並行処理プログラムにおける共有変数のデータフローテスト基準†

古川善吾†† 有村耕治††☆ 牛島和夫††

並行処理プログラムのテスト基準として，共有変数のデータフローに基づくテスト基準（広域データフローテスト基準）を提案する．広域データフローテスト基準は，共有変数についてのデータフロー（代入と参照の対）がテストの実施において少なくとも1回実現されることを要求する．このテスト基準を満たすことによって，共有変数を用いたプロセスの間の通信における不良を発見する可能性が高い．しかしながら，デッドロックやライブロックというプロセスの間の同期における不良を発見する可能性は低い．

1. はじめに

並行処理プログラムが普及するに従い，その信頼性向上手段としてのテスト法が注目を集めている[1)-3)]．逐次処理プログラムに比較すると，並行処理プログラムは複雑であるので，逐次処理プログラムのテスト法をそのまま適用することが困難である．逐次処理プログラムのテスト法を並行処理プログラムのテストに適用すると，(1)テストすべき項目の数が多くなりすぎる，(2)並行処理プログラムの特性をテストできない，という問題がある．これらを解決する1つの方法として，並行処理プログラムにおける共有変数のデータフローに基づくテスト基準（広域データフローテスト基準）[4)]を提案し，その有効性について検討する．

並行処理プログラムの記述言語としてここではAda[5)]を取り上げる．ただし，Adaだけでなく，並行に実行されるプロセス（Adaではタスク）の間の通信手段として共有変数を使用することのできる言語で記述された並行処理プログラムのテストにも広域データフローテスト基準を適用できる．Ada並行処理プログラムでは，一般に，共有変数を使用しないことが推奨されている．それにもかかわらず，共有変数のデータフローに着目するテスト基準を考察する理由として以下の3つがある．

(1)基本的なものである：並行処理プログラムにおいて，共有変数は，並行に実行されるプロセスの間の通信手段として最も基本的なものである．そのために，共有変数に基づくテスト基準は，並行処理プログラムの特性についてのテスト十分性を評価することができる．また，共有変数に基づくテスト基準の有効性を示すことができると，共有変数の使用法に制限を加えて考察されたセマフォやモニタ，遠隔手続き呼び出しなどの同期と通信の機構に基づくテスト基準を共有変数のデータフローに着目したテスト基準を拡張して作成できる．

(2)プログラムが複雑である：共有変数を用いた並行処理プログラムは，一般に複雑である．共有変数のみを用いて同期や通信を行うプログラムでは，共有変数の名前だけでタスクの間の同期や通信を行う．そのために，プログラマがプログラムの作成において誤りを犯す可能性が高い．

(3)データフローは制御の流れと変数の利用法との2つを表現する：逐次処理プログラムにおいてデータフローに基づく各種のテスト基準が提案されている[6),7)]．データフローは，変数への値の代入を表す「定義」と式の中での変数の利用を表す「参照」を用いて，制御フローグラフの上で参照に到達する可能性のある定義を表すことができる．これによって，プログラムの中の文の実行順序に対応する制御の流れと，プログラムの中の変数の使用方法[8),9)]とを同時にデータフローが表現するものであるために，データフローに基づくテスト基準がテストすべき条件をうまく表現できる．このように，データフローは，制御の流れと変数の利用法とを表現しているために，逐次処理プログラムだけでなく，並行処理プログラムでもデータフローに基づくテスト基準を検討することは妥当である．

† The Testing Criterion on Dataflows with Common Variables for Concurrent Programs by ZENGO FURUKAWA, KOJI ARIMURA and KAZUO USHIJIMA (Department of Computer Science and Communication Engineering, Faculty of Engineering, Kyushu University).

