

プログラムモデルに基づくデータフローテスト基準の比較

廣 田 豊 彦†

一般にプログラム中には無限個のパスが存在するので、そのすべてをテストすることは不可能である。そこで、データフロー情報に基づいてテストすべきパスを選択するのが、データフローテストである。具体的なテスト基準としては、単純に定義と使用の対を用いるもの、述語使用に着目するもの、複数の変数の組合せに着目するものなど、さまざまなものが提案されてきている。これらのテスト基準を比較する手法として、従来、テスト基準の理論的な包含関係が用いられてきた。本論文では、具体的なプログラム上において、さまざまなデータフローテスト基準がどのような効果をもつのかを明らかにすることを目的として、データフローテストのためのプログラムモデルを導入した。このモデルは、順次、*if-then*、*if-then-else*、*while-do*、*repeat-until* の五つの基本モデルと、それを組み込むプログラム全体のモデルからなる。本論文では、五つの代表的なデータフローテスト基準、すなわち、定義使用対、基本データコンテキスト、全述語使用、全使用、要求対を取り上げ、プログラムモデル上でそれらのテスト基準を満足するためのパスについて議論することによって、それぞれのテスト基準の特徴を明らかにすることができた。

A Comparison of Data Flow Testing Criteria Based on a Program Model

TOYOHICO HIROTA†

Because there are infinite number of paths in a program in general, we cannot test all of them. Data flow testing is a method to select testing paths based on the data flow information. There have been proposed various testing criteria, one to use simply definition-use pairs, one to deal with predicate uses, one to deal with the combination of variables, and so on. To compare the testing criteria, the theoretical inclusion relation among the criteria has been used. In this paper, we intend to investigate the effects of various data flow testing criteria upon a concrete program. For that purpose, we have introduced a program model for data flow testing. Our model consists of five prime models which are *sequence*, *if-then*, *if-then-else*, *while-do* and *repeat-until* and a model of a whole program to embed the prime models in. In this paper, we deal with five representative data flow testing criteria: 2-dr interaction, elementary data context, all-p-uses, all-uses and required pairs. We have clarified the feature of each criterion by counting the paths necessary to satisfy the criterion on our model.

1. はじめに

プログラムの信頼性を高めるためにさまざまな工夫がなされるが、その最も基本となるのがテストである。無計画にデバッグの手助けとして実施されるようなテストはあまり信頼性の向上には貢献しないが、事前にテストを設計してプログラムを開発することは、テストを実施すること以上にプログラムの信頼性に貢献するといわれている²⁾。

プログラムテストは、大きく機能テストと構造テストとに二分される。機能テストは対象プログラムを

ブラックボックスとみなして、ある入力を与えたときに、仕様どおりの出力が得られるかどうかを検証する。それに対して構造テストは、対象プログラムの内部構造に基づいて設計され、実施される。

構造テストの代表的なものは、1) 全文テスト、2) 全分岐テスト、3) 全パステストの三つである。すべての文を実行するようにテストケースを設計することは容易ではないが、実際にテストが実行されたときに、すべての文が実行されたかどうかを測定することは可能である。Unix の *tcov* コマンド¹⁶⁾を使うと、CまたはFortranで書かれたプログラムに対して、全文テストを実行したかどうかを確認することができる。全分岐テストは全テストを包含し、さらに新たな文を含まない分岐のテストを要求する。たとえば *else* のない *if* 文に対しては、全文テストでは条件が偽と

† 九州工業大学情報工学部知能情報工学科
Department of Artificial Intelligence, Faculty of
Computer Science and Systems Engineering,
Kyushu Institute of Technology

なる場合をテストする必要はないが、全分岐テストでは条件の真と偽がともにテストされる。

全文テストや全分岐テストでは特定のパスしかテストされないのが、重大なバグが未発見のまま残される可能性が大きい。しかし実行可能なパスの数は膨大な数（場合によっては無限）になることが多く、全パステストは事実上不可能である。そこでループの実行回数を制限するなどの手法²⁾が用いられるが、データフロー情報に基づいてテストすべき実行パスを減らそうとするのがデータフローテストである。

