Transformational query solving

Eindhoven University of Technology

Department of Mathematics and Computing Science

Transformational Query Solving

by

Rik van Geldrop

Computing Science Note 91/18
Eindhoven, September 1991

COMPUTING SCIENCE NOTES

This is a series of notes of the Computing Science Section of the Department of Mathematics and Computing Science Eindhoven University of Technology. Since many of these notes are preliminary versions or may be published elsewhere, they have a limited distribution only and are not for review. Copies of these notes are available from the author.

Copies can be ordered from:
Mrs. F. van Neerven
Eindhoven University of Technology
Department of Mathematics and Computing Science
P.O. Box 513
5600 MB EINDHOVEN
The Netherlands
ISSN 0926-4515

Transformational Query Solving

Rik van Geldrop*

July 12, 1991

Abstract

A transformational programming method is used to derive algorthms for a class of database queries. The purpose is twofold: to illustrate a transformational programming method on the basis of a non trivial example and to give a program derivation for database queries (these are scarce in literature). The results of our efforts are two generic algorithms for files (or databases with only one relation) with minimimal restrictions to the system. It is shown how these algorithms may serve as building blocks in solving more complex problems.

*Department of Mathematics and Computing Science, Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands.

1. INTRODUCTION

In the branch of computing science described by information systems, databases play an important role. Several models have been developed for databases, the relational model, [3,4], is one of them. Systems concerned with the implementation of databases are called Database Management Systems (DBMS), the 4^{th} Generation (4G) systems claim to support the relational modelling of databases. In the relational model, based on set theory, specifications for database queries are given in a set theoretical notation. Unfortunately in most 4G DBMS's, the implementation language is a rather restricted set theoretical one and efficiency is mainly a matter of the system. So, if a programmer likes to have control of efficiency, a lower-level system should be used. However, these are restrictive with respect to set operations so that query specifications have to be transformed to the level of files. Since databases may have a complex internal structure, program derivation for database queries becomes very laborious in this (usual) way.

In another branche of computing science, the Bird Meertens formalism (BM), research has been made in the transformation of specifications based on algebraic properties of the underlying data-type [1,2,5]. Here an important role is played by 'structure preserving' functions, so-called morphisms. It is claimed that specifications are programs and executibility depends on the intended machine. Hence one can speak of abstract programs. One of the advantages of such an approach is the separation between the architecture of a solution and the details of its implementation.

Observing that in many database queries the attribute-values to be constructed are resulting from morphism applications, we would like to use the BM tools in our derivations. Program derivation in such a way will be less laborious than a derivation in the usual way.

We don't require that the reader is familiar with the concepts of database theory or BM. The ingredients needed in our derivation will be introduced in section 2. The class of database queries, which we aim to solve, is built up by problems for files. In section 3, the problem class is described and generic algorithms for file problems are derived. How those algorithms apply to database queries is illustrated in section 4. In the algorithms developed, a particular representation is required, and it is claimed that this might be achieved by the system orderings. What can be expected if the representation condition has to be satisfied by the ordering facilities of existing systems is subject of section 5.

2. PRELIMINARIES

In the first part of this section we introduce some notions from database theory and the definition of the (essential part of the) DBMS interface. The second part deals with some basic concepts of BM such as datatypes and transformations (laws). A link between system interface and BM is subject of the last part of this section.

2.1 Databases and their implementing system.
As mentioned before, we intend to use BM in our derivations. This implies that the algebraic properties of the databasetype (the external behaviour) will be exploited. Therefore, an informal description of some database notions will do.
Databasetypes are used to model the data in some organization. Mostly, the data can be divided into several kinds. In the simplest case, which suffices to introduce the notions needed in section 3, only one kind is involved (in database terms: only one *object* exists). An object can be described by its relevant characteristics (in database terms: each object is determined by a set of *attribute-names*). Each attribute name corresponds to an *attribute-value* set. We will assume that in the organization involved, an occurrence of the object (in database terms: a tuple) can be represented by an element of the (labeled) cartesian product of attribute-value sets, the label set being the attribute names. Usually, the organization deals with several occurrences of an object, i.e. a subset of the cartesian product. But, not every subset may represent a possible state of the object, a so-called *table-constraint* has to be satisfied. (A frequently occurring table-constraint is the *key* constraint, e.g. see the definition of SR in section 4.) The subsets of the labeled cartesian product which satisfy the table-constraint constitute the *tabletype* for the object. Common usage in database theory is the word "relation" for an element of a tabletype. Since we are also interested in (binary) relations on elements of a tabletype, we will prevent confusion by using the word "*table*" for the database notion "relation". In this simple case, databases are equivalent to tables.
For the moment these notions suffice. In section 4, where we use our schemes to solve database queries, the general case is described and an example databasetype is given.

Implementations for databases are realized by DBMS's. We prefer to have a hold on efficiency, so 4G systems are left out of consideration. Apart from facilities needed in the solution of our intended problem class, we don't fix the system involved and specify the implementing interface by:

- databases are table-valued functions over the set of objects
- tables may be considered as
  1. sets with operations:

empty,	initialization of a table on $\emptyset$
fetchc,	retrieval of a tuple, given one of its key-values.
store,	insertion of a tuple, if the table-constraint is satisfied.
delete,	deletion of a tuple, if the table-constraint is satisfied.

     (Direct organization of tables.)
  2. files:

     Let α be a type, then File(α) $\triangleq [\alpha] \times \mathbb{N}$. (See 2.2 for the definition of $[\alpha]$.)
     For f : File(α) with f = (F,p), the following operations are available

$$\begin{array}{lll} \textit{start(f)} & \equiv & p := 1 \\ \textit{eof(f)} & \equiv & p = \# F + 1 \\ \textit{fetchn(f,t)} & \triangleq & \textbf{if } 1 \leq p \leq \# F \rightarrow t := F.p;\ p := p + 1 \\ & & [\!]\ p < 1 \vee p > \# F \rightarrow \text{abort} \\ & & \textbf{fi} \end{array}$$

(Sequential organization of tables.)

- There exist ordering facilities for some types.
  Suppose R $\subseteq \alpha \times \alpha$, then Ordfile($\alpha$, R) is the subtype
  $$\text{Ordfile}(\alpha, R) \triangleq \{ (F,p) \in \text{File}(\alpha) \mid \forall\, i,j\colon 1 \leq i \leq j \leq \#F : (F.i,F.j) \in R \}$$
  In the summary of relations on tables (see appendix) the predicate
  ($\forall$ i,j: $1 \leq i \leq j \leq \#F$: (F.i,F.j) $\in$ R) is described by "F satisfies R".

Systems equipped with those operations are called File Management Systems (FMS). Some relevant facts about its interfaces are:
- generally they are embedded in imperative languages,
- FMS operations are time consuming, so the objective is to minimize their use.
Our goal will be the derivation of imperative FMS programs.

2.2 BM tools.
The approach of BM applies to a large class of datatypes and exploits only the algebraic properties of those types. The intention is, after fixing an algebraic datatype, to transform the functional expressions, which can be formed in the language of the algebra, in order to exchange expensive operators for cheaper ones. The datatypes we are interested in are lists and sets, but before we introduce the relevant part of BM for these types, we recapitulate some basic facts about algebra.

An algebra is a set together with a family of operators on the set and a (possibly empty) set of equations which have to hold for the operators.
Example. $\mathcal{M} = < \mathcal{A}, \Sigma, \{ l_0, l_1 \} >$ where

$$\begin{array}{lll} \Sigma & = & \{ + : \mathcal{A} \times \mathcal{A} \rightarrow \mathcal{A},\ 0 : \rightarrow \mathcal{A} \}, \\ l_0 & : & \forall x : 0 + x = x + 0 = x \\ l_1 & : & \forall x,y,z : (x + y) + z = x + (y + z) \end{array}$$

is an algebra. $\mathcal{M}$ models the monoids. Several instances (models) of a given algebra may exist. E.g. the natural numbers with addition, M, and the booleans with disjunction, M', are models for the algebra above. Between two models of the same algebra ("two algebras of the same kind") homomorphisms can be defined.
Let (A, σ) and (A', σ') be algebras of the same kind. A homomorphism h from (A, σ) to (A', σ') is a function h : A $\rightarrow$ A' such that

$$h \circ f = f' \circ h^n \qquad\qquad n \text{ is the arity of } f'$$

for each f $\in \sigma$ and corresponding f' $\in \sigma$'. E.g. in our previous example, a homomorphism h from M to M' is a function h : $\mathbb{N} \rightarrow \mathbb{B}$ such that

$$h.0 = \text{false}$$
$$h.(m + n) = h.m \vee h.n.$$

It is not difficult to prove that (functional) composition preserves homomorphisms. Generally several homomorphisms may exist between two algebras of the same kind, but sometimes there is a model which has exactly one homomorphism to each other model. This (minimal) model is the so-called initial algebra.
The algebra $\mathcal{M}$ of our example has a minimal model and it can be represented by the monoid $(\{1_\oplus\}, \oplus, 1_\oplus)$. $1_\oplus$ is the (only) generator of the type. A set of generators can be obtained by parametrization with a set α as follows.

Consider the algebras

$$\begin{array}{llll}
 & \mathcal{C}_i & = < \mathcal{A}(\alpha), \Sigma, L_i > & i = 0,\ldots,3 \\
\text{where} & \Sigma & = \{ \ + : \mathcal{A}(\alpha) \times \mathcal{A}(\alpha) \rightarrow \mathcal{A}(\alpha) & \\
 & & \quad , \ 0 : \rightarrow \mathcal{A}(\alpha) & \\
 & & \quad , \ \tau : \alpha \rightarrow \mathcal{A}(\alpha) & \\
 & & \quad \} & \\
\text{and} & L_i & = \{ \ l_0,\ldots, l_i \ \} & \\
\text{with} & l_0 & : \ \forall x : 0 + x = x + 0 = x & \\
 & l_1 & : \ \forall x,y,z : (x + y) + z = x + (y + z) & \\
 & l_2 & : \ \forall x,y : x + y = y + x & \\
 & l_3 & : \ \forall x : x + x = x &
\end{array}$$

A homomorphism h from $(A, \{ \oplus, \underline{0}, \underline{\tau} \})$ to $(A', \{ \oplus', \underline{0}', \underline{\tau}' \})$ in $\mathcal{C}_i$ is now defined by

$$h \circ \underline{0} = \underline{0}'$$
$$h \circ \underline{\tau} = \underline{\tau}'$$
$$h.(x \oplus y) = h.x \oplus' h.y$$

It is clear that a homomorphism in $\mathcal{C}_i$ is a homomorphism in $\mathcal{C}_j$, if $i \geq j$. Moreover, homomorphisms in $\mathcal{C}_i$, $i \geq 1$, are homomorphisms in $\mathcal{M}$. From the theory of algebras it is known, that $\mathcal{C}_i$ has an initial algebra IC_i, $i = 0,\ldots,3$. Together these initial algebras constitute the (Boom) type hierarchy[1]. The datatypes that play a role in our application are IC_i, $i \geq 1$. We introduce the following representation for them

$$IC_1 = ([\alpha], \mathbin{+\!\!+}, [\,], [\text{-}]) \text{ with } [\text{-}].a = [a]$$
$$IC_2 = (\prec\alpha\succ, \uplus, \prec\succ, \prec\text{-}\succ) \text{ with } \prec\text{-}\succ.a = \prec a\succ$$
$$IC_3 = (\{\alpha\}, \cup, \emptyset, \{\text{-}\}) \text{ with } \{\text{-}\}.a = \{a\}$$

The set of generators of IC_1 is $\{ [a] \mid a \in \alpha \}$. $[\alpha]$ consists of elements obtained by finitely many $\mathbin{+\!\!+}$ applications on elements of this generator set, so $[\alpha]$ denotes the finite lists over α. Similar remarks can be made for the other initial types. Note that a homomorphism on the initial type is completely determined by its behaviour

[1]The (Boom) type hierarchy, trees - lists - bags - sets, is a hierarchy of binary structures over a given domain. For the definition of binary structures over α, see [7].

on sigletons.

