

VDM case studies — Formal Methods II/2006

J.N. Oliveira *

June 12, 2006

1 Hash table representation of finite sets

1.1 Abstract data type

1.1.1 Abstract data models

$$\begin{aligned} \textit{Collection} &= \textit{Data-set}; \\ \textit{Data} &= \mathbb{Z} \end{aligned}$$

1.1.2 Abstract functionality

Insert data into collection:

$$\begin{aligned} \textit{insert} : \textit{Data} &\rightarrow \textit{Collection} \rightarrow \textit{Collection} \\ \textit{insert}(d)(S) &\triangleq \\ S \cup \{d\}; \end{aligned}$$

Find data in collection:

$$\begin{aligned} \textit{find} : \textit{Data} &\rightarrow \textit{Collection} \rightarrow \mathbb{B} \\ \textit{find}(d)(S) &\triangleq \\ d \in S \end{aligned}$$

1.2 Concrete data type

1.2.1 Concrete data models

Define

$$\begin{aligned} \textit{HTable} &= \textit{Location} \xrightarrow{m} \textit{Data-set} \\ \textit{inv HT} &\triangleq \forall l \in \text{dom HT} \cdot \\ &\quad \textit{HT}(l) \neq \{\} \wedge \\ &\quad \forall d \in \textit{HT}(l) \cdot \textit{hash}(d) = l; \\ \textit{Location} &= \mathbb{N} \end{aligned}$$

assuming some given hash function, for instance

*Dep. Informática, Universidade do Minho, Campus de Gualtar, 4700 Braga, Portugal. EMAIL: jno@di.uminho.pt.

```

hash : Data → Location
hash (d)  $\triangleq$ 
  d rem 13;

```

1.2.2 Concrete functionality

Implementing *insert*:

```

insertHT : Data → HTable → HTable
insertHT (d)(HT)  $\triangleq$ 
  let l = hash (d) in
  HT † {l ↦ (if l ∈ dom HT
    then HT (l)
    else {} ) ∪ {d}};

```

Implementing *find*:

```

findHT : Data → HTable →  $\mathbb{B}$ 
findHT (d)(HT)  $\triangleq$ 
  find (d) (HT (hash (d)));

```

1.3 Refinement step

1.3.1 Abstraction function

```

absf : HTable → Collection
absf (HT)  $\triangleq$ 
   $\bigcup$  (rng HT);

```

1.3.2 Representation function

```

repf : Collection → HTable
repf (S)  $\triangleq$ 
  {hash (x) ↦ {d | d ∈ S · hash (d) = hash (x)} | x ∈ S}

```

1.4 Test data

values

```

S : Collection = {1, 13, 2, 34, 5, 10, 26};

```

```

S1 : Collection = insert (45) (S);

```

$HT : HTable = \text{repf}(S);$

$HT_{\text{illegal}} : HTable = \text{repf}(S) \uparrow \{1 \mapsto \{1, 13\}\};$

$HT1 : HTable = \text{insert}_{HT}(45)(HT);$

$S1' : \text{Collection} = \text{absf}(HT1)$

2 ECTS problem

2.1 Abstract data type

2.1.1 Abstract data models

Given

$\text{Course} :: \text{name} : \text{char}^*$
 $\text{credits} : \text{ECTSs};$
 $\text{ECTSs} = \text{ACMtopic} \xrightarrow{m} \mathbb{N}_1;$
 $\text{CourseId} = \text{token};$
 $\text{IdDeg} = \text{token};$
 $\text{StudentId} = \text{token};$
 $\text{ACMtopic} = \text{token};$
 $\text{Classif} = \mathbb{R};$

define

$\text{Univ} :: \text{courses} : \text{CourseId} \xrightarrow{m} \text{Course}$
 $\text{degrees} : \text{IdDeg} \xrightarrow{m} \text{ECTSs}$
 $\text{students} : \text{StudentId} \xrightarrow{m} (\text{IdDeg} \times \text{CourseId} \xrightarrow{m} \text{Classif})$
 $\text{topics} : \text{ACMtopic} \xrightarrow{m} \text{ACMtopic}$
 $\text{inv } u \triangleq \text{degreesOk}(u) \wedge \text{coursesOk}(u) \wedge \text{topicsOk}(u)$

