安全性の破壊に対する順序列テスト基準の信頼性評価

2N－1

伊東栄典*, 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

並行処理プログラムの信頼性向上手法の一つとしてテストは重要である。並行処理プログラムは複雑であるため、逐次処理プログラムとは異なった開発方法、特にテスト方法が必要である。既に、並行処理プログラムを特徴付ける通信同期文の順序列に基づく順序列テスト基準を提案した[2]。その中で、通信誤りと同期誤りの中のデッドロックとして知られる生存性の破壊とに対する順序列テスト基準を用いたテストの誤り発見能力について分析してきた。

並行処理プログラムの同期誤りには、生存性の破壊の他にも、安全性の破壊や公平性の破壊という誤りが知られている[1]。安全性の破壊は、並行処理で使用する資源の中で、複数個の処理が共有する資源を秩序立てて使用するための使用権を確保する相互排除の失敗によって発生する。安全性の破壊は、使用権を確保しなければならない資源であることの認識を忘れることにより、比較的良く発生する誤りであるため、テストによる誤り発見の可能性を検討することはプログラムの信頼性向上に役立つ。今回、安全性の破壊を定義し、安全性の破壊に対する順序列テスト基準の信頼性について定性的に分析する。

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、文を単位とするインターリーブモデルで考える[1]。このモデル上では、プロセス間の通信や同期を行う文(通信同期文)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。そこで通信同期文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]。

通信同期文は、これまで通信や同期を行なう文であるとして、具体的には C のシステムコール呼出文を扱ってきた。安全性の破壊に対する信頼性を検討するために、今回、同期通信文をプロセス間での共有資源を操作する文であるとする。システムコールは、セマフォやソケットと言う共有資源に対する操作を実現したものであるので、今回の拡張は、定義をより一般化したものである。

順序列テスト基準の定義を以下に示す。

定義 1 順序列テスト基準

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

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。

3　安全性の破壊

安全性の破壊とは相互排除の失敗によって資源の秩序だった使用がなされないことである。資源としては、変数やバッファ、ファイルがある。秩序だった使用とは、データの一貫性が確保された使用やファイル内容の独立した存在である。安全性の破壊の正確な定義は後述する。例えば、次の例は安全性の破壊の例である。

(1) ファイルの操作における安全性の破壊

P1,P2: プロセス, file1,file2,file3: ファイル

```
     P1                              P2
1 while(～EOF){              4 while(～EOF){
2  read(file1,buf,512);      5  read(file2,buf,512);
3  file3=write(buf);}        6  file3=write(buf);}
```

プロセス1が file1 の内容を file3 に書き込み、プロセス2が file2 の内容を file3 に書き込む場合を考える。書き込みは 512 バイトごとに順次実行される。file1 と file2 が 512 バイトよりも大きく、かつ P1 と P2 が連続して実行されると file3 の中で、file1 と file2 がそれぞれひと塊として存在する。しかしながら、P1 と P2 とが並行に実行されると、file3 の中で file1 と file2 とは、混在してしまう。file3 が出力ファイルである場合には、混在したファイルは内容の分析が困難になる。すなわち、P1 と P2 はそれぞれ際どい領域として別々に実行されるべきものである。しかしながら、P1 と P2 を際どい領域として認識していないために秩序立った利用がなされておらず、安全性の破壊が発生している。

(2) 共有変数の操作

r: 共有変数, P1,P2: プロセス,

```
     P1              P2
1  r:=r+x;      2  r:=r+y;
```

これは、並行処理においてデータの一貫性が失われる例として良く知られた例である。文1あるいは文2では、r の値を参照し、それから加算を計算し、その後代入する。この代入が行われる前にもう一方のプロセスが r の値を参照してしまうと、データの一貫性が失われ、安全性の破壊が発生する。

安全性の破壊を定義する前提として以下の項目を明かにする必要がある。

1. 並行処理単位の定義

文単位の並行処理を対象とする。例 (2) の共有変

Reliability of Ordered Sequnece Testing Criteria for Safeness Errors.
E. Itoh, Z. Furukawa and K. Ushijima.
Dept. of Comp. Sci. and Comm. Eng., Kyushu Univ.

