A MODEL FOR DATA STRUCTURES

Mila E. Majster
Institut für Informatik
Technische Universität München

INTRODUCTION

Using a computer for the solution of problems - either by programming some algorithm or by performing retrieval operations in a data bank system usually involves at each level of abstraction data items as well as structured data, i.e. collections of data items which are somehow related to each other. If we group data items together into a set and endow this set with certain operations we get the concept of the elementary data type. If we now group structured data together and want to endow this set with certain operations, we can destinguish between three kinds of operations: those which consider structured data as a whole (without referring to their structure or to their components explicitly), those which deal only with the structure between the data items ("structural operations") and those which concern a data item itself. In this paper we want to restrict ourselves to structured data and structural operations.

At this point I should like to motivate the need of a formalization of structured data and structural operations. The arguments can be summarized as follows:

(1) Structured data are used at each level of abstraction. The desription of these data varies considerably from one level to the other. Even at the same level, e.g. in high level programming languages, it is seldom uniform. The question, e.g., as to whether the operations which can be applied to the data should be part of the description has no uniform answer. Thus communication is difficult.

(2) The design and efficiency of an algorithm depend strongly on the choice of the data structure which represents the objects to be manipulated. In certain areas this dependency has been early recognized (e.g. for sorting and searching problems) and led to the development of specific structured data (e.g. different kind of

trees, binary trees, balanced trees, B-trees, etc.) These structured data have been formally described and analysed. Until now, however, there exist very few approaches to the definition of general data structures.

(3) A rigorous definition of the notion "realization" presupposes the formalization of the concept "data structure".

(4) For proofs of correctness of programs the involved data structures must be formally describable.

Our formalism is chosen such that it is independent of a specific programming language and a specific computer model.

I. FORMAL MODEL

When we speak of structured data we usually think of a composite entity consisting of "simpler" constituents which are related somehow. The nature of the relation may differ considerably in the different levels of abstraction. It may be e.g. an order relation in a set, a relation describing a family tree, a relation describing the access mechanism in a computer memory etc. In general, the relation is the appropriate mathematical concept for handling composite data. As usually more than one relation is involved in describing structured data we choose the extended directed graph for our purposes. Our claim is that there is an extended directed graph corresponding to any structured data.

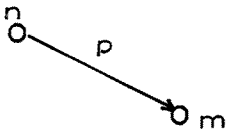
DEFINITION 1.
An extended directed graph is a pair (N,P) consisting of a nonempty set N of nodes which is at most countable and a finite set P of partial mappings from N to N.

If p is a partial mapping from N to N which is defined for some $n \in N$ then np denotes the value of p at n.

DEFINITION 2.
By $H(P)$ we denote the semigroup with identity generated by P under functional composition.

There is a simple way to represent an extended directed graph graphi-

cally. If $p \varepsilon P$ and $n \varepsilon N$ and $m = np$ we display the nodes by circles and express the fact that $m = np$ by an arrow from n to m with designation p.



We want now to give some examples of extended directed graphs.

1. Let $N = (n_{ij})_{\substack{i = 1...8 \\ j = 1...8}}$

$P = \{p_1, p_2, p_3, p_4\}$

where

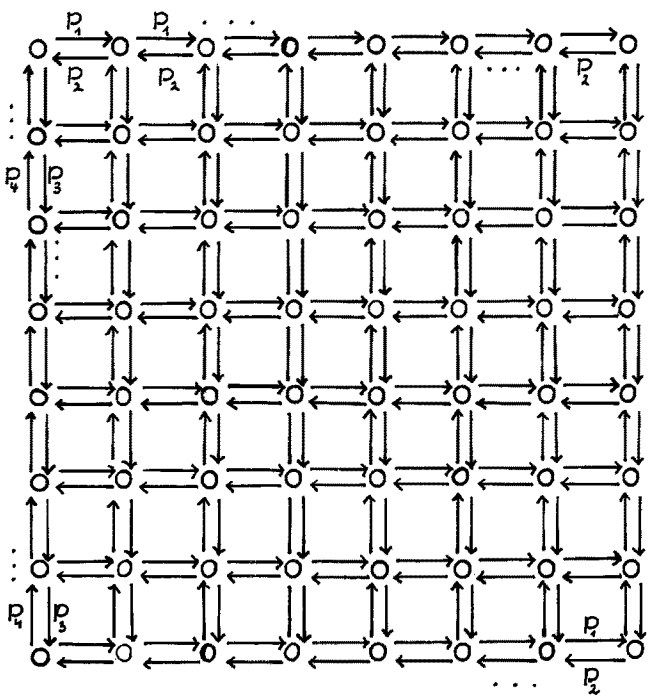
$n_{ij}\, p_1 = n_{ij+1}$ $\quad j = 1...7;\ \forall i$