Remark on notation. Here, . denotes functional application, the notation for functional composition is $\circ$. In general, the use of . and $\circ$ will be omitted

So far some basic algebraic facts. Now we can be more detailed about BM and its functional transformations. Often, these transformations are based on homomorphisms in $\mathcal{M}$ (morphisms for short). Special forms of morphisms that are of interest for our application are the following:

- f-map, denoted by f*
  Let $f : \alpha \rightarrow \beta$. Then $f^* : [\alpha] \rightarrow [\beta]$ is such that
  $f^*.[\,] = [\,]$
  $f^*.[a] = [f.a]$
  $f^*.(x +\!\!+\, y) = f^*.x +\!\!+\, f^*.y$
- $\oplus$-reduce, denoted by $\oplus/$
  Let $\oplus : \alpha \times \alpha \rightarrow \alpha$ be associative with unit $1_\oplus$. Then $\oplus/ : [\alpha] \rightarrow \alpha$ is such that
  $\oplus/.[\,] = 1_\oplus$
  $\oplus/.[a] = a$
  $\oplus/.(x +\!\!+\, y) = \oplus/.x \oplus \oplus/.y$
- p-filter, denoted by $p\triangleleft$
  Let $p : \alpha \rightarrow \mathbb{B}$. Then $p\triangleleft : [\alpha] \rightarrow [\alpha]$ is such that
  $p\triangleleft.[\,] = [\,]$
  $p\triangleleft.[a] = \textbf{if}\ p.a \rightarrow [a] \ [\!] \neg\, p.a \rightarrow [\,]\ \textbf{fi}$
  $p\triangleleft.(x +\!\!+\, y) = p\triangleleft.x +\!\!+\, p\triangleleft.y$

f^*, $\oplus/$ and $p\triangleleft$ are (unique) homomorphisms:
$f^* : ([\alpha], +\!\!+, [\,], [\text{-}]) \rightarrow ([\beta], +\!\!+, [\,], [\text{-}]\, f)$
$\oplus/\!: ([\alpha], +\!\!+, [\,], [\text{-}]) \rightarrow (\alpha, \oplus, 1_\oplus, id_\alpha)$
$p\triangleleft\ ([\alpha], +\!\!+, [\,], [\text{-}]) \rightarrow ([\alpha], +\!\!+, [\,], pf)$
with pf.a = [a] if p.a, [ ] otherwise.

For the initial datatypes IC_2 and IC_3, f-map, $\oplus$-reduce and p-filter are defined in an analogous way. Map, reduce and filter are standard functions in BM. Examples of standard functions which are not necessarily morphisms are the directed reductions. We will use left-reduce.

- Left-reduce
  Let $\oplus : \beta \times \alpha \rightarrow \beta$ and $e : \beta$. Then $(\oplus \not\rightarrow e) : [\alpha] \rightarrow \beta$ is such that
  $(\oplus \not\rightarrow e).[\,] = e \quad \wedge \quad (\oplus \not\rightarrow e).(x +\!\!+\, [a]) = (\oplus \not\rightarrow e).x \oplus a$
  or alternatively
  $(\oplus \not\rightarrow e).[\,] = e \quad \wedge \quad (\oplus \not\rightarrow e).([a] +\!\!+\, x) = (\oplus \not\rightarrow (e \oplus a)).x$

For our purpose the relevant transformations are:

L1. the composition of morphisms is a morphism

L2. promotion law. If h : $(\alpha,\oplus,1_\oplus) \rightarrow (\beta,\otimes,1_\otimes)$ is a morphism, then
$$h \oplus/ = \otimes/ h^*$$
L3. h is a morphism iff $\exists \oplus, f : h = \oplus/ f^*$
L4. $f^* g^* = (f\, g)^*$
L5. filter-map rule.
$$\varphi\triangleleft\ f^* = f^*\ (\varphi\ f)\triangleleft$$
L6. a morphism can be written as a directed reduction.
If $h = \oplus/ f^*$ then $h = (\ominus \not\rightarrow 1_\oplus)$, where
$$u \ominus a = u \oplus f.a$$
L7. formal differentiation. Let $\oplus : [\beta] \times \alpha \rightarrow \beta$, $G : [\beta] \rightarrow \gamma$, $e : [\beta]$. Suppose $\odot : \gamma \times \alpha \rightarrow \gamma$ is such that $G.(y \oplus a) = G.y \odot a$, then
$$G\ (\oplus \not\rightarrow e) = (\odot \not\rightarrow G.e)$$

Not only morphisms can be written as a directed reduction, non-morphisms may have a directed reduction form too. However, finding such a form may cause intricate calculations.

2.3 Linking system interface and BM tools.

We aim at using BM in developing FMS algorithms, so we need some link between the two approaches. As mentioned before, BM is concerned with the algebraic properties of functions, not with the implementation of functions and their evaluations. So, if a programmer wants to compute a function application he has a new problem at hand outside the scope of BM. He has to consider this as a specification in a suitable new programming environment. The use of BM in the development of programs is mainly the calculation of transformations of (possible) equivalent defining forms of the functions involved. In particular, if one is interested in applying standard functions of BM certain standard programs in other environments arise. Since the functions we use are restricted to those that are in fact directed reductions, we will give such a standard program for them only. Our interest is in imperative File algorithms, but due to the few algebraic properties of files, a direct link from BM to File level is rather complicated. An intermediate level may simplify the desired link and Conslists is an appropiate candidate for it:

- morphism applications can be computed by directed reductions
- left reductions are easy to implement as functions over conslists
- operations on files look slightly like conslists ones. (2.3.a)

In the sequel, we will refer to the following lemma as the standard conversion of a left-reduce.

<u>Lemma 1</u>

Let $\ominus : \beta \times \alpha \rightarrow \beta$ and $e : \beta$. Then $\mathcal{S}_1$ is a correct program fragment for the computation of $w = (\ominus \not\rightarrow e)x$.

$\mathcal{S}_1$: w := e; r := x

; **do** $r \neq [\,]$
$\rightarrow$ a := *first* r; r := *tail* r
; w := w $\ominus$ a
od

Invariant: $(\ominus \not\rightarrow w)r = (\ominus \not\rightarrow e)x \wedge r \in$ *tails* x

Proof. Follows immediately from the definition of left-reduce. □

Note that this computation of a left-reduce on conslists leads to an iterative program with time-complexity $\mathcal{O}(N)$, if $\#\ x = N$. Since these costs are relative to the costs of $\ominus$, an efficiency comparison of the computation of $(\oplus \not\rightarrow f)x$ and $(\ominus \not\rightarrow e)x$ can be made from the definitions of $\oplus$ and $\ominus$.

3. GENERIC ALGORITHMS FOR FILES

In this section, we solve some problem classes for files. The classes are interrelated: a fundamental problem has an interesting subproblem and can be extended in several ways. The fundamental problem and its subproblem are solved via transformations in 3.2 and 3.1 and with these solutions, we solve the extensions in 3.3. Our transformational solution method consists of two steps: First, a transformation from Sets to Conslists is made by using BM laws. This step is followed by a standard conversion to conslists. Afterwards, the resulting Conslists program is implemented by an FMS algorithm. The correctness of this step is based on statespace transformations.
To avoid repeating assumptions in our solutions, we will use the conventions that α and β are types, $\otimes$ is an associative, commutative and idempotent operator on β with unit $1_\otimes$, $h : (\{\alpha\}, \cup, \emptyset) \rightarrow (\beta, \otimes, 1_\otimes)$ is a morphism, E is an equivalence relation on α, $X \in \{\alpha\}$ and $q : X \rightarrow X/E$ defined by

$$q.t = \{\ v \in X \mid v\ E\ t\ \}.$$

is the quotientmap.

3.1 FMS algorithm to compute w = hX.

Set specification : $w = hX$ (3.1.a)

step 1. Let sets of α-elements be implemented by lists with representation function $\mathcal{R}_s$

$$\mathcal{R}_s = \cup/\{\text{-}\}^*$$

and representation invariant $P_s(x)$

$$P_s(x) : \#\ \mathcal{R}_s\ x = \#\ x$$

(Informal: a representation contains no duplicates.)
We have to find a function $\tilde{h}$ such that $\tilde{h} = h\ \mathcal{R}_s$.
Graphically

$$\begin{tikzcd}
\begin{array}{c}\textit{abstract level}\\ (\{\alpha\},\cup,\emptyset)\end{array} \arrow[d, "h"] & & \begin{array}{c}\textit{concrete level}\\ ([\alpha],+\!\!+,[\,])\end{array} \arrow[ll, "\mathcal{R}_s"'] \arrow[d, "\tilde{h}"] \\
(\beta,\otimes,1_\otimes) & & (\beta,\otimes,1_\otimes) \arrow[ll, "id"]
\end{tikzcd}$$

Since $\tilde{h}$ is a composition of morphisms it is a morphism itself. Furthermore, it holds that

$$\begin{array}{ll}
 & \tilde{h} \\
= & \\
 & h\ \mathcal{R}_s \\
= & \quad \{\ \text{def}\ \mathcal{R}_s\ \} \\
 & h \cup/\ \{\text{-}\}^* \\
= & \quad \{\ \text{L2}\ \} \\
 & \otimes/\ h^*\ \{\text{-}\}^* \\
= & \quad \{\ \text{L4}\ \} \\
 & \otimes/(h\ \{\text{-}\})^*
\end{array}$$

and using L6, $\tilde{h}$ can be written as a directed reduction:

$$\tilde{h} = (\ominus \not\twoheadrightarrow 1_\otimes) \tag{3.1.b}$$

where $\ominus : \beta \times \alpha \rightarrow \beta$ is defined by $u \ominus a = u \otimes h\{a\}$. Given a representation x of X, we may instantiate the standard conversion of a left-reduce (Lemma 1).

Conslist program for (3.1.a)

```
   w := 1⊗; r := x
 ; do  r ≠ [ ]
   →  a := first r; r := tail r                    (3.1.c)
      ; w := w ⊗ h{a}
   od
```

step 2. Let conslists be implemented by files. Let f : File(α) then f = (F,p). The representation function $\mathcal{R}_c$ is given by

$$\mathcal{R}_c(\text{F,p}) = \begin{cases} [\,] & \text{if } p = \#F + 1 \\ [F.p] +\!\!+ \mathcal{R}_c(F, p+1) & \text{otherwise} \end{cases}$$

and the representation invariant is

$$1 \le p \le \#\ F + 1 \wedge F \text{ is a bijection over } [1..\#\ F]$$

Given a representation f of x, an FMS algorithm for (3.1.c) requires implementations for the

- initialisation r := x,
- guard $r \neq [\,]$,
- statement sequence a := *first* r; r := *tail* r.

Straightforward are r := x $\mapsto$ start(f) and $r \neq [\,] \mapsto \neg$ eof(f). It is not difficult to prove that, from the validity of

$$\{Q^{a,r}_{first\ r, tail\ r)}\}\ a := \mathit{first}\ r;\ r := \mathit{tail}\ r\ \{Q\}$$

it follows that

$$\{(Q^{a,r}_{first\ r, tail\ r})^{r}_{\mathcal{R}_c(f)}\}\ \mathrm{fetchn(f, a)}\ \{Q^{r}_{\mathcal{R}_c(f)}\}$$

holds. So fetchn(f, a) is a correct implementation of a := *first* r; r := *tail* r. Applying those statement transformations to (3.1.c) results in a solution for the subproblem.

