Formal Specification Examples

Rui Carlos A. Gonçalves (41031)

January 6, 2007

Contents

1. Problem 2 - Modelling with Sequences

1. The squadrons of a large military contingent can be in two possible states: operational or resting. Once a squadron is in operation for some time limit, it must return to the military base and change to the resting state, being replaced by the squadron which has the longest rest.

   Use `seq of` modelling to complete the ... in

   ```
   Contingent :: operational: ...
                 resting: ...
   ```

   and specify the operation of squadron replacement outlined above.

2. Use sequence comprehension to model the following requirements from a mobile phone manufacturer:

   > (...) For each list of calls stored in the mobile phone (eg. numbers dialed, SMS messages, lost calls), the store operation should work in a way such that (a) the more recently a call is made the more accessible it should be; (b) no number appears twice in a list; (c) only the last 10 entries in each list are stored.

3. Transliterate the following (polymorphic) function definition to VDM-SL syntax

   ```
   filter p [] = []
   filter p (a:l) | p a       = a : (filter p l)
                  | otherwise = (filter p l)
   ```

   and check its behaviour in the toolbox.

1.1. Pretty-print

types

1.0 $Contingent :: operational : \mathbb{Z}^*$
.1 $resting : \mathbb{Z}^*$

.2 inv $c \triangleq \mathsf{elems}\ (c.operational) \cap \mathsf{elems}\ (c.resting) = \{\}$

functions

2.0 $replace : Contingent \to Contingent$
.1 $replace\,(x) \triangleq$
.2 mk-$Contingent\,(\mathsf{tl}\ (x.operational) \curvearrowright [\mathsf{hd}\ (x.resting)],$
.3 $\mathsf{tl}\ (x.resting \curvearrowright [\mathsf{hd}\ (x.operational)]))$
.4 pre $x.operational \neq [] \wedge x.resting \neq []$;

3.0 $addcall : \mathbb{Z} \times \mathbb{Z}^* \to \mathbb{Z}^*$

.1 $addcall\,(n, l) \triangleq$

.2 $\quad$ if card $(\text{elems}\ (l \frown [n])) = 10 \vee \text{len}\ (l) < 10$

.3 $\quad$ then $[n] \frown [l\,(i) \mid i \in \text{inds}\ l \cdot l\,(i) \neq n]$

.4 $\quad$ else $[n] \frown [l\,(i) \mid i \in \{1, \ldots, 9\}]$

.5 pre $\text{len}\ (l) \leq 10 \wedge \text{len}\ (l) = \text{card}\ (\text{elems}\ (l))$;

4.0 $filter[@a] : (@a \to \mathbb{B}) \times @a^* \to @a^*$

.1 $filter\,(p, l) \triangleq$

.2 $\quad$ cases l :

.3 $\quad\quad [] \to [],$

.4 $\quad\quad [h] \frown t \to$ if $p\,(h)$

.5 $\quad\quad\quad\quad$ then $[h] \frown filter[@a]\,(p, t)$

.6 $\quad\quad\quad\quad$ else $filter[@a]\,(p, t)$

.7 $\quad$ end;

5.0 $lessSix : \mathbb{Z} \to \mathbb{B}$

.1 $lessSix\,(x) \triangleq$

.2 $\quad (x < 6);$

6.0 $prefixA : \text{char}^* \to \mathbb{B}$

.1 $prefixA\,(c) \triangleq$

.2 $\quad (\text{hd}\ (c) = \text{'}a\text{'})$

values

7.0 $c1 = \text{mk-}Contingent\,([1, 2, 3], [4, 5, 6, 7, 8, 9, 0]);$

8.0 $c2 = \text{mk-}Contingent\,([], [2, 3, 4, 5]);$

9.0 $calls = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$

1.2. Source code

```
--------------------------------------------------------------------------------
-- Rui Carlos A. Goncalves
-- LMCC - MFP I
-- 2006/07
-- Problema 2
--------------------------------------------------------------------------------
```

```
types
Contingent::operational:seq of int
              resting    :seq of int
inv c==elems(c.operational) inter elems(c.resting)={};


functions
-- 1
replace:Contingent->Contingent
replace(x)==mk_Contingent(tl(x.operational)^[hd(x.resting)],
                          tl(x.resting^[hd(x.operational)]))
pre x.operational<>[] and x.resting<>[];

-- 2
addcall:int*seq of int->seq of int
addcall(n,l)==if card(elems(l^[n]))=10 or len(l)<10
                 then [n]^[l(i)|i in set inds l & l(i)<>n]
                 else [n]^[l(i)|i in set {1,...,9}]
pre len(l)<=10 and len(l)=card(elems(l));

--3
filter[@a]:(@a->bool)*seq of @a->seq of @a
filter(p,l)==cases l:
                []->[],
                [h]^t->if p(h)
                          then [h]^filter[@a](p,t)
                          else filter[@a](p,t)
             end;

lessSix:int->bool
lessSix(x)==(x<6);

prefixA:seq of char->bool
prefixA(c)==(hd(c)='a');


values
c1=mk_Contingent([1,2,3],[4,5,6,7,8,9,0]);
c2=mk_Contingent([],[2,3,4,5]);

calls=[1,2,3,4,5,6,7,8,9,10];
```

2. Problem 3 - Modelling with Sets

To improve the operation of fire-brigades fighting forest fires in the hot season, the National Service for Civil Protection is developing a mobile station-based geographical information system which keeps track of rural area accessibility.

In this exercise we are concerned with the design of the central database component which records road maps (distances, road alternatives and so on) - a component which we are going to model with sets.

We start from a very simple model (to be improved at a later stage): a road-map is a directed graph whose arcs $A \rightarrow B$ record the *you can drive from A to B* relation:

```
Map = set of Path;
Path :: From: Location
        To: Location;
Location = token;
```

1. Specify functions in VDM-SL to obtain:
   1. The symmetric of a map (ie. the map of the "all ways back");
   2. The symmetric closure of a map (ie. the map and all its ways back);
   3. The composition of two maps (ie. the map whose paths are obtained by arriving at a location accessible in the first map and proceeding from there to another location accessible in the second map);
   4. The reflection of a map (set of all trivial paths between a location in the map and itself);
   5. The transitive closure of a map (set of all direct or composed paths in a map).
2. The combined symmetric, transitive and reflexive closure of a map is a map which records an equivalence relation on locations. What is the informal meaning of this equivalence?
3. Upgrade your `Map` model by adding further information to each path, namely the `name` of the road, the `distance` involved and its (average) `speed`:

   ```
   UMap = set of UPath;
   UPath :: From: Location
            Info: PathInfo
              To: Location;
   PathInfo ::     name: token
                distance: real
                   speed: real;
   Location = token;
   ```

   Scale up the function definitions above from `Map` to `UMap`. Clearly, you have to pay attention to the handling of `PathInfos`, by eg. summing up distances, computing average speeds and concatenating road names. (**Hint**: recall the notion of a *weighted graph*.)

