パステストに本質的な分岐に着目した網ら率尺度の提案†

中所 武司††

ソフトウェアのテストを効率的かつ効果的に行うために，プログラム内の全分岐方向のテスト実行をめざす分岐テスト法に用いる新しい網ら率尺度を提案する．すなわち，従来の尺度では，すべての分岐方向を対等に扱っているため，品質が過大評価されたり，冗長なテストケースが選択されやすいという欠点があった．そこで，われわれは，まず，パス網ら性に本質的な分岐を意味する原始アークと，そうでない分岐を意味する相続子アークの概念を導入した．そして，被テストプログラムの制御フローグラフから，原始アークのみから成る相続子簡約グラフを求めるアルゴリズムを導出するとともに，本アルゴリズムがアーク数最小の相続子簡約グラフを導くことを示した．さらに，パス網ら性をこの相続子簡約グラフ上で考える方式が，従来方式よりも品質評価基準に適していることを実験的に確かめた．

1. まえがき

ソフトウェアプログラムのテストとデバッグは，その開発費用の半分以上を占め，生産性向上に重要であるばかりでなく，ソフトウェアの信頼性向上のための重要課題である．とくに最近のマイクロコンピュータをはじめとする計算機応用分野の広がりに伴い，ソフトウェアの品質保証に関する要求が高まっている[1]．

これまでテスト支援ツールとして種々のものが開発，使用されてきたが，最近注目されているものとして，パス網ら率測定ツールがある．これは，プログラム内のすべての分岐方向に着目し，準備したテストデータをすべて実行したときに被テストプログラム内の分岐方向がどの程度実行されたかを測定するものである．この網ら率は，テストの十分性を推定したり，未実行の分岐方向を検出して，そこを通過するテストデータを追加することに使用される．

このような手法は，E. F. Miller[2] によって提案され，Fortran 用の RXVP[3]，SADT[4]，ATA[5]，Cobol 用の CIP[6]，SMOTL[7] などのテスト支援ツールで実用化されている．

しかしながら，従来のようにプログラム内のすべての分岐方向に着目するような尺度は，

（1） 品質評価に用いた場合，実際より過大な評価値になる

（2） テストデータ選択に用いた場合，冗長なデータが混入しやすい

などの問題がある．そこで，われわれは，すべての分岐方向のなかでとくに網ら率に本質的なものにだけ着目することにより，これらの問題を改善する新しい尺度を考案した．本文では，その技法と品質評価への応用について述べる．

† Coverage Measure Based on Essential Branches for Path Testing by TAKESHI CHUSHO (Systems Development Laboratory, Hitachi Ltd.).

†† 日立製作所システム開発研究所

2. 従来方式

2.1 分岐テスト法

通常，ソフトウェアのテストには，幾つかの入力データをテストデータとして与え，被テストプログラムを実行後，その結果を確認するという動的テスト法がとられる．この方法では，すべての可能な入力データについてテストすることは不可能なので，限られた時間と費用のなかで高い品質保証の与えられるテストデータを選択する必要がある．

その一手法としてパステスト法がある．これは，被テストプログラムの制御構造を有向グラフで表現し，その入口から出口に至るすべての実行可能なパスのうちのできるだけ多くをテストしようとするものである．この場合の網ら率尺度 C_{path} は次式になる．

$$C_{path}=\frac{\text{実行済みのパスの数}}{\text{実行可能なパスの数}}$$

ところが，実用的なプログラムの多くは繰り返し処理を含み，実行可能なパスの数が膨大になるため，この尺度は現実的でない．そこで，実際には，パスの構成要素である分岐点から分岐点までの部分パスに着目した次のような尺度が用いられる．なお，この部分パスを dd パス (decision-to-decision path) と呼ぶ．

$$C_{dd}=\frac{\text{実行済みの dd パスの数}}{\text{dd パスの総数}}$$

これは，被テストプログラム内のすべての分岐方向のテスト実行を意図することから分岐テスト法と呼ば

れ，この尺度はおもに次のような用途を有する．

（1） テスト十分性の指標として用いられ，この尺度が高いほど被テストプログラムの品質も高いとみなす．

（2） テストデータの漏れの検出に用いられ，未実行 dd パスを通過するようなテストデータを作成する．

2.2 従来方式の問題点

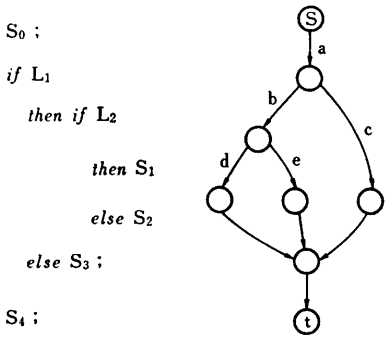
ここで，C_{dd} の性質をみるため，**図1**のプログラムを考え，次のようなテストケースを選んだとする．