†† 九州大学工学部情報工学科
☆ 現在 松下電器産業(株)

以下，第2章では，並行処理プログラムの特性について議論し，第3章では，共有変数のデータフローに基づくテスト基準（広域データフローテスト基準）を定義すると同時に拡張可能性について検討する．さらに，第4章では，広域データフローテスト基準の特徴について議論する．

2. 並行処理プログラム

2.1 並行処理プログラムの記法

Ada 並行処理プログラムには，複数のタスクが存在し，互いに通信や同期を行いながら処理を進めていく．これまで，並行処理プログラムにおいて通信や同期のための方法が数多く研究されてきている[10]．通信や同期の代表的な方法は，以下のとおりである．

(1)共有変数：共有変数は，複数のタスクが共有する変数である．その値の参照や代入という操作をそれぞれのタスクで行うことができる．共有変数のみを用いて同時に実行されるタスクの同期を取るためのアルゴリズムが研究されてきた[10]．しかしながら，共有変数に対するタスクでの操作を自由に行うと2.2節で述べるようにいろいろな不都合が生じる可能性が高い．そこで，操作として Test and Set 命令や Lock/Unlock 命令が考えられた．共有変数に対する操作に制限を加えて系統的にしたものが，項番(2)，(3)で述べる概念である．

(2)セマフォ：セマフォは，1つの共有変数とP命令，V命令からなり，同期を系統的に取ることができる．

(3)モニタ：同期を取るために複数のタスクが共有するデータをモニタと呼ばれる手続きに保存し，データに対する操作をモニタが提供する wait と signal とによって行う．これによって，データの管理を一元化する．

(4)メッセージ渡し：タスクの間の通信をメッセージの送信と受信によって行うもので，共有変数を用いない．共有変数の使用によって発生する不都合を防止することができる．

(5)遠隔手続き呼び出し：タスクの中で宣言されたエントリの呼び出しと，そのエントリ呼び出しを受け付ける文との間のランデブーによって通信と同期を行う．遠隔手続き呼び出しでは，エントリ呼び出しの受付文の中の実行文は逐次的に実行される．

並行処理プログラムにおけるタスクの間の通信と同期は，共有変数に対する代入と参照という操作だけで実現できる．しかしながら，いろいろ不都合を起こしやすいので項番(2)から項番(5)に述べた機構が考案されてきた．それぞれ，使い方についての自由度は減少しているが，不都合を起こす可能性を少なくすることを狙っている．

Ada では，通信や同期の機構として共有変数と遠隔手続き呼び出しとを備えている．並行処理プログラムでの不都合を少なくするために，共有変数の使用を少なくすることが推奨されている．

2.2 並行処理プログラムの不良

並行処理プログラムの誤りにはいろいろなものがある．誤りを起こす原因（不良）によって分類すると以下の分類が考えられる[3),10)]．

(1)タスク内の計算不良．タスク内の処理は，逐次的に実行される．逐次処理プログラムにおいて考えられた不良が発生する可能性がある[11)]．

(2)通信不良．並行に実行されるタスクの間でのデータの受け渡し時に発生する不良である．

(3)同期不良．並行に実行されるタスクの間の同期の取り方が間違っている．この不良によって発生する代表的な誤りには，以下のものがある．

(a)安全性の破壊．並行に実行される動作の相互排除が失敗したためにデータの一貫性が失われる．

(b)生存性の破壊．並行処理プログラムが意味のある動作を行わない誤りであり，デッドロックとも呼ばれる．並行に実行されるタスクが互いに待状態になると発生する．

(c)公平性の破壊．並行処理プログラム全体としては意味のある動作を行っているが，特定のタスクだけが待状態になったままである．ライブロックと呼ばれることがある．

2.3 並行処理プログラムのテスト法の信頼性

プログラムのテスト法の信頼性として Howden が路テスト法の信頼性については検討した[11)]．すなわち，プログラムの制御フローグラフの節点や枝の被覆率を100%にするテスト法によって発見できる不良を分類した．また，テスト法が不良に対して「信頼できる」ことを以下のように定義している．

定義1

テスト法Hはテストの集合Tを作成するとする．Hが信頼できるとは，プログラムPが正しいか，Pが

不良を持っているならば ∃t(t∈T→P(t) は誤っている)，が言えるときである．

この定義は並行処理プログラムにも適用することができる．ただし，逐次処理プログラムでは，テスト法はテストとして入力データ（テストデータ）を指定すれば十分であったけれども，並行処理プログラムでは，テストデータと同時に実行のタイミングを決めるための実行系列を指定する必要がある．

