TOWARDS A THEORY OF ENCODED DATA STRUCTURES AND DATA TRANSLATION

Ben Shneiderman

Stuart C. Shapiro

Computer Science Department
Indiana University
Bloomington, Indiana

TECHNICAL REPORT NO. 13

TOWARDS A THEORY OF
ENCODED DATA STRUCTURES AND DATA TRANSLATION

BEN SHNEIDERMAN
STUART C. SHAPIRO

JULY, 1974

TOWARDS A THEORY OF ENCODED DATA STRUCTURES AND DATA TRANSLATION

by Ben Shneiderman and Stuart C. Shapiro

Department of Computer Science, Indiana University

Abstract

Several models of data base systems have distinguished levels of abstraction ranging from the high level entity set model down to the low level physical device level. This paper presents a model for describing data encodings, an intermediate level which focuses on the relationship among data items as demonstrated by contiguity or by pointer connections. Multiple data encodings for a file are shown and transformation functions which describe the translation between data encodings are discussed.

Key Words: data encoding, data translation, data base systems, data description.

Numerous attempts have been made to develop a theoretical foundation for describing data base systems. Recent work has suggested a multileveled approach which clearly separates the logical aspects from the physical aspects.

The well thought-out DIAM model [1], provides a comprehensive four-level view of data base systems. The highest level, the entity set model, reflects the users view of the data and is heavily influenced by Codd's work [2] on the relational model. The next level, the string model, describes the logical access path structure and draws heavily on graph theoretic notions [3,4]. More closely related to the implementation details is the encoding model, which focuses on the internal representation and encoding of storage structures. Finally, the physical device model deals with the placement of encoded data on the physical storage media.

Earley's work [5,6] distinguishes relational level, access path level and an implementation level. He envisions programming languages at each level and the progression through stepwise refinement from abstract to concrete algorithms [7]. The lower level languages enable the user to carefully specify more implementation details, with the goal of improving efficiency.

Childs's early work [8] on set theoretic models to describe the high level logical view has been supplemented by work on extended set theory [9] to describe the implementation details.

These multilevel approaches provide useful divisions for dealing with the complexity of a sophisticated data base system. Psychologists can be employed to assist in the selection of high level models and languages while experts in the operation of physical devices can focus their attention on the machine-oriented aspects.

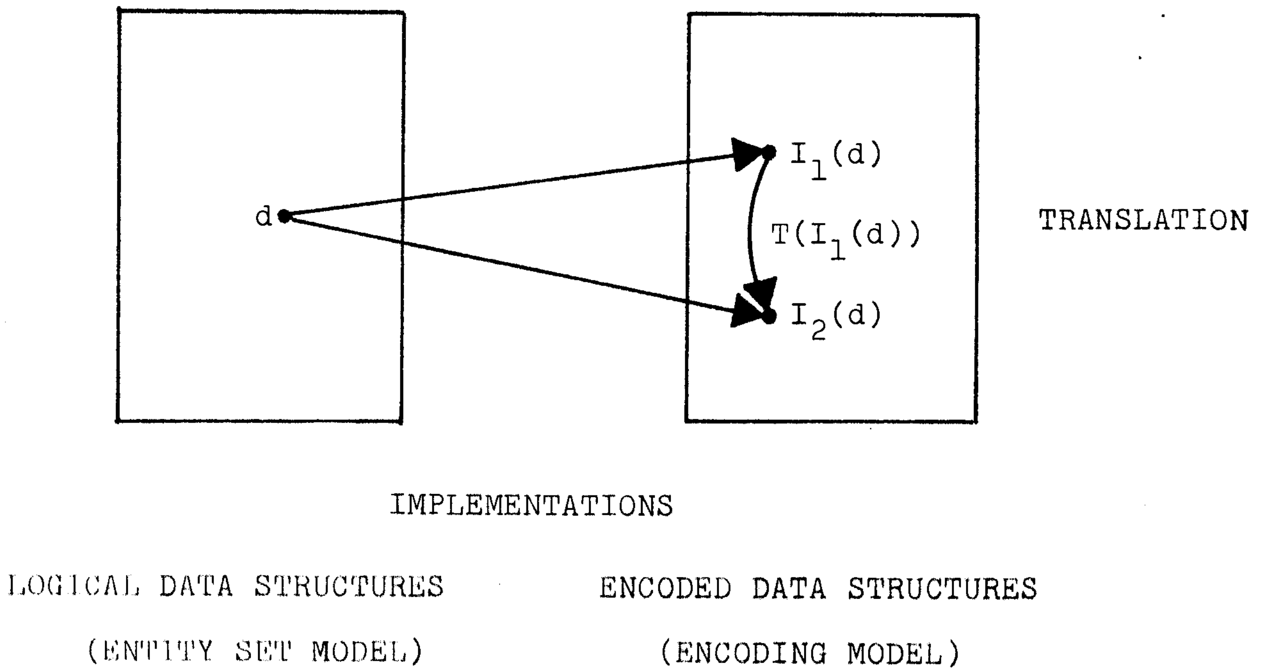
Mapping the entity set model into data encodings