ケース 1: L_1, L_2 ともに真

ケース 2: L_1 は真，L_2 は偽

ケース 3: L_1 が偽

このプログラムは，図1（2）の制御フローグラフに示すように5個の dd パス a, b, c, d, e を有する．まず，ケース1により，a, b, d が実行され，C_{dd} は 3/5 になる．さらに，ケース 2, 3 により，新たに e, c がおのおの実行され，C_{dd} は 4/5, 5/5 となる．しかしながら，このプログラムは入口から出口に至る3個のパスを有するため，本来の尺度 C_{path} で考える



（1） プログラム （2） 同左制御フローグラフ

図 1 品質が過大評価される例

Fig. 1 A sample program with quality overestimation by branch testing.

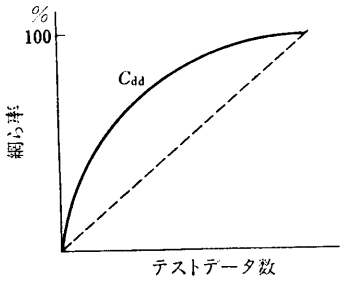


図 2 全分岐方向の網羅率曲線

Fig. 2 Coverage rate on branch testing.

と1個のテストケースを実行するごとに 1/3 ずつ網羅率が増加することが望ましい．このような差違が生じた理由は，パス網羅性に本質的でない dd パス a, b を他の c, d, e と同等の扱いにしたため，C_{dd} への各テストデータの寄与率がそれらの実行順序に依存したことによる．結局，C_{path} の代りに C_{dd} を用いた場合，**図2**に示すように，網羅率が100%に満たないところで品質が過大に評価されるという問題がある．

3. 原始アークの概念の導入

3.1 原始アーク

前章で指摘したように，dd パスには，パス網羅性という観点からみて本質的なものとそうでないものがある．そこで，本章では，それらを識別するために制御フローグラフ上で原始アークおよび相続子アークという概念を導入する．

[定義 1] プログラムを連続して実行される文の集合である基本ブロックに分割し，それをノードとする一方，基本ブロック間に制御の移行があるときはその対応するノード間を有向アークで結ぶ．ただし，プログラムの開始点と終了点はおのおの独立したノードとする*．このようにして作成した有向グラフを制御フローグラフと呼び，ノードおよびアークの集合をおのおの N, A として $G(N, A)$ で表す． （終）

なお，以降ではノードを x, y, z，x から y へのアークを (x, y) または a, b, c などで表す．

[定義 2] あるノード x について，そこへ入る入力アークの数を $IN(x)$，そこから出る出力アークの数を $OUT(x)$ と表す．そして，$IN(x)=0$ なる x を入口ノード，$OUT(x)=0$ なる x を出口ノードと呼ぶ． （終）

[定義 3] 入口ノードから出口ノードに至る任意のパスについて，もしそのパスにアーク a が含まれるならば必ずアーク b も含まれるとき，b を a の相続子またはたんに相続子と呼び，a を b の被相続子と呼ぶ． （終）

[定義 4] 他のアークの相続子になりえないアークを原始アークと呼ぶ． （終）

[定義 5] 有向グラフにおいて，そのすべてのアークが原始アークのとき，それを相続子簡約グラフと呼

* 基本ブロックと文との対応は個々の言語の有する制御文によって異なるが，本論に関係しないので略す．また，Hecht[8] らの制御フローグラフの定義ではプログラムの開始点と終了点をおのおの独立ノードにはしないため分岐のないプログラムが1ノードになるが，本論ではパスの網羅性を議論するため，独立ノードとする．

ぶ. (終)

次章以下では，従来の dd パスを基本にした分岐テスト法を改良し，原始アークに基づく方法を提案するが，その準備として，次節で原始アークに関する定理，3.3 節では制御フローグラフを相続子簡約グラフに変換するアルゴリズムについて述べる.

