順序列テスト基準における測定事象の最適化方法について

7M－2　伊東栄典*, 川口豊*, 片山徹郎*, 古川善吾**, 牛島和夫*
* 九州大学工学部情報工学科,　** 九州大学情報処理教育センター

1. はじめに

プログラムをテストする場合、テスト基準に従いテストを行なう方法がある。従来の逐次処理プログラムに対しては様々なテスト基準が提案されている。並行処理には実行の非決定性、プロセス間の通信や同期といった逐次処理にはない動作の複雑さを持つ。このため逐次処理プログラムのテスト基準だけで、並行処理プログラムをテストするのは妥当ではない。我々は、現在までに並行処理プログラムの新たなテスト基準として順序列テスト基準を提案してきた[2]。

順序列テスト基準では、通信や同期に関する文(以後、通信同期文と呼ぶ)の列をテストにおける測定事象とする。列を測定事象とすることにより、並行処理プログラムの特徴であるプログラムの非決定的な動作や、プロセス間の同期や通信をテストすることが可能になる。しかしながら、順序列テスト基準では通信同期文を機械的に並べた列を測定事象としているため、プログラムの実行において実現不可能な列を測定事象に含めてしまうという問題がある。

本稿では、事象グラフ[3]を利用して、実現不可能な順序列を検出する方法について考察する。

2. 順序列テスト基準

並行処理プログラムは、複数のプロセスが互いに通信や同期を行いながら処理を進めるプログラムである。並行処理プログラムの実行をインターリーブモデルで考える[1]。プロセス間の通信や同期の実行順序が変化すれば、プログラムの実行結果が変化する可能性がある。そこでプロセス間の通信や同期の実行順序をテストにおける測定事象とする順序列テスト基準を提案した[2]。以下にその定義を示す。

[定義 1]

$$\mathrm{OSC}_k = \{< s_1, s_2, \cdots s_k > | s_i \in SYNC\}$$

s　：通信同期文
$SYNC$　：通信同期文の集合

列の長さ k $(k \geq 1)$ を変化させることにより、順序列テスト基準は様々なレベルのテストの要求に応えることができる。例えば、$k = 1$ ならばプロセス間の通信同期文を少くとも1回実行させるテスト基準となる。$k = 2$ ならばプロセス間通信や2つのプロセス間の同期をテストするテスト基準となる。

Taylorらは並行状態グラフを定義し、このグラフ上の節点や路を被覆するテスト基準をいくつか提案している[4]。しかしながら並行状態グラフには、(1)節点や枝の数がプロセス数に対して組合せ論的に増大する、(2)プログラムの実行前にプロセス数が静的に決ってなければならない、といった問題がある。これに対し順序列テスト基準では、プログラム内の通信や同期に関わる文の列だけを測定事象とするために、Taylorの並行状態グラフにおける問題点の(2)は回避できる。

3. 測定事象の最適化

順序列テスト基準では、通信同期文の列を測定事象にしているため、実際のプログラムの実行においては実現不可能な順序列を測定事象に含む場合がある。事象グラフ[3]を用いて、プログラムの実行において実現不可能な順序列を検出する方法を考案した。

事象グラフとは、通信や同期に関わる文とそれに関する分岐文とを節点とし、節点から節点への制御の移行を枝とするグラフである。1つのプロセスに対し、1つの事象グラフ EG が出来る。事象グラフの形式的な記述は以下の通りである。

事象グラフ：$EG =< N, E, s, f >$
$N = \{u\}$,: 節点集合.
$E = \{< u, v > \mid u, v \in N\}$,: 枝集合.
s：開始節点. ただし $< u, s > \notin E$.
f：終了節点. ただし $< f, u > \notin E$.

事象グラフの例を図示する。

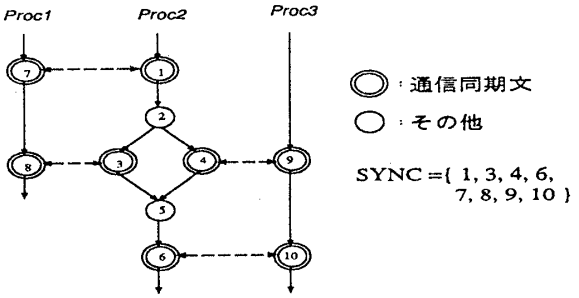

図 1: 事象グラフの例

長さ2の順序列テスト基準の測定事象はこのグラフの場合、次のようになる。

$< 1,1 > < 1,3 > \cdots \quad < 1,10 >$
$< 3,1 > < 3,3 > \cdots \quad < 3,10 >$
$\vdots$
$< 10,1 >< 10,2 > \cdots \quad < 10,10 >$

Optimization of Test events of Ordered Sequnece Criteria.
E. Itoh, Y. Kawaguchi, T. Katayama,
Z. Furukawa and K. Ushijima.
Dept. of Comp. Sci. and Comm. Eng., Kyushu Univ.

