Trenchard More, Jr.

Axioms and Theorems for a Theory of Arrays

Abstract: Array theory combines APL with set theory, transfinite arithmetic, and operationally transformed functions to produce an axiomatic theory in which the theorems hold for all arrays having any finite number of axes of arbitrary countable ordinal lengths. The items of an array are again arrays. The treatment of ordinal numbers and letters is similar to Quine's treatment of individuals in set theory. The theory is developed first as a theory of lists.

This paper relates the theory to the eight axioms of Zermelo-Fraenkel set theory, describes the structure of arrays, interprets empty arrays in terms of vector spaces, presents a system of axioms for certain properties of operations related to the APL function of reshaping, deduces a few hundred theorems and corollaries, develops an algebra for determining the behavior of operations applied to empty arrays, begins the axiomatic development of a replacement operator, and provides an informal account of unions, Cartesian products, Cartesian arrays, and outer, positional, separation, and reduction transforms.

Preamble: The intention of the work reported here is to develop a theory of arrays, in terms of standard set theory, that will provide rigorous definitions for programming operations and thereby a logical foundation for the construction of programming languages. The present contribution is a first approach to establishing a comprehensive theory of arrays, and it is anticipated that future papers will extend and refine the concepts presented.

The objects used in programming are represented in a mathematical system of ordered collections as arrays of arrays. Simple axioms define certain primitive operations on the arrays. Most data operations in programming can then be represented by operations built upon the axioms. One of the starting points for the development of the theory is APL. Programs written in APL are used to generate corollaries of the axioms and to check the consistency of the system.

Although the paper was written for the specialist in mathematical logic, it should be of interest to the computer scientist and particularly to one who specializes in programming languages. Its chief contribution is an axiomatic theory for data structures, which includes explicit rules for transforming any operation into a new operation that applies to all items in any array. H. B. M.

Contents

1. Introduction
2. Syntax
3. Extensionality
4. Elementary arrays
5. Recounting
6. Reshaping
7. Patterns
8. Extracting
9. Plans
10. Replacement
11. Positional
12. Logic
13. Sets
14. Separation
15. Powers
16. Unions
17. Choice
18. Infinity
19. Reduction
20. Arrays
21. Cartesian products
22. Motes
23. Proofs

References

Appendix I. Axioms, definitions, theorems, and corollaries

Appendix II. Theorems with proofs

1. Introduction

Programs, like mathematical proofs, should be verifiably correct. Formal methods of algebra and logic reduce the likelihood of error by deducing many results from a few, relatively simple assumptions. The literature of mathematics controls the occurrence of error by recording assumptions and results as axioms and theorems, which can be checked, revised, and used in the proof of new results. The degree to which mathematics has been able to reduce error suggests that the art of programming might profit by the same methods.

In the programming language APL\360 [1-4], programs are written along mathematical lines. The language represents data by *simple arrays*, which hold numbers and characters as items. Simple arrays are manipulated by a few, conceptually fundamental operations. An intuitive understanding of the primitive operations of APL, together with the use of several identities, enables the user of the language to predict the effect any given operation will have on any given simple array.

If the items of an array are again arrays, then it becomes more difficult to determine the effect of an operation. The nesting of arrays introduces the relation of membership. It is well known that with the exception of certain constructions in the theory of categories [5], the whole of mathematics can be based on a first-order theory in which membership is the only primitive predicate. In view of the contradictions that have occurred in the development of set theory, Quine [6] has remarked that "Intuition here *is* bankrupt". To avoid error, set theorists have had to use the axiomatic method.

The work reported here combines some of the operations of APL with set theory to produce a mathematical theory of arrays. Array theory may be spoken of as the mathematics of arrays just as set theory is the mathematics of sets or classes. The theory is developed axiomatically to avoid error and to deduce the behavior of all operations by formal calculation. The axiomatization of certain properties has been motivated by its potential applicability to the problems of programming rather than by a need to economize primitive operations. Although every effort has been made to remove contradictions, the question of consistency relative to an axiomatization of set theory has not been studied.

Lists of axioms, theorems, and proofs, such as those in Appendices I and II, serve a number of purposes. The proofs can be checked for error. A contradiction, if one occurs, can be traced back to the faulty axiom. The consequences of a change in the axioms can be quickly determined. One axiomatization can be compared with or converted to another. The axioms and theorems lay the groundwork for a metatheoretic study of the system.

The present paper, which is Part I of a longer work on arrays, develops the analogy with set theory and focuses on an axiomatization of operations related to the APL function of reshaping. Sections 3, 4, and 14 to 18, on extensionality, elementary arrays, separation, powers, unions, choice, and infinity, follow in name and order the original seven axioms of Zermelo's set theory. The eighth, the Skolem-Fraenkel axiom schema of replacement, is inserted at Section 10. The ninth, the von Neumann-Zermelo axiom of foundation, is discussed in Section 9. Part II* includes an analysis of the depth and uniformity of arrays and focuses on the axioms and theorems for unions, Cartesian products, intersections, and arithmetical and logical sums and products.

It was evident from a study of APL that useful operations would result if arbitrary monadic and dyadic operations could be applied to the items of an arbitrary array in the same way that the scalar functions of APL are applied to simple arrays. The analogy was difficult to follow because simple arrays and scalar functions have special properties not possessed by arbitrary arrays and functions.

For example, the negative - 4 3 6 of the list 4 3 6 of numbers equals the list (-4) (-3) (-6) of the corresponding negative numbers. The first negation applies to the vectors in a vector space. The second negation applies to the scalars in a field. APL identifies the two operations. The same process of identification can be applied to a few other operations, which in APL are called monadic scalar functions.

Given an arbitrary monadic operation Δ, how does one produce another operation that applies Δ to the items of an array rather than to the array itself? The answer is suggested by Heaviside's (1892) operational calculus [7,8]. Let Δ be transformed to the monadic operation $:\Delta$ by an operator, called the *replacement operator* [9a,9b]. Each item of an array can be replaced by its image under a monadic operation. $:\Delta$ applied to the triple $P\ Q\ R$ holding the arrays P, Q, and R equals the triple $(\Delta\ P)\ (\Delta\ Q)\ (\Delta\ R)$ holding the arrays $\Delta\ P$, $\Delta\ Q$, and $\Delta\ R$.

An empty component vector corresponds to the zero vector in a vector space of dimension zero. The negative of a zero vector equals the zero vector. Hence the negation of an empty list of numbers must equal the same empty list. Does the application of the replacement transform of an arbitrary monadic operation to an empty list of arbitrary arrays leave the empty list unchanged? It is easy to construct an operation Δ that maps numbers into complicated arrays. If A is an empty array, what is $:\Delta\ A$? These questions lead to Axiom 33.

If a theory is to encompass the replacement operator, then a few equations should determine $:\Delta\ A$ for every monadic operation Δ and every array A. The author assumed at first that $:\Delta\ A$ equals A for empty A and began an axiomatization in the hope of deducing a contradiction that would show what $:\Delta\ A$ should be (Section 10). The problem of degenerate reductions was also in question (Section 19). No contradiction occurred.

The theory was then developed with the aid of an interactive method for generating semigroups and proving relevant theorems. The method operates interactively in APL, generates the associative closure of a system of equations, derives from the Skolem-Herbrand theorem [10], and does not require the statement of algebraic conjectures in the lower predicate calculus [11]. The sought-for contradiction at the end of Section 10 was not found until Cartesian products were axiomatized.

The interactive method applies because the class of all monadic operations under composition is a full transformation semigroup [12] on the universe of arrays. If a monadic operation Δ sends every array to the array B, then $\Delta\ A = B$ for every A. If $\underline{\Delta}$ is any monadic opera-

*To be published at a later date.

tion, then $\underline{\Delta}\ A$ is an array and so $\Delta\ \underline{\Delta}\ A = \Delta\ A$ for every A. Thus to every array B there corresponds a unique left zero Δ in the full transformation semigroup.

Since arrays are ordered, it is possible to represent dyadic operations as monadic operations defined on pairs, which are arrays on one axis of length 2. Some of the most important dyadic operations, such as the operations for unions, Cartesian products, intersections, arithmetic sums and products, and logical sums and products, have monadic counterparts that are defined for all arrays. Dyadic operations can also be represented in the full transformation semigroup.

In the course of developing a table of compositions for more than 110 monadic operations, a few contradictions were isolated, deduced mechanically from tabular entries representing the axioms and theorems, and removed by modifying the axioms. Some of the entries in the table revealed unexpected identities that were subsequently deduced by hand.

An array can be visualized as a warehouse in which every item can be found by address. For example, the item at address 4 7 3 0 can be found in isle 4, section 7, stack 3, tier 0. Each number in an address is called an *index* and is visualized as lying on an index axis. The number of axes is called the *valence* [13-15], which may be any finite ordinal number.

Each index axis for an array has a *length*, which is the number of indices on the axis. Lengths and indices may be any countable ordinal numbers: 0, 1, 2, ..., ω, $\omega + 1$, $\omega + 2$, ... A restriction of indices to the finite ordinal numbers is a needless limitation that obscures the essential processes of counting and indexing. A file consisting of a succession of finite, open-ended records can be indexed as follows:

0 1 2 ... 1W 1W1 1W2 ... 2W 2W1 2W2 ....

The limit ordinals 0, 1W, 2W, ..., which have no direct predecessor, are interpreted as starting new records.

An array for which there are no addresses is said to be *empty*. Since no ordinal index can be less than zero, an array is empty if and only if it has at least one axis of zero length. Empty arrays, like the empty class in set theory and the imaginary numbers in complex arithmetic, permit simple equations and constructions to hold in all cases.

To each address for an array there corresponds an item. Different addresses may correspond to the same item. An array is said to have or *hold* its items, which *occur in* the array. An empty array holds no items because it has no addresses. A nonempty array holds one item at each address. Identical copies of an item are considered to occur at every address corresponding to that item.

Common experience with the location of objects by address suggests that the items of an array should again be arrays. For example, the retrieval of particular data in a city may require a nested succession of addresses of the following sort: (street avenue) (wing floor isle room) (shelf drawer) (folder) (document). Addresses are nested because items at different addresses may have incompatible systems of addresses.

Arrays are assumed to be *rectangular* in the sense that a finite sequence of ordinal numbers is a legitimate address for an array if each index in the address is less than the length of the corresponding axis. Rectangularity makes the consequences of using an address predictable and simplifies the arithmetic of converting one system of addresses to another. The nesting of arrays within arrays compensates for the restriction to rectangularity. For example, a page of words in which the lines differ in length can be represented as a list in which each item is a list of words. An array on one axis is called a *list* [16].

The essential principles of array theory are those of nesting, counting, and valence. Nesting is treated in the set theory of Zermelo (1908), Skolem (1922), and von Neumann (1929) [17]. Cantor's (1879) transfinite arithmetic shows how counting can be based on the purely structural operations of concatenating lists, putting the items of a list cyclically into longer lists, and deleting initial segments from a list [18,19]. These fundamental operations occur as the primitive functions of catenating, reshaping, and dropping in APL\360. Sections 2 through 19 combine the principles of nesting and counting in a theory of lists.

Since the arrangement of items in rows, columns, and layers tends to obscure the fundamental nature of nesting and counting, valence is not treated in detail until Sections 20 and 21. The basic ways of combining multivalent arrays stem from the tensor calculus of Ricci (1888) [20-22]. APL generalizes component tensors to rectangular arrays of scalars, enlarges the domain of scalars to include characters, generalizes the contraction of component tensors to the transposition and reduction of arrays, and extends the addition, outer multiplication, and inner multiplication of component tensors to the item-by-item, outer, and inner combination of simple arrays by arbitrary dyadic scalar functions.

Most programming languages and mathematical systems are *many-sorted* [23] in the sense that different sorts of variables (often in distinctive fonts) range over different universes of discourse, such as the domains of integers, real numbers, classes, lists, trees, etc. The logic of quantification applies most directly and advantageously to a *standard theory* [24] in which there is only one sort of variable ranging over one universe. Set theory standardizes to the universe of classes by representing numbers as classes [25] and relations, such as func-

tions, families, and lists, as classes of ordered pairs [26]. Ordered trees can be represented by the successive nesting of lists within lists.

The computational inefficiency encountered in representing numbers and lists by classes has lead to various attempts to use modified versions of set theory as a basis for programming languages [27-29]. The programming languages APL and LISP [30] take ordered collections as the point of departure. LISP, which is based on the theory of recursive functions, represents a finite list by the repeated nesting of ordered pairs and "atomic symbols" [30] within ordered pairs. Classes can then be represented as lists. In APL, a class in which the elements are classes of scalars can be represented by a matrix in which the rows or columns are interpreted as classes. Deeper levels of nesting are represented by component tensors having a greater number of index axes [1]. A component vector in which the items are distinct is called an ordered set in APL.

The theories of arrays and lists developed here standardize to the universe of arrays or lists. Both order and repetition of the items in an array or list are ignored in class-theoretic contexts, thereby representing classes directly as arrays or lists. The theories differ from LISP in treating arrays or lists as primitive objects instead of pairs alone. The theories differ from APL in nesting arrays within arrays or lists within lists according to the class-theoretic relation of membership. Computational inefficiency is avoided by representing numbers and characters as arrays or lists that contain themselves as sole members [24,31,32].

The nth ordinal power of a class U is the class $U\underline{*}n$ containing all families that map the ordinal n into U, where n is the class containing the n preceding ordinals. $U\underline{*}n$ represents the class of all lists of n elements of U. An *n-ary operation* [33] on the class U is a function that maps $U\underline{*}n$ into U. An *n-adic operation* is the intuitive counterpart of an n-ary operation defined on the universe of arrays or lists.

In a theory of classes, functions are defined to be classes of ordered pairs. The basic operations in the theory are used to represent, as functions, n-ary operations on classes. Similarly, in a theory of arrays or lists, functions can be defined as arrays or lists of pairs. The basic operations in the theory can be used to represent, as functions, n-ary operations on arrays or lists.

2. Syntax*

Monadic and dyadic operations have short right scope. Dyadic operations have long left scope. Grouping is also indicated by parentheses, spaces in a strand, and dots.

The basic, *dot-free* syntax used here is the same as the usual syntax of addition, subtraction, and negation in arithmetic. The names of monadic and dyadic operations are called *prefixes* and *infixes*, respectively. Prefixes and infixes take the shortest possible well-formed string to the right as right operand. Infixes take the longest possible well-formed string to the left as left operand. For example,

```
-A+B-(F-G+H)--D-E
```

is fully parenthesized as

```
((((-A)+B)-((F-G)+H))-(-D))-E.
```

The syntax, like that of APL, has no precedence or hierarchy. Within any given well-formed string, all infixes may be replaced by plus signs, and all prefixes may be replaced by negative signs without in any way changing the intended grouping or full parenthesization of the string (except for the internal parenthesization of the operation names themselves).

The syntax of suffixed dots is the same as Church's dot notation [34,35]. A dot suffixed to an infix or prefix causes the operation name to take the longest possible well-formed string to the right as right operand.

For example, the suffixed dots in Axiom 15 (in the appendices)

```
#A≤#C  →.  ~AρB=.~Aρ.~CρB
```

are replaced by left parentheses with the mating right parentheses as far to the right as is consistent with the nesting of paired parentheses already in place:

```
#A≤#C  →  (~AρB=(~Aρ(~CρB))).
```

Dot-free syntax then determines the full parenthesization:

```
((#A)≤(#C)) → (((~A)ρB)=((~A)ρ((~C)ρB))).
```

In the presence of suffixed dots, infixes take as left operand the longest possible well-formed string to the left up to the nearest unenclosed suffixed dot. For example, the left operand of the dyadic minus sign in

```
A+(A+.A+.-B+(A+.A+A)+C-A)+A
```

is `-B+(A+.A+A)+C`. The suffixed dot in the operand is enclosed in parentheses and so has no effect outside the parentheses.

This paper introduces prefixed dots that reassert long left scope over suffixed dots. The use of prefixed dots permits a symmetrical presentation of certain logical, Boolean, and arithmetical formulas. A dot prefixed to an infix causes the infix to take as left operand the longest

*The notation, syntax, and terminology adopted for the purposes of this paper do not conform to standard APL usage. Every grouping dot appearing at the end of a sentence should be read as a period.

possible well-formed string to the left including unenclosed suffixed dots.

The first step in fully parenthesizing a string is to replace the prefixed dots, if any, by right parentheses with the mating left parentheses as far to the left as is consistent with the nesting of paired parentheses already in place. For example, Theorem 2

$A=B \wedge .\ B=C\ .\rightarrow .\ A=C$

becomes

$(A=B \wedge .\ B=C) \rightarrow .\ A=C.$

Parenthesization is then completed according to the syntax of suffixed dots and dot-free syntax.

A *strand* is a string of characters composed of one or more variables, numerals, quoted strings, terms enclosed in parentheses, and constants separated by spaces. A space adjacent to a parenthesis may be omitted. Strands denote lists. For example, $D\ \bar{}3.14\ \text{'WORD'}\ (5+2)\ \Theta$ is a quintuple in which the items are the array D, the negative number $\bar{}3.14$, a list of four letters, the number 7, and the null Θ (which is an empty list).

The concatenating operation implicit in the formation of a strand is done before any other operation. For example, $-A\ B\ C$ equals $(-A)\ (-B)\ (-C)$ and $A\ B+C\ D$ equals $(A+C)\ (B+D)$.

Juxtaposed identifiers, such as 'COS' and 'ΔX' in 'COS ΔX' or the letters 'Δ' and 'A' in 'Δ A', are separated by spaces to mark where identifiers begin and end. Spaces between identifiers and basic operation symbols are needless but may be inserted to improve the legibility of a term or formula.

3. Extensionality

Nonempty arrays are equal if they hold the same items at the same addresses. For the purposes of the theory, equal arrays are identical.

Except for Sections 20 and 21, this section and all following sections hold for list theory as well as array theory. In list theory, the words "array" and "shape" are construed as "list" and "count," respectively. The axioms are stated for array theory but may be interpreted in list theory.

Every object in the universe of a standard theory of arrays (lists) is called an array (list). A *monadic operation* Δ maps the universe into itself by sending an array A to the array $\Delta\ A$. A *dyadic operation* Δ sends an ordered pair of the arrays A and B to the array $A\ \Delta\ B$.

Zermelo's Axiom I of *extensionality* is based on the principle that a "class is determined by its elements" [36]. By definition alone, equality is left-compatible (see Axiom 4 below) with the membership relation in the sense that equal classes contain the same classes. Extensionality grants to equality the force of logical identity by postulating that equal classes belong to the same classes, thereby making equality right-compatible (see Axiom 3) with the membership relation.

The same considerations guide the axiomatization of array theory. According to the array-theoretic *principle of extensionality*, a nonempty array is determined by its items and addresses. The principle is extended to empty arrays in Section 9. Nonempty arrays are taken to be *equal* if they hold the same items at the same addresses. Axioms 2, 3, and 4 give equality the force of logical identity by making equality compatible with all operations. Compatibility with the primitive operations would suffice. Axiom 1, together with Theorems 1 and 2, then establish that equality is a congruence on the universe of arrays. Substitutivity of equality then follows by induction on the number of primitive operation symbols in a formula.

Axiom 1. An array equals itself.

$A = A.$

Axiom 2. Any monadic operation applied to equal arrays returns equal values. Equality is compatible with every monadic operation Δ. (Read '→' as "implies.")

$A = B \rightarrow .\ \Delta\ A = \Delta\ B.$

Axioms 3 and 4. Equality is right-compatible and left-compatible with every dyadic operation Δ.

$A = B \rightarrow .\ A\ \Delta\ C =.\ B\ \Delta\ C$

$A = B \rightarrow .\ C\ \Delta\ A =.\ C\ \Delta\ B.$

4. Elementary arrays

Pairs, singles, and the null exist. In list theory, shape becomes count. The count of an array is a mote, which is a single that holds itself.

Zermelo's Axiom II of *elementary classes* [36] assumes the existence of empty classes, unit classes, and unordered pairs. The array-theoretic *postulate of elementary arrays* assumes the existence of empty lists, singles, and pairs. There exists the *null* Θ, which is one of infinitely many empty lists in the universe of arrays. Definition 2 (in the appendices) takes the null as primitive. For the present, let the singling operation be primitive. Singling sends an array A to the *single* $\circ A$ that holds A as sole item. The primitive pairing operation sends arrays A and B, given in the order stated, to the list A,B, called the *pair* of A *with* B, that holds A as first item and B as second item.

The *count* $\#A$ of an array A is an ordinal number representing the axis length (order type) of the list got by listing the items of A in main order. *Main order* is in-

duced by listing the addresses to the items in *lexicographic order*. For example, an odometer cycles lexicographically through a domain of addresses. Each wheel of the odometer is interpreted as cycling through the indices on an index axis. The length of an axis corresponds to the ordinal number of indices on a wheel. Since a list has one axis, the items of a list occur in main order by virture of being in a list.

The count of a list gives complete information on how the items of the list are addressed. Similarly, the *shape* $\sim A$ of an array A, which is essentially a list of the axis lengths of the zero or more index axes for A, gives complete information on how the items of A are addressed (Section 20). When arrays are restricted to lists, shape is restricted to count.

In the particular case of APL, the size of a list or component vector V is a list holding the count of V as sole item. The shape of a list in array theory is its count. For example, if the count of a list is 7 in a theory of lists or arrays, then 7 itself, rather than the one-item list holding 7, is the shape of the same list in the theory of arrays. The shape of an array other than a list is the same as the APL size of the array.