データフローテスト基準の最初のもは Herman⁶⁾によって提案された。これは単純に変数の定義と使用の対をテストするというものである。しかし、この方針は全分岐テストを包含しないことが知られている。その欠点を克服するために、Rapps ら¹⁴⁾は述語使用を導入し、新たなテスト基準を提案した。さらに Ntafos¹¹⁾は繰返しに関してより充実したテストを行う要求対テスト基準を提案した。一方、Laski ら¹⁰⁾は、一つの文に現れる定義使用対をまとめて基本データ・コンテキストとよび、それに基づくテスト基準を提案した。

これらのさまざまなデータフローテスト基準を相互に比較する研究^{4), 12)~15), 17)}も行われている。テスト基準の比較はある基準が理論的に別の基準を包含するかどうかに基づいて行われるが、比較できない場合が多く、半順序関係となる。また、他の基準を包含するようなテスト基準はより多くのテストパスを必要とすることになることから、包含関係のみからテスト基準の優位性を判定することはできない。

本論文では、具体的なプログラム上において、さまざまなデータフローテスト基準がどのような意味をもつのかを明らかにすることを目的として、データフローテストのためのプログラムモデルを導入する。このモデルは、1) プログラム中のある一つの文に着目し、2) 定義・使用変数を最小化している。これらの条件によって、データフローテスト基準を評価するという目的に対して十分な一般性をもち、しかも可能な限り簡単なモデルを構築することができる。

このプログラムモデル上でそれぞれのデータフローテスト基準を満足するためには、どのようなパスをテストしなければならないかについて議論する。本論文のプログラムモデルは、ある一つの文に着目することから、それぞれのテスト基準について、具体的にどのようなパスが必要であることを明らかにすることができ

る。さらに、プログラムモデル上で、それぞれのテスト基準を満たすために必要なパス数が計算されるので、それに基づいてテスト基準の相互の全順序関係を決定することができる。

以下、2章でプログラムモデルについて述べ、3章でそれぞれのデータフローテスト基準について、4章でテスト基準相互の比較について議論する。

2. プログラムモデル

データフローテストは、プログラムの構造だけではなく、変数の出現についても考慮しなければならない。そこでプログラムモデルも構造と変数の両面から考えることにする。

プログラムの基本構造は、順次、選択、反復の三つである。選択としては *if-then-else* を、反復としては *while-do* を考える。さらに、選択肢に代入文があるかどうか、ループを少なくとも1回実行するかどうかで、データフローに大きな違いが出ることを考慮して、*if-then*ならびに、*repeat-until* も対象とする。

プログラム中では一般に多数の変数が使われるが、データフローは個々の変数ごとに追跡するという点を考えると、1変数だけの代入文、

$$x \leftarrow f(x)$$

だけでも意味のあるモデルとなる。しかし、基本データコンテキストに基づくテストでは、ある一つの代入文で用いられる変数の組を扱うので、そのことを考慮して、代入文の右辺にもう一つ別の変数が用いられている代入文、

$$x \leftarrow f(x, y)$$

を用いる。条件部については1変数述語、

$$p(x)$$

だけを考えるものとする。

以上のことから、図1に示すような五つの基本モデルを構築することができる。

現実のプログラムは、基本構造の連結や入れ子で構成され、多数の変数が用いられるが、それらを直接モデル化するのは容易ではない。そこでプログラムの一部分に注目することとし、プログラム全体を図2のようにモデル化する。基本部は先に定義した基本モデルのいずれかであるとし、周辺部Ⅰ、Ⅱ、Ⅲは任意の構造である。周辺部Ⅲは、プログラム中には基本部を経由しないパスも存在することを表すためのものである。

一般に、テスト基準はプログラム（あるいはモジ

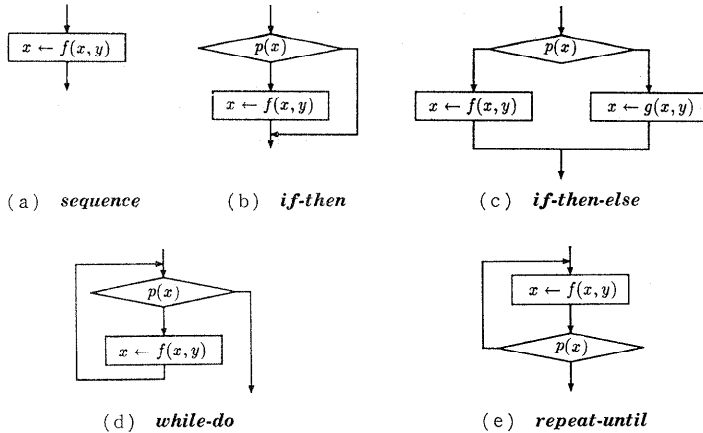


図 1 五つの基本モデル
Fig. 1 Five prime models.