$n_{ij}\, p_2 = n_{ij-1}$ $\quad j = 2...8;\ \forall i$

$n_{ij}\, p_3 = n_{i+1j}$ $\quad i = 1...7;\ \forall j$

$n_{ij}\, p_4 = n_{i-1j}$ $\quad i = 2...8;\ \forall j$

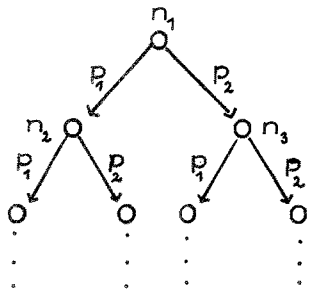
(N,P) is represented in the plane as follows



(N,P) can be thought of as one possible representation of a chessboard.

2. Let $N = (n_i)_{i \in \mathbb{N}}$ and
$P = \{p_1, p_2\}$ where
$n_i p_1 = n_{2i}$ $i = 1,2,\ldots$
$n_i p_2 = n_{2i+1}$ $i = 1,2,\ldots$

Hence

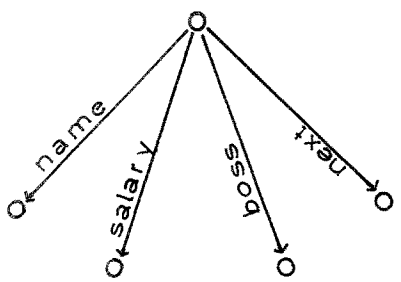


(N,P) is an infinite binary tree.

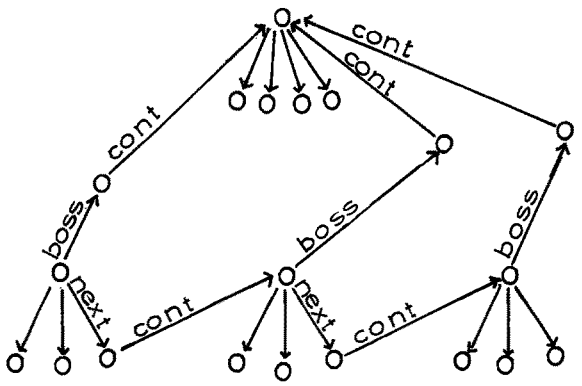
3. Consider the following mode declaration in Algol 68

```
mode employee = (string name, real salary, ref employee boss,
                 ref employee next)
```

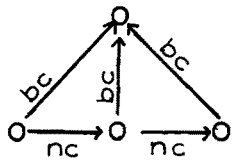
An object of this mode can be described by



We consider now three employees in alphabetic order, who have the same boss



If we are only interested in the relations between the four employee we get



Extended directed graphs can be investigated similarly as directed graphs. Typical results are: decomposition of an extended directed graph into disjoint constituents, determination of generating sets, maximal constituents and so on. For these questions the reader is referred to [1].

II. DATA STRUCTURES

We have now introduced our basic concept, the extended directed graph. We have seen how structured data can be described using these graphs. Until now, the relations between the data items - these are represented by the nodes in the graph - were supposed to be given a priori and fixed. Note that we never mentioned something like an "operation" on the data. We are now interested in the dynamic behaviour of structured data. Before going into detail in the discussion of operations let us first consider three well-known examples of data structures:

a list, a pushdown-list, a queue. With each of these data structures we have basically associated an ordered set of elements into which we can insert or from which we can delete an element according to specific rules. The distinction between these data structures stems from the different kinds of rules: in the first case, any element of the ordered set can be removed and an insertion of an element is allowed between (with respect to the given order) any two elements. In the second case both, insertion and deletion, can only affect the last element of the set. In the third case insertion affects only the last element, whereas deletion affects only the first one.