The operation of counting is primitive in list theory just as the operation for taking the shape of an array is primitive in array theory. The changing of shape to count is the essential step for converting the array-theoretic results listed in the appendices to list-theoretic results. The conversion is done as follows:

> Omit Definition 1, Axiom 5, and Theorems 5, 10, 11, 12, and 75, together with their corollaries. The corollary $\times\Theta=1$ of Axiom 5 may be postulated as an isolated result. Omit all variations of the form $\#\sim A=0$, such as $\sim,\sim\circ A=0$. In all definitions, axioms, theorems, corollaries, and proofs, replace equations of the form $\sim A=\Theta$ by $\#A=1$ and terms of the form $\Theta\rho A$ by $1\rho A$. Replace all tildes by sharps. Omit Definition 5 and all commas.

If counting is an array-theoretic operation and ordinal numbers are to be immediately accessible for computation, then $\#A$ must be construed as a relatively simple array. As suggested by Quine's representation of an individual as a unit class that contains itself [24], let a *mote* be an array that holds itself as sole item. A pure theory of arrays has no motes, just as a pure theory of classes has no unit classes containing themselves.

Ordinal numbers might be represented in a pure theory of arrays by von Neumann's (1923) construction. For reasons of practicality, however, ordinal numbers are appended to the theory as motes. Complex numbers, which include the reals and finite ordinals, and characters, which include the letters and operation symbols, are also appended to the theory as motes.

If an array A is a mote, then both A and the single $\circ A$ of A hold A as sole item. If A and $\circ A$ are also made to have the same number of axes, as would certainly be the case in a theory of lists where all arrays have one axis, then A and $\circ A$ must hold the same item at the same address. By the principle of extensionality, A and $\circ A$ are equal. An array is a mote if and only if it equals its single.

> *Axiom 6.* The count of an array is a mote.
>
> $\#A = \circ\#A$.

An ordinal number, which exists as an array because it is the count of an array, is a single that contains itself as sole item. The count of a pair is 2, of a single, 1, of an empty list, 0.

> *Definitions 2 and 3.* Zero is the count of the null. One is the count of zero.
>
> 0 for $\#\Theta$
>
> 1 for $\#0$.

If arrays A and B are *equal*, identical, or the same, then $'A=B'$ denotes the ordinal number 1, which is interpreted as truth, and $'A\neq B'$ denotes the ordinal number 0, which is interpreted as falsity. Zero and one are *truth-values* as well as numbers. If A differs from B, then $A\neq B$ is 1 and $A=B$ is 0. The primitive operation of equality has the same meaning and notation in array theory that it has in a first-order theory with equality [35].

5. Recounting

Lists, initial segments, and solitaries are defined in terms of recounting, which is the list-theoretic counterpart of reshaping. Counts are ordered.

The primitive array-theoretic operation of reshaping, which is based on the dyadic ρ function in APL\360, becomes the list-theoretic operation of *recounting* when restricted to the formation of lists. If A is an arbitrary array, then $\#A$ is an arbitrary ordinal number or count. The recounting of an array B to the count of A produces a list $\#A\rho B$ on an axis of length $\#A$ in which the items are taken cyclically in main order from B. For example, if B is the pair $P\ Q$ of arrays P and Q, then $5\rho B$ equals $P\ Q\ P\ Q\ P$ and $3\rho\circ B$ equals the triple $(P\ Q)\ (P\ Q)\ (P\ Q)$.

$\#A\rho A$ is the list that holds the items of A in main order. If A is not a list, then it is meaningful to define the *list* $,A$ of A to be $\#A\rho A$. Listing corresponds to raveling in APL\360.

> *Definition 5.* The listing of an array equals the recounting of the array to its count.
>
> $,A$ for $\#A\rho A$.

An array A is a list if and only if $A=,A$. In a theory of

lists, $A=,A$ for every A because every object A in the universe is a list. The listing operation is superfluous in a theory of lists.

The listing operation permits Axiom 21 to be restated for list theory in a way that is also correct for array theory. When so stated, Axiom 21 becomes $\#A\rho A=,A$ and takes the place of Definition 5. The comma is ignored in list theory but observed in array theory.

Axiom 21. The reshaping of an array to its own shape produces the array itself.

$\sim A\rho A = A.$

Since the items of $\#A\rho B$ are taken cyclically in main order from B, the items may as well be taken cyclically from the list of B. Thus $\#A\rho B=.\#A\rho,B$, which is a corollary of Theorem 28. The count of an array A is taken to be the axis length of the list of A, which is the import of Axiom 17. Counting and reshaping treat arrays as lists, as do many of the basic operations in array theory.

Axiom 17. The count of an array equals the count of its list.

$\#,A = \#A.$

Axiom 10. The count of an array equals the shape of its list.

$\sim,A = \#A.$

The primitive operation of *ordering* is defined first for motes and then extended to arbitrary arrays by the assumption that ordering is a mote operation (Section 11). If A and B are both ordinal numbers or both real numbers, then ordering is defined in the sense that $A\leq B$ is either true or false. If A and B are motes and $A\leq B$ is neither true nor false, then ordering is undefined in the sense that $A\leq B$ is the cipher ⍬, which is a mote other than a number or character.

$\#A\leq\#B$ equals 0 or 1 according as $\#A$ does or does not follow $\#B$ in the succession of ordinal numbers. If A and B are real numbers, then $A\leq B$ equals 0 or 1 according as A is or is not greater than B. $A<B$ equals $\neg.B\leq A$ by definition. (Read '$\neg$' as "not.")

If $\#A\leq\#B$, then only the first $\#A$ items in main order are taken from B to make the list $\#A\rho B$, which is called an *initial segment* of B. For example, $2\rho P\ Q\ R$ equals the pair $P\ Q$ and $1\rho P\ Q\ R$ equals the solitary $,\circ P$ holding P as sole item. A *solitary* is a list that holds one item.

In a theory of lists, every single is a solitary. In a theory of arrays, according to Theorem 10, no single is a solitary. Theorems asserting the nonequality of various arrays, other than arrays involving characters or the cipher, follow from Axiom 11. Without this axiom, all arrays constructed from numbers alone might be the same.

Axiom 11. Zero differs from one.

$0 \neq 1.$

6. Reshaping

If certain premisses hold, then it does not matter on a further reshaping whether items are taken cyclically from an array or the array reshaped. Vacating is zero reshaping.

Axioms 15, 16, and 19 amount to one postulate, called the *reshaping* or *recounting postulate*. When phrased in terms of recounting, this postulate asserts that $\#A\rho B =. \#A\rho.\#C\rho B$ whenever A has no more items than C, or B is empty, or the count of B both divides the count of C and is no greater than the count of C. For example, if A has 4 items, C has 5 items, and B is the triple $P\ Q\ R$, then $5\rho B$ equals $P\ Q\ R\ P\ Q$ and both $4\rho B$ and $4\rho(5\rho B)$ equal $P\ Q\ R\ P$.

If $\#C\rho B$ is a list formed by cycling through all of the items of B an exact ordinal number of times, which is vacuously the case if B is empty, then it does not matter on a further recounting whether the items are taken cyclically from B or from $\#C\rho B$. For example, if C has 6 items and B is the pair $P\ Q$, then $5\rho P\ Q$ and $5\rho P\ Q\ P\ Q\ P\ Q$ are the same.

Except for the shape of the final result, whether it be an array of shape $\sim A$ or a list of count $\#A$, the recounting postulate is essentially the same as the reshaping postulate. If the premisses hold, then it does not matter on a further recounting or reshaping whether the items are taken cyclically from B or from $\sim C\rho B$.

Axioms 15, 16, and 19. The reshaping postulate. If the count of A is no greater than the count of C, or if B is empty, or if the count of B both divides the count of C and is no greater than the count of C, then the reshaping of B to the shape of A equals the reshaping of B first to the shape of C and then to the shape of A.

$\#A\leq\#C \rightarrow. \sim A\rho B =. \sim A\rho.\sim C\rho B$

$\#B=0 \rightarrow. \sim A\rho B =. \sim A\rho.\sim C\rho B$

$\#B\leq\#C \wedge. \#B|\#C=0 .\rightarrow. \sim A\rho B =. \sim A\rho.\sim C\rho B$

Axioms 12, 13, 14, 18, and 20 postulate the few elementary facts about the ordering and divisibility of the ordinal numbers needed to satisfy the premisses of Axioms 15 and 19. The sequence of axioms is determined by the principle that no axiom is introduced until it is needed to deduce further results.

Axioms 12, 13, and 14. Zero is no greater than any count. A count is no greater than itself. The count of a nonempty array is no less than one.

$0 \leq \#A$

$\#A \leq \#A$

$\#A\neq 0 \rightarrow. 1\leq\#A.$

For every nonzero ordinal $\#A$, called the *divisor*, and every ordinal $\#B$, called the *dividend*, there exist unique

ordinals $\#A\lfloor\#B$ and $\#A|\#B$, called the *quotient* and *remainder* [19], respectively, for which $\#A\lfloor\#B\times\#A + (\#A|\#B) = \#B$ and $\#A|\#B<\#A$. If the remainder is zero, then the divisor *divides* the dividend.

The $\#A$ quotient of $\#B$ is the greatest ordinal number of lists of length $\#A$ that can be dropped or deleted as successive initial segments from a list $,B$ of length $\#B$. The $\#A$ remainder of $\#B$ is the length of the final segment of $,B$ remaining after the first $\#A\lfloor\#B\times\#A$ items have been dropped. For the present, the remainder operation is taken as primitive.

Axioms 18 and 20. A count divides itself. One divides any count.

$\#A|\#A = 0$

$1|\#A = 0.$

The recounting of an array A to the count of an empty array produces a list $0\rho A$ of count 0, called the *empty list* of A, in which the items are taken cyclically from A, or equivalently, from the list of A.

Definition 6. The *vacating* or empty listing of an array equals the zero recount of the array.

$\backslash A$ for $0\rho A.$

According to Theorems 15 and 16, an array A is empty if and only if $,A=\backslash A$. In particular, an array A is an empty list if and only if $A=\backslash A$. By a second corollary to Theorem 28, if an array B is empty, then $\#A\rho B$ and $\#A\rho,B$ and $\#A\rho\backslash B$ are equal. Vacating is a useful operation in the study of empty arrays because many operations, such as counting and recounting, treat arrays as lists and therefore empty arrays as empty lists.

The recounting of an empty array B to the count of A produces a list $\#A\rho B$ on an axis of length $\#A$ in which the items are taken cyclically from B, or equivalently, from the empty list of B. $\#A\rho B$ and $\#.\#A\rho B$ are undefined in APL\360 when B is empty and A is not. Rather than provide for this one exception by adding restrictive premisses to many theorems, thereby missing some of the advantages of a standard theory, Axiom 9 postulates in list theory that $\#(\#A\rho B)=\#A$ for all A and B.

Axiom 9. The reshaping of an array B to the shape of an array A produces an array having the shape of A.

$\sim(\sim A\rho B) = \sim A.$

Axiom 9 requires $3\rho B$ to have 3 items even if B is empty. The items of $3\rho B$ are taken cyclically from $\backslash B$. What items come out of the empty list of an array?

The answer provided in the next section was originally found by interpreting earlier versions of the axioms [9b] and observing in APL\360 that a solitary zero or a solitary blank results from the zero expansion of the empty numerical list or the empty character list, respectively. The same answer can be inferred directly from the properties of a vector space.

7. Patterns

The vector sum of a vacated list of vectors is the zero vector, which in certain cases may be taken as the pattern of the list of vectors.

An array built up from numbers alone or characters alone is said to be *homogeneous*. Each more or less deeply nested number or character in an array can be reached by a *path*, which is a list of successive addresses. A path amounts to an address in depth. Roughly speaking, an array is said to be *uniform* if it is rectangular in depth.

The mathematics of patterns evolved from the search for a simple construction based on simple axioms that would typify the items of an array having some regularity of structure. Patterns typify the items of uniform, homogeneous arrays in the same way that the zero and the blank typify arrays of numbers and characters in APL. Essentially the same construction can be deduced from an interpretation of empty lists of vectors in a vector space.

The class containing the zero vector alone is a vector space, called a *zero space* or *zero subspace*. In a 3-dimensional vector space, planes, lines, and points through the origin are subspaces having dimensions 2, 1, and 0, respectively. The unique zero subspace has dimension 0 and must be spanned by the empty class. The zero vector in the zero space is a linear combination of the vectors in the empty class. Consequently, the vector sum of no vectors must be the zero vector [37].

For example, if addition and multiplication of pairs of real numbers are defined so that the pairs combine like complex numbers, then the class of all such pairs is a field. Each pair of real numbers is then called a scalar. The zero scalar is the pair 0 0. If P equals (2 ¯3) (¯1 4) (0 5) and Q equals (1 2) (3 ¯2) (¯7 0), then P and Q are vectors in the vector space of triples of scalars.

If $A = P\ Q$, then A is a list of vectors. The sum $+A$ of the list A is best interpreted as a vector sum of vectors. $+A$ is calculated as follows:

(2 ¯3) (¯1 4) (0 5) + (1 2) (3 ¯2) (¯7 0)

(2 ¯3 + 1 2) (¯1 4 + 3 ¯2) (0 5 + ¯7 0)

((2+1)(¯3+2))((¯1+3)(4+¯2))((0+¯7)(5+0)).

Thus $+A$ is the vector (3 ¯1) (2 2) (¯7 5). This example illustrates the positional operator (Section 11) and shows why dyadic addition is taken to be a mote operation.

As shown earlier, the vector sum of no vectors is the zero vector. Hence the sum $+\backslash A$ of the empty list $\backslash A$ of the list A in the example above must be the zero vector (0 0) (0 0) (0 0). If B is a list of vectors in the

vector space of all quadruples of real numbers, then $+\backslash B$ equals 0 0 0 0. Nearly all of the theorems about a vector space still hold if the field is relaxed to a division ring of scalars, such as the division ring of quaternions [16]. If C is a list of vectors in the module of all pairs of scalars over the division ring of quaternions, then $+\backslash C$ equals (0 0 0 0) (0 0 0 0).

All empty classes are equal in set theory. If M is an empty class of triples of pairs of real numbers and N is an empty class of quadruples of real numbers, then M and N are the same. The vector sum of M differs from the vector sum of N because the summing operations are defined for different vector spaces.

In a theory of arrays, there is only one summing operation. If the sum $+\backslash A$ differs from the sum $+\backslash B$, then the burden of the difference is borne by $\backslash A$ and $\backslash B$. Even though $\backslash A$ has no items, there is residual information associated with $\backslash A$, called the *pattern* [9] of A (or of $\backslash A$), which indicates that $\backslash A$ is the result of vacating an array A holding triples of pairs of real numbers. If the pattern of A is taken to be the zero vector corresponding to the vectors in A, then the sum $+\backslash A$ of the empty list of A may be taken directly as the pattern of A, provided that characters and the cipher are not used in the construction of A (Section 11).

The pattern of A is (0 0) (0 0) (0 0) and the pattern of B is 0 0 0 0. The pattern of a nonempty array is the same as the pattern of its empty list. In answer to the question asked at the end of the preceding section, each item coming out of an empty array, got by recounting the array to a nonempty list, is taken to be the pattern of the array. For example, if A has the pattern above, then $2\rho\backslash A$ is the pair ((0 0) (0 0) (0 0)) ((0 0) (0 0) (0 0)).

8. Extracting

The extract of an array is either the pattern or the first item of the array. Picking is defined in terms of extracting and dropping.

The *extract* $\supset A$ of a nonempty array A is the first item (in main order) in A. The primitive operation of extracting takes the first item, rather than the last, because certain infinite arrays have no last item.

Axiom 7. The extract of a singled array is the array itself.

$\supset\circ A = A$.

If A is a mote, then $A=\supset A$ by Theorem 4. Any finite number of extractings or singlings of a mote returns the mote as value. This observation suggests that a mote may be interpreted as an infinite nesting $\circ\circ\circ\ldots B$ of singles within singles.

According to Theorem 37, $1\rho A=,\circ\supset A$. The recounting of an array A to a list of length 1 is the solitary holding the extract or first item of A. (The comma is ignored or omitted in list theory.) Theorem 37 is the list-theoretic counterpart of Axiom 8.

Axiom 8. The null reshape of an array is the singled extract of the array.

$\Theta\rho A = \circ\supset A$.

If A is empty, then, as suggested by the properties of a vector space, the item got by recounting A is the pattern of A. Thus $1\rho A$, for A empty, is the solitary holding the pattern of A. If Theorem 37 is to hold for all arrays, as would be desirable in a standard theory, then $1\rho A$ must also be the solitary holding $\supset A$. By the principle of extensionality (cf. Theorems 3 and 32), the extract of an empty array must be the pattern of the array.

Since extracting takes the first item of an array in main order, $\supset A=\supset,A$ by Theorem 31. If A is empty, then $,A=\backslash A$ and so $\supset A=\supset\backslash A$ by Theorem 33. As defined earlier, the residual information associated with $\backslash A$ is called the pattern of A. Since $\backslash\backslash A=\backslash A$ by Theorem 18, the pattern of A is the same as the pattern of $\backslash A$, which is the extract of $\backslash A$ because $\backslash A$ is empty. Thus $\supset\backslash A$ is the pattern of A for every A.

According to Theorem 46, $\backslash\#A=\Theta$. The null is the empty list of any ordinal number. Theorem 46 is the list-theoretic counterpart to Axiom 22. $\backslash\Theta=\Theta$ because $\backslash\backslash\#A=\backslash\#A$ by Theorem 18.

Axiom 22. Vacating a shape produces the null.

$\backslash\sim A = \Theta$.

The field of real numbers over itself is a vector space. Each real number, and therefore each finite ordinal, may be interpreted as a vector. In particular, 0 is the zero vector. If an array A is *finite* in the sense that its count is finite, then $\backslash\#A$ is the empty list of vectors. $+\backslash\#A=0$ and $+\Theta=0$ because the vector sum of no vectors is the zero vector. The sum of the empty list of an array (without characters or the cipher) is taken as the pattern of the array. Hence 0 must be the pattern $\supset\backslash\Theta$ of the null. $\supset\Theta=\supset\backslash\Theta$ because $\Theta=\backslash\Theta$.

Axiom 23. The extract of the null is zero.

$\supset\Theta = 0$.

The sum of an array is defined to be the sum of all the items in the array in main order. Thus $+A=+,A$ for all A. The sum $+\circ A$ of a single equals the sum $+,\circ A$ of the corresponding solitary. In the context of a vector space, the sum of a single or solitary vector is the vector itself. For example, $+\circ$ (2 4) (6 1) equals (2 4) (6 1). This observation suggests that the sum of the empty list of an array be taken as the sum of the singled pattern of the array, i.e., $+\backslash A = +\circ\supset\backslash A$.

If A is empty, then $+A=+\circ\supset A$. The sum of an empty

array is converted to the sum of a nonempty array (Section 11).

Operations for fetching items by address are defined in terms of operations for moving and then extracting the desired items. The definition of picking in terms of extracting and dropping illustrates the fundamental nature of the picking operation. Axioms for dropping are deferred to Part II [to be published].

The primitive operation of dropping is essentially the same as the APL function of dropping. The $\#A$ *drop* of the list of B is the list $\#A\downarrow,B$, called a *final segment* of B, that holds all but the first $\#A$ items of B listed in main order. When combined with dropping, the operation of extracting can be used to pick any particular item from an array. The $\#A$ *pick* of the list of B is the item $\#A\supset,B$ at index $\#A$ in the list of B. $\#A\supset,B$ is the first item in the final segment of B beginning at $\#A$. Thus

$$\#A\supset,B = \supset.\#A\downarrow,B.$$

For example, $2 \supset P\ Q\ R\ S = R$. These list-theoretic equations follow from the array-theoretic definition of $\sim A\supset B$ as $\supset.\sim A\downarrow B$.

The preceding equation certainly holds if $\#A<\#B$. If the equation holds for all A and B and if $\#A\downarrow,B$ is defined to be the empty list of B whenever $\#B\leq\#A$, then

$$\#B\leq\#A \rightarrow .\ \#A\supset,B = \supset\backslash B.$$

If the index $\#A$ lies outside the range of indices for the list of B, then $\#A$ picks the pattern of B. Since $\#\backslash B\leq 0$ for all B, $0\supset\backslash B=\supset\backslash B$. Thus $0\supset,A$ equals both $\supset,A$ and $\supset A$ for every A.

9. Plans

Patterns and plans are calculated in terms of each other, extend extensionality to empty arrays, identify data types, and, when equal, provide the basis for an axiom of foundation.

In the recounting of an array A, it does not matter whether the items are taken cyclically from A or from the list of A. In particular, $0\rho A=.0\rho,A$, which proves that $\backslash A=\backslash,A$ (Theorem 29). According to the recounting postulate, $0\rho A=.0\rho.1\rho A$. Hence $\backslash A=\backslash,\circ\supset A$ by the list-theoretic version of Axiom 8. The foregoing proves Theorem 26, $\backslash A = \backslash\circ\supset A$, which is a simple yet fundamental result that has an important corollary:

$$\supset\backslash A = \supset\backslash\circ\supset A.$$

Theorem 26 means that the vacating of an array A can be done by vacating the single or solitary holding the first item or extract of A. According to the corollary, the pattern of an array equals the pattern of its singled extract.