3. 広域データフローテスト基準

本章では，共有変数のデータフローに基づいた，Ada 並行処理プログラムの広域データフローテスト基準を定義する．Ada についての詳細は文献5)にある．

3.1 広域データフロー

最初に，共有変数に対する値の代入と参照とをそれぞれ「広域定義」と「広域参照」として定義する．

定義2

広域定義とは，共有変数への値の代入あるいはデータの読み込みである．広域定義 def を次の3つ組で表す．

$$\mathrm{def}=(T, V, n)$$

ただし，Tは広域定義を行うタスクまたは実行ブロックの名前，Vは共有変数名，nは広域定義を行う文の番号である．

定義3

広域参照とは，代入文の右辺あるいは判定文の式，データの書き出しにおける共有変数の参照である．広域参照 ref を次の3つ組で表す．

$$\mathrm{ref}=(T, V, n)$$

ただし，Tは広域参照を行うタスクまたは実行ブロックの名前，Vは共有変数名，nは広域定義を行う文の番号である．

同一の変数に対する広域定義と広域参照の対が「広域データフロー」である．

定義4

広域データフローとは，次の式で表される2つ組 gdf である．

$$\mathrm{gdf}=(\mathrm{def}, \mathrm{ref}) \qquad (1)$$

ただし，$\mathrm{def}=(T_i, V, m)$ で $\mathrm{ref}=(T_j, V, n)$，T_i と T_j がタスクタイプ以外のときは $i\neq j$ である．

ひとつのタスク内でのデータフローについては制御フローグラフ上で定義の到達可能性を検討しなければならない．しかしながら，タスク間の広域データフローは，異なったタスクの中の実行文の実行順序が規定されていないため同じ共有変数に対する広域定義と広域参照の対すべてについて考慮する必要がある．

また，式(1)で定義した広域データフローは1つのタスク内でのデータフローを含んでいない．なぜなら，広域データフローは並行処理プログラムの共有変数に着目しているためである．並行処理プログラムが逐次処理プログラムと異なる点は，並行処理プログラムでは共有変数をタスク間のデータの受け渡しや同期に使うことである．タスク内でのデータフローにおける不良（2.2節で述べたタスク内不良に含まれる）は，それぞれのタスクを逐次処理プログラムと考えテストを行うことによって対処できる．したがって，広域データフローは，タスク間のデータフローのみを含むものとする．

ただし，タスクタイプでは，プログラムの実行時にタスクの実体が複数個生成される可能性があるので，タスク内での広域定義と広域参照の対を広域データフローに加える．この場合も，タスクタイプの制御の流れに基づいた定義の到達可能性は考慮しない．

3.2 広域データフローテスト基準の定義

並行処理プログラムをPとし，$\boldsymbol{S}$はテストによって実現された実行系列の集合とする．組 $(P, \boldsymbol{S})$ がテスト基準Cを満足するとき，$C(P, \boldsymbol{S})$ と表す．まず，「広域データフローテスト基準」C_{GDF} を定義する．

定義5

$C_{\mathrm{GDF}}(P, \boldsymbol{S}) \Leftrightarrow$

並行処理プログラムP中のすべての広域データフロー gdf に対して，$\boldsymbol{S}$の中に広域データフロー gdf を実現する実行系列が少なくとも1つ存在する．

広域データフローテスト基準に基づくテスト十分性を定量的に評価するための「広域データフロー被覆率」を定義する．まず，テストの十分性を評価するための一般的な「被覆率」Wを定義した後に，「広域データフロー被覆率」W_{GDF} を定義する．

定義6

Pをプログラム，$\boldsymbol{Te}$をテストの集合とするとき $\boldsymbol{Te}$ によるテスト十分性を表す被覆率Wを，次式で定義する．

$$W(P, \boldsymbol{Te})=\frac{\left|\bigcup_{t\in \boldsymbol{Te}} \boldsymbol{E}_P(t)\right|}{|\boldsymbol{M}_P|} \qquad (2)$$