数の操作では変数の参照と代入が独立した文になっていないので対象外である。

2. 際どい領域 (Critical Section)
相互排除が行われなければ誤った出力となるプログラムの部分である。この際どい領域には、一般に共有資源を操作する文が含まれている。

3. 相互排除の機構
相互排除を実現するための機構として、ここではセマフォを考える。

4. 相互排除の実行系列
正しく相互排除が行なわれた時の実行系列を定義する。逆にこれを満たさない列は、相互排除の失敗であるとする。正しい相互排除の実行系列を以下に述べる。

プロセス P1 の際どい領域を CS1, プロセス P2 の際どい領域を CS2 とする。それぞれの際どい領域内の文を、s1.1, s1.2, … および s2.1, s2.2, …とする。

```
P1,P2：プロセス   CS1,CS2：際どい領域
s1,s2...sk：文
        P1              P2
       s1.1            s2.1
CS1    s1.2      CS2   s2.2
       s1.3            s2.3
        :               :
```

図 1 際どい領域の例

図 1の場合、正しく相互排除が行なわれたならば、プログラムの実行系列は、{CS1}, {CS2} (s1.1, s1.2, s1.3 … s2.1, s2.2, s2.3 …)、あるいは {CS2}, {CS1} (s2.1, s2.2, s2.3 … s1.1, s1.2, s1.3 …)、となる。際どい領域内の文が混ざり合わずに実行された場合、正しく相互排除が行われている。すなわち、共有資源が 1 つの場合には、際どい領域内の文は連続して実行される必要がある。そこで、安全性の破壊を以下のように定義する。

定義 2 安全性の破壊

際どい領域があるとする。この領域は共有資源 r を操作する文の集合である。ある際どい領域内に存在する文の実行が終了する前に、他の際どい領域内の文が実行されたならば、これを安全性の破壊という。

4 順序列テスト基準の信頼性

テスト基準 Cr が「信頼できる」とは、テスト基準 Cr を満足したならば、プログラム内の誤りを必ず発見できることをいう [3]。しかしながら、任意のプログラムに信頼できるテスト基準は徹底テストしかない。このため現実のテスト基準では、部分的にしか信頼できない。ここでは順序列テスト基準の安全性の破壊に対する信頼性を考察する。

前節で述べた様に、異る際どい領域内の複数の文が混ざり合った実行系列が発生した際に安全性の破壊となる。すなわち、図 1の場合、

s1.1 s2.1 s1.2 ···

のような,CS1 の文と CS2 の文が混ざり合った実行系列が起ると安全性の破壊である。

1 つの際どい領域内の文が終了する前に、他の際どい領域内の文が実行されたならば安全性の破壊である。そこで、際どい領域内の資源を操作する文 (通信同期文) で長さ 3 の順序列を作れば、安全性の破壊となる文の実行系列を測定事象の中に含むことになる。故に次の定理が成り立つ。

定理 1

相互排除の失敗による安全性の破壊に対し、長さ 3 の順序列テスト基準が信頼できる。

証明は省略する。

5 おわりに

本稿では、安全性の破壊に対する信頼性を定性的に分析した。今回の分析では共有資源を操作する文をすべて通信同期文と考えた場合の順序列テスト基準の安全性の破壊に対する信頼性を分析した。共有資源が 1 つの時は長さ 3 の順序列テスト基準が信頼できる。共有資源が複数個存在する場合については、今後分析する。また、共有変数に対するデータの一貫性を保証するための相互排除については、今回は、並行処理の単位を文としたために対象外とした。

共有資源が 1 つの場合は安全性の破壊に対して長さ 3 の順序列テスト基準が信頼できるけれども、これをそのまま実際のプログラムに適用することは、実用的とはいえない。順序対 (長さ 2 の順序列) テスト基準を実用プログラムに適用する [2] 場合にも測定対象の数の増大が問題になった。長さ 3 の順序列テスト基準でも測定事象数を削減する必要がある。また順序列テスト基準の測定事象の実行可能性を検討する必要もある。

参考文献

[1] M. Ben-Ari : *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 : 順序列テスト基準に基づく並行処理プログラムのテスト充分性評価, 情報処理学会論文誌, Vol.36, No.9, 1995.(掲載予定)

[3] W. E. Howden : *Reliability of the Path Analysis Testing Strategy*, IEEE Trans. Softw. Eng., Vol.SE-3, No.4, pp.226-278 (1976).