2.1. Pretty-print

types

10.0 $Location = \mathsf{token};$

11.0 $Map = Path\text{-}\mathsf{set};$

12.0 $Path :: From : Location$
.1 $\quad To : Location;$

13.0 $UMap = UPath\text{-}\mathsf{set};$

14.0 $UPath :: From : Location$
.1 $\quad Info : PathInfo$
.2 $\quad To : Location;$

15.0 $PathInfo :: name : \mathsf{token}$
.1 $\quad distance : \mathbb{R}$
.2 $\quad speed : \mathbb{R}$

functions

16.0 $symmetric : Map \to Map$
.1 $symmetric\,(m) \triangleq$
.2 $\quad \{\mathsf{mk}\text{-}Path\,(x.To, x.From) \mid x \in m\};$

17.0 $symClosure : Map \to Map$
.1 $symClosure\,(m) \triangleq$
.2 $\quad symmetric\,(m) \cup m;$

18.0 $composition : Map \times Map \to Map$
.1 $composition\,(m1, m2) \triangleq$
.2 $\quad \{\mathsf{mk}\text{-}Path\,(x, z) \mid$
.3 $\quad\quad \mathsf{mk}\text{-}Path\,(x, y) \in m1,$
.4 $\quad\quad \mathsf{mk}\text{-}Path\,(y, z) \in m2\};$

19.0 $reflection : Map \to Map$
.1 $reflection\,(m) \triangleq$
.2 $\quad \{\mathsf{mk}\text{-}Path\,(x, x) \mid \mathsf{mk}\text{-}Path\,(x, y) \in m\} \cup$
.3 $\quad \{\mathsf{mk}\text{-}Path\,(y, y) \mid \mathsf{mk}\text{-}Path\,(x, y) \in m\} \cup$
.4 $\quad m;$

```
20.0  transClosure : Map → Map
 .1   transClosure (m) ≜
 .2     if m = composition (m, m) ∪ m
 .3     then m
 .4     else transClosure (composition (m, m) ∪ m);

21.0  usymmetric : UMap → UMap
 .1   usymmetric (m) ≜
 .2     {mk-UPath (x.To, x.Info, x.From) | x ∈ m};

22.0  usymClosure : UMap → UMap
 .1   usymClosure (m) ≜
 .2     usymmetric (m) ∪ m;

23.0  ucomposition : UMap × UMap → UMap
 .1   ucomposition (m1, m2) ≜
 .2     {mk-UPath (x, mk-PathInfo (mk-token ({inf1, inf2}),
 .3                                d1 + d2,
 .4                                (s1 + s2)/2),
 .5                z) |
 .6          mk-UPath (x, mk-PathInfo (inf1, d1, s1), y) ∈ m1,
 .7          mk-UPath (y, mk-PathInfo (inf2, d2, s2), z) ∈ m2};

24.0  ureflection : UMap → UMap
 .1   ureflection (m) ≜
 .2     {mk-UPath (x, mk-PathInfo (mk-token ("NULL"), 0, 0), x) |
 .3          mk-UPath (x, inf, y) ∈ m} ∪
 .4     {mk-UPath (y, mk-PathInfo (mk-token ("NULL"), 0, 0), y) |
 .5          mk-UPath (x, inf, y) ∈ m} ∪
 .6     m;

25.0  utransClosure : UMap → UMap
 .1   utransClosure (m) ≜
 .2     if m = ucomposition (m, m) ∪ m
 .3     then m
 .4     else utransClosure (ucomposition (m, m) ∪ m)

values

26.0  l1 : token = mk-token ('A');
```

27.0 $l2 : \mathsf{token} = \mathsf{mk\text{-}token}\,('B');$

28.0 $l3 : \mathsf{token} = \mathsf{mk\text{-}token}\,('C');$

29.0 $l4 : \mathsf{token} = \mathsf{mk\text{-}token}\,('D');$

30.0 $l5 : \mathsf{token} = \mathsf{mk\text{-}token}\,('E');$

31.0 $l6 : \mathsf{token} = \mathsf{mk\text{-}token}\,('F');$

32.0 $m1 : Map = \{\mathsf{mk\text{-}}Path\,(l1, l2), \mathsf{mk\text{-}}Path\,(l3, l2), \mathsf{mk\text{-}}Path\,(l2, l4)\};$

33.0 $m2 : Map = \{\mathsf{mk\text{-}}Path\,(l2, l5), \mathsf{mk\text{-}}Path\,(l4, l6)\};$

34.0 $um1 : UMap = \{\mathsf{mk\text{-}}UPath\,(l1, \mathsf{mk\text{-}}PathInfo\,(l1, 2, 3), l2),$
.1 $\mathsf{mk\text{-}}UPath\,(l3, \mathsf{mk\text{-}}PathInfo\,(l3, 3, 4), l2),$
.2 $\mathsf{mk\text{-}}UPath\,(l2, \mathsf{mk\text{-}}PathInfo\,(l2, 1, 4), l4)\}$

