A. L. Rosenberg
J. W. Thatcher

What Is a Multilevel Array?

Abstract: In intuitive terms, a multilevel array is either a scalar or an array each of whose elements is a multilevel array. The "semantics" of multilevel arrays can be easily expressed in terms of a notion of selector, which is basically that of the Vienna Definition Language. These selectors provide both a notational device for accessing multilevel arrays and a clean mathematical definition of "multilevel array with data domain D." However, the definition so obtained lacks the recursive flavor of the intuitive definition. By means of an axiomatic characterization of multilevel arrays, the selector-based definition and the recursive definition are shown to be equivalent.

1. Introduction

As data structures increase in complexity, one finds that a combination of natural language description and graphical examples no longer suffices to describe the structures. Mathematical techniques are needed to specify the semantics of a complex data structure, e.g., how the data items of the structure are accessed, how substructures are delineated and accessed, and how the structure is transformed by the various applicable operations.

The need for techniques to specify the semantics of data structures has been recognized for several years. A number of authors have proposed general frameworks for studying data types and structures; see, for example, the interesting papers by Gotlieb [1], Mealy [2], Scott [3], and Turski [4] for four quite distinct approaches. Several authors have devised specific models for describing complex data structures; see, for instance, Earley's V-graphs [5], Turski's "natural selector" model [6], and the data objects of the Vienna Definition Language (VDL) [7, 8]. Our purpose in this paper is to study two mathematical definitions of "multilevel array." Our main result is a theorem that exposes a sense in which the two definitions are equivalent.

In intuitive terms, a multilevel array is either a scalar (i.e., a data item) or an array each of whose elements is a multilevel array [9]. Special instances of multilevel arrays are arrays, arrays of arrays, and Sitton's [10] "general arrays," which are arrays whose elements are either scalars or arrays of scalars. Ghandour and Mezei [11] describe an impressive assortment of operations on multilevel arrays (which they term *general arrays*). Because description of these operations was their main concern, they adopted an informal approach, depending on natural language descriptions and graphical examples to describe both the entities and operations of interest.

Further study of multilevel arrays—and the development in [11, 12] illustrates that these generalized arrays do merit further consideration—will require an approach that is more formal than that of Ghandour and Mezei. It is difficult to illustrate notions concerning multilevel arrays graphically because of the limitations of the media: Examples cannot have more than two dimensions, nor can the depth of nesting of the arrays of arrays of $\cdots$ of arrays be deeper than photographic resolution will permit. Indeed the dimensionality problem exists when one describes even ordinary arrays. In this simpler case, one overcomes the limitations of examples by using the well-developed mathematical formalism for arrays. For instance, say that one wishes to define an operation on arrays which reverses the roles of the axes; in two dimensions, this is transposition. Then, denoting the transformed image of the D-dimensional array A by ΔA, this operation can be defined precisely by:

For all positive integers $I1, I2, \cdots, ID$,

$$(\Delta A)[I1; I2; \cdots; ID] = A[ID; \cdots; I2; I1],$$

without recourse to the reader's dexterity in generalizing from simple examples. Our purpose in this paper is to propose a notion of multilevel array *access domain* which, in the sense of the example of defining the operation Δ, will put multilevel arrays on the same footing as ordinary arrays. The formalism proposed in Sections 2B and 3B has the following desirable properties:

1. When applied to an ordinary array, it reduces to the conventional formalism for arrays.
2. It affords one a precise definition of the term "multilevel array" analogous to the mathematical definition of "array."

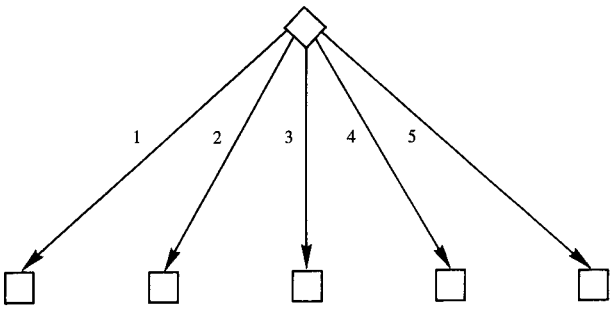

Figure 1 A P-domain tree, for the set of position names $P = \{1, 2, 3, 4, 5\}$.

3. It is based on a formalism applicable to the multilevel version of any data structure. Indeed, the presentation in Section 2 is intended to emphasize the generality of the notions presented.