If arrays A and B have the same pattern, then $\supset\backslash A=\supset\backslash B$ and so $\backslash\circ\supset\backslash A=\backslash\circ\supset\backslash B$. Hence $\backslash A=\backslash B$ by Theorem 26 and the idempotency of vacating (Theorem 18). Arrays have the same pattern if and only if ($\equiv$) they have the same empty list (Theorem 27):

$$\supset\backslash A=\supset\backslash B \equiv .\ \backslash A=\backslash B.$$

According to Theorem 42, if arrays A and B have the same shape and the same listing, then $\sim A\rho A=.\sim B\rho B$ and so $A=B$ by Axiom 21. Empty arrays have the same listing if and only if they have the same pattern. Thus Theorem 42 proves the principle of extensionality for empty arrays.

Extensionality. Empty arrays are equal if they have the same pattern and shape. Nonempty arrays are equal if they hold the same items at the same addresses.

Let A be the list $P\ Q$, where P equals $(2\ {}^{-}3)\ ({}^{-}1\ 4)\ (0\ 5)$ and Q equals a similar vector as in the example of Section 7. The pattern of the single of an array is called the *plan* of the array. According to the corollary of Theorem 26, the pattern of A equals $\supset\backslash\circ\supset A$, which is the plan of P. Thus the plan of P equals $(0\ 0)\ (0\ 0)\ (0\ 0)$, which is the zero vector in the vector space containing P and Q.

The pattern of P is the zero vector $0\ 0$ in the vector space of pairs of real numbers. Thus the plan $\supset\backslash\circ P$ of P equals the list $\#P\rho\circ\supset\backslash P$ holding three repetitions of the pattern of P. By the corollary of Theorem 26 again, the pattern of P equals $\supset\backslash\circ\supset P$, which is the plan of the first item of P. The pattern of the first item $\supset P$ of P is the zero vector 0 in the vector space of real numbers over itself. The pattern of P, which equals the plan of $\supset P$, is the list $\#\supset P\rho\circ\supset\backslash\supset P$ holding two repetitions of the pattern of $\supset P$. For the list P in this example,

$$\supset\backslash\circ P =.\ \#P\rho\circ.\#\supset P\rho\circ\supset\backslash\supset P.$$

The preceding example suggests that if $\supset\backslash\circ,A=.\ \#A\rho\circ\supset\backslash A$ is taken as an axiom in list theory, or equivalently, if $\supset\backslash\circ A=.\sim A\rho\circ\supset\backslash A$ is taken as an axiom in array theory, then the pattern of an array B can be computed as the plan of the extract of B. The plan of $\supset B$ is an array having the same shape as $\supset B$, in which each item is the pattern of $\supset B$. The pattern of $\supset B$ is then computed as the plan of $\supset\supset B$, etc.

The equation $\supset\backslash\circ A=.\sim A\rho\circ\supset\backslash A$ can be reduced to a simpler form. For all arrays A and B, $\sim A\rho\circ\supset\backslash B$ equals $\sim A\rho.1\rho\backslash B$, which equals $\sim A\rho\backslash B$ by the recounting postulate because $\backslash B$ is empty (Theorem 24). Thus the plan of A may be taken as $\sim A\rho\backslash A$.

Axiom 24. The plan of an array A is an array of the same shape as A in which each item, being taken from the empty list of A, is the pattern of A. $\supset\backslash\circ A =.\ \sim A\rho\backslash A$.

Theorems 50 and 52 show that the alternate calcula-

tion of patterns and plans leads to the following two theorems, which can be extended indefinitely:

$$\supset\backslash B =. \sim\supset B\rho\circ .\sim\supset\supset B\rho\circ .$$
$$\sim\supset\supset\supset B\rho\circ .\sim\supset\supset\supset\supset B\rho\circ\supset\backslash\circ\supset\supset\supset\supset\supset B$$

$$\supset\backslash\circ C =. \sim C\rho\circ .\sim\supset C\rho\circ .$$
$$\sim\supset\supset C\rho\circ .\sim\supset\supset\supset C\rho\circ\supset\backslash\circ\supset\supset\supset\supset C.$$

The second theorem follows from the first by substituting $\circ C$ for B. According to Theorem 53, an empty array equals its plan.

The pattern of B is an array of shape $\sim\supset B$ in which each item is an array of shape $\sim\supset\supset B$, etc., until $\supset\supset\ldots\supset\supset B$ is a mote. Then $\supset\backslash\circ\supset\supset\ldots\supset\supset B$ equals $\supset\backslash\supset\ldots\supset\supset B$, which is the pattern of the mote. According to the corollary $\supset\backslash\#A=0$ of Theorem 48, every ordinal number has 0 as pattern. If provision is made to distinguish the floating-point zero from the integer zero, then the floating-point numbers having floating-point zero as pattern can be distinguished from the integers having integer zero as pattern. Patterns can be used to identify data types.

Axioms 26 and 27. The pattern of the cipher is the cipher. The cipher differs from zero.

$$\supset\backslash\boxed{0} = \boxed{0}$$

$$\boxed{0} \neq 0.$$

Axioms 29, 30, and 31. The pattern of the blank is the blank. The blank differs from zero and the cipher.

$$\supset\backslash\text{' '} = \text{' '}$$

$$\text{' '} \neq 0$$

$$\text{' '} \neq \boxed{0}.$$

Theorems 74, 79, and 85 assert that if the plan of an array A equals zero, the cipher, or the blank, then A is a mote and the pattern of A is zero, the cipher, or the blank, respectively. The introduction of motes is controlled so that only numbers have zero as pattern and only characters have the blank as pattern. Axiom 28 assumes that the cipher is the only mote having the cipher as pattern. Thus an arbitrary array A is a number, the cipher, or a character according as the plan of A is zero, the cipher, or the blank.

Axiom 28. Only the cipher has a plan equal to the cipher.

$$\supset\backslash\circ A=\boxed{0} \rightarrow .\ A=\boxed{0}.$$

Definitions 7, 8, and 9. The *void* is the empty list of the cipher. The *nil* is the empty list of the blank. A pair of adjacent quotes also denotes the nil.

$\underline{0}$ for $\backslash\boxed{0}$

⍬ for $\backslash$' '

'' for ⍬.

If an array A holds itself as first item, then $A=\supset A$ and the pattern and plan of A are equal because $\supset\backslash A=\supset\backslash\circ\supset A$. In particular, the pattern and plan of a mote are equal. Since arrays have the same pattern if and only if they have the same empty list, the pattern and plan of an array A are equal if and only if $\backslash A=\backslash\circ A$.

Zermelo's (1930) axiom of *Fundierung* [6], foundation [38], or regularity [39], which is based on von Neumann's (1929) axiom, prevents a class from containing itself at any deeper level of nesting by postulating that every nonempty class M contains a class having no members in common with M. The full strength of this axiom is not wanted for array theory because motes are assumed to exist. Axiom 25 assumes instead that no array other than a mote holds itself as first item or has a pattern equal to its plan.

Axiom 25. If an array has the same empty list as its single, then the array is a mote.

$$\backslash A=\backslash\circ A \rightarrow .\ A=\circ A.$$

10. Replacement

The replacement transform of a monadic operation applies the operation to each item of an array. Cartesian products show how replacement transforms apply to empty arrays.

Skolem (1922) and Fraenkel (1922) added to Zermelo's seven axioms the following axiom schema of *replacement* [40] or substitution [38]: If M is any class and Δ is any singulary function sending classes to classes, then the range of all $\Delta\ x$ for x in M is again a class. When augmented by replacement, Zermelo's system is called *Zermelo-Fraenkel* set theory.

Replacement. For every array A and every monadic operation Δ, the result of replacing at each address in A an item X by $\Delta\ X$ is again an array.

To each monadic operation Δ there corresponds by the *replacement operator*, which is denoted by a colon, another monadic operation $:\Delta$ that applies Δ to the items of an array according to the principle of replacement. $:\Delta\circ A=\circ\Delta\ A$ and $:\Delta(A\ ;B)=.\Delta\ A\ ;\Delta\ B$ for all A and B. The operation $:\Delta$ is called *replacement* Δ or the *replacement transform* of Δ. Operator names, such as the colon, have the same dot-free syntax with respect to operation names that operation names have with respect to array names.

Axiom 32. If Δ is a monadic operation and A and B are nonempty arrays, then it does not matter whether replacement Δ is applied before or after the reshaping of B to the shape of A.

$$\#A\neq 0\ \wedge .\ \#B\neq 0\ .\rightarrow .\ :\Delta(\sim A\rho B)=.\sim A\rho:\Delta\ B.$$

For example, $:\Delta.3\rho P\ Q$ equals $:\Delta\ P\ Q\ P$ equals $(\Delta\ P)\ (\Delta\ Q)\ (\Delta\ P)$ equals $3\rho(\Delta\ P)\ (\Delta\ Q)$ equals $3\rho:\Delta\ P\ Q$.

The *Cartesian product* $\times A$ of a finite list A of lists is an

array having $\#A$ axes in which the first axis has the length of the first list in A, the second axis has the length of the second list in A, etc. (Section 21). The items of $\underline{\times}A$ are all possible lists of count $\#A$ in which the first item is taken from the first list in A, the second item is taken from the second list in A, etc. In a theory of lists, the Cartesian product of A may be taken as the list of $\underline{\times}A$.

For example, let A be the triple $(P\ Q)\ (,\circ R)\ (S\ T)$. The Cartesian product $\underline{\times}A$ is a 2 by 1 by 2 array of triples.

$$,\underline{\times}A = (P\ R\ S)\ (P\ R\ T)\ (Q\ R\ S)\ (Q\ R\ T).$$

The Cartesian product of a solitary list is a list of solitaries. For example, $\underline{\times},\circ P\ Q\ R$ equals $(,\circ P)\ (,\circ Q)\ (,\circ R)$. In particular, the Cartesian product $\underline{\times},\circ\backslash A$ of the solitary $,\circ\backslash A$ of an empty list $\backslash A$ should be an empty list $\backslash\circ,\circ B$ in which each item is a solitary of the form $,\circ B$. Given A, what is B? If $A = P\ Q\ R$, then the first item of $\underline{\times},\circ A$ is the solitary $,\circ P$ holding the first item of A. If $A=\backslash P\ Q\ R$, so that $A=\backslash A$, then the extract of $\underline{\times},\circ\backslash A$ is the solitary $,\circ\supset\backslash A$ holding the extract of $\backslash A$. Thus for every array A,

$$\underline{\times},\circ\backslash A = \backslash\circ,\circ\supset\backslash A.$$

The list $:(,\circ)P\ Q\ R$ equals $(,\circ P)\ (,\circ Q)\ (,\circ R)$. Hence, by the preceding example, $\underline{\times},\circ P\ Q\ R = :(,\circ)\ P\ Q\ R$ for all arrays P, Q, and R. In array theory, the Cartesian product of a single or solitary array is an array of singles or solitaries. The following two equations hold intuitively for every nonempty array A and are postulated to hold for every A.

$$\underline{\times}\circ A = :\circ A$$

$$\underline{\times},\circ A = :(,\circ)A.$$

In particular, $\underline{\times},\circ\backslash A$ must equal $:(,\circ)\backslash A$. Consequently, for every A,

$$:(,\circ)\backslash A = \backslash\circ,\circ\supset\backslash A.$$

This last equation suggests Theorem 91, which asserts for every monadic operation Δ and every array A that $:\Delta\backslash A = \backslash\circ\Delta\supset\backslash A$. When empty listing is generalized to empty reshaping, Theorem 91 becomes Axiom 33.

> *Axiom 33.* If Δ is a monadic operation and A is an empty array, then replacement Δ applied to the empty array resulting from a reshaping of an array B to the shape of A produces an empty array of the shape of A in which each item is Δ applied to the pattern of B.
>
> $$\#A=0 \rightarrow .\ :\Delta(\sim A\rho B)=.\sim A\rho\circ\Delta\supset\backslash B.$$

Axiom 33 and its immediate result, Theorem 91, show why Axiom 32 is restricted to nonempty arrays:

$$\#A\neq 0\ \wedge .\ \#B\neq 0\ .\rightarrow .\ :\Delta(\sim A\rho B)=.\sim A\rho:\Delta\ B.$$

If $A=\Theta$ and $B=1$ and the premiss $\#A\neq 0$ is omitted, then $\sim A=0$ and $\#B\neq 0$ and so $:\Delta\backslash 1=\backslash:\Delta\ 1$. But $\backslash:\Delta\ 1$ equals $\backslash:\Delta\circ 1$, which equals $\backslash\circ\Delta\ 1$. By Theorem 91, $:\Delta\backslash 1$ equals $\backslash\circ\Delta\supset\backslash 1$, which equals $\backslash\circ\Delta\ 0$. If Δ is chosen so that $\Delta\ 0 = 0$ and $\Delta\ 1 = \boxed{0}$, then $\backslash\circ 0=\backslash\circ\boxed{0}$. This is a contradiction because zero and the cipher have different patterns.

If $A=,0$ and $B=\Theta$ and the premiss $\#B\neq 0$ is omitted, then $\#A\neq 0$ and $\sim A=1$ and so $:\Delta,\circ\supset\Theta=,\circ\supset:\Delta\Theta$. But $:\Delta,\circ\supset\Theta$ equals $,\circ\Delta\supset\Theta$, which equals $,\circ\Delta\ 0$. Hence $\Delta\ 0=\supset:\Delta\Theta$ by Theorem 32. According to Theorem 91, $\supset:\Delta\Theta$ equals $\supset:\Delta\backslash\Theta$ equals $\supset\backslash\circ\Delta\supset\backslash\Theta$ equals $\supset\backslash\circ\Delta\ 0$. If Δ is chosen so that $\Delta\ 0=1$, then $1=\supset\backslash\circ 1$ and so $1=0$.

Even though Axioms 32 and 33 have restrictive premisses, the two axioms work together to prove theorems without premisses. For example, Theorems 98, 99, and 100 hold for every array A:

$$:\Delta\ A =.\ \sim A\rho:\Delta\ A$$

$$\sim:\Delta\ A = \sim A$$

$$:\Delta,A = ,:\Delta\ A.$$

The operator of composition, which is denoted by juxtaposition, combines two monadic operations to produce a third. For example, the composition of patterning $\supset\backslash$ with singling $\circ$ produces the operation $\supset\backslash\circ$ of planning. $(\supset\backslash)\circ$ and $\supset(\backslash\circ)$ are the same operation because composition is associative. The class of all monadic operations is a semigroup under composition. The replacement operator is an endomorphism of the semigroup. For all monadic operations Δ and $\underline{\Delta}$ and for every array A,

$$:(\underline{\Delta}\ \Delta)A = :\underline{\Delta}:\Delta\ A.$$

If $:(,\circ)\backslash A$, which equals $\underline{\times},\circ\backslash A$ and therefore $\backslash\circ,\circ\supset\backslash A$, were to equal $\backslash A$, then a contradiction would follow from the equation $\supset\backslash\circ,\circ\supset\backslash A=\supset\backslash A$. By corollaries to Theorems 65 and 26, $\supset\backslash\circ,\circ\supset\backslash A=,\circ\supset\backslash A$. But according to Theorem 75, no array equals its solitary.

11. Positional

The positional transform of a dyadic operation applies the operation to the positionally corresponding items of two arrays. Mote operations are the same as their replacement or positional transforms.

To each dyadic operation Δ there corresponds by the *positional operator* [9a], which is also denoted by a colon, another dyadic operation $:\Delta$ called *positional* Δ or the *positional transform* of Δ. For all arrays A and B, each item in the array $A:\Delta\ B$ is the result of applying Δ to the *positionally corresponding* items in A and B. The positional operator is based on the treatment of dyadic

scalar functions in APL. For example, for all arrays P, Q, R, S, T, and U,

$P\ Q\ R{:}\Delta\ S\ T\ U = (P\ \Delta\ S)\ (Q\ \Delta\ T)\ (R\ \Delta\ U)$

$P\ Q\ R{:}\Delta\circ S = (P\ \Delta\ S)\ (Q\ \Delta\ S)\ (R\ \Delta\ S)$

$\circ P{:}\Delta\ S\ T\ U = (P\ \Delta\ S)\ (P\ \Delta\ T)\ (P\ \Delta\ U)$

$\circ P{:}\Delta\circ S = \circ.P\ \Delta\ S$

$P\ Q{:}\Delta\ S\ T\ U = (P\ \Delta\ S)\ (Q\ \Delta\ T)$

$P\ Q\ R{:}\Delta\backslash\circ S = \backslash\circ.P\ \Delta\supset\backslash\circ S$

$\backslash\circ P{:}\Delta\backslash\circ S = \backslash\circ.\supset\backslash\circ P\ \Delta\supset\backslash\circ S.$

Items at the same indices in lists of the same length correspond positionally. A *singular* array has one item. Two lists of different lengths are made to have the same length by recounting a singular list to the length of the other list or truncating the longer nonsingular list to the length of the shorter nonsingular list. The items in a recounted or truncated list then correspond positionally to the items at the same indices in the other list. This method of establishing positional correspondence extends to arbitrary arrays.

If arrays A and B have the same shape, then an item of A corresponds positionally to an item of B if and only if the two items have the same address. A *singular axis* has length `1`. If two arrays have different numbers of axes, then enough singular axes are appended to the array having the lesser number of axes to make the arrays have the same number of axes. The two arrays are then made to have the same shape by (i) replicating each array along its singular axes so as to match the lengths of the corresponding axes in the other array, and (ii) truncating to the lengths of the shorter axes each array having longer nonsingular axes corresponding to shorter nonsingular axes in the other array.

For example, if an array A has shape `4 1 3 1 5 1 7 8 0`, then the 0-axis of A has length `4`, the 1-axis has length `1`, etc. If an array B has shape `4 2 6 1 0 0 1`, then $A{:}\Delta\ B$ has shape `4 2 3 1 0 0 7 8 0`. The positional operator is defined by an equation discussed in Section 21.

If several arrays are combined by various positional transforms, then regardless of the order in which the arrays are combined, or the order in which the transforms act, or the number of times an array appears as an argument of the transforms, the final value has as many axes as the argument having the greatest number of axes. An axis in the final value is singular if all corresponding axes of the arrays are singular. A nonsingular axis in the final value is as long as the corresponding shortest, nonsingular axis among the arrays.

The positional transforms of commutative operations are commutative, those of associative operations are associative, and those of operations distributive one through another are distributive one through another.

If nothing is gained by distinguishing an operation from its replacement or positional transform, as is the case for most arithmetic and logical operations, then, as suggested by APL, the operation becomes more useful if it is identified with its transform. A monadic operation Δ is said to be a *mote operation* if and only if $\Delta\ A{=}{:}\Delta\ A$ for every array A. A dyadic operation Δ is mote if and only if $A\ \Delta\ B{=}.A{:}\Delta\ B$ for all A and B.

Mote operations applied to motes return motes as values. If Δ is a monadic mote operation and A is a mote, then $\Delta\ A$ equals $:\Delta\circ A$ equals $\circ\Delta\ A$. If Δ is a dyadic mote operation and B is also a mote, then $A\ \Delta\ B$ equals $\circ A{:}\Delta\circ B$ equals $\circ(A\ \Delta\ B)$. The arrays $\Delta\ A$ and $A\ \Delta\ B$ are motes because they equal their singles.

Monadic negation and dyadic addition are defined to be mote operations by postulating that $-A{=}{:}-A$ and $A{+}B{=}.A{:}{+}B$ for all A and B. A mote operation works down through the levels of an array until it reaches the most deeply nested motes. The operation goes no deeper because it starts returning motes as values.

The addition of `0` to an array returns the same array except for replacing every nonnumerical mote by a cipher. For example, the pairs in the following four lines are equal:

```
0 + ((2 3) (4 'A')) (5 (⊡ 'BC'))
(0 + (2 3) (4 'A')) (0 + 5 (⊡ 'BC'))
((0 + 2 3) (0 + 4 'A')) ((0 + 5) (0 + ⊡ 'BC'))
((2 3) (4 ⊡)) (5 (⊡ (⊡ ⊡))).
```

A sum of two motes equals the cipher if at least one of the motes is nonnumerical. Hence the sum of a list of two or more arrays is devoid of characters. As suggested by the preceding example, $+\circ A = .\ A{+}0$ for every A. Thus $+\backslash A$ equals $+\circ\supset\backslash A$ equals $\supset\backslash A{+}0$. Similarly, for every A, $\times\circ A = .\ A{\times}1$ and $\times\backslash A = .\ \supset\backslash A{+}1$. Sums and products of arrays are devoid of characters.

12. Logic

Mote operations for the algebra of propositions are defined in terms of the primitive mote operations of logical denial and summation.

Logical *denial* is a primitive operation that complements the truth-values `0` and `1` and sends every other mote to the cipher. Denial is defined for all arrays by postulating that it is a mote operation. $\neg A{=}{:}\neg A$ for every array A. For example, `¬ 0 1 (2 1 ⊡)` equals `1 0 (⊡ 0 ⊡)`.

Since ¬ is a mote operation, $\neg\backslash A$ equals $:\neg\backslash A$, which

equals $\circ\neg\supset\backslash A$ by Theorem 91. By a corollary to Theorem 26, $\backslash\circ B=\backslash\circ\supset\backslash\circ B$. Hence $\neg\backslash A$ equals $\backslash\circ\supset\backslash\circ\neg\supset\backslash A$. If the pattern of A is devoid of blanks, then $\supset\backslash\circ\neg\supset\backslash A$ equals $\supset\backslash\circ\supset\backslash A$, which equals $\supset\backslash A$ by another corollary of Theorem 26. Thus if the pattern of A is devoid of blanks, then $\neg\backslash A$ equals $\backslash\circ\supset\backslash A$, which equals $\backslash A$ by Theorem 26. Exactly the same result and proof holds for arithmetic negation in place of logical denial. Both $\neg\Theta$ and $-\Theta$ equal Θ.