2.2. Source code

```
-------------------------------------------------------------------------------
-- Rui Carlos A. Goncalves
-- LMCC - MFP I
-- 2006/07
-- Problema 3
-------------------------------------------------------------------------------

types
Location = token;

Map = set of Path;
Path :: From : Location
        To : Location;

UMap = set of UPath;
UPath :: From: Location
         Info: PathInfo
         To: Location;
PathInfo :: name: token
            distance: real
            speed: real;
```

```
functions
-- 1
-- 1.1
symmetric:Map->Map
symmetric(m)=={mk_Path(x.To,x.From)|x in set m};

-- 1.2
symClosure:Map->Map
symClosure(m)==symmetric(m) union m;

-- 1.3
composition:Map*Map->Map
composition(m1,m2)=={mk_Path(x,z)
                    |mk_Path(x,y) in set m1
                    ,mk_Path(y,z) in set m2};

-- 1.4
reflection:Map->Map
reflection(m)=={mk_Path(x,x)|mk_Path(x,y) in set m}
         union {mk_Path(y,y)|mk_Path(x,y) in set m}
         union m;

-- 1.5
transClosure:Map->Map
transClosure(m)==if m=composition(m,m) union m
                 then m
                 else transClosure(composition(m,m) union m);

-- 2
-- Se no grafo original e possivel chegar de um nodo a outro, no final vai
--   existir uma ligacao directa entre eles nos dois sentidos.

-- 3
-- 3.1
usymmetric:UMap->UMap
usymmetric(m)=={mk_UPath(x.To,x.Info,x.From)|x in set m};

-- 3.2
usymClosure:UMap->UMap
usymClosure(m)==usymmetric(m) union m;

-- 3.3
ucomposition:UMap*UMap->UMap
ucomposition(m1,m2)=={mk_UPath(x,mk_PathInfo(mk_token({inf1,inf2})
                                             ,d1+d2
                                             ,(s1+s2)/2)
                               ,z)
                     |mk_UPath(x,mk_PathInfo(inf1,d1,s1),y) in set m1
                     ,mk_UPath(y,mk_PathInfo(inf2,d2,s2),z) in set m2};

-- 3.4
ureflection:UMap->UMap
```

```
ureflection(m)=={mk_UPath(x,mk_PathInfo(mk_token("NULL"),0,0),x)
                |mk_UPath(x,inf,y) in set m}
          union {mk_UPath(y,mk_PathInfo(mk_token("NULL"),0,0),y)
                |mk_UPath(x,inf,y) in set m}
          union m;

-- 3.5
utransClosure:UMap->UMap
utransClosure(m)==if m=ucomposition(m,m) union m
                    then m
                    else utransClosure(ucomposition(m,m) union m);


values
l1:token=mk_token('A');
l2:token=mk_token('B');
l3:token=mk_token('C');
l4:token=mk_token('D');
l5:token=mk_token('E');
l6:token=mk_token('F');
m1:Map={mk_Path(l1,l2),mk_Path(l3,l2),mk_Path(l2,l4)};
m2:Map={mk_Path(l2,l5),mk_Path(l4,l6)};

um1:UMap={mk_UPath(l1,mk_PathInfo(l1,2,3),l2),
          mk_UPath(l3,mk_PathInfo(l3,3,4),l2),
          mk_UPath(l2,mk_PathInfo(l2,1,4),l4)};
```

3. Problem 6 - Modelling with Mappings

1. Complete the "....." in the VDM-SL model below of a toy *bank account management system* (`BAMS`) in which accounts (`Account`) are uniquely identified by account identifiers (`AccId`). Every account records both a set of account holders (`set of AccHolder`) and the account's current balance (`Amount`):

```
types

BAMS = map AccId to Account;
Account :: H: set of AccHolder
           B: Amount;

AccId     = seq of char;
AccHolder = seq of char;
Amount = int;

functions

Init : () -> BAMS
Init () == {|->};

OpenAccount : BAMS * AccId * set of AccHolder * Amount -> BAMS
OpenAccount(bams,n,h,m) ==
      bams munion .........................
pre not n in set dom bams;

AddAccHolders: BAMS * AccId * set of AccHolder -> BAMS
AddAccHolders(bams,n,h) ==
      bams ++ { n |-> let a = bams(n)
                      in mk_Account(a.H union h,a.B) }
pre ...................;

Credit: BAMS * AccId * Amount -> BAMS
Credit(bams,n,m) == ..............................
pre n in set dom bams;

Withdraw: BAMS * AccId * Amount -> BAMS
Withdraw(bams,n,m) ==
      bams ++ { n |-> let a = bams(n)
                              in mk_Account(a.H, a.B - m) }
pre ....................................;

GoodCostumers: BAMS * Amount -> set of AccId
GoodCostumers(bams,m) ==
      --- should select the accounts whose balance is greater than m
```

2. Attach an invariant to BAMS ensuring:

   1. that every account has at least one account holder;
   2. that the balance of every account is positive.

3.1. Pretty-print

types

35.0 $BAMS = AccId \xrightarrow{m} Account;$

36.0 $Account :: H : AccHolder$-set

.1 $B : Amount$

.2 inv $a \triangleq (a.H \neq \{\}) \wedge (a.B > 0);$

37.0 $AccId =$ char*;

38.0 $AccHolder =$ char*;

39.0 $Amount = \mathbb{Z}$

functions

40.0 $Init : () \to BAMS$

.1 $Init\,() \triangleq$

.2 $\{\mapsto\};$

41.0 $OpenAccount : BAMS \times AccId \times AccHolder$-set $\times\ Amount \to BAMS$

.1 $OpenAccount\,(bams, n, h, m) \triangleq$

.2 $bams \overset{m}{\cup} \{n \mapsto$ mk-$Account\,(h, m)\}$

.3 pre $\neg\, n \in$ dom $bams$;

42.0 $AddAccHolders : BAMS \times AccId \times AccHolder$-set $\to BAMS$

.1 $AddAccHolders\,(bams, n, h) \triangleq$

.2 $bams \dagger \{n \mapsto$ let $a = bams\,(n)$ in

.3 mk-$Account\,(a.H \cup h, a.B)\}$

.4 pre $n \in$ dom $bams$;

43.0 $Credit : BAMS \times AccId \times Amount \to BAMS$

.1 $Credit\,(bams, n, m) \triangleq$

.2 $bams \dagger \{n \mapsto$ let $a = bams\,(n)$ in

.3 mk-$Account\,(a.H, a.B + m)\}$

.4 pre $n \in$ dom $bams$;

44.0 $Withdraw : BAMS \times AccId \times Amount \to BAMS$

.1 $Withdraw\,(bams, n, m) \triangleq$

.2 $\quad bams \dagger \{n \mapsto \mathsf{let}\ a = bams\,(n)\ \mathsf{in}$

.3 $\quad\quad \mathsf{mk}\text{-}Account\,(a.H, a.B - m)\}$

.4 $\mathsf{pre}\ \ n \in \mathsf{dom}\ bams \wedge bams\,(n).B \geq m$;

