IBM Research

A JUNCTION BETWEEN COMPUTER SCIENCE AND CATEGORY THEORY, I: BASIC CONCEPTS AND EXAMPLES (PART I)

J. A. Goguen/J. W. Thatcher/E. G. Wagner/
J. B. Wright

September 11, 1973

RC 4526

pt. 1

Yorktown Heights, New York

San Jose, California

Zurich, Switzerland

A JUNCTION BETWEEN COMPUTER SCIENCE AND CATEGORY THEORY, I:

BASIC CONCEPTS AND EXAMPLES (PART 1)

J. A. Goguen,* J. W. Thatcher, E. G. Wagner, and J. B. Wright

Mathematical Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

ABSTRACT

This is the first part of the first report in a series devoted to exploring the interface or "junction" between computer science and category theory. We expect that both will benefit from the exploration: computer science by a powerful set of tools and a general methodology providing a rigorous and uniform approach to many of its basic concepts, methods, and questions; and category theory by a nontrivial collection of practical applications and illustrations, plus a number of new problems and results. Our present general purposes are to provide a clear, leisurely, and well-illustrated introduction to the basic language of category theory, and to give introductory formulations of some of the computer science topics we treat later in greater depth, including machines, automata, and languages. Later reports will contain the research results which led us to undertake this series.

Section 0 contains a general introduction, a discussion of the special relevance of category theory to computer science, and intuitive interpretations for the key categorical concepts. Section 1 contains a compendium of the background definitions and notation assumed in subsequent sections, especially oriented toward our computer application and category theoretic viewpoint. Section 2 contains the two most basic definitions, of category and functor. There are many examples and frequent intuitive discussions.

RC 4526 (#20134)
September 11, 1973
Computer Sciences

* Current address: Computer Science Department, UCLA, Los Angeles, Calif. 90024
On leave from: University of Chicago, Committee on Information Sciences,
5640 Ellis Avenue, Chicago, Illinois 60637

0. INTRODUCTION

This is the first part of the first report in a series devoted to exploring the interface or "junction" between computer science and category theory. We expect that both will benefit from the exploration: computer science by a powerful set of tools and a general methodology providing a rigorous and uniform approach to many of its basic concepts, methods, and questions; and category theory by a nontrivial collection of practical applications and illustrations, plus a number of new problems and results.

This section outlines the contents of this report and its first few successors. Some reasons for the special relevance of category theory are given in Subsection 0.1, and a discussion of the intuitive meanings of the key categorical concepts follows in Subsection 0.2.

Our general purposes in this report are to provide a clear, leisurely, and well-illustrated introduction to the basic language of category theory, and to give introductory formulations of some of the computer science topics we treat later in greater depth. Hopefully these purposes will reinforce each other so that the reader will be inspired, and able with relative ease, to learn the category theory needed for the deeper applications later. The computer science topics treated in this report include machines, automata, grammars, languages, programs, and tree manipulation.

The immediate successor to this report treats so-called "universal" or "standard" constructions. The next two treat algebraic theories (the first garden, and the second exotic greenhouse varieties) with applications to switching theory, tree manipulation, program schemes, and recursive definitions.

In the fifth report a unified theory of "canonical factorizations" is developed and applied via algebraic theories to explicate certain simple concepts for flow diagram programs and recursion schemes which have not been satisfactorily treated before. For example, the notions of "constant", "variable", and "dummy" or "inessential" variable are discussed. The same theory provides notation independent methods for treating the frontier and interior of a tree, and cuts and ordering of cuts in a tree. The sixth report then characterizes the algebraic theories in which certain canonical factorizations always exist, i.e., in which one can uniformly consider such concepts as indicated above.

The seventh report treats "substitution algebras", which provide a framework for considering Post productions (Post (1943), Rosenbloom (1950)) in arbitrary theories, and many other things, such as an unencumbered treatment of the general substitution of subroutines for specified function symbols in programs. This approach also leads naturally to a generalization for tree automata (Doner (1965, 1970), Thatcher and Wright (1968)) of the conventional monoid of transitions concept.

The eighth report contains a general theory of the kind of "minimal realization" situations which will have already come up several times; e.g., in connection with algebraic functors, canonical factorizations, tree frontiers, and machines. Such situations have a number of interesting general properties.

At this writing, the numbering is not so clear after the eighth report, but we intend to go deeper into tree manipulation, its relationship to operating systems and computer semantics, and into program schemes, recursive definitions, and so on.

This report begins in Section 1 with a compendium of the background definitions and notation which we want to assume in subsequent sections. Section 1 was written especially for our computer application and category theoretic viewpoint, and therefore has some not entirely standard features, including an explicit construction of the finite ordinals, several different arrow notations for functions and relations, a discussion of commutative diagrams, and basic definitions with some examples for ordinary, many-sorted, and partial algebras and their homomorphisms. The reader should at least skim Section 1 for notation, but should probably not try to read it straight through. Rather, he might begin serious reading in Section 2, and fall back on Section 1 if he gets stuck. The main text has explicit references back to Section 1 when this might be especially useful.

