並行処理プログラムのテストケース作成ツールについて

5T−4

菰田敏行　片山徹郎　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

従来、テストデータを選定する基準となるテストケースが人の手によって作られることが多かったため、テストデータに漏れや重複が多くあった。テストケースを系統的に作成することにより、テストの質を高めることができる。逐次処理プログラムにおいては、テストデータやテストケースについての研究が広くなされているけれども、並行処理プログラムに対しては十分とは言い難い。これまでに、並行処理プログラムのモデルとテストケースを定義し、プログラミング言語 Ada を対象としたテストケース自動作成ツール TCgen を試作した。本発表では、TCgen の概要と Ada プログラムにツールを適用した結果について述べる。

2. 並行処理プログラムのモデル化

並行処理プログラムのモデルとして、事象グラフと制約関係からなる事象制約モデルを定義した[1]。プログラム中の並行動作に関係する文(並行事象文)および分岐文を節点とし、制御の移行を枝とする有向グラフが事象グラフ(Event Graph)である。タスク間の通信を表す事象グラフの節点の対の集合が制約関係(Constraints)である。事象グラフと制約関係とを用いて事象制約モデル(ECM)として並行処理プログラムをモデル化する。事象グラフ上の節点の順序列を路(Path)とし、路を制約関係が満たされるように組み合わせた協調路(CoPath)が並行処理プログラムのテストケースである。

3. ツールの概要

TCgen は Ada 並行処理プログラムのソースプログラムからテストケースとして協調路を作成する。ツール TCgen の概要を図1に示す。

まず、Ada のソースプログラムから制御フローグラフと制約関係を抽出する。制御フローグラフはプログラムの実行文を節点とする有向グラフである。次に、制御フローグラフから並行事象文および分岐文以外の実行文を削除して事象グラフを作成する。各事象グラフにおいて、路を作成する。このとき4章で述べるテスト基準を満足させる。最後に5章で述べるアルゴリズムに従って路と制約関係からテストケースを作成する。

4. テスト基準

テストケースの作成においては、テストの質を明らかにするためにテスト基準が必要である。今回、並行処理プログラムのテストケース作成のためのテスト基準として、以下の基準を設定した[2]。

i) 枝被覆基準－事象グラフの全ての枝を少なくとも1回は通る
ii) ループ基準－ループを含む場合は0回と1回繰り返す場合を考える。
iii) 制約被覆基準－並行処理プログラム内の全ての制約が、少なくとも1回は実現される。

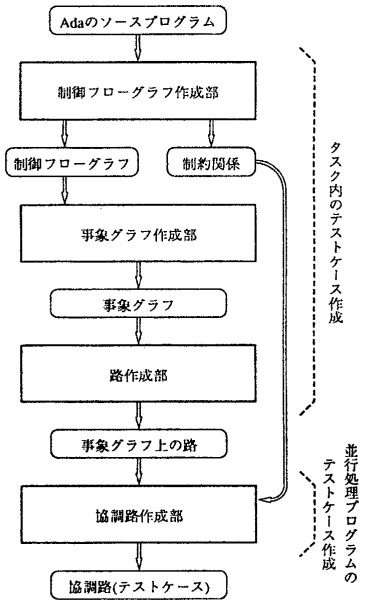

図1: ツールの概要

5. テストケース作成のアルゴリズム

路と制約関係から協調路を生成するアルゴリズムを以下に示す。

1 初期設定
1.1 各タスクに対してタスク訪問フラグを *false* にする。
1.2 各路に対して使用可能フラグを *true* にする。
2 タスク0についてタスク訪問を行う。
3 手続き タスク訪問
3.1 訪問するタスクの使用可能フラグが *true* である路それぞれについて以下のことを行う。

- 路の制約関係に含まれる節点と対をなす節点を持つタスクについて、対をなす節点を持たない路の使用可能フラグを *false* にする。
- ここで、全てのタスク訪問フラグが *true* であるとき、各タスクについて使用可能フラグが *true* である路が存在すれば、それらの

A Testcase Generation Tool for Concurrent Programs
Toshiyuki KOMODA, Tetsuro KATAYAMA, Zengo FURUKAWA and Kazuo USHIJIMA
Kyushu University.

