作トレースの表示
Fig. 2　Showing an action trace.

x		
参照	値	代入
	35214	int x[5]={3,5,2,1,4};
if(x[j-1]>x[j]){	35214	
if(x[j-1]>x[j]){	35214	
temp=x[j];	35214	
x[j]=x[j-1];	35224	x[j]=x[j-1];
	35124	x[j-1]=temp;
if(x[j-1]>x[j]){	35124	
temp=x[j];	35124	
x[j]=x[j-1];	35524	x[j]=x[j-1];
	31524	x[j-1]=temp;
if(x[j-1]>x[j]){	31524	
temp=x[j];	31524	
x[j]=x[j-1];	33524	x[j]=x[j-1];
	13524	x[j-1]=temp;

図 3　配列変数の動作トレース
Fig. 3　The action trace of array x.

配列を選択したときの動作トレースを**図 3** に示す．各行での処理に配列のどの要素が関わっているかが一目で分かる．図 3 の値の列を見ると，参照・代入される要素が右から左へと移っていくのが分かる．これは，右端の要素から比較を行い，小さい値が左側へと入れ替えられていることを示している．

4.2.2　データ依存

変数の値がどの行で代入されたか，代入時にどの変数を参照したかの関係を表すデータ依存を示す．1 つの変数に関係する処理を一覧できる動作トレースに加え，データ依存を表示することで，複数表示された動作トレース間の関係を示し，変数間の関わりを示す．

表の中から変数を選択することで，その変数に関する表（「値」と「代入」の 2 列からなる表）が追加で表示され，関係する変数の値がどう計算されたかを調べられる．1 つの変数が関わった処理とそのときの値の一覧を示すことで，プログラムを通して変数がどのように変化し，使用されたかが分かる．

© 2017 Information Processing Society of Japan

図 4　データ依存の表示
Fig. 4　Showing the data dependence.

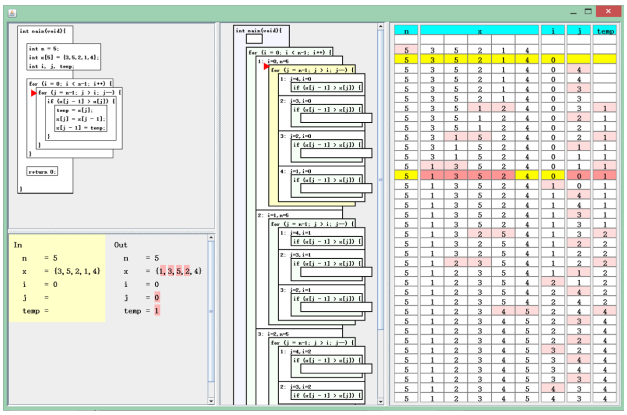

図 5　ブロックごとの可視化
Fig. 5　Visualizing the change of each blocks.

図 4 は，一番下の行の配列 x の 0 番目の要素（値 1）のデータ依存を表示している．選択された値が依存している処理と値の背景色を強調する．値から処理へ伸びる矢印は，その値が指し先の処理で使われていることを示す．強調部分と矢印をたどることで，その値が求められた経路が分かる．また，その値を求めるために他の変数がどう関わったかが分かる．図 4 の強調された部分と矢印をたどると，x の 4 番目の要素だった値 1 が，x の 0 番目の要素へと移ってきたことが分かる．

4.3　ブロックごとの変化の可視化

プログラムのブロック単位での実行を可視化する．ブロック単位で変数の値の変化が可視化されることにより，1 ステップずつ実行を追いかけるだけでは気付きにくい，ループ 1 回ごとの変数の値の変化など，複数ステップによる変化に気付きやすくなる．

ツールの実行画面を図 5 に示す．ソースコードのブロック構造，ブロックによる実行経路，ブロックにおける変数の値，ブロックごとのトレースを表示する．

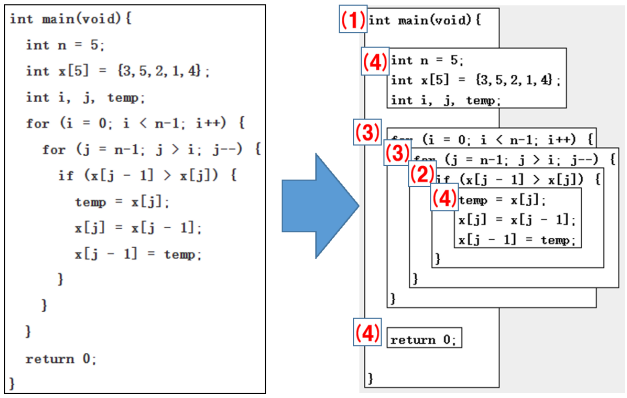

図 6　プログラムのブロック
Fig. 6　Blocks of program.