where

```

degreesOk : Univ → ℬ
degreesOk (u) ≜
  {a.#1 | a ∈ rng u.students} ⊆ dom u.degrees;

coursesOk : Univ → ℬ
coursesOk (u) ≜
  ⋃ {dom a.#2 | a ∈ rng u.students} ⊆ dom u.courses;

topicsOk : Univ → ℬ
topicsOk (u) ≜
  ⋃ {dom x | x ∈ rng u.degrees} ∪
  ⋃ {dom x.credits | x ∈ rng u.courses} ⊆
  allTopics (u);

allTopics : Univ → ACMtopic-set
allTopics (u) ≜
  dom u.topics ∪ rng u.topics;

```

2.1.2 Abstract functionality

```

allCred : StudentId × Univ → ECTSs
allCred (a, u) ≜
  let stCs = dom (u.students (a).#2) <Δ u.courses,
    allEcts = {c ↦ stCs (c).credits | c ∈ dom stCs} in
    cupBagRngFold[CourseId, ACMtopic] (allEcts)
pre a ∈ dom u.students
;

```

```

calCred : StudentId × Univ → ECTSs
calCred (a, u) ≜
  is not yet specified

```

2.1.3 Abstract level test data

Courses:

values

```
d3 : Course = mk-Course ("Met.Form.Prog.I",
  {mk-token ("FormMod") ↦ 2,
   mk-token ("RelCalc") ↦ 2,
   mk-token ("VDM-SL") ↦ 1});

d2 : Course = mk-Course ("Met.Form.Prog.II",
  {mk-token ("FormMod") ↦ 1,
   mk-token ("RelCalc") ↦ 1,
   mk-token ("ProgRef") ↦ 2,
   mk-token ("VDM++") ↦ 1});

d1 : Course = mk-Course ("Met.Prog.I",
  {mk-token ("FunProg") ↦ 3,
   mk-token ("AlgProg") ↦ 2});
```

Students:

```
s = {mk-token ("Mary") ↦
12, mk-token ("MFP-I") ↦ 16}}; mk- (mk-token ("LMCC"), {mk-token ("MP-I") ↦
```

Universities:

```
u1 = mk-Univ
  (
    {
      mk-token ("MP-I") ↦ d1,
      mk-token ("MFP-II") ↦ d2,
      mk-token ("MFP-I") ↦ d3,
    }
    {
      mk-token ("LMCC") ↦
      {mk-token ("FormMod") ↦ 5,
       mk-token ("RelCalc") ↦ 3,
       mk-token ("ProgRef") ↦ 4,
       mk-token ("VDM++") ↦ 1},
      mk-token ("LESI") ↦
      {mk-token ("FormMod") ↦ 5,
       mk-token ("RelCalc") ↦ 3,
       mk-token ("ProgRef") ↦ 4,
       mk-token ("VDM++") ↦ 1}},
    s,
    {↦})
```

Question: $u1$ below doesn't typcheck. Why?

2.2 Refinement step

2.2.1 Data level calculation

To be completed

2.2.2 Abstraction function

To be completed

2.2.3 Representation function

To be completed

2.2.4 Migration of abstract test data

To be completed

3 FS (file system) model

3.1 Abstract data type

3.1.1 Abstract data models

File system:

```
FS :: contents : Id  $\xrightarrow{m}$  Node;  
Node = File | FS;  
Id = char*;  
File :: contents : token*;
```

Tar (tape archive) file:

```
Tar = Path  $\xrightarrow{m}$  token*;  
Path = Id+
```

3.1.2 Abstract functionality

Get all identifiers:

```
ids : FS → Id-set  
ids (fs)  $\triangleq$   
   $\bigcup (\{\text{let } x = \text{fs.contents}(k) \text{ in}$   
    if is-File (x)  
    then {k}  
    else {k}  $\cup$  ids (x) |  
    k ∈ dom fs.contents});
```

Find:

```
find-name : Id → FS → Path-set  
find-name (i)(fs)  $\triangleq$   
  {p | p ∈ allPaths (fs) · i ∈ elems p};
```

where *allPaths* is function

```
allPaths : FS → Path-set  
allPaths (fs)  $\triangleq$   
  dom tar (fs);
```

and tar is function

```

tar : FS → Tar
tar (fs)  $\triangleq$ 
  merge ( {let x = fs.contents (k) in
    if is-File (x)
    then {[k]  $\mapsto$  x.contents}
    else prefix (k, tar (x)) |
    k  $\in$  dom fs.contents } );