An array is *simple* if all of its items are motes. Every item of a *Boolean array* is a truth-value. The *logical sum* $\vee A$ of a Boolean array A is 0 if and only if every item of A is 0. $\vee A=\vee,A$. In particular, an empty Boolean array has a logical sum of 0. A Boolean array in which there exists a 1 has a logical sum of 1. Every other simple array has the cipher as logical sum. Logical products are defined by De Morgan's law. Disjunction is defined as the logical sum of a pair:

$\wedge A$ for $\neg\vee\neg A$

$A\vee B$ for $\vee.A,B$.

The preceding definitions extend to nonsimple arrays if disjunction is assumed to be a mote operation. For all A and B, $A\vee B=.A:\vee B$. Conjunction $\wedge$, implication $\rightarrow$, and coimplication $\equiv$ are defined in terms of denial and disjunction by the usual propositional formulas [35]. Thus $A\rightarrow B=.\neg A\vee B$ for all arrays A and B. The logical connectives '$\neg$', '$\vee$', '$\wedge$', '$\rightarrow$', and '$\equiv$' are the same as those used by Heyting [41] and Kuratowski and Mostowski [19]. The propositional operations of nor $\barwedge$, nand $\barvee$, logical subtraction $\neg$, and exclusive or $\not\equiv$ are defined by the denials of $\vee$, $\wedge$, $\rightarrow$, and $\equiv$, respectively.

By analogy with sums and products,

$\vee\circ A =. A\vee 0$ $\qquad$ $\vee\backslash A = \vee\circ\supset\backslash A$

$\wedge\circ A =. A\wedge 1$ $\qquad$ $\wedge\backslash A =. \supset\backslash A\vee 1$

for every array A. In particular, $\vee\Theta=0$ and $\wedge\Theta=1$.

The usual equations [42] for Boolean algebras ($\vee$, $\wedge$, monadic $\neg$), classical subtractive lattices ($\vee$, $\wedge$, dyadic $\neg$), Boolean rings ($\not\equiv$, $\wedge$), classical implicative lattices ($\wedge$, $\vee$, $\rightarrow$), and dual Boolean rings ($\equiv$, $\vee$) holds for all arrays when converted to balanced form. An equation is unbalanced if a variable has occurrences on one side of the equation but not the other. The valid equation $A\rightarrow A=.B\rightarrow B$ of propositional algebra fails for arrays because it is unbalanced. An equation is balanced by entering variables tautologously on the deficient sides. For example, $B\rightarrow B\wedge A$ is logically equivalent to A but involves the shape of B. Thus $B\rightarrow B\wedge A\rightarrow A =.A\rightarrow.B\rightarrow B$ is balanced and holds for all A and B.

If A is the quadruple 1 0 ((1 1 0) 2) 3 and B is the triple 0 (1 (0 4)) ((0 1) 5 (0 1)), then $A\rightarrow B$ equals 0 (1 (1 ⊡)) ((0 1) ⊡). Further calculation shows that both $B\rightarrow.A\rightarrow B$ and $A\rightarrow.B\rightarrow B$ equal the triple 1 (1 (1 ⊡)) ((1 1) ⊡). Thus $B\rightarrow.A\rightarrow B$ equals the obvious tautology $A\rightarrow.B\rightarrow B$.

A formula is a term or well-formed string that denotes a truth-value. If f and g are formulas, then $f=g$ and $f\equiv g$ have the same interpretation. The proofs of Theorems 1 and 2 hinge on the equivalence of equality with coimplication when restricted to truth-values.

13. Sets

An array is called a set if the ordering and repetition of its items are neglected. Membership and inclusion are defined in terms of equality, replacement, and operational abstraction.

Universal and existential quantification over the items of an array involve bound variables that can be treated in array theory by operational or functional abstraction [6,43]. In the notation of Church [34] $\lambda Z(Z\times Z)$ is the operation of squaring, $\lambda Z(Z)$ is the neutral or identity operation, and $\lambda Z(\Theta)$ is a constant operation that returns the null as value for every array as argument. If the preceding monadic operations are applied to an array A as argument, then $\lambda Z(Z\times Z)A$ equals $A\times A$, $\lambda Z(Z)A$ equals A, and $\lambda Z(\Theta)A$ equals Θ.

A *term* is a well-formed string that denotes an array. A *formula* is a term that denotes a truth-value. If w is a variable and t is a term, then $\lambda w(t)$ is a prefix designating a monadic operation. If w is a variable and t is a formula, then $\exists w(t)$ and $\forall w(t)$ are formulas. The strings λw, $\exists w$, and $\forall w$ are called *binding prefixes*. In each case, t is the *scope* of the binding prefix.

A given occurrence of a variable w in a term is *bound* if and only if it occurs either in a binding prefix or in the scope of a binding prefix having an occurrence of w. Otherwise the given occurrence of w is *free*. The *free* [*bound*] *variables* of a term are the variables having at least one free [bound] occurrence in the term.

If s and t are terms and w is a variable, then $\lambda w(t)$ (s) is a term, which may be written as $\lambda w(t)s$ or $\lambda w(t).s$ according to the syntax of monadic operations. If the bound variables of t are distinct both from w and from the free variables of s, then the array denoted by $\lambda w(t)$ (s) is denoted also by the term got by substituting (s) for every occurrence of w in t [34].

If w is a variable, t is a term, and Δ is a prefix having no free occurrences of w, then $\lambda w(\Delta(t))$ and $\Delta\lambda w(t)$ designate the same monadic operation. If t is the variable w, then $\lambda w(\Delta\ w)$ and Δ designate the same operation.

The occurrence of arrays as items in other arrays

defines an operation $\in$ of *membership* that corresponds to the membership relation in the theory of classes. $A \in B$ if and only if A equals at least one of the items of B.

$A \in B$ for $\vee{:}\lambda Z(A=Z)B$

$A \notin B$ for $\neg.A \in B.$

Thus $A \in B = \vee.\circ A{:}=B$. This last equation is not used to define membership because the positional operator is not primitive.

If B is the triple $P\ A\ Q$, then $:\lambda Z(A=Z)B$ equals

$(\lambda Z(A=Z)P)\ (\lambda Z(A=Z)A)\ (\lambda Z(A=Z)Q),$

which equals $(A=P)\ (A=A)\ (A=Q)$. Thus $A \in B$ equals $\vee\ 0\ 1\ 0$, which is 1. Hence $A \in P\ A\ Q$.

If B is an empty array and Δ is the operation $\lambda Z(A=Z)$, then $A \in B$ equals $\vee{:}\Delta\ B$ equals $\vee{,}{:}\Delta\ B$ equals $\vee{:}\Delta{,}B$ equals $\vee{:}\Delta\backslash B$ equals $\vee\backslash\circ\Delta\supset\backslash B$ equals $\vee\backslash\circ\lambda Z(A=Z)\supset\backslash B$ equals $\vee\backslash\circ.A=\supset\backslash B$ equals $\vee\backslash.A=\supset\backslash B$ equals $\vee\Theta$ equals 0. Hence $\#B=0 \rightarrow .\ A \notin B$.

If $A \in B$, then A *belongs* to B, B *contains* A, and A is said to be an *element* or *member* of B. The words "element" and "member" connote a disregard of order and repetition.

An array is interpreted as a class or set and is called a *set* if the ordering and repetitions of the items in the array are neglected in the interpretation. For example, $A \in B$ if and only if the array A belongs to the set B. Every set is an array. In discussions outside array theory, unordered collections are called *classes*.

If every member of set A is a member of set B, then $A \subset B$.

$A \subset B$ for $\wedge{:}\lambda Z(Z \in B)A.$

Thus $A \subset B = \wedge.A{:}\in\circ B$. For example, $P\ Q\ Q \subset Q\ R\ P\ S$. By an argument almost identical to the proof that no array belongs to an empty set, one can prove that an empty set is a subset of every set:

$\#A=0 \rightarrow .\ A \subset B.$

The inclusion relation $\subset$ is transitive and reflexive on the universe of arrays and supports the definition of a concomitant equivalence relation $\sim$, called *likeness*:

$A \sim B$ for $A \subset B\ \wedge.\ B \subset A.$

Two sets are *alike* if and only if each is included in the other. Array theory or list theory reduces to a system of sets under the equivalence relation of likeness. Many of the equations in the theory of classes hold also in array theory or list theory if '=' is replaced by '$\sim$'. Likeness of sets differs from equality of classes in that the members of like sets must be identical rather than merely alike.

The definitions of membership and inclusion indicate that the following equations hold for every A provided that a formula is substituted for the ellipsis:

$\vee{:}\lambda Z(\ldots)A = \exists Z(Z \in A\ \wedge\ (\ldots))$

$\wedge{:}\lambda Z(\ldots)A = \forall Z(Z \in A \rightarrow (\ldots)).$

If $:\Delta\ A \subset 0\ 1$, then Δ is a *monadic predicate on* A in the sense that Δ returns a truth-value for every item of A. If a formula, such as '0', '1', '$A=Z$', or '$Z \in B$', is substituted for the ellipsis, then $\lambda Z(\ldots)$ is a monadic predicate on every array.

14. Separation

The separation transform of a monadic operation separates a sublist from an array by sublisting the array according to its value under the replacement transform of the operation.

Zermelo's Axiom III of *separation* postulates that if a propositional function Δ is definite for all elements of a class M, then M possesses a subclass containing as elements precisely those elements x of M for which $\Delta\ x$ is true [36]. If M is known to be a class, then so is the aggregate $\{x \in M\ [\!]\ \ldots\}$ of all elements x in M for which a formula (in place of the ellipsis) holds.

In the theory of classes, the axiom schema of separation is a corollary of the axiom schema of replacement. Those elements of a class M that are to be separated from M or kept in a subclass of M are sent to themselves by a function Δ. All other elements of M are sent by Δ to a class N not in M. The desired subclass is obtained by deleting N from the range of Δ. If this construction is used to separate a subset from an array A, then there are as many copies of N in $:\Delta\ A$ as there are unwanted items in A. Separation follows less directly from replacement in array theory, because all occurrences of N in $:\Delta\ A$ have to be found and then deleted from $:\Delta\ A$.

The process of selecting some of the elements of a class but not others suggests that a list of truth-values be used to indicate which items are to be taken from a list of the same length. For example,

$1\ 1\ 0\ 1\ 0\ /\ 0\ 1\ 2\ 3\ 4 = 0\ 1\ 3.$

This is the APL function for compressing component vectors. A similar operation, which is described below, can be defined for all arrays. The axiomatization is deferred to Part II.

The *sublisting* of an array B by an array A is the list A/B in main order of those items in the list of B that occur at indices addressing items equal to 1 in $\#B\rho A$. Sublisting is primitive. For all arrays A and B,

$A/B = {,}.A/B$

$A/B =. \#B\rho A/{,}B$

$A/B =. :\lambda Z(Z=1)A/B.$

For example, 0 1 2/B lists every third item of B beginning with the second. If B is the octuple $P\ Q\ R\ S\ T\ U\ V\ W$, then 0 1 2/B equals $Q\ T\ W$. For all arrays A and B, the list A/B is called a *sublist* of B.

The following two properties of sublisting are analogous to Axioms 32 and 33. For all arrays A and B and for every monadic operation Δ,

$$1\in A\ \wedge.\ \#B\neq 0\ .\rightarrow.\ :\Delta(A/B)=.A/:\Delta\ B$$

$$1\notin A\ \rightarrow.\ :\Delta(A/B)=:\Delta\backslash B.$$

Thus if A or B are empty, then $:\Delta(A/B)=:\Delta\backslash B$. For every A, $\Theta/A=\backslash A$ and $0/A=\backslash A$ and $1/A=,A$.

If Δ is a monadic operation, then $:\Delta\ A/A$ lists the items of A that Δ sends to 1. For example, $:\#A/A$ lists the singular items of A. $\neg:\#A/A$, which equals $:\neg:\#A/A$ and $:(\neg\#)A/A$, lists the empty items of A. If Δ is a monadic predicate on A, then $:\Delta\ A/A$ is the list of all arrays X in A for which $\Delta\ X$ is true. This array-theoretic counterpart to the axiom of separation suggests the following definition of the *list abstraction* or *separation operator* /.

$$/\Delta\ A \quad \text{for} \quad :\Delta\ A/A.$$

The separation transform $/\Delta$ of a monadic operation Δ is a monadic operation that deletes from the list $,A$ of an array A each item X for which $\Delta\ X\neq 1$. Thus $/\Delta\ A$ corresponds to $\{X\in A\ [\!]\ \Delta\ X\}$. If a formula is substituted for the ellipsis, then for every A, $/\lambda Z(\ldots)A$ corresponds to $\{Z\in A\ [\!]\ \ldots\}$.

The preceding correspondence suggests that intersection and set difference be defined as in the theory of classes:

$$A\cap B \quad \text{for} \quad /\lambda Z(Z\in B)A$$

$$A\sim B \quad \text{for} \quad /\lambda Z(Z\notin B)A.$$

The intersection of an array A with a set B is the list $A\cap B$ in main order of the items in A that belong to B. For example, $P\ Q\ R\ Q\ \cap\ R\ S\ Q$ equals $Q\ R\ Q$.

Operational abstraction can be used to define operations that combine particular items in their arguments. For example, the monadic operation $\lambda X\ Y\ Z(\ldots)$ applied to an array A assigns values to the bound variables X, Y, and Z by letting the triple $X\ Y\ Z$ equal $3\rho A$. The value of $\lambda X\ Y\ Z(\ldots)A$ is determined by combining the 0-item X, the 1-item Y, and the 2-item Z of $3\rho A$ according to a term substituted for the ellipsis. If A is an array of triples, then $/\lambda X\ Y\ Z(X\in Y\wedge.X\subset Z)A$ is the list of all triples $X\ Y\ Z$ in A for which X is both a member of Y and a subset of Z.

Operational abstraction can also be used to define dyadic operations and dyadic operations that combine particular items in their arguments. For example, $\lambda X.Y(\ldots)$ is a dyadic operation and $\lambda V\ W.X\ Y\ Z(\ldots)$ is a dyadic operation in which V and W are taken cyclically as the first two items of the left argument and X, Y, and Z are taken cyclically as the first three items of the right argument.

15. Powers

The power array of a finite array holds all of the finitely many sublists of the array.

Zermelo's Axiom IV of the *power class* assumes that to every class M there corresponds another class, called the power class of M, that contains precisely all the subclasses of M. Similarly, in array theory, to every finite array A there corresponds another array $\underline{*}A$, called the *power array* of A, that holds precisely all the sublists of A. In list theory, the list of $\underline{*}A$ is the *power list* of A.

$$\underline{*}A \quad \text{for} \quad :\lambda Z(Z/A)\underline{\times}.\#A\rho\circ 0\ 1.$$

The definition of $\underline{*}A$ is perhaps best explained by an example. If A has two items, then $2\rho\circ 0\ 1$ equals (0 1) (0 1) and $\underline{\times}.\#A\rho\circ 0\ 1$ equals a 2 by 2 array holding (0 0) (0 1) (1 0) (1 1) in main order. The operation $\lambda Z(Z/A)$ causes its argument to sublist A. Thus $\underline{*}A$ is a 2 by 2 array in which each item is a sublist of A got by using a Boolean list of length 2 to sublist A.

$$,\underline{*}P\ Q = (\backslash\circ P)\ (,\circ Q)\ (,\circ P)\ (P\ Q).$$

All arrays have finitely many axes. Axis lengths, and therefore indices, are restricted to countable (denumerable) ordinal numbers [38]. Thus every array is *countable* in the sense that its items can be put in a list indexed by the finite ordinals 0, 1, 2, ... A *finite* array holds finitely many items and has axes of finite lengths. Since Cartesian products take only finite arrays as arguments, power arrays hold only finitely many finite lists as items. The continuum of all infinite Boolean sequences cannot be constructed (Section 17).

16. Unions

The union of an array holds all the items of items of the array. Dyadic unions and monadic intersections are defined in terms of monadic unions.

Zermelo's Axiom V of the *union* assumes that to every class M there corresponds another class, called the union of M, that contains precisely all the elements of members of M. The items of the items of an array are called *components*. The *union* $\cup A$ of an array A is a list holding precisely all the components of A. $\cup A$ holds the items in main order of the first item of A followed by the items in main order of the second item of A, etc. For

example, the union of the quadruple $(P\ Q)\ (\circ R)\ (\backslash\circ S)\ (Q\ P\ T)$ equals $P\ Q\ R\ Q\ P\ T$.

The union $\cup A$ of A equals $,\cup A$ and $\cup,A$ and $\cup:,A$. Both $\cup\circ A$ and $\cup:\circ A$ equal $,A$. Both $\cup\backslash A$ and $\cup:\backslash A$ equal $\backslash\supset A$. The count $\#\cup A$ of a union equals the sum $+:\#A$ of the counts. $:\Delta\cup A=\cup::\Delta\ A$ for every monadic operation Δ. The generalized associative law holds for unions, intersections, sums, products, etc., in the sense that $\cup:\cup A=\cup\cup A$ and $\cap:\cap A=\cap\cup A$ and $+:+A=+\cup A$ and $\times:\times A=\times\cup A$. The preceding equalities hold for every array A.

The *union* $A\cup B$ of an array A with an array B is the list holding the items of A in main order followed by the items of B in main order. Dyadic union is defined as the monadic union of a pair:

$A\cup B$ for $\cup.A;B$.

For example, $P\ Q\ R\ \cup\ Q\ S$ equals $P\ Q\ R\ Q\ S$. If B is empty, then $A\cup B=,A$. Thus $\backslash A\cup\backslash B=\backslash A$. If B is nonempty and A is empty, then $A\cup B=,B$. The operation of uniting treats its arguments as lists. $A\cup B=.,A\cup,B$. As in the monadic case, $\#(A\cup B)=.\#A+\#B$.

The union of arrays may be interpreted as the union of sets under the equivalence relation of likeness. If A and B are nonempty and differ, then $A\cup B$ and $B\cup A$ are alike but unequal.

The *link* A,B of an array A with an array B is the list holding the items of A in main order followed by B itself:

A,B for $A\cup\circ B$.

For example, $P;Q,R,S = P\ Q\ R\ S$. If F is a monadic operation, X is a mote, and Y and Z are arbitrary arrays, then $F(X,Y,Z)$ is the value returned by F from the triple X,Y,Z taken as argument. If A is empty, then $A,B=,\circ B$.

The intersection $\cap A$ of an array A is the sublist of those items in $\supset A\cap\cup A$ that belong to every item of A. If A is nonempty, then $\supset A\cap\cup A=\supset A$. Thus $\cap(A;B)=.A\cap B$. If A is empty, then both $\supset A\cap\cup A$ and $\cap A$ equal $\backslash\supset A$. The class-theoretic definition of intersection guides the array-theoretic definition.

$\cap A$ for $\{Z\in\cup A\ [\!]\ \forall B(B\in A\rightarrow.\ Z\in B)\}$

$\cap A$ for $/\lambda Z(\wedge:\lambda B(Z\in B)A).\supset A\cap\cup A$.

17. Choice

Since every array is well-ordered, the axiom of choice is implicit in the theory. Cartesian products are defined only for finite arrays.

Zermelo's Axiom VI of *choice* postulates that if M is a disjointed (pairwise disjoint) class of nonempty classes, then the union of M includes at least one subclass having one and only one element in common with each element of M. In the general principle of choice, which Zermelo deduced from Axiom VI [36], the class M need not be disjointed. Zermelo (1904) used the principle of choice to prove the well-ordering theorem, which asserts that for any class there exists a well-ordered class having the same members [44]. The well-ordering theorem, in turn, implies the axiom of choice [45].

The axiom of choice is implicit in array theory because every array is well-ordered. Every nonempty sublist of an array holds a first item.

The axiom of choice is also equivalent to Russell's (1906) multiplicative principle: if M is a disjointed class of nonempty classes, then the Cartesian product of M is nonempty. Unlike the Cartesian product in array theory, the Cartesian product used in the multiplicative principle is not ordered.

Let $1W$ be the count of the infinite list $0\ 1\ 2\ \ldots$ of the finite ordinals. If A is the list $1W\rho\circ 0\ 1$, which equals $(0\ 1)\ (0\ 1)\ (0\ 1)\ \ldots$, then the unordered, class-theoretic Cartesian product of A contains every infinite Boolean sequence. Cantor's diagonal method shows that this Cartesian product cannot be made into a list of length $1W$. In general, there appears to be no appropriate way of ordering the elements of a Cartesian product of an infinite array [45]. Thus in array theory, Cartesian products are defined only for finite arrays.

If A is finite, then the count of the Cartesian product of A equals the product of the counts of the items in A. $\#A<1W\rightarrow.\ \#\times A=\times:\#A$. If A equals $1W\rho\circ 0\ 1$, then $\times A$ is undefined and $\#\times A$ is certainly not $1W$. However, $\times:\#A$ equals $\times 2\ 2\ 2\ \ldots$, which equals the limit $1W$ of the sequence of ordinals $1,\ 2,\ 4,\ 8,\ \ldots$ [46]. The restriction to finite A cannot be omitted.

18. Infinity

Given the existence of an infinite list, one can construct arrays having finitely many axes of various transfinite countable ordinal lengths.