Thus, informally we can say that a data structure - as opposed to structured data - is not only determined by the relations between its data items but also by the operations which can be performed on the structured data. We are going to suggest a definition of the notion data structure based on our previous model in which the above considerations have been taken into account. In order to do so we need a series of auxiliary definitions.

DEFINITION 3.
Let $\Gamma = (N,P)$ be an extended directed graph, $\omega \notin N$. A pair (Γ,m), where $m \in N \cup \{\omega\}$, is called a <u>configuration</u> of (N,P).

DEFINITION 4.
Let G be a set of finite extended directed graphs and τ a set of configurations of elements of G. A partial mapping

$$o : \tau \longrightarrow \tau$$

is called an elementary

1) <u>access operation</u>, if
$(\Gamma,m)o = (\Gamma', m')$
implies $\Gamma = \Gamma'$ and either
i) there are no edges starting from m and $m' = \omega$
or
ii) there exists a mapping q which is defined at m, such that
$m' = mq$
In this case we say that we access m' along a path named q starting from m.

2) <u>node-insert operation</u>, if
$(\Gamma,m)o = (\Gamma',m')$

implies

i) $m' \neq \omega$

ii) card (N) = card(N')-1

iii) the subgraph of Γ' which is defined by $N' \setminus \{m'\}$ is - up to renaming - a partial graph of Γ such that $n_1 p = n_2$ $n_i \in N$ $p \in P$ implies $\overline{n}_1 \overline{p} = \overline{n}_2$, where n_i (resp. p) corresponds to $\overline{n}_i$ (resp. $\overline{p}$) according to the renaming.

iv) $m = \omega$ or $\overline{m}$ (i.e. the element corresponding to m) is connected to m' by a single edge.

In the same way we can define node-deletion, edge-insertion, and -deletion.

There are some interesting features of the above definitions. Looking e.g. at the definition of the access operation we see that we can permit or prohibit the access to some element by defining the mapping O in the right way. Moreover, our definition guarantees that a node n can be accessed from a node m only if n and m are related in the graph.

Having defined what an elementary operation should be we can now construct the data structures.

DEFINITION 5.
Let $\mathfrak{M}$ be a set of finite extended directed graphs. An extended directed graph

$$\mathfrak{K} = (\tau_{\mathfrak{M}}, T)$$

consisting of

i) a set of configurations of elements of $\mathfrak{M}$, such that for each graph $\Gamma \in \mathfrak{M}$ there is at least one configuration of Γ in $\tau_{\mathfrak{M}}$.

ii) a finite set T of elementary operations on $\tau_{\mathfrak{M}}$ is called a <u>data structure class</u>. The elements in H(T) are called <u>operations</u>.
A pair $(\Gamma, \mathfrak{K})$, where Γ is an extended directed graph and $\mathfrak{K}$ a data structure class is called a <u>data structure</u>, if there is a configuration of Γ which is a node of $\mathfrak{K}$.

EXAMPLES.
Let $\mathfrak{M}' = \{(N,P) : (N,P)$ is extended directed graph, $P = \{p\}$ and $p^t = \bigcup_{i=1}^{\infty} p^i$ is an order relation on N$\}$.

For each natural number k we choose exactly one graph Γ_k with k nodes. Then we put

$$\mathfrak{M} = \{\Gamma_k : k \in \mathbb{N}\}$$

(Here $\Gamma_k = (N_k, \{p_k\})$ with $N_k = \{n_{k1}, \ldots n_{kk}\}$

$$n_{ki} p_k = n_{ki+1} \quad i = 1 \ldots k-1)$$

Based on this set of graphs we construct now the following data structure classes.