This series is intended to be read by one with the general preparation of a computer science graduate student having an interest in theory or of an advanced undergraduate mathematics student with an interest in computer science. For such readers the series is essentially self-contained. We have tried to identify and clarify points which would cause trouble for readers familiar with either computer science or category theory, but not both. In particular, we have noticed that many persons first learning category theory are put off by those pure mathematics expositions which leave out many details and contain little explicit motivation. Consequently, we have tried to be generous with details, especially in the beginning of this report, and have tried throughout to provide at least clear indications of which details are being omitted. Moreover, we have supplied many examples and frequent intuitive discussions (e.g., Subsection 0.2). Of course, not all examples are extremely concrete, since what is relatively concrete to category theory can still seem fairly abstract to computer science. For example, we assume it is not in general necessary to provide a reader with specific amusing examples of the various automata we discuss, but that he can construct his own, perhaps with some reference to a standard text on automaton theory, such as Hopcroft and Ullman (1969).

Hereafter, the n^{th} report in the series is referred to by "ADJ $\rho(n)$", where $\rho(n)$ is the Roman numeral for n. The notation ADJ$\rho(n)/j$ will refer to the j^{th} part of ADJ $\rho(n)$; this is ADJ I/1. "Section $\rho(n).m.k$" refers to the k^{th} Subsection of the m^{th} Section of the n^{th} report. Within the n^{th} report, $\rho(n)$ may be omitted as a prefix. Similarly, "result $\rho(n).m.k$" refers to the k^{th} result in Section $\rho(n).m$, and within ADJ $\rho(n)$, it may be referred to by "result m.k". "Results" include facts, lemmas, corollaries, propositions, and theorems, but not definitions or examples, which are each numbered separately and sequentially, and are referred to in the same style as the results. Sometimes equations and even diagrams are also numbered and referenced in this style. It should be noted that only the most important items are given explicit numbers. In particular, many important, but not crucial, definitions and results occur in the context of examples and do not have numbers of their own. They can be subsequently referred to by the example number. Footnote numbering starts anew in each section.

We shall precede with a star material which is particularly isolated from the rest of the text and can be omitted without loss of continuity for any but other starred material. Stars may precede definitions, results, examples, and even subsections (such as Section 0.1 following).

The bibliography is alphabetical by author and then chronological for a fixed author. References are of the form NAME (DATE) and multiple publications in a given year are distinguished by postfixing a letter to the date; e.g., McCarthy (1963) and McCarthy (1963a). The bibliography for this sequence of reports will be cumulative.

Our typographical conventions are that single solid underlines,_______, denote boldface, while single broken underlines,_ _ _ _ _, denote italics, as used for emphasis. A "wiggly" underline on a letter, as $\underset{\sim}{P}$, denotes a script font, and double underline,$\underline{\underline{\qquad\qquad}}$, denotes some special font reserved for categories. The symbol "□" indicates the termination of a proof or of an example.

*0.1 THE RELEVANCE OF CATEGORY THEORY

Not all areas of computer science should be expected to benefit equally from a category oriented approach. Indeed, some areas of computer science may not be particularly amenable to <u>any</u> sort of mathematical approach at the present time. In discussing the special relevance of category theory, we should first distinguish it from the usefulness of mathematics generally in computer science. It might seem unnecessary to make much of a point of the usefulness of mathematics, but mathematical thinking has recently been attacked in computer science (Wegner (1972)). This is discussed in some detail in Goguen (1972c). The following list of uses should suffice for this report:

1) Mathematical treatment permits precision. For example, a mathematical definition of a programming language should allow two different implementors to be certain that programs run under one implementation will give identical results when run under the other.

2) Mathematical treatments are rigorous. For example, we should feel much more confident using an algorithm that has a mathematical proof of its correctness than one that does not.

3) Mathematical treatment permits generality and unification. For example, the standard theory of finite state automata, especially its decidability results, can be applied to problems in compiler design, switching theory, and intercommunicating processes, without having to be derived over again each time.

4) Mathematical treatment allows impossibility, such as undecidability, to be shown. It is very hard to be convincing about impossibility arguments unless they are very clear and precise indeed. Some useful results of this

kind are recent proofs of the impossibility of automatically carrying out certain kinds of code optimization in compilers. This saves people from wasting time seeking algorithms for these problems.

As part of mathematics, category theory has all the above properties, and we now turn toward the special nature of category theory itself. First, category theory arose as a special part of algebra in response to the need for a conceptual foundation for certain topics in algebraic topology.[1.] It might therefore be expected that the best topics for a categorical approach will be of an algebraic or combinatorial [2.] character, and perhaps within areas needing foundational attention. We feel that many parts of computer science fit this description. In particular, the examples given for the properties 1) - 4) above (program syntax, semantics, and correctness; automata; and undecidability) seem to be of this sort. All except undecidability are discussed somewhere in this series of reports [3.].

One way to look at category theory is as a "language of structure", in somewhat the same sense that set theory is, but operating at a higher level of abstraction and with a different set of primitive concepts. This is discussed in more detail soon. But first, we continue our earlier list

[1.] A detailed discussion of the origin and applications within pure mathematics of category theory would take us very far afield. The interested reader may consult the original papers by Eilenberg and Mac Lane (1942),(1945) and the definitive text of Mac Lane (1971a).