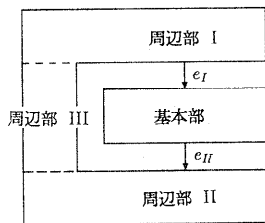


図 2 プログラム全体のモデル
Fig. 2 A model of a whole program.

ュール) 全体のテストを対象とするものであるが、ここでは周辺部の構造が不明であるので、基本部に関するテストのみを考慮する。しかし、データフローテストのためには、周辺部についても、

- 1) 周辺部 I における変数 x あるいは y の定義の中で、エッジ e_I を経由して基本部に到達する定義
- 2) 周辺部 II における変数 x の使用の中で、基本部の定義がエッジ e_{II} を経由してそこに到達するような使用

に関する情報が必要である。3章ではこのモデルを用いて、テスト基準を達成するためのパス数を数えるので、次のような値を仮定する。

$D_x=1$)に該当する変数 x の定義の数

$D_y=1$)に該当する変数 y の定義の数

$U_x=2$)に該当する変数 x の使用の数

なお、「到達する」(reach)の厳密な定義については、データフロー解析に関する文献1), 9)などを参照されたい。

このようなプログラム全体のモデルの基本部に、基

本モデル1~5を組み込むことによって、五つのプログラムモデルを構築することができる。以後、そのようなプログラムモデルを単にモデル1~5とよぶことにする。

3. データフローテスト基準

本論文では、データフローテスト基準として、定義使用対⁶⁾(2-dr interactions), 基本データコンテキスト¹⁰⁾(elementary data contexts), 全述語使用¹⁴⁾(all-p-uses), 全使用¹⁴⁾(all-uses), 要求対¹¹⁾(required pairs)を取り上げる。

以下の節でそれぞれのテスト基準について簡単に説明した後、2章で定義したプログラムモデルの基本部における変数の定義と使用に関して、テスト基準を満たすために必要なパスについて議論する。

3.1 定義使用対基準

ある変数の使用に対して、そこへ到達するすべての定義をテストするのが、文献6)で定義されている定義使用対(2-dr interactions)基準である。例として図3のプログラム断片(C言語、以下同じ)を考える。6行目で変数 y が使用されているが、ここへは2行目と4行目の両方の変数 y の定義が到達する。したがってこの基準を満たすためには、

1-2-5-6

1-3-4-5-6

の二つのパスをテストしなければならない。

モデル1の場合、基本部での変数 x, y の使用に対して、到達する定義の数がそれぞれ D_x, D_y なので、変数 x の定義使用対に対して D_x 個のパス、変数 y の定義使用対に対して D_y 個のパスが必要である。また基本部での変数 x の定義と周辺部の使用の対に対しては U_x 個のパスが必要である。これらのパスは互いに重なり合うこともあるが、最悪の場合には、

```

1  if( x > 0 )
2      y = 1;
3  else
4      y = 0;
5  if( z > 0 )
6      y += 2;

```

図 3 例題プログラム 1
Fig. 3 Example program 1.

$$D_x + D_y + U_x \quad (1)$$

となる。データフロー異常^{(3), (5), (7), (8)}が存在しないことを前提とすると、 D_x 個のパスと U_x 個のパスは、少なくとも一つは重なり合うはずであるが、ここではそこまで細かい議論は置いておく。

モデル2は条件判定があるが、条件が偽のときには何の処理もしない。つまり、定義使用対基準の立場からは、条件が真の場合だけをテストすればよい。条件部では代入文と同じ変数 x を参照するだけなので、それによってテストすべきパス数が増えることはない。したがって、必要なパス数はモデル1と同じ(1)式になる。

モデル3は、条件が真の場合と偽の場合のそれぞれに処理があり、それぞれがモデル1と同じだけのパスが必要であるので、必要なパス数は、

$$2D_x + 2D_y + 2U_x \quad (2)$$

となる。

モデル4はループを含むが、基本部と周辺部との間の定義使用対はモデル1の場合と同じであり、それぞれのパスがループを1回以上実行していればよい。ループ中の代入文の定義と、同じ代入文の使用との対をテストする必要があるが、これはループを2回以上実行することによってテストされる。これは新たなパスを増やす必要はなく、すでに考慮に入っているパスのどれかでループを2回以上実行するだけでよい。したがって、必要なパス数はモデル1と同じ(1)式になる。

