ソフトウェアテストシンポジウム 2005
JaSST'05: Japan Symposium on Software Testing 2005

データフローパステスト法の分析と改良

河野 哲也† 西 康晴†

†電気通信大学大学院電気通信学研究科システム工学専攻 〒182-8585 東京都調布市調布ヶ丘1-5-1
E-mail: †{kouno, nishi}@se.uec.ac.jp

あらまし 現在、プログラマが自分の作ったソースコードのテストを行っているため、テストの質は各プログラマのスキルに依存している．プログラマにはテスト法を用いることが望まれ、優れた手法が必要である．プログラマが行うテスト(ホワイトボックステスト)の代表的なものにパスを選択しテストを行う方法がある．ただし、実際の開発現場ではむやみにパスを選択し、テスト工数を増やすことは許されない．そこで、必要十分なパスを選択するのがデータフローパステスト法である．本研究では、既存のデータフローパステスト法の分析を行い、改良した方法を提案する．
キーワード ホワイトボックステスト，パステスト法，データフローパス，テストパス

Analysis and Improvement of Data Flow Path Testing

TetsuyaKOUNO† YasuharuNISHI†

†Department of Systems Engineering, The University of Electro-Communications,

1-5-1 Chofugaoka, Chofu, Tokyo 182-8585 Japan
E-mail: †{kouno, nishi}@se.uec.ac.jp

Abstract Today, programmers test programs by themselves. As a result, the quality of a testing is dependent on each programmer's skill. Programmers need to use a testing method. A typical White Box Test is a method to test programs to select paths. But we can't select paths recklessly. Because we don't have a lot of effort on software development. Data Flow Path Testing is a method to select testing paths a necessary and sufficient. This paper analyzes to improve an existing Data Flow Path Testing, and proposes improvement of Data Flow Path Testing.
Keyword White Box Test, Path Testing , Data Flow Path, Testing Path

1. まえがき

1.1. 研究の背景

現在のソフトウェア開発の現場では、約40～50%の工数がテストに費やされている．[1]テストが効率よくできるかどうかが、ソフトウェア開発全体の生産性を大きく左右している．テストの良し悪しがソフトウェア開発に与える影響は極めて大きい．

ソフトウェアテストは一般的にホワイトボックステストとブラックボックステストに分けられる．ホワイトボックステストは構造テストとも呼ばれ、コーディング用のテスト技法として重要とされている．ホワイトボックステストのテスター(プログラマ)は日常的に自分が作ったソースコードをテストし、それが正しく動作することを確認している．プログラマによるテストは各個人のスキルに依存していることが多く、ある技法が用いられていることは稀である．このことにより、プログラマの成果物(ソースコード)に含まれるバグの数は各プログラマによってまちまちである．よって、各プログラマがテスト法を用いてテストすることが望まれる．

1.2. パステスト法について

ホワイトボックステストの代表的方法にパステスト法がある．パスとはプログラム中の命令の実行の順序のことであり、パステストとはプログラム中からパスを選びテストを行うことである．

パステスト法とはすべてのパスをもらすことなくテストすることであり、この方法を実施すればすべてのバグは発見できる．しかし、一般的にプログラム中には天文学的な数のパスが存在するといわれている．Myers によると 100 行のプログラムでも 100 兆のパスが存在する．[2]実際の開発現場では、100 行のプログラムというのは現実的でなく、数千、数万行ということがあたりまえである．このことを考えると、実際のプログラムにはどれくらいのパスが存在するのだろうか．もちろん膨大な数になるはずである．

よって、実際にパステスト法を実施することは不可能である．そこで、データの流れに基づいてテストすべきパスを減らそうとするのがデータフローパステスト法[1]である．

1.3. 従来のデータフローパステスト法

現在、データフローパステスト法では参照網羅法や定義参照網羅法などが有名であるが、これらの方法は定義や参照の網羅性に注目するあまり、その実行結果が出力に現れるとは限らない．そこで、これらの方法より優れているとされているのは中條らが提案した「定義参照関係の連鎖に基づくテストパス決定法」[3]である．この方法は先ず実行結果に着目しパスを選択する.また､定義･参照の網羅性も十分考慮されている.