<u>FMS algorithm for (3.1.a)</u>

$$\begin{array}{ll} & w := 1_{\otimes};\ \mathrm{start(f)} \\ ; & \mathbf{do}\ \neg\ \mathrm{eof(f)} \\ & \rightarrow\ \mathrm{fetchn(f, a)} \\ & \quad ;\ w := w \otimes h\{a\} \\ & \mathbf{od} \end{array} \qquad (3.1.d)$$

Some remarks on this derivation

1. *Tupled version for* (3.1.a). Instantiate (3.1.a) with the (product) morphism

   $$h = \langle h_1,...,h_m\rangle : (\{\alpha\}, \cup, \emptyset) \rightarrow (\beta, \otimes, 1_{\otimes}),$$

   where $h_i : (\{\alpha\}, \cup, \emptyset) \rightarrow (\beta_i, \otimes_i, 1_{\otimes_i})$, $1 \leq i \leq m$, is a morphism, $\beta = \beta_1 \times ... \times \beta_m$, $\otimes = (\otimes_1,...,\otimes_m)$ is defined by

   $$(b_1,...,b_m)\ (\otimes_1,...,\otimes_m)\ (c_1,...,c_m) = (b_1 \otimes_1 c_1,...,b_m \otimes_m c_m)$$

   $1_{\otimes} = (1_{\otimes_1},...,1_{\otimes_m})$ and $h\ X = \langle h_1\ X,...,h_m\ X\rangle$. Then the computation of h X might be done "componentwise".

   $$\begin{array}{ll} & w_1,...,w_m := 1_{\otimes_1},...,1_{\otimes_m};\ \mathrm{start(f)} \\ ; & \mathbf{do}\ \neg\ \mathrm{eof(f)} \\ & \rightarrow\ \mathrm{fetchn(f, a)} \\ & \quad ;\ w_1,...,w_m := w_1 \otimes_1 h_1\{a\},...,\ w_m \otimes_m h_m\{a\} \\ & \mathbf{od} \end{array} \qquad (3.1.e)$$

2. *Conditional version for* (3.1.a). Instantiate (3.1.a) with the morphism $h\ \varphi\triangleleft$, where $\varphi : \alpha \rightarrow \mathbb{B}$. I.e. generalize the specification by

   $$w = h\ \{\ x \in X \mid \varphi\ x\ \} \qquad (3.1.a')$$

Again we calculate a directed reduction, now for the morphism h $\varphi\lhd$ $\mathcal{R}_s$. This yields h $\varphi\lhd$ $\mathcal{R}_s = (\ominus \;\not\rightarrow\; 1_\otimes)$, where

$$u \ominus a = \textbf{if}\ \varphi\ a \rightarrow u \otimes h\ \{\ a\}\ [] \neg\ \varphi\ a \rightarrow u\ \textbf{fi},$$

which results in the following FMS algorithm for (3.1.a')

```
    w := 1⊗; start(f)
  ; do  ¬ eof(f)
    →  fetchn(f, a)                                        (3.1.d')
       if  φ a   →   w := w ⊗ h { a }
       []  ¬φ a →   skip
       fi
    od
```

3. *Superfluous conditions on h.* The representation invariant guarantees that $\mathcal{R}_s$ doesn't appeal to the idempotency of $\cup$ (the commutativity is used since $\mathcal{R}_s$ [a, b] = $\mathcal{R}_s$ [b, a]). This implies that the same derivation can be made for the application of bag homomorphisms. Let k : $(\prec \alpha \succ, \uplus, \prec\succ) \rightarrow (\beta, \oplus, 1_\oplus)$ be a morphism with commutative $\oplus$. Then the following FMS algorithm is correct.

```
    w := 1⊕; start(f)
  ; do ¬ eof(f)
    → fetchn(f, a)                                         (3.1.f)
      ; w := w ⊕ k{a}
    od
```

4. For (3.1.a') and (3.1.e) a remark similar to remark 3 can be made. Note that k $\varphi\lhd$: $(\prec \alpha \succ, \uplus, \prec\succ) \rightarrow (\beta, \oplus, 1_\oplus)$ is a morphism with commutative $\oplus$, if k is. Lemma 2 follows.

<u>Lemma 2</u>

Let X $\in \{\alpha\}$ and f $\in$ File(α) such that f is a representation for X.[2]
Let k : $(\prec \alpha \succ, \uplus, \prec\succ) \rightarrow (\beta, \oplus, 1_\oplus)$ be a morphism with commutative $\oplus$.
Then $\mathcal{S}_2$ is a correct FMS algorithm to compute k X, where

```
S2:      |[ X : {α}; f : File(α); k : (≺ α ≻, ⊎, ≺≻) → (β, ⊕, 1⊕);
           |[ w : β
             |[ a : α;
               w := 1⊕; start(f)
             ; do ¬ eof(f)
               → fetchn(f, a)
```

[2]We require that the representation invariant will be established by the declaration mechanism of the system.

; w := w $\oplus$ k{a}
od

]|]|]|

Conditional and tupled versions of Lemma 2 are straightforward.

3.2 FMS algorithm to compute W = { h.q.t | t $\in$ X }

Set specification : W = h*q* X (3.2.a)
(h*q* X is the BM notation for { h.q.t | t $\in$ X }.)

As mentioned before, we use a transformational solution method consisting of two steps. The method is illustrated in 3.1 on the basis of a simple problem. Here we have to tackle a special case of (3.1.a) because h*q* is a morphism too. There is no indication to change our former strategy, so we take the same approach as in 3.1.

step 1. Let elements of $\{\alpha\}$ be implemented by lists with representation function $\mathcal{R}_s$ and representation invariant $P_s(x)$ as defined in step 1 of 3.1. Assume $\{\beta\}$ is an available type. We have to find a function $\tilde{H}$ such that $\tilde{H} = h^*\, q^*\, \mathcal{R}_s$.
Graphically

$$\begin{tikzcd}
\textit{abstract level} & & \textit{concrete level} \\
(\{\alpha\},\cup,\emptyset) \arrow[d, "q^*"] & & ([\alpha],+\!\!+,[\,]) \arrow[ll, "\mathcal{R}_s"'] \arrow[dd, "\tilde{H}"] \\
(\{\{\alpha\}\},\cup,\emptyset) \arrow[d, "h^*"] & & \\
(\{\beta\},\cup,\emptyset) & & (\{\beta\},\cup,\emptyset) \arrow[ll, "id"]
\end{tikzcd}$$

$\tilde{H}$ is a morphism and, since $h^*\, q^*\, \cup/\ \{\text{-}\}^* = \cup/(h^*\, q^*\, \{.\})^*$, it holds that

$$\tilde{H} = (\oslash \not\twoheadrightarrow \emptyset),$$

where

$$u \oslash a = u \cup \{\ h.q.a\} \qquad (3.2.b)$$

At this point in 3.1, we converted the directed reduction to Conslist level. We don't do this now because we aim at an efficient implementation of $(\oslash \not\twoheadrightarrow \emptyset)$. To achieve such an implementation we have to minimize the number of $\cup$-operations and find an efficient computation of h.q.a given a representation x of X. Although this last task resembles the subproblem of 3.1, it must be noted that the definition of q.a is relative to the implicit universe X, i.e. (E a)$\lhd$ x is a representation for q.a. Without any assumptions about x, an $\mathcal{O}(N^2)$ algorithm may be obtained for $\tilde{H}$ x, if # x = N. Therefore, we shall assume that equivalent elements in X are consecutive in x.

In the appendix, this (ordering) constraint is described formally and we introduced the notion 'x is an E-segmentation' for it.

Representation requirement: Elements of $\{\alpha\}$ are implemented by E-segmentations.

For X $\in \{\alpha\}$ represented by an E-segmentation, a representation for its partition is easy to construct. Define a function $\theta : [\alpha] \rightarrow [[\alpha]]$ by

$$\theta[\,] = [\,]$$
$$\theta[a] = [[a]]$$
$$\theta(y +\!\!+ [a]) = \begin{cases} \textbf{if} & a \; E \; \textit{first last } \theta y \rightarrow \textit{init } \theta y +\!\!+ [\textit{last } y +\!\!+ [a]] \\ [] & \neg \; a \; E \; \textit{first last } \theta y \rightarrow \theta y +\!\!+ [[a]] \\ \textbf{fi} & \end{cases}$$

θ has several interesting properties, e.g.

$$q^* \mathcal{R}_s = \mathcal{R}_s{}^* \mathcal{R}_s \, \theta \tag{3.2.c}$$
$$x = [\,] \equiv \theta x = [\,] \tag{3.2.d}$$
$$x \neq [\,] \Rightarrow \forall \; y \in \theta x : y \neq [\,] \tag{3.2.e}$$

Moreover, $P_s(\theta x)$ holds, i.e. there are no duplicates in θx, or equivalently, each equivalence class of X corresponds to exactly one element of θx.

Unfortunately, θ is a not a morphism from $([\alpha], +\!\!+, [\,])$ to $([[\alpha]], +\!\!+, [\,])$. (This is easily seen by taking a list of two equivalent elements [a,b]. Then $\theta[a,b] = [[a,b]] \neq [[a],[b]] = \theta[a] +\!\!+ \theta[b]$.) However, from its definition it is clear that there exists an directed reduction form for θ

$$\theta = (\odot \not\rightarrow [\,]) \tag{3.2.f}$$

where

$$u \odot a = \begin{cases} [[a]] & \text{if } u = [\,] \\ \textit{init } u +\!\!+ [\textit{last } u +\!\!+ [a]] & \text{if } a \; E \; (\textit{first last } u) \\ u +\!\!+ [[a]] & \text{if } \neg \, a \; E \; (\textit{first last } u) \end{cases}$$

Remark. Replacing *first* by *any* results in a more general form for $\odot$. Here, the choice for *first* arose from the conslist level.

(3.2.c) expresses that the elements of θx are representations of equivalence classes. Since we are interested in efficiency, we will compare the directed reduction for $q^*\mathcal{R}_s x$ and θx. In (3.2.f) the directed reduction for θ is given. For $q^*\mathcal{R}_s$ we know that $q^*\mathcal{R}_s = q^* \cup/ \{\text{-}\}^* = \cup/(q\,\{\text{-}\})^*$, so (L6),$q^*\mathcal{R}_s = (\oplus \not\rightarrow \emptyset)$ where

$$u \oplus a = u \cup \{ q.a\}$$

Clearly, a computation via θ is more efficient, moreover we implicitly satisfy the remaining task in improving efficiency, the minimization of $\cup$-operations.

Convinced that our approach can be improved by adding the intermediate level of $([[\alpha]], +\!\!+, [\,])$, we are left with the remaining problem of computing $h^*\mathcal{R}_s{}^*\mathcal{R}_s$ if θx is known. Since θx consists of representations of equivalence classes, we will explore the use of 3.1, where we showed how morphism application on a set can be computed if a representation of the set is known. Let $\tilde{\mathrm{h}}$ be such as introduced in 3.1, then

$$\mathrm{h}^* \mathcal{R}_s{}^* \mathcal{R}_s = \mathcal{R}_s\, \tilde{\mathrm{h}}^* \tag{3.2.g}$$

because both, $\mathrm{h}^* \mathcal{R}_s{}^* \mathcal{R}_s$ and $\mathcal{R}_s\, \tilde{\mathrm{h}}^*$ are morphisms and

$$\begin{aligned}
&\mathrm{h}^* \mathcal{R}_s{}^* \mathcal{R}_s[\mathrm{x}] \\
=\ &\\
&\mathrm{h}^* \mathcal{R}_s{}^* \{\mathrm{x}\} \\
=\ &\\
&\{\mathrm{h}\ \mathcal{R}_s\ \mathrm{x}\} \\
=\ &\\
&\mathcal{R}_s\ [\tilde{\mathrm{h}}\ \mathrm{x}] \\
=\ &\\
&\mathcal{R}_s\ \tilde{\mathrm{h}}^*\ [\mathrm{x}]
\end{aligned}$$

Summarizing these considerations, we conclude that driven by efficiency the computation of $\tilde{\mathrm{H}}$ x has to be refined by

$$\begin{aligned}
\mathrm{xs} &= (\odot \not\rightarrow [\,]\,)\ \mathrm{x} \\
\mathrm{W} &= \mathcal{R}_s\ \tilde{\mathrm{h}}^*\ \mathrm{xs}
\end{aligned} \tag{3.2.h}$$

Graphically

$$\begin{tikzcd}
\textit{abstract level} & & \textit{concrete level} \\
(\{\alpha\},\cup,\emptyset) \arrow[d, "q^*"] & & ([\alpha],+\!\!+,[\,]\,) \arrow[ll, "\mathcal{R}_s"'] \arrow[d, "\theta"] \\
(\{\{\alpha\}\},\cup,\emptyset) \arrow[d, "h^*"] & & ([[\alpha]],+\!\!+,[\,]\,) \arrow[ll, "\mathcal{R}_s{}^*\mathcal{R}_s"'] \arrow[d, "\mathcal{R}_s\,\tilde{h}^*"] \\
(\{\beta\},\cup,\emptyset) & & (\{\beta\},\cup,\emptyset) \arrow[ll, "id"]
\end{tikzcd}$$

In contrast to our previous approach, it is not immediately clear that the successive computations in (3.2.h) can be replaced by the computation of one directed reduction, if a representation x for X is given. Therefore, we try to apply formal differentiation, L7.
Since $\odot$: $[[\alpha]] \times [\alpha] \rightarrow [[\alpha]]$, $\mathcal{R}_s\tilde{\mathrm{h}}^*$: $[[\alpha]] \rightarrow \{\beta\}$ and $[\,]$: $[[\alpha]]$, we look for an operator $\oplus$: $\{\beta\} \times [\alpha] \rightarrow \{\beta\}$ such that $\mathcal{R}_s\ \tilde{\mathrm{h}}^*(\odot \not\rightarrow [\,]\,) = (\oplus \not\rightarrow \emptyset)$.

