<<もくじ：Formal Verification Memo

排他制御問題の例

この例は、Basic Spin Manual[*1]で取り上げられているもの[*2]で、もともとは 1966 年に H. Hyman が Communication of the ACM で発表したものだそうです。また、これは、Dekker の初期の解としてしられているそうですが、いわゆる Dekker's algorithm ではありません。

いわゆる Dekker's algorithm と違い、この方法には欠陥があります。その欠陥をいかにして検出するかというのが、この例題の趣旨です。

モデル： hyman1

```
bool want[2];
bool turn;
byte cnt;

proctype P(bool i)
{
  want[i] = 1;
  do
  :: (turn != i) -> (!want[1-i]); turn = i
  :: (turn == i) -> break
  od;

  /* critical section */

  cnt = cnt + 1;
  assert(cnt == 1);

  cnt = cnt - 1;
  want[i] = 0;
}

init { run P(0); run P(1); }
```

グローバル変数、want[n], turn は排他制御のためのものです。各プロセスは、これらの変数を用いて、排他制御を行います。

cnt は、排他が実現されているかどうかをチェックするためのグローバル変数です。

[*1] http://spinroot.com/spin/Man/Manual.html

[*2] このページのコード例は、すべて Basic Spin Manual に載っているものです。

検証プログラムの実行

では、前述のソースファイルで表現された検証モデルを実行してみましょう。

```
% spin -a hyman1
% gcc -DSAFETY -o pan pan.c
% ./pan
pan: assertion violated (cnt==1) (at depth 12)
pan: wrote hyman1.trail

(Spin Version 5.1.4 -- 27 January 2008)
Warning: Search not completed
        + Partial Order Reduction

Full statespace search for:
        never claim             - (none specified)
        assertion violations    +
        cycle checks            - (disabled by -DSAFETY)
        invalid end states      +

State-vector 36 byte, depth reached 23, errors: 1
       60 states, stored
       16 states, matched
       76 transitions (= stored+matched)
        0 atomic steps
hash conflicts:         0 (resolved)

    4.653       memory usage (Mbyte)
```

検証プログラムは、assert 違反があったことを報告しています。エラーシーケンスの追跡は以下のようにして得られます。

```
% spin -t -p hyman1
Starting :init: with pid 0
Starting P with pid 1
  1:    proc  0 (:init:) line  22 "hyman1" (state 1)    [(run P(0))]
Starting P with pid 2
  2:    proc  0 (:init:) line  22 "hyman1" (state 2)    [(run P(1))]
  3:    proc  2 (P) line   7 "hyman1" (state 1) [want[i] = 1]
  4:    proc  2 (P) line   9 "hyman1" (state 2) [((turn!=i))]
```

```
  5:    proc  2 (P) line   9 "hyman1" (state 3) [(!(want[(1-i)]))]
  6:    proc  1 (P) line   7 "hyman1" (state 1) [want[i] = 1]
  7:    proc  1 (P) line  10 "hyman1" (state 5) [((turn==i))]
  8:    proc  2 (P) line   9 "hyman1" (state 4) [turn = i]
  9:    proc  2 (P) line  10 "hyman1" (state 5) [((turn==i))]
 10:    proc  2 (P) line  15 "hyman1" (state 10)        [cnt = (cnt+1)]
 11:    proc  2 (P) line  16 "hyman1" (state 11)        [assert((cnt==1))]
 12:    proc  1 (P) line  15 "hyman1" (state 10)        [cnt = (cnt+1)]
spin: line  16 "hyman1", Error: assertion violated
spin: text of failed assertion: assert((cnt==1))
 13:    proc  1 (P) line  16 "hyman1" (state 11)        [assert((cnt==1))]
spin: trail ends after 13 steps
#processes: 3
                want[0] = 1
                want[1] = 1
                turn = 1
                cnt = 2
 13:    proc  2 (P) line  18 "hyman1" (state 12)
 13:    proc  1 (P) line  18 "hyman1" (state 12)
 13:    proc  0 (:init:) line  22 "hyman1" (state 3) <valid end state>
3 processes created
```

なるほど、なるほど。グローバル変数 turn, want[1-i] を確認するのと、turn の更新がアトミックじゃないので、期待どおりにいかない場合があるんですね。

上のコーディング例では、assert 文を各プロセスのクリティカル・セッション内に書きましたが、独立の監視プロセスにまかせることもできます。

```
bool want[2];
bool turn;
byte cnt;

proctype P(bool i)
{
  want[i] = 1;
  do
  :: (turn != i) -> (!want[1-i]); turn = i
  :: (turn == i) -> break
  od;
  cnt = cnt+1;
  /* critical section */
  cnt = cnt-1;
```

```
  want[i] = 0
}

proctype monitor()
{
  assert(cnt == 0 || cnt == 1)
}

init { run P(0); run P(1); run monitor() }
```

>>次のページ：Dekker0 by Murphi