45.0 $GoodCostumers : BAMS \times Amount \to AccId\text{-}\mathsf{set}$

.1 $GoodCostumers\,(bams, m) \triangleq$

.2 $\quad \{x \mid x \in \mathsf{dom}\ bams \cdot bams\,(x).B > m\}$

3.2. Source code

```
------------------------------------------------------------------------------
-- Rui Carlos A. Goncalves
-- LMCC - MFP I
-- 2006/07
-- Problema 6
------------------------------------------------------------------------------

types
BAMS = map AccId to Account;
Account :: H: set of AccHolder
           B: Amount
inv a==(a.H<>{}) and (a.B>0);

AccId     = seq of char;
AccHolder = seq of char;
Amount = int;


functions
Init : () -> BAMS
Init () == {|->};

OpenAccount : BAMS * AccId * set of AccHolder * Amount -> BAMS
OpenAccount(bams,n,h,m) ==
      bams munion {n|->mk_Account(h,m)}
pre not n in set dom bams;

AddAccHolders: BAMS * AccId * set of AccHolder -> BAMS
AddAccHolders(bams,n,h) ==
      bams ++ { n |-> let a = bams(n)
                      in mk_Account(a.H union h,a.B) }
pre n in set dom bams;

Credit: BAMS * AccId * Amount -> BAMS
Credit(bams,n,m)==bams ++ {n |-> let a=bams(n)
                                 in mk_Account(a.H,a.B+m)}
pre n in set dom bams;
```

```
Withdraw: BAMS * AccId * Amount -> BAMS
Withdraw(bams,n,m) ==
      bams ++ { n |-> let a = bams(n)
                          in mk_Account(a.H, a.B - m) }
pre n in set dom bams and bams(n).B>=m;

GoodCostumers: BAMS * Amount -> set of AccId
GoodCostumers(bams,m)=={x|x in set dom bams & bams(x).B>m};
```

4. Problem 7 - Multisets "are" Mappings

A multiset (vulg. "bag") is an "intermediate" notion between a set (`set of A`) and a sequence (`seq of A`). In a multiset, a given element a of A may occur more than once, its number of occurrences being called its *multiplicity*:

```
map @A to nat
```

Multisets are very common in formal modelling. For instance, an inventory

What	Quantity
chair	12
table	3
cupboard	1

is modelled by a multiset:

```
m = {"chair"|->12,"table"|->3,"cupboard"|->1}
```

Invoices are multisets, bank statements are multisets, etc.

The empty mapping {`|->`} models the empty multiset and set-union generalizes to multiplicity addition, that is, given two multisets `n` and `m`, their union is defined as follows:

```
mseCup[@A] : map @A to nat * map @A to nat -> map @A to nat
mseCup(m,n) == m ++ n ++
               { a |-> m(a) + n(a) | a in set dom n inter dom m };
```

1. What is the outcome of `mseCup(m,m)`? Check it in the toolbox for m the multiset above. Check that `mseCup(m,m)` has the same elements of `m` but that their multiplicity is two times larger.

   Generalize this by defining

   ```
   mseExMul[@A] : map @A to nat * nat -> map @A to nat
   ```

   the multiplication of a multiset by a numeric factor.

2. Define a function

   ```
   mseDiff[@A] : map @A to nat * map @A to nat -> map @A to nat
   ```

   which computes multiset *difference*.

3. Generalize multiset union to a finite set of multisets:

   ```
   mseCUP[@A] : seq of map @A to nat -> map @A to nat
   ```

4.1. Pretty-print

functions

46.0 $mseCup[@A] : @A \xrightarrow{m} \mathbb{N} \times @A \xrightarrow{m} \mathbb{N} \to @A \xrightarrow{m} \mathbb{N}$

.1 $mseCup\,(m, n) \triangleq$

.2 $\quad m \dagger n \dagger$

.3 $\quad \{a \mapsto m\,(a) + n\,(a) \mid a \in \mathsf{dom}\ n \cap \mathsf{dom}\ m\};$

47.0 $mseExMul[@a] : @a \xrightarrow{m} \mathbb{N} \times \mathbb{N} \to @a \xrightarrow{m} \mathbb{N}$

.1 $mseExMul\,(m, n) \triangleq$

.2 $\quad \mathsf{if}\ n = 1$

.3 $\quad \mathsf{then}\ m$

.4 $\quad \mathsf{else}\ mseCup[@a]\,(m, mseExMul[@a]\,(m, n-1));$

48.0 $mseExMul2[@a] : @a \xrightarrow{m} \mathbb{N} \times \mathbb{N} \to @a \xrightarrow{m} \mathbb{N}$

.1 $mseExMul2\,(m, n) \triangleq$

.2 $\quad \{a \mapsto m\,(a) \times n \mid a \in \mathsf{dom}\ m\};$

49.0 $mseDiff\,[@a] : @a \xrightarrow{m} \mathbb{N} \times @a \xrightarrow{m} \mathbb{N} \to @a \xrightarrow{m} \mathbb{N}$

.1 $mseDiff\ (m1, m2) \triangleq$

.2 $\quad \{a \mapsto (m1\,(a) - m2\,(a)) \mid a \in \mathsf{dom}\ m1,$

.3 $\qquad a \in \mathsf{dom}\ m2 \cdot$

.4 $\qquad\quad m1\,(a) - m2\,(a) > 0\}\ \biguplus_{\mathrm{m}}$

.5 $\quad \{a \mapsto m1\,(a) \mid a \in \mathsf{dom}\ m1 \cdot$

.6 $\qquad\quad \neg\,(a \in \mathsf{dom}\ m2)\};$

50.0 $mseCUP[@a] : @a \xrightarrow{m} \mathbb{N}^* \to @a \xrightarrow{m} \mathbb{N}$

.1 $mseCUP\,(s) \triangleq$

.2 $\quad \mathsf{if}\ \mathsf{len}\ s = 1$

.3 $\quad \mathsf{then}\ \mathsf{hd}\ s$

.4 $\quad \mathsf{else}\ mseCup[@a]\,(\mathsf{hd}\ s, mseCUP[@a]\,(\mathsf{tl}\ s))$

.5 $\mathsf{pre}\ \ \mathsf{len}\ s \geq 1$

values

51.0 $m1 = \{\texttt{"}a\texttt{"} \mapsto 3, \texttt{"}b\texttt{"} \mapsto 3, \texttt{"}c\texttt{"} \mapsto 1, \texttt{"}d\texttt{"} \mapsto 1\};$

52.0 $m2 = \{\texttt{"}a\texttt{"} \mapsto 3, \texttt{"}c\texttt{"} \mapsto 4, \texttt{"}e\texttt{"} \mapsto 1, \texttt{"}g\texttt{"} \mapsto 5\}$