The construction base is correct: $\mathcal{R}_s\ \tilde{\mathrm{h}}^*\ (\odot \not\rightarrow [\,]\,)\ [\,] = \emptyset = (\oplus \not\rightarrow \emptyset)\ [\,]$.

Construction hypothesis: $\mathcal{R}_s\ \tilde{\mathrm{h}}^*\ \theta\mathrm{y} = (\oplus \not\rightarrow \emptyset)\ \mathrm{y}$

Step: $\quad \mathcal{R}_s\ \tilde{\mathrm{h}}^*(\odot \not\rightarrow [\,]\,)\ (\mathrm{y} +\!\!+ [\mathrm{a}])$

case 1 $\equiv$ { case 1: y = [ ] , $\odot$ }

$\mathcal{R}_s\ \tilde{h}^*\ [[a]]$

$=$ { def * }

$\mathcal{R}_s\ [\tilde{h}[a]]$

$=$ { def $\tilde{h}$ and $\mathcal{R}_s$ }

$\{h\{a\}\}$

case 2 { case 2: $y \neq [\ ] \wedge a\ E\ (\mathit{first\ last}\ \theta y)$ }

$\mathcal{R}_s\ \tilde{h}^*\ (\mathit{init}\ \theta y \mathbin{+\!\!+} [\mathit{last}\ \theta y \mathbin{+\!\!+} [a]])$

$=$ { $\mathcal{R}_s\ \tilde{h}^*$ is a morphism, def * and $\mathcal{R}_s$ }

$\mathcal{R}_s\ \tilde{h}^* \mathit{init}\ \theta y \cup \{\tilde{h}\ (\mathit{last}\ \theta y \mathbin{+\!\!+} [a])\}$

$=$ { $\tilde{h} = (\ominus \not\rightarrow 1_\otimes)$, (3.1.b) }

$\mathcal{R}_s\ \tilde{h}^*\ \mathit{init}\ \theta y \cup \{\tilde{h}\ \mathit{last}\ \theta y \otimes h\{a\}\}$

case 3 { case 3: $y \neq [\ ] \wedge \neg(a\ E\ (\mathit{first\ last}\ \theta y))$ }

$\mathcal{R}_s\ \tilde{h}^*\ (\theta y \mathbin{+\!\!+} [[a]])$

$=$ { $\mathcal{R}_s\ \tilde{h}^*$ is a morphism, def *, $\mathcal{R}_s$ and $\tilde{h}$ }

$\mathcal{R}_s\ \tilde{h}^*\ \theta y \cup \{h\{a\}\}$

These calculations show that we may succeed in obtaining a directed reduction for (3.2.h) if we tuple the essential components: $\mathcal{R}_s\ \tilde{h}^*\ \theta y$, $\mathcal{R}_s\ \tilde{h}^*\ \mathit{init}\ \theta y$, $\tilde{h}\ \mathit{last}\ \theta y$ and *first last* θy. Consequently, the type of $\oplus$ has to be extended and the construction hypothesis will be strengthened in an appropriate way. To adjust the construction base, we define

$\mathit{init}\ [\ ] = [\ ]$
$\mathit{last}\ [\ ] = [\ ]$
$\mathit{first}\ [\ ] = \omega_\alpha$

where ω_α is a fictitious element of α. It follows that

$\mathcal{R}_s\ \tilde{h}^*\ \mathit{init}\ \theta[\ ] = \emptyset$
$\tilde{h}\ \mathit{last}\ \theta[\ ] = 1_\otimes$
$\mathit{first\ last}\ \theta[\ ] = \omega_\alpha$

We continue our construction for the three additional components.

Strengthened construction hypothesis:

$(\mathcal{R}_s\ \tilde{h}^*\ \theta,\ \mathcal{R}_s\ \tilde{h}^*\ \mathit{init}\ \theta,\ \tilde{h}\ \mathit{last}\ \theta,\ \mathit{first\ last}\ \theta)\ y = (\oplus \not\rightarrow (\emptyset, \emptyset, 1_\otimes, \omega_\alpha))\ y$

Step: $\mathcal{R}_s\ \tilde{h}^*\ \mathit{init}\ \theta(y \mathbin{+\!\!+} [a])$

case 1,2 $\equiv$ { case 1 and 2: $\mathit{init}\ \theta(y \mathbin{+\!\!+} [a]) = \mathit{init}\ \theta y$ }

$\mathcal{R}_s\ \tilde{h}^*\ \mathit{init}\ \theta y$

case 3 { case 3: $\mathit{init}\ \theta(y \mathbin{+\!\!+} [a]) = \theta y$ }

$\mathcal{R}_s\ \tilde{h}^*\ \theta y$

Step: $(\tilde{h}, \mathit{first})\ \mathit{last}\ \theta(y \mathbin{+\!\!+} [a])$

case 1 $\equiv$ { case 1: $\mathit{last}\ \theta(y \mathbin{+\!\!+} [a]) = [a]$, def $\tilde{h}$ }

$(h\{a\}, a)$

case 2 { case 2: $\mathit{last}\ \theta(y \mathbin{+\!\!+} [a]) = \mathit{last}\ \theta y \mathbin{+\!\!+} [a]$, (3.1.b), (3.2.e) }

$(\tilde{h}\ \mathit{last}\ \theta y \otimes h\ \{a\}, \mathit{first}\ \mathit{last}\ \theta y)$

case 3 { case 3: $\mathit{last}\ \theta(y \mathbin{+\!\!+} [a]) = [a]$, (3.1.b) }

$(h\{a\}, a)$

The recognition of case 1 seems to be a final obstacle in the definition of $\oplus$, but by (3.2.d) it can be shown that

$$y = [\,] \quad \equiv \quad \theta y = [\,] \quad \equiv \quad \mathit{first}\ \mathit{last}\ y = \omega_\alpha \tag{3.2.i}$$

Summarizing our calculations, we have constructed an operator $\oplus$ of type $(\{\beta\} \times \{\beta\} \times \beta \times \alpha) \times \alpha \to (\{\beta\} \times \{\beta\} \times \beta \times \alpha)$ defined by

$$(V, I, w, pt) \oplus a = \tag{3.2.j}$$

$$\begin{cases} (\ \{h\{a\}\}, \emptyset, h\{a\}, a\ ) & \text{if } pt = \omega_\alpha \\ (\ I \cup \{w \otimes h\{a\}\}, I\ , w \otimes h\{a\}, pt) & \text{if } pt \neq \omega_\alpha\ \wedge\ pt\ E\ a \\ (\ V \cup \{h\{a\}\}, V, h\{a\}, a) & \text{if } pt \neq \omega_\alpha\ \wedge\ \neg\ pt\ E\ a \end{cases}$$

such that

$$W = \pi_1\ (\oplus\ \not\rightarrow\ (\emptyset, \emptyset, 1_\otimes, \omega_\alpha))\ x$$

Now we instantiate the standard conversion of a left-reduce

```
  W, I, w, pt := ∅, ∅, 1⊗, ωα; r := x
; do r ≠ [ ]
  → a := first r; r := tail r
   ; if  pt = ωα→  W, w, pt := {h{a}}, h{a}, a
     []  pt ≠ ωα→  if  pt E a →  W, w := I∪{w⊗h{a}}, w⊗h{a}
                   []¬ pt E a →  W∪{h{a}}, W, h{a}, a
                   fi
     fi
  od
```

This code can be
- *smoothened* by unfolding the repetition, since only initially $pt = \omega_\alpha$ holds.

- *improved*, because $W = I \cup \{w\}$ is a property of the 4-tuple, if $pt \neq \omega_\alpha$.

Conslist program for (3.2.a)

$$
\begin{array}{l}
\quad W := \emptyset;\ r := x \\
;\ \textbf{if}\ r = [\,] \rightarrow \text{skip} \\
\ [\!]\ r \neq [\,] \rightarrow a := \textit{first}\ r;\ r := \textit{tail}\ r \\
\qquad\qquad ;\ w, pt := h\{a\}, a \\
\qquad\qquad ;\ \textbf{do}\ r \neq [\,] \\
\qquad\qquad\quad \rightarrow a := \textit{first}\ r;\ r := \textit{tail}\ r \\
\qquad\qquad\qquad ;\ \textbf{if}\ pt\ E\ a \rightarrow w := w \otimes h\{a\} \\
\qquad\qquad\qquad\quad [\!]\ \neg\ pt\ E\ a \rightarrow W, w, pt := W \cup \{w\}, h\{a\}, a \\
\qquad\qquad\qquad\quad \textbf{fi} \\
\qquad\qquad\quad \textbf{od} \\
\qquad\qquad ;\ W := W \cup \{w\} \\
\quad \textbf{fi}
\end{array}
\tag{3.2.k}
$$

step 2. Let conslists be implemented by files. To meet the representation requirement, only system orderings can be useful. In the appendix, Lemma p.33, it is proved that an appropriate system ordering R establishes an $R \cap R^{\leftarrow}$ segmentation. A representation f of x can be given if an appropriate system ordering exists such that $E = R \cap R^{\leftarrow}$.

Assumption: $E = R \cap R^{\leftarrow}$ and cqo R is a system ordering.
Let $f : \text{Ordfile}(\alpha, R)$ and $f = (F,p)$. Let $\mathcal{R}_c$ be the representation function as given in 3.1, with the same representation invariant. In the same way as in 3.1, (3.2.k) can be transformed to an FMS algorithm.

FMS algorithm for (3.2.a)

$$
\begin{array}{l}
\quad W := \emptyset;\ \text{start}(f) \\
;\ \textbf{if}\ \text{eof}(f) \rightarrow \text{skip} \\
\ [\!]\ \neg\ \text{eof}(f) \rightarrow \text{fetchn}(f,a) \\
\qquad\qquad ;\ w, pt := h\{a\}, a \\
\qquad\qquad ;\ \textbf{do}\ \neg\ \text{eof}(f) \\
\qquad\qquad\quad \rightarrow \text{fetchn}(f,a) \\
\qquad\qquad\qquad ;\ \textbf{if}\ pt\ E\ a \rightarrow w := w \otimes h\{a\} \\
\qquad\qquad\qquad\quad [\!]\ \neg\ pt\ E\ a \rightarrow W, w, pt := W \cup \{w\}, h\{a\}, a \\
\qquad\qquad\qquad\quad \textbf{fi} \\
\qquad\qquad\quad \textbf{od} \\
\qquad\qquad ;\ W := W \cup \{w\} \\
\quad \textbf{fi}
\end{array}
\tag{3.2.l}
$$

Some remarks on this derivation

1. A *tupled version* for 3.2.a is obtained by instantiating h with the product morphism defined in 3.1, remark 1. The elaborations are straightforward.