[2.] Algebraic topology was commonly called "combinatorial" topology in its early days, and many parts of it still have a distinctly combinatorial flavor. Moreover, general combinatorial theory itself is now showing signs of merging into algebra, and has even found uses for category theory; see Rota (1969). It has in fact been claimed by Eilenberg (1969) that all of mathematics is becoming more algebraic.

[3.] Undecidability could also be discussed, following Lawvere (1969), but the category theoretic prerequisites involved go beyond what we are currently planning to cover. Topics which it does not presently seem particularly profitable to approach categorically include numerical analysis, job scheduling, memory management, and parts of complexity theory.

of properties, now particularly with particular reference to category theory:

5) A categorical framework provides a powerful guide to research directions. Thus, one feels that certain categorical properties (such as initial or free object) should play a fundamental role, if they exist, so one looks for them, and then tries to use them. An example is our treatment of trees in the second part of this report. On the other hand, if one's ideas do not seem to fit smoothly in a categorical framework, one is encouraged to rethink them, and this often results in greater clarity. Most topics in this series have developed in this way, and could probably be developed even further.

6) Being highly abstract, category theory is well-suited to eliminating distracting detail from highly structured situations. In fact, one tends to study objects through "universal properties" which characterize them, rather than through descriptions of their anatomy. This leads to an independence from the peculiarities of particular notations, which is illustrated by our treatment of trees. Moreover, one often obtains simpler proofs of basic properties after the distracting detail has been eliminated, as again illustrated by our discussion of trees.

7) The abstractness of categorical formulations often permits one to see intriguing similarities between seemingly quite different situations. Thus, a "disjoint union" construction for sets and concatenation of strings are both examples of the "coproduct" construction of category theory.

8) In fact, one can sometimes exploit this abstractness to obtain theories of a generality unattainable in more ordinary mathematics. Thus ADJ VIII treats minimal realization in such a way as to apply to finite automata,

systems of differential equations, and frontiers of trees. Results obtained this way can be entirely new, even in well-studied special cases.

9) The general theory of categories is sufficiently well-developed to provide rather powerful theorems for use in situations which have been formulated in its language. These results (including criteria for the existence of various universal constructions such as limits and free objects) can help one avoid case-by-case analyses, as illustrated in Wagner's (1971a) work on recursive definitions.

Properties 6), 7), and 8) are in a sense only strong forms of special cases of property 3). But category theory really does seem to be more abstract and general than other branches of mathematics, excluding possibly logic and set theory. Abstractness and generality have disadvantages as well as advantages. One disadvantage can be a greater initial difficulty in learning and accepting the material. We try to overcome this with liberal quantities of examples and explanations, hoping that such philosophizing as is now being given will be skipped by those to whom it is repugnant. Another disadvantage (for some people) can be the apparent triviality of many definitions and results once they are finally understood; thus, some mathematicians have dismissed category theory as hopelessly shallow [4.]. We hope to show, more in later reports than the present one, that this is far from true [5.]. Moreover, we tend to be <u>pleased</u> if concepts and proofs look simple, especially when they look simpler than before categoricization.

[4.] Much the same objections were made against Georg Cantor's theory of sets during its infancy.

[5.] Mitchell's "embedding theorem" and Grothendieck's injective envelope theorem (see Mitchell (1965)) are difficult, and important for algebraic topology. See also ADJ V and ADJ VII.

All this is well illustrated by Lawvere's (1963) notation-free categorical approach to algebraic theories and structures, which we introduce in ADJ III.

It is sometimes objected that category theory has failed to produce any new results, and is merely an elaborate and elegant way of presenting already known results. We hope to show this quite untrue [6] by discussing abstract recursive definitions, canonical factorizations, minimal realization theory, program schemes, tree manipulation, and so on in subsequent reports. In fact, some of our topics seem impossible to treat at all except by using category theory. These include abstract system theory (Goguen (1971)), abstract recursive definitions (Wagner (1971, 1971a)) and notation-independent syntax and semantics for programming languages (Goguen (1972a), Wagner (1973)).

There is much more which could be said here (see also Goguen (1972c)), but we hope that the substance of this series of reports will speak more eloquently for itself than we have been able to speak for it here.

[6] It has already been shown untrue, especially in the context of pure mathematics. In fact, most results mentioned in the papers of Footnote 5 were first obtained using category theory. Much of modern algebraic geometry (in the French style, see Grothendieck-Dieudonne (1971)) is conducted entirely in the language of categories, so that <u>all</u> the results are obtained this way.

0.2. INTUITIVE MEANING OF KEY CATEGORICAL CONCEPTS

Some vague idea of the technical content of the key concepts, namely category, functor, natural transformation, and adjoint functor, may be obtained from the following "basic doctrines" (after Goguen (1971)) and accompanying examples. Generally speaking category theorists have been reluctant to commit themselves to such a "dictionary" between technical and intuitive terms, presumably because of its highly nonmathematical character and possibility of being "wrong". However, we shall find these principles very useful in motivating the technical definitions. Moreover, they are an explicit form of the research guidance promised in property 5) of Section 0.1. Obviously, such "doctrines" cannot be proved: they are distillations of our experience in applying category theory; they might even "fail" in some special areas, in the sense that the structures they tell us to look for cannot easily be found. Note that this formulation emphasizes the "normative" character of the doctrines; they tell us things to look for. Also note that we are explicitly not being technically precise in this section; the intention is merely to reinforce the reader's intuition.