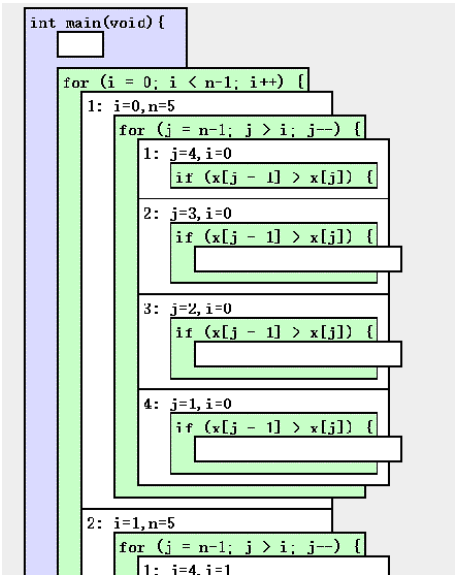

図 7　ブロックによる実行経路の表示
Fig. 7　Showing execution path using blocks.

4.3.1　ソースコードのブロック構造

本手法では以下の 4 つをブロックとして扱う．

(1) 関数
(2) if などの分岐構造
(3) while などのループ構造
(4) 逐次実行される文の列

バブルソートのソースコードを入力したときのブロックを図 6 に示す．枠で囲まれた部分が 1 つのブロックである．ブロックの左側の数字は説明のために書き加えたものであり，上記の (1)〜(4) に対応する．

4.3.2　実行経路

このブロックを用いて実行経路を表示する．図 7 は外側の for ループの 1 回目の終了までを示す．for ブロックの中のブロックは繰返しを意味しており，図 7 では内側の for ループが 4 回繰り返されたことを表す．繰返しを示すブロックの左上には，繰返しの回数と，ループ条件に関わる変数の値が表示される．また，分岐のブロックでは，

© 2017 Information Processing Society of Japan

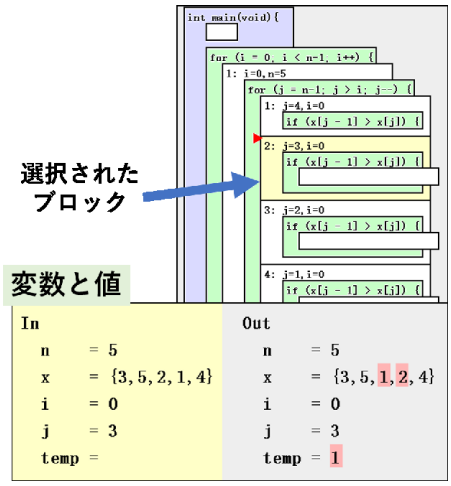

図 8　ブロック内における変数の値
Fig. 8　The values of variables in the blocks.

条件式が真であった場合は内側にブロックを表示し，偽であった場合は，内側のブロックは表示しない．ブロックを用いて実行経路を示すことで，ループが何回繰り返されたか，どの分岐を通ったかが一目で分かる．

4.3.3　変数の値の表示

実行経路の中のブロックが選択されると，そのブロックの変数の値を表示する．図 7 の内側の for ループの 2 回目を選択したときの表示を**図 8** に示す．そのブロックに入るときの値と，そのブロックから出るときの値の両方を表示し，ブロック内に代入がある場合，代入されたすべての変数の値を強調表示する．選択されたブロックが強調され，そのブロック内での変数の値が表示されている．左側に入るときの値，右側に出るときの値が表示され，代入された値が強調されている．これにより，ブロック単位での変数の値の変化が見られる．図 8 の配列 x の強調されている要素を見ると，ブロックに入るときと出るときで値が入れ替わっていることが分かる．

また，入るときまたは出るときの値が実行経路のどの位置での値であるかをブロックの左側の三角形で明示する．図 8 では入るときの値が強調されており，選択されたブロックの左上に三角形がある．出るときの値に切り替えると三角形の位置がブロックの左下に変わり，出るときの値が強調される．

4.3.4　ブロックごとのトレース

ブロック単位での変数の値のトレースを表示することで，全ステップのトレースに比べ，小さいサイズの表でプログラム全体での変数の値の変化を示せる．ブロックごとの変数の値のトレースを**図 9** に示す．

ブロックは，入るときと出るときの 2 つの値を持っている．前後のブロックとの間や，内側または外側にあるブロックとの間で，値が重複する場合がある．たとえば，if 文の内側のブロックから出るときの値と，if 文のブロックから出るときの値は同じである（**図 10**）．このような重複