2. *Conditional versions for* 3.2.a. An instantiation of (3.2.a) with the morphism $\mathrm{h}\ \varphi\triangleleft$ yields a conditional computation for w (see 3.1, remark 2), resulting in a conditional version of (3.2.1). Independent of this instantiation, we can generalize (3.2.a) to the computation of

$$\mathrm{W} = \mathrm{h}^*\mathrm{q}^*\psi\triangleleft\ \mathrm{X} \tag{3.2.a'}$$

where $\psi \in \alpha \rightarrow \mathbb{B}$ such that

$$\mathrm{u\ E\ t} \Rightarrow\ \psi\ \mathrm{u} = \psi\ \mathrm{t} \tag{**}$$

Note that (**) means that we may define a predicate ψ' on classes such that $\psi'\triangleleft\ \mathrm{q}^* = \mathrm{q}^*\ \psi\triangleleft$, e.g. $\psi' = \mathit{some}\ \psi$, where $\mathit{some}\ \psi = \vee/\ \psi^*$. Consequently, we may interchange partitioning and filtering and transform (3.2.a') into

$$\mathrm{W} = \mathrm{h}^*\ \psi'\triangleleft\ \mathrm{q}^*\ \mathrm{X} \tag{3.2.b'}$$

The derivation of an FMS algorithm for (3.2.b') follows the same line as the one for (3.2.a), only some slight modifications have to be made.
- A generalization of (3.2.g) is needed

$$\mathrm{h}^*\mathcal{R}_s{}^*\varphi\triangleleft\ \mathcal{R}_s = \mathcal{R}_s\ \tilde{\mathrm{h}}^*\varphi\triangleleft \tag{3.2.g'}$$

Its proof is completely analogous to that of (3.2.g).
- (3.2.h) has to be modified into

$$\begin{aligned} &\mathrm{xs} = (\odot\ \not\rightarrow\ [\,]\,)\ \mathrm{x} \\ &\mathrm{W} = \mathcal{R}_s\ \tilde{\mathrm{h}}^*\ \psi''\triangleleft \end{aligned} \tag{3.2.h'}$$

where ψ'' is used as an abbreviation for $\psi'\ \mathcal{R}_s$. The correctness of this refinement follows immediately from the filter-map rule and (3.2.g')

$$\begin{aligned} & \mathrm{h}^*\ \psi'\triangleleft\ \mathcal{R}_s{}^*\ \mathcal{R}_s \\ = \quad & \{\ \mathrm{L5}\ \} \\ & \mathrm{h}^*\ \mathcal{R}_s{}^*\ (\psi'\ \mathcal{R}_s)\triangleleft\ \mathcal{R}_s \\ = \quad & \{\ (3.2.\mathrm{g'})\ \} \\ & \mathcal{R}_s\ \tilde{\mathrm{h}}^*(\psi'\mathcal{R}_s)\triangleleft \end{aligned}$$

- The construction for $\oplus$ must be adapted. The reader may easily check the need for tupling $\mathcal{R}_s\ \tilde{\mathrm{h}}^*\ \psi''\triangleleft\ \theta\mathrm{y}$, $\mathcal{R}_s\ \tilde{\mathrm{h}}^*\ \psi''\triangleleft\ \mathit{init}\ \theta\mathrm{y}$, $\tilde{\mathrm{h}}\ \mathit{last}\ \theta\mathrm{y}$, $\mathit{first}\ \mathit{last}\ \theta\mathrm{y}$ and the modified definition of $\oplus$

$$(\mathrm{V, I, w, pt}) \oplus \mathrm{a} = \tag{3.2.j'}$$
$$\begin{cases} (\{\mathrm{h}\{\mathrm{a}\}\}, \emptyset, \mathrm{h}\{\mathrm{a}\}, \mathrm{a}) & \text{if } \psi''[\mathrm{a}] \wedge \mathrm{pt} = \omega_\alpha \\ (\emptyset, \emptyset, \mathrm{h}\{\mathrm{a}\}, \mathrm{a}) & \text{if } \neg\,\psi''[\mathrm{a}] \wedge \mathrm{pt} = \omega_\alpha \\ (\mathrm{I} \cup \{\mathrm{w} \otimes \mathrm{h}\{\mathrm{a}\}\}, \mathrm{I}, \mathrm{w} \otimes \mathrm{h}\{\mathrm{a}\}, \mathrm{pt}) & \text{if } \psi''(\mathit{last}\,\theta\mathrm{y} \mathbin{+\!\!+} [\mathrm{a}]) \wedge \mathrm{pt} \neq \omega_\alpha \wedge \mathrm{ptEa} \\ (\mathrm{V}, \mathrm{I}, \mathrm{w} \otimes \mathrm{h}\{\mathrm{a}\}, \mathrm{pt}) & \text{if } \neg\psi''(\mathit{last}\,\theta\mathrm{y} \mathbin{+\!\!+} [\mathrm{a}]) \wedge \mathrm{pt} \neq \omega_\alpha \wedge \mathrm{ptEa} \\ (\mathrm{V} \cup \{\mathrm{h}\{\mathrm{a}\}\}, \mathrm{V}, \mathrm{h}\{\mathrm{a}\}, \mathrm{a}) & \text{if } \psi''[\mathrm{a}] \wedge \mathrm{pt} \neq \omega_\alpha \wedge \neg\mathrm{ptEa} \\ (\mathrm{V}, \mathrm{V}, \mathrm{h}\{\mathrm{a}\}, \mathrm{a}) & \text{if } \neg\psi''[\mathrm{a}] \wedge \mathrm{pt} \neq \omega_\alpha \wedge \neg\mathrm{ptEa} \end{cases}$$

Remains the computation of ψ''. With the definition of ψ' as suggested above, we find $\psi" = \textit{some}\ \psi$ and a computation rule for our specific applications might be

$$t \in z \Rrightarrow \psi" z = \psi\ t$$

In particular, if $\theta y \neq [\,]$, then

$$\psi"\ \textit{last}\ \theta y = \psi\ \textit{first last}\ \theta y$$

The standard conversion for leftreduce can be instantiated and smoothened as before. Again an improvement can be made, since

$$(\psi\ pt \Rrightarrow V = I \cup \{w\}) \wedge (\neg\psi\ pt \Rrightarrow V = I)$$

is a property of the 4-tuple, if $pt \neq \omega_\alpha$. The following FMS algorithm results

```
      W := ∅; start(f)
    ; if eof(f)    →  skip
      [] ¬ eof(f) →  fetchn(f,a)
                   ; w, pt := h{a}, a
                   ; do ¬ eof(f)
                     → fetchn(f,a)                                  (3.2.1')
                       ; if pt E a    →  w := w ⊗ h{a}
                         [] ¬ pt E a →  if ψ pt → W := W ∪ {w}
                                        []¬ ψ pt → skip
                                        fi
                                      ; w, pt := h{a}, a
                         fi
                     od
                   ; if ψ pt → W := W ∪ {w} []¬ ψ pt → skip fi
      fi
```

3. *Superfluous conditions on h.* As explained in 3.1, remark 3, the scheme of (3.2.1) solves a generalization of (3.2.a)

   $$W = k^*q^*X$$

   where $k : (\prec \alpha \succ, \uplus, \prec\succ) \rightarrow (\beta, \oplus, 1_\oplus)$ is a morphism with commutative $\oplus$. Remember that $k\ \varphi\triangleleft : (\prec \alpha \succ, \uplus, \prec\succ) \rightarrow (\beta, \oplus, 1_\oplus)$ is a morphism with commutative $\oplus$, if k is.

4. For tupled and conditional versions a remark similar to remark 3 can be made and Lemma 3 follows.

<u>Lemma 3</u>

Let $X \in \{\alpha\}$ and E an equivalence relation on α with quotientmap q. Let $R \subseteq \alpha \times \alpha$ be a cqo satisfying $E = R \cap R^{\leftarrow}$ and $f \in$ Ordfile (α, R) such that f is a representation for X.[3] Let $k : (\prec \alpha \succ, \uplus, \prec\succ) \rightarrow (\beta, \oplus, 1_\oplus)$ be a morphism with commutative $\oplus$,

[3]The system ordering R in this lemma is a means to meet the representation requirement that elements of Ordfile(α, R) are E-segmentations.

and ψ a predicate on α satisfying

$$\text{u E t} \Rrightarrow \ \psi\ \text{u} = \psi\ \text{t}.$$

Then $\mathcal{S}_3$ is a correct FMS algorithm to compute $\text{k}^*\text{q}^*\psi\triangleleft$ X, where

```
S3: |[ X : {α}; E, R : α × α; f : Ordfile(α, R);
       k : (≺ α ≻, ⊎, ≺≻) → (β, ⊕, 1⊕); ψ : α → 𝔹 ;
       |[ W : {β}
         |[ w : β; pt, a : α;
           W := ∅; start(f)
           if   eof(f) →  skip
           [] ¬ eof(f) →  fetchn(f,a)
                       ; w, pt := k{a}, a
                       ; do ¬ eof(f)
                         → fetchn(f,a)
                           ; if pt E a    → w := w ⊕ k{a}
                             [] ¬ pt E a → if ψ pt → W := W ∪ {w}
                                            [] ¬ ψ pt → skip
                                            fi
                                          ; w, pt := k{a}, a
                             fi
                         od
                       ; if  ψ pt → W := W ∪ {w} []¬ ψ pt → skip fi
           fi
    ]| ]| ]|
```

Conditional and tupled versions of Lemma 3 are straightforward.

3.3 Extensions.

There are several problems which contain the former ones as a subproblem. Sometimes the solution of the subproblem may serve as a building block for the overall solution. We will treat two of them.

3.3.1 FMS algorithm to compute $W = \{\ [h.q.t, g.q.t] \mid t \in X\ \}$ (3.3.1.a)
where $g = (\oslash \ \not\rightarrow\ e)$.

Solution. Instantiate Lemma 3 with $\psi \equiv$ **true**. Add the invariant for the computation of $(\oslash \ \not\rightarrow\ e)$ Q where Q is the representation of q.pt.
Then the correctness of the following FMS algorithm is easily checked.

```
  W := ∅; start(f)
; if eof(f)   →  skip
  [] ¬ eof(f) →  fetchn(f,a)
              ; w, pt, g0 := h{a}, a, e ⊘ a
              ; do ¬ eof(f)
```

```
        → fetchn(f, a)                                          (3.3.1.b)
          ; if pt E a  →  w, g0 := w ⊗ h {a}, g ⊘ a
            [] ¬ pt E a→  W, w, pt, g0 := W ∪ {w}, h{a}, a, e ⊘ a
            fi
        od
      ; W := W ∪ { [w,g0] }
fi
```

3.3.2 FMS algorithm to compute

$$W_1 = \{ h_1.q_1.t \mid t \in X \} \wedge W_2 = \{ h_2.q_2.t \mid t \in X \} \qquad (3.3.2.a)$$

where $h_1 : (\{\alpha\}, \cup, \emptyset) \to (\beta_1, \otimes, 1_\otimes)$ and $h_2 : (\{\alpha\}, \cup, \emptyset) \to (\beta_2, \oplus, 1_\oplus)$ are morphisms, E_1 and E_2 equivalence relations on α such that $E_2 \subsetneq E_1$ and q_1 and q_2 the quotientmap of E_1 and E_2.

Solution. Assume X is an E_1 and E_2 segmentation. Instantiate Lemma 3 with $\psi \equiv$ **true** for E_1 for E_2 simultaneously. Using the fact that $pt_2\ E_1\ pt_1 \wedge \neg\ pt_1\ E_1\ t \Rrightarrow \neg\ pt_2\ E_2\ t$, we obtain

```
W1, W2 := ∅, ∅; start(f)
if eof(f)   →  skip
[] ¬ eof(f) →  fetchn(f,a)
             ; w1, pt1, w2, pt2 := h1{a}, a, h2{a}, a
             ; do ¬ eof(f)
               → fetchn(f,a)                                    (3.3.2.b)
                 ; if pt1 E1 a →
                          if pt2 E2 a   →  w2 := w2 ⊕ h2{a}
                          [] ¬pt2 E2 a →  W2, w2, pt2 := W2 ∪ {w2}, h2 {a}, a
                          fi
                        ; w1 := w1 ⊗ h1{a}
                   [] ¬ pt1 E1 a →
                          W1, w1, pt1 := W1 ∪ {w1}, h1{a}, a
                        ; W2, w2, pt2 := W2 ∪ {w2}, h2{a}, a
                   fi
               od
             ; W1, W2 := W1 ∪ {w1}, W2 ∪ {w2}
fi
```

Note that it is not difficult to adjust (3.3.2.b) to the computation of

$$W_1 = \{ [h_1.q_1.t, \{ h_2.q_2.u \mid u \in q_1.t \}] \mid t \in X \}$$

4. GENERIC FILE ALGORITHMS APPLIED TO DATABASES

Now, we shall illustrate the use of the algorithms for database queries. In section

2.1, we described a simple databasetype and introduced some database notions. Generally, databasetypes are more complex and additional notions are involved. We will continue our informal database description of section 2.1 for the general case and illustrate the several notions on the basis of our example databasetype SR defined below.

SR models the registration of students and their examinations. Two objects, ST (student) and RES (result), are distinghuised. The attribute-names for ST and RES are inventoried and fixed in the so-called *database-skeleton* g, [3]. STR gives the relation between attribute-name and corresponding attribute-value set for ST. For RES this is done in RESR. The tabletype for ST consists of subsets T of the labeled cartesian product $\prod$ STR, which satsify the table-constraint key({RNR},T). For RES the tabletype RESF is given.

To arrive at a databasetype, we have to consider the (labeled) cartesian product of tabletypes, the label set being the set of objects (in our example: $\prod$ DK). Often, not every combination represents a possible state, a so-called *database-constraint* has to be satisfied (in our example: $\forall\, t \in v(\text{RES})\ \exists\, s \in v(\text{ST}) : t(\text{RNR}) = s(\text{RNR})$). Finally, the databasetype consists of all combinations which satisfy the database-constraint (in our case: SR).