The mathematical specification of a (class of) data structure(s) should not depend on the peculiarities of particular notations or representations. For this reason, a single set-theoretic definition of a class of structures is not adequate to the task, since such definitions inevitably fix representations. For example, the definition of "multilevel array" that emerges from array access domains is not related in any transparent way to our intuitive recursive definition. A more definitive method of specification is called for; and the axiomatic approach affords us the necessary freedom from representation (cf. [13, 14]). In Section 2C, we characterize axiomatically the class of multilevel structures; and we show that the intuitive recursive definition (expressed in suitably mathematical terms) and the notationally attractive access-domain-based definition both define representational variants of this class.

We close the paper with a number of illustrations of the use of our formalism.

2. Domains and structures

A. Strings and access domains

Given any set S, we denote by S^* the set of all finite strings of elements of S, including the *empty string* e. Given strings $x, y \in S^*$, we denote by xy the concatenation of x and y. If $z = xy \in S^*$, then we call x a *prefix of* z; thus, e is a prefix of every string.

A set of strings L is *prefix-closed* if, whenever a string z is in L, so also are all prefixes of z; L is *prefix-free* if no string in L is a proper prefix of any other string in L. For any set L of strings and any string $x \in L$, the set $L^{(x)} = \{y | xy \in L\}$ is variously called the *left quotient of* L *by* x or the *derivative of* L *with respect to* x. If L is prefix-closed, then so also is $L^{(x)}$: $yz \in L^{(x)}$ iff $xyz \in L$; prefix-closure of L yields $xy \in L$, which in turn implies $y \in L^{(x)}$.

An *access domain* over a set S is a finite nonempty prefix-closed subset of S^*. There is an obvious interpretation of an access domain A over a finite set S as a rooted directed tree with edges labeled by elements of S (the "simple selectors" [8]): The strings in A (the "composite selectors" [8]) are the nodes of the tree; there is an edge labeled $\sigma \in S$ from each node $x \in A$ to its σ-*successor* $x\sigma \in A$. (Of course, if $x\sigma \notin A$, node x has no σ-successor.) Thus the empty string e is the root of the tree. Associated with each string x in an access domain A over S is the set $E_A(x) = \{\sigma \in S | x\sigma \in A\}$ of *extensions* of x.

We call $x \in A$ a *fiber* precisely when $E_A(x) = \phi$ (the empty set); returning to the interpretation of access domains as trees, a fiber is a string which describes a path from the root to a leaf. We let $\Phi(A)$ denote the set of fibers of A; $\Phi(A)$ is clearly prefix-free.

B. From domains to structures

Graphically, domains are trees which indicate how to access the positions of data structures and, when these data structures are multilevel, how to access the positions of the substructures at the various levels of the structure.

P-structures. Let P be a nonempty finite set; think of P as the set of position names of a data structure. The *P-domain* is the access domain $A = P \cup \{e\} \subseteq P^*$. Note that $E_A(e) = P$, and $E_A(p) = \phi$ for $p \in P$; hence, viewed as a tree, A has a root with $\#P$ successors, all of which are fibers. Figure 1 depicts the tree corresponding to the $\{1, 2, 3, 4, 5\}$-domain. In this and subsequent portrayals of trees, the root node is diamond shaped, leaf nodes are squares, and interior nodes are circles. A *P-structure* with data set D is a total function from P to D. We denote by $[P \rightarrow D]$ the set of such functions.

Let $\mathscr{P}$ be a set of nonempty sets; think of $\mathscr{P}$ as the family of sets of position names of a kindred family of data structures, e.g., the family of all array-position names or finite-tree-position names. A *$\mathscr{P}$-structure* with data D is any P-structure with data D, with $P \in \mathscr{P}$.

Multilevel structures. A *multilevel $\mathscr{P}$-domain* is an access domain A over the set $\bigcup\mathscr{P}$ such that, for all $x \in A$, either $E_A(x) = \phi$ (so x is a fiber), or $E_A(x) \in \mathscr{P}$. Thus each nonleaf node has successors labeled by an entire set $P \in \mathscr{P}$ of position names. Figure 2 graphically depicts a multilevel $\mathscr{P}$-domain with $\mathscr{P} = \{\{1, 2, 3\}, \{\langle 1, 1\rangle, \langle 1, 2\rangle, \langle 2, 1\rangle, \langle 2, 2\rangle\}, \{0, 00, 01, 010, 011\}\}$.

Multilevel $\mathscr{P}$-domains afford one not only a graphical portrayal of the accessing structure of multilevel data structures, but also a convenient mechanism for naming the positions of the multilevel structure (so that they can be referred to directly). Specifically, the fibers $\Phi(A)$ of a multilevel $\mathscr{P}$-domain A comprise a systematic and

precise mechanism for accessing atomic positions of a multilevel $\mathscr{P}$-structure. The full access domain A is an analogous mechanism for accessing substructures as well as atomic positions.