Doctrine 1. Any species of mathematical structure is represented by a category, whose objects "are of that structure", and whose morphisms "preserve" it.[7]

[7] The converse of this "doctrine" is not being asserted. In fact, many interesting categories seem to arise in other ways. For example, one can view a monoid as a category (see Example 2.4) by viewing its elements as morphisms. All the paths in a graph serve as morphisms for a category with nodes as objects (see Section 3.2). One can also form categories in which machines appear as morphisms, rather than objects. Thus, sometimes the morphisms are the "interesting" part, and the objects serve to keep track of which morphisms are allowed to be composed. However, there is generally still a sense in which even fairly bland objects take on an "additional structure" by virtue of the morphisms which are present. This "deeper" aspect of the doctrine seems to be difficult to discuss in the absence of the actual definitions and examples given later.

Thus a category, as we shall define it, has two sorts of things in it, objects and morphisms. The category of sets will have sets as objects and set mappings, i.e. functions, as morphisms; the category of groups will have groups as objects and group homomorphisms as morphisms; the category of finite-state machines will have finite-state machines as objects, and machine homomorphisms as morphisms; the category of finite-state machine behaviors has their assignments of output symbols to input strings as objects, and has appropriate structure-preserving maps as morphisms (see Section 2 following for details).

Doctrine 2. Any mathematical scheme for constructing objects of one type from objects of another type as data, is represented by a functor between the corresponding categories.

For example, there is a "behavior" functor, from the category of finite-state machines to the category of finite-state behaviors, corresponding to the construction of a machine's behavior from its state-transition description. There is also a functor the other way: given the input-output behavioral description, it constructs a machine with smallest number of internal states having that behavior, i.e., it constructs a "minimal realization" of the behavior. For a more purely mathematical example, there is a functor from the category of sets to that of groups which constructs the free group on the given set as generators.

It is important to note that functors take morphisms to morphisms, as well as objects to objects. This provides a check on whether the structures of the data and values of the construction have been properly defined. For the morphisms are structure preserving maps, such as subobject inclusions and projections to a quotient, and a reasonable construction should take such relationships between objects in the first category to corresponding (though possibly quite different) relationships between objects in the value category. Metaphorically speaking, functors give, for objects of one species, an image, or transmuted version, of another species.

Doctrine 3. Any natural translation from one construction to another, between the same categories, is represented by a natural transformation between the corresponding functors.

That is, a natural transformation provides a "natural" or consistent, systematic translation from one way of constructing images to another. An important class of natural transformations are the natural isomorphisms (also called natural equivalences) which express structural equivalences of constructions. For example, a number of different explicit set-theoretic free group constructions are known, but they are all naturally isomorphic to one another. So are all minimal realization constructions for finite-state behaviors. Natural transformations which are not isomorphisms tend to be somewhat more exotic, and examples may be found later in the text.

Doctrine 4. Any natural construction is represented by an adjoint functor.

We mean "natural" in the intuitive sense of being "best possible" with respect to some measure and in some context. For example, a minimal realization functor gives a simplest possible state-transition model for a given behavior (see Goguen (1973) or ADJ VIII). In fact, adjointness is a relationship defined between a pair of functors, one "on the left" and one "on the right", each having the other's data category as its value category. Each functor provides a context with respect to which the other is "best possible". It seems to turn out that those constructions which are intuitively "very natural", which do not involve some inherent "arbitrariness", are in fact adjoint to some other functor which provides an appropriate context. For example, any minimal realization functor for finite state behaviors is right adjoint to the behavior functor for reachable machines.

One key fact is that a given functor determines its right (or left) adjoint uniquely up to natural isomorphism, if it exists at all. For example, the minimal-state machine construction is determined up to isomorphism of machines as the right adjoint of the behavior functor. (Note that when we say "*the* minimal-state machine construction", we are in effect already intuitively identifying equivalent functors.) For another example, the free group construction is determined (up to isomorphism) as the left adjoint to the "forgetful" functor from groups to sets which just extracts the underlying set of a group, and "forgets" the rest of the group structure.

These "doctrines" and examples will all be discussed in much more detail later. (Note that adjoints are discussed in ADJ II rather than in this report.) Indeed, one of the purposes of this series of reports is to illustrate the doctrines and show how they provide a methodology for theoretical computer science research. Actually, they are common to all mathematical disciplines, and have significant instances everywhere; see Mac Lane (1971) for further mathematical illustrations. This series of reports will give many more computer science instances, once we have built up the necessary prerequisites.

1. BACKGROUND INFORMATION ON SETS AND ALGEBRAS

This section contains information, primarily notation and definitions, on the following: naive logic and set theory; relations and functions; basic archery, including commutativity; and ordinary, many-sorted, and partial algebras and their homomorphisms. The later topics are taken up again in much greater depth in ADJs III and IV, but are used in a more naive form before then. It is not assumed that the reader is necessarily familiar with these topics.