4.2. Source code

```
------------------------------------------------------------------------------
-- Rui Carlos A. Goncalves
-- LMCC - MFP I
-- 2006/07
-- Problema 7
------------------------------------------------------------------------------

functions
mseCup[@A] : map @A to nat * map @A to nat -> map @A to nat
mseCup(m,n) == m ++ n ++
               { a |-> m(a) + n(a) | a in set dom n inter dom m };

-- 1
mseExMul[@a]:map @a to nat*nat->map @a to nat
mseExMul(m,n)==if n=1 then m
                      else mseCup[@a](m,mseExMul[@a](m,n-1));

-- versao alternativa
mseExMul2[@a]:map @a to nat*nat->map @a to nat
mseExMul2(m,n)=={a|->m(a)*n|a in set dom m};


-- 2
mseDiff[@a]:map @a to nat*map @a to nat->map @a to nat
mseDiff(m1,m2)=={a|->(m1(a)-m2(a))|a in set dom m1
                                  ,a in set dom m2
                                  &m1(a)-m2(a)>0}
         munion {a|->m1(a)|a in set dom m1
                           &not(a in set dom m2)};

-- 3
mseCUP[@a]:seq of map @a to nat->map @a to nat
mseCUP(s)==if len s=1 then hd s
                      else mseCup[@a](hd s,mseCUP[@a](tl s))
pre len s>=1;


values
m1={"a"|->3,"b"|->3,"c"|->1,"d"|->1};
m2={"a"|->3,"c"|->4,"e"|->1,"g"|->5};
```

5. Problem 8 - Hash tables "are" Mappings

Hash tables are well known data structures whose purpose is to efficiently combine the advantages of both static and dynamic storage of data. Static structures such as *arrays* provide random access to data but have the disadvantage of filling too much primary storage. Dynamic, pointer-based structures (*e.g.* search lists, search trees, *etc.*) are more versatile with respect to storage requirements but access to data is not as immediate.

The idea of *hashing* is suggested by the informal meaning of the term itself: a large database file is "hashed" into as many "pieces" as possible, each of which is randomly accessed. Since each sub-database is smaller than the original, the time spent on accessing data is shortened by some order of magnitude. Random access is normally achieved by a so-called *hash function*,

```
hash : Data -> Location
```

which computes, for each data item, its location in the *hash table*. Standard terminology regards as *synonyms* all data competing for the same location. A set of synonyms is called a *bucket*. There are several ways in which data collision is handled, *e.g. linear probing* or *overflow handling*. Below we consider the latter.

1. Write and validate a VDM-SL model *HTable* for hash-tables (with overflow handling) in which you should model (a) locations as integers; (b) collision buckets as finite sets of (data) synonyms; (c) hash tables as mappings from locations to collision buckets.
2. Assuming some predefined hash function *hash*, add to *HTable* an invariant stating that all data in the same collision bucket share the same hash-index. Can you infer from this invariant that buckets are mutually disjoint?
3. Specify the following functionality on top of *HTable* so that your invariant (above) is preserved:

   ```
   Init : () -> HTable

   Insert : Data * HTable -> HTable

   Find   : Data * HTable -> bool

   Remove : Data * HTable -> HTable
   ```

4. Specify a function `rep` able to represent any `set of Data` as a `HTable`.
5. Further specify a left inverse `abs` of `rep`, *i.e.* a function able to retrieve the collection of data which is stored into a hash-table.

5.1. Pretty-print

types

53.0 $Data = \mathsf{token}$;

54.0 $Locations = \mathbb{Z}$;

55.0 $Buckets = Data\text{-}\mathsf{set}$;

56.0 $HTable = Locations \xrightarrow{m} Buckets$

.1 inv $h \triangleq \forall\, n \in \mathsf{dom}\ h \cdot \forall\, a \in h\,(n) \cdot hash\,(a) = n$

functions

57.0 $hash : Data \to \mathbb{Z}$

.1 $hash\,(x) \triangleq$

.2 is not yet specified;

58.0 $Init : () \to HTable$

.1 $Init\,() \triangleq$

.2 $\{\mapsto\}$;

59.0 $Insert : Data \times HTable \to HTable$

.1 $Insert\,(x, h) \triangleq$

.2 let $aux =$ if $hash\,(x) \in \mathsf{dom}\ h$

.3 then $h\,(hash\,(x))$

.4 else $\{\}$ in

.5 $h \dagger \{hash\,(x) \mapsto \{x\} \cup aux\}$;

60.0 $Find : Data \times HTable \to \mathbb{B}$

.1 $Find\,(x, h) \triangleq$

.2 let $aux = h\,(hash\,(x))$ in

.3 $x \in aux$;

61.0 $Remove : Data \times HTable \to HTable$

.1 $Remove\,(x, h) \triangleq$

.2 let $aux = h\,(hash\,(x))$ in

.3 $h \dagger \{hash\,(x) \mapsto aux \setminus \{x\}\}$

.4 pre $Find\,(x, h)$;

62.0 $rep : Data\text{-set} \to HTable$

.1 $rep\,(s) \triangleq$

.2 $\{hash\,(x) \mapsto \{y \mid y \in s \cdot hash\,(y) = hash\,(x)\} \mid x \in s\}$;

63.0 $abs1 : HTable \to Data\text{-set}$

.1 $abs1\,(h) \triangleq$

.2 $\bigcup \{h\,(a) \mid a \in \mathsf{dom}\ h\}$

5.2. Source code

```
------------------------------------------------------------------------------
-- Rui Carlos A. Goncalves
-- LMCC - MFP I
-- 2006/07
-- Problema 8
------------------------------------------------------------------------------

types
Data=token;

--1,2
Locations=int;
Buckets=set of Data;

HTable=map Locations to Buckets
inv h==forall n in set dom h & forall a in set h(n) & hash(a)=n;

functions
hash:Data->int
hash(x)==is not yet specified;

--3
Init:()->HTable
Init()=={|->};

Insert:Data*HTable->HTable
Insert(x,h)==let aux=if hash(x) in set dom h
                     then h(hash(x))
                     else {}
             in h++{hash(x)|->{x} union aux};

Find:Data*HTable->bool
Find(x,h)==let aux=h(hash(x))
           in x in set aux;

Remove:Data*HTable->HTable
Remove(x,h)==let aux=h(hash(x))
             in h++{hash(x)|->aux\{x}}
pre Find(x,h);

--4
rep:set of Data->HTable
rep(s)=={hash(x)|->{y|y in set s & hash(y)=hash(x)}|x in set s};

--5
abs1:HTable->set of Data
abs1(h)==dunion {h(a)|a in set dom h};
```

6. Problem 10 - A Naive Model of the WWW

Consider the following VDM-SL model specifying, at abstract level, the structure of an information system based on the 'World Wide Web' over the INTERNET, where `Ref` (page address) is a datatype of which no further details are required:

```
WWW  = map Ref to URL;           -- (URL=Universal Resource Location)
URL  = seq of Unit;
Unit = PlainText | HyperLink;
PlainText = seq of Word;
Word = seq of char;
HyperLink :: link: Ref
             txt: PlainText;     -- "underlined text"
```

1. Add an invariant to `WWW` ensuring that no URL mentions a non-existing URL (NB: although this cannot be ensured for the WWW as a whole, it makes sense for the particular fragment which embodies our information system).