(2.1) Let D be an arbitrary nonempty set (the *data domain*). A *multilevel $\mathscr{P}$-structure* with data D is a total function $\sigma:\Phi(A) \to D$, where A is a multilevel $\mathscr{P}$-domain.

For each string $x \in A$, the *substructure* of σ at position x is the total function $\sigma_x:\Phi(A^{(x)}) \to D$ defined by $\sigma_x(y) = \sigma(xy)$ for all $y \in \Phi(A^{(x)})$.

We illustrate this definition in Section 3 when we specialize the general development in this Section to the study of multilevel arrays.

The initial challenge for any model which specifies the semantics of a family of data structures must be the specification of the assignment operation. Indeed, definition (2.1) affords us a simple and precise way of specifying the semantics of assignments among multilevel $\mathscr{P}$-structures.

(2.2) Let σ and σ' be multilevel $\mathscr{P}$-structures with multilevel $\mathscr{P}$-domains A and A' and data D and D', respectively. Let $x \in A$ and $x' \in A'$ be arbitrary. The *assignment operation* $\sigma_x \leftarrow \sigma'_{x'}$, yields the multilevel $\mathscr{P}$-structure σ'' with $\mathscr{P}$-domain A'' and data D'' defined as follows:

(a) $D'' = D \cup D'$.

(b) $A'' = (A - \{x\}A^{(x)}) \cup \{x\}A'^{(x')}$.

(c) For all $z \in \Phi(A'')$,

$$\sigma''(z) = \begin{cases} \sigma(z) & \text{if } z \in A - \{x\}A^{(x)} \\ \sigma'_{x'}(y) & \text{if } z = xy \text{ for some } y \in A'^{(x')}. \end{cases}$$

Graphically, the selector tree for A'' is obtained by replacing the subtree of A rooted at x by the subtree of A' rooted at x'.

Vienna objects. There is a more than casual relation between our notions of domain and structure on the one hand and the data objects of VDL [7, 8, 14] on the other. Let S be the set of (simple) VDL *selectors*, and let $\mathscr{P}$ be the set of nonempty subsets of S. Let E be a set (of elementary objects). Any multilevel $\mathscr{P}$-structure $\sigma:\Phi(A) \to E$ is a Vienna object since its graph $(= \{\langle x, \sigma(x)\rangle | x \in \Phi(A)\})$ is a *characteristic set*; the strings $x \in A$ are *composite* selectors. When one introduces the special *null object* Ω of [7, 8, 14], any Vienna object becomes a multilevel $\{S\}$-structure with data $E \cup \{\Omega\}$. The *replacement* operation μ of [8] is a special case of our assignment operation (2.2); specifically, $\mu(\sigma; \langle x, \sigma'\rangle)$ yields the same result as $\sigma_x \leftarrow \sigma'$. Accessing in Vienna objects corresponds to substructure selection in (2.1): Where we write "σ_x" [15], Lucas [8] would write "$x(\sigma)$." The *identity* selector is in (2.1) automatically: $e(\sigma) = \sigma_e = \sigma$. It is clear that there are only minor extensional differences between multilevel $\mathscr{P}$-structures with data D and Vienna objects with elementary objects D. We feel, however, that our presentation of these objects has advantages over that of the papers on VDL in that it isolates the language used to access the objects defined, all the while using only well-understood mathematical notions.

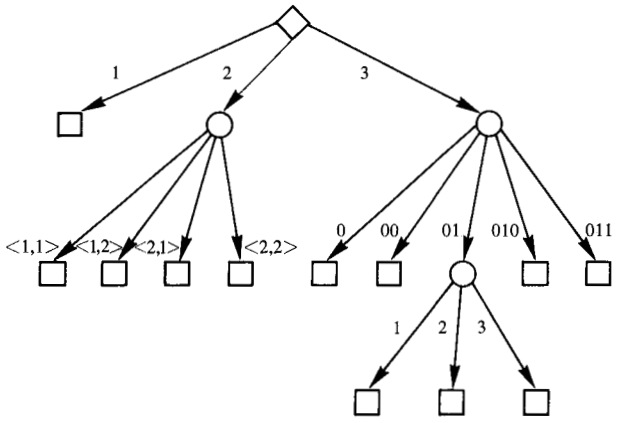

Figure 2 A multilevel $\mathscr{P}$-domain tree.

We turn our attention now to a central issue in this paper, namely, making precise the sense in which (2.1) captures faithfully the intuitive notion of multilevel data structure.