3.2 相続子の簡約規則

ここでは，有向グラフ上の相続子を削除するための幾つかの簡約規則を導く.

[定義 6] あるノードが複数のアークを有するとき，お互いを隣接アークと呼ぶ. また，互いに隣接アークとなる二つのアークは隣接関係にあるという（なおこの場合，アークの向きは考えない). (終)

[定理 1] 隣接関係にない二つのアーク間に相続関係があるとき，その相続関係において相続子となるアークは，その隣接アークの相続子でもある. (終)

[証明] 入口ノードから出口ノードに至るパスをそれを構成するアークの列として表すことにすると，アーク a がアーク b の相続子でありかつ隣接関係にないとき，b を通過するすべてのパスは次のいずれかで表される.

(1) $l_1, \cdots, l_i, a, m_1, \cdots, m_j, b, n_1, \cdots, n_k$ $(j \neq 0)$

(2) $l_1, \cdots, l_i, b, m_1, \cdots, m_j, a, n_1, \cdots, n_k$ $(j \neq 0)$

まず(1)の場合を考える. いま，a は m_1 の相続子でないと仮定すると，m_1 を通るパスで a を通らないパスが存在する. このパスの入口ノードから m_1 までと(1)の m_1 から出口ノードまでを結合したパスを考えると，これは b を含み，かつ，a を含まないため，a が b の相続子であるという前提に反する. したがって，a はその隣接アーク m_1 の相続子である. (2)の場合も同様. (終)

[定義 7] 同一ノードを結ぶアーク (x, x) を単ループと呼ぶ. (終)

[定理 2] 単ループは原始アークである. (終)

[証明] あるノードが単ループを有するとき，それがその隣接アークの相続子にならないことは自明である. したがって，定理1から任意のアークが単ループを相続子としえないことがいえる. (終)

[定義 8] 入口ノードからノード x に至るすべてのパスがノード y を通るとき，y を x の支配子[8] と呼び，x の支配子の集合を DOM(x) と表す. また，ノード x から出口ノードに至るすべてのパスがノード z を通るとき，z を x の逆支配子と呼び，x の逆支配子の集合を IDOM(x) で表す. (終)

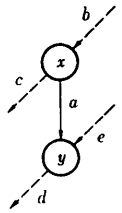


図 3 単ループでないアークの一般形
Fig. 3 General form of an arc and its two nodes.

ここで，定理2から，相続子になりうるアークは異なるノード間を結ぶものに限られるため，まず，そのようなアークとその隣接アークとの相続関係について調べる. すなわち，図3に示す一般形において，a がその隣接アーク b, c, d, e の相続子になりうる条件をおのおのの場合に分けて考える.

(1) a が b の相続子になる場合

x は出口ノードでありえないため，b を通るパスは必ず a または c を通る. そこで，b を通るパスが必ず a を通る条件は，次のいずれかである.

(i) c がない.

(ii) c があって，c を通るパスは必ず再び x に戻ること，すなわち，c の先端のノードに対して x が逆支配子である.

(2) a が c の相続子になる場合

(1)の(ii)と同様である.

(3) a が d の相続子となる場合

y は入口ノードでありえないため，d を通るパスは必ず a または e を通る. そこで，d を通るパスが必ず a を通る条件は次のいずれかである.

(i) e がない.

(ii) e があって，e を通るパスは必ずその前に y を通過していること，すなわち，e の根元のノードに対して y が支配子である.

(4) a が e の相続子になる場合

(3)の(ii)の場合と同様である.

以上の四つの条件に基づき，隣接関係にあるアーク間に関する簡約規則 R1～R4 を導入する.

[条件 1] 有向グラフ $G(N, A)$ において，$x, y \in N \wedge x \neq y \wedge (x, y) \in A$ (終)

[定義 9] 条件1において，次のような有向グラフの簡約化をノード x と y の統合という.

(1) N からノード x と y を除き，新たに z を加える.