モデル5では、必ずループを1回以上実行する。しかし、定義使用対基準では、上で述べたようにモデル4でもすべてのパスはループを1回以上実行するので、モデル5との違いはない。したがって、必要なパス数はモデル1と同じ(1)式となる。

3.2 基本データコンテキスト基準

文献10)で定義されている基本データコンテキスト (elementary data context) とは、あるパスを経由してある文に到達する変数定義の中で、その文で使用されている変数の定義を集めたものである。例として図

```

1  y = 0;
2  z = 0;
3  if( x > 0 )
4      y = 1;
5  if( x > 2 )
6      z = 10;
7  x = y + z;
```

図4 例題プログラム2

Fig. 4 Example program 2.

4のプログラム断片を考える。7行目では変数 y と変数 z が使用されているので、この文の基本データコンテキストは、 (y_1, z_2) , (y_1, z_6) , (y_4, z_2) , (y_4, z_6) (添字は行に対応) の四つになる。このような基本データコンテキストをすべてテストするのが基本データコンテキスト基準である。図4の7行目に関しては、

1-2-3-5-7

1-2-3-5-6-7

1-2-3-4-5-7

1-2-3-4-5-6-7

の四つのパスをテストしなければならない。これに対して、定義使用対基準であれば、うまく選べば上のうちの二つのパスだけで基準を満たすことができる。ただし、一つの文で一つの変数しか使用していない場合には、基本データコンテキスト基準は定義使用対基準と同じになる。

モデル1に基本データコンテキスト基準を適用する場合、代入文における変数 x , y の使用に対して、到達する定義のすべての組合せをテストしなければならないので、パスの数は $D_x D_y$ となる。したがって、必要なパス数は、

$$D_x D_y + U_x \quad (3)$$

である。 U_x 個のパスについても、周辺部での変数 x の使用の状況によってはパス数が増えるが、それは周辺部の構造に依存するので、ここでは考えないことにする。

モデル2, 4, 5については、3.1節で議論した定義使用対基準と同様に、モデル1と同じ(3)式になる。

モデル3では、条件が真の場合と偽の場合の2倍、すなわち、

$$2D_x D_y + 2U_x \quad (4)$$

となる。

3.3 全述語使用基準

変数が条件部で使用されているときに、その使用をそれぞれの分岐条件に割り当てたものを述語使用 (p-use) という。例として図5のプログラム断片を考える。2行目は条件部であるので、変数 x の使用は条件が真の場合と偽の場合のそれぞれに割り当てられるこ

```

1  x = m;
2  if( x > 0 )
3      y = 1;
```

図5 例題プログラム3

Fig. 5 Example program 3.

となる。文献 14) で定義されている全述語使用 (all-p-uses) 基準は、そのような述語使用のすべてに対して、それぞれすべての到達する定義をテストするものである。したがって、図 5 の例では、2 行目の *if* の条件が真の場合と偽の場合がともにテストされる。

データフロー異常が存在しないかぎり、それぞれの述語使用に対して一つ以上の到達する定義が存在するはずであり、すべての分岐が 1 度以上テストされる。したがって全述語使用基準は全分岐テストを包含する。それに対して、定義使用対基準の場合、*if-then* の基本構造では条件が偽の分岐がテストされるとはかぎらないので、全分岐テストを包含しない。ただし、全述語使用基準では、述語使用以外の使用 (計算使用) は考慮しないので、定義使用対基準を包含するわけではなく、両者は直接比較はできない¹³⁾。

モデル 1 では、条件部が存在しないので、全述語使用基準の対象となるパスは存在しないことになる。ただし、プログラム中に全く分岐がないというわけではないので^{*}、モデル 1 の基本部がまったくテストされないということではない。

モデル 2 では、*if* の条件部で変数 x が述語使用されている。条件部の真と偽のそれぞれに対して、 D_x 個の到達する定義がテストされるので、必要なパス数は、

$$2D_x \quad (5)$$

となる。

モデル 3 は、述語使用という点では、モデル 2 とまったく同じであり、必要なパス数はモデル 2 と同じ (5) 式になる。またモデル 4 では、*while* 文の条件部で変数 x が述語使用されているが、これもモデル 2 と同じであり、必要なパス数は (5) 式になる。