This paper addresses the problem of describing the static relationships among data and provides the basis for mappings which describe the translation from one data encoding to another. The dynamics of insertion, deletion and updating are beyond the scope of this work.

We adopt the abstract perspective that the entity set model can be mapped into one or more data encodings. Each of these mappings is an implementation of the logical view of the data (see Figure 1), that is, $I: \hat{L} \rightarrow \hat{E}$, where I is the implementation, $\hat{L}$ is the space of logical data structures, and $\hat{E}$ is the space of data encoding.

To clarify this basic notion, consider a one way list. The implementation might be by a linked list strategy within the high speed storage, by a linked list stretching over several disk blocks, by contiguous allocation within a block, by contiguous allocation plus links to overflow blocks, and so on. For a more complex case, consider the logical view put forth by Codd's relational model. A number of widely varying implementations can be envisioned for this logical view.

While a number of formulations have been proposed for dealing with the logical view of data structures, there is a dearth of techniques for describing data encodings produced by a specific imple-



$$I:\hat{L} \rightarrow \hat{E}$$

Figure 1.

mentation. Although the present model does not precisely describe the machine level details, it does serve as an intermediate descriptive model.

The model

The basic components of a data encoding are blocks. A block is an addressable, contiguous segment of storage. A block is divided into elements, each of which is a field or a block. A field is a contiguous segment of storage and is the smallest meaningful unit, such as a byte or a word, of a data encoding. There are two kinds of fields: data fields and pointer fields. The contents of a data field is a data item which is an encoding of some piece of information from the entity set model. The contents of a pointer field is a pointer which addresses a block. For example, $[f_1,f_2,f_3,f_4]$ represents four fields in a single block which are contiguous in the specified sequence, and $[[f_1,f_2],[f_3,f_4]]$ represents a block consisting of two contiguous sub-blocks, each of which contains two fields.

Def. 1. We define how blocks may be constructed from fields.

Let Λ be the null block.

Let F be some set of fields.

We now define $\bar{B}_F$, which is the set of blocks using the fields in F. $\bar{B}_F$ does not include Λ.

i) if $f_1,\ldots,f_n$ for $n \geq 1$ are in F, then $[f_1,\ldots,f_n]$ is in $\bar{B}_F$. This shows how fields can be combined to form a block.

ii) if $e_1, \ldots, e_n$ for $n > 1$ are in F or in $\bar{B}_F$, then $[e_1, \ldots, e_n]$ is in $\bar{B}_F$. This shows how blocks and fields may be combined to form a block. Note that a block which contains only a block is not valid.

iii) that is all that is in $\bar{B}_F$.

Now, let $\hat{B}_F = \bar{B}_F \cup \{\Lambda\}$. $\hat{B}_F$ is the set of all blocks that can be constructed from the fields in F plus the null block.

If $b = [e_1, \ldots, e_n]$ is in $\hat{B}_F$ we call $e_i, 1 \leq i \leq n$, the <u>elements</u> of b. We will want to talk about the number of elements in a block.

Def. 2. If $b = [e_1, \ldots, e_n]$ is in $\hat{B}_F$ for some F, then $|b|_e = n$.