<u>Definition of SR</u>

database-skeleton g

```
g      = { (ST   ;            STUDENT
                 {RNR         registration number
                 , Y1         first year of registration
                 , ENQ        entrance qualification
                 , DEPTC      department code
                 , SEV        additional information
                 })
         , (RES  ;            RESULT
                 {RNR
                 , CI         course index
                 , EDATE      date of examination
                 , MARK
                 })
         }
```

tabletype for ST is STF

$$\text{STF} = \{\ \text{T} \in \{\prod \text{STR}\} \mid \text{key}(\{\ \text{RNR}\}, \text{T})\ \}$$

tabletype for RES is RESF

$$\text{RESF} = \{\ \text{T} \in \{\prod \text{RESR}\} \mid \text{key}(\{\ \text{RNR, CI, EDATE}\}, \text{T})\ \}$$

where

$$
\begin{array}{llll}
\text{STR} & = \{ & (\text{RNR} & ; [1..1000000]) \\
 & , & (\text{Y1} & ; [00..99]\ ) \\
 & , & (\text{ENQ} & ; \text{CHAR}(80)\ ) \\
 & , & (\text{DEPTC} & ; [1..100]\ ) \\
 & . & (\text{SEV} & ; \text{CHAR}(80)\ ) \\
 & \} & & \\
\text{RESR} & = \{ & (\text{RNR} & ; [1..1000000]\ ) \\
 & , & (\text{CI} & ; \text{CHAR}(5)\ ) \\
 & , & (\text{EDATE} & ; \text{integer}\ ) \\
 & , & (\text{MARK} & ; \text{integer}\ ) \\
 & \} & &
\end{array}
$$

and key(S,T) $\equiv (\forall\ t_1, t_2 \in \text{T} : t_1 \restriction \text{S} = t_2 \restriction \text{S} \Rightarrow t_1 = t_2)$

databasetype SR

$$\text{SR} \quad = \{\ v \in \prod \text{DK} \mid \forall \text{ t} \in v(\text{RES})\ \exists \text{ s} \in v(\text{ST}) : \text{t(RNR)} = \text{s(RNR)}\ \}$$

with DK = { (ST; STF), (RES; RESF) }.

Furthermore a tabletype AVF exists modelling averages of results per course index:

$$\text{AVF} \quad = \{\ \text{T} \in \{\prod \text{AVR}\} \mid \text{key}(\{\text{CI}\}, \text{T})\ \}$$

$$
\begin{array}{lllll}
\text{and AVR} & = \{ & (\text{CI} & ; \text{CHAR}(5)) & \\
 & , & (\text{NB} & ; [1..1000000]\ ) & \text{number of students} \\
 & , & (\text{AV} & ; \text{real}) & \text{average result} \\
 & \} & & &
\end{array}
$$

Given a database $v \in$ SR. Construct a table W : AVF containing the following information:
For each course with one or more results for A-students

- course index
- number of A-students with one or more results in this course
- average number of results for this course per A-student

Here an A-student is a student with entrance qualification 'A'.

<u>Solution</u>
Specification in set notation

$$\text{W} = \{\ \{(\text{CI},\text{w}_1), (\text{NB}, \text{w}_2), (\text{AV}, \text{w}_3)\} \mid \text{t} \in v(\text{RES}) \wedge \text{w}_0 > 0\ \}$$

where

$$\text{w}_1 = \text{t(CI)} \tag{4.a}$$

$$\text{w}_2 = \#\ \{\ \text{u(RNR)} \mid \text{u} \in v(\text{RES}) \wedge \text{u(CI)}=\text{t(CI)} \wedge \varphi(\text{u})\ \}$$

$$\text{w}_3 = \text{w}_0/\text{w}_2$$

$$w_0 = \# \{ u \in v(RES) \mid u(CI)=t(CI) \wedge \varphi(u) \}$$

with

$$\varphi(u) = \exists\, s \in v(ST) : s(RNR)=u(RNR) \wedge s(ENQ)='A'$$

Analyzing this specification, we may conclude that part of the attribute-values to be computed arise from morphism applications on equivalence classes, so we direct our solving strategy to the application of some version of Lemma 3.

Instantiate α with $\prod$ RESR.
Since RESF $\subseteq \{\prod \text{RESR}\}$ it holds that $v(RES) \in \{\prod \text{RESR}\}$. Define

$$u\ E_1\ t \quad \equiv \quad u(CI)=t(CI)$$

Clearly, E_1 is an equivalence relation on $\prod$ RESR. Let q_1 be the quotientmap of E_1 on $v(RES)$. Let R be a linear ordering on CHAR(5), then we define cqo R_1 by

$$u_1\ R_1\ u_2 \equiv u_1(CI)\ R\ u_2(CI)$$

It holds that $E_1 = R_1 \cap R_1^{\leftarrow}$, see appendix. Assume that R_1 is a system ordering.
Let $f \in$ Ordfile($\prod$ RESR, R_1) such that f is a representation for $v(RES)$.
Given an element $t \in v(RES)$, w_0 and w_1 can be defined as morphism applications on $q_1.t$, as follows.

- Consider the monoid $M_0 = (\mathbb{N}, +, 0)$. + is associative and commutative.
Define $k_0 : (\{\prod \text{RESR}\}, \cup, \emptyset) \rightarrow M_0$ by

$$k_0\ \{u\} = 1$$

Then $w_0 = k_0 \{ u \in q_1.t \mid \varphi(u) \}$

- Let $b \in$ CHAR(5). Consider the two-element set $\{1_\diamond, b\}$ with operation $\diamond$ defined by

$\diamond$	$1_\diamond$	b
$1_\diamond$	$1_\diamond$	b
b	b	b

then $\diamond$ is associative, commutative and idempotent and $M_1 = (\{1_\diamond, b\}, \diamond, 1_\diamond)$ is a monoid. Define $k_1 : (\{\prod \text{RESR}\}, \cup, \emptyset) \rightarrow M_1$ by

$$k_1\ \{u\} = u(CI)$$

Then $w_1 = k_1\ q_1.t$

The predicate $w_0 > 0$ contains a free variable t and trivially satisfies the required condition for ψ, since $u_1\ E_1\ u_2 \equiv q_1.u_1 = q_1.u_2$.

A first approximation for a partial solution for (4.a) is obtained by instantiating a tupled version for Lemma 3, with a conditional version for one of its components.

<u>first approximation</u>

|[f : Ordfile($\prod$ RESR, R_1)
  |[W : AVF

```
|[ w0 : ℕ ; w1 : CHAR(5); pt1, a : ∏ RESR;
   empty(W); start(f)
 ; if eof(f)   →  skip
   [] ¬ eof(f) →  fetchn(f,a)                                        (4.b)
                ; if  φ a → w0 := 1 []¬ φ a → w0 := 0 fi
                ; w1, pt1 := a(CI), a
                ; do ¬ eof(f)
                  → fetchn(f,a)
                    ; if pt1 E1 a →
                           if φ a → w0 := w0 + 1 []¬ φ a → skip fi
                         ; w1 := w1 ◇ a(C)
                      []¬ pt1 E1 a →
                           if w0 > 0 → store(W, {(CI,w1),-,-})
                           [] w0 ≤ 0 →  skip
                           fi
                         ; if φ a → w0 := 1 []¬ φ a → w0 := 0 fi
                         ; w1, pt1 := a(CI), a
                      fi
                  od
                ; if  w0 > 0 → store(W, {(CI,w1),-,-)})
                  []  w0 ≤ 0 → skip
                  fi
   fi
]| ]| ]|
```

Two tasks remain before a complete solution for (4.a) is obtained: the computations of φ a and of w_2. But firstly we mention that (4.b) can be improved, due to the definition of $\diamond$. The computation of w_1 can even be omitted, since $w_1 = pt_1(CI)$.

- Task 1: computation of φ a

From the definition of SR it follows that there exists an $s \in v(ST)$ such that s(RNR) = a(RNR). Since key({RNR}, v(ST)) holds we can retrieve s via the fetchn operation. So a correct implementation for φ a is

```
  fetchc(v(ST), a↾{RNR}, s)
; φ a := s(ENQ) = 'A'
```

There are two reasons why we do not plug in this computation in (4.b)

1. The code would be unnecessarily obscure. We propose the introduction of a procedure phi with input parameter r : $\prod$ RESR, denoted by $\downarrow$ r, and output parameter b : $\mathbb{B}$, denoted by $\uparrow$ b, to define by

```
proc phi = (↓ a : ∏ RESR, ↑ b : 𝔹 |
     |[ s : ∏ STR;
```

```
fetchc(v(ST), a↾{RNR}, s)
; if s(ENQ) = 'A' → b := true
[] s(ENQ) ≠ 'A' → b := false
fi ]|)
```

2. It may lead to redundant table-operations. Once the entrance qualification of a student is known, this suffices for its other results (of this course). So we decide for a clustering per student within an equivalence class. Formally, this results in the introduction of an new equivalence relation

Define equivalence relation E_2 by

$$u_1\ E_2\ u_2 \equiv u_1(CI) = u_2(CI) \wedge u_1(RNR) = u_2(RNR)$$

with quotientmap q_2. It holds that

$$u_1\ E_2\ u_2 \Rrightarrow \varphi\, u_1 = \varphi\, u_2$$

and we may define a predicate φ' such that

$$u \in q_2.t \Rrightarrow \varphi'\, q_2.t = \varphi\, u$$

Therefore we choose for an extension like 3.3.2, however now the elements of the refined relation are of temporary interest and we do not explicitly construct the variable W_2. Let $\leq$ the usual ordering on $\mathbb{N}$. Define cqo R_2 by

$$u_1\ R_2\ u_2 \equiv u_1\ R_1\ u_2 \wedge (u_1(CI) \neq u_2(CI) \Rrightarrow u_1(RNR) \leq u_2(RNR)).$$

Then $E_2 = R_2 \cap R_2^{\leftarrow}$. Assume R_2 is a system ordering and $f \in$ Ordfile($\prod$ RESR, R_2) such that f is a representation for v(RES). A refinement of (4.b) is then given by

<u>second approximation</u>

```
|[ f : Ordfile(∏ RESR, R₂)
  |[ W : AVF
    |[ w₀ : ℕ ; pt₁, pt₂, a : ∏ RESR; s : ∏ STR; b : 𝔹 ;
      proc phi = ( {definition as above } );
      empty(W); start(f)
    ; if eof(f)   → skip
      [] ¬ eof(f) → fetchn(f,a)                                  (4.c)
                  ; phi(a,b)
                  ; if  b → w₀ := 1 [] ¬ b → w₀ := 0 fi
                  ; pt₁, pt₂ := a, a
                  ; do ¬ eof(f)
                    → fetchn(f,a)
                     ; if pt₁ E₁ a →
                              if pt₂ E₂ a  →  skip
                              [] ¬ pt₂ E₂ a  →  phi(a,b); pt₂ := a
                              fi
```

```
                    ; if b → w0 := w0 + 1 []¬b → skip fi
                 []¬ pt1 E1 a →
                    if w0 > 0 →  store(W,{(CI,pt1(CI)),-,-})
                    [] w0 ≤ 0 →  skip
                    fi
                    ; phi(a,b)
                    ; if b → w0 := 1 []¬ b → w0 := 0 fi
                    ; pt1, pt2 := a,a
                 fi
              od
            ; if  w0 > 0 →  store(W,{(CI,pt1(CI)),-,-})
              []  w0 ≤ 0 →  skip
              fi
      fi
  ]| ]| ]|
```

- Task 2 : computation of w_2.

Given that f is an E_1 and an E_2 segmentation, w_2 can be defined by

$$w_2 = (\ominus \not\rightarrow 0)\ q_1.t$$

where

$$u \ominus a = \begin{cases} u & \text{if } (\varphi a \wedge pt_2E_2a) \vee \neg\varphi a \\ u+1 & \text{if } (\varphi a \wedge \neg pt_2E_2a) \end{cases}$$

Using an extension like 3.3.1 and adding some details, (4.a) is solved.

