並行プログラムにおける広域データフローを用いたテスト基準の提案

1R－5

有村耕治　古川善吾　牛島和夫
(九州大学工学部情報工学科)

1.はじめに

並行プログラムのテストは、逐次プログラムのテストに比べて困難である。逐次プログラムでは、同一入力に対する実行結果が同じであるが、並行プログラムでは、同じ入力に対して実行のタイミングによって実行結果が異なることが有り得るためである。本報告では、並行プログラムにおける広域データフローを導入した後、広域データフローに基づく並行プログラムのテスト基準について考察する。逐次プログラムでは、データフローを用いたテスト基準が提案されている。[Clarke 85]

広域データフローは広域変数の代入参照を調べるものである。そのために、広域データフローを用いたテスト基準は並行に実行されるプロセス間での広域変数の解釈の違いを発見するためのテスト基準である。最近の並行プログラムではプロセス間でのデータのやり取りを行うために広域変数を用いることは少なくなっている。並行処理の記述ができる言語Adaに於いても並行に実行されるタスク(プロセス)間ではentry callとaccept文によるランデブによって、データの受渡しを行うことが多い。しかし、広域変数を用いたデータの受渡しは並行プログラムにおいてのモデル化に良く使われるものであるために、広域データフローを用いたテスト基準は、並行プログラムのテスト基準として基本的なものである。また、Adaにおけるランデブする可能性のあるentry callとaccept文の対(ランデブ通路)を用いたテスト基準[古川 89]へと発展可能なテスト基準である。

2.広域データフローの形式化

広域データフローの形式的な定義を以下に行う。広域変数を用いた並行プログラムは、Adaによって記述されているとする。

定義2.1 データの広域定義とは、広域変数への値の代入あるいはデータの入力である。データの広域定義を次のように表す。

$$def=(T,V,n)$$

ただし、Tはデータの広域定義を行うタスクの名前または広域変数の初期値を与える実行ブロック、Vは広域変数名、nはデータの広域定義を行う文の番号である。

定義2.2 データの広域参照とは、代入文の右辺あるいは判定文の式での広域変数の参照である。データの広域参照を次のように表す。

$$ref=(T,V,n)$$

ただし、Tはデータの広域参照を行うタスクの名前、Vは広域変数名、nはデータの広域参照を行う文の番号である。

定義2.3 広域データフローとは、次の式で定義される２つ組gdfである。

$$gdf=(def,ref),$$
$$def=(T_i,V,m)\cap ref=(T_k,V,n)\cap i\neq k$$

一つのタスク内でのデータフローは到達可能性を検証する必要があるが、タスク間のデータフローは、異なったタスクでの実行文の実行順序が規定されないので全ての可能性を考慮する必要がある。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。

次に、テスト基準を与えるためにテスト充分性の評価値について定義する。

定義2.4 Pをプログラムの集合、Testをテストの集合とするときにテスト充分性の評価値Wを、次の式で定義する。

$$W=|\Sigma E_p(t)|/|M_p|$$

ただし、M_pはプログラムp∈Pの中の測定対象の事象の集合、$E_p(t)$はテストt∈Testでプログラムpを実行した時に発生する事象の集合である。||は集合の要素数を表す。

定義2.5 広域データフローに基づいたテスト充分性の評価値（広域データフローの被覆率）W_{gdf}は、M_pを広域データフローの集合GDFとしたときのテスト充分性の評価値である。

$$Wgdf=|\Sigma E_p(t)|/|GDF|$$
$$GDF=\{gdf\}$$

{}は集合を表す。

3.デッカーのアルゴリズムにおける広域データフロー

並行プログラムの広域データフローをデッカーのアルゴリズム[Ben-Ari 82]について検討する。２つのプロセス間の排他制御を行うデッカーのアルゴリズムを図3.1の制御フロー図に示す。図中の２つのプロセスは排他制御を必要とする際どい部分$crit_i$(i=1,2)と残りの部分rem_i(i=1,2)から成っている。この２つのプロセスは無限ループの中で実行される。