1. Let $\mathfrak{T}^L_{\mathfrak{M}}$ be the set of all possible configurations of all elements of $\mathfrak{M}$. We choose the elementary operations as:

   $t_i : \mathfrak{T}^L_{\mathfrak{M}} \longrightarrow \mathfrak{T}^L_{\mathfrak{M}} \qquad i = 1,2,3$

   $t_1 : (\Gamma_k, n_{k1} p_k^m) \longmapsto (\Gamma_{k+1}, n_{k+1,1} p_{k+1}^m)$ for $k>o$, $k-1 \geq m \geq o$ and

   $(\Gamma_k, \omega) \longmapsto (\Gamma_{k+1}, n_{k+1,1} p_{k+1}^k)$, $(\Gamma_o, \omega) \longmapsto (\Gamma_1, n_{11})$

   t_1 is the operation "insert one element in front of the one at which you point now.

   $t_2 : (\Gamma_k, n_{k1} p_k^m) \longmapsto (\Gamma_k, n_{k1} p_k^{m+1})$ for $k>o$ $k-1 \geq m \geq o$ and

   $(\Gamma_k, n_{k1} p_k^{k-1}) \longmapsto (\Gamma_k, \omega)$

   t_2 is the operation "go to next element"

   $t_3 : (\Gamma_k, n_{k1} p_k^m) \longmapsto (\Gamma_{k-1}, n_{k-1,1} p_{k-1}^m)$ $k>o$ $k-1 \geq m \geq o$

   $(\Gamma_k, n_{k1} p_k^{k-1}) \longmapsto (\Gamma_{k-1}, \omega)$ $k>o$

   t_3 is the operation "delete the element which is pointed at".

   The extended directed graph

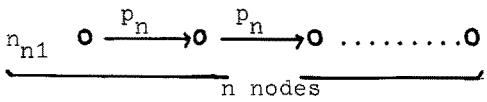
   $$\mathfrak{L} = (\mathfrak{T}^L_{\mathfrak{M}}, \{t_1, t_2, t_3\})$$

is called list class. A pair

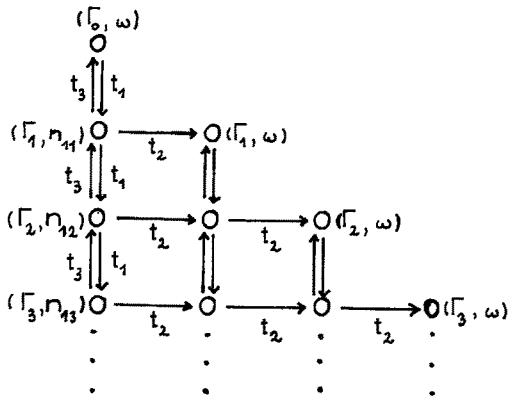
$$(\Gamma_n, \mathfrak{L})$$

is called <u>list of length n</u>.

Γ_n is graphically illustrated by



The graph $\mathcal{L}$ which describes the list class is illustrated by



$\mathcal{L} = (\tau_m^L, \{t_1, t_2, t_3\})$ is an <u>infinite</u> extended directed graph with entry. Please note that a list can only be traversed in one direction.

2. Let

$$\tau_m^K = \{(\Gamma_k, n_{k1}) \quad k \geq 1\} \cup \{(\Gamma_0, \omega)\}$$

we choose the following operations

$$k_i : \tau_m^K \longrightarrow \tau_m^K$$

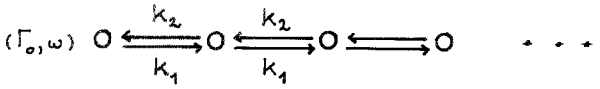
$$k_1 : (\Gamma_k, n_{k1}) \longmapsto (\Gamma_{k+1}, n_{k+1,1}) \quad k \neq 0, \quad (\Gamma_0, \omega) \longmapsto (\Gamma_1, n_{11})$$

$$k_2 : (\Gamma_{k+1}, n_{k+1,1}) \longmapsto (\Gamma_k, n_{k1}) \quad k \neq o, \quad (\Gamma_1, n_{11}) \longmapsto (\Gamma_o, \omega)$$

The extended directed graph

$$\gamma = (\tau^K_{\mathfrak{M}}, \{k_1, k_2\})$$

is a data structure class describing the cynamic behaviour of pushdown lists. The class is illustrated by



