データフローパステスト法の分析と改良

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
河野 哲也

ソフトウェアテストシンポジウム2005　2005年1月24日(月)

目次

- 研究の背景
- 研究の目的
- 既存の方法
- 既存の方法の分析
- 提案する方法
- 提案する方法の検証
- まとめ

研究の背景

- プログラム中のバグが社会的な問題にいたるとは今日珍しい出来事ではない
- ソフトウェア開発において品質を作りこむ重要な工程はテストである

⇩

ソフトウェアテストが重要

研究の背景

結合テスト，システムテストで
つまらないバグが頻発

⇩

きちんとした結合テスト，システムテストができない

研究の背景

つまらないバグとは？

⇩

普通に単体テストをやっていれば見つかるバグ

- ちゃんと単体テストやってるの・・・
- そもそもテストはやってるの・・・
- あいつが作ったものはバグが多い・・・

研究の背景

プログラマは自分の作ったソースコードを自分でテストを行っている

⇩

テスト教育はやられているの?
テストが文化的に根付いてるの?

テストの質は各プログラマのスキルに依存している

研究の背景

よって、プログラマは何らかのテスト法が必要である

⇩

できることなら優れたテスト法を使いたい

プログラマが行うテストの代表的な手法：パステスト法

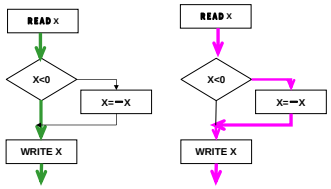
7

研究の背景

パステスト法について

- パス：プログラム中の命令を実行する順序
- パステスト：プログラムからパスを選択しテストを行う

全パステスト法：すべてのパスをもらすことなくテストを行う



条件分岐の数:n

パスの数は2のn乗になる

もし条件分岐数が5とすると

32本のパスになる

8

研究の背景

パスの数は？

- 一般的にプログラム中には莫大な数のパスが存在する

⇩

全パステスト法の実施は不可能

9

研究の背景

データフローパステスト法

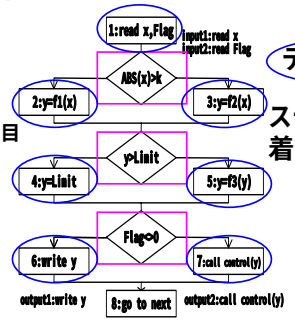
- データの流れに基づいて必要十分なパスを選択しテストする方法

10

研究の背景

制御フロー

条件分岐に着目



データフロー

ステートメントに着目

11

研究の背景

代表的データフローパステスト法

⇩

中條らが提案した方法[1]

2変数間の定義と参照の関係を連鎖と捉えパスを選択する方法

しかし、本質的に必要なパスを選択しているのか議論されていない

12

研究の目的

- 既存の方法として中條らの方法を対象とし
  どのようなパスが抜けているのか分析する
- 改良した方法を提案する

13

既存の方法

＜概要＞

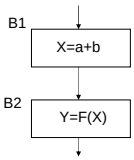
中條らの方法では2変数間定義参照関係を用いる

2変数間定義参照関係について

ブロックB2でYは定義されている

そのYはXを参照している

そのXはブロックB1で定義されている



Y[B2]←X[B1]

2変数の関係を代入と代入のための呼び出しとする

Y[B2]←X[B1]:**代入文**

14

既存の方法

＜概要＞

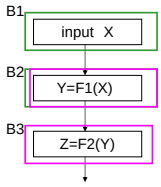
代入文を連鎖的に表したものをデータフローパスとする

データフローパスについて

Z[B3]←Y[B2] ←X [B1]

:**代入**

:**代入のための呼び出し**



15

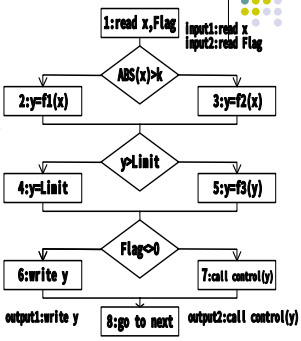
既存の方法

＜パス選択の手順＞

例を用いて説明する

この例は偏差xに応じて制御量yを計算し

ハードウェアに出力するものである



16

既存の方法

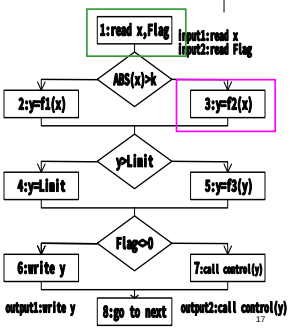
(手順1)