We will also need projection functions which select an element from a block.

Def. 3. If $b = [e_1, \ldots, e_n]$ is in $\hat{B}_F$ for some F and $1 \leq i \leq n$, then $\pi_i(b) = e_i$.

Def. 4. We will use the symbol $||_{i=1}^{n}$ for a sequence in the same way that $\sum\limits_{i=1}^{n}$ is used for a sum. Thus, $[||_{i=1}^{n} e_i]$ is the same as $[e_1, \ldots, e_n]$.

To describe a particular data encoding, E, we must describe:

1) D, the set of data fields.

2) P, the set of pointer fields (sometimes empty).

3) B, the set of blocks. B will be subset of $\hat{B}_{D \cup P}$.

4) g, a function from P into B, which describes the pointer relationships among the blocks.

<u>Examples of data encodings</u>

At this point a clarifying example to contrast four possible implementations is useful. Consider the representation of a file, a, consisting of records, b_i, where $1 \le i \le N$, which in turn consist of a student number field d_{i0} and three exam grade fields: d_{i1}, d_{i2}, and d_{i3}.

A) The first implementation shows the records to be arranged sequentially with contiguous fields within each record (see Figure 2a).

$D = \{d_{ij} \mid 1 \le i \le N, 0 \le j \le 3\}$

$P = \phi$, the empty set

$B = \{a\} \cup \{b_i \mid 1 \le i \le N\}$

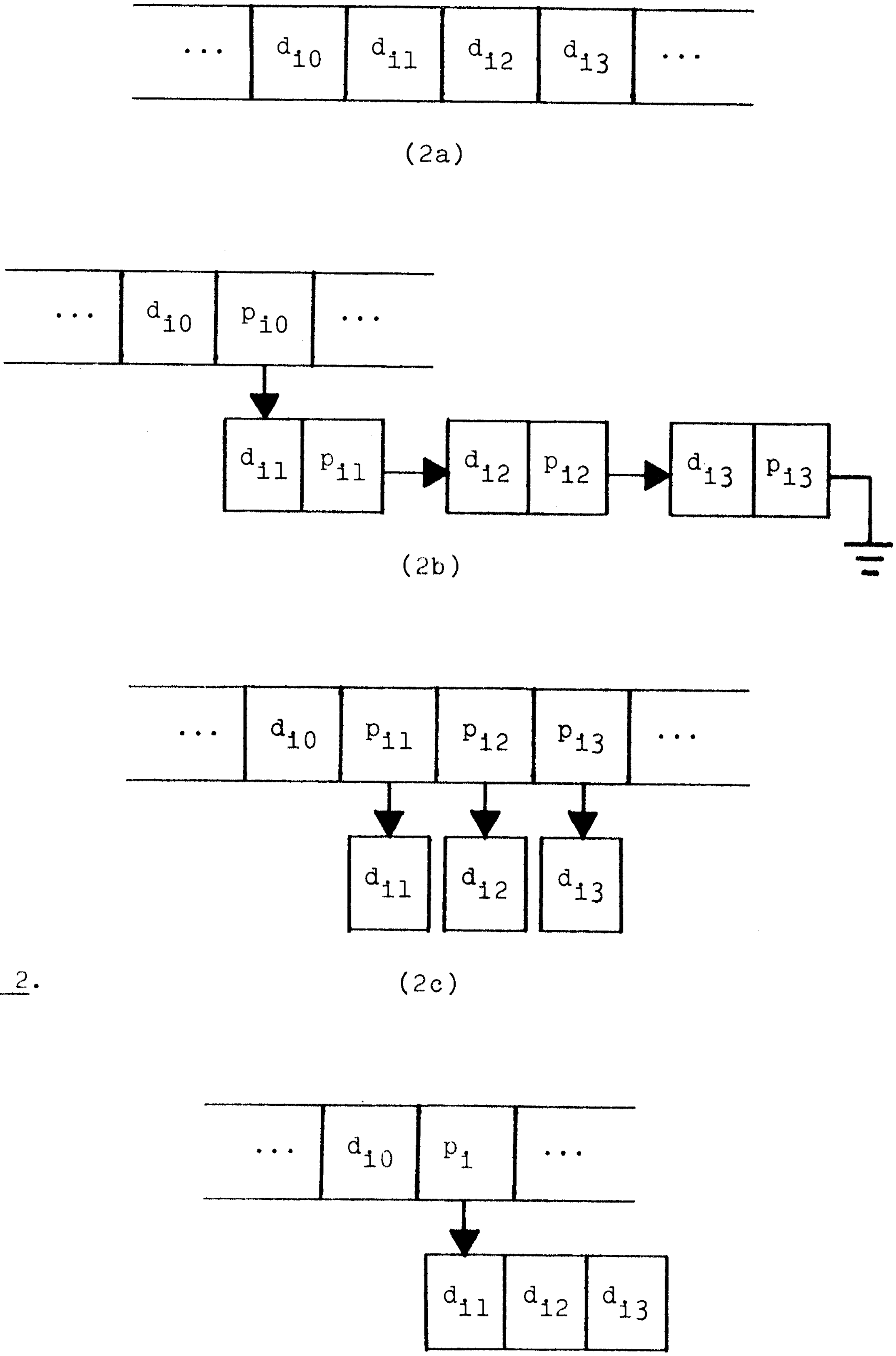
$a = [\|_{i=1}^{N} b_i]$ i.e., $|a|_\ell = N$ and $\pi_i(a) = b_i$