<u>Solution of (4.a)</u>

```
  |[ f : Ordfile(∏ RESR, R2)
    |[ W : AVF
      |[ w0, w2 : ℕ ; pt1, pt2, a : ∏ RESR; s : ∏ STR; b : 𝔹 ;
        proc phi = (↓a : ∏ RESR, ↑b : 𝔹 |
                |[ s : ∏ STR;
                   fetchc(v(ST), a↾{RNR}, s)
                 ; if s(ENQ) = 'A' → b := true
                   [] s(ENQ) ≠ 'A' → b := false
                   fi  ]|);
        empty(W); start(f)
      ; if eof(f)   →  skip
        [] ¬ eof(f) →  fetchn(f,a)                                     (4.d)
                     ; phi(a,b)
                     ; if  b → w0, w2 := 1, 1 [] ¬ b → w0, w2 := 0, 0 fi
                     ; pt1, pt2 := a, a
                     ; do ¬ eof(f)
                       → fetchn(f,a)
                         ; if pt1 E1 a →
```

```
                        if pt₂ E₂ a →  skip
                        []¬ pt₂ E₂ a →
                              phi(a,b)
                            ; if b → w₂ := w₂ + 1 []¬b → skip fi
                            ; pt₂ := a
                        fi
                      ; if b → w₀ := w₀ + 1 []¬b → skip fi
                    []¬ pt₁ E₁ a →
                        if w₀ > 0 →
                              store(W,{(CI,pt₁(CI)),(NB,w₂),(AV,w₀/w₂)})
                        [] w₀ ≤ 0 →  skip
                        fi
                      ; phi(a,b)
                      ; if b → w₀, w₂ := 1, 1 []¬ b → w₀, w₂ := 0, 0 fi
                      ; pt₁, pt₂ := a,a
                    fi
                od
              ; if w₀ > 0 →  store(W,{(CI,pt₁(CI)),(NB,w₂),(AV,w₀/w₂)})
                [] w₀ ≤ 0 →  skip
                fi
        fi
  ]| ]| ]|
```

5. SYSTEM-REQUIREMENTS

In our problem solution we supposed the existence of a ordering R such that $E = R \cap R^{\leftarrow}$. Here, we shall discuss some pragmatic aspects of ordering facilities in the available systems. Ordering facilities are given by syntactical constructs whose semantics require some primary facts about tabletypes:
- a tabletype is defined over a heading, i.e. a set of attribute names or table indices.
- each attribute(name) uniquely determines a domain, the attributevalue-set.
So let TT be a tabletype, then we shall denote its heading by $\mathcal{H}(TT)$ while F(A) is used for the domain of an attribute $A \in \mathcal{H}(TT)$. Let T : TT.
In the syntactical constructs to achieve relations on T, the user is only allowed to give an enumeration of $D \subseteq \mathcal{H}(TT)$. This so-called ordering list is part of the basis for some lexicographical relation S on T $|\!\upharpoonright$ D which in its turn defines (the semantics of) the relation R on T:

$$u \mathrel{R} t \;\equiv\; u{\upharpoonright}D \mathrel{S} t{\upharpoonright}D \qquad (*)$$

We continue with $D = \{A_1, A_2\}$ and orderingslist ed $= [A_1, A_2]$ in our illustrations. A complete basis for a lexicographical relation consists of a unique (labeled) cartesian product together with a relation for each of its factors. Such a unique cartesian product is implicitly derived from the ordering list

$$\prod (A_1 : F(A_1), A_2 : F(A_2))$$

The relations on the factors are taken to be standard orderings available by the system. This implies the existence of at most one such ordering per domain. Generally those standard orderings are linear ones and a user must give its preference with respect to ascending or descending traversal by adding an adjective to each element of the ordering list, e.g. ed = [asc A_1, asc A_2].
Given that standard orderings are linear, it follows that S is linear and consequently, relations R on T defined by (*) have interesting properties. However, this puts its restrictions on the possibilities for E, even so that the only relation satisfying our requirement is the discrete relation on D. For readers familiar with the relational approach, the following proof will do:
Let R_1 and R_2 be linear orderings on $F(A_1)$ and $F(A_2)$ respectively, then $S = R_1 \lhd\!\parallel R_2$. By the definition of backward image relation, it holds that

$$R = \ll^{\leftarrow} \circ S \circ \ll$$

which equals, fact 10,

$$R = S \parallel \top\!\!\top$$

Now, we calculate

$$\begin{aligned}
& E \\
\equiv\ & \\
& R \cap R^{\leftarrow} \\
\equiv\ & \\
& (S \parallel \top\!\!\top) \cap (S \parallel \top\!\!\top)^{\leftarrow} \\
=\ & \qquad \{\ ^{\leftarrow} \text{ distributes over } \parallel\ \} \\
& (S \parallel \top\!\!\top) \cap (S^{\leftarrow} \parallel \top\!\!\top) \\
=\ & \qquad \{\ \cap \text{ distributes over } \parallel\ \} \\
& (S \cap S^{\leftarrow}) \parallel \top\!\!\top \\
=\ & \qquad \{\ S \text{ is linear, fact 5 }\} \\
& I \parallel \top\!\!\top
\end{aligned}$$

□

6. CONCLUSIONS

- We constructed a program for a database query by instantiating appropriate schemes for files. Since the problems which were solved in these schemes, relate to the application of structure preserving functions, a transformational programming method could be used in the derivation of those schemes.

- The advantage of schemes is trivial: no correctness proofs have to be given, instantiation of the schemes suffices. Especially in the development of imperative programs for database queries one may profit by schemes, because deriving imperative programs for databases is almost unfeasible in the usual way.

- Since imperative programs were our goal we had to put up with extensive code for our schemes. If abstract programs would suffice as solution for our problems, then we could have terminated the derivations at the point where the definition of the directed reduction is known. In both cases, the transformation of the directed reduction to file level is the same and can be considered as a standard one.

- $\mathcal{S}_2$, the generic scheme in Lemma 2, can be viewed as a (4G system) conversion to file level, if tables are implemented by sets. Each function k satisfying the conditions of Lemma 2 can be made available as a standard function. However, if tables are implemented by bags, a function such as k_0 in section 4, is not correctly implemented by $\mathcal{S}_2$.

- An optimal use of $\mathcal{S}_3$, the generic scheme in Lemma 3, can only be made if the programmer is acquainted with the theory of relations, in particular relations on cartesian products.[4] E.g. during our derivation the need for an extra representation requirement (E segmentation) arises. Having only system orderings to satisfy such a condition on the data we had to prove that these suffice to meet the requirement. See appendix, Lemma, p.33.

Acknowledgments
I would like to thank Kees Hemerik and Jaap van der Woude for their useful comments on earlier drafts of this paper. Their suggestions relating to both accuracy and clarity considerably improved the presentation.

[4] We could not find an appropriate reference for this subject. Therefore we collected the relevant notions and facts ourselves and added this as an appendix to this note. For ease of manipulation we chose for a pointless notation.

APPENDIX.

For reader's reference we give a short overview of our approach to relations on tables and some definitions and facts from the theory of relations, [6], as used in our calculations. Although several notions have a wider scope of applicability, our main interest is in homogeneous relations.

For each tabletype TT, there exists a labeled cartesian product LCT such that

LCT is the smallest product such that $\bigcup$ TT $\subseteq$ LCT.

By fixing the ordering of the labelset, LCT is isomorphic to a cartesian product CT. Relations defined on CT induce relations on each element of TT.

Relations are mathematical objects on which, apart from the set operations $\neg$, $\cup$ and $\cap$, the following operations are defined:

unary:	reversion	denoted by $^{\leftarrow}$
binary:	composition	denoted by $\circ$

Since the definitions of those operators is common knowledge, it suffices to give priority rules with respect to their use

- unary operators bind stronger than binary ones,
- composition has a higher priority than $\cup$ or $\cap$

Some concrete relations are frequently used. Let X be a set, then

$$I_X = \{ (x, x) \mid x \in X \}$$
$$\top\!\!\top_X = X \times X$$

are relations. We omit subscripts and infer type information from the context.

Several laws hold for the relational structure, e.g.:

$$R \circ I = I \circ R = R$$
$$R \subseteq S \Rightarrow (R \circ T \subseteq S \circ T \wedge T \circ R \subseteq T \circ S)$$
$$\neg (R^{\leftarrow}) = (\neg R)^{\leftarrow}$$

$\circ$ distributes forward and backward over $\cup$

Relations may have several properties. Let R be a relation, then

<u>Definition</u> *[functional properties]*

$R \circ R^{\leftarrow} \subseteq I$	$\triangleq$	R is functional
$R^{\leftarrow} \circ R \subseteq I$	$\triangleq$	R is injective
$R^{\leftarrow} \circ R \supseteq I$	$\triangleq$	R is total
$R \circ R^{\leftarrow} \supseteq I$	$\triangleq$	R is surjective

Well-known combination: R is bijective $\triangleq R \circ R^{\leftarrow} = R^{\leftarrow} \circ R = I$

<u>Definition</u> *[ordering properties]*

$I \subseteq R$	$\triangleq$	R is reflexive
$R \circ R \subseteq R$	$\triangleq$	R is transitive
$R \subseteq R^{\leftarrow}$	$\triangleq$	R is symmetric
$R \cap R^{\leftarrow} \subseteq I$	$\triangleq$	R is antisymmetric
$\neg R^{\leftarrow} \subseteq I \cup R$	$\triangleq$	R is connective

Apart from the well-known combinations, linear ordering (reflexive, transitive, antisymmetric and connective) and equivalence relation (reflexive, transitive and symmetric), we use

R is a quasi ordering (qo)	$\triangleq$	R is reflexive and transitive
R is a cqo	$\triangleq$	R is a connective qo

A special instance of a bijection is found in

<u>Definition</u> [*enumeration*]
Let V be a set and $\# V = N$.

$$F \text{ is an enumeration of } V \triangleq F : [1..N] \rightarrow V \text{ is a bijection.}$$

<u>Fact 1.</u> If R is a qo then $R \cap R^{\leftarrow}$ is an equivalence relation.

<u>Fact 2.</u> An equivalence relation E on X defines a quotientmap $q_E : X \rightarrow X/E$ by

$$q_E(x) = \{\ y \in X \mid x \mathrel{E} y\ \}$$

q_E is total, surjective and functional. (If no confusion can occur, subscripts are omitted.)

<u>Fact 3.</u> If R is a cqo, $R \subseteq S$ and S is transitive, then S is a cqo.

<u>Fact 4.</u> If f is a function, then

$$f^{\leftarrow} \circ (R \cap S) = f^{\leftarrow} \circ R \cap f^{\leftarrow} \circ S$$
$$(R \cap S) \circ f = R \circ f \cap S \circ f$$

<u>Fact 5.</u> If R is linear, then $R^{\leftarrow}$ is and $R \cap R^{\leftarrow} = I$

The following common way of defining relations is of particular interest for sets on which no ordering structure is imposed.

<u>Definition</u> [*backward image relation*]
Let $f : X \rightarrow Y$ and $T \subseteq Y \times Y$. The backward image relation S of T under f is defined by

$$S = f^{\leftarrow} \circ T \circ f$$

<u>Definition</u> [*forward image relation*]
Let $f : X \rightarrow Y$ and $R \subseteq X \times X$. The forward image relation S of R under f is defined by

$$S = f \circ R \circ f^{\leftarrow}$$

Depending on the (functional) properties of f, the (ordering) properties of R or T are transferred to S. For our purpose we mention

Fact 6. The image of a linear ordering under f is linear, if f is a bijection.
E.g. If F is an enumeration of V then $F \circ \leq \circ F^{\leftarrow}$ is linear.

Fact 7. If f is a total, surjective function and T is linear, then S is a cqo.

Fact 8. If R is a cqo and q is the quotientmap of $R \cap R^{\leftarrow}$, then S is linear.

On cartesian products, relations are often defined via relations on the factors. Using projection functions, the well-known product and lexicographical relation can be introduced via the backward image relation. Let X, Y and Z be sets and $R \subseteq X \times X$ and $T \subseteq Y \times Y$. Then

Definition [*projection functions* $\ll$ *and* $\gg$]
$\ll\, : X \times Y \rightarrow X$ and $\gg\, : X \times Y \rightarrow Y$ are defined by

$$\ll (x,y) = x \qquad \gg (x,y) = y$$

$\ll$ and $\gg$ are total and surjective functions.