すべての代入文
を列挙する

:**代入**

:**代入のための呼び出し**

y[3]←x[1]



17

既存の方法

手順1により

すべての代入文が
得られる

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

18

既存の方法

(手順2)
代入文をデータフローパスにする

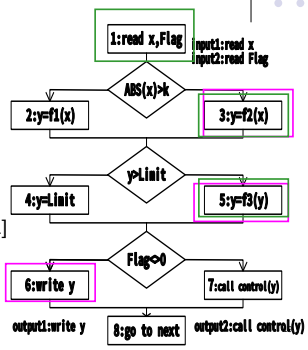
出力から行う

:代入

:代入のための呼び出し

output1[6]←y[5] ←y[3]←x[1]

すべてのデータフローパスを列挙する



既存の方法

手順2により
データフローパスが得られる

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]

20

既存の方法

(手順3)

代入文を多く含むデータフローパスを選ぶ

output1[6]←y[5]←y[3]←x[1]←input1

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]

21

既存の方法

(手順4)

output1[6]←y[5]←y[3]←x[1]←input1

これに含まれる代入文に印をつける

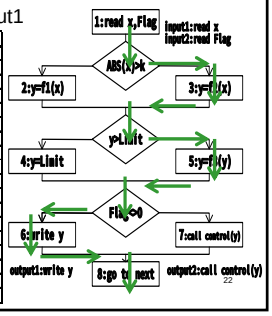
このデータフローパスの並びをパスとする

パス:1-3-5-6-8

(手順3)に戻る

データフローパスを選ぶ

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



22

既存の方法

(手順5)

代入文を多く含むデータフローパスを選ぶ(手順3)

これに含まれる代入文に印をつける(手順4)

テストパスが得られる

＊すべの代入文に印がつくまで繰り返す

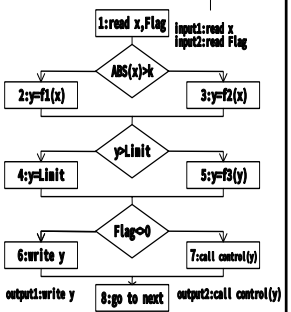
23

既存の方法の分析

通常のユーザがこの例を使用することを想定する

入力の条件によりプログラムの持っている機能のどの機能を使うのか決めている
機能とは代入文でつながっているパス

すべての機能をテストする事を考えると代入文の関係でつながっているパスをすべて選択しなければならない



24

既存の方法の分析

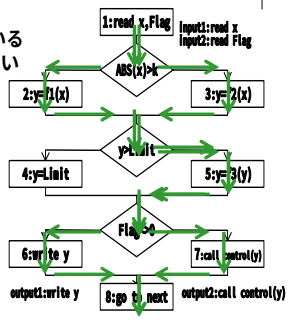
代入文の関係でつながっている
パスをすべて選択できていない

パス：1-3-5-7-8

パス：1-2-5-6-8

既存の方法で抽出したテストパス

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



既存の方法の分析

**なぜ、代入文の関係でつながっているパスを
すべて選択できないのか？**

パス選択手順に問題がある

既存の方法の分析

パス選択の問題点

(手順3) データフローパスを選ぶ

output1[6]←y[5]←y[3]←x[1]←input1

代入文にばらす

x[1]←input1　y[3]←x[1]

y[5]←y[3]　output1[6]←y[5]

テスト済みの印を
つける

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	y[1]←input1
18	Flag[1]←input2

既存の方法の分析

手順3を図上で眺める

output1[6]←y[5]←y[3]←x[1]←input1

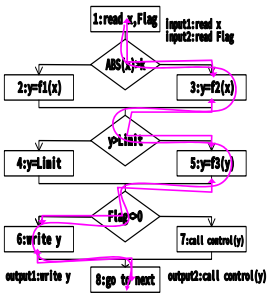
代入文にばらす

x[1]←input1　y[3]←x[1]

y[5]←y[3]　output1[6]←y[5]