```

where

```

prefix : Id  $\times$  Tar → Tar
prefix (k, M)  $\triangleq$ 
  {[k]  $\curvearrowright$  p  $\mapsto$  M (p) | p  $\in$  dom M};

```

cf.

$$\begin{array}{ccc}
 FS & \xrightarrow{out} & Id \rightarrow (File + FS) \\
 \downarrow tar & & \downarrow id \rightarrow (id + tar) \\
 Tar & \xleftarrow{s} & Id \rightarrow (File + Tar)
 \end{array}$$

$tarx$ is the converse of tar :

```

tarx : Tar → FS
tarx (tf)  $\triangleq$ 
  let tf1 = {p  $\mapsto$  tf (p) | p  $\in$  dom tf  $\cdot$  len p = 1},
      tf2 = nest (dom tf1  $\triangleleft$  tf) in
  mk-FS
    (
      {hd p  $\mapsto$  mk-File (tf1 (p)) | p  $\in$  dom tf1}  $\sqcup$ 
      {k  $\mapsto$  tarx (tf2 (k)) | k  $\in$  dom tf2});

```

where

```

nest : Tar → Id  $\xrightarrow{m}$  Tar
nest (tf)  $\triangleq$ 
  pcurry[Path, token*, Id] ({mk- (hd p, tl p)  $\mapsto$  tf (p) | p  $\in$  dom tf})

```

3.1.3 Abstract level test data

values

```

fs : FS = mk-FS
  (
    {"f1"  $\mapsto$  mk-File ([]),
     "d1"  $\mapsto$  mk-FS ({ "f1"  $\mapsto$  mk-File ([]), "f2"  $\mapsto$  mk-File ([])}),
     "d2"  $\mapsto$  mk-FS ({  $\mapsto$  })})

```

Exercise: eval $tarx(tar(fs)) = fs$ and draw conclusions.

4 Pedigree problem

4.1 Abstract data type

4.1.1 Abstract data model

```
GenDia :: indiv : token  
         mother : [GenDia]  
         father : [GenDia];
```

4.1.2 Abstract functionality

4.1.3 Abstract level test data

4.2 First Refinement step

4.2.1 Concrete model

```
GenDia' =  $\mathbb{N} \xrightarrow{m} (\text{token} \times [\mathbb{N}] \times [\mathbb{N}])$ 
```

4.2.2 Abstraction relation

Is simple:

```
F-GenDia : GenDia'  $\times$   $\mathbb{N} \rightarrow$  GenDia  
F-GenDia (M, k)  $\triangleq$   
  let mk- (i, k1, k2) = M (k) in  
  mk-GenDia (i,  
    if k1 = nil  
    then nil  
    else F-GenDia (M, k1),  
    if k2 = nil  
    then nil  
    else F-GenDia (M, k2))
```

4.2.3 Representation relation

To be completed

4.2.4 Test data at this level

values

```
Db : GenDia' = {
    77777777 ↦ mk- (mk-token ( " Me " ), 99999999, 88888888),
    99999999 ↦ mk- (mk-token ( " Mother " ), 11111111, nil ),
    88888888 ↦ mk- (mk-token ( " Father " ), nil , nil ),
    11111111 ↦ mk- (mk-token ( " GrandMother " ), nil , nil );

Db1 : GenDia' = {
    77777777 ↦ mk- (mk-token ( " Me " ), 99999999, 88888888),
    99999999 ↦ mk- (mk-token ( " Mother " ), 11111111, nil ),
    88888888 ↦ mk- (mk-token ( " Father " ), nil , nil ),
    11111111 ↦ mk- (mk-token ( " GrandMother " ), nil , nil );

Db2 : GenDia' = {
    77777777 ↦ mk- (mk-token ( " Me " ), 99999999, 88888888),
    99999999 ↦ mk- (mk-token ( " Mother " ), 11111111, nil ),
    88888888 ↦ mk- (mk-token ( " Father " ), nil , nil ),
    11111111 ↦ mk- (mk-token ( " GrandMother " ), nil , 77777777)}
```