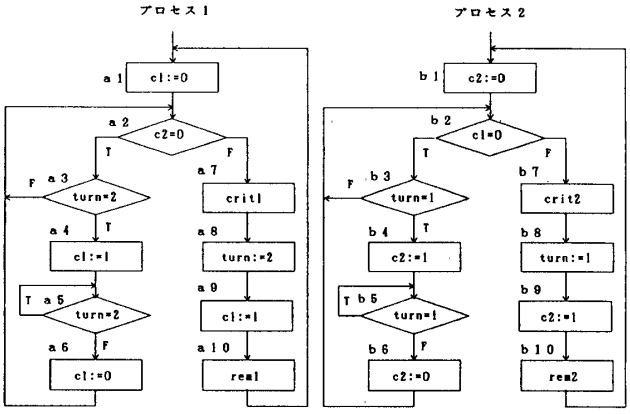

図3.1　デッカーのアルゴリズムの制御フロー図

プログラムの処理を示す制御フローグラフの節点に名札を付ける。図3.1の２つのプロセスにはそれぞれ１０個の節点がある。逐次プログラムのテストで用いられているc_0被覆率は、命令網羅に対する指標である。プログラム中の全ステートメントの中で何%のステートメントが実行されたかを示すので、図3.1では実行された節点の割合である。

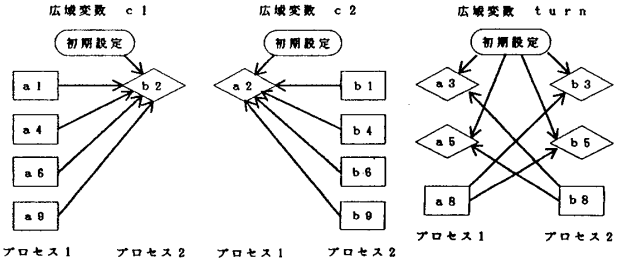

図3.2　デッカーのアルゴリズムの広域データフロー

図3.1のデッカーのアルゴリズムにおいて広域変数(turn,c1,c2)の広域データフローを考える。デッカーのアルゴリ

A Testing Criterion using Global Variable Data Flow for Concurrent Programs
Koji Arimura, Zengo Furukawa and Kazuo Ushijima
Kyushu University

ズムのcrit$_i$とrem$_i$の部分では広域変数へのアクセスは行われないものとする。図3.2に、デッカーのアルゴリズムにおける広域データフローを示す。

図中の長方形は広域変数の代入である広域定義defを表し、ひし形は広域変数の参照である広域参照refを表している。長方形、ひし形の中は制御フローグラフの節点につけられた名札である。また、広域データフローであるdefとrefの対を有向枝で表している。デッカーのアルゴリズムでは18の広域データフローが存在する。

4.広域データフローの被覆率測定

広域データフローの被覆率の測定する試みを行った。デッカーのアルゴリズムをAdaで記述したプログラムと、Ben-Ariがデッカーのアルゴリズムに至るまでの試みとして述べている4つの試み[Ben-Ari 82]をそれぞれAdaで記述した4つのプログラムとを計測の対象として用いた。

4.1.測定方法

広域データフローの発生を記録するために、広域定義及び広域参照の度に、記録用として設けたタスク(モニタ)を呼び出した。モニタは、広域定義及び広域参照のタスク名、広域変数及び文番号を出力する。出力された情報を解析して発生した広域データフローを記録した。今回の実験では、モニタを呼び出すための文の挿入は手作業で行った。プログラムは、Vax8800上のAdaで実行した。

この様なプログラム変換を行うことによってもとのプログラムと動作が異なることが有り得る。しかし、もとのプログラムで有り得ない動作がプログラム変換で起きなければ問題はない。その場合、プログラム変換された後の動作が元のプログラムの動作の一つとなるので変換前の動作をテストで実現したことになる。プログラム変換により元のプログラムで有り得ない動作が導入されていないかは形式的な議論が必要である。

4.2.測定結果

デッカーのアルゴリズムとBen-Ariの第1から第4の試みとを記述したAdaプログラムの広域データフローの個数及び実行したときの広域データフローの被覆率を、表4.1に示す。

表4.1　広域データフローの個数及び被覆率

	デッカー	第1	第2	第3	第4
広域データフローの個数	18	4	6	6	10
発生した広域データフローの個数	12	4	4	4	4
広域データフローの被覆率(%)	67	100	67	67	40

