`smcgen`: Statistical Model-Checking Generator

Andrew Butterfield Jim Woodcock

March 26, 2018

0.1 Abstraction, Level One


```
module Abs1 where
import Data.List
```

Here we take the `Hack` code and abstract it, by developing generic functions.

The key issue with `Flash.prism` was the presence of two arrays of size b. The trick we had to implement was to declare each array component as a unique variable, with its index as part of its name. We still have to do this to produce Prism code, but now we should be able to describe this much more abstractly.

In this module, we now have the `Hack` code as comments, rather than the corresponding Prism code.

```
hack b
 | b < 2 = putStrLn "smcgen with b less than two is somewhat pointless"
 | otherwise = writeFile ("models/gen/Flash"++show b++".prism") $ prismcode b

prismcode b
 = unlines $ intercalate [""]
     [ sem, params b, control
     , mdl b, vars b
     , step1 b, step2 b, step3, step4, step5, step6 b, step7, endm
     , writeable b, dirty b
     , cand b, candidates b, can_erase b
     , diff b, toobig b ]
```

We will defer this until later.

```
abs1 = putStrLn "Abs1: not much to see yet."
```

```
sem = ["dtmc"]
```

Looks like an enumeration for now:

```
data SMCSem = DTMC | SMCSother deriving (Eq,Show,Read)
```

```
params b
  = [ "const int b="++show b++"; // 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
  = [ "// control flow"
    , "const int INIT = 1;    // startup"
    , "const int WRITE = 2;   // page writes"
    , "const int SELECT = 3;  // wear-levelling"
    , "const int FINISH = 4;  // done: memory full or worn out"
    ]
```

These are constant/parameter declarations: names, types.

```
type Param = (Name,Type,Maybe Value)
type Name = String
data Type = TypInt | TypOther deriving (Eq, Show, Read)
data Value = I Int | TO deriving (Eq, Show, Read)
```

```
mdl b = [ "module Flash"++show b ]
```

Defer until the shape of things is better understood.

```
vars :: Int -> [String]
vars b
  = ( ("// fm_clean_i, for i in 1.."++show b++" - no of clean pages in block i")
    : map (idecl "fm_clean_" ": [0..p];") [1..b] )
    ++
    ( ("// fm_erase_i, i in 1.."++show b++", no of times block i has been erased.")
    : map (idecl "fm_erase_" ":[0..w];") [1..b] )
    ++
    [ "pc: [INIT..FINISH] init INIT; // program counter"
    , "i: [0..c] init 0; // number of writes done since last wear-levelling."
    ]

idecl root typ i = root ++ show i ++ typ

step1 b
  =    "// Step 1"
    :  "[] pc=INIT ->"
    :  (map (iinit "  (fm_clean_" "'=p) &") [1..b])
    ++ (map (iinit "  (fm_erase_" "'=p) &") [1..b])
    ++ [ "  (pc'=WRITE);" ]

iinit root val i = root ++ show i ++ val

step2 b
  =    "// Step 2"
    : "[] pc=WRITE & i<c & writeable!=0 ->"
    : map (iwrite b) [1..b]

iwrite b i
  =     "  (fm_clean_"++show i
     ++ ">0?1/writeable:0): (fm_clean_"++show i
     ++ "'=fm_clean_"++show i
     ++ "-1) & (i'=i+1)"
     ++ addend b i

addend b i = if i == b then ";" else " +"

step3
  = [ "// Step 3"
    , "[] pc=WRITE & i<c & writeable=0 -> (pc'=FINISH);"
    ]

step4
  = [ "// Step 4"
    , "[] pc=WRITE & i=c -> (pc'=SELECT);"
    ]
```

```
step5
  = [ "// Step 5"
    , "[] pc=SELECT & (candidates=0 | !can_erase) -> (pc'=FINISH);"
    ]

step6 b
  =    "// Step 6"
    :  "[] pc=SELECT & candidates!=0 & can_erase ->"
    : (concat $ map (ierase $ lst b) $ ndTuples b)

ndTuples n = filter nonDiag $ [(i,j)| i <- [1..n], j <- [1..n]]
nonDiag (i,j) = i /= j

lst n = (n,n-1)

ierase last curr@(from,to)
  = [    "  (cand_"++show from++"_"++show to
      ++ " ? 1/candidates : 0): (fm_clean_"++show to
      ++ "'=fm_clean_"++show to++"-dirty_"++show from++") &"
    ,    "                                  (fm_clean_"++show from
      ++ "'=p) & (fm_erase_"++show from++"'=fm_erase_"++show from++"+1) &"
    ,    "                                  (i'=0) & (pc'=WRITE)"
      ++ addend last curr
    ]

step7
  = [ "// Step 7"
    , "[] pc=FINISH -> true;"
    ]

endm = [ "endmodule" ]

writeable b
  =   "// a block is writeable if it has at least one clean page"
    : "// We need to know how many of these there are."
    : "formula writeable ="
    : map (iwriteable b) [1..b]

iwriteable b i = "  (fm_clean_"++show i++"!=0 ? 1 : 0)" ++ addend b i

dirty b
  =   ( "// dirty_i, for i in 1.."++show b
        ++" - number of dirty pages in block i" )
    : map (idirty b) [1..b]

idirty b i = "formula dirty_"++show i++" = p-fm_clean_"++show i++";"

cand b
  =    ( "// cand_i_j, for i,j in 1.."++show b++", i /= j" )
    :  "//  block i is dirty but there is space in block j for its pages"
    : ( map icand $ ndTuples b)

icand (from,to)
 =    "formula cand_"++show from++"_"++show to
   ++ " = dirty_"++show from
   ++ ">0 & fm_clean_"++show to++" >= dirty_"++show from++";"
```

```
candidates b
  =  "// 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 ="
   : ( map (icandidate $ lst b) $ ndTuples b )

icandidate last curr@(from,to)
  =  "  (cand_"++show from++"_"++show to++"?1:0)" ++ addend last curr

can_erase b
  =   "// true when it is still possibe to erase ANY block,"
    : "// without exceeding the maximum allowable erase operations."
    : "formula can_erase ="
    : map (icanerase b) [1..b]

icanerase b i = "  fm_erase_"++show i++"<w" ++ andend b i

andend b i = if b == i then ";" else " &"

diff b
  =  ( "// diff_i_j, for i,j in 1.."++show b++", i /= j" )
   : "// the difference in number of erasure of blocks i and j"
   : ( map idiff $ ndTuples b )

idiff (i,j)
  =    "formula diff_"++show i++"_"++show j
    ++ " = fm_erase_"++show i++"-fm_erase_"++show j++";"

toobig b
  =   "// true if difference in wear equals some limit."
    : "formula toobig ="
    : ( map (itoobig $ lst b) $ ndTuples b )

itoobig last curr@(i,j)
  = "  diff_"++show i++"_"++show j++" >= MAXDIFF" ++ orend last curr

orend last curr = if last == curr then ";" else " |"
```