ただし，$\boldsymbol{M}_P$ はプログラムPに存在する測定対象となる事象の集合，$\boldsymbol{E}_P$ はテスト $t\in \boldsymbol{Te}$ に基づいてプログ

ラムPを実行したときに発生する事象の集合である．$|\cdot|$は集合の要素数を表す．

定義7

広域データフローに基づいたテスト十分性の評価値（広域データフローの被覆率）W_{GDF}は，式(2)のM_Pを広域データフローの集合GDFとしたときの被覆率である．

$$W_{GDF}(P, Te)=\frac{|\bigcup_{t\in Te} E_P(t)|}{|GDF|} \quad (3)$$

ただし，$GDF=\{gdf\}$であり，$\{\cdot\}$は集合を表す．

3.3 広域データフローテスト基準の拡張

広域データフローテスト基準の適用度を増大するために，逐次処理プログラムのテスト基準で使われるP-useとC-useの考え[12]を採り入れて，広域データフローテスト基準の拡張を行う．まず，広域参照を次のように2つの種類に分ける．

(1) P-ref：条件文での参照

P-refは，条件文で参照されているので制御フローの路選択に直接影響を与えることができる．そこで，テスト十分性を評価する場合，単に条件文を実行したか否かだけでなく，条件が成立した場合と不成立の場合とをそれぞれ実行したか否かまで評価する．

(2) C-ref：代入文の右辺，出力文での参照

C-refは，直接的には，文の中での計算だけに影響を与えるので広域データフロー被覆率を考えるときに参照のみを検討すればよい．

上で分類したC-refに基づく広域データフローテスト基準$C_{GDF\text{-}C}$は，定義5の場合と同様に考えることができるので，以下のようになる．

定義8

$C_{GDF\text{-}C}(P, S) \Leftrightarrow$

並行処理プログラムP中の広域参照refがC-refであるすべての広域データフローgdf=(def, ref)に対して，Sの中に広域データフローgdfを実現する実行系列が少なくとも1つ存在する．

一方，P-refに基づく広域データフローテスト基準$C_{GDF\text{-}P}$は，条件文を実行した後に実行する分岐についての条件を導入し，以下のように定義する．

定義9

$C_{GDF\text{-}P}(P, S) \Leftrightarrow$

並行処理プログラムP中のrefがP-refであるすべての広域データフローgdf=(def, ref)と，ref=(T, V, n)なるnのすべての後続点mとに対して，Sの中に広域データフローgdfを実現し，かつ有向枝(n, m)を実現する実行系列が少なくとも1つ存在する．

さらに，すべての広域データフローについてその後続点を実行することを要求する広域データフローテスト基準$C_{GDF\text{-}DU}$を以下のように定義する．

定義10

$C_{GDF\text{-}DU}(P, S) \Leftrightarrow$

並行処理プログラムP中のすべての広域データフローgdf=(def, ref)とref=(T, V, n)なるnのすべての後続節点mとに対して，Sの中に広域データフローgdfを実現し，かつ有向枝(n, m)を実行する実行系列が少なくとも1つ存在する．

3.4 広域データフローテスト基準の包含関係

前節までに定義したそれぞれのテスト基準についてそれぞれの包含関係を調べる．テスト基準の包含関係についてまず定義する．

定義11

テスト基準C_1がテスト基準C_2を包含しているとは，

$C_1 \rightarrow C_2$

が成立していることである．

前節までに挙げたテスト基準の包含関係を解析すると，**図1**のようになる．図1では，矢印の元のテスト基準が矢印の先のテスト基準を包含していることを表している．

広域データフローテスト基準C_{GDF}はテスト基準$C_{GDF\text{-}DU}$に包含されているが，テスト基準$C_{GDF\text{-}C}$を包含している．また，テスト基準$C_{GDF\text{-}DU}$は，テスト基

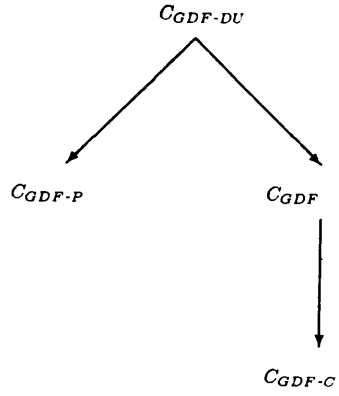

図1 広域データフローテスト基準の包含関係
Fig. 1 The dependency of the testing criteria of global dataflows.