1.1 LOGIC

Logical equivalence, or "if and only if" is frequently abbreviated by "iff", and occasionally by the double-headed double arrow " <===> ". Implication is indicated by the right-headed double arrow " ===>", and "A ===> B" is false iff the statement (or proposition) A is true but the statement B is false; this is the basis for "proof by contradiction", in which "A ===> B" is proved by showing it can never happen that A is true and B is false. To show "A <===> B" one commonly proves "A ===> B" and "A <=== B" (that is, "B ===> A") separately, and in such proofs it is convenient to label the two parts by " ===>: " and " <===: " respectively; these parts are the "sufficiency" and "necessity" for B of the condition A. The termination of a proof is indicated by " □ " as a sort of punctuation mark. It is not quite equivalent to "Q.E.D." since we may omit some of the concluding steps of a proof; if the proof is entirely absent, we put " □ " at the end of the result's statement. This symbol is also used to indicate the termination of a (numbered formal) example.

The following symbols will sometimes be used, essentially as abbreviations: $\top$ for "true"; $\bot$ for "false"; $\wedge$ for "and"; $\vee$ for "or"; — for "not"; $\forall$ for "for all"; $\exists$ for "there exists"; and $\exists!$ for "there exists a unique". For example, $(\forall a)(\exists! b)(A(a) \wedge B(a,b))$ should be read "for all a there is a unique b such that A(a) is true and B(a,b) is true", where A(a) and B(a,b) are some statements or propositions whose truth value depends on a, and on a and b, respectively. Such notation is avoided wherever possible.

Let A(n) be a statement depending on a nonnegative integer n (such as "$\sum_{i=0}^{n} i^2 = n(n+1)/2$").. Then a "proof by induction" of the validity of A(n) consists of the following two steps:

1) show that $A(1)$ (or possibly $A(0)$) is true; and

2) show that if $A(n)$ is true then $A(n+1)$ is true -- this is called the "inductive step".

The conclusion is that $A(n)$ is true for all integers greater than or equal to the initial value. This "principle of mathematical induction" can be taken as asserting that there is one, and only one, true-false-valued function A satisfying conditions (1) and (2) above: namely the constant true valued function, $A(n) = \top$ for all n. This will show that "mathematical induction" is a special case of the notion of "inductive (or recursive)definition" to be discussed later; somewhat later still, we give a very simple category theoretic version of these notions, using initial objects. This will give a very simple treatment of trees, their properties, and applications.

1.2 SETS

We shall operate on an entirely intuitive or "naive" level in this section, but in such a way that everything could be readily formalized in any of a number standard systems, such as Gödel-Bernays or Zermello-Fraenkel (see Cohen (1966) for a basic exposition; Mac Lane (1971b) and Feferman (1969) discuss ways to treat the more complex problems which come up in foundational problems for category theory); this is pretty much the usual practice today. The basic concept of a "set" is synonymous in ordinary mathematical English with "collection", "aggregate", and so on; the members of sets are also called "elements". The basic relationship is membership, "$x \in X$" for "x is an element of X"; we write "$x \notin X$" for "x is not an element of X". One exhibits (small) finite sets by listing the elements between {,} braces; thus the set containing elements 0,1,5 is denoted $\{0,1,5\}$. Note that $\{0,1,5\} = \{5,1,0\} = \{0,1,5,1\}$, etc.; i.e., variations in order, and repetitions, do not effect the set defined. This is because by definition two sets are equal iff they have the same elements; i.e., $X = Y$ iff for all x, $x \in X \Longleftrightarrow x \in Y$. There is one and only one empty set, having no elements: it is denoted $\emptyset$ (incidentally, $\emptyset$ is not the Greek letter "phi", but perhaps a Swedish letter, pronounced something like "eh"). Note that $\emptyset \neq \{\emptyset\}$.

X is a subset of Y, written $X \subseteq Y$, iff for all x, $x \in X \Longrightarrow x \in Y$. Then for any X, $\emptyset \subseteq X$ and $X \subseteq X$. The set of all subsets of a set X is here denoted 2^X or $\underset{\sim}{P}(A)$ and is often called the power set of X. $\underset{\sim}{P}_F(X)$ will denote the set of finite subsets of X. If X is a finite set with n elements, then $\underset{\sim}{P}(X) = 2^X$ has exactly 2^n elements. Another way to define sets is "intensionly", as all the elements satisfying some proposition $A(x)$; we write $\{x | A(x)\}$. Thus $\underset{\sim}{P}(X) = \{Y | Y \subseteq X\}$. If X, Y are sets, their union $\{x | x \in X$ or $x \in Y\}$ is denoted $X \cup Y$, their intersection $\{x | x \in X$ and $x \in Y\}$ is denoted $X \cap Y$, and their difference $\{x | x \in X$ and $x \notin Y\}$ is denoted $X-Y$. There are a number of laws satisfied by these operations. Many of them are summarized by saying $\underset{\sim}{P}(X)$ is a Boolean algebra (this is defined in Example 1.11, in Section 1.5).