モデル 5 は、モデル 4 と同様のループであるが、重要な違いがある。変数 x の述語使用において到達する定義は、ループ内の代入文の定義だけである。したがって必要なパス数は 2 である。

3.4 全使用基準

文献 14) で定義されている全使用 (all-uses) 基準は、すべての述語使用 (p-use) と、それ以外のすべての使用、すなわち計算使用 (c-use) に対して、すべての到達する定義をテストするものである。例として図 6 のプログラム断片を考える。2 行目と 6 行目に変数 x の述語使用があり、これをテストするためのパスと

```

1  x = m;
2  if( x > 0 )
3      y = 1;
4  else
5      y = 0;
6  if( x % 2 )
7      y += 2;

```

図 6 例題プログラム 4
Fig. 6 Example program 4.

しては、

1-2-3-6-7

1-2-4-5-6

の二つが考えられる。一方、変数 y の計算使用が 7 行目にあり、ここへは 3 行目と 5 行目の定義が到達する。上の二つのパスでは 5 行目の定義がテストされないで、新たに、

1-2-4-5-6-7

を追加する必要がある。

モデル 1 は、述語使用がなく、計算使用のみであるので、定義使用対基準の場合と同じになる。したがって、必要なパス数は、

$$D_x + D_y + U_x \quad (6)$$

となる。

モデル 2 では、まず計算使用について考えてみると、モデル 1 と同じで、必要なパス数は (6) 式になる。述語使用については 2 方向の分岐をテストしなければならないが、分岐の一方は計算使用のテストに含まれる。分岐の他方向をテストするために必要なパス数は D_x なので、合わせて、

$$2D_x + D_y + U_x \quad (7)$$

となる。

モデル 3 では、述語使用の 2 方向の分岐は計算使用のテストに含まれることになるので、定義使用対基準の場合と同じになる。したがって、必要なパス数は、

$$2D_x + 2D_y + 2U_x \quad (8)$$

となる。

モデル 4 は、モデル 2 とほぼ同じになる。違いはループ中の変数 x の定義が再びループ中の代入文に到達することである。これはあるパスでループを 2 回実行すればテストされるので、パスを増やすことにならない。したがって、必要なパス数はモデル 2 と同じで (7) 式になる。

モデル 5 では、全述語使用のところでも言及したように、変数 x の述語使用 $p(x)$ へ到達するのは、その直前の代入文による定義のみである。したがって、述語使用に対して特別なテストパスを考える必要はない。

* もしも分岐が 0 ならば、パスは一つしかなく、データフローテスト基準を考える必要はない。

く、必要パス数はモデル1と同じ(6)式になる。なお、ループ中の定義と使用の関係は、モデル4の場合と同様に、どれかのパスでループを2回以上実行することによってテストされる。

3.5 要求対基準

文献11)で定義されている要求対(required pairs)基準では、前節の全使用基準に加えて、定義あるいは使用がループ中に出現するときに、

- 1) ただちにループを出る
- 2) ループを1回以上実行する

の2通りの場合をテストする。例として図7のプログラム断片を考える。2行目の変数 x の述語使用はループ中にあるので、述語の真偽だけではなく、ループをただちに出的場合とループを1回以上実行する場合についてテストする必要がある。3行目の変数 x の定義と使用も同様に、ループをただちに出的場合とループを1回以上実行する場合についてテストする必要がある。しかし、3行目の定義と使用については、ループをただちに出的場合は、ループを1回以上実行する場合に含まれる。なぜならば、ループを1回以上実行した場合でも、最後の変数の定義あるいは使用に着目すれば、ただちにループから出たことになるからである。

表1にテストすべきすべての定義と使用の組合せを示している。表の右の欄にその行の定義と使用をテストするために必要なループの実行回数を示している。*はその回数以上であることを意味している。この表から、ループ実行回数0回、1回、4回の3通りをテストすればよいことになる。

モデル1、2、3はループを含まないので、全使用基準と同じであり、必要なパス数はそれぞれ、(6)、(7)、(8)式になる。

モデル4では、変数 x の述語使用に対しては、ループ実行回数0回、1回、2回以上の3通り、変数 x と y の計算使用に対しては、ループの実行回数1回、2回以上の2通りをテストする必要がある。またループ中の変数 x の定義とループ外の使用に対しては、ループの実行回数1回、2回以上の2通りをテストする必要がある。ループ中の定義と使用の関係に対しては

```

1  x = m;
2  while( x > 0 )
3      x--;

```

図7 例題プログラム5

Fig. 7 Example program 5.