n	x					i	j	temp
5	3	5	2	1	4			
5	3	5	2	1	4	0		
5	3	5	2	1	4	0	4	
5	3	5	2	1	4	0	4	
5	3	5	2	1	4	0	3	
5	3	5	2	1	4	0	3	
5	3	5	1	2	4	0	3	1
5	3	5	1	2	4	0	2	1
5	3	5	1	2	4	0	2	1
5	3	1	5	2	4	0	2	1
5	3	1	5	2	4	0	1	1
5	3	1	5	2	4	0	1	1
5	1	3	5	2	4	0	1	1
5	1	3	5	2	4	0	0	1
5	1	3	5	2	4	1	0	1
5	1	3	5	2	4	1	4	1
5	1	3	5	2	4	1	4	1
5	1	3	5	2	4	1	3	1
5	1	3	5	2	4	1	3	1
5	1	3	2	5	4	1	3	2
5	1	3	2	5	4	1	2	2
5	1	3	2	5	4	1	2	2
5	1	2	3	5	4	1	2	2
5	1	2	3	5	4	1	1	2
5	1	2	3	5	4	2	1	2
5	1	2	3	5	4	2	4	2
5	1	2	3	5	4	2	4	2
5	1	2	3	4	5	2	4	4
5	1	2	3	4	5	2	3	4
5	1	2	3	4	5	2	3	4
5	1	2	3	4	5	2	2	4
5	1	2	3	4	5	3	2	4
5	1	2	3	4	5	3	4	4
5	1	2	3	4	5	3	4	4
5	1	2	3	4	5	3	3	4
5	1	2	3	4	5	4	3	4
5	1	2	3	4	5	4	3	4

図 9　ブロックごとの変数の値のトレース
Fig. 9　The trace of variables in blocks.

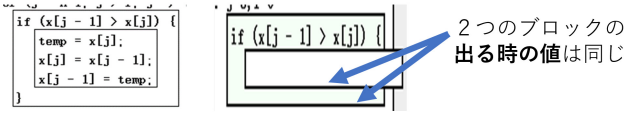

図 10　値が重複するブロック
Fig. 10　The blocks has duplicate values.

を残したまま，各ブロックの 2 つの値をすべて表示すると，無駄の多いトレースとなってしまう．そのため，ブロック間で実行される処理がない場合，表示を省略する．

変数の値が変更された箇所を強調することで，それぞれの変数がいつ変更されたかが分かりやすくなる．また，選択されているブロックに入るときの値と出るときの値の行を強調している．ブロックから出るときの値の行では，そのブロック内で変更されたすべての変数が強調される．

5.　編集による変化の可視化

1 行のコード編集により，プログラムの動作がどう変化するかを可視化することで，その行がプログラムにどのような影響を与えているかを示す．たとえば，ソートアルゴリズムの入替えの条件式の不等号を逆にすると，ソートの結果が逆順になる（昇順であれば降順になる）．これにより，入替えの条件式がソートの向きに関わっていることが分かる．このように，実際にコードを編集し，その結果を比較することで，コードの持つ意味の理解につなげる．また，この値を変えるとどうなるだろう，この式を変えると