For each i, $1 \le i \le N$

$b_i = [d_{i0}, d_{i1}, d_{i2}, d_{i3}]$

i.e., $|b_i|_\ell = 4$ and for $D \le j \le 3$ $\pi_{j+1}(b_i) = d_{ij}$

g is empty.



(2a)

(2b)

(2c)

(2d)

Figure 2.

B) The second implementation shows each record as a one-way list. The records are sequentially arranged (see Figure 2b).

$D = \{d_{ij} | 1 \leq i \leq N, 0 \leq j \leq 3\}$

$P = \{p_{ij} | 1 \leq i \leq N, 0 \leq j \leq 3\}$

$B = \{a\} \cup \{b_{ij} | 1 \leq i \leq N, 0 \leq j \leq 3\} \cup \{\Lambda\}$

For $1 \leq i \leq N, 0 \leq j \leq 3, 0 \leq k \leq 2$

$a = [\|_{i=1}^{N} b_{i0}]$ $|a|_e = N$ $\pi_i(a) = b_{i0}$

$b_{ij} = [d_{ij}, p_{ij}]$ $|b_{ij}|_e = 2$ $\pi_1(b_{ij}) = d_{ij}$ $\pi_2(b_{ij}) = p_{ij}$

$g(p_{ik}) = b_{ik+1}$

$g(p_{i3}) = \Lambda$

C) The third implementation is by the use of the pointer array technique for the grades within each record (see Figure 2c).

$D = \{d_{ij} | 1 \leq i \leq N, 0 \leq j \leq 3\}$

$P = \{p_{ik} | 1 \leq i \leq N, 1 \leq k \leq 3\}$

$B = \{a\} \cup \{b_i | 1 \leq i \leq N\} \cup \{c_{ik} | 1 \leq i \leq N, 1 \leq k \leq 3\}$

$a = [\|_{i=1}^{N} b_i]$

For $1 \leq i \leq N, 0 \leq j \leq 3, 1 \leq k \leq 3$

$|a|_e = N$ $\pi_i(a) = b_i$

$b_i = [d_{i0}, p_{i1}, p_{i2}, p_{i3}]$

$|b_i| = 4$ $\pi_1(b_i) = d_{i0}$ $\pi_{k+1}(b_i) = p_{ik}$

$c_{ik} = [d_{ik}]$ $|c_{ik}|_e = 1$ $\pi_1(c_{ik}) = d_{ik}$

$g(p_{ik}) = c_{ik}$

D) The fourth implementation simply splits the first data field in each record from the remaining three (see Figure 2d).

