並行プログラムに対するテスト・デバッグ方式

5ZC−6

岡本 渉、川田 秀司、田村 文隆、内平 直志
株式会社東芝 研究開発センター

1　はじめに

並行プログラムでは、複数のタスクあるいはスレッドが絡み合って、ひとつのプログラムを構成している。それらが１つの変数を共有する場合も多い。このため、入力される値が同じでも、スレッドの共有変数へのアクセスのタイミングによって、プログラムの挙動が変化する可能性がある。これは、不具合の顕在化が実行のタイミングに依存することを意味する。このような不具合は「再現性のない不具合」と呼ばれ、並行プログラムのテストにおける本質的な問題である。

こうした再現性のない不具合を発見するためには、スレッド実行のタイミングも考慮したテストが必要である。グローバル状態遷移解析技術 ([1]) は、このような問題を解決する技術であり、すべての動作を状態遷移図の形式で表し、それをチェックするという概念に基く。この手法を用いれば、実際に発生し得る実行パターンすべてを把握することが可能である。しかし現実問題として、タイミングの組み合わせの数は膨大であり、到底すべてをチェックすることはできない。グローバル状態遷移解析の分野では、すべての状態遷移を調査したものと同等の結果を導けることを保証しつつ、いかに小さな状態遷移図を生成するかといった研究がなされている (半順序法 ([2])、スリープセット法 ([4]))。

これらの手法では、枝刈り手法と呼ばれる、「プログラムの振舞いとして同等とみなせるものは１つの実行パターンに帰着させる」という方法が取られる (１つの実行パターンを“枝”と呼び、複数のパターンを１つに帰着させる様を“枝を刈る”と表現する)。しかし、これらの手法を現実のプログラムのテストに適用するためには、正確な依存性解析など解決すべき問題が多く存在する。そこで我々は、動的に依存性を解析することで、正確な依存性を取得し、枝刈りを行う手法を開発した。さらに観察同値という概念を導入することで並行プログラムのテスト作業の効率化を図った。

2　動的枝刈り

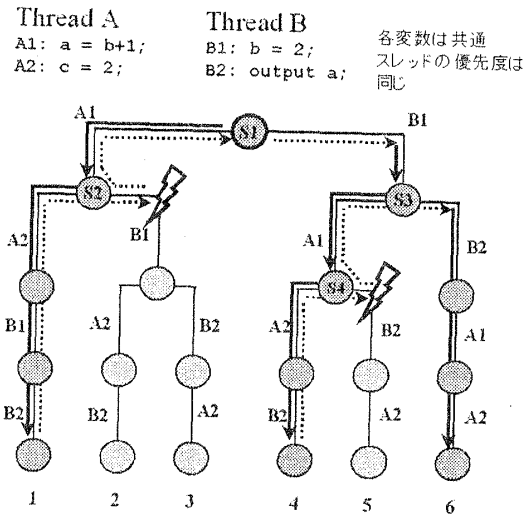

図 1: 状態遷移図

効率的に枝刈りを行うためには、正確な依存関係の取得が必要である。枝刈りは「互いに依存関係のない命令は、その実行順序を入れ換えても、プログラムの挙動としては等しい」という概念に基いているからである (実行パターンの同値性)。したがって、いかに正確な依存関係を得るかが、枝刈りの効率を決める最も重要な要素となる。前節で紹介した手法では、静的解析が用いられるのが一般的である。しかし参照型の変数が多く使われる場合や再帰呼出しなどが存在する場合、静的な依存関係解析は困難である。そこで我々は、実際にプログラムを実行して状態遷移図を作成しつつ、依存関係を調べ、その結果により枝刈りを行う手法を考案した。

動的な枝刈り手順は、まず１つの実行パターンを実際に動作させ、そのときに変数間の依存関係を記録する。次に、他の実行パターンを動作させようとする際、記録しておいた変数の依存関係を調べ、既に実行したパターンとの同値性を調べる。依存関係がなければ、新たなパターンは実際に実行する必要はない。「依存関係のない命令は実行順序に依らないので、どちらか一方の順序に従ってのみテストすれば十分」だからである。一方、依存関係があれば、実際に実行する必要がある。その場合は、再び実行時に変数の依存関係を記録しておき、次の枝刈りに利用する。

動的枝刈り機構を単純な例を用いて説明する。図 1. は A、B ２つのスレッドから構成されるプログラムの

The testing and debugging method for concurrent programs
Wataru Okamoto, Hideji Kawata, Fumitaka Tamura, and Naoshi Uchihira　Toshiba corp. Research & Development Center
e-mail: wa.okamoto@toshiba.co.jp