Zermelo's Axiom VII of *infinity* postulates the existence of a class M that contains the empty class and the unit class of every element in M. Similarly, in array theory, there exists a list $\Theta\ (\circ\Theta)\ (\circ\circ\Theta)\ \ldots$ in which the null is the first item and each item is followed by its single. The count of this list is $1W$, which is the first transfinite ordinal number.

In APL, $\iota 4$ equals $0\ 1\ 2\ 3$ in 0-origin indexing. Let $\iota\#A$ be the list in ascending order of all ordinal numbers less than $\#A$. Thus $\iota\#A$ is the array-theoretic counterpart of the class-theoretic definition of an ordinal num-

ber. ⍳1W lists the finite ordinal numbers, which serve also as the nonnegative integers.

The lists 0 1 2 3 ... and 3 4 5 6 ... have the same count 1W. The first list equals the union of 0 1 2 with the second list. Since the count of a union equals the sum of the counts, 3+1W=1W. The union of 3 4 5 6 ... with (followed by) 0 1 2 has count 1W3, which equals 1W+3 and differs from 1W. Ordinal addition is not commutative.

Array theory uses a nonstandard notation for the transfinite ordinals to accommodate the conflicting demands of different disciplines. The notation is based on a lexicographic ordering for multiplication rather than the usual antilexicographic ordering [19]. 2×1W=2W. The product #A×#B of the *multiplier* #A with the *multiplicand* #B equals the sum +.#A⍴#B of #A repetitions of #B. The theory follows Cantor's [18] original convention in placing the multiplier to the left of the multiplicand. Modern notation follows Cantor's later reversal of the convention [45]. Ordinal multiplication is not commutative.

The exponentiation #A*#B of the ordinal #A raised to the ordinal power #B equals the product ×.#B⍴#A of #B repetitions of #A. The convention of lexicographic ordering for multiplication requires that successively applied exponents be multiplied in reverse order.

#A*#B*#C =. #A*.#C×#B.

Table 1 shows how the list of the first 1W*3 ordinals can be visualized as the main list of an array having three axes, each of length 1W. All ordinal numbers in the left-hand column of the table are *limit ordinals*, which have no direct predecessor.

The ordinal 2W may also be written as '0W2W0', which indicates that 2W is located in layer 0, row 2, and column 0 of the array. 1WW equals both 1W*2 and 1W0W0. The operations of counting, uniting, and reshaping construct from the list ⍳1W a class of countable, transfinite ordinal numbers. For example, 1WW equals #∪.1W⍴∘⍳1W.

19. Reduction

The reduction transform of a dyadic operation is a monadic operation defined for finite, nonempty, nonsingular arrays. The fold transform of a monadic operation is dyadic.

Reductions involving scalar functions in APL suggest the definition of a *reduction operator* ⌿ that transforms an arbitrary dyadic operation Δ into a monadic operation ⌿Δ called the *reduction transform* of Δ. A reduction transform depends only on the list of its arguments: ⌿Δ A=⌿Δ,A for every A. For every dyadic operation Δ and all arrays A, B, and C,

⌿Δ(B⍪C) =. B Δ C

3≤#A ∧. #A<1W .→. ⌿Δ A =. ⊃A Δ ⌿Δ.1↓,A.

For example, if A = P Q R, then ⊃A=P and 1↓,A=Q R. Hence ⌿Δ A equals P Δ.Q Δ R.

Sums +A, products ×A, logical sums ∨A, logical products ∧A, unions ∪A, and intersections ∩A are defined for all arrays. If Δ is one of the six preceding monadic-dyadic operations, then Δ(B⍪C)=.B Δ C for all B and C. Further, if A is nonempty, nonsingular, and finite, then Δ A=⌿Δ A. ⍁A is not the same as ⌿⍁A, although ⍁(B⍪C)=.B⍁C. For example, ⍁P Q R is an array of triples, whereas ⌿⍁P Q R equals P⍁.Q⍁R, which is an array of pairs.

If an array A is infinite or holds less than two items, then there appears to be no uniform way of determining ⌿Δ A for any given Δ. For example, if A is a finite Boolean list, then ⌿=A equals 1 or 0 according as A holds an even or odd number of zeros. Is the number of zeros in an infinite list even or odd? How does one define ⌿Δ∘A uniformly for arbitrary Δ if +∘A=.A+0 and ∪∘A=,A? What is ⌿Δ\A in general if +\A=+∘⊃\A and ∪\A=\⊃A? These questions suggest that reduction transforms should be left undefined for empty, singular, and infinite arrays. For such arrays, monadic union differs from the reduction transform of dyadic union.

The reduction operator transforms a dyadic operation

Table 1 The ordinals less than 1W×1W×1W.

		2 ⟶ 1W			
0	1				
	↓	0	1	2	...
		1W	1W1	1W2	...
		2W	2W1	2W2	...
	1W	...	...	...	...
		1WW	1WW1	1WW2	...
		1W1W	1W1W1	1W1W2	...
		1W2W	1W2W1	1W2W2	...
		...	...	...	...
		2WW	2WW1	2WW2	...
		2W1W	2W1W1	2W1W2	...
1W		2W2W	2W2W1	2W2W2	...
		...	...	...	...
		...	...	...	...

into a monadic operation. Monadic operations can also be transformed into dyadic operations. The *fold* [47] operator ⍁ transforms an arbitrary monadic operation Δ into a dyadic operation ⍺⍁Δ that takes a finite ordinal number as left argument and any array as right argument. (The fold operator in [47] results in a monadic operation.) The left argument indicates the number of times Δ is to be applied to the right argument. A fold transform, like any other dyadic operation, is subject to operators. For example,

```
0 1 2 3 :⍁∘∘A = A (∘A) (∘∘A) (∘∘∘A).
```

20. Arrays

Singles are 0-valent arrays. Singles and nonsingular lists are suits. Shapes and addresses are suits of lengths and indices. The null is the shape and address for all singles.

In comparison with array theory, list theory has the advantage of confining addresses to the simplest form. Although an array on more than one axis can be represented by the nesting of lists within a list, the process of fetching an item of the array, as represented in the list of lists, is complicated by considerations of both counting and nesting. If a uniform array is represented by a uniform list, there is no information available in the list to indicate the depth of nesting required for the representation of the axes of the array.

Matrix algebra, tensor algebra, and APL illustrate the convenience of locating the items of an array by addresses consisting of two or more indices. If the universe of arrays is to be closed under the tensor operations of outer multiplication, contraction, and inner multiplication, then the universe must contain arrays having any finite number of axes.

Let V be an n-dimensional vector space over a field. Disregarding considerations of contravariance and covariance, a tensor of valence q on V is a multilinear mapping of the Cartesian product of the list $V\ V\ \ldots\ V$ of length q into a vector space [48]. If V has a base, then a tensor of valence q on V can be represented by a *component tensor*, which is an array on q axes, each of length n. The items in a component tensor are scalars belonging to the field. The scalars may be pairs of real numbers representing complex numbers.

The number of items in a component tensor is determined by raising the dimension of the underlying space to the power of the valence. If this rule is to hold for all valences and all dimensions, then every component tensor of valence 0 must hold exactly one item. For example, if the dimension is 3, then 0-, 1-, 2-, and 3-valent component tensors have 1, 3, 9, and 27 items, respectively. If the dimension is 0, then 0-, 1-, 2-, and 3-valent component tensors have 1, 0, 0, and 0 items. Component tensors of nonzero valence over a space of zero dimension are empty.

The inner product of two component tensors of valence 1 is a component tensor of valence 0 holding just one scalar. Since an inner product of component vectors produces a scalar in both vector and matrix algebra, tensor algebra usually identifies a 0-valent component tensor with the scalar it holds.

A mote is an array that holds itself as sole item. Complex numbers, which are scalars in a field, and real numbers, which are scalars in a subfield, are appended to array theory as motes having 0 as pattern. The preceding considerations from tensor algebra suggest that motes should be 0-valent arrays. Since motes are singles that hold themselves, the operation of singling is now defined to return 0-valent arrays as values. All arrays on no axes, such as ordinal numbers and characters, are called *singles*.

All singles and all nonsingular lists are called *suits*. A suit of no items is an empty list. A suit of one item is the single of the item. A suit of two or more items is the list of the two or more items. Every nonsingular suit is a nonsingular list and vice versa. Singular suits are singles. Singular lists are solitaries. A *string* is a suit of characters. Thus a string holding one character is the character itself.

The *shape* $\sim A$ of an array A is a finite suit holding the lengths of the axes for A. The 0-item in a nonempty shape is the length of the 0-axis, the 1-item is the length of the 1-axis, etc. The shape of an array on one axis is a suit holding one length. Thus the shape of a list is the axis length itself. The shape of an array on no axes is a suit holding no lengths. Thus the shape of a single is the null ⍬, which is the empty list or suit with pattern 0.

Axiom 5. The shape of zero is null.

```
~0 = ⍬.
```

Definition 1. The count of an array is the product of its shape.

```
#A     for     ×~A.
```

An *address* for an array is a finite suit of indices. An address for an array on one axis is a suit holding one index. Thus an address for a list is an index. An address for an array on no axes is a suit holding no indices. The null is the only empty suit of pattern 0. Thus the null is the unique address for a single [9a]. Unlike APL\360, the item in a number or character has the null as address. Since the address for a single is unique, a single holds precisely one item, which is located at ⍬. The axioms and theorems may now be interpreted in terms of arrays.

21. Cartesian products

Outer transforms and Cartesian arrays are defined in terms of Cartesian products. Boxing, exponentiating, and positional transforms are defined in terms of outer transforms.

The diagram in Fig. 1 represents a *table* or 2-valent array A in which B is a list of length 7, C is a single, D is a table of shape 9 5, and E is a solitary. Component T of A is at address 0 1 in D, which is at 1 0 in A. Figure 2 represents the Cartesian product of A.

The figures show various aspects of a diagrammatic method for representing arrays [9a]. The tail, body, point, and head of an arrow show, respectively, the axis number, axis label, direction of increasing indices, and length of an axis. Some of the layers, rows, or columns normal to a long axis may be omitted. Only those cells are filled that are needed to illustrate a point. Some of the hyperplanes (rows, layers, etc.) may be indexed explicitly.

The axes labeled 0, 1, 2, 3 in the four diagrams for the items of A in Fig. 1 become the axes numbered 0, 1, 2, 3 in the diagram for $\underline{\times}A$ in Fig. 2. Axes I and J in Fig. 1 become axes I and J in Fig. 2.

An array S is said to be a *section* of an array A if A and S have the same shape and if the first item of S is an item of the first item of A, the second item of S is an item of the second item of A, and so on in main order through all the items of A and S. The items of S are components of A.

The *Cartesian product* $\underline{\times}A$ of a finite array A is an array of size $\cup:\sim A$ holding every possible section of A. For example, if A is the pair $B;C$ and $I\supset B$ is an item of B at address I and $J\supset C$ is an item of C at address J, then $I\supset B;.J\supset C$ is the section of A that occurs in $\underline{\times}A$ at address $I\cup J$. If A is infinite, then $\underline{\times}A=\boxed{\circ}$.

If $\underline{\times}A$ is nonempty, then $\supset\underline{\times}A=:\supset A$. A finite array has the same components as its Cartesian product in the sense that $\cup A$ and $\cup\underline{\times}A$ are alike. $::\Delta\underline{\times}A=\underline{\times}::\Delta\ A$ for every monadic operation Δ. The dyadic Cartesian product $A\underline{\times}B$ is defined to be the monadic Cartesian product $\underline{\times}(A;B)$ of a pair.

To each dyadic operation Δ there corresponds by the outer operator, which is designated by a macron, another dyadic operation $\bar{\Delta}$ called *outer* Δ or the *outer transform* of Δ:

$$\#A\neq 0\ \wedge.\ \#B\neq 0\ .\rightarrow.\ A\bar{\Delta}\ B\ =\ :\neq\Delta.A\underline{\times}B$$

$$\#A\times\#B=0\ \rightarrow.\ A\bar{\Delta}\ B\ =.\ \sim A\cup\sim B\rho\circ.\supset A\ \Delta\supset B.$$

Outer transforms are the dyadic counterpart of replacement transforms.

$$\circ A\bar{\Delta}\ B\ =\ :\lambda Z(A\ \Delta\ Z)B$$

$$A\bar{\Delta}\circ B\ =\ :\lambda Z(Z\ \Delta\ B)A.$$

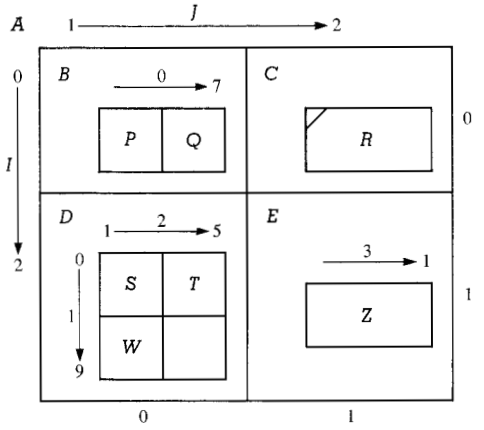
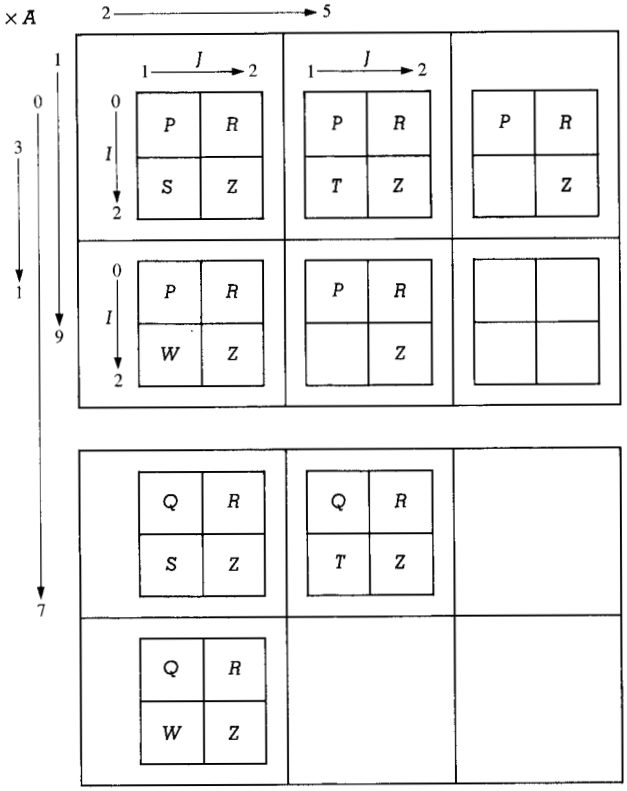


Figure 1 A 2 by 2 table A holding a septuple B, a single C, a 9 by 5 table D, and a solitary E.

Figure 2 The Cartesian product $\underline{\times}A$ of the array A in Fig. 1.



Some of the properties of Cartesian products and outer transforms stem from the following identities:

$\underline{\times}:\circ A=\circ A$ $\quad\underline{\times},A=:,\underline{\times}A$ $\quad\underline{\times}\backslash A=\circ\backslash\supset A$

$\underline{\times}\circ A=:\circ A$ $\quad,\underline{\times}A=,\underline{\times}:,A$ $\quad\backslash\underline{\times}A=\backslash\underline{\times}:\backslash A.$

An analogue $:\cup\underline{\times}:\underline{\times}A=\underline{\times}\cup A$ of the generalized associa-

tive law holds for Cartesian products. A monadic operation Δ is *general associative* if $\Delta{:}\Delta\ A{=}\Delta\cup A$ for every A. If Δ is general associative, then so is $:\Delta\underline{\times}$. $:\not\Delta\underline{\times}$ is the same operation as $(:(\not\Delta))\underline{\times}$. If A is a finite, nonempty, nonsingular array and Δ is a dyadic operation, then $\not\Delta\ A{=}:\not\Delta\underline{\times}A$. If Δ is associative, then both $\not\Delta$ and $\not\Delta$ are general associative for finite, nonempty, nonsingular arrays.

Monadic union $\cup$ extends the domain of definition of the reduction transform $\not\cup$ of dyadic union from finite, nonempty, nonsingular arrays to empty, singular, and infinite arrays. $:\cup\underline{\times}$ extends the domain of definition of the reduction transform $\not\cup$ of outer union to empty and singular arrays.

The *Cartesian array* $\iota{\sim}A$ of an array A equals $\underline{\times}{:}\iota{\sim}A$. Each item of $\iota{\sim}A$ equals its address in $\iota{\sim}A$. Every address to an item in A occurs in $\iota{\sim}A$. Thus $\iota 0{=}\Theta$ and $\iota\Theta{=}\circ\Theta$. The shape of $\iota{\sim}A$ equals $\sim A$. $\iota 3\ 0\ 4$ is the empty array of shape $3\ 0\ 4$ with pattern $0\ 0\ 0$.

If B is an array of addresses to items in A in the sense that $B{\subset}\iota{\sim}A$, then $B\bar{}\supset\circ A$ is an array of shape $\sim B$ of items selected from A. Let $B\#A$ and $A[B]$ equal $B\bar{}\supset\circ A$. For every array A, both $\iota{\sim}A\#A$ and $A[\iota{\sim}A]$ equal A. The dyadic operation $\#$ is called *boxing*. If $I{\in}\iota{\sim}A$, then $I\supset A{\in}A$. If $B{\subset}\iota{\sim}A$, then $B\#A{\subset}A$.

The exponentiation of a class M to the power of a class N is the class of all functions mapping N into M. Let the *exponentiation* $A\underline{*}B$ of an array A to the power of an array B be an array holding all possible arrays got by replacing the items of B by items of A:

$$A\underline{*}B \quad \text{for} \quad \underline{\times}(\sim B\rho\circ\iota{\sim}A)\bar{}\#\circ A.$$

$$\#B{<}1W \rightarrow .\ \#(A\underline{*}B){=}.\#A{*}\#B.$$

Positional transforms are defined in terms of a replicating transpose operation $\lozenge$, which works in the same way as the APL dyadic transpose except for diagonal slices involving singular axes. The replicating transpose causes singular axes to replicate to the length of the shortest nonsingular axis. For example, if $A{=}.1\ 3\rho$ `'PQR'`, then $0\ 0\lozenge A$ equals `,'P'` in APL but `'PQR'` in array theory.

In APL, if A and B are simple arrays of the same size, then $A{+}B$ equals $((\iota\rho\rho A),\iota\rho\rho B)\lozenge A\circ.{+}B$. This equation suggests the following definition in terms of the replicating transpose. For every dyadic operation Δ and for all arrays A and B regardless of conformability,

$$A{:}\Delta\ B\ =.\ \iota\#{\sim}A\cup\iota\#{\sim}B\lozenge.A\bar{}\Delta\ B.$$

If A has 3 axes and B has 4, then $\iota\#{\sim}A\cup\iota\#{\sim}B$ equals $0\ 1\ 2\ 0\ 1\ 2\ 3$.

22. Motes

Moting sends arrays to motes. Unmoting is inverse to moting. Moting and unmoting have no effect on basic motes. Unmoting an array is the same as unmoting its extract.

The *mote* $\underline{\circ}A$ of an array A is, in intuitive terms, the array $\circ\circ\circ\ldots A$. The moting operation has the property that the mote of an array is a mote:

$$\underline{\circ}A{=}\circ\underline{\circ}A.$$

The *unmote* $\underline{\supset}A$ of an array A is the array $\ldots\supset\supset\supset A$ got by $1W$ successive extractions of A. The unmote of an array equals the unmote of the extract of the array:

$$\underline{\supset}A{=}\underline{\supset}\supset A.$$

The unmote of the mote of an array is the array itself. $\underline{\supset}\underline{\circ}A{=}A$. An array A is said to be a *basic mote* if and only if $A{=}\underline{\circ}A$. If A is a basic mote, then $\underline{\supset}A{=}A$. Any finite number of motings or unmotings of a basic mote returns the basic mote as value. The foregoing suggests that a basic mote may be interpreted as a transfinite nesting $1W^{W}$ deep of singles within singles:

$$\circ\circ\circ\ldots\ \circ\circ\circ\ldots\ \circ\circ\circ\ldots\ \ldots B.$$

The operations of moting and unmoting permit arbitrary arrays to be moted and then treated in essentially the same way as basic motes. The mote of any array other than a basic mote can be used to augment the store of basic motes. If basic motes are the only motes needed in the theory, then the operations of moting and unmoting are superfluous, and there is no need to distinguish between motes and basic motes. Singling makes an arbitrary array behave enough like a mote to satisfy most applications.

When moting is included in the theory, the pattern of the mote of an array equals the mote of the plan:

$$\supset\backslash\underline{\circ}A\ =\ \underline{\circ}\supset\backslash\circ A.$$

Both singling and moting commute with planning:

$$\supset\backslash\circ\circ A\ =\ \circ\supset\backslash\circ A$$

$$\supset\backslash\circ\underline{\circ}A\ =\ \underline{\circ}\supset\backslash\circ A.$$

23. Proofs

The proofs from hypotheses in Appendix II use the deduction theorem, *reductio ad absurdum*, the law of the excluded middle, and the inferential rules of *modus ponens* and substitution.

Appendix I lists the axioms, definitions, and theorems in an order that is prescribed by a desire to deduce as much as possible from each assumption before introducing a new assumption. Corollaries are listed below each axiom, definition, and theorem. Appendix II lists the

proofs of the theorems. The rules of inference are *modus ponens* and substitution.