表4.1に示した被覆率の計測では、排他制御を行うそれぞれのプロセスでの繰り返し回数を50回に制限した他は、実行順序等の制御は行っていない。

デッカーのアルゴリズムでは、それぞれの初期設定による広域定義からのデータフローが発生していない。

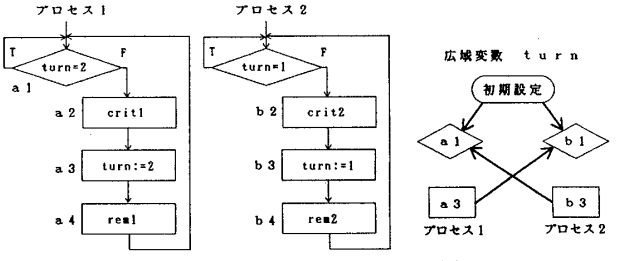

図4.2　第1の試みの制御フロー図及び広域データフロー

第1の試みは共有変数turnによって2つのプロセスの相互排除を行っている(図4.2参照)。このアルゴリズムは際どい部分に入る権利が一方のプロセスから他方のプロセスに明示的に渡されるために、1)プロセスの実行の速度は遅い方のプロセスに強制される、2)一方のプロセスが完了しなければ他方はデッドロックに陥る、という欠点を持つ。

第1の試みでは表4.1を得るための実行の他に、プロセス1を無限ループさせ、プロセス2を0、1、2回と実行させてデッドロックの状態を作った。その結果、プロセス2を0回実行させたときには広域データフローの2つの枝が実行された時点でデッドロックに陥った。つまり、広域データフローの50%を実行するとデッドロックが見つかった。また、プロセス2を1回実行させた場合は4つの広域データフローが実行された後にデッドロックに陥り、2回実行させたときも同様に4つ全部実行された。1回及び2回実行の時は100%の被覆率でデッドロックを発見した。

第2の試みは、広域変数c1とc2を使っているが、相互排除に関する安全性が満たされていないアルゴリズムである。この実行においては4つの広域データフローが実現されるが安全性の不保持は発見できなかった。

第3の試みは、広域変数c1とc2を用いている。これは、デッドロックに陥る可能性のあるアルゴリズムである。このうち実行により4つ(67%)の広域データフローが実現された。この場合にも、デッドロックには陥らなかった。デッドロックに陥る場合としてBen-Ariが示した実行系列では広域データフローは2つが実現できる。

第4の試みも広域変数c1とc2を用いており、プロセスが際どい部分にはいることのない状態が無限に続き、生存性が保たれない可能性があるというアルゴリズムである。実行により発生した広域データフローは4つ(40%)であった。ロックアウトの状態が続くとして、Ben-Ariが示した実行系列では広域データフローは4つ(40%)が実現されることがわかる。

今回の実験では、プロセス間の通信のタイミング不良に起因する誤りを含んだプログラムで広域データフローの被覆率を計測した。広域データフローの被覆率が100%に達しても誤りを発見できない場合があった。これは、広域データフローに基づくテスト基準は、広域変数の解釈が2つのプロセス間で異なる誤りを発見するのに適したテスト基準であるためである。

5.結論及び今後の課題

広域データフローを用いたテスト充分性の評価法を定義し、広域データフローの被覆率を計測した。広域データフローの被覆率と実際のプログラムの信頼性の関係を明らかにすることが今後の課題である。つまり、広域データフローの被覆率が100%に達した時にプログラム信頼性がどれだけあるのかを明確にする必要がある。

さらに、今後の課題として、誤りを発生するようなある特定の実行系列でプログラムを実行させるような強制実行のための手段を考える必要がある。その一つとして、モニタに実行系列を与えて事象を発生させる方法や与えられた実行系列のみを実行する逐次プログラムにプログラム変換をする方法が考えられる。

参考文献

[Clarke 85] Clarke L.A., et al. : "A Comparison of Data Flow Path Selection Criteria", Proc. of 8th ICSE, pp.244-251, 1985.

[Ben-Ari 82] Ben-Ari M. : "Principles of Concurrent Programming(並行プログラミングの原理:渡辺榮一訳)", Prentice Hall Internaltional, Inc., 1982.

[古川 89] 古川善吾、牛島和夫 : "並行処理プログラムのテスト法に関する一考察", 日本ソフトウェア科学会第6回大会論文集, pp.185-188, 1989.