4.3 Second Refinement step

4.3.1 Concrete model

$$GenDia'' = (\mathbb{N} \xrightarrow{m} \text{token}) \times (\mathbb{N} \times \mathbb{B}) \xrightarrow{m} \mathbb{N};$$

4.3.2 Abstraction relation

To be completed

4.3.3 Representation relation

To be completed

A Data refinement calculus

VDM-SL supports polymorphic, higher-order functions but not user defined (polymorphic) parametric types. So we define datatype “variables”

```
A = token;
B = token;
C = token;
D = token;
```

and datatypes

```
MaybeA = [JustA];
JustA :: val : A;
JustB :: val : B;
JustC :: val : C;
BorC = JustB | JustC
```

wherever @A, @B etc are not acceptable.

A.1 Pointer representation

Law: $A \leq 1 + A$

Representation is the *mk* of *JustA*

Abstraction: it is the converse of the representation:

```
Unjust : MaybeA → A
Unjust (x) ≜
  x.val
pre x ≠ nil
;
```

A.2 Sets refined by sequences

Law: $2^A \leq A^*$

The abstraction is VDM-SL standard function *elems*. The maximal representation is

```
Listify (S : token-set) L : token*
post
S = elems L;
```

of which

```
Listify'[@A] : @A-set → @A*
Listify' (S) ≜
  if S = {}
  then []
  else let a ∈ S in
    [a] ∘ Listify'[@A] (S \ {a});
```

is an implementation.

A.3 Mapping refinement calculus

Law: $A \rightarrow (B + C) \leq (A \rightarrow B) \times (A \rightarrow C)$.

The representation is

$$\begin{aligned} \text{uncojoin} &: A \xrightarrow{m} \text{Bor} C \rightarrow (A \xrightarrow{m} B) \times (A \xrightarrow{m} C) \\ \text{uncojoin}(f) &\triangleq \\ &\text{mk-}(\{a \mapsto f(a).\text{val} \mid \\ &\quad a \in \text{dom } f \cdot \text{is-Just} B(f(a))\}, \\ &\quad \{a \mapsto f(a).\text{val} \mid \\ &\quad a \in \text{dom } f \cdot \text{is-Just} C(f(a))\}); \end{aligned}$$

Law: $A \rightarrow (D \times (B \rightarrow C)) \leq (A \rightarrow D) \times ((A \times B) \rightarrow C)$:

The representation is

$$\begin{aligned} \text{unnjoin} &: A \xrightarrow{m} (D \times (B \xrightarrow{m} C)) \rightarrow (A \xrightarrow{m} D) \times (A \times B \xrightarrow{m} C) \\ \text{unnjoin}(M) &\triangleq \\ &\text{mk-} \\ &(\{a \mapsto M(a).\#1 \mid a \in \text{dom } M\}, \\ &\quad \text{merge } \{\{\text{mk-}(a, b) \mapsto M(a).\#2(b) \mid b \in \text{dom } M(a).\#2\} \mid a \in \text{dom } M\}); \end{aligned}$$

The abstraction is

$$\begin{aligned} \text{njoin} &: (A \xrightarrow{m} D) \times (A \times B \xrightarrow{m} C) \rightarrow A \xrightarrow{m} (D \times (B \xrightarrow{m} C)) \\ \text{njoin}(M, N) &\triangleq \\ &\{a \mapsto \text{mk-}(M(a), \{b \mapsto N(\text{mk-}(a, b)) \mid \text{mk-}(a, b) \in \text{dom } N\}) \mid a \in \text{dom } M\}; \end{aligned}$$

Law $B \times C \rightarrow A \cong (C \rightarrow A)^B$:

$$\begin{aligned} \text{scurry} &: A \times B \xrightarrow{m} C \rightarrow (A \rightarrow B \xrightarrow{m} C) \\ \text{scurry}(M)(a) &\triangleq \\ &\{b \mapsto M(\text{mk-}(a', b)) \mid \text{mk-}(a', b) \in \text{dom } M \cdot a' = a\}; \end{aligned}$$