C. Abstract multilevel structures

We began in the Introduction with an intuitive definition of "multilevel array." By means of the development in Section 2B, we can render this informal notion precise, at least in terms of $\mathscr{P}$-structures: A multilevel $\mathscr{P}$-structure with data D is either an element of D or a $\mathscr{P}$-structure with data "multilevel $\mathscr{P}$-structures with data D." The awkwardness in this definition disappears as we recast it mathematically.

(2.3) The set of *functional multilevel $\mathscr{P}$-structures* with data D is the smallest set $\mathscr{F}$ satisfying

$$\mathscr{F} = D \cup \bigcup_{P \in \mathscr{P}} [P \to \mathscr{F}].$$

A minimum solution to this equation is given by: $\mathscr{F}_0 = D$; $\mathscr{F}_{i+1} = D \cup \bigcup_{P \in \mathscr{P}} [P \to \mathscr{F}_i]$; and

$\mathscr{F} = \cup_i \mathscr{F}_i$. [16].

We have inserted the qualifier "functional" in (2.3) for, while (2.3) is fully as precise as (2.1), the two definitions yield different sets of objects. Some reflection on the two definitions will suffice to establish the following cor-

respondence between the two sets of defined objects. Let $\mathscr{A}$ denote the set of multilevel $\mathscr{P}$-structures [as defined in terms of access domains in (2.1)]. Define the functions $\beta:\mathscr{F} \to \mathscr{A}$ and $\beta^{\#}:\mathscr{A} \to \mathscr{F}$ as follows.

(A) (1) For $d \in D$, $\beta(d) = \{\langle e, d\rangle\}$; [17],

(2) for $f:P \to \mathscr{F}$,

$$\beta(f) = \{\langle \pi w, d\rangle \mid \pi \in P \text{ and } \langle w, d\rangle \in \beta(f(\pi))\}.$$

(B) (1) For $d \in D$, $\beta^{\#}(\{\langle e, d\rangle\}) = d$;

(2) say $\sigma:\Phi(A) \to D$ and $A \neq \{e\}$;

then $\beta^{\#}(\sigma):E_A(e) \to \mathscr{F}$

is given by $\beta^{\#}(\sigma)(\pi) = \beta^{\#}(\sigma_{\pi})$.

Proposition. $\beta:\mathscr{F} \to \mathscr{A}$ *is* a bijection with inverse $\beta^{\#}$. (That is, β maps $\mathscr{F}$ one-to-one onto $\mathscr{A}$).

This Proposition will follow immediately from the general result of this section. But note that the statement of the Proposition is materially too weak to convince anyone that (2.1) and (2.3) are defining different representations of the same concept. In fact, the Proposition really asserts only that $\mathscr{A}$ and $\mathscr{F}$ have the same cardinality. We need a correspondence in the nature of an algebraic isomorphism, that is, a correspondence which preserves some crucial structure. But what is the crucial structure that characterizes multilevel data structures?

The basic property of multilevel $\mathscr{P}$-structures, functional or not, is that we can construct new ones and select from old ones in a systematic way. More precisely, given any set $P \in \mathscr{P}$ and a multilevel structure associated with each $\pi \in P$, we can construct a new multilevel structure which is totally characterized by this association. It is convenient and loses no generality to identify these "associations" with functions from $P \in \mathscr{P}$ into the set of multilevel structures. Thus, while we are under no compulsion to assert that a (multilevel) $\mathscr{P}$-structure *is* a function, we are saying that such structures *can be constructed* from functions. We are thus led to the following definition.

(2.4) *A system of abstract multilevel $\mathscr{P}$-structures* with data D is a pair $\langle \mathscr{S}, \mu\rangle$ where $\mathscr{S}$ is a set, and μ is a total one-to-one function [18],

$$\mu:(D \cup \bigcup_{P \in \mathscr{P}} [P \to \mathscr{S}]) \to \mathscr{S},$$

subject to the well-foundedness (or induction) axiom:

If $\mathscr{S}' \subseteq \mathscr{S}$; and if $\mu(D) \subseteq \mathscr{S}'$; and if $\mu(\sigma) \in \mathscr{S}'$ for all $\sigma:P \to \mathscr{S}'(P \in \mathscr{P})$, then $\mathscr{S}' = \mathscr{S}$.

One often associates with $\mathscr{P}$-structures not only a *constructor* μ, but also a *selector* function which selects out components of composite objects. The reader can easily convince himself that, by dint of μ's being one-to-one, such selector functions exist automatically for any abstract multilevel structure system. In fact, the existence of such selector functions is equivalent to μ's being one-to-one.