表1 例題プログラム5の要求対
Table 1 Required pairs of example program 5.

定 義 行 ループ	使 用 行 述語 ループ	ループ 合計
1 —	2 No —	0
1 —	2 Yes 0	1
1 —	2 Yes 1*	2*
3 0*	2 No —	1*
3 0*	2 Yes 0	2*
3 0*	2 Yes 1*	3*
3 0*	3 — 0	2*
3 0*	3 — 1*	3*
3 1*	2 No —	2*
3 1*	2 Yes 0	3*
3 1*	2 Yes 1*	4*
3 1*	3 — 0	3*
3 1*	3 — 1*	4*

* はその数以上を示す。

ループを4回以上実行する必要があるが、これは他の場合を含めることができる。したがって、必要なパス数は、

$$3D_x + 2D_y + 2U_x \quad (9)$$

となる。

モデル5では、ループ実行回数0回というのは存在しないので、パス数は

$$2D_x + 2D_y + 2U_x \quad (10)$$

となる。

4. テスト基準の比較

表2に3章で取り上げたテスト基準、および全分岐テスト、全パステストに対して、それぞれのモデルにおける必要なパス数の一覧を示している。ここで、 P_I と P_{II} は、

P_I = プログラム開始から基本部までの異なるパスの数

P_{II} = 基本部からプログラム終了までの異なるパスの数

を表している。

表2のどの列も互いに異なっており、モデル1～5が、ここで議論しているテスト基準を区別するのに十分なものであることを示している。また、どの行も互いに異なっており、モデル1～5のそれぞれが、データフローテストに対して特徴的な要素をもっていることを示している。

全分岐テストとデータフローテストを比較すると、

表 2 テスト基準を満たすためのパス数
Table 2 Number of paths to satisfy the testing criteria.

モデル	定義使用対	基本データコンテキスト	全述語使用	全 使 用	要 求 対	全分岐	全パス
1	$D_x + D_y + U_x$	$D_x D_y + U_x$	0	$D_x + D_y + U_x$	$D_x + D_y + U_x$	1	$P_I + P_{II}$
2	$D_x + D_y + U_x$	$D_x D_y + U_x$	$2D_x$	$2D_x + D_y + U_x$	$2D_x + D_y + U_x$	2	$2(P_I + P_{II})$
3	$2D_x + 2D_y + 2U_x$	$2D_x D_y + 2U_x$	$2D_x$	$2D_x + 2D_y + 2U_x$	$2D_x + 2D_y + 2U_x$	2	$2(P_I + P_{II})$
4	$D_x + D_y + U_x$	$D_x D_y + U_x$	$2D_x$	$2D_x + D_y + U_x$	$3D_x + 2D_y + 2U_x$	1	∞
5	$D_x + D_y + U_x$	$D_x D_y + U_x$	2	$D_x + D_y + U_x$	$2D_x + 2D_y + 2U_x$	1	∞

データフローテストの方がはるかに詳細なテストを行っていることがわかる。モデル1に対する全述語使用基準のパス数が0となっているが、これは本論文のモデルが基本部にのみ着目したことによるものであり、プログラム全体を対象としたときには、プログラムのすべての文および分岐がテストされる。定義使用対基準は、3.3節で述べたように、モデル2において全分岐テストを包含しない。しかしこの欠陥は、*if* の条件部が偽となるようなパスを一つ追加するだけで除去できるものであり、この基準の重大な欠陥であるとは考えられない。

全パステストとデータフローテストを比較すると、ループのないモデルでは、全述語使用基準以外のデータフローテストは、全パステストとほぼ同じ程度のテストをすることになる^{*}。一方、ループを含むモデルでは、全パステストではパス数が無限大になるが、データフローテストでは、分岐だけの場合と同程度のパス数であり、現実にはテスト可能な数値であると考えられる。