For more complicated examples we refer to [1] where a large number of interesting data structures has been described using our model of extended directed graphs. Here we restrict ourselves to point at some of the benefits of our model: The fact that a configuration of an extended directed graph can be a node in two different data structure classes enables one to construct a "connection" between these two classes. Let be

$$\mathfrak{K}_1 = (\tau^1_{\mathfrak{M}_1}, T_1)$$

$$\mathfrak{K}_2 = (\tau^2_{\mathfrak{M}_2}, T_2)$$

two data structure classes. We set

$$\mathfrak{K}_1 \cup \mathfrak{K}_2 = (\tau^1_{\mathfrak{M}_1} \cup \tau^2_{\mathfrak{M}_2}, T_1 \cup T_2)$$

If a configuration occurs in $\tau^1_{\mathfrak{M}_1}$ and in $\tau^2_{\mathfrak{M}_2}$ then in $\mathfrak{K}_1 \cup \mathfrak{K}_2$ - which is also a data structure class according to our definition - the operations of both T_1 and T_2 can be applied to that configuration. Thus, starting from some given classes new datastructure classes can be created.

Another point to be mentioned is that equivalence of data structures

as well as questions concerning the simulation of one data structure by another data structure can be treated within this model.

Moreover, those data structure classes which can be defined by specific methods, e.g. so-called definition only by operations, can be characterized with the help of our model.

III. REALIZATIONS OF DATA STRUCTURES

How can we formally define the realization of data structures? As a data structure is given by a pair

$$(\Gamma, \mathfrak{A})$$

one part of our task will be to allocate storage to Γ. The other part of our task will then be to declare how the operations have to be treated. As time is lacking, we will here concentrate on the first project.

DEFINITION 6.
Let (N,P) be an extended directed graph an A a set, $|A| \geq |N|$. A pair (r,ρ) consisting of an one-to-one mapping

$$r : N \to A$$

and an one-to-one homomorphism of semigroups

$$\rho : H(P) \to A^{(A)}$$

($A^{(A)}$ is the set of partial mappings from A to A.)

such that

i) $np \in N \Leftrightarrow (nr)(p\rho) \in Nr$

ii) $(np)r = (Nr)(p\rho)$, if p is defined at n.

REMARK.
Restricted to strongly connected extended directed graphs this definition coincides with the one given in [2].

We can interprete our definition as follows: Let A be a contiguous set of addresses, i.e. $A = \{x \in N : a \leq x \leq b\}$ for some $a,b \in \mathbb{N}$. If the elements of N are considered as external names, the mapping r can be considered as the addressbook for the structured data. It assigns a unique

address to each element of N. Pρ is both a representation for the edges of (N,P) and a mechanism for calculating the address of np from the address of n according to

$$(np)r = (nr)(p\rho).$$

When using structured data we often do not refer explicitly to an element by an external name but we refer to by its relation to some other element. Imagine for example the use of binary trees, where in most cases a node is referred to as "son of node n", n being its father node. In this case we need not keep the whole addressbook r in memory. We just keep

$$n_o r$$

where n_o is the root of the tree. The address of some element in the tree can be calculated with the help of

$$p\rho \qquad p \in P$$

IV. DISPLACING OF REALIZATIONS

As one can easily see, any extended directed graph (N,P) can be realized in any set A, $|A| \geq |N|$. The question which interests us now is: What happens if such a realization has to be shifted within our memory A? In the worst case we must set up a completely new addressbook r and redefine ρ. For a big class of extended directed graphs, however, there exists a possibility to avoid this effort of complete redefinition.

DEFINITION 7.
A node n_o of an extended directed graph is called entry, if for every $n \in N$ there exists a mapping p such that $n_o p = n$.

DEFINITION 8.
Let (N,P) be an extended directed graph with entry n_o. (N,P) is said to have a displacement mapping in A, $|A| \geq |N|$, if there exists

$$\delta^* : N \to A^A$$

with the following properties
i) δ^* is one-to-one

ii) for any $a \in A$ there exists a realization (r_a, ρ_a) such that

a) $n_o r_a = a$

$nr_a = a(n\delta^*)$

b) the mapping β_a

$\beta_a : \{t \in H(P)\rho_a : at \in Nr_a\} \to A^A$

which is defined by for all $b \in A$

$b(p\rho_a)\beta_a = (n_o p)r_b$

is well defined and one-to-one onto $N\delta^*$.