Theorem. Any two systems of abstract multilevel $\mathscr{P}$-structures with data D are isomorphic.

Proof. Let $\langle \mathscr{S}, \mu\rangle$ and $\langle \mathscr{S}', \mu'\rangle$ be two such systems. We claim that the following equations (2.5) define an isomorphism between the systems.

(2.5) Define ι by:

(1) For $d \in D$, $\iota(\mu(d)) = \mu'(d)$.

(2) For $f:P \to \mathscr{S}$, $\iota(\mu(f)) = \mu'(\iota \cdot f)$ [19].

The equations in (2.5) define a relation between $\mathscr{S}$ and $\mathscr{S}'$ which, by well-foundedness [cf. (2.4)], is total and onto.

Let $\mathscr{U}$ be the largest subset of $\mathscr{S}$ on which ι is single-valued and one-to-one. By (2.5(1)) and the fact that μ and μ' are one-to-one (2.4), $\mathscr{U}$ contains $\mu(D)$. In addition, by (2.5(2)) and the fact that μ and μ' are one-to-one, $\mathscr{U}$ must contain $\mu(f)$ for every function $f:P \to \mathscr{U}$ $(P \in \mathscr{P})$. By induction, then, $\mathscr{U} = \mathscr{S}$.

Thus, ι is an isomorphism between the two systems, which, by its very definition, preserves the "makestructure" functions.

It is clear the $\langle \mathscr{F}, 1_{\mathscr{F}}\rangle$ is a system of abstract multilevel $\mathscr{P}$-structures, where $1_{\mathscr{F}}$ is the identity function on $\mathscr{F}$. Moreover, letting $\mathscr{A}$ again denote the objects defined in (2.1), the reader can easily verify that $\langle \mathscr{A}, \beta\rangle$ is a system of abstract multilevel $\mathscr{P}$-structures, where $\beta:\mathscr{F} \to \mathscr{A}$ is as defined earlier. By the Theorem, $\langle \mathscr{F}, 1_{\mathscr{F}}\rangle$ and $\langle \mathscr{A}, \beta\rangle$ are isomorphic.

It would appear that the correspondence in our Proposition was recognized, at least intuitively, in [7, 8], but it is not easily discerned there, where the two representations are often confused. On the other hand, the need for a representation-independent (axiomatic) approach is recognized in [13, 14], but the axiom systems there fail to characterize multilevel structures for lack of induction axioms. Standish [13] corrects this shortcoming when he considers his "constructive" models; however, he does not attempt any analog of our Isomorphism Theorem, which result allows us to render precise the sense in which distinct definitions of "multilevel structure" are equivalent.

D. Research problem

In the next section, we specialize the notions developed here to the case where every P is the set of positions of an array. Before we leave the present abstract framework, we should mention an offshoot of this development which

merits further study. Our access domains assume a tree-like structure. Hence, all substructures of a multilevel structure represent distinct physical data structures, since they are accessed distinctly. (The intention of the qualifier "physical" is that, even though we may find $\sigma_x = \sigma_y$ for some substructures, this coincidence can be changed by a subsequent assignment. If the substructures σ_x and σ_y were "shared," no mere assignment statement could ever distinguish them; they would, in fact, be the same physical data structure.) Since our concern is with multilevel arrays, this restriction to tree-like selectors seems to be a natural one. However, were we to study data structures with richer interconnections, say, multilevel lists, our restriction to tree-like structuring would be hard to justify. In general, the graphical notion of selector should probably be an edge-labeled directed acyclic graph whose departures from tree-hood represent *shared substructures.* Although this graphical generalization of access domain and selector is a simple one, the corresponding linguistic (or string-oriented) generalization is far from simple. What kind of analog of access domains would maintain the expressional simplicity of these sets of strings and, yet, convey the information about shared substructures portrayed so accurately by a directed acyclic graph? This tempting question merits serious consideration. (It goes without saying that the linguistic version of *cyclic* structures, which could, for instance, model recursion in data structures, represents an even stiffer challenge.)

3. Multilevel arrays

A. Notation

Although our notation does not conform precisely with APL notation, we shall avoid conflicting with that notation so that the reader can more easily apply our notions to the operations proposed in [11].

Let N denote the positive integers. For any $d \in N$, N^d is the set of d-tuples of positive integers, an arbitrary d-tuple being denoted $\langle n_1, \cdots, n_d \rangle$. For $n \in N$, $[n]$ denotes the set $\{1, \cdots, n\}$; thus $[n_1] \times [n_2] \times \cdots \times [n_d]$, each $n_i \in N$, denotes that subset of N^d comprising all and only tuples $\langle k_1, \cdots, k_d \rangle$ for which each $k_i \in [n_i]$.