すべての代入文がテストされる

代入文の組み合わせがテストされたか分からない



既存の方法の分析

代入文をすべてテストすることを

パス選択終了条件としていることが問題

代入文の組み合わせをテストすることが重要である

組み合わせにより爆発的な
パスの数になるのでは？

代入文の組み合わせの数とは
プログラムが持っている機能の数である
機能数が爆発的なものは
ありえない

提案する方法

＜概要＞

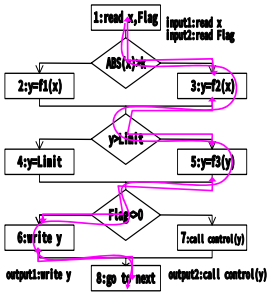
**代入文の組み合わせをテストすることを
パス選択の終了条件する**

提案する方法

＜概要＞

既存の方法では代入文がテストされるかに注目している

提案する方法では代入文のつながりがテストされたかに注目する

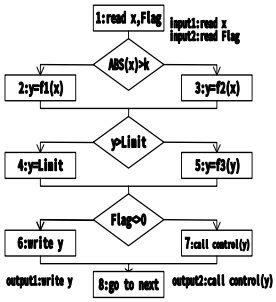


提案する方法

＜パス選択の手順＞

次の例を用いて説明する

既存の方法のデータフロー抽出までは同じとする



提案する方法

代入文とデータフローパスが得られる

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]

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

提案する方法

(手順1)

output2[7]←y[5]←y[2]←x[1]←input1

これに含まれるデータフローパスに印をつける

ここで既存の方法では代入文に印をつけたが提案する方法ではデータフローパスに印をつける

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]

提案する方法

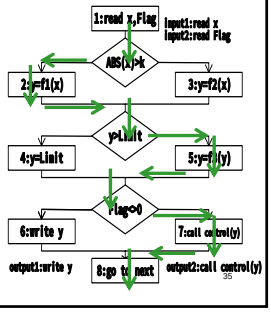
(手順2)

output2[7]←y[5]←y[2]←x[1]←input1

このデータフローパスの並びをテストパスとする

パス：1-2-5-7-8

(手順1)に戻る
出力から入力までつながったデータフローパスを選ぶ



提案する方法

選択候補がなくなるまで手順1～手順2を繰り返す

次のテストパスが得られる

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

提案する方法

印のついたデータフローパスは次のようになる

印のついていないデータフローパスが残る

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]

37

提案する方法

(手順3)

印のついていないものは
それに含まれる代入文がすべて実行されるようにする

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]

38

提案する方法

以上より、抽出されたテストパス

表2のNo	テストパス
1	1－2－5－7－8
11	1－3－5－6－8
2	1－3－5－7－8
10	1－2－5－6－8
7	1－2－4－7－8
17	1－3－4－6－8

39

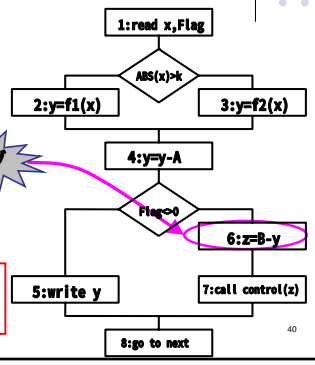
提案する方法の検証

バグを作りそれをパスとして発見できるか検証する

このバグは

2-4-6の組み合わせでは発見できない

3-4-6の組み合わせでは発見できる



よってパス1-3-4-6-7-8を抽出しなければならない

40

提案する方法の検証

既存の方法と提案する方法のパスを比較する

パス1-3-4-6-7-8

既存の方法

No	テストパス
1	1－3－4－5－8
2	1－2－4－6－7－8

提案する方法

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

重要なパスの抜けを防止できることが示せた

41

まとめ

＜考察＞

代入文の関係でつながっているパスをすべて抽出した

3変数間・4変数間で起こるバグが発見可能になった

＊既存の方法では2変数間で起こるバグのみ

42

まとめ

＜考察＞

既存の方法と比較するとパスが増えた

増えたパスは本来抽出しなければいけない
パスである

また、むやみにパスを増やしたわけではない

まとめ

＜結論＞

- パス選択の手順を改良した

重要なテストパスの抜けが防止できた

- パス選択手順の一般性は保ったままである

大規模なプログラムに適用可能

まとめ

＜今後の課題＞

自動パス選択ツールを作成し
提案する方法のソフトウェア開発現場での適用

ご清聴ありがとうございました