この論文では、プログラム中のデータの流れを２変数間の定義と参照の関係の連鎖と捉えテストパスを選択する方法を提案している．しかし、ここでは本質的なテストパスを抽出しているのかという議論はなされていない．

1.4. 研究の目的

本論文では、既存の方法として「定義参照関係の連鎖に基づくテストパス決定法」を対象とし、どのようなテストパスが抽出できていないか分析を行う．そして、その分析結果より改良したデータフローパステスト法を提案する．

1.5. 本論文の流れ

本論文の 2 章で既存の方法のパス選択の流れを説明し、3 章ではその分析を行い選択できないテストパスの特徴を示す．4 章では分析に基づき既存の方法の改良を行い､新たな方法を提案する.5 章で検証を行い、6 章ではあとがきとして本論文のまとめを行う．

2. 既存の方法のパス選択の流れ

「定義参照関係の連鎖に基づくテストパス決定法」は、1992 年に中條らによって提案された方法である．下記にこの方法の概要、手順を示し、プログラム例に適用する．

2.1. 概要

プログラム中の２変数間の定義と参照の関係を次の２種類で示している．例として図１のプログラム例を用いる．

・データ的関係

ブロック 3 で定義された y がブロック 5 における y の定義で参照されるときの関係.

y[5]←y[3]

・制御的関係

ブロック 1 で定義された x が条件判定で参照され、その結果で実行・不実行が決まるブロック 3 で変数 y が定義されるときの関係.

y[3]⇦ x [1]

個々の２変数間定義参照関係を用いて連鎖的に表すと次のようになる．

y[5]←y[3] ⇦ x[1]

これをデータフローパスと言う．

また、入力から出力までを２変数間定義参照関係で連鎖的にとらえたものをテストパスとしている．

2.2. テストパス選択の手順

プログラム中の変数の定義と参照の関係の流れを２変数間定義参照関係の連鎖と捉えた場合、データフローパスの抽出が可能になると提案している．データフローパスの抽出からテストパス選択までの流れを次のような手順にて示している．

手順 1:仕様と照合する出力を決める．

手順 2:すべての２変数間定義参照関係を列挙する．

手順 3:手順 2 で構成されるデータフローパスのうち手順 1 を含むものを抽出する．

手順 4:手順 3 で求めたデータフローパスを実行できるテストパスの中からすべての２変数間定義参照関係をテストするのに十分なものを選ぶ.

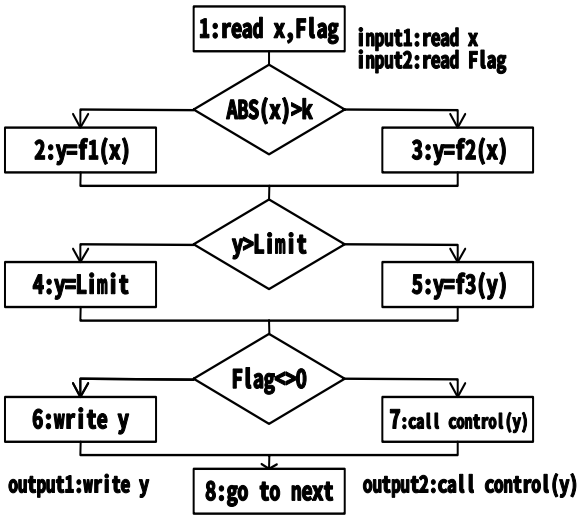

図１　プログラム例１

表 1:２変数間定義参照関係の一覧表

No	2変数間定義参照関係
1	output2[7]⇐Flag[1]
2	output2[7]←y[4]
3	output2[7]←y[5]
4	output1[6]⇐Flag[1]
5	output1[6]←y[4]
6	output1[6]←y[5]
7	y[5]⇐y[3]
8	y[5]⇐y[2]
9	y[5]←y[3]
10	y[5]←y[2]
11	y[4]⇐y[3]
12	y[4]⇐y[2]
13	y[3]⇐x[1]
14	y[3]←x[1]
15	y[2]⇐x[1]
16	y[2]←x[1]
17	x[1]←input1
18	Flag[1]←input2