準 $C_{GDF\text{-}P}$ を包含している．ただし，広域データフローテスト基準 C_{GDF} と，テスト基準 $C_{GDF\text{-}P}$ との間には，包含関係はない．したがって，時間的，コスト的に余裕があるテストにおいては，広域データフローテスト基準 C_{GDF} ではなくテスト基準 $C_{GDF\text{-}DU}$ を適用する．反対に，余裕がないテストのときは，テスト基準 $C_{GDF\text{-}C}$ を適用する．

4. 考　　察

4.1 広域データフローテスト基準の不良発見能力

広域データフローテスト基準に基づいて発見できる不良を検討するために，共有変数を用いた Ada の sieve.a プログラムの広域データフローを考える．このプログラムの副プログラム SIEVE とタスク型 PROCESSOR の制御フローグラフを**図 2** に示す．

このプログラムは，エントリの引き数としてデータを受け渡すプログラム[13]を共有変数で受け渡すものに変更したものである．プログラムは，入力した自然数の値以下の素数を求めるものである．素数を見つけるとタスク PROCESSOR を生成して，見つけた素数の値を保持させる．各タスクでは，新たに伝えられた値が保持している素数の倍数でないときに次のタスク PROCESSOR に値を伝達する．この値の伝達に共有変数 M を用いる．

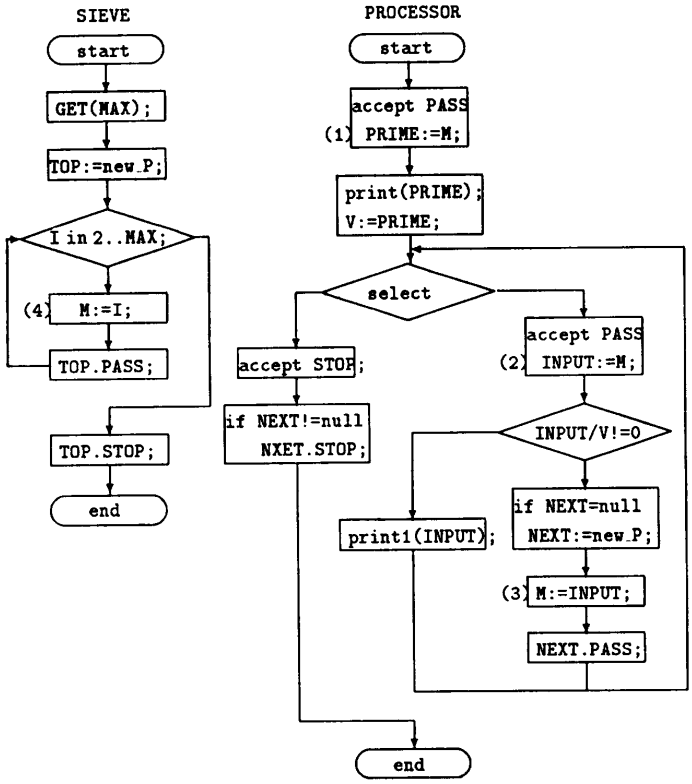

図 2 sieve.a プログラムの制御フローグラフ
Fig. 2 The control flow graph of the program sieve.a.

表 1 sieve.a プログラムの広域データフロー
Table 1 The global dataflows of the program sieve.a.

