`smcgen`: Statistical Model-Checking Generator

Andrew Butterfield Jim Woodcock

March 23, 2018

Contents

Chapter 1

smcgen Application

1.1 Main Program

```haskell
module Main where
import Hack

main :: IO ()
main = putStrLn hack
```

Chapter 2

smcgen Libraries

2.1 Hacking

```
module Hack
where
```

Currently we are hacking ways to generalise `Flash.prism` from having parameter `b` fixed equal to 3.

```
hack = "hacking Flash.prism"
```

Chapter 3

smcgen Tests

```
module Main where

import Test.HUnit
import Test.Framework as TF (defaultMain, testGroup, Test)
import Test.Framework.Providers.HUnit (testCase)

import Hack

main = defaultMain tests

tests :: [TF.Test]
tests
 =  [ testCase "1+1=2" (1+1 @?= 2)
    , testCase "hack" (hack @?= "hacking Flash.prism")
    ]
```

Chapter 4

Model Files

4.1 Flash Prism Model

The original model developed by Jim Woodcock and Andrew Butterfield in March 2018 in TCD. Here the b parameter was fixed at 3, as the number of terms in some commands depend on this value.

```
dtmc

const int b=3; // Block Count: Our problematic parameter

const int p; // Pages per Block
const int c; // Number of page writes between wear levelling
const int w; // Maximum wear tolerance (no. of erasures)
const int MAXDIFF; // Maximum desired difference in wear across blocks.

// control flow
const int INIT = 1;     // startup
const int WRITE = 2;    // page writes
const int SELECT = 3;   // wear-levelling
const int FINISH = 4;   // done: mempory full or worn out

module Flash

// fm_clean_i, for i in 1..b - the number of clean pages in block i
fm_clean_1: [0..p];
fm_clean_2: [0..p];
fm_clean_3: [0..p];

// fm_erase_i, i in 1..b, the number of times block i has been erased.
fm_erase_1: [0..w];
fm_erase_2: [0..w];
fm_erase_3: [0..w];

pc: [INIT..FINISH] init INIT;

// count the number of page writes done since last wear-levelling.
i: [0..c] init 0;

[] pc=INIT ->
  (fm_clean_1'=p) & (fm_clean_2'=p) & (fm_clean_3'=p) &
  (fm_erase_1'=0) & (fm_erase_2'=0) & (fm_erase_3'=0) &
  (pc'=WRITE);
[] pc=WRITE & i<c & writeable!=0 ->
  (fm_clean_1>0?1/writeable:0): (fm_clean_1'=fm_clean_1-1) & (i'=i+1) +
  (fm_clean_2>0?1/writeable:0): (fm_clean_2'=fm_clean_2-1) & (i'=i+1) +
  (fm_clean_3>0?1/writeable:0): (fm_clean_3'=fm_clean_3-1) & (i'=i+1);
```

```
[] pc=WRITE & i<c & writeable=0 -> (pc'=FINISH);
[] pc=WRITE & i=c -> (pc'=SELECT);
[] pc=SELECT & (candidates=0 | !can_erase) -> (pc'=FINISH);
[] pc=SELECT & candidates!=0 & can_erase ->
  (cand_1_2 ? 1/candidates : 0): (fm_clean_2'=fm_clean_2-dirty_1) &
                                 (fm_clean_1'=p) & (fm_erase_1'=fm_erase_1+1) &
                                 (i'=0) & (pc'=WRITE) +
  (cand_1_3 ? 1/candidates : 0): (fm_clean_3'=fm_clean_3-dirty_1) &
                                 (fm_clean_1'=p) & (fm_erase_1'=fm_erase_1+1) &
                                 (i'=0) & (pc'=WRITE) +
  (cand_2_1 ? 1/candidates : 0): (fm_clean_1'=fm_clean_1-dirty_2) &
                                 (fm_clean_2'=p) & (fm_erase_2'=fm_erase_2+1) &
                                 (i'=0) & (pc'=WRITE) +
  (cand_2_3 ? 1/candidates : 0): (fm_clean_3'=fm_clean_3-dirty_2) &
                                 (fm_clean_2'=p) & (fm_erase_2'=fm_erase_2+1) &
                                 (i'=0) & (pc'=WRITE) +
  (cand_3_1 ? 1/candidates : 0): (fm_clean_1'=fm_clean_1-dirty_3) &
                                 (fm_clean_3'=p) & (fm_erase_3'=fm_erase_3+1) &
                                 (i'=0) & (pc'=WRITE) +
  (cand_3_2 ? 1/candidates : 0): (fm_clean_2'=fm_clean_2-dirty_3) &
                                 (fm_clean_3'=p) & (fm_erase_3'=fm_erase_3+1) &
                                 (i'=0) & (pc'=WRITE);
[] pc=FINISH -> true;


endmodule

// a block is writeable if it has at least one clean page
// We need to know how many of these there are.
formula writeable = (fm_clean_1!=0 ? 1 : 0) +  (fm_clean_2!=0 ? 1 : 0)
                  + (fm_clean_3!=0 ? 1 : 0);

// dirty_i, for i in 1..b - number of dirty pages in block i

formula dirty_1 = p-fm_clean_1;
formula dirty_2 = p-fm_clean_2;
formula dirty_3 = p-fm_clean_3;

// cand_i_j, for i,j in 1..b, i /= j
//  block i is dirty but there is space in block j for its pages

formula cand_1_2 = dirty_1>0 & fm_clean_2 >= dirty_1;
formula cand_1_3 = dirty_1>0 & fm_clean_3 >= dirty_1;
formula cand_2_1 = dirty_2>0 & fm_clean_1 >= dirty_2;
formula cand_2_3 = dirty_2>0 & fm_clean_3 >= dirty_2;
formula cand_3_1 = dirty_3>0 & fm_clean_1 >= dirty_3;
formula cand_3_2 = dirty_3>0 & fm_clean_2 >= dirty_3;

// the number of ways in which we can relocate dirty pages from one block
// to another so we can erase (clean) the first block.
formula candidates =
  (cand_1_2?1:0) + (cand_1_3?1:0) + (cand_2_1?1:0) +
  (cand_2_3?1:0) + (cand_3_1?1:0) + (cand_3_2?1:0);

// true when it is still possibe to erase ANY block,
// without exceeding the maximum allowable erase operations.
formula can_erase = fm_erase_1<w & fm_erase_2<w & fm_erase_3<w;

// diff_i_j, for i,j in 1..b, i /= j
// the difference in number of erasure of blocks i and j
formula diff_1_2 = fm_erase_1-fm_erase_2;
formula diff_1_3 = fm_erase_1-fm_erase_3;
formula diff_2_1 = fm_erase_2-fm_erase_1;
formula diff_2_3 = fm_erase_2-fm_erase_3;
formula diff_3_1 = fm_erase_3-fm_erase_1;
formula diff_3_2 = fm_erase_3-fm_erase_2;


// true if difference in wear equals some limit.
formula toobig =
  diff_1_2 >= MAXDIFF |
  diff_1_3 >= MAXDIFF |
  diff_2_1 >= MAXDIFF |
  diff_2_3 >= MAXDIFF |
  diff_3_1 >= MAXDIFF |
  diff_3_2 >= MAXDIFF;
```

4.2 Flash Prism Properties

```
!E [ F "deadlock" ];
E [ F toobig ];
P =? [ F toobig ];
```

Bibliography