(2) A から (x, y) を除いた後，任意のノード ω

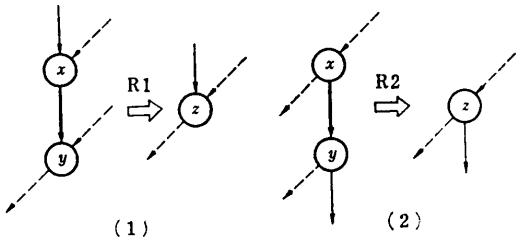


図 4 簡約規則 R1, R2 の適用
Fig. 4 Applications of R1 and R2.

について，(x,ω), (ω,x), (y,ω) あるいは (ω,y) なるアークが有れば，そのおのおのを (z,ω), (ω,z), (z,ω), (z,ω) で置き換える． (終)

[簡約規則 R1] 条件1において，

$\mathrm{IN}(x)\neq 0\wedge\mathrm{OUT}(x)=1$

ならば A から (x,y) を除き，x と y をノードに統合する（**図4**）． (終)

以降の図4および**図5**におけるアークを示す矢印のうち，太線は除かれるアーク，細線はその他の必ず存在するアーク，点線は0個以上のあってもなくてもよいアークを表す．

[簡約規則 R2] 条件1において，

$\mathrm{IN}(y)=1\wedge\mathrm{OUT}(y)\neq 0$

ならば，A から (x,y) を除き，x と y を1ノードに統合する（図4）． (終)

[簡約規則 R3] 条件1において，

$\mathrm{OUT}(x)\geq 2$,

かつ，

$x\in\mathrm{IDOM}(w)$

ただし $\forall w\in\{w|(x,w)\in A\wedge w\neq y\}$

ならば，A から (x,y) を除き，x と y を1ノードに統合する（図5）． (終)

[簡約規則 R4] 条件1において，

$\mathrm{IN}(y)\geq 2$,

かつ，

$y\in\mathrm{DOM}(w)$

ただし $\forall w\in\{w|(w,y)\in N\wedge w\neq x\}$

ならば A から (x,y) を除き，x と y を1ノードに統合する（図5）． (終)

3.3 簡約アルゴリズム

これらの簡約規則を用いて，与えられた有向グラフを相続子簡約グラフに変換するアルゴリズムを次に示し，その正当性を明らかにする．

[アルゴリズム A1] 与えられた有向グラフ $G(N,A)$ を次の手順で変換する．

(1) A に含まれる任意のアークについて，R1の条件を満たすものがあればそれを適用する．

(2) 対象となるアークがなくなるまで(1)を繰り返す．

(3) A に含まれる任意のアークについて，R2の条件を満たすものがあればそれを適用する．

(4) 対象となるアークがなくなるまで(3)を繰り返す．

(5) A に含まれる任意のアークについて，R3の条件を満たすものがあり，かつ，(x,y) 以外の x の出力アークから出発して再び x に戻るまでのパスを構成するアークのなか，あるいは x の入力アークのなかに少なくとも一つの相続子印のないアークが存在すれば，(x,y) に相続子印をつける．

(6) 対象となるアークがなくなるまで(5)を繰り返す．

(7) A に含まれる任意のアークについて，R4の条件を満たすものがあり，かつ，(x,y) 以外の y の入力アークから出発して逆方向に再び y に戻るまでのパスを構成するアークのなか，あるいは y の出力アークのなかに少なくとも一つの相続子印のないアークが存在すれば，(x,y) に相続子印をつける．

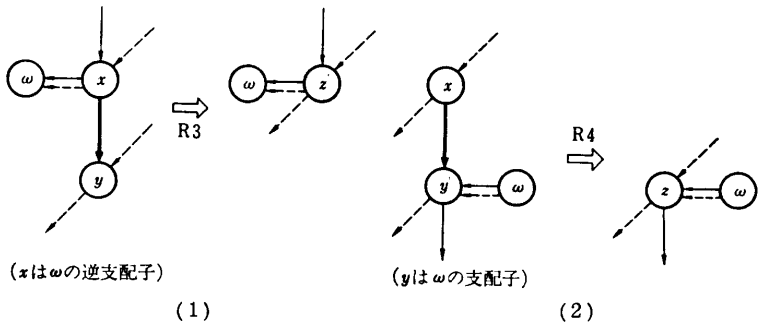


図 5 簡約規則 R3, R4 の適用
Fig. 5 Applications of R3 and R4.