Law $(C \times A) \rightarrow B \leq C \rightarrow (A \rightarrow B)$

Representation:

$$\begin{aligned} \text{pcurry}[\text{@}A, \text{@}B, \text{@}C] &: (\text{@}C \times \text{@}A) \xrightarrow{m} \text{@}B \rightarrow \text{@}C \xrightarrow{m} (\text{@}A \xrightarrow{m} \text{@}B) \\ \text{pcurry}(f) &\triangleq \\ &\text{let } y = \{x.\#1 \mid x \in \text{dom } f\} \text{ in} \\ &\{a \mapsto \{p.\#2 \mapsto f(p) \mid \\ &\quad p \in \text{dom } f \cdot p.\#1 = a\} \mid \\ &\quad a \in y\}; \end{aligned}$$

Abstraction:

$$\begin{aligned} \text{unpcurry}[\text{@}A, \text{@}B, \text{@}C] &: \text{@}C \xrightarrow{m} (\text{@}A \xrightarrow{m} \text{@}B) \rightarrow (\text{@}C \times \text{@}A) \xrightarrow{m} \text{@}B \\ \text{unpcurry}(f) &\triangleq \\ &\text{merge } \{\text{let } g = f(a) \text{ in} \\ &\quad \{\text{mk-}(a, b) \mapsto g(b) \mid b \in \text{dom } g\} \mid \\ &\quad a \in \text{dom } f\}; \end{aligned}$$

B Auxiliary functions

B.1 Finite sequences

List cons:

$$\begin{aligned} cons[\mathbb{A}] : \mathbb{A} &\rightarrow \mathbb{A}^* \rightarrow \mathbb{A}^* \\ cons(a)(l) &\triangleq \\ [a] \curvearrowright l; \end{aligned}$$

The VDM equivalent of Haskell's *foldr*:

$$\begin{aligned} foldr[\mathbb{A}, \mathbb{B}] : (\mathbb{A} \times \mathbb{B} \rightarrow \mathbb{B}) \times \mathbb{B} &\rightarrow \mathbb{A}^* \rightarrow \mathbb{B} \\ foldr(f, u)(l) &\triangleq \\ \text{if } l = [] & \\ \text{then } u & \\ \text{else } f(\text{hd } l, foldr[\mathbb{A}, \mathbb{B}](f, u)(\text{tl } l)); & \end{aligned}$$

and an application:

$$\begin{aligned} nat1Sum : \mathbb{N}_1^* &\rightarrow \mathbb{N}_1 \\ nat1Sum(L) &\triangleq \\ foldr[\mathbb{N}_1, \mathbb{N}_1](\lambda x : \mathbb{N}_1, y : \mathbb{N}_1 \cdot x + y, 0)(L); & \end{aligned}$$

B.2 Finite sets

Set “cons”ing: function

$$\begin{aligned} puts[\mathbb{A}] : \mathbb{A} \times \mathbb{A}\text{-set} &\rightarrow \mathbb{A}\text{-set} \\ puts(e, S) &\triangleq \\ \{e\} \cup S; & \end{aligned}$$

Also useful is this subrelation of *puts*:

$$\begin{aligned} Puts[\mathbb{A}] : \mathbb{A} \times \mathbb{A}\text{-set} &\rightarrow \mathbb{A}\text{-set} \\ Puts(e, S) &\triangleq \\ puts[\mathbb{A}](e, S) & \\ \text{pre } \neg e \in S & \\ ; & \end{aligned}$$

fmapSet(*f*)(*S*) applies a function *f* to every element of set *S*.

$$\begin{aligned} fmapSet[\mathbb{A}, \mathbb{B}] : (\mathbb{A} \rightarrow \mathbb{B}) &\rightarrow \mathbb{A}\text{-set} \rightarrow \mathbb{B}\text{-set} \\ fmapSet(f)(S) &\triangleq \\ \{f(x) \mid x \in S\}; & \end{aligned}$$

B.3 Projections

$g \cdot M \cdot f^\circ$ in VDM-SL:

```

proj[@A, @B, @C, @D] : (@A → @C) → (@B → @D) → @A  $\xrightarrow{m}$  @B → @C  $\xrightarrow{m}$  @D
proj (f)(g)(M)  $\triangleq$ 
  {f (a)  $\mapsto$  g (M (a)) | a  $\in$  dom M}
pre true
;

```

B.4 Bags

$cardBag(M)$ computes the "cardinal" of a multiset M :

```

cardBag[@A] : @A  $\xrightarrow{m}$   $\mathbb{N}_1 \rightarrow \mathbb{N}_1$ 
cardBag (M)  $\triangleq$ 
  nat1Sum (Listify'[@A] (rng M));