$D = \{d_{ij} | 1 \le i \le N, 0 \le j \le 3\}$

$P = \{p_i | 1 \le i \le N\}$

$B = \{a\} \cup \{b_i | 1 \le i \le N\} \cup \{c_i | 1 \le i \le N\}$

For $1 \le i \le N, 1 \le k \le 3$

$a = [||_{i=1}^{N} b_i] \quad |a|_e = N \quad \pi_i(a) = b_i$

$b_i = [d_{i0}, p_i] \quad |b_i|_e = 2 \quad \pi_1(b_i) = d_{i0} \quad \pi_2(b_i) = p_i$

$c_i = [d_{i1}, d_{i2}, d_{i3}] \quad |c_i|_e = 3 \quad \pi_k(c_i) = d_{ik}$

$g(p_i) = c_i$

A final example describes the DBTG Report concept of a set implemented by chain with next and prior pointers. The set S consists of an owner record with three data fields, r, s, and t, and N member records each with two data fields, u and v (see Figure 3):

$D = \{r,s,t\} \cup \{u_i, v_i | i \le 1 \le N\}$

$P = \{n_i, p_i | 0 \le i \le N\}$

$B = \{a\} \cup \{b_i | 1 \le i \le N\}$

$$a = [r,s,t,n_0,p_0] \quad |a|_e = 5 \quad \begin{array}{ll} \pi_1(a) = r & \pi_4(a) = n_0 \\ \pi_2(a) = s & \pi_5(a) = p_0 \\ \pi_3(a) = t & \end{array}$$