（8） 対象となるアークがなくなるまで(7)を繰り返す.

（9） A に含まれる任意のアークについて，相続子印があれば，それを A から除き，両端のノードを1ノードに統合する.

(10) 対象となるアークがなくなるまで(9)を繰り返す. (終)

次に，本アルゴリズムによって導かれる有向グラフ上で，全分岐方向を網らするパス集合は，もとのグラフ上でも全分岐方向を網らすること，および，そのような性質を有するグラフのなかでアーク数が最小のものであることを，いくつかの補助定理を用いて示す.

[定理 3] R1 または R2 の適用によって，新たなパスが生じたり，消滅することはない. (終)

[証明] 図4から自明.

[定理 4] R3 または R4 の適用によって，適用前に存在したパスが消滅することはない. (終)

[証明] 図5より自明.

[定理 5] 簡約規則にしたがって相続子を除いたとき，新たに相続関係が発生することはない. (終)

[証明] いま，(x, y) を除く前に，(x, y) 以外のアーク a と b の間で a が b の相続子でなかったとする．この場合，b を通過して，かつ，a を通過しないパスが少なくとも一つ存在する．このパスは，定理3および定理4から簡約規則の適用後も存在するため，新たに a が b の相続子になることはない. (終)

[定理 6] 二つのアーク間に相続関係があるとき，その相続子は R1～R4 の簡約規則のいずれかを適用して除くことができる. (終)

[証明] 相続関係にある二つのアークが隣接するときは，2.2 節で述べたように簡約規則 R1～R4 のいずれかの前提条件を満たすので自明．二つのアークが隣接しない場合は，この相続関係にある二つのアークのうちの相続子となるアークは，定理1より，必ずその隣接アークの相続子にもなるので，やはり簡約規則 R1～R4 のいずれかの前提条件を満たす.

[定理 7] アルゴリズム A1 によって導かれる有向グラフの任意の二つのアークはもとのグラフ上で相続関係にない. (終)

[証明] まずアルゴリズムのステップ(2)の終了時には，R1 の条件,

$$\mathrm{IN}(x)\neq 0 \wedge \mathrm{OUT}(x)=1$$

を満たすノード x は存在しない．次にステップ(3)において，アーク (x, y) に R2 を適用後，統合されたノード z について考えると，R2 の条件,

$$\mathrm{IN}(y)=1 \wedge \mathrm{OUT}(y)\neq 0$$

から,

$$\mathrm{IN}(z)=\mathrm{IN}(x),$$

かつ,

$$\mathrm{OUT}(z)\geq \mathrm{OUT}(x)$$

である．そこで，z が R1 の条件を満たすためには,

$$\mathrm{IN}(x)\neq 0 \wedge \mathrm{OUT}(x)=1$$

でなければならず，これはステップ(3)の開始時の条件に反する．したがって，ステップ(3)で R1 の条件を満たすノードが生成されることはない．そこで，ステップ(4)終了時には，R1 または R2 の条件を満たすノードは存在しない．この時点で相続関係を有し，かつ，ステップ(10)で相続子が除かれないものは，定理6から，ステップ(5)またはステップ(7)において R3 または R4 の条件を満たし，かつそのすべての被相続子が相続子印を有するものだけである．ところが，これらの相続関係は，その被相続子がすべてステップ(9)において削除されるため，ステップ(10)終了時には存在しない．結局，ステップ(1)開始時に存在したすべての相続関係はステップ(10)終了時には存在しない. (終)

[定理 8] アルゴリズム A1 によって導かれる有向グラフは相続子簡約グラフである. (終)

[証明] 実理5および定理7より自明.

[定理 9] 任意の異なるアーク a, b, c において，a が b の相続子で，かつ，b が c の相続子ならば，a は c の相続子でもある. (終)

[証明] 相続子の定義より自明.

[定理 10] アルゴリズム A1 によって削除されたアークをもとのグラフ上で相続子とするアークが必ずアルゴリズム適用後のグラフ内に存在する. (終)

[証明] 定理9より自明.

結局，これまでの定理からテスト網ら性に関する次の定理が導かれる.

[定理 11] アルゴリズム A1 によって導かれる有向グラフは，次の性質を有する.

