Heading one

## Heading two

And some text!

%%VDM%%

module M

exports all

definitions

types

Quadrant = seq of M

-- inv Q == forall a in set elems Q &

-- (not exists b in set elems Q \ {a} &

-- (b.start >= a.start and b.start <= a.stop) or

-- (b.stop >= a.start and b.stop <= a.stop))

;

M ::

type: <USED> | <FREE>

start: nat

stop: nat

inv mk\_M(-, a, b) == (b >= a)

state Memory of

rseed: nat

Q3: Quadrant

Q4: Quadrant

init q ==

q = mk\_Memory(

87654321,

[mk\_M(<FREE>, 0, MAXMEM-1)],

[mk\_M(<FREE>, 0, MAXMEM-1)])

end

values

MAXMEM = 10000;

CHUNK = 1000;

functions

fits: nat1 \* Quadrant -> nat1

fits(size, Q) ==

cases Q:

[] -> MAXMEM + 1,

[h] ^ tail ->

if h.type = <FREE> and h.stop - h.start + 1 >= size

then h.stop - h.start + 1 else fits(size, tail)

end;

least: nat1 \* nat1 -> nat1

least(a, b) == if a < b then a else b;

bestfit: nat1 \* Quadrant -> nat1

bestfit(size, Q) ==

cases Q:

[] -> MAXMEM + 1, -- as we're looking for the smallest

[h] ^ tail ->

if h.type = <FREE> and h.stop - h.start + 1 >= size

then least(h.stop - h.start + 1, bestfit(size, tail))

else bestfit(size, tail)

end;

add: nat1 \* nat1 \* Quadrant -> Quadrant

add(size, hole, Q) ==

cases Q:

[h] ^ tail ->

if h.type = <FREE> and h.stop - h.start + 1 = hole then

if hole = size then

[mk\_M(<USED>, h.start, h.stop)] ^ tail

else

[mk\_M(<USED>, h.start, h.start + size - 1),

mk\_M(<FREE>, h.start + size, h.stop)] ^ tail

else

[h] ^ add(size, hole, tail),

others -> Q

end

pre hole >= size;

combine: Quadrant -> Quadrant

combine(Q) ==

cases Q:

[h1, h2] ^ tail ->

if h1.type = <FREE> and h2.type = <FREE>

then combine([mk\_M(<FREE>, h1.start, h2.stop)] ^ tail)

else [h1] ^ combine(tl Q),

others -> Q

end;

delete: M \* Quadrant -> Quadrant

delete(item, Q) ==

if hd Q = item

then

combine([mk\_M(<FREE>, item.start, item.stop)] ^ tl Q)

else

[hd Q] ^ delete(item, tl Q);

fragments: Quadrant -> nat

fragments(Q) ==

card {x | x in set elems Q & x.type = <FREE>} - 1;

operations

seed: nat1 ==> ()

seed(n) == rseed := n;

rand: nat1 ==> nat1

rand(n) ==

(

rseed := (rseed \* 69069 + 5) mod 4294967296;

return rseed mod n + 1;

);

firstFit: nat1 ==> bool

firstFit(size) ==

(

let q4 = fits(size, Q4) in

if q4 <= MAXMEM

then Q4 := add(size, q4, Q4)

else let q3 = fits(size, Q3) in

if q3 <= MAXMEM

then Q3 := add(size, q3, Q3)

else return false;

return true;

);

bestFit: nat1 ==> bool

bestFit(size) ==

(

let q4 = bestfit(size, Q4) in

if q4 <= MAXMEM

then Q4 := add(size, q4, Q4)

else let q3 = bestfit(size, Q3) in

if q3 <= MAXMEM

then Q3 := add(size, q3, Q3)

else return false;

return true;

);

reset: () ==> ()

reset() ==

(

Q3 := [mk\_M(<FREE>, 0, MAXMEM-1)];

Q4 := [mk\_M(<FREE>, 0, MAXMEM-1)];

);

tryFirst: nat ==> nat

tryFirst(loops) ==

(

dcl count:int := 0;

reset();

while count < loops and firstFit(rand(CHUNK)) do

(

if rand(2) = 1

then Q3 := delete(Q3(rand(len Q3)), Q3)

else Q4 := delete(Q4(rand(len Q4)), Q4);

count := count + 1;

);

return count;

-- return fragments(Q3) + fragments(Q4);

);

tryBest: nat ==> nat

tryBest(loops) ==

(

dcl count:int := 0;

reset();

while count < loops and bestFit(rand(CHUNK)) do

(

if rand(2) = 1

then Q3 := delete(Q3(rand(len Q3)), Q3)

else Q4 := delete(Q4(rand(len Q4)), Q4);

count := count + 1;

);

return count;

-- return fragments(Q3) + fragments(Q4);

);

main: nat1 \* nat1 ==> seq of (<BEST> | <FIRST> | <SAME>)

main(tries, loops) ==

(

dcl result: seq of (<BEST> | <FIRST> | <SAME>) := [];

for i = 1 to tries do

(

dcl best:int, first:int;

seed(i);

best := tryBest(loops);

seed(i);

first := tryFirst(loops);

if best = first

then result := result ^ [<SAME>]

elseif best > first

then result := result ^ [<BEST>]

else result := result ^ [<FIRST>];

);

return result;

)

end M

%%VDM%%

%%VDM%%

module B

end B

%%VDM%%