広域データフロー	広域定義	広域参照
gdf_1	(PROCESSOR, M, 3)	(PROCESSOR, M, 1)
gdf_2	(PROCESSOR, M, 3)	(PROCESSOR, M, 2)
gdf_3	(SIEVE, M, 4)	(PROCESSOR, M, 1)
gdf_4	(SIEVE, M, 4)	(PROCESSOR, M, 2)

共有変数は M だけであるので，広域定義は図 2 の (PROCESSOR, M, 3) と (SIEVE, M, 4) である．一方，広域参照は (PROCESSOR, M, 1) と (PROCESSOR, M, 2) である．文番号は関係ある部分のみに設定した．このプログラムの広域データフローは**表 1** に示す 4 つがある．

プログラム sieve.a は，広域データフロー gdf_1, gdf_2, gdf_3 の 3 つだけを実現したときには，正しく動作する．しかし，このプログラムで広域データフロー gdf_4 が実現されると誤りが発生する．

入力値として，2 および 3 を与えたときは，広域データフローは gdf_1, gdf_2, gdf_3 の 3 つだけが実現されて，正しく動作する．入力値として 4 を与えたとき，実行のタイミングによって gdf_1, gdf_2, gdf_3 の 3 つだけが実現される場合と広域データフロー gdf_4 まで実現される場合とがある．さらに，5 以上の入力を与えると必ず広域データフロー gdf_4 まで実現される．そのために，3 つだけが実現される入力 2，3 については正しく，2，3 を素数と表示する．しかしながら，広域データフロー gdf_4 まで実現される 5 以上の入力については誤りが発生して不良が存在することが分かる．

図 2 の sieve.a プログラムは，広域データフローの被覆率が 100% で必ず不良を発見することのできるプログラムである．すなわち，sieve.a プログラムに対して広域データフローテスト基準は，2.3 節で述べた信頼で

きるテスト法である.

このように，広域データフローテスト基準を満たしたときに必ず発見できる不良について次に考える．定義1から広域データフローテスト基準が信頼できるのは，プログラムが正しいかあるいは，広域データフローテスト基準を満足させるためのテスト集合の中の要素の少なくとも1つのテストによって誤りを発見できる場合である．広域データフローテスト基準は，広域データフローが少なくとも1回実現されることを要求している．そのために，広域データフローの中の1つが不良である場合には広域データフローテスト基準を満足することによって不良の存在する広域データフローを実現する．広域データフローは，共有変数を用いたタスクの間の通信を表している．そこで，広域データフローテスト基準によって2.2節で述べた通信不良を発見する可能性があることがわかる.

しかしながら，不良のある広域データフローを実現しても必ずしもプログラムにおける誤りとして顕在化する保証はない．広域データフローテスト基準を満足することによって発見できる不良は，広域データフローの不良であってそれを実現すると必ず顕在化する不良である.

2.2節で述べたタスク内不良は，3.1節でも一部述べたように，逐次処理プログラムのテスト基準を用いることによって対処できる．一方，同期不良は，並行処理プログラム内の各文の実行順序に依存して発生することが多いために，ここで述べた広域データフローテスト基準によって発見される可能性は少ない.

4.2 広域データフローテスト基準の複雑さ

次に，広域データフローテスト基準を満たすためのテストの複雑さについて検討する．テストの複雑さとして被覆率を求めるときの測定対象の事象の数を考える．並行処理プログラムのテストにおける不良の種類に応じたテストの複雑さについては，別途検討した[3]．ここでは，広域データフローテスト基準のテストの複雑さについて検討する.

広域データフローは，1つの変数についての広域定義と広域参照の対である．広域定義と広域参照は，各タスクのそれぞれの文で行われる可能性がある．そのために，並行処理プログラムのすべての実行文の数を n としたとき，1つの共有変数について n 個の広域定義と n 個の広域参照が考えられるので最悪の場合 n^2 個の広域データフローが存在する可能性がある．ゆえに，広域データフローテスト基準のテストの複雑さは，共有変数の個数を v，文の数を n としたとき，$O(v \times n^2)$ である.

表 2 並行状態と広域データフローの比較
Table 2 The number of the concurrent states and the global dataflows.

タスク数	並行状態数	広域データフロー数
3	84	4
4	238	4

Taylor は，各タスクの状態の組として並行状態を定義し，並行状態を節点とし，各タスクの状態の遷移に応じた並行状態の移行を枝とする並行状態グラフを定義した[2]．この並行状態グラフの節点や枝を測定対象とする被覆率をテスト十分性評価に用いることを提案している．このテスト基準は，並行処理プログラムの同期不良を発見できる可能性が高い[3]．しかしながら，このテスト法のテストの複雑さは組合せ論的である．実際に4.1節で述べた素数を求める並行処理 sieve.a プログラムについてテストの複雑さを調べると表2のようになる．sieve.a の広域データフロー数は常に4である．一方，Taylor の並行状態数は，あらかじめシステム内のタスク数が判っていなければ決定できない．しかし，sieve.a は入力によってタスクが動的に生成される．そこで，タスク数が3と4の場合を表には示した．タスクの数が3というのは sieve.a を実行した場合のタスクの最小数である.