```

$mulBag(n, M)$ is the multiset external multiplication operation (cf. vector spaces):

```

mulBag[@A] :  $\mathbb{N} \times @A \xrightarrow{m} \mathbb{N}_1 \rightarrow @A \xrightarrow{m} \mathbb{N}_1$ 
mulBag (n, M)  $\triangleq$ 
  {t  $\mapsto$  n  $\times$  M (t) | t  $\in$  dom M};

```

$subBag(M, N)$ models the bag-subset partial order:

```

subBag[@A] : (@A  $\xrightarrow{m}$   $\mathbb{N}_1$ )  $\times$  (@A  $\xrightarrow{m}$   $\mathbb{N}_1$ )  $\rightarrow \mathbb{B}$ 
subBag (M, N)  $\triangleq$ 
   $\forall a \in \text{dom } M \cdot a \in \text{dom } N \wedge M(a) \leq N(a)$ ;

```

$diffBag(M, N)$ extends set difference to multisets:

```

diffBag[@A] : (@A  $\xrightarrow{m}$   $\mathbb{N}_1$ )  $\times$  (@A  $\xrightarrow{m}$   $\mathbb{N}_1$ )  $\rightarrow$  (@A  $\xrightarrow{m}$   $\mathbb{N}_1$ )
diffBag (M, N)  $\triangleq$ 
  {a  $\mapsto$  M (a) - N (a) | a  $\in$  dom M  $\cap$  dom N  $\cdot$  M (a) > N (a)};

```

$cupBag(M, N)$ is the multiset union of two multisets (cf. vector addition):

```

cupBag[@A] : (@A  $\xrightarrow{m}$   $\mathbb{N}_1$ )  $\times$  (@A  $\xrightarrow{m}$   $\mathbb{N}_1$ )  $\rightarrow$  (@A  $\xrightarrow{m}$   $\mathbb{N}_1$ )
cupBag (M, N)  $\triangleq$ 
  M  $\dagger$  N  $\dagger$  {a  $\mapsto$  M (a) + N (a) | a  $\in$  dom M  $\cap$  dom N};

```

$cupBagFold(L)$ is the catamorphism associated to $cupBag$:

```

cupBagFold[@A] : (@A  $\xrightarrow{m}$   $\mathbb{N}_1$ )*  $\rightarrow$  @A  $\xrightarrow{m}$   $\mathbb{N}_1$ 
cupBagFold (L)  $\triangleq$ 
  foldr[@A  $\xrightarrow{m}$   $\mathbb{N}_1$ , @A  $\xrightarrow{m}$   $\mathbb{N}_1$ ] (cupBag[@A], { $\mapsto$ }) (L);

```

cupBagRngFold extends *cupBagFold* to a finite collection of bags:

$$\begin{aligned} \text{cupBagRngFold}[\mathbb{A}, \mathbb{B}] : \mathbb{A} \xrightarrow{m} (\mathbb{B} \xrightarrow{m} \mathbb{N}_1) &\rightarrow \mathbb{B} \xrightarrow{m} \mathbb{N}_1 \\ \text{cupBagRngFold}(M) &\triangleq \\ \text{cupBagFold}[\mathbb{B}](\text{Listify}'[\mathbb{B}](\text{rng } M)); \end{aligned}$$

C evdm.sty package

EVDM combines `article.sty` and `vdmsl-2e.sty` with a few other \LaTeX styles in order to produce enhanced and standardized documentation for VDMTOOLS specifications in a *literate programming* style. By using `makeindex` (type `man makeindex`) it automatically generates a cross-reference index which is intended for package maintainance.

EVDM is open to improvement, so please send comments and suggestions. For the moment only the declaration of types and functions is considered.

D \LaTeX packages involved

This file is `mii0506.vdm` and its heading contains:

```
\documentclass{article}
\usepackage{evdm}
```

The other packages

```
\usepackage{times}
\usepackage{a4}
```

are optional but will produce a better graphical output. Package `evdm` itself calls a few others,

```
\usepackage{vdmsl-2e}
\usepackage{makeidx}
\usepackage{pstcol}
```

which are now standard in \LaTeX distributions.