表 2:データフローパスの一覧表

No	データフローパス
1	output2[7]←y[5]←y[2]←x[1]←input1
2	output2[7]←y[5]←y[3]←x[1]←input1
3	output2[7]←y[5]←y[2]⇐x[1]
4	output2[7]←y[5]←y[3]⇐x[1]
5	output2[7]←y[5]⇐y[2]
6	output2[7]←y[5]⇐y[3]
7	output2[7]←y[4]⇐y[2]
8	output2[7]←y[4]⇐y[3]
9	output2[7]⇐Flag[1]
10	output1[6]←y[5]←y[2]←x[1]←input1
11	output1[6]←y[5]←y[3]←x[1]←input1
12	output1[6]←y[5]←y[2]⇐x[1]
13	output1[6]←y[5]←y[3]⇐x[1]
14	output1[6]←y[5]⇐y[2]
15	output1[6]←y[5]⇐y[3]
16	output1[6]←y[4]⇐y[2]
17	output1[6]←y[4]⇐y[3]
18	output1[6]⇐Flag[1]

2.3. 適用例

上記手順を図 1 のプログラム例 1 に適用する．このプログラムは偏差 x に応じて制御量 y を計算し、ハードウェアに出力する一連の制御を行うことを目的としたものである．

適用結果を表 1~3 に示す．表 1 は手順 2 で得られた 2 変数間定義参照関係であり、表 2 は手順 3 で抽出されたデータフローパスを示す．表 3 は手順 4 で選択されたテストパスである．

表 3:既存の方法で選択されたテストパス

表2のNo	テストパス
1	1－2－5－7－8
11	1－3－5－6－8
8	1－3－4－7－8
16	1－2－4－6－8

表 4:重要なテストパス

No	表2のNo	テストパス
1	2	1－3－5－7－8
2	1	1－2－5－7－8
3	11	1－3－5－6－8
4	10	1－2－5－6－8

3. 既存の方法の分析

表 3 の選択されたテストパスが必要十分であるか分析を行う．

3.1. 重要なパスの抜け(適用例において)

通常のユーザが本適用例を使用することを想定すると、ある入力(ここでは x)を与えて、ある出力(write y,control (y))を受ける．この入力の変化で、通るパスが変化し、結果が大きく変わる．つまり、入力の条件により、プログラムが持っている機能のどの機能を使うのかを決めていることになる．ここでの機能とは、入力から出力までデータ的関係でつながったパスのことを意味している．よって、すべての機能をテストするということを考えると、入力から出力までデータ的関係でつながっているパスをすべて選択しなければいけないこと意味している．本研究では、入力から出力までデータ的関係でつながっているパスを重要なパスと考える．

適用例での重要なパスは表 4 のテストパスである．そこで、表 3 と表 4 を見比べると表 4 の No1,4 のテストパスが選択されていないことが分かる．

3.2. 既存の方法で選択できないテストパス

既存の方法では手順 3 でデータフローパスを抽出するにもかかわらず、テストパス選択のときには 2 変数間の関係にしか注目していない．つまり、テストパス選択の際にはデータフローパス(データの定義と参照つながり)のことは考えないことになる．よって、入力から出力までデータ的関係でつながったデータフローパスを基本的にすべて抽出できないことが分かる．

このことは、2 変数間で起こるバグは発見可能であるが、3 変数間、4 変数間で起こるバグは発見不可能であることを示している．

4. 分析に基づいた既存の方法の改良

分析を踏まえて、出力がデータ的関係でつながったデータフローパスをすべて抽出でき、かつすべての 2 変数間定義参照関係が必ず 1 回以上実行される方法を提案する．初めに概要、手順を述べ、最後に適用例を示す．