B. Arrays and multilevel arrays

Arrays. We wish to delimit a collection of sets comprising all and only "position names" of arrays. The *array schemes* of [20] yield the desired collection:

(3.1) Let d, n_1, $\cdots$, n_d be positive integers. The *d-dimensional array scheme of size* $\langle n_1, \cdots, n_d \rangle$ is the set $\mathbf{A} = [n_1] \times \cdots \times [n_d]$.
Each element of **A** is called an array *position.* The size of the array scheme **A** is denoted $\rho\mathbf{A}$.

Table 1 The multilevel array M in tabular form (cf. Figure 3).

M's array scheme	*Data entries*
⟨1,1⟩	1
⟨1,2⟩⟨1⟩⟨1⟩	2
⟨1,2⟩⟨1⟩⟨2⟩	3
⟨1,2⟩⟨2⟩	4
⟨2,1⟩	5
⟨2,2⟩	6

In accord with the prescription of Section 2, a definition of *array* follows immediately from the definition of array scheme:

(3.2) Let D be a set (of data). A *d-dimensional D-array* is a total function A:**A** $\rightarrow D$ where **A** is a d-dimensional array scheme. By extension, $\rho A = \rho\mathbf{A}$.

Multilevel arrays. Let $\mathcal{M}$ denote the collection of all array schemes; that is, $\mathcal{M} = \{[n_1] \times \cdots \times [n_d] \mid d, n_1, \cdots, n_d \in N\}$.

(3.3) A *multilevel array scheme* is a multilevel $\mathcal{M}$-domain. The *size* of the multilevel array scheme **M** is $\rho\mathbf{M} = \rho E_M(e)$. (Recall that the empty string e belongs to all access domain and that the nonempty extension of e is an array scheme.) Thus, the size of **M** is the size of its "first-level" array scheme.

A definition of multilevel array follows directly from (3.3) and (2.1).

(3.4) Let D be a set (of data). A *multilevel D-array* is a total function M: $\Phi(\mathbf{M}) \rightarrow D$, where **M** is a multilevel array scheme. By extension, $\rho M = \rho\mathbf{M}$. M is *proper* iff $\mathbf{M} \neq \{e\}$.

C. Examples

We now have formal correspondents of the notions we wished to capture. Our final task is to illustrate these formal ideas at work, both as an aid to the reader's intuition and as evidence of the descriptive and manipulative capabilities of our approach. To aid the reader in comparing our approach with an informal presentation, we draw our examples from [11].

A multilevel array with data domain $\{1, \cdots, 6\}$ and its subarrays. We present the function M in tabular form, thereby automatically specifying the underlying array scheme. M is specified in Table 1; the associated tree appears in Fig. 3, and the Ghandour-Mezei [11] specification of M in Fig. 4(a). In Figs. 4(b, c, d) appear, respectively, Ghandour and Mezei's *paths* to the subarrays $M\langle 1,2\rangle$, $M\langle 1,2\rangle\langle 1\rangle$, and $M\langle 1,2\rangle\langle 1\rangle\langle 2\rangle$ [21]. The reader can easily construct these subarrays from Table 1 and/or pick out their subtrees from Fig. 3 (at the nodes labeled A, B, 3, respectively).

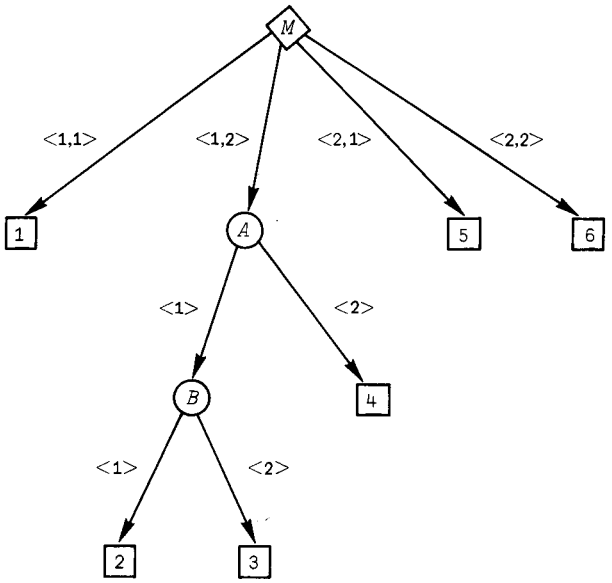

Figure 3 The tree representation of the multilevel array M.