4.3 通信方法に対応したテスト基準

共有変数は，2.1節で述べたように並行処理プログラムの通信方法として基本的なものである．この共有変数を用いた広域データフローテスト基準では，共有変数に対する操作としての広域定義と広域参照のみに着目している．セマフォやモニタでは，複数のタスクが共有するデータに対するアクセス方法をP命令とV命令，wait と signal，にそれぞれ制限している．そのために，広域データフローと同程度の信頼性を求めるテスト基準は，P命令とV命令の対，wait 命令と signal 命令の対，をそれぞれ考慮することによって構築することができる.

同様にメッセージ渡しについても，メッセージの送信と受信を対にして考えることによって同様のテスト基準を構築することができる．しかしながら，メッセージの送信あるいは受信が一箇所のみで行われる場合には，逐次処理プログラムの制御フローグラフの節点の被覆率を100% にすることと同じであるために，新た

なテスト基準は不要である.

Ada 並行処理プログラムでは，ランデブー通路テスト基準が提案されている[14]. Ada のランデブーは，エントリ呼び出しと accept 文によって実現される遠隔手続き呼び出しである. このランデブーに基づいてランデブー通路がエントリ呼び出し文と accept 文の対として定義されている. 広域な名前として公開されているエントリについての定義（accept 文）と参照（エントリ呼び出し）の対としてランデブー通路を見ると，広域データフローに対応している.

5. おわりに

本稿では，並行処理プログラムの共有変数に基づく，広域データフローテスト基準を提案して分析した. 並行処理プログラムのテストで関連した仕事としては，以下のものがある.

(1)その他のテスト十分性評価法：並行処理プログラムのテスト十分性評価基準として広域データフローテスト基準だけでなく，並行状態グラフの節点（並行状態）や枝を測定対象とするテスト基準が提案されている. しかしながら，この基準は現実的な時間内に実現することとが困難である[3]. また，Chang は，並行状態グラフの節点や枝の被覆率と同等のものを初め5つのテスト基準を提案している[15]. それらの中には，今回提案した広域データフローテスト基準と同様なものはない.

(2)テストケースの作成：テストケースあるいはテストデータの作成が逐次処理プログラムに対してはすでに検討されている. 機能テストのためのテストケース[16]は，機能を実現するプログラムが逐次処理であるか並行処理であるかに依存しないのでそのままテストケース作成法を利用できる. 並行処理プログラムを対象とするテストケースの作成については，片山らが検討している[17].

(3)テストの実施：並行処理プログラムでは，テストの条件が明確になったり，テストケースが作成できても，テストの条件を実現したりテストケースを実現することは，逐次処理プログラムに対するほど容易ではない. この点については，Tai が検討している[1].

本稿での提案と分析は，広域データフローテスト基準の有効性について解析的なものである. 現在，Ada 並行処理プログラムの広域データフローの被覆率測定プログラムの開発を進めている. 今後，それを完成させ，実際の Ada 並行処理プログラムで広域データフローテスト基準の有効性を検証していく予定である.

さらに，実際の並行処理プログラムでの不良の発生状況を分析し，より適切なテスト基準を構築する必要がある.

謝辞 並行処理プログラムの検証，デバッグ，テストについて議論していただいた九州大学工学部情報工学科の荒木啓二郎助教授，程京徳助教授に感謝いたします.