状態遷移図である。プログラムの動作として考えられるパターンは6通り存在する。

実行パターン選択の際、どちらのスレッドも実行可能な場合は、A から実行するとしている。まず最初に枝1が作成される。次に枝刈り機構は、スレッドの選択肢がある位置 (状態 $S2$) まで戻る。この時点で、枝1において $A2$ はそれ以降の命令 $B1$、$B2$ とも依存関係を持っていないことが判明している。つまり $B1$、$B2$ を $A2$ より先に実行したとしても結果は枝1と変わらない。したがって、$B1$ を実行するパターン (枝2、3) は枝1と同等とみなすことができ、枝2、3を刈ることができる。

次にさらに上の状態、$S1$ へ戻る。ここで、命令 $A1$ は $B1$ との依存関係を持つことがわかっている。そのため、状態 $S2$ のように枝刈りを行うことは不可能であり、$B1$ から始まるパターンも実際に調査しなくてはならない。先ほどと同様の処理、判定を行うことにより、状態 $S4$ で枝5を刈ることができる。最終的に、テストすべき動作パターンとしては、枝 1,4,6 の3つが残る。

3 観察同値

我々は上記の動的枝刈りに加え、チェックすべき実行パターン数のさらなる減少、及び、プログラム修正前のテスト結果の再利用を目的として、「観察同値」という概念を導入した。この概念は、プログラム内部の振舞いは異なっても、ユーザから見て同じ振舞いをする実行パターンは1つのパターンとして扱おうというものである。つまり観察同値では、プログラムの入出力列に注目し、入出力列の等しい実行パターンは同じとみなす。これにより、枝刈り後の実行パターンをさらに減らし、並行プログラムのテスト作業を軽減させることができる。

4 テスト方式

最後に枝刈り手法、観察同値を用いたテスト方法を提示する (図2.)。まずテスターは基本となるプログラムの入出力列とその正誤をシステムに登録する。システムは枝刈りを行い、得られた実行パターンに対して観察同値を適用する。ここでテスターには、正誤を含めて登録された入出力列とは異なる実行パターンだけを示し、その正誤を問う。プログラムに修正が必要な場合でも、登録された入出力列は次回からのテストに再利用される。同じ入出力列に対する判定はシステムが行い、テスターは再び同じ判定をすることはない。この方式により、テスターに示すチェックすべき実行パターンを最小限に抑えることができる。

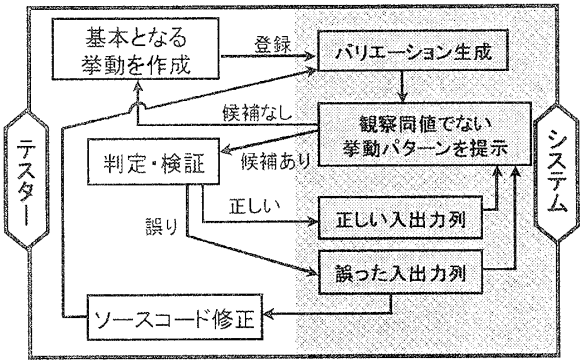

図 2: テスト手法

5 まとめ

「再現性のない不具合」を発見するためには、1つのテストケースに対して、並行プログラムのあらゆる実行パターンをテストしなくてはならない。この問題に対し、同等とみなせる実行パターンは1つの実行パターンとみなすという手法が用いられる (枝刈り)。

我々が開発した手法の特徴的な点は、動的に依存関係を調べる点にある。プログラムの実行時に依存関係を調べ、それを枝刈りの判定に用いる。動的な依存関係解析は、正確な依存関係の取得につながり、それは効率的な枝刈りを導く。これにより、本質的に挙動の異なるパターンだけをピックアップできる。さらに観察同値という概念を導入することで、テストすべき実行パターンを減らし、並行プログラムに対して、効率的にテストを行うことが可能となる。

参考文献

[1] E.M. Clarke *et al.* , Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications, ACM TOPLAS, Vol.8, No.2 (1986).

[2] P. Godefroid & P. Wolper , Using Partial Orders for the Efficient Verification of Deadlock Freedom and Safety Properties, *Proc. CAV'91*, LNCS Vol.575, Springer-Verlag (1991).

[3] N. Uchihira and H. Kawata , 'Scenario Based Hypersequential Programming', 2nd International Workshop on Software Engineering for Parallel and Distributed Systems, IEEE Computer Society Press, (1997).

[4] P. Wolper & P. Godefroid , Lecture Notes in Computer Science, Springer-Verlag, Hildesheim, August, (1993).