If an extended directed graph with entry n_o has such a displacement mapping δ^* we proceed in the following way: Instead of maintaining information about a specific realization which has to be updated all the time we keep δ^* in memory. Then, if one specific value $a \in A$ is given, which is supposed to be the future address of n_o we can easily calculate the position of $n \in N$ with respect to the realization (r_a, ρ_a)

$$nr_a = a(n\delta^*).$$

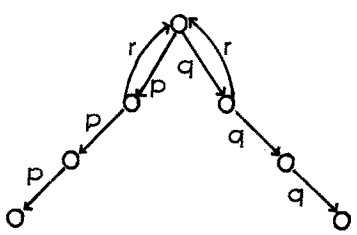
Property ii,b constructs an intrinsic correspondence between the partial transformations in $H(P)\rho_a \subset A^{(A)}$ which do not lead out of the realization r_a of (N,P) and the total transformations in $N\delta^* \subset A^A$. Via this correspondence all realizations are related in the following way: Let $a,b \in A$ and β_a, β_b the corresponding mappings. Then we get for $p \in Fun(n_o)$

$$p\rho_b = (p\rho_a)(\beta_a \beta_b^{-1})$$

and hence have built up a relation between ρ_a and ρ_b.

EXAMPLES.

1. The first three examples of section I have a displacement mapping.
2. An extended directed graph with entry which has no displacement mapping is given by



We proceed now in the investigation of extended directed graph with displacement mappings and look for a simple criterion for determining whether an extended directed graph has such a mapping.

PROPOSITION 1.
Let (N,P) be an extended directed graph with entry n_o. $|A| \geq |N|$; the following statements are equivalent

i) (N,P) has a displacement mapping $\delta^* : N \to A^A$ such that $n_o\delta^* = id_A$.

ii) for all p,q defined at n_o $n_op = n_oq$ implies $p = q$.

Proof:

i) implies ii). Let us assume that $n_op_1 = n_op_2$ and $p_1 \neq p_2$.

Then for all $b \in A$ $b(p_1\rho_a)\beta_a = (n_op_1)r_b = (n_op_2)r_b = b(p_2\rho_a)\beta_a$.

Hence we get $(p_1\rho_a)\beta_a = (p_2\rho_a)\beta_a$.

As β_a and β_a are one-to-one we get $p_1 = p_2$.

ii) implies i). For any $a \in A$ let us choose a realization (r_a,ρ_a) with $n_or_a = a$.

Let $n \in N$ be given. There is a unique $p \in H(P)$ such that $n = n_op$.

We put for all $a \in A$ $a(n\delta^*) := a(p\rho_a)$.
$\delta^* : N \to A^A$ is a displacement mapping.

Proposition 1 gives us an useful criterion to decide if an extended directed graph has a displacement mapping. Moreover, if we once found out that n_o is an entry which satisfies ii) and choose for each $a \in A$ a realization (r_a,ρ_a) with $n_or_a = a$ we can construct a displacement mapping δ^*. This is described in the following corollary.

COROLLARY 1.
Let n_o be an entry satisfying ii). Let for each $a \in A$ a realization (r_a,ρ_a) with $n_or_a = a$ be given. We define

$$a(n\delta^*) := a(p\rho_a)$$

where $n = n_op$. δ^* is a well defined mapping from N to A^A and satisfies the conditions of a displacement mapping.

In order to make the involved concepts clear to the reader we demonstrate our techniques for the case $A = \{0,1,2,\ldots M\}$.

PROPOSITION 2.
Let (N,P) be an extended directed graph with entry n_o.

$$A = \{0,1,\ldots M\}, \quad M \geqslant |N|.$$

Let $(r_o \rho_o)$ be any realization with $n_o r_o = 0$. Then there exists a displacement mapping $\delta^* : N \to A^A$ such that

$$a(n\delta^*) = a + nr_o \bmod |A|$$

(i.e. the total function

$$n\delta^* : A \to A$$

is additive).

