組み込みソフトウェアのためのテスティングツールの開発

1. 背景

ソフトウェアの検証、試験工程（検証というときには機能的な仕様を対象とし、試験というときにはプログラムがその対象であるとする）では、つぎのことが課題として取り上げられる。

- 試験時間に関する課題
- 検証、試験の質に関する課題

最初の試験時間に関する課題とは、限られた開発時間の中で検証や試験に充当できる時間が限られており満足のいくレベルまで検証、試験ができないというものである。つぎの検証、試験の質に関する課題とは、その質が担当者の経験に依存した方式に陥りがちで検証、試験の質にばらつきが生じてしまうということである。前者は、近年ソフトウェアの規模が増大しているにもかかわらず開発期間中で検証、試験に十分な時間を取れないことに原因がある。また、後者の原因としてはつぎのことが考えられる。検証、試験を効率的に実施するためのツールが十分に整備されていないために生じる。

これらの問題を解決するために、形式的な手法は有効であると考えられる。モデル検証は形式的にソフトウェアの可能な状態すべてを検証する手法である。モデル検証ツールとしてすでに SPIN などが知られている[1]。これらは機能的な仕様やプログラムの振舞いを表した抽象的なモデルとそれに望まれる性質（デッドロックを生じないことなど）を与え、モデルがその性質を充たしていることで正当性を示す。

これまでのツールは検証または試験のいずれかに利用するように作成されている。またいくつかのツールはソースコードから適当なモデルへ変換する必要がある。機能的な振舞いを形式化したモデルをプログラム試験にまで拡張することは試験時間の短縮と試験の質の向上に寄与すると考えられる。

2. 目的

ここでは、前述ふたつの課題を解決することをめざしたモデル検証ツールを開発する。

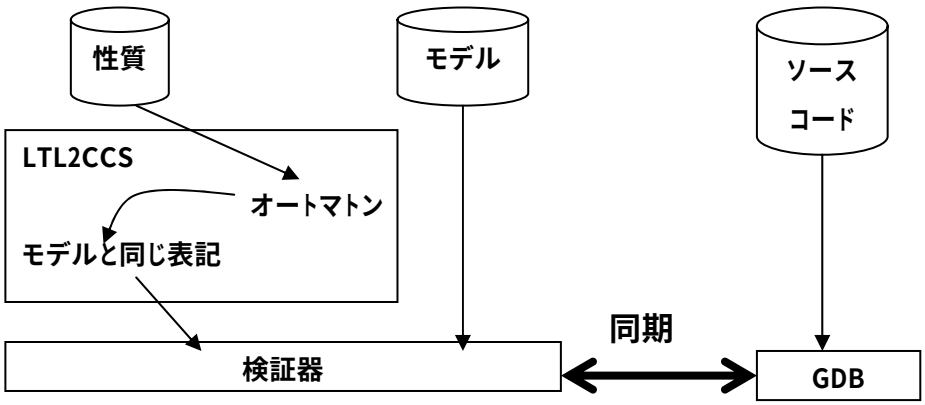

図 1 検証ツールのダイアグラム

[1] http://spinroot.com/spin/wahtisspin.html
http://javapathfinder.sourceforge.net/index.html
http://cm.bell-labs.com/who/god/verisoft/index.html