© 2017 Information Processing Society of Japan

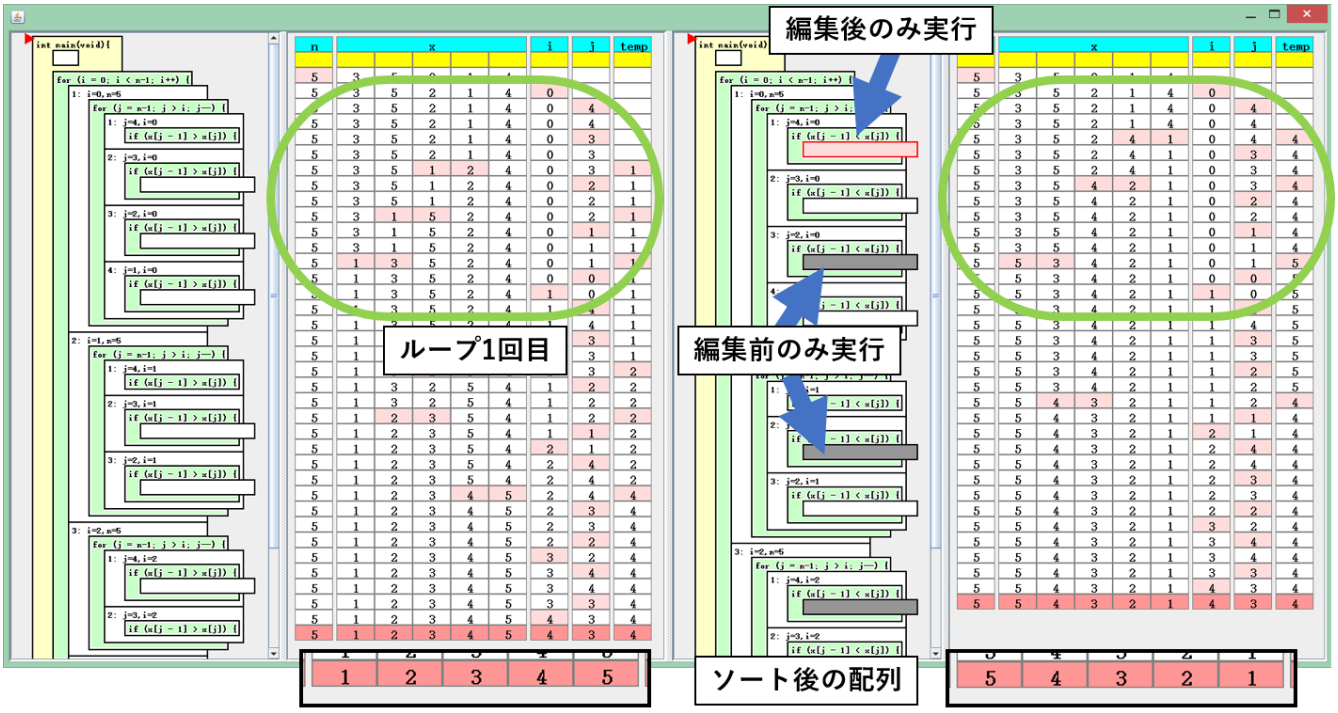

図 11　実行の比較
Fig. 11　Compare executions.

どうなるだろう，といった学習者の疑問に対して即座に答えを示せる．

図 11 に，バブルソートの入替えの条件式の不等号を逆にしたときの，編集前と編集後の両方の実行結果を示す．左側が編集前，右側が編集後の実行結果である．実行結果として，4.3 節で説明した，ブロックによる実行経路と，ブロックごとのトレースが表示される．編集後の実行経路では，編集後のみ実行されるブロックと，編集前のみ実行されるブロックを，それぞれ別の色で強調する．これにより，実行経路がどう変わったかが一目で分かる．編集前と編集後のどちらかの実行経路のブロックを選択すると，もう一方の実行経路の対応するブロックが選択される．トレースの黄色で強調される行を比較することで，編集による値の変化を調べられる．

それぞれのトレースの最後の行を見ると，配列の数字が逆順に並んでいることが分かる．外側のループの 1 周目を見ると，編集前と編集後で入替えを行ったときの要素の大小関係が逆になっているが，どちらも同じように配列の左端から右端へと入替えを行っていることが分かる．

6.　ツールの実装

本研究では，提案する 3 つの可視化手法を理解支援ツールとして実現した．本ツールは C 言語のプログラムを可視化する．様々なプラットフォームで実行できるよう，Java 言語でツールを実装した．本ツールは，起動時に GDB [7] を用いて入力されたプログラムを最後まで実行し，あらかじめ実行開始から実行終了までの実行情報（実行トレース）

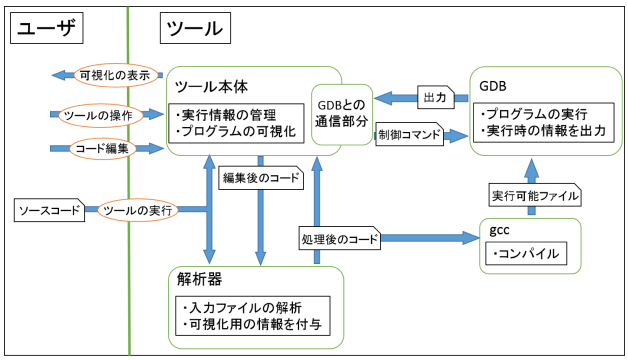

図 12　ツールの構成
Fig. 12　The structure of the tool.

を取得する．この実行トレースを用いてプログラムの実行を再現し，可視化する．図 12 にツール全体の構成を示す．本ツールは，ユーザからソースコードを受け取り，解析・実行・可視化の順に処理する．

6.1　解析器

図 12 の下側に示されている解析器について説明する．入力されたソースコードを解析し，コードの加工と解析情報の付与をする．加工と情報付与を行ったソースコードを，新しいファイルに保存し，編集後のソースコードを GDB で実行する．図 13 に，二分探索のソースコードと解析器で加工したコードの一部を示す．以下に解析器の詳細を述べる．

6.1.1　ソースコードの解析

構文解析を行い，各行について，宣言・代入・参照され

© 2017 Information Processing Society of Japan

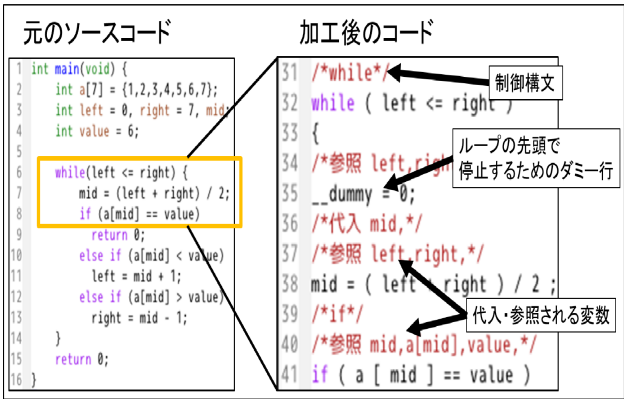

図 13　ソースコードの加工
Fig. 13　Processing the source code.