参考文献

1) Tai, K.C.: On Testing Concurrent Programs, *Proc. of Compsac '85*, pp. 310-317 (1985).
2) Taylor, R. N. and Kelly, C. D.: Structural Testing of Concurrent Programs, *Proc. of Workshop on Software Testing*, pp. 164-169 (1986).
3) 古川善吾，牛島和夫：並行処理プログラムに対する事象グラフを用いたテストの複雑さについて，日本ソフトウェア科学会第8回大会論文集，pp. 153-156 (1991).
4) 有村耕治，古川善吾，牛島和夫：並行プログラムにおける広域データフローテスト基準の拡張，第42回情報処理学会全国大会論文集，pp. 5-220-221 (1991).
5) 情報処理振興事業協会(編)：Ada 基準文法書，共立出版(株) (1984).
6) Rapps, S. and Weyuker, E. J.: Selecting Software Test Data Using Data Flow Information, *IEEE Trans. Softw. Eng.*, Vol. 11, No. 4, pp. 367-375 (1985).
7) Clarke, L. A.: A Formal Evaluation of Data Flow Path Selection Criteria, *IEEE Trans. Softw. Eng.*, Vol. 15, No. 11, pp. 1318-1332 (1989).
8) 古川善吾，野木兼六："定義の関係" を導入したデータフロー解析手法，第 20 回情報処理学会全国大会論文集，pp. 239-240 (1979).
9) Cheng, J.: Process Dependence Net: A Concurrent Program Representation, 日本ソフトウェア科学会第 8 回大会論文集，pp. 513-516 (1991).
10) Ben-Ari, M. (渡辺榮一 (訳)): Principles of Concurrent Programming (並行プログラミングの原理), Prentice-Hall International, Inc. (1982).
11) Howden, W. E.: Reliability of the Path Analysis Testing Strategy, *IEEE Trans. Softw. Eng.*, Vol. SE-2, pp. 208-215 (1976).
12) Weyuker, E. J.: The Cost of Data Flow Testing: An Empirical Study, *IEEE Trans. Softw. Eng.*, Vol. 16, No. 2, pp. 121-128 (1990).
13) 石畑 清，疋田輝雄：Ada プログラミング言語，

岩波書店 (1986).
14) 古川善吾，牛島和夫：ランデブー通路を用いたAda並行処理プログラムのテスト十分性評価，電子情報通信学会論文誌 D-I, Vol. J 75-D-I, No. 5, pp. 288-296 (1992).
15) Chang, C. K., Song, C.-C. and Chang, Y.-F.: INTEGRAL—An Integrated Framework for Distributed Software Validation and Verification, *Proc. Workshop on the Future Trend of Distributed Computing Systems in the 1990s*, pp. 301-310 (1988).
16) 古川善吾，野木兼六，徳永健司：AGENT：機能テストのためのテスト項目作成の一手法，情報処理学会論文誌，Vol. 25, No. 5, pp. 736-744 (1984).
17) 片山徹郎，古川善吾，牛島和夫：並行処理プログラムのテストケースについて，第43回情報処理学会全国大会論文集，pp. 5-321-322 (1991).

(平成3年10月28日受付)
(平成4年 9 月10日採録)

古川 善吾（正会員）

1952年生．1977年九州大学工学部情報工学科卒業．1977年同大学院情報工学専攻修士課程修了．同年(株)日立製作所システム開発研究所入社．1986年九州大学工学部勤務．1992年同大学情報処理教育センター助教授，現在に至る．工学博士．ソフトウェア工学，特にソフトウェアテスト法，日本語文書出力方式，に興味を持つ．電子情報通信学会，ソフトウェア科学会，ACM, IEEE CS 各会員．

有村 耕治（正会員）

1965年生．1989年九州大学工学部情報工学科卒業．1991年同大学院情報工学専攻修士課程修了．同年松下電器産業(株)に入社．技術統括室九州地区研究推進室所属．マルチメディアデータの伝送方式に興味を持つ．

牛島 和夫（正会員）

1937年生．1961年東京大学工学部応用物理学科（数理工学コース）卒業．1963年同大学院修士課程修了．同年九州大学中央計数施設勤務．1977年九州大学工学部情報工学科教授（計算機ソフトウェア講座担当），現在に至る．1990年4月から九州大学大型計算機センター長を兼務．1991年度本会九州支部長．工学博士．ソフトウェア科学会，電子情報通信学会，ACM 各会員．