組を協調路としてテストケースに登録する。

- タスク訪問フラグが *false* であるタスクが残っているとき、対をなす節点を持つタスクに使用フラグが *true* である路が存在すれば、そのタスクについて再帰的にタスク訪問を行う。

3.2 路の使用可能フラグをこの手続きが呼び出される前の状態に戻す。

3.3 この手続きから戻る。

6. ツールの実行例と限界

図2の被テストプログラムに対してツールを適用したときの出力結果を図3に示す。図2で実行文の左側の数字は節点に対応する番号である。タスクT1,T2,T3および手続きexample本体は、それぞれTASK 0,1,2,3と図3で表される。図3の[Constraints]は、タスクT1の実行文1とタスクT3の実行文2, タスクT2の実行文2とタスクT3の実行文4 がそれぞれ通信することを表す。[Event Graph]は、各タスクの事象グラフを節点の対である枝の集合で表す。0および-1は、それぞれ仮想的に設けた開始点, 終了点を表す。[Path], [Testcase]は、作成された路, 協調路を節点の列として表す。実行に要する時間は、SUN SPARCstation2において 0.2 秒であった。

TCgenは現在のところ、以下の構造のAdaプログラムに適用できない。

- タスクの中にタスクを宣言したもの
- 例外処理のあるもの
- 複数の節点との通信があるもの
- 同じタスク内の節点間に通信があるもの

7. おわりに

Ada並行処理プログラムからテストケースを自動的に作成するツールTCgenの概要と限界を述べた。今後、限界についてTCgenの改良を検討する。

並行処理プログラムでは、入力データを与えただけでは実行系列を決定することができない。そこで、TCgenが作成したテストケースのとおりに並行処理プログラムを実行するためには強制実行の機構が必要である。今後、強制実行のためのツールについて検討を行う。

参考文献

[1] 片山徹郎, 菰田敏行, 古川善吾, 牛島和夫:"並行処理プログラムのためのテストケース生成系の試作," 情報処理学会研究報告,Vol.92,No.59,pp.9-16,1992年7月

[2] 片山徹郎, 菰田敏行, 古川善吾, 牛島和夫:"並行処理プログラムのテストケース生成におけるタスク型に関する一考察," 日本ソフトウェア科学会第9回大会,1992年9月(発表予定)

```
    procedure example is
      task T1;
      task T2;
      task T3 is
        entry E1;
        entry E2;
      end;

      task body T1 is
      begin
1       T3.E1;
2     end T1;

      task body T2 is
      begin
1       loop
2         T3.E2;
3       end loop;
4     end T2;

      task body T3 is
      begin
1       loop
2         select
3           accept E1;
          or
4           accept E2;
5         end select;
6       end loop;
7     end T3;

    begin
1     null;
2   end example;
```

図2: 被テストプログラム

```
[Constraints]
TASK:0 - 1 = TASK:2 - 3
TASK:1 - 2 = TASK:2 - 4

[Event Graph]
TASK:0 - 0 1   1 -1
TASK:1 - 0 1   1 2   2 3   3 1   1 -1
TASK:2 - 0 1  1 2  2 3  2 4  3 5  4 5  5 6  6 1  1 -1
TASK:3 - 0 -1

[Path]
TASK:0
0 1 -1
TASK:1
0 1 -1
0 1 2 3 1 -1
TASK:2
0 1 -1
0 1 2 3 5 6 1 -1
0 1 2 4 5 6 1 -1
TASK:3
0 -1
[Testcase:0]
TASK:0 Path:0 - 0 1 -1
TASK:1 Path:0 - 0 1 -1
TASK:2 Path:1 - 0 1 2 3 5 6 1 -1
TASK:3 Path:0 - 0 -1
[Testcase:1]
TASK:0 Path:0 - 0 1 -1
TASK:1 Path:1 - 0 1 2 3 1 -1
TASK:2 Path:3 - 0 1 2 4 5 6 1 2 3 5 6 1 -1
TASK:3 Path:0 - 0 -1
[Testcase:2]
TASK:0 Path:0 - 0 1 -1
TASK:1 Path:1 - 0 1 2 3 1 -1
TASK:2 Path:4 - 0 1 2 3 5 6 1 2 4 5 6 1 -1
TASK:3 Path:0 - 0 -1
```

図3: 実行結果