Ordered pairs are characterized by the condition that $\langle x,y \rangle = \langle x',y' \rangle$ iff $x = x'$ and $y = y'$. One set-theoretic definition for $\langle x,y \rangle$ is $\{\{x\},\{x,y\}\}$; see Halmos (1960) for details. Hereafter we just assume some construction satisfying the characteristic property. The product of sets X,Y is $\{\langle x,y \rangle | x \in X$ and $y \in Y\}$, denoted $X \times Y$, and sometimes called Cartesian product after Descartes' use of pairs in coordinate plane analytic geometry. An ordered triple $\langle x,y,z \rangle$ can be defined using ordered pairs to be $\langle x,\langle y,z \rangle\rangle$; similarly for quadruples, etc., if desired.

We will repeatedly use the "ordinal notation" convention that $n = \{0,1,\cdots,n-1\}$, for n a nonnegative number. This convention appears very natural [1] when the nonnegative numbers are set-theoretically constructed "from nothing" as follows: define 0 to be the empty set $\emptyset$; and if n is a nonnegative number, define its successor n^+ to be the set $n \cup \{n\}$. Then for example, $1 = \{\emptyset\} = \{0\}, 2 = \{\emptyset,\{\emptyset\}\} = \{0,1\}$, and $3 = \{\emptyset,\{\emptyset\},\{\emptyset,\{\emptyset\}\}\} = \{0,1,2\}$; in general, $n = \{0,1,\cdots,n-1\}$. Let ω denote the set of all these nonnegative numbers. Note that for $m,n\in\omega$, $m<n$ in the conventional numerical sense, iff $m\in n$ as sets. Mathematical induction now appears as a valid mathematical theorem about the set ω: for $S \subseteq \omega$, if $0\in S$, and if $n\in S$ implies $n^+\in S$, then $S = \omega$.

1.3 RELATIONS AND FUNCTIONS

It is customary to define a relation to be a subset R of a product $A \times B$ of some sets A and B. This is rather unsatisfactory, in that given the set R of ordered pairs, one will in general be quite unable to figure out what A and B were; the extreme case is when $R = \emptyset$ since then, A and B could be any sets whatsoever, since $\emptyset \subseteq A \times B$ for all sets A,B. We therefore define a relation from A to B to be a triple $\langle A,R,B\rangle$, where $R \subseteq A \times B$ is called the graph of the relation, and we sometimes use the notations $A \xrightarrow{R} B$ or $R:A \to B$, the latter being typographically easier even though the letters are out of order. Call A the source and B the target of the relation $\langle A,R,B\rangle$. Actually, now that we have made precise what we mean by relation, we shall feel free to denote $\langle A,R,B\rangle$ by just R and to call R the relation, when this will not be too confusing. We also write aRb for $\langle a,b\rangle\in R$.

[1] From a more algebraic (and categorical) point of view, it is more natural to describe the nonnegative integers as an algebra $\underline{N}$ having a distinguished constant element $0\in\underline{N}$, and a successor operation $+:\underline{N} \to \underline{N}$. This point of view is explained further, and used, in Part 2 of this report.

An *inclusion relation* is an example of a relation for which it is essential to keep track of source and target. In fact, let E be the subset of the set ω consisting of the even numbers, and let D denote (for the moment) the set $\{\langle n,n\rangle | n\in E\}$ of equal pairs of even numbers. Then the "inclusion relation" $\subseteq : E \to \omega$ and the "identity relation" $E \to E$ have the same graph D, and there is no way to distinguish between them without explicitly knowing their targets. That is, the inclusion is the triple $\langle E,D,\omega\rangle$, and the identity on E is the triple $\langle E,D,E\rangle$. Hereafter, the *identity relation* $\langle S,\{\langle s,s\rangle | s\in S\},S\rangle$ on a set S will be denoted 1_S, and its graph $\{\langle s,s\rangle | s\in S\}$ will be called the *diagonal* set on S (so named for the "way it would look if plotted").

A relation often considered is $\le$, "less than or equals", on ω (note that to say "R is a relation *on* S" means we have $R:S \to S$), defined by $m \le n$ iff $m \subseteq n$ as sets; we could quite reasonably have used the notation $\subseteq$ if $\le$ were not completely standard. Another relation on ω is "successor", with graph $\{\langle n,n\cup\{n\}\rangle | n\in\omega\}$. This relation is actually a *function*, that is, it is a relation $R:A \to B$ satisfying the *functional condition*

for each $a\in A$ there is a *unique* $b\in B$ such that aRb.

For functions, we use the special notation aR for the unique "image" element $b\in B$ R-related to $a\in A$, that is, with $\langle a,b\rangle\in R$; but sometimes we may use the more conventional notation $R(a)$. Often in defining a function it is convenient to give a formula for computing a unique target element from a given source element, as for successor with $n^+ = n\cup\{n\}$. A convenient notation here is an arrow with a "foot", for example, $n \mapsto n\cup\{n\}$, to define $^+ : \omega \to \omega$ (note the slight abuse of functional notation with n^+ instead of (say) $n+$ or $+(n)$). Inclusion and identity relations are always functions.