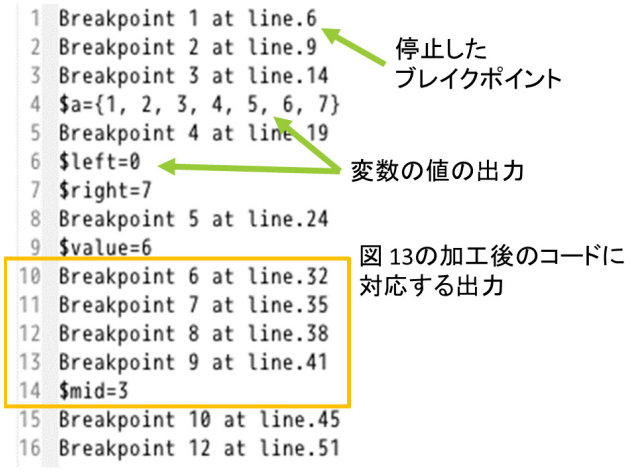

図 14　GDB の出力
Fig. 14　The output of GDB.

た変数，その行の制御構文の種類を調べる．解析で得た情報は，コメントとしてソースコードに付与する．たとえば，`for (i=0; i<n-1; i++)` であれば，制御構文は `for`，参照される変数は `i` と `n`，代入される変数は `i`，といった解析結果をこの行のコメントとして付与する．図 13 のように，コメントは各行の前に記述する．ツール本体で再びソースコードの解析をしなくても，コメントを読むことで，解析結果が得られる．

6.1.2　GDB による実行のための加工

本ツールでは，ブレイクポイントの設定と，6.1.1 項の解析情報を行単位で管理する．そのため，ユーザのソースコード中で，1 行に複数の文が書かれている場合，それぞれの文に対してブレイクポイントを設定できない．また，1 文が複数行にわたって書かれている場合，1 文に対する情報を分割して管理してしまう．そのため，1 行 1 文となるように，改行の挿入と削除を行う．

すべての行にブレイクポイントを設定し，continue コマンドを反復実行することで，プログラムの実行を制御する．実行中の値の出力や，制御の管理のために，元のソースコードでは停止しない行でも停止する必要がある．そのため，必要な箇所に意味のない処理を行うダミーの行を追加する（例：`__dummy = 0;`）．たとえば，continue コマンドによる再開では，ループの先頭行（`while (left <= right)`）にループ 2 回目は停止しない．図 13 のようにダミーの行を挿入する．

6.2　実行と再現

GDB によるプログラムの実行と，ツールによる実行の再現について説明する．本節の内容は，図 12 の中央のツール本体の処理と，右側の GDB との通信にあたる．

GDB を用いてプログラムを実行した場合，実行の停止と再開は可能であっても，前の状態へは戻せない．そこで，あらかじめプログラムを最後まで実行し，取得した実行情報を用いた実行の再現により，すべての実行ステップへの自由なアクセスを可能にする．

6.2.1　GDB によるプログラムの実行

解析器で加工したソースファイルを，gcc を用いてコンパイルし，GDB で実行する．プロセス間通信により，GDB の出力の送信と，出力に応じた制御コマンドの送信を繰り返す．

本ツールは，すべての行にブレイクポイントを設定し，continue コマンドのみで実行を制御する．使用する GDB のコマンドを以下に示す．

- breakpoint n：n 行目にブレイクポイントを設定
- run：プログラムを実行
- print e：式 e の値を出力
- continue：実行の再開

GDB が “Program exited normally.” や，“Program terminated with signal” などの終了メッセージを出力した場合，プログラムが停止したと判断し，GDB との通信を終了する．

ブレイクポイントで停止したとき，print コマンドで出力される変数の値は，停止行を実行する前の値である．そのため，変更される変数の値を，次に停止したブレイクポイントで出力する必要がある．停止行と出力する変数を 1 ステップずらし，変更後の値を取得するために，停止行で変更される変数名のリストを保持し，次のブレイクポイントでリストに含まれる変数の値を出力する．

GDB の出力は，停止したブレイクポイントと行番号，print コマンドで出力した変数の値，終了メッセージ，の 3 種類である．GDB が 1 行出力するごとに，その出力に応じてコマンド（continue または print）が送信される．図 13 の二分探索のプログラムを実行したときの，GDB の出力の一部を**図 14** に示す．図 14 の枠で囲まれた箇所は，図 13 に示した部分のコードに停止したときの出力である．加工後のコードの 38 行目に `mid` への代入文があるが，この行に停止したとき（図 14 の 12 行目）に `mid` の値を出力せ