Definition [*product relation and lexicographical relation*, $\|$ *and* $\triangleleft\!\!\|$]
On $X \times Y$, the product relation $R \| T$ and the lexicographical relation $R \triangleleft\!\!\| T$ are defined by

$$R \| T = (\ll^{\leftarrow} \circ R \circ \ll) \cap (\gg^{\leftarrow} \circ T \circ \gg)$$
$$R \triangleleft\!\!\| T = (R \| \top\!\!\top) \cap ((\neg I \| \top\!\!\top) \cup (\top\!\!\top \| T))$$

$X \times (Y \times Z)$ differs from $(X \times Y) \times Z$, so there is no associativity for $\|$ and $\triangleleft\!\!\|$ in the usual way. However, the cartesian products are isomorphic and therefore we model associativity of those operators by isomorphism.

Still using implicit typing we mention some relevant laws for the product and lexicographical relations

$$\ll^{\leftarrow} \circ \top\!\!\top \circ \ll\; = \top\!\!\top = \;\gg^{\leftarrow} \circ \top\!\!\top \circ \gg$$
$$\top\!\!\top \| \top\!\!\top = \top\!\!\top$$

$\|$ and $\triangleleft\!\!\|$ distribute forward and backward over $\cap$ and $\cup$
$^{\leftarrow}$ distributes over $\|$ and $\triangleleft\!\!\|$

Fact 9. If R and T are linear, then $R \triangleleft\!\!\| T$ is.

Fact 10. $R \| \top\!\!\top = \;\ll^{\leftarrow} \circ R \circ \ll\; = R \triangleleft\!\!\| \top\!\!\top$

Definition [*monotonicity*]
Let $f : X \rightarrow Y$, $R \subseteq X \times X$ and $T \subseteq Y \times Y$.

$$f \text{ is monotonic w.r.t. } R \text{ and } T \triangleq R \subseteq f^{\leftarrow} \circ T \circ f$$

Fact 11. If f is monotonic w.r.t. R and T and g is monotonic w.r.t. T and U, then

$g \circ f$ is monotonic w.r.t. R and U.

Fact 12. If f is total, then f is monotonic w.r.t. R and its forward image.
E.g. If F is an enumeration of V, then F is monotonic w.r.t. $\leq$ and $F \circ \leq \circ F^{\leftarrow}$.

Fact 13. $\|$ and $\nparallel$ are monotonic w.r.t. $\subseteq$, in both arguments.

Definition [*F satisfies R*]
Let F be an enumeration of V and $R \subseteq V \times V$, then

$$F \text{ satisfies } R \triangleq \leq \subseteq F^{\leftarrow} \circ R \circ F$$

Fact 14. F satisfies R $\Rightarrow$ F is monotonic w.r.t. $\leq$ and R

Fact 15. If F satisfies R and $R \subseteq S$ then F satisfies S.

Definition [*E-segmentation*]
Let F be an enumeration of V and $E \subseteq V \times V$ an equivalence relation with quotientmap q. Let $\sqsubseteq$ be a linear ordering of q^*V, then

$$F \text{ is an E-segmenation of } V \triangleq q \circ F \text{ is monotonic w.r.t. } \leq \text{ and } \sqsubseteq.$$

Lemma If F satisfies R, then

$$R \text{ is a cqo} \Rightarrow F \text{ is an } R \cap R^{\leftarrow} \text{ segmentation}$$

Proof

$$\begin{array}{ll}
& F \text{ is an } R \cap R^{\leftarrow} \text{ segmentation} \\
\Leftarrow & \\
& q \circ F \text{ is monotonic w.r.t. } \leq \text{ and } q \circ R \circ q^{\leftarrow} \wedge q \circ R \circ q^{\leftarrow} \text{ is linear} \\
\Leftarrow & \quad \{ \text{ F satisfies R, facts 11, 14 } \} \\
& q \text{ is monotonic w.r.t. R and } q \circ R \circ q^{\leftarrow} \wedge q \circ R \circ q^{\leftarrow} \text{ is linear} \\
\Leftarrow & \quad \{ \text{ q is quotientmap of } R \cap R^{\leftarrow} \text{, facts 1, 2, 8, 12 } \} \\
& R \text{ is a cqo}
\end{array}$$

□

Corollary If $R \subseteq S$ and S is transitive, then

$$F \text{ satisfies cqo } R \Rightarrow F \text{ is a } S \cap S^{\leftarrow} \text{ segmentation.}$$

Proof. Immediately from facts 3, 15.

9. BIBLIOGRAPHY

[1] R.S. Bird. An introduction to the theory of lists. In M.Broy, editor, *Logic of Programming and Calculi of Discrete Design.* Springer-Verlag, 1987. NATO ASI Series, vol. F36.

[2] R.S. Bird. Lectures on constructive functional programming. In M.Broy, editor, *Constructive Methods in Computing Science*, pages 151-216. Springer-Verlag, 1989. NATO ASI Series, vol. F55.

[3] E.O. de Brock. *Foundations of semantical databases.* Prentice Hall, 1991.

[4] E.F. Codd. A relational model of data for large shared data banks. In *Communications of the ACM*, Vol.13, no.6, June 1970, pages 377-387.

[5] L. Meertens. Algorithmics - towards programming as a mathematical activity. In *Proceedings of the CWI Symposium on Mathematics and Computer Science*, pages 289-334. North-Holland, 1986.

[6] A. Tarski. On the calculus of relations. *Journal of Symbolic Logic*, 6(3):73-89, 1941.

[7] H. Zantema. Binary structures in program transformations. Technical Report RUU-CS-88-24, Dept. Comp. Sci. Utrecht University, 1988.

In this series appeared:

89/1	E.Zs.Lepoeter-Molnar	Reconstruction of a 3-D surface from its normal vectors.
89/2	R.H. Mak P.Struik	A systolic design for dynamic programming.
89/3	H.M.M. Ten Eikelder C. Hemerik	Some category theoretical properties related to a model for a polymorphic lambda-calculus.
89/4	J.Zwiers W.P. de Roever	Compositionality and modularity in process specification and design: A trace-state based approach.
89/5	Wei Chen T.Verhoeff J.T.Udding	Networks of Communicating Processes and their (De-)Composition.
89/6	T.Verhoeff	Characterizations of Delay-Insensitive Communication Protocols.
89/7	P.Struik	A systematic design of a parallel program for Dirichlet convolution.
89/8	E.H.L.Aarts A.E.Eiben K.M. van Hee	A general theory of genetic algorithms.
89/9	K.M. van Hee P.M.P. Rambags	Discrete event systems: Dynamic versus static topology.
89/10	S.Ramesh	A new efficient implementation of CSP with output guards.
89/11	S.Ramesh	Algebraic specification and implementation of infinite processes.
89/12	A.T.M.Aerts K.M. van Hee	A concise formal framework for data modeling.
89/13	A.T.M.Aerts K.M. van Hee M.W.H. Hesen	A program generator for simulated annealing problems.
89/14	H.C.Haesen	ELDA, data manipulatie taal.
89/15	J.S.C.P. van der Woude	Optimal segmentations.
89/16	A.T.M.Aerts K.M. van Hee	Towards a framework for comparing data models.
89/17	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra.

90/1	W.P.de Roever- H.Barringer- C.Courcoubetis-D.Gabbay R.Gerth-B.Jonsson-A.Pnueli M.Reed-J.Sifakis-J.Vytopil P.Wolper	Formal methods and tools for the development of distributed and real time systems, p. 17.
90/2	K.M. van Hee P.M.P. Rambags	Dynamic process creation in high-level Petri nets, pp. 19.
90/3	R. Gerth	Foundations of Compositional Program Refinement - safety properties - , p. 38.
90/4	A. Peeters	Decomposition of delay-insensitive circuits, p. 25.
90/5	J.A. Brzozowski J.C. Ebergen	On the delay-sensitivity of gate networks, p. 23.
90/6	A.J.J.M. Marcelis	Typed inference systems : a reference document, p. 17.
90/7	A.J.J.M. Marcelis	A logic for one-pass, one-attributed grammars, p. 14.
90/8	M.B. Josephs	Receptive Process Theory, p. 16.
90/9	A.T.M. Aerts P.M.E. De Bra K.M. van Hee	Combining the functional and the relational model, p. 15.
90/10	M.J. van Diepen K.M. van Hee	A formal semantics for Z and the link between Z and the relational algebra, p. 30. (Revised version of CSNotes 89/17).
90/11	P. America F.S. de Boer	A proof system for process creation, p. 84.
90/12	P.America F.S. de Boer	A proof theory for a sequential version of POOL, p. 110.
90/13	K.R. Apt F.S. de Boer E.R. Olderog	Proving termination of Parallel Programs, p. 7.
90/14	F.S. de Boer	A proof system for the language POOL, p. 70.
90/15	F.S. de Boer	Compositionality in the temporal logic of concurrent systems, p. 17.
90/16	F.S. de Boer C. Palamidessi	A fully abstract model for concurrent logic languages, p. p. 23.
90/17	F.S. de Boer C. Palamidessi	On the asynchronous nature of communication in logic languages: a fully abstract model based on sequences, p. 29.

90/18	J.Coenen E.v.d.Sluis E.v.d.Velden	Design and implementation aspects of remote procedure calls, p. 15.
90/19	M.M. de Brouwer P.A.C. Verkoulen	Two Case Studies in ExSpect, p. 24.
90/20	M.Rem	The Nature of Delay-Insensitive Computing, p.18.
90/21	K.M. van Hee P.A.C. Verkoulen	Data, Process and Behaviour Modelling in an integrated specification framework, p. 37.
91/01	D. Alstein	Dynamic Reconfiguration in Distributed Hard Real-Time Systems, p. 14.
91/02	R.P. Nederpelt H.C.M. de Swart	Implication. A survey of the different logical analyses "if...,then...", p. 26.
91/03	J.P. Katoen L.A.M. Schoenmakers	Parallel Programs for the Recognition of *P*-invariant Segments, p. 16.
91/04	E. v.d. Sluis A.F. v.d. Stappen	Performance Analysis of VLSI Programs, p. 31.
91/05	D. de Reus	An Implementation Model for GOOD, p. 18.
91/06	K.M. van Hee	SPECIFICATIEMETHODEN, een overzicht, p. 20.
91/07	E.Poll	CPO-models for second order lambda calculus with recursive types and subtyping, p.
91/08	H. Schepers	Terminology and Paradigms for Fault Tolerance, p. 25.
91/09	W.M.P.v.d.Aalst	Interval Timed Petri Nets and their analysis, p.53.
91/10	R.C.Backhouse P.J. de Bruin P. Hoogendijk G. Malcolm E. Voermans J. v.d. Woude	POLYNOMIAL RELATORS, p. 52.
91/11	R.C. Backhouse P.J. de Bruin G.Malcolm E.Voermans J. van der Woude	Relational Catamorphism, p. 31.
91/12	E. van der Sluis	A parallel local search algorithm for the travelling salesman problem, p. 12.
91/13	F. Rietman	A note on Extensionality, p. 21.
91/14	P. Lemmens	The PDB Hypermedia Package. Why and how it was built, p. 63.

91/15	A.T.M. Aerts K.M. van Hee	Eldorado: Architecture of a Functional Database Management System, p. 19.
91/16	A.J.J.M. Marcelis	An example of proving attribute grammars correct: the representation of arithmetical expressions by DAGs, p. 25.
91/17	A.T.M. Aerts P.M.E. de Bra K.M. van Hee	Transforming Functional Database Schemes to Relational Representations, p. 21.
91/18	Rik van Geldrop	Transformational Query Solving, p. 35.
91/19	Erik Poll	Somê categorical properties for a model for second order lambda calculus with subtyping, p. 21.
91/20	A.E. Eiben R.V. Schuwer	Knowledge Base Systems, a Formal Model, p. 21.
91/21	J. Coenen W.-P. de Roever J.Zwiers	Assertional Data Reification Proofs: Survey and Perspective, p. 18.
91/22	G. Wolf	Schedule Management: an Object Oriented Approach, p. 26.
91/23	K.M. van Hee L.J. Somers M. Voorhoeve	Z and high level Petri nets, p. 16.
91/24	A.T.M. Aerts D. de Reus	Formal semantics for BRM with examples, p. .
91/25	P. Zhou J. Hooman R. Kuiper	A compositional proof system for real-time systems based on explicit clock temporal logic: soundness and complete ness, p. 52.
91/26	P. de Bra G.J. Houben J. Paredaens	The GOOD based hypertext reference model, p. 12.
91/27	F. de Boer C. Palamidessi	Embedding as a tool for language comparison: On the CSP hierarchy, p. 17.
91/28	F. de Boer	A compositional proof system for dynamic proces creation, p. 24.