2. Assuming

   ```
   wc : seq of char -> nat
   -- counts the number of words in a string
   ```

   specify

   ```
   nrofwd : URL -> nat
   -- counts the number of words in a URL
   ```

3. Specify

   ```
   refsTo : Ref * WWW -> set of Ref
   -- retrieves the addresses of all URLs which point at Ref
   ```

4. Search engines on the WWW are based on text inversion, that is, on a function

   ```
   invert : WWW -> map Word to map Ref to nat1
   ```

   which computes, for every word, the URLs which mention it, weighted by the number of occurrences, ie. a multiset. Specify `invert`. (NB: this inversion operation is far more elaborate in practice!)

6.1. Pretty-print

types

64.0 $Ref = \mathbb{Z}$;

65.0 $HyperLink :: link : Ref$
.1 $txt : PlainText$;

66.0 $Word = \mathsf{char}^*$;

67.0 $PlainText = Word^*$;

68.0 $Unit = PlainText \mid HyperLink$;

69.0 $URL = Unit^*$;

70.0 $WWW = Ref \xrightarrow{m} URL$

.1 inv $www \triangleq \forall\, x \in refs\,(\mathsf{rng}\ www) \cdot x \in \mathsf{dom}\ www$

functions

71.0 $refsaux : Unit \to Ref\text{-}\mathsf{set}$

.1 $refsaux\,(u) \triangleq$

.2 if is-$HyperLink\,(u)$

.3 then $\{u.link\}$

.4 else $\{\}$;

72.0 $refs : URL\text{-}\mathsf{set} \to Ref\text{-}\mathsf{set}$

.1 $refs\,(urls) \triangleq$

.2 let $x1 = \{\mathsf{elems}\ (x) \mid x \in urls\}$,

.3 $x2 = \bigcup x1$,

.4 $x3 = \{refsaux\,(x) \mid x \in x2\}$ in

.5 $\bigcup x3$;

73.0 $wc : \mathsf{char}^* \to \mathbb{N}$

.1 $wc\,(x) \triangleq$

.2 is not yet specified;

74.0 $wcunit : Unit \to \mathbb{N}$

.1 $wcunit\,(unit) \triangleq$

.2 if is-$HyperLink\,(unit)$

.3 then $sum\,([wc\,(unit.txt\,(i)) \mid i \in \mathsf{inds}\ unit.txt])$

.4 else $sum\,([wc\,(unit\,(i)) \mid i \in \mathsf{inds}\ unit])$;

75.0 $sum : \mathbb{N}^* \to \mathbb{N}$

.1 $sum\,(s) \triangleq$

.2 if len $s = 0$

.3 then 0

.4 else hd $(s) + sum\,(\mathsf{tl}\ (s))$;

76.0 $nrofwd : URL \to \mathbb{N}$

.1 $nrofwd\,(url) \triangleq$

.2 $\quad sum\,([wcunit\,(url\,(i)) \mid i \in \mathsf{inds}\ url])$;

77.0 $refsTo : Ref \times WWW \to Ref\text{-}\mathsf{set}$

.1 $refsTo\,(ref, www) \triangleq$

.2 $\quad \{r \mid r \in \mathsf{dom}\ www \cdot ref \in refs\,(\{www\,(r)\})\}$

.3 pre $ref \in \mathsf{dom}\ www$;

78.0 $extract : Unit \to Word^*$

.1 $extract\,(x) \triangleq$

.2 $\quad \mathsf{if}$ is-$HyperLink\,(x)

.3 $\quad \mathsf{then}\ x.txt$

.4 $\quad \mathsf{else}\ x$;

79.0 $join : URL \to Word^*$

.1 $join\,(url) \triangleq$

.2 $\quad \mathsf{conc}\,[extract\,(url\,(i)) \mid i \in \mathsf{inds}\ url]$;

80.0 $words : WWW \to Ref \xrightarrow{m} Word^*$

.1 $words\,(x) \triangleq$

.2 $\quad \{a \mapsto join\,(x\,(a)) \mid a \in \mathsf{dom}\ x\}$;

81.0 $count : Word \times Ref \xrightarrow{m} Word^* \to Ref \xrightarrow{m} \mathbb{N}_1$

.1 $count\,(w, m) \triangleq$

.2 $\quad \mathsf{let}\ r = \{x \mapsto \mathsf{len}\,[(m\,(x))\,(i) \mid i \in \mathsf{inds}\ m\,(x) \cdot (m\,(x))\,(i) = w] \mid$

.3 $\qquad\qquad x \in \mathsf{dom}\ m\}\ \mathsf{in}$

.4 $\quad r \rhd \{x \mid x \in \mathsf{rng}\ r \cdot x > 0\}$;

82.0 $invert : WWW \to Word \xrightarrow{m} Ref \xrightarrow{m} \mathbb{N}_1$

.1 $invert\,(www) \triangleq$

.2 $\quad \mathsf{let}\ ws = words\,(www),$

.3 $\qquad wd = \bigcup \{\mathsf{elems}\ (a) \mid a \in \mathsf{rng}\ ws\}\ \mathsf{in}$

.4 $\quad \{x \mapsto count\,(x, ws) \mid x \in wd\}$

values

83.0 $www : WWW = \{1 \mapsto [["abc", "cde"], ["cde", "efg"]],$
.1 $2 \mapsto [["hij", "abc", "xyz"], ["xpto"], ["abc", "cde"]],$
.2 $3 \mapsto [["hij", "xpto", "www"], ["www", "xpto", "net"]],$
.3 $4 \mapsto [["a", "b"]],$
.4 $5 \mapsto [["a", "b"]],$
.5 $6 \mapsto [\mathsf{mk\text{-}}HyperLink\,(2, ["aaaa", "bbbb"]), \mathsf{mk\text{-}}HyperLink\,(3, ["cccc"])],$
.6 $7 \mapsto [\mathsf{mk\text{-}}HyperLink\,(3, ["aaaa"])]\}$