（1） このグラフ上で全アークを網らするパス集合は，もとのグラフ上でも全アークを網らする.

（2） このグラフは(1)の性質を有するグラフのなかで，アーク数が最小のものである. (終)

[証明] まず(1)について述べる．定理 10 より，アルゴリズム A1 によって除かれたすべてのアークは，アルゴリズム適用後のグラフ内のいずれかのアー

クを必ずもとのグラフ上で被相続子としていることがいえる．したがって，被相続子を通過するパスは必ずその相続子も通過するという相続関係から(1)がいえる．次に(2)について述べる．まず定理7より，元のグラフ上で存在したすべての相続関係の相続子はアルゴリズム A1 によって除かれるといえる（ただし，二つのアークがお互いに相手の相続子となりうるような相続関係においてはいずれか一方が除かれる）．さらに，定理5より，相続子アークを除去することにより新たな相続関係が発生しないことがいえている．以上のことから(2)がいえる． (終)

なお，一般に簡約規則は逐次的に適用できるほうが実用的なので，アルゴリズム A1 でも R1, R2 についてはそのようにした．しかしながら，R3 と R4 については適用後に新たなパスが発生して，適用前に存在した相続関係が消滅することがあるため，逐次適用にはしなかった．また，本質的ではないが，本アルゴリズムのように R2 よりも R1 を優先しておくと，

(1) 相続子簡約グラフのアークと元の有向グラフのアークとの対応が一意に決まる

(2) さらに，その対応する元の有向グラフのアークは分岐アーク（同一ノードの有する複数の出力アーク）となり，dd パスとの対応も一意に決まる

などの利点がある．

最後にアルゴリズム A1 によって導かれた簡約グラフ上のパスを元のグラフ上のパスに対応づけるアルゴリズムについて述べておく（ただし，本アルゴリズムにどのようなものを用いるかは定理 11 の第1項の証明に影響しない）．

まず，その準備として，アルゴリズム A1 を次のように変形する．

[アルゴリズム A2] アルゴリズム A1 に次の処理を加える．

(i) ステップ(1)または(3)で得られる，過渡的な簡約グラフをすべて保存する．i 番目に得られたものは G_i とする．

(ii) このときに除かれる相続子アークもすべて記憶する．i 番目に除かれたものを d_i とする．

(iii) ステップ(5)または(7)で相続子印をつけた相続子アークもすべて記憶する．j 番目に相続子印をつけたものを d_k とする．ただし，i の最大値を l として $k=l+j$ とする．

(iv) ステップ(9)，(10)における相続子印のついたアークの除去は，その印をつけた順に行うとともに，d_k を除いたときに得られる過渡的な簡約グラフ G_k をすべて保存する． (終)

このアルゴリズムを用いてパスの対応づけを行う．

[アルゴリズム A3] 与えられた有向グラフ $G_0(N, A)$ にアルゴリズム A2 を適用して簡約グラフ G_m を導いたとき，G_m 上の任意の一つのパスから，対応する G 上のパスを次の手順で求める．ただし，m はアルゴリズム A2 における k の最大値とする．

いま，G_m 上のあるパス $p^m=a_1^m, a_2^m, \cdots, a_{n_m}^m$ を考える．a_t^m はパスを構成する t 番目のアークとする．

(1) i を m とする．

(2) G_i 上でのパス P^i を構成するおのおののアークに対応する G_{i-1} 上のアークを選んでパス P^{i-1} を構成する．このとき，不連続点が生じればそこに d_i を挿入する．

(3) i が1でなければ i を $i-1$ として(1)へ行く．

(4) i が1ならば終了． (終)

4. 品質評価への応用

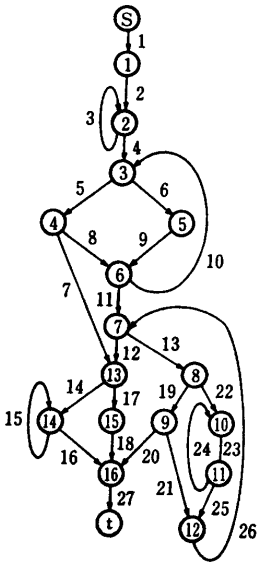
4.1 新しい網ら率尺度

すでに2章で述べたように，テスト網ら率を品質評価に用いる場合，一般に dd パスに基づく尺度 C_{dd} が使われるが，これは品質を過大に評価する傾向があった．その原因は，パステストに本質的な分岐とそうでないものを対等に扱うことにある．そこで，ここでは対象とする被テストプログラムの制御フローグラフにアルゴリズム A1 を適用して得られる相続子簡約グラフ上での網ら性を調べる次のような尺度を導入する．