E How to use

Type

```
latex mii0506.vdm.tex
```

once you've generated it, and then

```
dvips -o mii0506.vdm.ps mii0506.vdm.dvi
```

or

```
dvipdf mii0506.vdm
```

to obtain the output. The next time you process the file, type

```
makeindex mii0506.vdm.idx
```

in order to update the cross-reference index (like with \BibTeX).

By the way: in the cross-reference index, $a < b$ means “function a is called by function b ”.

Contents

1	Hash table representation of finite sets	1
1.1	Abstract data type	1
1.1.1	Abstract data models	1
1.1.2	Abstract functionality	1
1.2	Concrete data type	1
1.2.1	Concrete data models	1
1.2.2	Concrete functionality	2
1.3	Refinement step	2
1.3.1	Abstraction function	2
1.3.2	Representation function	2
1.4	Test data	2
2	ECTS problem	3
2.1	Abstract data type	3
2.1.1	Abstract data models	3
2.1.2	Abstract functionality	4
2.1.3	Abstract level test data	4
2.2	Refinement step	5
2.2.1	Data level calculation	5
2.2.2	Abstraction function	6
2.2.3	Representation function	6
2.2.4	Migration of abstract test data	6
3	FS (file system) model	6
3.1	Abstract data type	6
3.1.1	Abstract data models	6
3.1.2	Abstract functionality	6
3.1.3	Abstract level test data	7
4	Pedigree problem	8
4.1	Abstract data type	8
4.1.1	Abstract data model	8
4.1.2	Abstract functionality	8
4.1.3	Abstract level test data	8
4.2	First Refinement step	8
4.2.1	Concrete model	8
4.2.2	Abstraction relation	8
4.2.3	Representation relation	8
4.2.4	Test data at this level	9
4.3	Second Refinement step	9
4.3.1	Concrete model	9
4.3.2	Abstraction relation	9
4.3.3	Representation relation	9
A	Data refinement calculus	10
A.1	Pointer representation	10
A.2	Sets refined by sequences	10
A.3	Mapping refinement calculus	11

B	Auxiliary functions	12
B.1	Finite sequences	12
B.2	Finite sets	12
B.3	Projections	13
B.4	Bags	13
C	evdm.sty package	14
D	\LaTeX packages involved	14
E	How to use	14

Function/Method Cross-Reference Index

A, **10**, 10, 11
absf, **2**, 3
ACMtopic, **3**, 3, 4
allCred, **4**
allPaths, **6**, 6
allTopics, **4**, 4

B, **10**, 10, 11
BorC, **10**, 11

C, **10**, 10, 11
calCred, **4**
cardBag, **13**
Classif, **3**, 3
Collection, **1**, 1–3
cons, **12**
Course, **3**, 3, 5
CourseId, **3**, 3, 4
coursesOk, **3**, 4
cupBag, **13**
cupBagFold, **13**
cupBagRngFold, **14**

D, **10**, 11
Data, **1**, 1, 2
degreesOk, **3**, 4
diffBag, **13**

ECTSs, **3**, 3, 4

F-GenDia, **8**, 8
File, **6**, 6, 7
find, **1**, 2
find-name, **6**
findHT, **2**
fmapSet, **12**
foldr, **12**
FS, **6**, 6, 7

GenDia, **8**, 8
GenDia', **8**, 8, 9
GenDia", **9**

hash, **1**, 2, 2
HTable, **1**, 2, 3

Id, **6**, 6, 7
IdDeg, **3**, 3
ids, **6**, 6
insert, **1**, 2
insertHT, **2**, 3

JustA, **10**, 10
JustB, **10**, 10, 11
JustC, **10**, 10, 11

Listify, **10**
Listify', **10**
Location, **1**, 1, 2

MaybeA, **10**, 10
mulBag, **13**

nat1Sum, **12**, 13
nest, **7**, 7
njoin, **11**
Node, **6**, 6

Path, **6**, 6, 7
pcurry, **11**
prefix, **7**, 7
proj, **13**
Puts, **12**
puts, **12**

repf, **2**, 3

scurry, **11**
StudentId, **3**, 3, 4
subBag, **13**

Tar, **6**, 7
tar, **6**, 7, 7
tarx, **7**, 7
topicsOk, **3**, 4

uncojoin, **11**
Univ, **3**, 4, 5
Unjust, **10**
unnjoin, **11**
unpcurry, **11**