© 2017 Information Processing Society of Japan

ず，次のブレイクポイントに停止したときに mid の値を出力している．

6.2.2　GDB の結果による実行の再現

GDB でプログラムを実行することで，停止した行と，そのときの変数の値の情報が得られる．実行結果を記録するために，2 つのデータ構造を用いる．1 つ目は，停止行とその行で使用される変数からなる対を要素としたリスト（以降，ステップのリスト）である．GDB から得た停止行と，6.1.1 項の解析で得た各行の情報を用いて，各ステップで実行された行と使用された変数を，ステップのリストに格納することで，実行経路を記録する．2 つ目は，値が変更されるステップ番目と変更後の値からなる対を要素としたリスト（以降，変数値の履歴）である．変数ごとに変数値の履歴を生成し，変更されるステップと変更後の値を記録する．

ステップのリストと，変数値の履歴を用いて，実行を再現する．n ステップ目の実行を再現する場合，ステップのリストの n 番目の要素を参照することで，停止行と使用される変数を取得し，各変数の変数値の履歴から，n ステップ目までで最後に変更された値を検索し，n ステップ目における各変数の値を取得する．

6.3　提案手法の実装

本研究で提案する 3 種類の可視化手法の実装方法について説明する．

6.3.1　動作トレースとデータ依存

図 13 に示した二分探索のプログラムの変数について，動作トレースとデータ依存を表示したものを**図 15** に示す．中央の位置を示す mid，左端の位置を示す left，右端の位置を示す right，探索する配列 a，の 4 つの動作トレースが表示されている．それぞれの動作トレースは，ステップのリストと変数値の履歴から生成される．ステップのリストから着目する変数が使用されている要素を抽出し，変数値の履歴から抽出したステップでの値を参照することで，関係する文とそのときの値を取得する．

データ依存は，セルの強調と矢印で示す．着目する行から上方向に，代入文が見つかるまで探索し，探索した行の値と代入文を強調する．その行に依存する変数があれば，その変数との間に矢印を表示し，依存する変数から再び探索する．依存する変数がなければ探索を終了する．

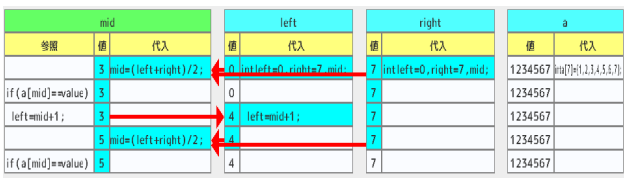

図 15　二分探索の動作トレース
Fig. 15　The action trace of binary search.

図 15 は，mid の最終行の値についてのデータ依存を表示している．このときのデータ依存の求め方を説明する．図 15 の mid の動作トレースでは，下から 2 行目に代入文 mid=(left+right)/2 がある．left と right に依存していることが分かる．それぞれの動作トレースの同じ行（下から 2 行目）から，mid に向けて矢印を表示し，left と right のそれぞれについて，同様にデータ依存を求める．

6.3.2　ブロックごとの変化

ソースコードのブロック構造と，各ブロックの先頭行と末尾行は 6.1.1 項の解析により得られる．各ブロックの先頭行と末尾行と，ステップのリストの停止行から，通過したブロックが分かる．通過したブロックを，関数を根とし，内側のブロックを子とする木構造で管理する．木構造の各ノードは以下の情報を持つ．

- ブロック名（先頭行やループ回数，または空白）
- if や for などの制御構文
- 内側のブロックのリスト
- 入るときと出るときの各変数の値
- 対応するトレースの行番号

ブロックごとのトレースは，重複する箇所を省略している．値の重複が起きるブロックを以下に示す．for ループは，ループごとにカウンタが更新されるため，前後のブロックと値が重複しない．

- 兄弟関係にある前後のブロックとの間（for ループを除く）
- n 回目のループを示すブロックと内側の最初のブロック
- 内側の最後のブロックとの間（for ループを除く）

6.3.3　コード編集による変化

ツール上でソースコードが編集された場合，編集後のコードを別のファイルに保存し，起動時と同様に，解析と実行を行う．ソースコードが 1 行編集された時点で，編集後の実行情報を取得するため，複数行が編集されることはない．

通過ブロックの木構造を比較することで，編集による実行経路の変化を取得する．編集による木構造の変化の例を**図 16** に示す．ソースコードのブロック構造が変化していないと仮定し，編集前と編集後の木構造の深さ優先探索を

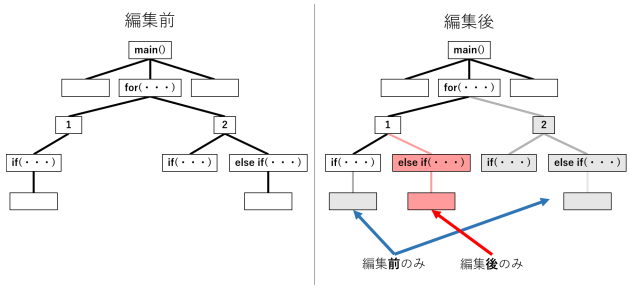

図 16　編集による木構造の変化
Fig. 16　The change of tree structure by editing.