4.1. 概要

4.1.1. 分析結果について

既存の方法ではすべての 2 変数間定義参照関係が実行されることをパス選択の終了条件としている．そのため、出力からデータ的関係でつながっているすべてのデータフローパスが抽出されない場合がある．

4.1.2. パス選択終了条件について

そこで本研究では、データフローパスが実行されることをパス選択の終了条件とし、出力からデータ的関係でつながっているすべてのデータフローパスを含むテストパスが選択できるように改良した方法を提案する．

4.2. パス選択の手順

既存の方法の 2 変数間定義参照関係を利用する．次にパス選択の手順を示す．改良部分は手順 4 からである．

手順 1:仕様と照合する出力を決める．

手順 2:すべての 2 変数間定義参照関係を列挙する．

手順 3:手順 2 で構成されるデータフローパスのうち手順 1 を含むものを抽出する．

手順 4-1:多くのデータフローパスを含むものを手順 3 から選択する．ただし、選択するデータフローパスは右端がデータ的関係に限る．

手順 4-2:手順 4-1 に含まれるデータフローパスを手順 3 で列挙したものにテスト済みの印をつけ手順 4-1 の選択候補から外す．

手順 4-3:選択候補がなくなるまで手順 4-1~2 を繰返す．

手順 5-1:右端が制御的関係であるデータフローパスが残っていれば 2 変数間定義参照関係に戻し列挙する．

手順 5-2: 2 変数間定義参照関係を多く含むデータフローパスを選びそれを順序とするテストパスを作る．

手順 5-3:手順 5-2 に含まれる 2 変数間定義参照関係を手順 5-1 から探しテスト済みの印をつける．すべての 2 変数間定義参照関係に印がついたデータフローパスは手順 5-2 の選択候補から外す．

手順 5-4:選択候補がなくなるまで手順 5-2~3 を繰返す．上記手順によりすべての 2 変数間定義参照関係は実行される．

4.3. 適用例

上記手順を図 2 のプログラム例 2 に適用する．表 5 が 2 変数間定義参照関係の一覧表である．表 6 が抽出されたデータフローパスの一覧表である．表 7 の上部が選択されたテストパスである．

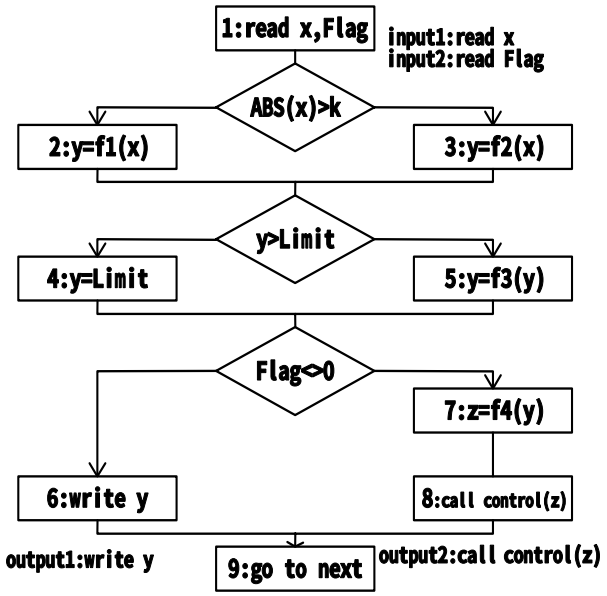

図 2　プログラム例 2

表 5: 2 変数間定義参照関係の一覧表

No	2変数間定義参照関係
1	output2[8]←z[7]
2	output1[6]⇐Flag[1]
3	output1[6]←y[4]
4	output1[6]←y[5]
5	z[7]⇐Flag[1]
6	z[7]←y[4]
7	z[7]←y[5]
8	y[5]⇐y[3]
9	y[5]⇐y[2]
10	y[5]←y[3]
11	y[5]←y[2]
12	y[4]⇐y[3]
13	y[4]⇐y[2]
13	y[3]⇐x[1]
14	y[3]←x[1]
15	y[2]⇐x[1]
16	y[2]←x[1]
17	x[1]←input1
18	Flag[1]←input2
20	Flag[1]←input2