For $1 \le i \le N$

$b_i = [u_i, v_i, n_i, p_i] \quad |b_i|_e = 4$

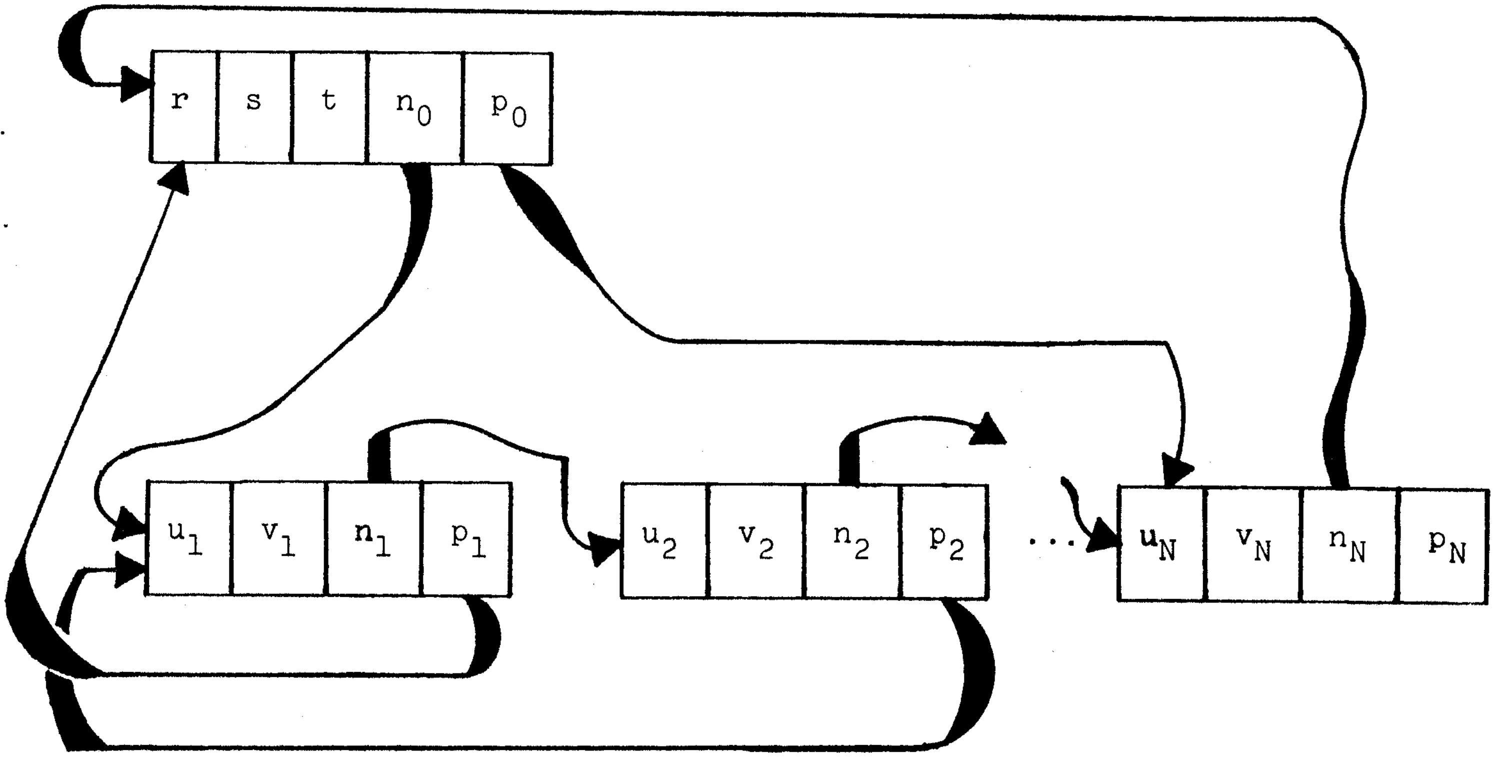


Figure 3.

$$\pi_1(b_i) = u_i \quad \pi_2(b_i) = v_i \quad \pi_3(b_i) = n_i \quad \pi_4(b_i) = p_i$$

$$g(n_i) = \begin{cases} b_{i+1} & 0 \le i < N \\ a & i = N \end{cases}$$

$$g(p_i) = \begin{cases} b_{i-1} & 1 < i \le N \\ a & i = 1 \\ b_N & i = 0 \end{cases}$$

Prescriptive model

The descriptive model presented thus far is useful as a formal tool for communication among implementers and serves as a basis for a component of the total data description task. Such data description facilities are needed by those attempting to create data base systems simulators [11,12] and to researchers in data translation [13,14,15,16].

The data translation paradigm is to develop a description of source and target data encodings and a procedural translation facility to describe the mapping. We elaborate on the data encoding model by adding transformation functions which describe translations from one data encoding to another.

Keep in mind that the prescriptive model describes the relationship between a source and a target data encoding; it is not a program for doing the translation.

In the following definitions, E is some particular data encoding.

Def. 5. If d is a piece of information from some data that E encodes, $\text{encode}_E(d)$ is the data item in E which encodes d.

Def. 6. If d is a data field in E, $\text{meaning}_E(d)$ is the meaning of the contents of d. That is, $\text{encode}_E(\text{meaning}_E(d))$ is the data item which is the contents of the data field d.

Def. 7. If d is a data field in a data encoding E' such that E' encodes the same data as E, $\text{trans}_{E'E}(d) = \text{encode}_E(\text{meaning}_{E'}(d)) =$ the translation into E of the contents of d.

Def. 8. If b is a block in E, $\text{ref}_E(b)$ is a pointer to b. If p is a pointer to b, $\text{fer}_E(p) = b$.

Def. 9. If d is a data field in E, $\text{val}_E(d)$ is the data item contained in d. If p is a pointer field in E, $\text{val}_E(p)$ is the pointer contained in p. If b is a block in E, $\text{val}_E(b) = b$. Note: if p is a pointer field in E, $g(p) = \text{fer}_E(\text{val}_E(p))$. If b is a block in E and p_1, p_2 both point to b, $\text{val}_E(p_1) = \text{val}_E(p_2) = \text{ref}_E(b)$.

Def. 10. If $e_1, \ldots, e_n$ are data items, pointers or blocks in E, then $\text{cons}_E(e_1, \ldots, e_n)$ is a block, b, in E such that $\text{val}_E(\pi_i(b)) = e_i$. That is, $b = [f_1, \ldots, f_n]$ where for $1 \le i \le n$, $\text{val}_E(f_i) = e_i$.

We will now show how data encodings A-D above are related. In what follows, we will use superscripts to show what data encoding is being used. We will show how to derive B from A, C from B, D from C, and A from D.