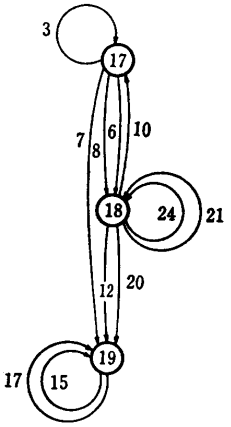
$$C_{pr}=\frac{\text{実行済みのアークの数}}{\text{アークの総数}}$$

4.2 C_{pr} の性質

まず，C_{pr} の性質を実験的に確かめるために，文献8) の図3(2)の制御フローグラフを例題（**図6**(1)）として借用する．これにアルゴリズム A1 を適用して図6(2)の相続子簡約グラフを得る．一方，(1)図のすべてのアークを一度は通過するようにテストデータを選ぶ．ここでは図(3)に示す P_1〜P_7 のテストデータをこの順に選んで実行したとすると，C_{dd} と C_{pr} は(4)図のようになり，C_{pr} のほうが C_{dd} より 2.1 節の C_{path} の線形特性（理想曲線）に近くなっている



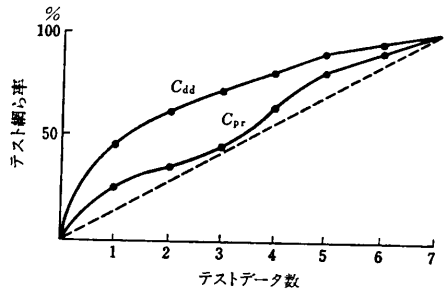
（1） 制御フローグラフ



（2） 同左の相続子簡約グラフ

パス＼アーク番号	(○印は原始アーク，分岐アークでない相続子は除去) 1 ③ 4 5 ⑥ ⑦ ⑧ ⑩ 11 ⑫ 13 14 ⑮ 16 ⑰ 19 ⑳ ㉑ 22 ㉔ 25	C_{dd} (%)	C_{pr} (%)
P_1	√ √ √ √ √ √ √ √ √	48	27
P_2	√ √ √ √ √ √ √ √	62	36
P_3	√ √ √ √ √ √ √ √ √	71	45
P_4	√ √ √ √ √ √ √ √ √	81	64
P_5	√ √ √ √ √ √ √ √ √	90	82
P_6	√ √ √ √ √ √ √ √ √	95	91
P_7	√ √ √ √ √ √ √ √ √	100	100
適用簡約規則	R4 R3R2 R4 R2R4 R3 R2 R4 R3		

（3） 分岐テストのためのテストデータのパス



（4） テスト網羅率の推移

図 6 テスト網ら率尺度 C_{dd}, C_{pr} の適用例

Fig. 6 A sample for comparison of C_{pr} with C_{dd}.

ことがわかる．

なお，(3)図にはアルゴリズム A1 によって削除されたアークに適用された簡約規則も示しておいたが，R1 により削除されるアークは，分岐アークではなく，C_{dd} との比較には無関係なので，(3)図から除いている．

次に解析的な比較を行う．いま，あるプログラムのすべての dd パスを1度以上実行するために必要なテストデータ数を n とする．ここで，このプログラムの制御フローグラフにアルゴリズム A1 を適用したときに R2～R4 によって除かれるアークを簡約分岐，その他のアークを非簡約分岐と呼び，おのおのの総数