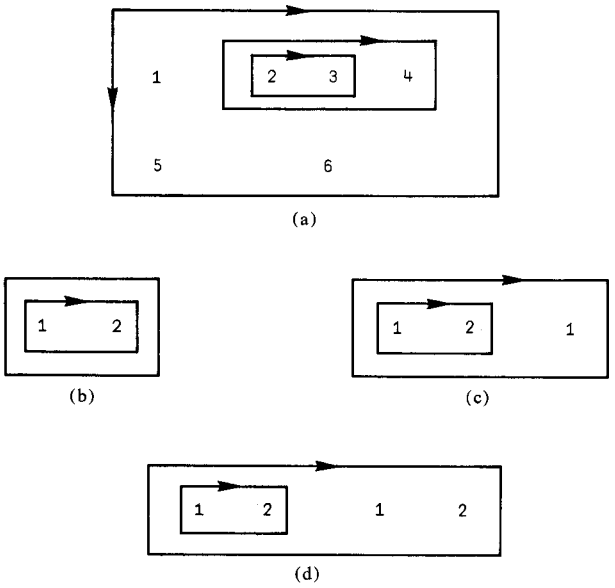

Figure 4 The Ghandour-Mezei notation for (a) the array M and (b–d) "paths" to three of M's substructures.

Predicates on arrays. Many ideas concerning logical relations among arrays follow directly from definitions (3.2) and (3.4) [or, even more basically, from (2.1)]. For instance, two multilevel arrays are *identical* iff they are equal as functions; i.e., they have the same source and target and yield the same argument-value pairs [22]. Our approach obviates the necessity for the following recursive definition from Ghandour and Mezei [23]:

"Arrays A and B are *identical* if and only if

1. either they are the same scalar,
2. or they are empty arrays of identical structure,
3. or they have identical size and have identical items at the same indices."

Similarly, in our model, one array is an *item* of another [23] iff the former array is a substructure of the latter according to (2.1).

Principal order on arrays. Ghandour and Mezei generalize the APL notion of principal order on items of an array as follows [24]:

"The components at level N+1 of A follow the components at level N of A. Given the components I and J at level N such that J follows I in principal order, the items of J follow the items of I with each collection of items of principal order."

In our framework the principal order of array items can be defined as follows: Let $M{:}\Phi(\mathbf{M}) \rightarrow D$ be a multilevel array.

Let x and y be strings in **M**. The substructure (or "item") Mx of M precedes the substructure My *in principal order* precisely when either (1) the string x is shorter than the string y, or (2) x and y are of the same length, and x precedes y lexicographically. As before, established notions suffice to describe array-oriented concepts if an appropriate framework is adopted.

Multilevel $\mathscr{P}$-domains can always be ordered lexicographically whenever all $P \in \mathscr{P}$ are ordered (as are array schemes, which are themselves ordered lexicographically). Thus, our notion of multilevel array scheme suggests a generalization of the APL principal ordering, which is an alternative to that of Ghandour and Mezei. Namely, let lexicographic order on multilevel domains be the generalized version of principal order on simple structures.

The function catenate. As our final example, we define, within our formulation, the simple operation *catenate* which "hangs" one vector on another. In the previous two examples, our point was to illustrate how our model facilitated drawing on established ideas in talking about arrays. Here we indicate how, even for simple operations, a formal definition promotes clarity and automatically exposes the accessing structure of the resulting array. We quote from Ghandour and Mezei [25]:

"The function *catenate*, denoted by ,, has arguments which are scalars/vectors whose items are general arrays. Consider $Z \leftarrow A,B$. Z is a vector whose items are the items of A followed by the items of B."

In our terminology, the operation's effects would be defined as follows:

$$\rho(A,B) = (\rho A) + (\rho B); \text{ for all } I \in [\rho(A,B)],$$

$$(A,B)\langle I\rangle = \begin{cases} A\langle I\rangle & \text{if } I \in [\rho A] \\ B\langle I - \rho A\rangle & \text{if } I \notin [\rho A]. \end{cases}$$

If **A** and **B** are the (multilevel) array schemes of A and B, respectively, then the array scheme of (A,B) is the union of **A** and **C**, where **C** is the set obtained by uniformly adding ρA to the first "index" of every string in **B**,

$$\mathbf{C} = \bigcup_{I \in [\rho B]} \{\langle I + \rho A\rangle\}\mathbf{B}^{(\langle I\rangle)}.$$

4. Conclusion

The benefits of a mathematical study of multilevel arrays are in large part the benefits of such study of any formalizable entity or phenomenon. The very act of constructing a mathematical model forces one to examine the notions to be modeled with an eye toward logical consistency, lack of ambiguity, and simplicity of structure. Once a model has been developed, it can often yield conciseness of description not available at an informal level, and precision and clarity that are wellnigh impossible using natural language, even supplemented with examples. Finally, and perhaps most important, are the insights that the right model can give. To illustrate this point in the context of the present study, we need only note that our view of multilevel arrays permits us to draw on the notions in [5, 7, 8, 14], none of which had an obvious relationship to arrays [26]; moreover, once the connections with these papers were crystallized, our model showed us that the study of multilevel arrays was actually just a narrow aspect of the study of multilevel data structures.