A "function of two variables" is a function whose source is a product set. Thus $R:X \times Y \to Z$ is a function of "variables" x,y in the sets X,Y, and one writes $\langle x,y\rangle R$ or $(x,y)R$ or $R(x,y)$ for the image element in the target. There is a standard way to obtain a function of one variable from a function of two variables. For example, from $R:X \times Y \to Z$ and $y \in Y$ one obtains $(\_,y)R:X \to Z$ defined by $x \longrightarrow \langle x,y\rangle R$ or, with our functional notation, $x(\_,y)R = (x,y)R$. Similarly from R and $x \in X$ one obtains $(x,\_)R:Y \to Z$. We say these functions arise from R by "fixing y" or by "fixing x" respectively. These considerations apply equally well to relations. Thus if $R:X \times Y \to Z$ is a relation and $y \in Y$, then $(\_,y)R:X \to Z$ is the relation with graph $\{\langle x,z\rangle \mid \langle x,y,z\rangle \in R\}$. Similarly we can treat functions and relations of more "variables"; e.g., given $R:X \times Y \times Z \to W$, then $(\_,y,\_)R:X \times Z \to W$ is the relation or function obtained by "fixing y". The "blank" symbol "_" stands for the arguments allowed to vary. Thus we avoid here the usual notions $R(x)$, $R(x,y)$, etc. for the functions themselves. There are still pitfalls in these notations, especially the blank, but they are nicely overcome in the context of syntactic *theories* (see ADJ III) in which variables are formal mathematical entities.

A *partial function* is a relation $R:A \to B$ satisfying the following somewhat weaker condition

for each $a \in A$ there is *at most* one $b \in B$ such that aRb.

It is sometimes suggestive to use the notation $R:A \mathrel{-\!\circ\!\!\rightarrow} B$ for such a function with "a hole" in it, and to call the set where R is defined its preimage or *set of definition*. More generally, the *preimage* of a relation $R:A \to B$ is $\{a \in A \mid \text{there is some } b \in B \text{ such that } aRb\}$; and dually, the *image* of a relation $R:A \to B$ is $AR = \{b \in B \mid \text{there is some } a \in A \text{ such that } aRb\}$. Note that for functions, preimage or set of definition, and source are equal. We avoid the word "range" which might ambiguously refer either to target or image. Sometimes the notation $R(A)$ is used for image, but we prefer AR. Also, for $A_0 \subseteq A$, let $A_0R = \{b \in B \mid aRB \text{ for some } a \in A_0\}$, called the *image* of A_0 *under* R.

If $R:A \to B$ is a relation and $A_0 \subseteq A$, $A_0 \upharpoonleft R : A_0 \to B$ is the *restriction* of R to A_0, a relation with source A_0 and graph $A_0 \upharpoonleft R = \{\langle a,b\rangle \in R \mid a \in A_0\} = R \cap (a_0 \times B)$ (the usual notation is unfortunately $R \upharpoonright A_0$). If R is a function or partial function, so is $A_0 \upharpoonleft R$. Sometimes one wants to speak of the *corestriction* of R to $B_0 \subseteq B$, which will have graph $R \cap (A \times B_0)$, source A, and target B_0; there is no standard notation for this, and though we suggest $R \upharpoonright B_0$, we will not often use it. For $A_0 \subseteq A$ and $B_0 \subseteq B$, one can also consider the *birestriction* $A_0 \upharpoonleft R \upharpoonright B_0 : A_0 \to B_0$ with graph $R \cap (A_0 \times B_0)$.

There are some useful special types of function and corresponding arrow notations. A function $R:A \to B$ is *injective* iff for all $a, a' \in A$, if $aR = a'R$ then $a = a'$; R is *surjective* iff for all $b \in B$, there is some $a \in A$ such that $b = aR$; and R is *bijective*, or is an *isomorphism*, iff it is both injective and surjective. The older language uses one-to-one (or one-one) and onto for injective and surjective; but one-to-one might be confused with the "one-to-one" correspondence which occurs with a bijection. We will sometimes write $R:A \rightarrowtail B$ if R is injective, $R:A \hookrightarrow B$ if R is an inclusion, $R:A \twoheadrightarrow B$ if R is surjective, and $R:A \overset{\sim}{=} B$ if R is bijective. We also say sets A,B are *equivalent* or *isomorphic* if there is a bijection $R:A \overset{\sim}{=} B$, and sometimes we write just $A \overset{\sim}{=} B$. The ideas of injection and surjection are also useful for partial functions, but we will not speak of bijections in this context.

The word "equivalence" suggests the connection with cardinality. A set A is finite iff there is some n such that $A \cong n$, and in this case we also write $\#A = n$ and say the cardinality of A is n. Similarly, A is countably infinite iff $A \cong \omega$, and we write $\#A = \omega$. Finally, A is countable iff finite or countably infinite, and is uncountable iff not countable.

The converse of a relation $R:A \to B$ is $R^{\vee}:B \to A$ where $R^{\vee} = \{\langle b,a\rangle \mid \langle a,b\rangle \in R\}$. Note that $R^{\vee\vee} = R$. For $B_0 \subseteq B$, the inverse image of B_0 under R is the image $B_0R^{\vee}$ of B_0 under the converse; the notation B_0R^{-1} is also common. In set terms, $B_0R^{\vee} = \{a \mid$ there exists some $b \in B_0$ such that $\langle a,b\rangle \in R\}$.