© 2017 Information Processing Society of Japan

並行して行う．片方にしか存在しないブロックを発見した場合，そのブロックとその内側のブロックを片方のみのブロックとし，共通するブロックから探索を再開する．

現在は，ソースコードのブロック構造が変化する場合に対応していない．しかし，ブロック構造が変化する 1 行の編集は，中括弧を用いない 1 行の if や for の追加・削除，改行を用いず 1 行に記述した中括弧を用いる if や for の追加・削除のみである．そのため，木構造の変化は 1 つのノードの枝の追加・削除のみである．変化したブロック構造は，編集行から分かるため，追加・削除されたブロック構造に対する処理を追加することで，比較方法を変えずに対応できる．

6.4　現実装で扱えないプログラム

解析器と可視化部分の実装の都合上扱えないプログラムと，それぞれに対する解決策を以下に示す．

- switch，break など未対応の構文
  if，else，for，while，do 以外の制御構文や，typedef などの構文に対応していない．すべての行にブレイクポイントを設定し，continue コマンドの繰返しで実行を制御している．そのため，switch や break，continue など，他の制御構文も，解析器を対応させることで，現在対応している構文と同様に扱える．
- ポインタ
  現実装ではポインタを扱っていない．しかし，ポインタの値は，GDB から取得できるため，ポインタの可視化に必要な情報は容易に得られる．
- 構造体
  構造体の構文には対応していないため扱えない．また，構造体変数に着目した動作トレースを表示した場合，値の列のセルが複雑になるため，可視化の工夫が必要になる．
- ユーザに入力を要求するプログラム
  現実装では，ユーザに入力を要求するプログラムは扱えない．入力を必要とする箇所を調べていないため，GDB が入力待ちをしていることを判断できない．また，可視化前に入力を要求するため，どの行での入力が要求されているかをユーザに示す工夫が必要になる．
- GDB で異常終了するプログラム
  ユーザのプログラムが異常終了した場合への対応ができていない．GDB のエラーメッセージが返されたときは，その時点までの実行トレースを用いることで，異常終了の直前までの実行を可視化できる．
- 終了しないプログラム
  可視化の前に，プログラムを最後まで実行するため，無限ループなどに陥り終了しないプログラムは，可視化できない．一定のステップで実行を打ち切り，その時点までの可視化を行うことで，何も表示されない状況の回避は容易である．
- 関数
  メイン関数以外の関数があるプログラムも，他のプログラムと同様に，GDB から実行情報を取得し，ツール上で実行を再現できる．しかし，ステップごとの可視化と動作トレースは関数に対応しているが，特別な表示は行っていない．ステップごとの可視化では，実行中の関数で有効な変数だけが表示され，呼び出し元の関数での変数の値は参照できない．動作トレースでは，どの関数の文であるかは表示されない．また，ブロックごとの可視化は，実行経路とトレースの表示方法が定まっていないため，関数に対応していない．

7.　関連研究との比較

2 章で紹介した，プログラムの実行を可視化する 4 種類のツール（Jeliot 3，ETV，AZUR，VILLE）と本ツールを比較した表を，**表 1** に示す．

編集した行と編集による変化の関わりを明示するために，本ツールではソースコードの編集を 1 行のみに制限している．Jeliot 3，AZUR，VILLE は，ツール上で自由にソースコードを編集し実行できるが，複数行の編集を許した場合，示される変化がどの行の編集に起因するかが分からなくなってしまう．

本ツールと ETV は，先に実行情報を取得し可視化するため，任意のステップへ移動できるが，無限ループがありプログラムが停止しない場合は表示できない．一定ステップで実行を打ち切ることで対応できる．

本ツールは他のツールに比べ，実行するプログラムに対する入出力の機能が乏しい．しかし，出力については，出力時の変数の値を調べることで出力内容を確認できる．入力についても，入力に相当する行を編集することで対応できる．

すべてのツールがステップごとの変数の値を表示しているが，ステップごとの変数の変化を示しているのは，本ツールと AZUR のみである．また，複数ステップ間の値の変化を可視化しているのは本ツールのみである．ステップごとの変数の値にのみ着目してしまうと，ループ 1 回ごとの変化や，ループ全体を通した変化など，複数ステップによる変化が見えにくくなるため，複数ステップ間の値の変化を可視化することは有用である．

編集によるプログラムの動作の変化を可視化しているのは，本ツールのみである．編集による動作の変化を確認し，その編集がプログラムにどのような影響を与えたかを理解することは重要である．そのため，編集による変化を可視化することは有用である．

複数ステップ間の値の変化の可視化と，編集による変化の可視化が，本ツールにおける独自性である．

© 2017 Information Processing Society of Japan

表 1　ツールの比較
Table 1　Comparison among tools.