<u>A → B</u>

$N^B = N^A = N$

For $1 \leq i \leq N, 0 \leq j \leq 3$, $\text{meaning}_B(d^B_{ij}) = \text{meaning}_A(d^A_{ij})$

$b^B_{i4} = \Lambda$

$b^B_{ij} = \text{cons}(\text{trans}_{AB}(\pi_{j+1}(\pi_i(a^A))),\text{ref}(b^B_{ij+1}))$

$a^B = \text{cons}(||^N_{i=1} b^B_{i0})$

<u>B → C</u>

$N^C = N^B = N$

For $1 \leq i \leq N, 0 \leq j \leq 3$

$\text{meaning}_C(d^C_{ij}) = \text{meaning}_B(d^B_{ij})$

$a^C = \text{cons}(||^{N^B}_{i=1}\text{cons}(\text{trans}_{BC}(\pi_1(\pi_i(a^B))),$

$\text{ref}_C(\text{cons}_C(\text{trans}_{BC}(\pi_1(g^B(\pi_2(\pi_i(a^B)))))),$

$\text{ref}_C(\text{cons}_C(\text{trans}_{BC}(\pi_1(g^B(\pi_2(g^B(\pi_2(\pi_i(a^B)))))))),$

$\text{ref}_C(\text{cons}_C(\text{trans}_{BC}(\pi_1(g^B(\pi_2(g^B(\pi_2(g^B(\pi_2(\pi_i(a^B))))))))))))))))$

<u>C → D</u>

$N^D = N^C = N$

For $1 \leq i \leq N, 0 \leq j \leq 3$

$\text{meaning}_D(d^D_{ij}) = \text{meaning}_C(d^C_{ij})$

$a^D = \text{cons}_D(||^{N^C}_{i=1}\text{cons}(\text{trans}_{CD}(\pi_1(\pi_i(a^C))),$

$\text{ref}_D(\text{cons}_D(\text{trans}_{CD}(\pi_1(g^C(\pi_2(\pi_i(a^C))))),$

$\text{trans}_{CD}(\pi_1(g^C(\pi_3(\pi_i(a^C))))),$

$\text{trans}_{CD}(\pi_1(g^C(\pi_4(\pi_i(a^C)))))))))$

D → A

$N^A = N^D = N$

For $1 \leq i \leq N, 0 \leq j \leq 3$

$$\text{meaning}_A(d^A_{ij}) = \text{meaning}_D(d^D_{ij})$$

$$\begin{aligned} a^A = \text{cons}(||^{ND}_{i=1}\text{cons}(&\text{trans}_{DA}(\pi_1(\pi_i(a^D))), \\ &\text{trans}_{DA}(\pi_1(g^D(\pi_2(\pi_i(a^D))))), \\ &\text{trans}_{DA}(\pi_2(g^D(\pi_2(\pi_i(a^D))))), \\ &\text{trans}_{DA}(\pi_3(g^D(\pi_2(\pi_i(a^D))))))) \end{aligned}$$

Strong equivalence

Two encodings are strongly equivalent if they have the same block structure, pointer structure, and the value of each of the data fields in one encoding is equal to the value of the corresponding data fields in the other encoding. Thus two copies of a record on the same or a different disk pack are strongly equivalent.

Weak equivalences

Two encodings are weakly equivalent if they have the same block structure and pointer structure but the values of the data fields differ in value. Thus, two DBTG record occurrences of the same

DBTG record type are weakly equivalent. If the fields had identical values, the records would be strongly equivalent.

Enhancements to the descriptive model

This basic descriptive notation can be enhanced in numerous ways. To evaluate the efficiency of a particular encoding, a cost function can be associated with each pointer, $C:P \to T$ where C is the cost function, P is the set of pointers, and T is the cost, typically in units of time or money. The cost of traversing a pointer within a block is generally less than inter-block traversals. The contiguous fields within a block are assumed to be available at zero cost. A probability of request may be associated with each field to further refine the evaluative model.