Acknowledgment

We are grateful to the referees for their many incisive suggestions and criticisms.

References and notes

1. C. C. Gotlieb, "Data types and structures: a synthetic approach," Univ. Toronto Technical Report 61, February, 1974.
2. G. H. Mealy, "Another look at data," *Proc. FJCC 67*, **31,** AFIPS Press, Montvale, N.J., pp. 525-534.
3. D. Scott, "Outline of a mathematical theory of computation," Oxford University Computing Laboratory Technical Monograph PRG-2, November, 1970.
4. W. M. Turski, "Data structures and their ordering," *IAG Journal* **3,** 141 (1970).
5. J. Earley, "Toward an understanding of data structures," *Comm. ACM* **14,** 617 (1971).
6. W. M. Turski, "A model for data structures and its applications," Part I: *Acta Informatica* **1,** 26 (1971); Part II: *Acta Informatica* **1,** 282 (1972).
7. John A. N. Lee, *Computer Semantics*, Van Nostrand Reinhold, New York, 1972.
8. P. Lucas, "Introduction to the method used for the formal definition of PL/1," IBM Vienna Technical Report, TR 25.081, October, 1967.
9. We admit scalars as multilevel arrays to simplify the mathematical development in Section 2. The reader who balks at this convention can follow through our development using "proper" multilevel arrays as defined in definition (3.4) of Section 3.
10. G. A. Sitton, "Operations on generalized arrays with the Genie compiler," *Comm. ACM* **13,** 284 (1970).
11. Z. Ghandour and J. Mezei, "General arrays, operators and functions," *IBM J. Res. Develop.* **17,** 335 (1973).
12. T. More, Jr., "Axioms and theorems for a theory of arrays," *IBM J. Res. Develop.* **17,** 135 (1973).
13. T. A. Standish, "Data structures, an axiomatic approach," Bolt, Beranek, and Newman Automatic Programming Memo 3, August, 1973.
14. P. Wegner, "The Vienna definition language," *Computing Surveys* **4,** 5 (1972).
15. In Section 3, we adopt the notationally preferable "σx" which seems somewhat awkward in (2.1) but which conforms better to programming usage.
16. The "composite" (non-data) elements of $\mathscr{F}$ are functions; they can, therefore, be viewed as sets of ordered pairs of the form $\langle \pi, A\rangle$ where $\pi \in P \in \mathscr{P}$ is a VDL simple selector, and A is a multilevel structure. Such pairs appear to be the *named objects* of [8] (*selector-object pairs* in [14]); from this vantage point, (2.3) underlies the assertion in [8] that a Vienna object can be described uniquely by a set of named objects.
17. We are identifying a function with its graph. Recall that e denotes the empty string.
18. Our "makestructure" function μ seems to correspond to the somewhat mysterious μ_0 applied to a set of ordered pairs in the description of Vienna objects.
19. $\iota \cdot f$ is functional composition: $\iota \cdot f(\pi) = \iota(f(\pi))$.
20. A. L. Rosenberg, "Allocating storage for extendible arrays," *Journ. ACM* **21,** 652 (1974).
21. Note that this notation for M's subarrays can be easily modified to yield a device for accessing multilevel arrays, which is consistent with APL's square bracket notation for accessing ordinary arrays.
22. We feel that this notion of equality for arrays should replace More's [12] extensionality axiom (1); for here, all empty arrays (of size $\langle n_1, \cdots, n_d\rangle$ with some $n_i = 0$) are equal.
23. Ghandour and Mezei, *op. cit.*, p. 344.
24. *Ibid.*, p. 341.
25. *Ibid.*, p. 337.
26. Of course, the relationship to arrays is obvious to one who views multidimensional arrays as a special case of lists of lists of . . . lists. However, as Earley [5] correctly notes, such a view is inconsistent with the notion of array in languages such as ALGOL or APL or FORTRAN since it does not accurately expose their accessing mechanisms for arrays.

Received August 15, 1974

The authors are located at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York 10598.