		本ツール	Jeliot 3	ETV	AZUR	VILLE
全般	可視化対象の言語	C	Java	C，Java	C	Java，擬似コード，C++
	プログラムの実行方法	GDB （トレースの再生）	Dynamic Java	gcc，jdi （トレースの再生）	GDB	内蔵インタプリタ
	ステップ実行	○	○	○	○	○
インタラクション	ソースコードの編集	△(1 行のみ)	○	×	○	○
	プログラムへの入力	×	○	×	○	×
	標準出力	×	○	○	○	○
	問題の出題	×	×	×	×	○
可視化	ステップごとの変数の値	○	○	○	○	○
	ステップごとの値の変化	○	×	×	○	×
	ブロック構造	○	×	×	○	×
	複数ステップ間の値の変化	○	×	×	×	×
	コールスタック	×	○	○	○	○
	編集による変化の可視化	○	×	×	×	×
	アニメーション	×	○	×	○	×

8. 結論と課題

プログラミング初学者がプログラムの動作を理解するために，変数の値の変化の可視化手法を提案した．1 つの変数に注目したトレースである「動作トレース」を提案し，そのトレースとデータ依存を用いて，プログラム実行時の変数の値の変化の理解を支援する．動作トレースにより，その変数がどのように参照・代入されたかを示す．データ依存を表示し，変数の値が求められた経路と変数間の関係を示す．ステップごとの変数の変化の可視化と，トレースとデータ依存を用いた可視化により，ステップごとの変化とプログラムを通した変化の両方の理解支援を行う．また，ブロックごとの変数の値の変化を可視化することで，複数ステップ間における変数値の変化の理解を支援する．

ソースコードを 1 行編集したときのプログラムの編集前と編集後の両方の実行結果を可視化することで，実行結果の比較を通じた各行の振舞いの理解を支援する手法を提案した．

これらの手法を実装することで，ステップごとの変数の値の変化，プログラムを通した変数の値の変化，ソースコード編集によるプログラムの動作の変化を可視化するツールを実現した．

今後の課題として，表示される情報量が多すぎることがあげられる．ステップごとの変化，動作トレース，ブロックごとの変化，編集による変化，の 4 種類の可視化を行っているが，それぞれの可視化の関連を十分に示せていない．そのため，どれを見ればよいか分からない，それぞれの可視化の関係が分からない，など学習者が戸惑う可能性がある．また，配列の要素数やループ回数が多くなると，動作トレースやブロックごとの実行経路・トレースが大きくなってしまう．スクロール機能があるため，画面内に収まらないものも表示できるが，表示量の増加は学習者の負担となる．そのため，表示の省略機能（配列やブロックの折り畳みなど）を実装する必要がある．

現在，本ツールの独自性である，複数ステップ間の値の変化と編集による変化の可視化の有用性を評価できていない．ツールによる理解支援の効果を評価するための実験を行う必要がある．

参考文献

[1] Moreno, A., Myller, N. and Sutinen, E.: Visualizing Programs with Jeliot 3, *Proc. AVI '04*, pp.373–376 (2004).

[2] 喜多義弘，片山徹郎，冨田重幸：Java プログラム読解支援のためのプログラム自動可視化ツール Avis の実装と評価（ソフトウェアシステム），電子情報通信学会論文誌 D，情報・システム，Vol.J95-D, No.4, pp.855–869 (2012).

[3] Terada, M.: ETV: A Program Trace Player for Students, *Proc. 10th Conference on Innovation and Technology in Computer Science Education*, pp.118–122 (2005).

[4] 古宮誠一，今泉俊幸，橋浦弘明，松浦佐江子：プログラミング学習支援環境 AZUR—ブロック構造と関数動作の可視化による支援，情報処理学会研究報告，ソフトウェア工学研究会報告，Vol.2014, No.5, pp.1–8 (2014).

[5] Rajala, T., Laakso, M.J., Kaila, E. and Salakoski, T.: VILLE – A Language-Independent Program Visualization Tool, *Proc. 7th Baltic Sea Conference on Computing Education Research*, Vol.88, pp.151–159 (2007).

[6] 掛下哲郎，柳田　崚，太田康介：穴埋め問題を用いたプログラミング教育支援ツール pgtracer の開発と評価，情報処理学会論文誌教育とコンピュータ（TCE），Vol.2, No.2, pp.20–36 (2016).

[7] GDB: The GNU Project Debugger, available from ⟨http://www.gnu.org/software/gdb/⟩.

© 2017 Information Processing Society of Japan

小山 秀明

1993 年生．2015 年三重大学工学部情報工学科卒業．2017 年同大学大学院修士課程修了．

山田 俊行　（正会員）

1972 年生．ソフトウェア科学会会員．1999 年筑波大学大学院博士課程工学研究科修了．工学博士．1999 年筑波大学電子・情報工学系助手．2002 年三重大学工学部情報工学科助手．2005 年三重大学大学院工学研究科情報工学専攻講師．ソフトウェア基礎の研究に従事．本会シニア会員．

© 2017 Information Processing Society of Japan