を G, F とする．なお，R1が除くアークは分岐でないため解析対象としない．また，i 番目のテストデータで実行される非簡約分岐の数を f_i，簡約分岐の数を g_i とし，それらの重複率 α, β を次のように定義する．

$$\alpha=\left(\sum_{i=1}^{n} f_i\right)\Big/F$$

$$\beta=\left(\sum_{i=1}^{n} g_i\right)\Big/G$$

これを用いると，f_i および g_i の平均 $\bar{f}$ および $\bar{g}$ は，

$$\bar{f}=\left(\sum_{i=1}^{n} f_i\right)\Big/n=\alpha F/n$$

$$\bar{g}=\left(\sum_{i=1}^{n} g_i\right)\Big/n=\beta G/n$$

となる．このとき，平均的なテストデータを最初に一つだけ実行したときの C_{dd}, C_{pr} は，

$$C_{dd}=\frac{\bar{f}+\bar{g}}{F+G}=\frac{\alpha F+\beta G}{F+G}\cdot\frac{1}{n}$$

$$C_{pr}=\frac{\bar{f}}{F}=\frac{\alpha}{n}$$

となり，その比は，

$$\frac{C_{dd}}{C_{pr}}=1+\frac{G}{F+G}\left(\frac{\beta}{\alpha}-1\right)$$

となる．この式から $\alpha<\beta$ のときに $C_{pr}<C_{dd}$ がいえる．この $\alpha<\beta$ の条件は常に成立するわけではないが，簡約規則 R2～R4 で除かれる相続子アークは複数のパスに含まれる可能性の高いものなので，この条件が成立する場合が多いと考えられる．たとえば，図6の例では，$\alpha=1.9$，$\beta=4.0$，$F=11$，$G=10$ なので，C_{dd}/C_{pr} は計算値で1.52となり，実験値の1.75とほぼ同じである．

5. むすび

ソフトウェアのテストを効率的かつ効果的に行うために，プログラム内の全分岐方向のテスト実行をめざす分岐テストに従来から用いられていた網ら率尺度は，品質が過大に評価されたり，冗長なテストケースが選択されやすいという欠点があった．われわれは，この原因をすべての分岐方向を対等に扱うためと考え，まず，パス網ら性に本質的な分岐を意味する原始アークと，そうでない分岐を意味する相続子アークの概念を導入した．そして，被テストプログラムの制御フローグラフから，原始アークのみから成る相続子簡約グラフを求めるアルゴリズムを導出するとともに，本アルゴリズムがアーク数最小の相続子簡約グラフを導くことを示した．

さらに，パス網ら性をこの相続子簡約グラフ上で考える方式が，従来方式に比べて，

（1） 品質の過大評価傾向の減少

という利点を有することを実験と計算式の双方で示した．本方式はこのほかにも，

（2） 冗長なテストケース選択の減少

（3） 網ら率測定用収集データ量の減少

などの利点を有するが，とくにテストケース選択への応用については，現在，具体的なアルゴリズムを研究中である．

謝辞 最後に，本研究の応用について有益なご討論をいただいた日立製作所戸塚工場 黒崎徹主任技師ならびに同社大みか工場 林利弘主任技師，および日頃ご指導いただく同社システム開発研究所 渡辺坦主任研究員に感謝します．

参考文献

1) 中所武司：ソフトウェアのテスト技法，信学会誌，Vol. 64, No. 5, pp. 549-552 (1981).
2) Miller, E. F.: Program Testing: Art Meets Theory, *Computer*, Vol. 10, No. 7, pp. 42-51 (1977).
3) Huang, J. C.: Error Detection through Program Testing, in *Current Trends in Programming Methodology*, Vol. II, pp. 16-43, Prentice-Hall, New Jersey (1977).
4) Voges, U. et al.: SADAT—An Automated Testing Tool, *IEEE Trans. Softw. Eng.*, Vol. SE-6, No. 3, pp. 286-290 (1980).
5) Holthouse, M. A. and Hatch, M. J.: Experience with Automoted Testing Analysis, *Computer*, Vol. 12, No. 8, pp. 33-36 (1979).
6) Sorkowitz, A. R.: Certification Testing: A Procedure to Improve Quality of Software Testing: *ibid.*, pp. 20-24 (1979).
7) Bicevskis, J. et al.: SMOTL—A System to Costruct Samples for Data Processing Program Debugging, *IEEE Trans. Softw. Eng.*, Vol. SE-5, No. 1, pp. 60-66 (1979).
8) Hecht, M. S.: *Flow Analysis of Computer Programs*, p. 232, Elsevier North-Holland, New York (1977).

（昭和56年10月27日受付）
（昭和57年4月19日採録）