データフローテスト基準相互のパス数を比較すると、

$$\begin{aligned} & \text{全述語使用} \leq \text{定義使用対} \\ & \leq \text{全使用} \leq \text{要求対} \\ & \leq \text{基本データコンテキスト} \quad (11) \end{aligned}$$

のような関係が成立する。ただし、全述語使用と定義使用対、および要求対と基本データコンテキストの関係は厳密なものではない。この中では全述語使用基準

^{*} (1)式や(3)式の値は、 $P_I + P_{II}$ を越える可能性がある。しかし、データフローはパスを通じて到達するものであり、異なるデータフローの数が異なるパスの数を越えることはありえない。つまり、データフローテストに必要なパス数は全パステストのパス数以下である。

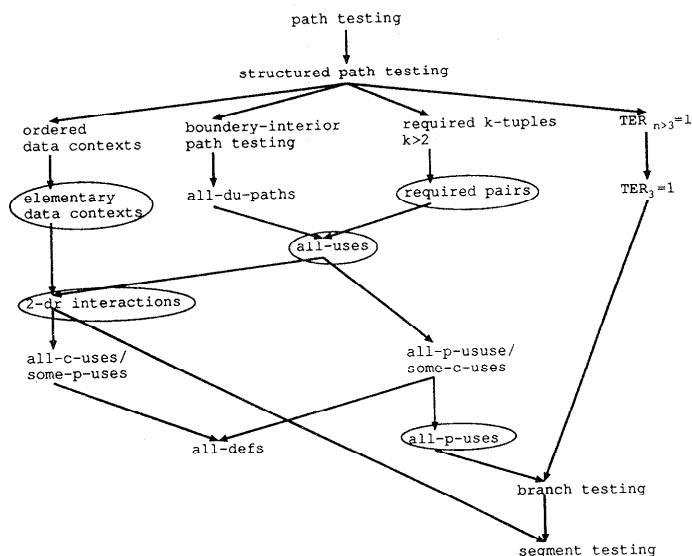


図 8 構造テスト方針の順序関係 (文献13)より引用)
Fig. 8 Partial ordering of some structural testing strategies.

がもっともパス数が少ないが、この基準は定義と使用の関係の一部しかテストしない。したがって、データフローテストの本来の目的を達成するテストの中では、定義使用対基準がもっとも少ないパスで達成可能であるということになる。

従来の比較研究^{4), 12)-15), 17)}は、任意のプログラムを対象として、テスト基準相互の完全な包含関係を議論してきた。代表的な例として、文献13)による包含関係を図8に示す。図中の楕円で囲んだ基準が、本研究で対象としたデータフローテスト基準である。この図によれば、全述語使用 (all p-uses) と定義使用対 (2-dr interactions), 要求対 (required pairs) と基本データコンテキスト (elementary data contexts), などは比較できない。それに対して本研究では、任意のプログラムの代わりにプログラムモデルを対象とすることによって、(11)式に示すような全順序関係を一応の目安として提示することができた。

5. おわりに

本論文では、代表的な五つのデータフローテスト基準の性質を明らかにすることを目的として、五つのモデルと、それを組み込んだプログラム全体のモデルを提示した。このプログラムモデルは、任意のプログラム中のある特定の文に着目したとき、その文を通過するテストパスの数をモデル化したものである。モデルは、一つの文の制御構造と変数の定義ならびに使用をモデル化したものであり、制御構造については、一般のプログラミング言語における代表的な五つの制御構造を網羅している。また、ほとんどのデータフローテスト基準では、それぞれの変数ごとに条件が設定されることから、モデルにおいて変数の定義と使用を限定していることは、モデルの一般性を失わせることにはならない。

このようなプログラムモデルを用いて、五つの代表的なデータフローテスト基準について議論を行い、それぞれのテスト基準がどのようなパスを要求するのかを、パスの種類ならびにパス数の形でより具体的に示すことができた。さらにこの議論の結果から、五つの基本モデルを組み込んだプログラムモデルが、データフローテスト基準を議論するための十分なモデルであることが明らかになった。

Weyuker¹⁸⁾ は全述語使用および全使用を満たすためのパス数の理論的上限值として、 $(d^2+4d+3)/4$ (d は分岐の数) を導き出している。本論文のモデルはそのような理論的上限を与えるものではないが、それぞれのテスト基準に対してより具体的な値を示すことができた。