Proof:
Let $n = n_o p$ then $nr_o = 0(p\rho_o) =: k_n \in A$. We put

$$a(n\delta^*) := a + k_n \bmod |A|.$$

Then we get

$$nr_o = k_n = 0 + k_n = 0(n\delta^*).$$

For each $a \in A$ we define now r_a, ρ_a, β_a. First we make the following observations.

1. $$n\delta^* : A \to A$$

$n\delta^*$ is a total mapping and $a(n\delta^*) \in A$ for any a. Hence

$$\delta^* : N \to A^A$$

2. δ^* is one-to-one: Suppose that

$$n\delta^* = n'\delta^*,$$

then

$$nr_o = 0(n\delta^*) = 0(n'\delta^*) = n'r_o. \text{ Hence } n = n'.$$

We put now for $a \in A$

$$n_o r_a := a \qquad nr_a := a\, n\delta^* = a + k_n \bmod |A|.$$

Surely r_a is a one-to-one mapping from N to A. By defining

$$p\rho_a = r_a^{-1}\, p\, r_a$$

we complete the proof.

COROLLARY 2.
Let n_o satisfy ii) in proposition 1. We define

$$\delta_o : N \longrightarrow A^{(A)}$$

$(n_o p)\delta_o := p\rho_o$, then for the mappings δ^*, r_a, ρ_a which are given in proposition 2 the following equations hold:

$$n\, r_a = a + (0)(n\delta_o) \bmod |A|$$

$$(np)r_a = a + (n\, r_o)(p\rho_o) \bmod |A|$$

$$= a + (0)(np\delta_o) \bmod |A|$$

We can now make the following conclusions:

1. It follows from corollary 2 that we can make use of the existence of a displacement mapping δ^* without explicitly calculating it. Once we found a realization (r_o,ρ_o) of (N,P) with $n_o r_o = 0$ for which moreover
   $n\ \delta_o$
   is easily determined and $n\ \delta_o$ is a "simple" function from A to A (e.g. additive) then for each $a \varepsilon A$ we get a nice realization (r_a,ρ_a) if we use

$$n\, r_a = a + (0)(n\delta_o) \bmod |A|$$

$$(np)r_a = a + (0)(np\delta_o) \bmod |A|$$

$$= a + (0)(n\delta_o)(p\rho_o) \bmod |A|$$

2. It is clear that the above propositions can also be stated if we choose a realization (r_s,ρ_s) with $n_o r_s = s$ as a starting point instead of (r_o,ρ_o).

3. If we define for each $a \varepsilon A$
   $(n_o p)\delta_a := p\rho_a$
   (where n_o is a basic entry)
   $\delta_a : N \to A^{(A)}$
   and
   $h_a : N\delta_a \to N\delta_o$
   $h_a \,/\, p\rho_a \to p\rho_o$
   then
   $\beta_a = h_a \cdot \beta_o$.

In this section we showed that there is an interesting technique for realizing structured data such that relocations can be easily performed. This technique works for extended directed graphs with basic entry. In [1] we demonstrate how these techniques can be easily adapted to the more general case. To give you an idea how this works: we introduce auxiliary "imaginary" edges and nodes which only appear

when addresses are determined. These auxiliary elements are n o t part of the structured data.

REFERENCES

[1] Mila E. Majster: Extended directed graphs, a model for data structures and data structure classes. Ph.D. thesis, Technische Universität München, 1975.

[2] A. L. Rosenberg: Data graphs and addressing schemes. Journal of Computer and Systems Sciences 5, 193-238 (1971).

Further references on this subject:

E. F. Codd: A relational model of data for large shared data banks. CACM, 13, No. 6, 1970.

J. Earley: Towards an understanding of data structures. CACM, 14, No. 10, 1971.

C.A.R. Hoare: Notes on data structuring. APIC Studies in Data Processing No. 8: Structured Programming.

B. Liskov, S. Zilles: Programming with abstract data types. SIGPLAN Notices, Vol. 4, 1974.

W. Turskii: A model for data structures and its applications I, II. Acta Informatica, 1, 1972.