表 6:データフローパスの一覧表

No	データフローパス
1	output2[8]←z[7]←y[5]←y[3]←x[1]←input1
2	output2[8]←z[7]←y[5]←y[3]⇐x[1]
3	output2[8]←z[7]←y[5]⇐y[3]
4	output2[8]←z[7]←y[5]←y[2]←x[1]←input1
5	output2[8]←z[7]←y[5]←y[2]⇐x[1]
6	output2[8]←z[7]←y[5]⇐y[2]
7	output2[8]←z[7]←y[4]⇐y[3]
8	output2[8]←z[7]←y[4]⇐y[2]
9	output1[6]←y[5]←y[3]←x[1]←input1
10	output1[6]←y[5]←y[3]⇐x[1]
11	output1[6]←y[5]⇐y[3]
12	output1[6]←y[5]←y[2]←x[1]←input1
13	output1[6]←y[5]←y[2]⇐x[1]
14	output1[6]←y[5]⇐y[2]
15	output1[6]←y[4]⇐y[2]
16	output1[6]←y[4]⇐y[3]
17	output1[6]⇐Flag[1]

5. 提案する方法の検証

4.3 の適用例を用い検証を行う．表 8 はデータ的関係でつながっているデータフローパスである．そのデータフローパスをテストパスとして表したものが表 9 である．表 7 の既存の方法により選択されたテストパスは表 9 のすべてのテストパスを含んでいない．しかし､提案する方法はすべてのテストパスを含んでいる．このことにより、データ的関係でつながったすべてのデータフローパスを含むテストパスが抽出できることが示せた．

表 7：テストパスの一覧表

表6のNo	テストパス	使用した方法
1	1－3－5－7－8－9	提案
4	1－2－5－7－8－9	提案
9	1－3－5－6－9	提案
12	1－2－5－6－9	提案
7	1－3－4－7－8－9	提案
15	1－2－4－6－9	提案
1	1－3－5－7－8－9	既存
12	1－2－5－6－9	既存
7	1－3－4－7－8－9	既存
15	1－2－4－6－9	既存

表 8：データ的関係でつながっているデータフローパス

No	データフローパス
1	output2[8]←z[7]←y[4]
2	output1[6]←y[4]
3	output2[8]←z[7]←y[5]←y[3]←x[1]←input1
4	output2[8]←z[7]←y[5]←y[2]←x[1]←input1
5	output1[6]←y[5]←y[3]←x[1]←input1
6	output1[6]←y[5]←y[2]←x[1]←input1

表 9：表 8 で構成されるテストパス

No	テストパス
1	1－2,3－4－7－8－9
2	1－2,3－4－6－9
3	1－3－5－7－8－9
4	1－2－5－7－8－9
5	1－3－5－6－9
6	1－2－5－6－9

6. あとがき

パス選択の手順を改良することにより重要なテストパスの抜けが防止できた．本論分の適用例は小規模なプログラムであったが、提案する方法は一般性を保ったままである．よって大規模なプログラムにも適用可能である．また、既存の方法に比べ提案する方法ではパス数が増加した．しかし増加した分のパスでソフトウェアの信頼性を高めることができる．

今後の課題としては、自動パス選択ツールを作成しソフトウェア開発現場での適用があげられる．

謝　辞

本研究を進めるにあたり､貴重な助言を頂きました電気通信大学西研究室の尾江等士氏,中田貴章氏,高繁光徳氏に感謝いたします．

文　献

[1] Boris Beizer：”Software Testing Techniques,Second Edition”，日経ＢＰ出版センター(1994).

[2] Cem Kaner,Jack Falk,Hung Quoc Nguyen:”Testing Computer Software,Second Edition”，日経ＢＰ社(2001).

[3] 中條 武志,山口 一郎,久米 均:”定義参照関係の連鎖に基づくテストパス決定法(テスト条件の選択と見逃しの型)”，電子情報通信学会論文誌 Vol.J75-D-I No.6，pp339-348(1992).