今回は抽象的なプログラムモデルのみを扱ったが、現実のプログラムから D_* , U_* などの平均値を得ることによって、必要なパス数の予測ができると期待され、今後の研究課題である。

謝辞 貴重な助言をいただいた本学橋本正明教授に感謝します。

参 考 文 献

- Allen, F. E. and Cocke, J.: A Program Data Flow Analysis Procedure, *Comm. ACM*, Vol. 19, No. 3, pp. 137-147 (1976).
- Beizer, B.: *Software Testing Technique*, Van Nostrand Reinhold (1990).
- Chan, F. T. and Chen, T. Y.: AIDA—A Dynamic Data Flow Analysis Detection System for Pascal Programs, *Softw. Pract. Exper.*, Vol. 17, No. 3, pp. 305-330 (1987).
- Clarke, L. A., Podgurski, A., Richardson, D. J. and Zeil, S. J.: A Formal Evaluation of Data Flow Path Selection Criteria, *IEEE Trans. Softw. Eng.*, Vol. 15, No. 11, pp. 1318-1332 (1989).
- Fosdick, L. D. and Osterweil, L. J.: Data Flow Analysis in Software Reliability, *ACM Computing Surveys*, Vol. 8, No. 3, pp. 305-330 (1976).
- Herman, P. M.: A Data Flow Analysis Approach to Program Testing, *The Australian Computer Journal*, Vol. 8, No. 3, pp. 92-96 (1976).
- Huang, J. C.: An Approach to Program Testing, *ACM Computing Surveys*, Vol. 7, No. 3, pp. 113-128 (1975).
- Jachner, J. and Agarwal, V. K.: Data Flow Anomaly Detection, *IEEE Trans. Softw. Eng.*, Vol. SE-10, No. 4, pp. 432-437 (1984).
- Kennedy, K.: A Survey of Data Flow Analysis Techniques, In Muchnick, S. S. and Jones, N. D. (Ed.), *Program Flow Analysis: Theory and Applications*, Prentice-Hall (1981).
- Laski, J. W. and Korel, B.: A Data Flow Oriented Program Testing Strategy, *IEEE Trans. Softw. Eng.*, Vol. SE-9, No. 3, pp. 347-354 (1983).
- Ntafos, S. C.: On Required Element Testing, *IEEE Trans. Softw. Eng.*, Vol. SE-10, No. 6, pp. 795-803 (1984).
- Ntafos, S. C.: A Comparison of Some Structural Testing Strategies, *Proc. 19th HICSS*, pp. 803-811 (1986).
- Ntafos, S. C.: A Comparison of Some Structural Testing Strategies, *IEEE Trans. Softw. Eng.*, Vol. 14, No. 6, pp. 868-874 (1988).
- Rapps, S. and Weyuker, E. J.: Data Flow Analysis for Test Data Selection, *Proc. 6th Int. Conf. Softw. Eng.*, pp. 272-278 (1982).
- Rapps, S. and Weyuker, E. J.: Selecting Software Test Data Using Data Flow Information, *IEEE Trans. Softw. Eng.*, Vol. SE-11, No. 4, pp. 367-375 (1985).
- Sun Microsystems: *Sun OS Reference Manual*, Sun Microsystems (1988).
- Weise, M. D., Cannon, J. D. and McMullin, P. R.: Comparison of Structural Test Coverage Metrics, *IEEE Software*, Vol. 2, No. 2, pp. 80-85 (1985).
- Weyuker, E. J.: The Complexity of Data Flow Criteria for Test Data Selection, *Inf. Process. Lett.*, Vol. 19, No. 2, pp. 103-109 (1984).

(平成5年1月7日受付)

(平成5年9月8日採録)

**廣田 豊彦** (正会員)

1954 年生. 1976 年京都大学工学部電気工学第二学科卒業. 1981 年同大学大学院工学研究科博士後期課程研究指導認定退学. 工学博士. 1981 年京都大学情報処理教育センター助手. 1988 年九州工業大学情報科学センター助教授. 1989 年同大学情報工学部知能情報工学科助教授, 現在に至る. プログラム開発環境, プログラムテスト支援, CAD システムなどの研究に従事. 著書「Cプログラミングの基礎」(培風館) ほか. 日本ソフトウェア科学会, 人工知能学会各会員.