The storage space required can be determined by a simple count of the number of fields. We write $|b|_f$ to indicate the number of fields in a block. $|f|_f = 1$ if f is in F and if $b = [e_1 \ldots e_n]$ then $|b|_f = \sum_{i=1}^{n} |e|_f$.

To attach more meaning to the fields, that is, to provide an interpretation for the abstract encoded data structure, a value function can be invoked. For example, to show that data fields $d_{10} \ldots d_{N0}$ are in ascending order, we write:

$$\mathrm{val}(d_{i0}) \le \mathrm{val}(d_{i+1,0}) \quad 1 \le i \le N$$

Finally, we may consider inclusion of undefined fields. An undefined field is different from a null pointer field. Undefined fields are useful in describing space in a block which has been reserved for future entries. This allows for descriptions of partially filled tables or available space lists which contain pointer fields and undefined fields. Garbage collection, compaction, and reorganization become special kinds of translations.

Conclusion

The material in this paper provides the basis for developing a model of encoded data structures. The fundamental motive has been to characterize the contiguous and pointer based relationship among fields in a storage facility. The model avoids issues related to physical devices and the details of pointer implementation, such as whether pointers indicate absolute or relative storage addresses or disk region addresses.

The model serves as a useful basis for describing part of the data translation task. The source and target data encodings can be described and then the prescriptive model can be used to show the relationship between them.

Further investigations are proceeding to describe hierarchically organized collections, implicit pointer techniques such as hash coding, and specific transformations such as the permutation of elements in a block or the replacement of a block by a pointer.

References

1. Senko, M.E.; Altman, E.B.; Astrahan, M.M.; and Fehder, P.L. Data structures and accessing in data-base systems (three parts). IBM Systems Journal 12, 1 (1973), pp. 30-93.

2. Codd, E.F. A relational model of data for large shared data banks Comm. ACM 13, 6 (1970), pp. 377-387.

3. Hsiao, D., and Harary, F. A formal system for information retrieval from files. Comm. ACM 13, 2 (1970), pp. 67-73.

4. Shneiderman, Ben, and Scheuermann, Peter. Structured data structures. To be published in CACM.

5. Earley, J. Towards an understanding of data structures. Comm. of the ACM 14, 10 (1971), pp. 617-618.

6. ————. Relational level data structures for programming languages. Acta Informatica 2 (1973), pp. 293-309.

7. Schwartz, J.T. Abstract and concrete problems in the theory of files. Data Base Systems, ed. Rustin, R., Prentice-Hall (1972), pp. 1-22.

8. Childs, D. Feasibility of a set-theoretical data structure--a general structure based on a reconstituted definition of relation proceedings. IFIP Congress, North Holland Pub. Co.

9. ————. Extended set theory: a formalism for the design implementation and operation of information systems. Unpublished manuscript.

10. CODASYL. Data Base Task Group Report (April, 1971). Available from ACM, 1133 Ave. of Americas, N.Y. 10036.

11. Scheuermann, Peter. A simulation model for data base management systems. Unpublished Doctoral Proposal, State University of N.Y. at Stony Brook (May, 1974).

12. Cardenas, A.F. Evaluation and selection of file organization--a model and a system. CACM 16, 9 (Sept., 1973).

13. Sibley, Edgar H., and Taylor, Robert W. A data definition and mapping language. Comm. of the ACM 16, 12 (December, 1973), pp. 750-759.

14. Fry, J.P.; Smith, D.P.; and Taylor, R.W. An approach to stored data definition and translation. Proc. ACM SIGFIDET Workshop on Data Description, Access and Control, November-December, 1972, pp. 77-106.

15. ————.; Frank, R.L.; and Hershey, E.A. A developmental model for data translation. Proc. ACM SIGFIDET Workshop on Data Description Access and Control, November-December, 1972, pp. 77-106.

16. Smith, D.P. A method for data translation using the stored data definition and translation task group languages. Proc. ACM SIGFIDET Workshop on Data Description, Access and Control, November-December, 1972, pp. 107-124.

17. Fry, J.P., and Merten, Alan G. A data description language approach to file translation. ACM SIGFIDET Workshop on Data Description, Access and Control, 1974.