6.2. Source code

```
--------------------------------------------------------------------------------
-- Rui Carlos A. Goncalves
-- LMCC - MFP I
-- 2006/07
-- Problema 10
--------------------------------------------------------------------------------

types
Ref=int;

HyperLink::link:Ref
           txt :PlainText;

Word=seq of char;

PlainText=seq of Word;

Unit=PlainText|HyperLink;

URL=seq of Unit;

WWW=map Ref to URL
inv www==forall x in set refs(rng www) & x in set dom www;


functions
--1
refsaux:Unit->set of Ref
refsaux(u)==if is_HyperLink(u)
               then {u.link}
               else {};

refs:set of URL->set of Ref
refs(urls)==let x1={elems(x)|x in set urls},
                x2=dunion x1,
                x3={refsaux(x)|x in set x2}
            in dunion x3;

--2
wc:seq of char->nat
wc(x)==is not yet specified;

wcunit:Unit->nat
```

```
wcunit(unit)==if is_HyperLink(unit)
                then sum([wc(unit.txt(i))|i in set inds unit.txt])
                else sum([wc(unit(i))|i in set inds unit]);

sum:seq of nat->nat
sum(s)==if len s=0
          then 0
          else hd(s)+sum(tl(s));

nrofwd:URL->nat
nrofwd(url)==sum([wcunit(url(i))|i in set inds url]);

--3
refsTo:Ref*WWW->set of Ref
refsTo(ref,www)=={r|r in set dom www & ref in set refs({www(r)})}
pre ref in set dom www;

--4
extract:Unit->seq of Word
extract(x)==if is_HyperLink(x)
              then x.txt
              else x;

join:URL->seq of Word
join(url)==conc [extract(url(i))|i in set inds url];

words:WWW->map Ref to seq of Word
words(x)=={a|->join(x(a))|a in set dom x};


count:Word*map Ref to seq of Word->map Ref to nat1
count(w,m)==let r={x|->len [(m(x))(i)|i in set inds m(x) & (m(x))(i)=w]
                  |x in set dom m}
            in r:>{x|x in set rng r & x>0};


invert:WWW->map Word to map Ref to nat1
invert(www)==let ws=words(www),
                 wd=dunion {elems(a)|a in set rng ws}
             in {x|->count(x,ws)|x in set wd};


values
www:WWW={1|->[["abc","cde"],["cde","efg"]],
         2|->[["hij","abc","xyz"],["xpto"],["abc","cde"]],
         3|->[["hij","xpto","www"],["www","xpto","net"]],
         4|->[["a","b"]],
         5|->[["a","b"]],
         6|->[mk_HyperLink(2,["aaaa","bbbb"]),mk_HyperLink(3,["cccc"])],
         7|->[mk_HyperLink(3,["aaaa"])]
        };
```

7. Problem 11 - Modelling a Memory Cache

Analyse the formal VDM-SL model below of a memory *cache*. *Try to improve it* and design a test suite in the toolbox to check its behaviour.

```
types
        Addr = token;
        Data = token;

        Memory :: addrs: set of Addr
                  mapx: map Addr to Data;

        MainMemory = Memory;

        Cache :: addrs: set of Addr
                 mapx: map Addr to Data
                 dirty: set of Addr;

        System :: cache: Cache
                  main: MainMemory
                  inv system == system.cache.addrs subset system.main.addrs;

functions

        badAddrs: System -> set of Addr
        badAddrs (s) == { x | x in set s.cache.addrs &
                              s.cache.mapx(x) <> s.main.mapx(x) };

        load: System * Addr -> System
        load (s, a) == mk_System(
                           mk_Cache(s.cache.addrs union {a},
                                    s.cache.mapx munion {a|->s.main.mapx(a)},
                                    s.cache.dirty),
                           s.main)
        pre a not in set s.cache.addrs;

        systemWrite: System * Data * Addr -> System
        systemWrite (s, d, a) ==
                           mk_System(
                              mk_Cache(s.cache.addrs,
                                       s.cache.mapx ++ {a|->d},
                                       s.cache.dirty union {a}),
                           s.main);

        flush: System -> System
        flush (s) == let x in set s.cache.addrs
                     in mk_System(
                              mk_Cache(s.cache.addrs \ {x},
                                       {x} <-: s.cache.mapx,
                                       s.cache.dirty \ {x}),
                              mk_Memory(s.main.addrs,
                                        s.main.mapx ++ {x|->s.cache.mapx(x)}));
```

7.1. Pretty-print

types

84.0 $Addr = \mathsf{token};$

85.0 $Data = \mathsf{token};$

86.0 $Memory :: addrs : Addr\text{-}\mathsf{set}$
.1 $mapx : Addr \xrightarrow{m} Data$
.2 $\mathsf{inv}\ m \triangleq \mathsf{dom}\ m.mapx \subseteq m.addrs;$

87.0 $MainMemory = Memory;$

88.0 $Cache :: addrs : Addr\text{-}\mathsf{set}$
.1 $mapx : Addr \xrightarrow{m} Data$
.2 $dirty : Addr\text{-}\mathsf{set}$
.3 $\mathsf{inv}\ c \triangleq$
$c.dirty \subseteq c.addrs \wedge$
$\mathsf{dom}\ c.mapx \subseteq c.addrs;$

89.0 $System :: cache : Cache$
.1 $main : MainMemory$
.2 $\mathsf{inv}\ system \triangleq system.cache.addrs \subseteq system.main.addrs$

functions

90.0 $badAddrs : System \to Addr\text{-}\mathsf{set}$
.1 $badAddrs\,(s) \triangleq$
.2 $\{x \mid x \in s.cache.addrs \cdot$
.3 $s.cache.mapx\,(x) \neq s.main.mapx\,(x)\};$