The premisses of a proof occur above the *proof mark* '⊢'. Conclusions occur after and below the proof mark. By the deduction theorem [35], the conjunction of all the premisses implies any conjunction of some or all of the conclusions; in particular, the last conclusion alone. The proof mark is omitted in proofs having no premisses.

Two proofs are required for the proof of a coimplication: one for each direction of implication. The proofs usually depend on different selections of axioms and so occur at different points in the sequence of theorems. For example, only the direct proof occurs below Theorem 4. The converse proof occurs at Theorem 70. Theorems 4 and 70 are different implications but the same coimplication.

The reasons justifying the appending of conclusions to a proof are shown in a right-hand column of theorems. The format is believed to be new in that the justifying theorems are displayed in full. The required substitutions are usually easy to determine because the proofs are short.

A definition, such as Definition 1,

$\#A$ for $\times\sim A$

is a metalinguistic rule that allows any occurrence of the definiens (on the right) to be replaced by an occurrence of the definiendum (on the left). For example, the theorem $\times\sim A=\#A$ follows by Definition 1 from $\times\sim A=\times\sim A$, which is a substitution instance of Axiom 1.

The deduction of a contradiction from a premiss, as in Theorem 10, proves the denial of the premiss. According to the law of the excluded middle, a conclusion is a theorem if it can be deduced from a premiss as well as the denial of the premiss. For example, Theorem 98 is implied by $\#A\neq 0$ alone and by $\#A=0$ alone. Hence $\#A\neq 0 \vee . \#A=0$ implies the theorem. The theorem then follows from no premisses.

Definition 4, which is not used in Appendix II, is stated in Appendix I only to show how singling is defined in terms of primitive operations. The properties of the definition are not needed until Part II of the axioms and theorems. The definition is analogous to that of a unit class as an unordered pair.

Definition 4. A singled array equals the null reshape of the array paired with itself.

$\circ A$ for $\theta\rho . A\,\overline{,}\, A$.

Acknowledgments

Many conversations with K. E. Iverson and A. D. Falkoff during the first stages of this work taught me much about their approach to the design of APL, stimulated fruitful ideas, and helped to close off unprofitable lines of development. They have supported and encouraged the work, despite certain differences in views, over a period of four years. My debt to them is both scientific and personal.

I taught and tested an early axiomatization of the theory at Yale University in the spring of 1970 and am indebted to the members of the class for many clarifying discussions [9b]. I am indebted also to W. H. Burge, W. C. Carter, Z. J. Ghandour, P. C. Gilmore, J. M. Kurtzberg, E. E. McDonnell, R. A. Nelson, Y. Ron, and particularly J. E. Mezei for various technical discussions.

The final choices of notation and terminology were tested and revised in conversation with W. G. Bouricius, who has diligently questioned the theory and examined its practical consequences. His early encouragement and interest helped to spur the work to completion. I am grateful to J. C. McPherson and H. A. Ernst for their interest and support.

References

1. K. E. Iverson, *A Programming Language*, John Wiley and Sons, Inc., New York, 1962.
2. ——, *Elementary Functions: an algorithmic treatment*, Science Research Associates, Inc., Chicago, 1966.
3. A. D. Falkoff and K. E. Iverson, *APL\360 User's Manual*, IBM Corporation (GH20-0683-1) 1970.
4. ——, ——, and E. H. Sussenguth, "A formal description of System/360," *IBM Systems Journal* **3**, 198–263 (1964).
5. W. S. Hatcher, *Foundations of Mathematics*, W. B. Saunders Company, Philadelphia, 1968.
6. W. V. Quine, *Set Theory and its Logic*, Revised ed., The Belknap Press of Harvard University Press, Cambridge, Mass., 1969.
7. Oliver Heaviside, *Electrical Papers*, Vols. I and II, The Macmillan Company, New York, 1892. Second ed., Chelsea Publishing Company, Bronx, New York, 1970.
8. B. van der Pol and H. Bremmer, *Operational Calculus*, Second ed., Cambridge University Press, London, 1955.
9. (a) Trenchard More, Jr., "Notes on the development of a theory of arrays," IBM Philadelphia Scientific Center Technical Report No. 320–3016, March 1972. See also (b) "Notes on the axioms for a theory of arrays," IBM Philadelphia Scientific Center Technical Report No. 320–3017, March 1972.
10. Jacques Herbrand, "Recherches sur la théorie de la démonstration," *Travaux de la Société des Sciences et des lettres de Varsovie*, **Classe III**, No. 33, 33–160 (1930). English translation in van Heijenoort [17], pp. 525–581.
11. Trenchard More, Jr., "An interactive method for algebraic proofs," Proceedings of the Twenty-Fifth Summer Meeting of the Canadian Mathematical Congress, June 16–18, 1971, Lakehead University, Thunder Bay, Ontario, 129–217. Also an IBM Philadelphia Scientific Center Technical Report, No. 320–3005, September 1971.
12. A. H. Clifford and G. B. Preston, *The Algebraic Theory of Semigroups*, Vol. I, American Mathematical Society, Providence, R.I., 1961.
13. J. A. Schouten, *Tensor Analysis for Physicists*, Second ed., Oxford University Press, London, 1954.

14. Louis Brand, *Vector and Tensor Analysis*, John Wiley and Sons, Inc., New York, 1947.
15. E. A. Guillemin, *The Mathematics of Circuit Analysis*, John Wiley and Sons, Inc., New York, 1949.
16. Saunders MacLane and Garrett Birkhoff, *Algebra*, The Macmillan Company, New York, 1967.
17. Jean van Heijenoort, *From Frege to Gödel, A Source Book in Mathematical Logic, 1879–1931*, Harvard University Press, Cambridge, Mass., 1967.
18. Georg Cantor, "Über unendliche, lineare Punktmannichfaltigkeiten," *Math. Annalen* **15,** 1–7 (1879); **17,** 355–358 (1880); **20,** 113–121 (1882); **21,** 51–58, 545–591 (1883).
19. Kazimierz Kuratowski and Andrzej Mostowski, *Set Theory*, North-Holland Publishing Company, Amsterdam, 1968.
20. M. M. G. Ricci, "Delle derivazioni covarianti e controvarianti," Studi éditi dall' Università di Padova ecc, Padova, 1888.
21. M. M. G. Ricci and Tullio Levi-Civita, "Méthodes de calcul différentiel absolu et leurs applications," *Math. Annalen* **54,** 125–201 (1901).
22. Cornelius Lanczos, *Space through the Ages*, Academic Press, New York, 1970.
23. S. C. Kleene, *Mathematical Logic*, John Wiley and Sons, Inc., New York, 1967.
24. W. V. Quine, "Unification of universes in set theory," *Journal of Symbolic Logic* **21,** 267–279 (1956).
25. A. M. Gleason, *Fundamentals of Abstract Analysis*, Addison-Wesley Publishing Company, Reading, Mass., 1966.
26. Norbert Wiener, "A simplification of the logic of relations," *Proceedings of the Cambridge Philosophical Society* **17,** 387–390 (1914). Reprinted in van Heijenoort [17], pp. 224–227.
27. Trenchard More, Jr., NSF Grant GK-305, July 1965–July 1967. Yale Junior Faculty Fellowship 1967–68, proposal, "Development of an algorithmic set theory, with applications to algebra," Feb. 1967.
28. D. L. Childs, "Feasibility of a set-theoretic data structure," The University of Michigan, Technical Report 6, CONCOMP Project, Aug., 1968.
29. J. T. Schwartz, "Abstract algorithms and a set-theoretic language for their expression," Preliminary draft, first part, 1970–71.
30. John McCarthy, "Recursive functions of symbolic expressions and their computation by machine, Part I," *Communications of the ACM* **3,** 184–195 (1960). Reprinted in Saul Rosen, ed., *Programming Systems and Languages*, McGraw-Hill Book Company, Inc., New York, 1967.
31. W. V. Quine, "New foundations for mathematical logic," *American Mathematical Monthly* **44,** 70–80 (1937).
32. ——, *Mathematical Logic*, Revised ed., Harvard University Press, Cambridge, Mass., 1951.
33. P. M. Cohn, *Universal Algebra*, Harper and Row, Publishers, New York, 1965.
34. Alonzo Church, *The Calculi of Lambda-Conversion*, Annals of Mathematics Studies No. 6, Princeton University Press, Princeton, N.J., 1941.
35. ——, *Introduction to Mathematical Logic*, Vol. I, Princeton University Press, Princeton, N.J., 1956.
36. Ernst Zermelo, "Untersuchungen über die Grundlagen der Mengenlehre I," *Math. Annalen* **65,** 261–281 (1908). English translation in van Heijenoort [17], pp. 199–215.
37. P. R. Halmos, *Finite-Dimensional Vector Spaces*, Second ed., D. van Nostrand Company, Inc., New York, 1958.
38. A. A. Fraenkel and Yehoshua Bar-Hillel, *Foundations of Set Theory*, North-Holland Publishing Company, Amsterdam, 1958.
39. Patrick Suppes, *Axiomatic Set Theory*, D. van Nostrand Company, Inc., Princeton, N.J., 1960.
40. Thoralf Skolem, "Einige Bermerkungen zur axiomatischen Begrüngung der Mengenlehre," *Wiss. Vorträge gehalten auf dem 5. Kongress der skandinav. Mathematiker in Helsingfors 1922*, 217–232 (1923). English translation in van Heijenoort [17], pp. 290–301.
41. Arend Heyting, *Intuitionism, an introduction*, North-Holland Publishing Company, Amsterdam, 1956.
42. H. B. Curry, *Foundations of Mathematical Logic*, McGraw-Hill Book Company, Inc., New York, 1963.
43. Gottlob Frege, *Grundgesetze der Arithmetik*, Vol. I, Jena, 1893.
44. Ernst Zermelo, "Beweis, dass jede Menge wohlgeordnet werden kann," *Math. Annalen* **59,** 514–516 (1904). English translation in van Heijenoort [17], pp. 139–141.
45. A. A. Fraenkel, *Abstract Set Theory*, North-Holland Publishing Company, Amsterdam, 1953. Second ed., 1961.
46. Alexander Abian, *The Theory of Sets and Transfinite Arithmetic*, W. B. Saunders Company, Philadelphia, 1965.
47. Z. J. Ghandour and J. E. Mezei, "General arrays, operators, and functions," *IBM J. Res. Develop.*, to be published.
48. G. D. Mostow and J. H. Sampson, *Linear Algebra*, McGraw-Hill Book Company, Inc., New York, 1969.

Received May 5, 1972

Revised September 18, 1972

The address of the author is the IBM Data Processing Division Scientific Center, 3401 Market Street, Philadelphia, Pennsylvania 19104.

Appendix 1: List of axioms, definitions, theorems and corollaries

A1 $A=A$

A2 $A=B \rightarrow .\ \Delta\ A=\Delta\ B$
$\Delta\ A\neq\Delta\ B \rightarrow .\ A\neq B$

A3 $A=B \rightarrow .\ A\ \Delta\ C=.B\ \Delta\ C$
$A\ \Delta\ C\neq.B\ \Delta\ C\ .\rightarrow .\ A\neq B$

A4 $A=B \rightarrow .\ C\ \Delta\ A=.C\ \Delta\ B$
$C\ \Delta\ A\neq.C\ \Delta\ B\ .\rightarrow .\ A\neq B$

1 $A=B \equiv .\ B=A$
$B\neq A \equiv .\ A\neq B$

2 $A=B\ \wedge .\ B=C\ .\rightarrow .\ A=C$
$A=B\ \wedge .\ A\neq C\ .\rightarrow .\ B\neq C$

D1 $\#A\ \ FOR\ \ \times\sim A$
$\times\sim A=\#A$

D2 $0\ \ FOR\ \ \#\Theta$
$\#\Theta=0$
$\times\sim\Theta=0$

D3 $1\ \ FOR\ \ \#0$
$\#0=1$
$\times\sim 0=1$
$\#\#\Theta=1$

A5 $\sim 0=\Theta$
$\times\Theta=1$
$\#\sim 0=0$
$\times\sim\sim 0=0$
$\sim A=\Theta \equiv .\ \#\sim A=0$

D4 $\circ A\ \ FOR\ \ \Theta\rho .A;A$
$\Theta\rho(A;A)=\circ A$

A6 $\#A=\circ\#A$
$0=\circ 0$
$1=\circ 1$
$\times\sim A=\circ\times\sim A$

A7 $\supset\circ A=A$

3 $\circ A=\circ B \equiv .\ A=B$

4 $A=\circ A \equiv .\ A=\supset A$
$\supset 0=0$
$\supset 1=1$
$\supset\#A=\#A$
$\supset\times\sim A=\times\sim A$

A8 $\Theta\rho A=\circ\supset A$
$\Theta\rho\circ A=\circ A$

A9 $\sim(\sim A\rho B)=\sim A$
$\#(\sim A\rho B)=\#A$
$\sim(\Theta\rho A)=\Theta$

5 $\sim\circ A=\Theta$
$\sim\#A=\Theta$
$\sim 1=\Theta$
$\sim\times\sim A=\Theta$
$\sim\circ A=\sim 0$
$A=\circ B \rightarrow .\ \sim A=\Theta$
$A=\circ\supset A \equiv .\ \sim A=\Theta$
$\#\sim\circ A=0$
$\times\sim\sim\circ A=0$

6 $\#\circ A=1$
$\#\#A=1$
$\#1=1$
$\sim A=\#A \equiv .\ \#\sim A=1$
$\sim A=\circ\sim A \equiv .\ \#\sim A=1$
$\#(\Theta\rho A)=1$
$\times\sim\circ A=1$
$\times\sim 1=1$
$\#\times\sim A=1$
$\times\sim\#A=1$
$\times\sim\times\sim A=1$

D5 $,A\ \ FOR\ \ \#A\rho A$
$\#A\rho A=,A$
$\#\sim A\rho\sim A=,\sim A$

7 $1\rho\circ A=,\circ A$
$1\rho 0=,0$
$1\rho 1=,1$
$1\rho\#A=,\#A$
$1\rho\times\sim A=,\times\sim A$

A10 $\sim,A=\#A$
$\sim,0=1$
$\sim,1=1$
$\sim,\#A=1$
$\#\sim,A=1$
$\sim,\sim,A=1$
$\sim,\circ A=1$
$\sim,\Theta=0$
$\sim,\sim 0=0$
$\sim,\sim\circ A=0$
$\sim,A=\circ\sim,A$
$\supset\sim,A=\sim,A$
$\sim\sim,A=\Theta$

$\sim,A=\times\sim A$
$1\rho\sim,A=,\sim,A$
$A=,A \equiv. \sim A=\#A$
$A=,A \equiv. \#\sim A=1$
$A=,\circ B \rightarrow. \sim A=1$
$A=,\circ\supset A \equiv. \sim A=1$
$,A=,\circ\supset A \equiv. \#A=1$
$\sim,A\rho A=,A$

8 $\sim(\#A\rho B)=\#A$
$\sim(0\rho A)=0$
$\sim(1\rho A)=1$

A11 $0\neq 1$
$\#\circ A\neq 0$
$\#\ominus 0\neq 0$
$\#1\neq 0$
$\#\ominus\neq 1$

9 $\circ A\neq\ominus$
$\#A\neq\ominus$
$0\neq\ominus$
$1\neq\ominus$

10 $\circ A\neq,\circ A$

11 $,A\neq\circ,A$
$\circ,A\neq\circ\circ,A$
$\circ\circ,A\neq\circ\circ\circ,A$

12 $,A\neq\circ\circ,A$
$\circ,A\neq\circ\circ\circ,A$

D6 $\backslash A$ *FOR* $0\rho A$
$0\rho A=\backslash A$

13 $\sim\backslash A=0$
$\#\sim\backslash A=1$
$A=\backslash A \equiv. \sim A=0$

14 $\sim\backslash A\rho B=\backslash B$

15 $\#A=0 \equiv. ,A=\backslash A$

16 $,A=\backslash A \equiv. \#A=0$

A12 $0\leq\#A$

A13 $\#A\leq\#A$

A14 $\#A\neq 0 \rightarrow. 1\leq\#A$

A15 $\#A\leq\#C \rightarrow. \sim A\rho B=.\sim A\rho.\sim C\rho B$
$\sim A\rho B=.\sim A\rho.\sim A\rho B$
$1\rho A=.1\rho.1\rho A$
$\#A=0 \rightarrow. \sim A\rho B=.\sim A\rho.\sim C\rho B$

17 $\#A=0 \rightarrow. \sim A\rho B=.\sim A\rho\backslash B$

18 $\backslash\backslash A=\backslash A$

19 $\#A\neq 0 \rightarrow. \supset(\sim A\rho B)=\supset B$

A16 $\#B=0 \rightarrow. \sim A\rho B=.\sim A\rho.\sim C\rho B$

20 $\#B=0 \rightarrow. \supset(\sim A\rho B)=\supset B$

A17 $\#,A=\#A$
$\sim A\rho B=.\sim A\rho.\#A\rho B$
$\ominus\rho A=.\ominus\rho.1\rho A$

21 $\times\#A=\#A$
$\times\circ\#A=\circ\#A$
$\times\times\sim A=\times\sim A$
$\times\sim,A=\sim,A$
$\times 0=0$
$\times 1=1$

22 $\#\backslash A=0$
$\sim A\rho\backslash B=.\sim A\rho.\sim C\rho\backslash B$
$\supset(\sim A\rho\backslash B)=\supset\backslash B$

23 $,\backslash A=\backslash A$

24 $\sim A\rho\backslash B=.\sim A\rho\circ\supset\backslash B$
$\#A\rho\backslash B=.\#A\rho\circ\supset\backslash B$

25 $\backslash(\sim A\rho B)=\backslash B$

26 $\backslash A=\backslash\circ\supset A$
$\sim A\rho\backslash B=.\sim A\rho\backslash\circ\supset B$
$\supset\backslash A=\supset\backslash\circ\supset A$
$\supset\backslash\circ\supset\backslash A=\supset\backslash A$
$A=\supset\backslash A \rightarrow. A=\supset\backslash\circ A$
$\supset\backslash\circ\supset\backslash\circ A=\supset\backslash\circ A$
$\backslash\circ\supset\backslash\circ\supset A=\backslash A$
$\backslash\circ\supset\backslash A=\backslash A$
$\backslash\circ A=\backslash\circ\supset\backslash\circ A$
$\supset\backslash\circ\supset A=\supset\backslash\circ A \equiv. \backslash A=\backslash\circ A$

27 $\supset\backslash A=\supset\backslash B \equiv. \backslash A=\backslash B$

A18 $\#A\mid\#A=0$

$A19$ $\#B \leq \#C \ \wedge . \ \#B \mid \#C = 0 \ .\rightarrow .$
$\sim A \rho B = . \sim A \rho . \sim C \rho B$

28 $\sim A \rho B = . \sim A \rho , B$
$\sim , A \rho , A = , A$
$\# A \rho B = . \# A \rho , B$
$\# B = 0 \ \rightarrow . \ \sim A \rho B = . \sim A \rho \backslash B$
$\# B = 0 \ \rightarrow . \ \sim A \rho B = . \sim A \rho \circ \supset \backslash B$

29 $\backslash , A = \backslash A$

30 $, , A = , A$

31 $\supset A = \supset , A$
$\supset , \circ A = A$
$\supset A = \supset , \circ \supset A$

32 $, \circ A = , \circ B \ \equiv . \ A = B$

33 $\# A = 0 \ \rightarrow . \ \supset A = \supset \backslash A$

34 $\# A \rho B = . \# A \rho . \sim A \rho B$
$1 \rho A = . 1 \rho . \theta \rho A$

35 $, (\sim A \rho B) = . \# A \rho B$
$, (\sim A \rho B) = . \sim , A \rho B$
$, (\# A \rho B) = . \# A \rho B$

36 $\# A = \# B \wedge . \sim A \rho B = A \ . \equiv . \ , A = , B$

37 $1 \rho A = , \circ \supset A$
$1 \rho A = , . \theta \rho A$
$1 \rho (A \dot{,} A) = , \circ A$
$\# 0 \rho A = , \circ \supset A$

38 $\# A = 1 \ \equiv . \ , A = , \circ \supset A$

39 $\# A = 0 \ \rightarrow . \ \supset (\sim A \rho B) = \supset \backslash B$

40 $\# (\# A \rho B) = \# A$
$\# (0 \rho A) = 0$
$\# (1 \rho A) = 1$

$A20$ $1 \mid \# A = 0$

41 $\# C \neq 0 \ \rightarrow . \sim A \rho \circ B = . \sim A \rho . \sim C \rho \circ B$
$\# C \neq 0 \ \rightarrow . \# A \rho \circ B = . \# A \rho . \# C \rho \circ B$
$\# B = 1 \ \wedge . \ \# C \neq 0 \ .\rightarrow .$
$\sim A \rho B = . \sim A \rho . \sim C \rho B$

$A21$ $\sim A \rho A = A$
$\sim A \rho , A = A$

42 $\sim A = \sim B \ \wedge . \ , A = , B \ . \equiv . \ A = B$
$\sim A = \# A \ \equiv . \ A = , A$

43 $\sim A = 0 \ \equiv . \ A = \backslash A$

44 $\sim A = 1 \ \equiv . \ A = , \circ \supset A$
$\sim A = 1 \ \rightarrow . \ A = , A$

45 $\sim A = \theta \ \equiv . \ A = \circ \supset A$

$A22$ $\backslash \sim A = \theta$
$\sim \theta = 0$
$\backslash \theta = \theta$
$, \theta = \theta$
$\sim \sim 0 = 0$
$\backslash \sim 0 = \theta$
$, \sim 0 = \theta$
$\sim \sim \circ A = 0$
$\backslash \sim \circ A = \theta$
$, \sim \circ A = \theta$

