<<もくじ：Formal Verification Memo

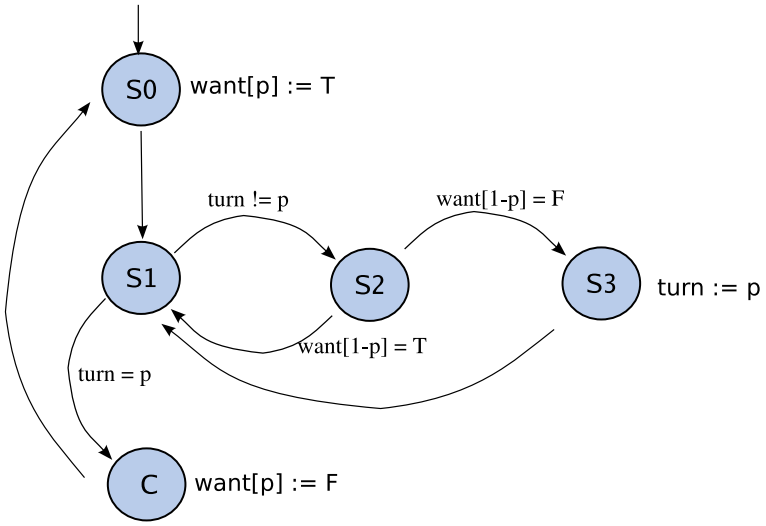
排他制御問題の例（つづき）

次に、Murphi で同じモデルの検証をしてみたいと思います。Murphi におけるモデルの記述は、Promela の場合とは少し違った感じになります。

（ご注意：前ページの promela モデルは、Basic Spin Manual に書いてあったものなので、おそらく間違っていないと思いますが、本ページのモデルは、さっき海野が書いたものなので、嘘っぱちである可能性大です。バグってたり、嘘っぱちだったりする箇所を見つけたら、おしえてください。uDiary の方[*1]につっこんでもらえるとありがたいです。）

モデル：dek0.m

コーディングに先だって、検証対象の状態遷移図を書いてみます。



上図は、先ほどの promela によるコードを見ながら書いたものです。状態遷移の形にできてしまえば、murphi で記述するのは簡単です。

```
-- Filename:    dek0.m
-- Version:     Murphi 3.1
-- Content:     Bad Example for mutual exclusion.

type
  proc_idx_t: 0..1;
      want_t: enum {T, F};
     state_t: enum {S0, S1, S2, S3, C};
```

[*1] http://uhideyuki.sakura.ne.jp/uDiary/?date=20080312#p01

```
var   turn: proc_idx_t;
var   want: Array [ proc_idx_t ] of want_t;
var   stat: Array [ proc_idx_t ] of state_t;


procedure goto(p: proc_idx_t; s: state_t);
begin
  stat[p] := s;
end;

ruleset p: proc_idx_t do

  rule "S0 -> S1 always"
    stat[p] = S0
  ==>
  begin
    want[p] := T;
    goto(p, S1);
  end;

  rule "S1 -> S2 if other's turn"
    stat[p] = S1 & turn != p
  ==>
  begin
    goto(p, S2);
  end;

  rule "S2 -> S3 if another is not wanting"
    stat[p] = S2 & want[1-p] = F
  ==>
  begin
    goto(p, S3);
  end;

  rule "S2 -> S1 if another is wanting"
    stat[p] = S2 & want[1-p] = T
  ==>
  begin
    goto(p, S1);
  end;
```

```
  rule "S3 -> S1 always"
    stat[p] = S3
  ==>
  begin
    turn := p;
    goto(p, S1);
  end;

  rule "S1 -> C (Critical) if its turn"
    stat[p] = S1 & turn = p
  ==>
  begin
    goto(p, C);
  end;

  rule "Critical Session"
    stat[p] = C
  ==>
  begin
    want[p] := F;
    goto(p, S0);
  end;

end;

startstate
begin
  for p : proc_idx_t do
    goto(p, S0);
    want[p] := F;
  end;
  turn := 0;
end;

invariant
  !(stat[0] = C & stat[1] = C);
```

検証プログラムの実行

muphi モデルを実行するには、まず mu で murphi モデル (今回のファイル名は dek0.m) をコンパイルして、さらに出来た C++ プログラムを (g++ などで) コンパイルします。

```
% mu dek0.m
% g++ -I ${Murphi3.1path}/include -o dek0 dek0.C
% ./dek0
This program should be regarded as a DEBUGGING aid, not as a
certifier of correctness.
Call with the -l flag or read the license file for terms
and conditions of use.
Run this program with "-h" for the list of options.

Bugs, questions, and comments should be directed to
"murphi@verify.stanford.edu".

Murphi compiler last modified date: Jan 29 1999
Include files   last modified date: Jan 29 1999
============================================================================

============================================================================
Murphi Release 3.1
Finite-state Concurrent System Verifier.

Copyright (C) 1992 - 1999 by the Board of Trustees of
Leland Stanford Junior University.

============================================================================

Protocol: dek0

Algorithm:
        Verification by breadth first search.
        with symmetry algorithm 3 -- Heuristic Small Memory Normalization
        with permutation trial limit 10.

Memory usage:
```

```
        * The size of each state is 40 bits (rounded up to 8 bytes).
        * The memory allocated for the hash table and state queue is
          8 Mbytes.
          With two words of overhead per state, the maximum size of
          the state space is 476219 states.
           * Use option "-k" or "-m" to increase this, if necessary.
        * Capacity in queue for breadth-first search: 47621 states.
           * Change the constant gPercentActiveStates in mu_prolog.inc
             to increase this, if necessary.

Warning: No trace will not be printed in the case of protocol errors!
         Check the options if you want to have error traces.

============================================================================

Result:

        Invariant "Invariant 0" failed.

State Space Explored:

        203 states, 202 rules fired in 0.10s.

Analysis of State Space:

        There are rules that are never fired.
        If you are running with symmetry, this may be why.  Otherwise,
        please run this program with "-pr" for the rules information.
```

やはり、Invariant (不変な表明) 違反が検出されました。
失敗ケースのトレースは、-tv オプション付きで実行することで得られます。

```
% ./dek0 -tv
... (略) ...
Startstate Startstate 0 fired.
turn:0
want[0]:F
want[1]:F
stat[0]:S0
stat[1]:S0
----------
```

```
Rule S0 -> S1 always, p:1 fired.
want[1]:T
stat[1]:S1
----------

Rule S1 -> S2 if other's turn, p:1 fired.
stat[1]:S2
----------

Rule S2 -> S3 if another is not wanting, p:1 fired.
stat[1]:S3
----------

Rule S0 -> S1 always, p:0 fired.
want[0]:T
stat[0]:S1
----------

Rule S1 -> C (Critical) if its turn, p:0 fired.
stat[0]:C
----------

Rule S3 -> S1 always, p:1 fired.
turn:1
stat[1]:S1
----------

Rule S1 -> C (Critical) if its turn, p:1 fired.
The last state of the trace (in full) is:
turn:1
want[0]:T
want[1]:T
stat[0]:C
stat[1]:C
----------

End of the error trace.
... (略) ...
```

検出された失敗ケースは、前ページ (Dekker0 by Spin) のモデルで検出されたものと同じですね。turn != p & want[1-p] = T を確認してから、turn := p の操作の間に、相手プロセスが割り込んでしまうことで、意図しない動作となってしまっていました。