91.0 $load : System \times Addr \to System$
.1 $load\,(s, a) \triangleq$
.2 $\mathsf{mk}\text{-}System$
.3 $($
.4 $\mathsf{mk}\text{-}Cache\,(s.cache.addrs \cup \{a\},$
.5 $s.cache.mapx \cup \{a \mapsto s.main.mapx\,(a)\},$
.6 $s.cache.dirty),$
.7 $s.main)$
.8 $\mathsf{pre}\ \ a \notin s.cache.addrs\ ;$

```
92.0  systemWrite : System × Data × Addr → System
 .1   systemWrite (s, d, a) △
 .2     mk-System
 .3           (
 .4            mk-Cache (s.cache.addrs,
 .5                      s.cache.mapx † {a ↦ d},
 .6                      s.cache.dirty ∪ {a}),
 .7            s.main)
 .8   pre  a ∈ s.cache.addrs ;


93.0  flush : System → System
 .1   flush (s) △
 .2     let x ∈ s.cache.addrs in
 .3     mk-System
 .4           (
 .5            mk-Cache (s.cache.addrs \ {x},
 .6                      {x} ⩤ s.cache.mapx,
 .7                      s.cache.dirty \ {x}),
 .8            mk-Memory (s.main.addrs,
 .9                       s.main.mapx † {x ↦ s.cache.mapx (x)}))
 .10  pre  s.cache.addrs ≠ {}

values

94.0  a0 : Addr = mk-token (0);

95.0  a1 : Addr = mk-token (1);

96.0  a2 : Addr = mk-token (2);

97.0  a3 : Addr = mk-token (3);

98.0  a4 : Addr = mk-token (4);

99.0  a5 : Addr = mk-token (5);

100.0 a6 : Addr = mk-token (6);
```

```
101.0   a7 : Addr = mk-token (7);

102.0   a8 : Addr = mk-token (8);

103.0   a9 : Addr = mk-token (9);

104.0   d0 : Data = mk-token ("");

105.0   d1 : Data = mk-token ("a");

106.0   d2 : Data = mk-token ("b");

107.0   d3 : Data = mk-token ("c");

108.0   d4 : Data = mk-token ("d");

109.0   d5 : Data = mk-token ("e");

110.0   d6 : Data = mk-token ("f");

111.0   d7 : Data = mk-token ("g");

112.0   d8 : Data = mk-token ("h");

113.0   d9 : Data = mk-token ("i");

114.0   sys : System = mk-System
    .1                 (
    .2                  mk-Cache ({},
    .3                            {↦},
    .4                            {}),
    .5                  mk-Memory ({a0, a1, a2, a3, a4, a5, a6, a7, a8, a9},
    .6                             {a0 ↦ d0, a1 ↦ d0, a2 ↦ d0, a3 ↦ d0, a4 ↦ d0,
    .7                              a5 ↦ d0, a6 ↦ d0, a7 ↦ d0, a8 ↦ d0, a9 ↦
d0}));
```

115.0 $sys1 = systemWrite\,(load\,(sys, a3), d1, a3);$

116.0 $sys2 = systemWrite\,(load\,(sys1, a2), d8, a2);$

117.0 $sys3 = systemWrite\,(load\,(sys2, a7), d4, a7);$

118.0 $sys4 = systemWrite\,(load\,(sys3, a4), d2, a4);$

119.0 $sys5 = flush\,(sys4);$

120.0 $sys6 = flush\,(sys5);$

121.0 $sys7 = flush\,(sys6)$

7.2. Source code

```
--------------------------------------------------------------------------------
-- Rui Carlos A. Goncalves
-- LMCC - MFP I
-- 2006/07
-- Problema 11
--------------------------------------------------------------------------------

types
Addr = token;
Data = token;

Memory :: addrs: set of Addr
          mapx: map Addr to Data
inv m == dom m.mapx subset m.addrs;

MainMemory = Memory;

Cache :: addrs: set of Addr
         mapx: map Addr to Data
        dirty: set of Addr
inv c == c.dirty subset c.addrs
     and dom c.mapx subset c.addrs;

System :: cache: Cache
          main: MainMemory
inv system == system.cache.addrs subset system.main.addrs;
```

```
functions

badAddrs: System -> set of Addr
badAddrs (s) == { x | x in set s.cache.addrs &
                      s.cache.mapx(x) <> s.main.mapx(x) };

load: System * Addr -> System
load (s, a) == mk_System(
                  mk_Cache(s.cache.addrs union {a},
                           s.cache.mapx munion {a|->s.main.mapx(a)},
                           s.cache.dirty),
                  s.main)
pre a not in set s.cache.addrs;

systemWrite: System * Data * Addr -> System
systemWrite (s, d, a) ==mk_System(
                           mk_Cache(s.cache.addrs,
                                    s.cache.mapx ++ {a|->d},
                                    s.cache.dirty union {a}),
                           s.main)
pre a in set s.cache.addrs;

flush: System -> System
flush (s) == let x in set s.cache.addrs
             in mk_System(
                   mk_Cache(s.cache.addrs \ {x},
                            {x} <-: s.cache.mapx,
                            s.cache.dirty \ {x}),
                   mk_Memory(s.main.addrs,
                             s.main.mapx ++ {x|->s.cache.mapx(x)}))
pre s.cache.addrs<>{};

values
a0:Addr=mk_token(0);
a1:Addr=mk_token(1);
a2:Addr=mk_token(2);
a3:Addr=mk_token(3);
a4:Addr=mk_token(4);
a5:Addr=mk_token(5);
a6:Addr=mk_token(6);
a7:Addr=mk_token(7);
a8:Addr=mk_token(8);
a9:Addr=mk_token(9);

d0:Data=mk_token("");
d1:Data=mk_token("a");
d2:Data=mk_token("b");
d3:Data=mk_token("c");
d4:Data=mk_token("d");
d5:Data=mk_token("e");
d6:Data=mk_token("f");
d7:Data=mk_token("g");
d8:Data=mk_token("h");
```

```
d9:Data=mk_token("i");

sys:System=mk_System(
                mk_Cache({},
                         {|->},
                         {}
                        ),
                mk_Memory({a0,a1,a2,a3,a4,a5,a6,a7,a8,a9},
                          {a0|->d0,a1|->d0,a2|->d0,a3|->d0,a4|->d0,
                           a5|->d0,a6|->d0,a7|->d0,a8|->d0,a9|->d0
                          }
                         )
                );

sys1=systemWrite(load(sys,a3),d1,a3);
sys2=systemWrite(load(sys1,a2),d8,a2);
sys3=systemWrite(load(sys2,a7),d4,a7);
sys4=systemWrite(load(sys3,a4),d2,a4);
sys5=flush(sys4);
sys6=flush(sys5);
sys7=flush(sys6);
```