46 $\backslash \# A = \theta$
$\backslash \circ \# A = \theta$
$\backslash 0 = \theta$
$\backslash 1 = \theta$
$\backslash \# A = \sim 0$
$\backslash \sim , A = \theta$
$\backslash \times \sim A = \theta$

47 $\sim \theta \rho A = \backslash A$

$A23$ $\supset \theta = 0$
$\supset \backslash \theta = 0$

48 $\supset \backslash \sim A = 0$
$\supset \backslash 0 = 0$
$\supset \backslash \# A = 0$
$\supset \backslash \circ \# A = 0$
$\supset \backslash 1 = 0$
$\supset \backslash \sim , A = 0$
$\supset \backslash \times \sim A = 0$
$\supset \backslash \circ \sim , A = 0$
$\supset \backslash \circ \times \sim A = 0$

49 $\sim A \rho \theta = . \sim A \rho 0$
$\theta \rho \theta = 0$
$1 \rho \theta = , 0$
$1 \rho \sim 0 = , 0$
$1 \rho \backslash 0 = , 0$

A24 $\supset\backslash\circ A = .\sim A \rho \backslash A$
$\supset\backslash\circ A = .\sim A \rho \circ \supset\backslash A$
$\sim\supset\backslash\circ A = \sim A$
$\#\supset\backslash\circ A = \# A$
$\supset\backslash\circ, A = .\# A \rho \backslash A$
$\supset\backslash\circ, A = .\# A \rho \circ \supset\backslash A$

50 $\supset\backslash\circ A = .\sim A \rho \circ \supset\backslash\circ\supset A$
$\supset\backslash\circ A = .\sim A \rho \circ .\sim\supset A \rho \circ \supset\backslash\circ\supset\supset A$
$\supset\backslash\circ A = .\sim A \rho \circ .$
$\sim\supset A \rho \circ .\sim\supset\supset A \rho \circ \supset\backslash\circ\supset\supset\supset A$

51 $\backslash A = \backslash\circ .\sim\supset A \rho \circ \supset\backslash\circ\supset\supset A$
$\backslash A = \backslash\circ .\sim\supset A \rho \circ .$
$\sim\supset\supset A \rho \circ \supset\backslash\circ\supset\supset\supset A$

52 $\supset\backslash A = .\sim\supset A \rho \circ \supset\backslash\circ\supset\supset A$
$\supset\backslash A = .\sim\supset A \rho \circ .$
$\sim\supset\supset A \rho \circ \supset\backslash\circ\supset\supset\supset A$

53 $\#A = 0 \rightarrow . \ A = \supset\backslash\circ A$

54 $\supset\backslash A = .\sim\supset A \rho \backslash\supset A$
$\sim\supset\backslash A = \sim\supset A$
$\#\supset\backslash A = \#\supset A$

55 $\backslash\supset\backslash A = \backslash\supset A$
$\supset\backslash\supset A = \supset\backslash\supset\backslash A$
$\backslash\supset\backslash\circ A = \backslash A$
$\supset\backslash\supset\backslash\circ A = \supset\backslash A$
$\supset\backslash\circ\supset\backslash A = \supset\backslash\supset\backslash\circ A$
$\backslash\supset\supset A = \backslash\supset\backslash\supset A$
$\backslash\supset\supset A = \backslash\supset\backslash\supset\backslash A$
$\backslash\supset\supset A = \backslash\supset\supset\backslash A$

56 $\supset\backslash\circ\backslash A = \backslash A$
$\supset\backslash\circ\Theta = \Theta$
$\supset\backslash\circ\backslash A = \backslash\supset\backslash\circ A$

57 $\supset\supset\backslash\circ A = \supset\backslash A$
$\supset\backslash\circ\supset A = \supset\supset\backslash\circ A$
$\supset\backslash\circ A = \supset\backslash\circ\supset A \rightarrow .$
$\supset\backslash\circ\supset A = \supset\backslash\circ\supset\supset A$

58 $\supset\supset\backslash A = \supset\backslash\supset A$
$\supset\supset\backslash A = \supset\backslash\supset\backslash A$
$\supset\supset\supset\backslash A = \supset\backslash\supset\supset A$
$\supset\supset\supset\backslash A = \supset\backslash\supset\backslash\supset\backslash A$

59 $\supset\backslash A = .\sim\supset A \rho \circ \supset\supset\backslash A$

60 $\sim\supset\backslash\supset A = \sim\supset\supset A$
$\sim\supset\supset\backslash A = \sim\supset\supset A$
$\sim\supset\backslash\supset\backslash A = \sim\supset\supset A$
$\#\supset\backslash\supset A = \#\supset\supset A$
$\#\supset\supset\backslash A = \#\supset\supset A$
$\#\supset\backslash\supset\backslash A = \#\supset\supset A$

61 $\supset\backslash\circ\circ A = \circ\supset\backslash\circ A$
$\supset\backslash\circ\circ\supset A = \circ\supset\supset\backslash\circ A$
$A = \supset\backslash A \rightarrow . \ \circ A = \supset\backslash\circ\circ\supset A$
$A = \circ A \equiv . \ \supset\backslash\circ A = \circ\supset\backslash\circ A$
$A = \circ A \rightarrow . \ \supset\backslash A = \circ\supset\backslash A$
$A = \circ\supset A \equiv . \ \supset\backslash\circ A = \circ\supset\supset\backslash\circ A$

62 $\sim A \rho \backslash B = .\sim A \rho \supset\backslash\circ B$
$\sim A \rho \supset\backslash\circ B = .\sim A \rho \backslash\circ\supset B$
$\sim A \rho \supset\backslash\circ B = .\sim A \rho \circ \supset\backslash B$
$\# A \rho \backslash B = .\# A \rho \supset\backslash\circ B$
$\# A \rho \supset\backslash\circ B = .\# A \rho \backslash\circ\supset B$
$\# A \rho \supset\backslash\circ B = .\# A \rho \circ \supset\backslash B$

63 $\sim A \rho \supset\backslash B = .\sim A \rho \backslash\supset B$
$\# A \rho \supset\backslash B = .\# A \rho \backslash\supset B$
$\sim A \rho \supset\backslash B = .\sim A \rho \backslash\supset\backslash B$
$\sim A \rho \supset\backslash B = .\sim A \rho \circ \supset\backslash\supset\backslash B$

64 $\supset\backslash\circ(\sim A \rho B) = .\sim A \rho \backslash B$
$\supset\backslash\circ(\sim A \rho B) = .\sim A \rho \backslash\circ\supset B$
$\supset\backslash\circ(\sim A \rho B) = .\sim A \rho \supset\backslash\circ B$
$\supset\backslash\circ(\sim A \rho B) = .\sim A \rho \circ \supset\backslash B$
$\supset\backslash\circ(\# A \rho B) = .\# A \rho \backslash B$

65 $\supset\backslash\circ, A = , \supset\backslash\circ A$
$\supset\backslash\circ, \circ A = , \circ\supset\backslash\circ A$
$\supset\backslash\circ, \circ\supset A = , \circ\supset\supset\backslash\circ A$
$A = , A \equiv . \ \supset\backslash\circ A = , \supset\backslash\circ A$

66 $\supset\backslash\circ A = , \supset\backslash\circ A \equiv . \ A = , A$

67 $\supset\backslash\circ A = \circ\supset\supset\backslash\circ A \equiv . \ A = \circ\supset A$

68 $\supset\backslash\circ A = , \circ\supset\supset\backslash\circ A \equiv . \ A = , \circ\supset A$

69 $\supset\backslash\circ\sim A = .\sim\sim A \rho 0$
$\supset\backslash\circ, \sim A = .\# \sim A \rho 0$

A25 $\backslash A = \backslash\circ A \rightarrow . \ A = \circ A$
$\backslash A = \backslash\circ A \equiv . \ A = \circ A$

70 $A = \supset A \equiv . \ A = \circ A$

```
71    A=⊃\A →. A=∘A

72    ⊃\A=⊃\∘A ≡. A=∘A

73    ⊃\∘A=∘⊃\∘A ≡. A=∘A

74    ⊃\∘A=0 ≡. A=∘A ∧. ⊃\A=0

75    A≠,∘A

A26   ⊃\⍬=⍬

76    ⍬=∘⍬
      ⊃\∘⍬=⍬
      ⍬=⊃⍬
      ~⍬=⍬
      #⍬=1
      #~⍬=0
      ⍬≠⍬

A27   ⍬≠0

77    ⍬≠1

78    ~Aρ0≠⍬

79    ⊃\∘A=⍬ ≡. A=∘A ∧. ⊃\A=⍬

A28   ⊃\∘A=⍬ →. A=⍬
      ⊃\∘A=⍬ ≡. A=⍬
      A=∘A ∧. ⊃\A=⍬ .≡. A=⍬

D7    O FOR \⍬
      \⍬=O
      \O=O
      ,O=O
      ~O=0
      #~O=1
      #O=0

80    ⊃O=⍬
      ⊃\O=⍬
      ⊃O≠0
      O≠⍬

81    ⊃\∘O=O

82    ∘A≠O
      #A≠O
      0≠O
      1≠O
      ⍬≠O

A29   ⊃\' '=' '

83    ' '=∘' '
      ⊃\∘' '=' '
      ' '=⊃' '
      ~' '=⍬
      #' '=1
      #~' '=0
      ' '≠⍬
      ' '≠O

A30   ' '≠0

A31   ' '≠⍬

84    ' '≠1

85    ⊃\∘A=' ' ≡.
        A=∘A ∧. ⊃\A=' '

D8    ⍫ FOR \' '
      \' '=⍫
      \⍫=⍫
      ,⍫=⍫
      ~⍫=0
      #~⍫=1
      #⍫=0

D9    '' FOR ⍫
      ''=⍫

86    ⊃⍫=' '
      ⊃\⍫=' '
      ⊃⍫≠0
      ⍫≠⍬
      ⊃⍫≠⍬
      ⍫≠O

87    ⊃\∘⍫=⍫
```

```
88      ∘A≠⍬
        #A≠⍬
        0≠⍬
        1≠⍬
        ⍟≠⍬
        ! '≠⍬
        '

A32     #A≠0 ∧. #B≠0 .→.
          :Δ(~AρB)=.~Aρ:Δ B

89      #A≠0 ∧. #B≠0 .→.
          :Δ(#AρB)=.#Aρ:Δ B

90      #A≠0 →. :Δ∘⊃A=∘⊃:Δ A

A33     #A=0 →.
          :Δ(~AρB)=.~Aρ∘Δ⊃\B
        #A=0 →.
          :Δ(#AρB)=.#Aρ∘Δ⊃\B

91      :Δ\A=\∘Δ⊃\A

92      :⊃\A=\⊃A

93      :Δ\A=\:Δ\A

94      #A=0 →.
          :Δ(~AρB)=.~Aρ:Δ\B
        #A=0 →. :Δ A=.~Aρ:Δ\A
        #A=0 →.
          :Δ(~Aρ\B)=.~Aρ:Δ\B

95      #A=0 →. :⊃(~AρB)=.~Aρ⊃B

96      #A=0 →. :⊃A=.~Aρ⊃A
        #A=0 →. :⊃A=.~Aρ\⊃A
        #A=0 →. :⊃A=.~Aρ⊃\A

97      #A=0 →. :Δ A=.~Aρ∘Δ⊃A

98      :Δ A=.~Aρ:Δ A

99      ~:Δ A=~A
        #:Δ A=#A
        ,:Δ A=.#Aρ:Δ A

100     :Δ,A=,:Δ A
        :Δ,A=.#Aρ:Δ A

101     :Δ A=.~Aρ:Δ,A

102     #A=0 →. :Δ\A=\:Δ A

103     #A=0 →. ⊃:Δ A=⊃\∘Δ⊃A

104     :Δ⍜=\∘Δ 0
        :∘⍜=⍜
        :⊃⍜=⍜
        :~⍜=\∘⍜
        :\⍜=\∘⍜
        ⊃:\⍜=⍜

105     :Δ⍜=\∘Δ⍟
        :∘⍜=⍜
        :⊃⍜=⍜
        :~⍜=\∘⍜
        :\⍜=\∘⍜
        ⊃:\⍜=⍜

106     :Δ⍬=\∘Δ' '
        :∘⍬=⍬
        :⊃⍬=⍬
        :~⍬=\∘⍜
        :\⍬=\∘⍬
        ⊃:\⍬=⍬

107     :#\A=⍜
        :#⍜=⍜
        :#\0=\0
        :#\~A=\~A
        :#\#A=\#A
        :#⍜=⍜
        :#⍬=⍜

108     #A=0 →. :#(~AρB)=.~Aρ0
        #A=0 →. :#A=.~Aρ0
        #A=0 →. ⊃:#A=0
```