このツールは GDB と同期して動作することでソースコードを直接、試験する。このとき、モデルの一部がいっしょに用いられる。図 1 にその概念図を示す。

3. 開発の内容

本開発では、モデル記述言語、検証ツールを開発した。この節では、本ツールを用いた、モデル検証からプログラム試験までのステップを示す。非常に簡単な例としてここでは Echo サービスを取り上げる。 Echo サービスのクライアントとサーバのモデルの振舞いを今回開発したモデル記述言語を用いて図 2 のように定義されたとする。この言語は Milner の CCS を基本にした体系を持つ。図中の ; はコメント行を表す。まず、モデルが

```
0: ;(bind srv “server:127.0.0.1:2005”)
1: ;(bind srv_res "target:echos:28")
2: ;; Timer
3: (define Timer()
4:   (start(init_v):Timer1(init_v)))
5: (define Timer1(time)
6:  (~tick(time-1):tick(time):
7:    (if (time=0)
8:        (~timeout(TRUE):stop)
9:        (Timer1(time)))))
10:(define Timeout()
11:  (timeout(zz):
12:    (if (zz)
13:     (~display("TIMEOUT"):
            ~srv("quit"):stop)
14:       (stop))))
…
```

```
…
15: ;; Echo client
16: (define Recv1()
17:  (srv_res(yyy):
18: ;  (srv(yyy):
19: ;    (~display(" ",yyy):Send(100,n-1))))
20:    (Send("TEST",n-1))))
21: (define Recv() (Recv1++Timeout))
22: (define Send(x,n)
23:   (if (n=0)
24:       (STOP)
25:       ((~srv(x):~start(5):
26:         (Recv)))))
27: (define Echo_client()
28:   (Send(100,5)||Timer))
29: ;; Echo server
30: (define Echo_server()
31:   (srv(req):~srv_res(req):STOP))
32: ;; Initial process
33: (Echo_server || Echo_client)
```

図 2 Echo サービスの簡単なモデル

完成したとしてその検証を行うために検証ツール起動する。

```
> ./rccs –f “<> !(yyy=x)”
```

と与える（この例ではいつか送信したものと受信したものが違う場合があるという性質をあらわしている）。 -f は性質をあらわす LTL の式を与える。その後、検証器のプロンプトが現れるので、以下のようにモデルをロードし実行する。

```
RCCS>(load(“echo.ccs”))
```

結果として以下のメッセージが表示される。

```
Emptyness: FALSE
```

先に示した性質は望ましくない性質をあらわしているので、このメッセージが表示され、モデルがただしいとわかる。もし、

Emptyness: TRUE

と表示されたときには違反したアクション列（アクションとは“:”の直前に現れる識別子のこと）を逆順に表示する。

仮にサーバのプログラムを作成したとする。つぎに、この試験のために図 2 のサーバに対応する部分が実際のコードに置き換えられるので 33 行目を (Echo_client) とし、1 行目を追加する（モデル記述時にはコメントにしている）。この行は sev_res というアクションが echos の 28 行目に対応することを表している。試験対象のプログラムにはこの行にブレイクポイントが設定される。プログラムの状態はこのようにブレイクポイントが設定された行での各変数の値の集合として表される。最後にもう一度このモデルを検証することによりソースコードの試験を行う。このために検証ツールをつぎのように起動する。

> ./rccs –f “<> !(yyy=x)” –d echos

echos はバイナリファイルの名前を指定している。この結果は、先のモデル検証のときと同様に

Emptyness: FALSE

または

Emptyness: TRUE

のいずれかのメッセージが表示される。

アクションとソースコードの行を対応させるときにはつぎのことに注意しなければならない。アクションに対応させるソースコード中の行は代入文でなければいけない。アクションが出力アクション（識別子の前に“~”がつく）のときにはその代入文の左辺の変数に引数の値を結び付けプログラムの実行を継続し、入力アクション（識別子の前に“~”がつかない）のときには代入文の右辺の値がそのアクションの引数に結び付けられ、モデルをさらに実行する。現在の版では、変数に具体的な値が必ず与えられることをユーザに期待している。検証器が他の状態を検証するときバックトラックが生じる。このとき、対応するブレイクポイントまでを再実行する。

4. 従来の技術（または機能）との相違

これまでのツールはモデルの検証やプログラムの試験に個別に利用されてきた。ここではプログラム試験にモデルも利用することを試みている。

5. 期待される効果

期待される効果を以下に列挙する。

- モデル検証からプログラム試験までを形式的な手法により実施する。
- モデルを下流工程でも利用することにより、プログラム試験のコストを低減する。
- ソフトウェアに含まれる不具合を開発のより早い段階で発見することを可能にする。
- 担当者の経験に依存せずに検証・試験の質を均一に保つことができる。

6. 普及（または活用）の見通し

現時点では、復旧の見通しは立っていないが、成果物を公開し普及を図ります。

7．開発者名（所属）
永藤　直行　（有限会社 プレシステム）

（参考）http://www4.ocn.ne.jp/~presys/