上記の測定事象には、実現不可能な列がある。これを除去する方法を考える。まず、ある関係を定義する。

[定義 2] クリアな後続点関係 $Clear(a,b)$
a,b を事象グラフ EG 上の通信同期文を表わす節点とする。EG 内で、a から b への路が存在し、かつその路に a,b 以外の通信同期文節点が存在しない場合、b は a のクリアな後続点である、といい、a,b の関係を $Clear(a,b)$ で表わす。

図 1 のグラフで、クリアな後続点関係は以下の通り。

$$Clear(1,3),\quad Clear(1,4),\quad Clear(3,6),$$
$$Clear(4,6),\quad Clear(7,8),\quad Clear(9,10)$$

この関係を用いて、プログラムの実行において実現不可能な測定事象を除去する定理を導く。

[定理 1]
事象グラフ EG_i で表されるプロセス P_i が、プログラムの実行において自分自身のコピーを作らないものとする。測定事象である順序列 seq から P_i 内の節点だけを取出した列 $ex\_seq =< a_1, a_2, \cdots, a_l >$ を考える。

列 ex_seq において、$l \geq 2$ のとき $Clear(a_i, a_{i+1})$ $(1 \leq i \leq l)$ が成立しなければ、順序列 seq はプログラムの実行において実現不可能である。

[証明]
並行処理プログラムの実行を、インターリーブモデル [1] で考える。インターリーブモデル上では、プロセッサはプロセス内の文を一時に一つ実行する。どのプロセス内の文を実行するかは任意である。

プロセスの実行系列は、プロセスを制御フローグラフで表わした場合におけるグラフ上の路 (path) として表現できる。並行処理プログラム全体の実行系列は、個々のプロセスの制御フローグラフ上の路を、シャッフルしたものとして表現できる。

プロセスの制御フローグラフ上の路から、同期や通信に関わる文だけを取出した列を考える。この列を $< a_1, a_2, \cdots, a_l >$ とすると、この列では必ず、

$$Clear(a_i, a_{i+1}),\quad 1 \leq i \leq l-1,$$

が成立する。

次にプログラム全体の実行系列から、同期や通信に関わる文だけを取出した列を考える。この列から、特定のプロセス P 内の文だけを取出す。これを $< b_1, b_2, \cdots, b_m >$ とすると、

$$Clear(b_i, b_{i+1}),\quad 1 \leq i \leq m-1,$$

という関係が成立する。

すなわち、プログラムの実行系列において同一プロセス内の同期や通信に関わる文は常にクリアな後続点の関係を持つ。

故に、クリアな後続点関係を持たない順序列はプログラムの実行において実現不可能である。

□

以上の証明を図 2に示す。

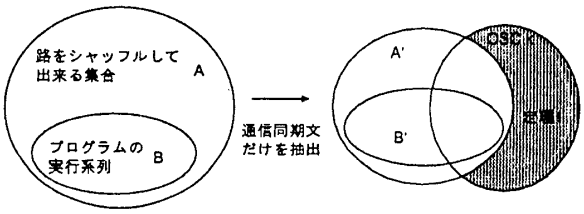

図 2: 証明図

図 1 の事象グラフにおいて、定理 1 から導き出される実現不可能な順序列を以下に挙げる。

$< 1,6 >, < 3,1 >, < 3,4 >, < 4,1 >,$
$< 4,3 >, < 6,1 >, < 6,3 >, < 6,4 >,$
$< 8,7 >, < 10,9 >$.

今回提案した方法の計算量を考察する。

並行処理プログラムから事象グラフを構築する計算量は、プログラム内の文の数を n とすると、$O(n^2)$ 以下である。次にクリアな後続点関係を検出する計算量を考える。この関係を検出する事は、グラフ上の最短経路を探索する事と等価である。故にクリアな後続点関係を検出する計算量は、プロセス P_i の事象グラフを $EG_i =< N, E, s, f >$ とすると、$O((|N|+|E|)\log|N|)$ である。

最後に順序列テスト基準の測定事象に対する、クリアな後続点関係を検出する計算量を考える。これは、順序列テスト基準の測定事象数を m 個とすると、検出のための計算量は、$O(mp)$ となる。ただし、p はプログラム内のプロセス数を表わす。

4. おわりに

今回、順序列テスト基準の測定事象の中から、実現不可能な列を検出する方法を考案した。この方法でテストにおける測定事象数を、より最適化することが可能になる。しかしながら、今回の方法で、すべての実現不可能な測定事象を検出できるわけではない。

今後、今回提案した方法を計算機上に実装する予定である。またより測定事象を精密化する方法を考案する必要がある。

参考文献

[1] Ben-Ari, M. : *Principles of Concurrent Programming*, Prentice Hall International, Inc. (1982).
翻訳 渡辺 榮一 : 並行プログラミングの原理, 啓学出版 (1986).

[2] 伊東栄典, 川口豊, 古川善吾, 牛島和夫 : C 並行処理プログラムのプロセス間通信に関するテスト充分性評価について, 情処研報, 93-SE-90, pp.9-16 (1993).

[3] 片山徹郎, 菰田敏行, 古川善吾, 牛島和夫 : 並列処理プログラムにおけるテストケースの定義と生成ツールの試作, 情報処理学会論文誌, Vol.34, No.11, pp.2223-2232 (1993).

[4] Taylor, R. N., Levine, D. L. and Kelly, C. D. : *Structural Testing of Concurrent Programs*, IEEE Trans. Softw. Eng., Vol 18, No.3, pp.206-215 (1992).