Appendix 2: List of theorems with proofs

```
1       A=B ≡. B=A                    1
   A=B
 → A=A=.B=A          A3        A=B →. A Δ C=.B Δ C
   A=A               A1        A=A
   B=A

2       A=B ∧. B=C .→. A=C
   A=B
   B=C
 → A=C=.B=C          A3        A=B →. A Δ C=.B Δ C
   A=C

3       ∘A=∘B ≡. A=B              A2
   ∘A=∘B
 → ⊃∘A=⊃∘B           HYP
   A=B               A7        ⊃∘A=A

4       A=∘A ≡. A=⊃A              70
   A=∘A
 → ⊃A=⊃∘A            HYP
     =A              A7        ⊃∘A=A

5       ~∘A=Θ
~∘A=~∘⊃∘A           A7        ⊃∘A=A
   =~.Θρ∘A          A8        ΘρA=∘⊃A
   =Θ               A9        ~(~AρB)=~A

6       #∘A=1
×~∘A=×Θ          5         ~∘A=Θ
    =1           A5        ~0=Θ

7       1ρ∘A=,∘A
,∘A=.#∘Aρ∘A         D5        #AρA=,A
   =.1ρ∘A           6         #∘A=1

8       ~(#AρB)=#A
~(~,AρB)=~,A         A9        ~(~AρB)=~A
~(#AρA)=#A           A10       ~,A=#A

9       ∘A≠Θ
#∘A≠0           A11       0≠1
   ≠#Θ          D2        #Θ=0
∘A≠Θ            A2        A=B →. Δ A=Δ B

10      ∘A≠,∘A
   ∘A=,∘A
 → 0=#~∘A            5         ~∘A=Θ
    =#~,∘A           HYP
    =1               A10       ~,A=#A
    ≠1               A11       0≠1
```

```
11      ,A≠∘,A
    ,A=∘,A
 → 0=#~∘,A          5          ~∘A=⍬
     =#~,A          HYP
     =1             A10        ~,A=#A
     ≠1             A11        0≠1

12      ,A≠∘∘,A
    ,A=∘∘,A
 → 0=#~∘∘,A         5          ~∘A=⍬
     =#~,A          HYP
     =1             A10        ~,A=#A
     ≠1             A11        0≠1

13      ~\A=0
#⍬=~.#⍬ρA           8          ~(#AρB)=#A
0=~.0ρA             D2         #⍬=0
 =~\A               D6         0ρA=\A

14      ~\AρB=\B
~\AρB=.0ρB          13         ~\A=0
      =\B           D6         0ρA=\A

15      #A=0 ≡. ,A=\A      16
    #A=0
 → ,A=.#AρA          D5         #AρA=,A
     =.0ρA          HYP
     =\A            D6         0ρA=\A

16      ,A=\A ≡. #A=0      15
    ,A=\A
 → #A=~,A           A10        ~,A=#A
     =~\A           HYP
     =0             13         ~\A=0

17      #A=0 →. ~AρB=.~Aρ\B
    #A=0
 → #A≤#\C                       A12     0≤#A
   ~AρB=.~Aρ.~\CρB              A15     #A≤#C →. ~AρB=.~Aρ.~CρB
       =.~Aρ\B                  14      ~\AρB=\B

18      \\A=\A
#\B≤#\B                         A13     #A≤#A
~\BρA=.~\Bρ.~\BρA               A15     #A≤#C →. ~AρB=.~Aρ.~CρB
\A=\\A                          14      ~\AρB=\B

19      #A≠0 →. ⊃(~AρB)=⊃B    SEE 39
    #A≠0
 → 1≤#A                 A14     #A≠0 →. 1≤#A
   #0≤#A                D3      #0=1
   ⍬ρB=.⍬ρ.~AρB         A15     #A≤#C →. ~AρB=.~Aρ.~CρB
   ∘⊃B=∘⊃.~AρB          A8      ⍬ρA=∘⊃A
   ⊃B=⊃.~AρB            3       ∘A=∘B ≡. A=B
```

```
20      #B=0 →. ⊃(~AρB)=⊃B
   #B=0
 → ϴρB=.ϴρ.~AρB          A16     #B=0 →. ~AρB=.~Aρ.~CρB
   ∘⊃B=∘⊃.~AρB           A8      ϴρA=∘⊃A
   ⊃B=⊃.~AρB             3       ∘A=∘B ≡. A=B

21      ×#A=#A
×#A=×~,A          A10     ~,A=#A
   =#,A           D1      ×~A=#A
   =#A            A17     #,A=#A

22      #\A=0
#\A=×~\A          D1      ×~A=#A
   =×0            13      ~\A=0
   =0             21      ×#A=#A

23      ,\A=\A
,\A=.#\Aρ\A       D5      #AρA=,A
   =.0ρ\A         22      #\A=0
   =\\A           D6      0ρA=\A
   =\A            18      \\A=\A

24      ~Aρ\B=.~Aρ∘⊃\B
#\B=0                     22      #\A=0
~Aρ\B=.~Aρ.ϴρ\B           A16     #B=0 →. ~AρB=.~Aρ.~CρB
     =.~Aρ∘⊃\B            A8      ϴρA=∘⊃A

25      \(~AρB)=\B
#\C=0                     22      #\A=0
#\C≤#A                    A12     0≤#A
~\CρB=.~\Cρ.~AρB          A15     #A≤#C →. ~AρB=.~Aρ.~CρB
\B=\.~AρB                 14      ~\AρB=\B

26      \A=\∘⊃A
\∘⊃A=\.ϴρA        A8      ϴρA=∘⊃A
    =\A           25      \(~AρB)=\B

27      ⊃\A=⊃\B ≡. \A=\B    A2
   ⊃\A=⊃\B
 → \∘⊃\A=\∘⊃\B        HYP
   \\A=\\B            26      \A=\∘⊃A
   \A=\B              18      \\A=\A

28      ~AρB=.~Aρ,B
#B≤#B                     A13     #A≤#A
  ≤#,B                    A17     #,A=#A
#B|#B=0                   A18     #A|#A=0
#B|#,B=0                  A17     #,A=#A
~AρB=.~Aρ.~,BρB           A19     #B≤#C ∧. #B|#C=0 .→.
                                      ~AρB=.~Aρ.~CρB
     =.~Aρ.#BρB           A10     ~,A=#A
     =.~Aρ,B              D5      #AρA=,A
```

```
29      \,A=\A
~\Bρ,A=.~\BρA          28      ~AρB=.~Aρ,B
\,A=\A                 14      ~\AρB=\B

30      ,,A=,A
#,Aρ,A=.#Aρ,A          A17     #,A=#A
      =.#AρA           28      ~AρB=.~Aρ,B
,,A=,A                 D5      #AρA=,A

31      ⊃A=⊃,A
ϴρA=.ϴρ,A          28      ~AρB=.~Aρ,B
∘⊃A=∘⊃,A           A8      ϴρA=∘⊃A
⊃A=⊃,A             3       ∘A=∘B ≡. A=B

32      ,∘A=,∘B ≡. A=B     A2
   ,∘A=,∘B
 → ⊃,∘A=⊃,∘B           HYP
   ⊃∘A=⊃∘B             31      ⊃A=⊃,A
   A=B                 A7      ⊃∘A=A

33      #A=0 →. ⊃A=⊃\A
   #A=0
 → ⊃A=⊃,A           31      ⊃A=⊃,A
     =⊃\A           15      #A=0 ≡. ,A=\A

34      #AρB=.#Aρ.~AρB
#A≤#A                  A13     #A≤#A
#,A≤#A                 A17     #,A=#A
~,AρB=.~,Aρ.~AρB       A15     #A≤#C →. ~AρB=.~Aρ.~CρB
#AρB=.#Aρ.~AρB         A10     ~,A=#A

35      ,(~AρB)=.#AρB
,(~AρB)=.#(~AρB)ρ.~AρB         D5      #AρA=,A
        =.#Aρ.~AρB             A9      ~(~AρB)=~A
        =.#AρB                 34      #AρB=.#Aρ.~AρB

36      #A=#B ∧. ~AρB=A .≡. ,A=,B     A21
   #A=#B
   A=.~AρB
 → ,A=.#AρB         35      ,(~AρB)=.#AρB
     =.#BρB         HYP
     =,B            D5      #AρA=,A

37      1ρA=,∘⊃A
1ρA=.#0ρA          D3      #0=1
   =,.ϴρA          35      ,(~AρB)=.#AρB
   =,∘⊃A           A8      ϴρA=∘⊃A

38      #A=1 ≡. ,A=,∘⊃A     A10
   #A=1
 → ,A=.#AρA         D5      #AρA=,A
     =.1ρA          HYP
     =,∘⊃A          37      1ρA=,∘⊃A
```

```
39      #A=0 →. ⊃(~AρB)=⊃\B    SEE 19
   #A=0
 → ⊃(~AρB)=⊃,.~AρB          31      ⊃A=⊃,A
          =⊃.#AρB           35      ,(~AρB)=.#AρB
          =⊃.0ρB            HYP
          =⊃\B              D6      0ρA=\A

40      #(#AρB)=#A
#(~,AρB)=#,A          A9      ~(~AρB)=~A
#(#AρB)=#,A           A10     ~,A=#A
        =#A           A17     #,A=#A

41      #C≠0 →. ~Aρ∘B=.~Aρ.~Cρ∘B
   #C≠0
 → #∘B=1                       6       #∘A=1
     ≤#C                       A14     #A≠0 →. 1≤#A
   #∘B|#C=0                    A20     1|#A=0
   ~Aρ∘B=.~Aρ.~Cρ∘B            A19     #B≤#C ∧. #B|#C=0 .→.
                                           ~AρB=.~Aρ.~CρB

42      ~A=~B ∧. ,A=,B .≡. A=B    A2
   ~A=~B
   ,A=,B
 → ~Aρ,A=.~Bρ,B          HYP
   ~AρA=.~BρB            28      ~AρB=.~Aρ,B
   A=B                   A21     ~AρA=A

43      ~A=0 ≡. A=\A      13
   ~A=0
 → A=.~AρA          A21     ~AρA=A
    =.0ρA           HYP
    =\A             D6      0ρA=\A

44      ~A=1 ≡. A=,∘⊃A    A10
   ~A=1
 → A=.~AρA          A21     ~AρA=A
    =.1ρA           HYP
    =,∘⊃A           37      1ρA=,∘⊃A

45      ~A=Θ ≡. A=∘⊃A     5
   ~A=Θ
 → A=.~AρA          A21     ~AρA=A
    =.ΘρA           HYP
    =∘⊃A            A8      ΘρA=∘⊃A

46      \#A=Θ
\#A=\~,A          A10     ~,A=#A
   =Θ             A22     \~A=Θ

47      ~ΘρA=\A
~ΘρA=.~\~BρA          A22     \~A=Θ
     =\A              14      ~\AρB=\B
```

```
48      ⊃\~A=0
⊃\~A=⊃Θ        A22     \~A=Θ
    =0         A23     ⊃Θ=0

49      ~AρΘ=.~Aρ0
~AρΘ=.~Aρ\~B           A22     \~A=Θ
    =.~Aρ∘⊃\~B         24      ~Aρ\B=.~Aρ∘⊃\B
    =.~Aρ∘0            48      ⊃\~A=0
    =.~Aρ0             A6      #A=∘#A

50      ⊃\∘A=.~Aρ∘⊃\∘⊃A
⊃\∘A=.~Aρ\A             A24     ⊃\∘A=.~Aρ\A
    =.~Aρ∘⊃\A           24      ~Aρ\B=.~Aρ∘⊃\B
    =.~Aρ∘⊃\∘⊃A         26      \A=\∘⊃A

51      \A=\∘.~⊃Aρ∘⊃\∘⊃⊃A
\A=\\A                  18      \\A=\A
  =\∘⊃\∘⊃A               26      \A=\∘⊃A
  =\∘.~⊃Aρ∘⊃\∘⊃⊃A       50      ⊃\∘A=.~Aρ∘⊃\∘⊃A

52      ⊃\A=.~⊃Aρ∘⊃\∘⊃⊃A
⊃\A=⊃\∘⊃A               26      \A=\∘⊃A
   =.~⊃Aρ∘⊃\∘⊃⊃A        50      ⊃\∘A=.~Aρ∘⊃\∘⊃A

53      #A=0 →. A=⊃\∘A
   #A=0
 → A=.~AρA          A21     ~AρA=A
    =.~Aρ\A         17      #A=0 →. ~AρB=.~Aρ\B
    =⊃\∘A           A24     ⊃\∘A=.~Aρ\A

54      ⊃\A=.~⊃Aρ\⊃A    SEE 60
⊃\A=⊃\∘⊃A               26      \A=\∘⊃A
   =.~⊃Aρ\⊃A            A24     ⊃\∘A=.~Aρ\A

55      \⊃\A=\⊃A
\⊃\A=\.~⊃Aρ\⊃A          54      ⊃\A=.~⊃Aρ\⊃A
    =\\⊃A               25      \(~AρB)=\B
    =\⊃A                18      \\A=\A

56      ⊃\∘\A=\A
⊃\∘\A=.~\Aρ\\A          A24     ⊃\∘A=.~Aρ\A
     =\\\A              14      ~\AρB=\B
     =\A                18      \\A=\A

57      ⊃⊃\∘A=⊃\A
#\A=0                   22      #\A=0
⊃⊃\∘A=⊃.~Aρ\A           A24     ⊃\∘A=.~Aρ\A
     =⊃\A               20      #B=0 →. ⊃(~AρB)=⊃B

58      ⊃⊃\A=⊃\⊃A
⊃⊃\A=⊃⊃\∘⊃A             26      \A=\∘⊃A
    =⊃\⊃A               57      ⊃⊃\∘A=⊃\A
```

```
59      ⊃\A=.~⊃Aρ∘⊃⊃\A
⊃\A=.~⊃Aρ\⊃A          54      ⊃\A=.~⊃Aρ\⊃A
   =.~⊃Aρ∘⊃\⊃A        24      ~Aρ\B=.~Aρ∘⊃\B
   =.~⊃Aρ∘⊃⊃\A        58      ⊃⊃\A=⊃\⊃A

60      ~⊃\⊃A=~⊃⊃A    SEE 54
~⊃\⊃A=~⊃\∘⊃⊃A       26      \A=\∘⊃A
     =~⊃⊃A           A24     ⊃\∘A=.~Aρ\A

61      ⊃\∘∘A=∘⊃\∘A
⊃\∘∘A=.~∘Aρ\∘A       A24     ⊃\∘A=.~Aρ\A
     =.ϴρ\∘A         5       ~∘A=ϴ
     =∘⊃\∘A          A8      ϴρA=∘⊃A

62      ~Aρ\B=.~Aρ⊃\∘B
#\B=0                 22      #\A=0
~Aρ\B=.~Aρ.~Bρ\B      A16     #B=0 →. ~AρB=.~Aρ.~CρB
     =.~Aρ⊃\∘B        A24     ⊃\∘A=.~Aρ\A

63      ~Aρ⊃\B=.~Aρ\⊃B
~Aρ⊃\B=.~Aρ⊃\∘⊃B      26      \A=\∘⊃A
      =.~Aρ\⊃B        62      ~Aρ\B=.~Aρ⊃\∘B

64      ⊃\∘(~AρB)=.~Aρ\B
⊃\∘(~AρB)=.~(~AρB)ρ\(~AρB)      A24     ⊃\∘A=.~Aρ\A
         =.~Aρ\(~AρB)           A9      ~(~AρB)=~A
         =.~Aρ\B                25      \(~AρB)=\B

65      ⊃\∘,A=,⊃\∘A
⊃\∘,A=.~,Aρ\,A        A24     ⊃\∘A=.~Aρ\A
     =.~,Aρ\A         29      \,A=\A
     =.~,Aρ⊃\∘A       62      ~Aρ\B=.~Aρ⊃\∘B
     =.#Aρ⊃\∘A        A10     ~,A=#A
     =.#⊃\∘Aρ⊃\∘A     A24     ⊃\∘A=.~Aρ\A
     =,⊃\∘A           D5      #AρA=,A

66      ⊃\∘A=,⊃\∘A ≡. A=,A    65
   ⊃\∘A=,⊃\∘A
 → ~⊃\∘A=~,⊃\∘A       HYP
       =~⊃\∘,A        65      ⊃\∘,A=,⊃\∘A
   ~A=~,A             A24     ⊃\∘A=.~Aρ\A
   ,A=,,A             30      ,,A=,A
   A=,A               42      ~A=~B ∧. ,A=,B .≡. A=B

67      ⊃\∘A=∘⊃⊃\∘A ≡. A=∘⊃A    61
   ⊃\∘A=∘⊃⊃\∘A
 → ~A=~⊃\∘A           A24     ⊃\∘A=.~Aρ\A
     =~∘⊃⊃\∘A        HYP
     =ϴ              5       ~∘A=ϴ
   A=∘⊃A             45      ~A=ϴ ≡. A=∘⊃A
```

```
68      ⊃\∘A=,∘⊃⊃\∘A ≡. A=,∘⊃A      65
   ⊃\∘A=,∘⊃⊃\∘A
 → ~A=~⊃\∘A            A24     ⊃\∘A=.~Aρ\A
     =~,∘⊃⊃\∘A          HYP
     =1                 A10     ~,A=≠A
   A=,∘⊃A               44      ~A=1 ≡. A=,∘⊃A

69      ⊃\∘~A=.~~Aρ0
⊃\∘~A=.~~Aρ\~A         A24     ⊃\∘A=.~Aρ\A
     =.~~Aρ⍬           A22     \~A=⍬
     =.~~Aρ0           49      ~Aρ⍬=.~Aρ0

70      A=⊃A ≡. A=∘A      4
   A=⊃A
 → \∘A=\∘⊃A          HYP
     =\A              26      \A=\∘⊃A
   A=∘A               A25     \A=\∘A →. A=∘A

71      A=⊃\A →. A=∘A
   A=⊃\A
 → ⊃⊃\A=⊃\⊃\A          58      ⊃⊃\A=⊃\⊃A
   ⊃A=A                HYP
   A=∘A                70      A=⊃A ≡. A=∘A

72      ⊃\A=⊃\∘A ≡. A=∘A      A2
   ⊃\A=⊃\∘A
 → \A=\∘A          27      ⊃\A=⊃\B ≡. \A=\B
   A=∘A            A25     \A=\∘A →. A=∘A

73      ⊃\∘A=∘⊃\∘A ≡. A=∘A      61
   ⊃\∘A=∘⊃\∘A
 → ⊃\A=⊃⊃\∘A           57      ⊃⊃\∘A=⊃\A
     =⊃∘⊃\∘A          HYP
     =⊃\∘A            A7      ⊃∘A=A
   A=∘A               72      ⊃\A=⊃\∘A ≡. A=∘A

74      ⊃\∘A=0 ≡. A=∘A ∧. ⊃\A=0      A2
   ⊃\∘A=0
 → 0=⊃\0               48      ⊃\~A=0
   ⊃\∘A=⊃\⊃\∘A         HYP
       =⊃\A            55      \⊃\A=\⊃A
   A=∘A                72      ⊃\A=⊃\∘A ≡. A=∘A

75      A≠,∘A
   A=,∘A
 → ~A=1              A10     ~,A=≠A
   A=,∘⊃A            44      ~A=1 ≡. A=,∘⊃A
   ,∘A=,∘⊃A          2       A=B ∧. B=C .→. A=C
   A=⊃A              32      ,∘A=,∘B ≡. A=B
    =∘A              70      A=⊃A ≡. A=∘A
   ∘A=,∘A            2       A=B ∧. B=C .→. A=C
     ≠,∘A            10      ∘A≠,∘A
```

```
76      ⍬=∘⍬
⊃\⍬=⍬       A26     ⊃\⍬=⍬
⍬=∘⍬        71      A=⊃\A →. A=∘A

77      ⍬≠1
⊃\⍬=⍬         A26     ⊃\⍬=⍬
   ≠0         A27     ⍬≠0
   ≠⊃\1       48      ⊃\~A=0
⍬≠1           A2      A=B →. Δ A=Δ B

78      ~Aρ0≠⍬
   ~Aρ0=⍬
 → ⊃\(~Aρ0)=⊃\0       25      \(~AρB)=\B
             =0       48      ⊃\~A=0
             =⊃\⍬     HYP
             =⍬       A26     ⊃\⍬=⍬
             ≠0       A27     ⍬≠0

79      ⊃\∘A=⍬ ≡. A=∘A ∧. ⊃\A=⍬     A2
   ⊃\∘A=⍬
 → ⍬=⊃\⍬             A26     ⊃\⍬=⍬
   ⊃\∘A=⊃\⊃\∘A       HYP
       =⊃\A          55      \⊃\A=\⊃A
   A=∘A              72      ⊃\A=⊃\∘A ≡. A=∘A

80      ⊃⍜=⍬
⊃⍜=⊃\⍬       D7      \⍬=⍜
  =⍬         A26     ⊃\⍬=⍬

81      ⊃\∘⍜=⍜
⊃\∘\⍜=\⍜      56      ⊃\∘\A=\A
⊃\∘⍜=⍜        D7      \⍬=⍜

82      ∘A≠⍜
#∘A≠0        A11     0≠1
   ≠#⍜       D7      \⍬=⍜
∘A≠⍜         A2      A=B →. Δ A=Δ B

83      ' '=∘' '
⊃\' '=' '      A29     ⊃\' '=' '
' '=∘' '       71      A=⊃\A →. A=∘A

84      ' '≠1
⊃\' '=' '      A29     ⊃\' '=' '
      ≠0       A30     ' '≠0
      ≠⊃\1     48      ⊃\~A=0
' '≠1          A2      A=B →. Δ A=Δ B

85      ⊃\∘A=' ' ≡. A=∘A ∧. ⊃\A=' '     A2
   ⊃\∘A=' '
 → ' '=⊃\' '        A29     ⊃\' '=' '
   ⊃\∘A=⊃\⊃\∘A      HYP
       =⊃\A         55      \⊃\A=\⊃A
   A=∘A             72      ⊃\A=⊃\∘A ≡. A=∘A
```

```
86      ⊃⍬=' '
⊃⍬=⊃\' '        D8      \' '=⍬
  =' '          A29     ⊃\' '=' '

87      ⊃\∘⍬=⍬
⊃\∘\⍬=\⍬        56      ⊃\∘\A=\A
⊃\∘⍬=⍬           D8      \' '=⍬

88      ∘A≠⍬
#∘A≠0          A11     0≠1
   ≠#⍬          D8      \' '=⍬
∘A≠⍬            A2      A=B →. Δ A=Δ B

89      #A≠0 ∧. #B≠0 .→. :Δ(#AρB)=.#Aρ:Δ B
   #A≠0
   #B≠0
 → #,A≠0                          A17     #,A=#A
   :Δ(~,AρB)=.~,Aρ:Δ B            A32     #A≠0 ∧. #B≠0 .→.
                                            :Δ(~AρB)=.~Aρ:Δ B
   :Δ(#AρB)=.#Aρ:Δ B              A10     ~,A=#A


90      #A≠0 →. :Δ∘⊃A=∘⊃:Δ A

   #A≠0
 → #0≠0                           A11     0≠1
   :Δ(~0ρA)=.~0ρ:Δ A              A32     #A≠0 ∧. #B≠0 .→.
                                            :Δ(~AρB)=.~Aρ:Δ B
   :Δ(⍬ρA)=.⍬ρ:Δ A                A5      ~0=⍬
   :Δ∘⊃A=∘⊃:Δ A                   A8      ⍬ρA=∘⊃A

91      :Δ\A=\∘Δ⊃\A
#⍬=0                            D2      #⍬=0
:Δ(~⍬ρA)=.~⍬ρ∘Δ⊃\A              A33     #A=0 →.
                                          :Δ(~AρB)=.~Aρ∘Δ⊃\B
:Δ\A=\∘Δ⊃\A                     47      ~⍬ρA=\A

92      :⊃\A=\⊃A    SEE 96
:⊃\A=\∘⊃⊃\A      91      :Δ\A=\∘Δ⊃\A
     =\⊃\A       26      \A=\∘⊃A
     =\⊃A        55      \⊃\A=\⊃A

93      :Δ\A=\:Δ\A
\∘Δ⊃\A=\\∘Δ⊃\A      18      \\A=\A
:Δ\A=\:Δ\A          91      :Δ\A=\∘Δ⊃\A

94      #A=0 →. :Δ(~AρB)=.~Aρ:Δ\B
   #A=0
 → :Δ(~AρB)=.~Aρ∘Δ⊃\B            A33     #A=0 →.
                                          :Δ(~AρB)=.~Aρ∘Δ⊃\B
          =.~Aρ\∘Δ⊃\B            17      #A=0 →. ~AρB=.~Aρ\B
          =.~Aρ:Δ\B              91      :Δ\A=\∘Δ⊃\A
```

```
95      #A=0 →. :⊃(~AρB)=.~Aρ⊃B
   #A=0
 → :⊃(~AρB)=.~Aρ:⊃\B          94      #A=0 →.
                                       :Δ(~AρB)=.~Aρ:Δ\B
            =.~Aρ\⊃B          92      :⊃\A=\⊃A
            =.~Aρ⊃B           17      #A=0 →. ~AρB=.~Aρ\B

96      #A=0 →. :⊃A=.~Aρ⊃A   SEE 92
   #A=0
 → :⊃A=:⊃.~AρA        A21     ~AρA=A
      =.~Aρ⊃A         95      #A=0 →. :⊃(~AρB)=.~Aρ⊃B

97      #A=0 →. :Δ A=.~Aρ∘Δ⊃A
   #A=0
 → :Δ A=.~Aρ:Δ\A           94      #A=0 →. :Δ(~AρB)=.~Aρ:Δ\B
      =.~Aρ\∘Δ⊃\A          91      :Δ\A=\∘Δ⊃\A
      =.~Aρ\∘Δ⊃A           33      #A=0 →. ⊃A=⊃\A
      =.~Aρ∘Δ⊃A            17      #A=0 →. ~AρB=.~Aρ\B

98      :Δ A=.~Aρ:Δ A
   #A≠0
 → :Δ A=:Δ.~AρA                  A21     ~AρA=A
      =.~Aρ:Δ A                  A32     #A≠0 ∧. #B≠0 .→.
                                           :Δ(~AρB)=.~Aρ:Δ B

   #A=0
 → :Δ A=.~:Δ Aρ:Δ A              A21     ~AρA=A
      =.~:Δ(~AρA)ρ:Δ A           A21     ~AρA=A
      =.~(~Aρ:Δ\A)ρ:Δ A          94      #A=0 →.
                                           :Δ(~AρB)=.~Aρ:Δ\B
      =.~Aρ:Δ A                  A9      ~(~AρB)=~A

99      ~:Δ A=~A
~:Δ A=~.~Aρ:Δ A       98      :Δ A=.~Aρ:Δ A
     =~A              A9      ~(~AρB)=~A

100     :Δ,A=,:Δ A
   #A≠0
 → :Δ,A=:Δ.#AρA              D5      #AρA=,A
      =.#Aρ:Δ A              89      #A≠0 ∧. #B≠0 .→.
                                      :Δ(#AρB)=.#Aρ:Δ B
      =.#:Δ Aρ:Δ A           99      ~:Δ A=~A
      =,:Δ A                 D5      #AρA=,A

   #A=0
 → ,:ΔA=,.~Aρ:Δ\A            94      #A=0 →.
                                      :Δ(~AρB)=.~Aρ:Δ\B
      =.#Aρ:Δ\A              35      ,(~AρB)=.#AρB
      =.0ρ:Δ\A               HYP
      =\:Δ\A                 D6      0ρA=\A
      =:Δ\A                  93      :Δ\A=\:Δ\A
      =:Δ,A                  15      #A=0 ≡. ,A=\A
```

```
101     :Δ A=.~Aρ:Δ,A
:Δ A=.~Aρ:Δ A          98      :Δ A=.~Aρ:Δ A
    =.~Aρ,:Δ A         28      ~AρB=.~Aρ,B
    =.~Aρ:Δ,A          100     :Δ,A=,:Δ A

102     #A=0 →. :Δ\A=\:Δ A
   #A=0
 → #:Δ A=0             99      ~:Δ A=~A
   :Δ,A=,:Δ A          100     :Δ,A=,:Δ A
   :Δ\A=\:Δ A          15      #A=0 ≡. ,A=\A

103     #A=0 →. ⊃:Δ A=⊃\∘Δ⊃A
   #A=0
 → ⊃:Δ A=⊃,:Δ A        31      ⊃A=⊃,A
       =⊃:Δ,A          100     :Δ,A=,:Δ A
       =⊃:Δ\A          15      #A=0 ≡. ,A=\A
       =⊃\∘Δ⊃\A        91      :Δ\A=\∘Δ⊃\A
       =⊃\∘Δ⊃A         33      #A=0 →. ⊃A=⊃\A

104     :Δ⍬=\∘Δ 0
:Δ⍬=:Δ\~A          A22     \~A=⍬
   =\∘Δ⊃\~A        91      :Δ\A=\∘Δ⊃\A
   =\∘Δ 0          48      ⊃\~A=0

105     :Δ⍜=\∘Δ⍟
:Δ⍜=:Δ\⍜           D7      \⍟=⍜
   =\∘Δ⊃\⍜         91      :Δ\A=\∘Δ⊃\A
   =\∘Δ⍟           80      ⊃⍜=⍟

106     :Δ⍥=\∘Δ' '
:Δ⍥=:Δ\⍥           D8      \' '=⍥
   =\∘Δ⊃\⍥         91      :Δ\A=\∘Δ⊃\A
   =\∘Δ' '         86      ⊃⍥=' '

107     :#\A=⍬
:#\A=\∘#⊃\A        91      :Δ\A=\∘Δ⊃\A
    =\#⊃\A         A6      #A=∘#A
    =⍬             46      \#A=⍬

108     #A=0 →. :#(~AρB)=.~Aρ0
   #A=0
 → :#(~AρB)=.~Aρ:#\B        94      #A=0 →.
                                     :Δ(~AρB)=.~Aρ:Δ\B
            =.~Aρ⍬          107     :#\A=⍬
            =.~Aρ0          49      ~Aρ⍬=.~Aρ0
```