Perhaps the most important operation on relations is composition. Given $R:A \to B$ and $S:B \to C$, their composition $RS:A \to C$ is defined by $aRSc$ iff there is some $b \in B$ such that aRb and bSc. The composite RS is also called the Peirce product [2] of R and S, and also denoted $R \circ S$. Note that this order for the factors R,S of the composite RS is opposite to the usual one. However, we claim it is more natural, as the sequence $aRbSc$ suggests, and more consistent when one is drawing diagrams, as below

$$\begin{tikzcd}
A \arrow[rr, "RS"] \arrow[dr, "R"'] & & C \\
& B \arrow[ur, "S"'] &
\end{tikzcd} .$$

We say a diagram of sets and relations commutes iff along any two paths with the same beginning and end, the composites (in order) of the relations found on the edges are equal. Thus the above diagram just says $R \circ S = RS$, and $RS = TU$ is expressed by commutativity of

$$\begin{tikzcd}
{} \arrow[r, "R"] \arrow[d, "T"'] & {} \arrow[d, "S"] \\
{} \arrow[r, "U"'] & {}
\end{tikzcd}$$

[2] After Charles Sanders Peirce (pronounced "purse") who seems to have been the first to use this notion, as part of his pioneering theory of relations; See Peirce (1931).

(note that sometimes we omit the sets on the vertices). It is easily shown that composition of relations is associative, in the sense that for $R:A \to B$, $S:B \to C$, and $T:C \to D$, we have

$$(*) \qquad R(ST) = (RS)T.$$

This permits us to omit parentheses and write RST without ambiquity.

Furthermore, for $R:A \to B$ and $S:B \to C$,

$$(**) \qquad 1_B S = S \quad \text{and} \quad R1_B = R.$$

These equations further justify our calling 1_B the "identity relation" on B; for these are the *identity laws* for 1_B.

The equations (*) and (**) also hold when R,S and T are functions or partial functions. These facts will give us categories with relations, functions, and partial functions as morphisms, once we have defined the notion of category. Note that the composite of two functions is a function, and of two partial functions is a partial function.

A technique of great utility in this and subsequent reports is "diagram chasing" in which one deduces the commutativity of some parts of a diagram from that of others. One of the simplest instances is "pasting together" two commutative squares to obtain a third; we shall use it often in the following.

Proposition 1.1. If the first two squares below commute, then so does the third.

$$\begin{tikzcd} A \arrow[r, "f"] \arrow[d, "a"'] & B \arrow[d, "b"] \\ A' \arrow[r, "g"'] & B' \end{tikzcd} \qquad \begin{tikzcd} B \arrow[r, "f'"] \arrow[d, "b"'] & C \arrow[d, "c"] \\ B' \arrow[r, "g'"'] & C' \end{tikzcd} \qquad \begin{tikzcd} A \arrow[r, "ff'"] \arrow[d, "a"'] & C \arrow[d, "c"] \\ A' \arrow[r, "gg'"'] & C' \end{tikzcd}$$

Proof. This proof is easily given "dynamically" on a blackboard, but we shall have to use "static" equations here. To show that $a(gg') = (ff')c$, we show that each side equals fbg'. Postmultiplying the first square by g' gives $(ag)g' = (fb)g'$; and premultiplying the second by f gives $f(f'c) = f(bg')$. Associativity now completes the proof. The diagram below illustrates this. □

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[d, "a"'] & B \arrow[r, "f'"] \arrow[d, "b"] & C \arrow[d, "c"] \\
A' \arrow[r, "g"] & B' \arrow[r, "g'"] & C'
\end{tikzcd}$$

The following easily proved result is an amusing combination of previously given ideas.

Proposition 1.2. Let $A_0 \subseteq A$ and $B_0 \subseteq B$ be sets and subsets, and let $f:A \to B$ be a function. Then there is a function $g:A_0 \to B_0$ such that the square

$$\begin{tikzcd}
A_0 \arrow[r, "g"] \arrow[d, hook] & B_0 \arrow[d, hook] \\
A \arrow[r, "f"] & B
\end{tikzcd}$$

commutes iff $A_0 f \subseteq B_0$, and in this case g is the birestriction $A_0|f|B_0$, i.e., the factorization through B_0 of the restriction of f to A_0. □

The following special properties of functions are particularly worth noting, and are fairly easily proved.

Proposition 1.3.

(1.) A function $f:A \to B$ with $A \neq \emptyset$ is injective iff there is a function $g:B \to A$ such that $fg = 1_A$;

(2.) a function $f:A \to B$ is surjective iff there is a function $g:B \to A$ such that $gf = 1_B$;

(3.) a function $f:A \to B$ is bijective iff there is a function $g:B \to A$ such that $fg = 1_A$ and $gf = 1_B$;

(4.) the function g in case (3.)is uniquely determined by f (if it exists), and is called the <u>inverse</u> of f, denoted f^{-1}; moreover, $f^{-1} = f^{\vee}$, whenever f^{-1} exists;

(5.) a function $f:A \to B$ is injective iff for all $h, h':C \to A$, $hf = h'f$ implies $h = h'$; and

(6.) a function $f:A \to B$ is surjective iff for all $h, h':B \to C$, $fh = fh'$ implies $h = h'$. □

The properties occurring in (5.) and (6.) are called "monic" and "epic" respectively, and will be generalized in Section 4.

If A and B are sets, we let B^A denote the set of all functions from A to B. We may call B^A the <u>exponential</u> of B by A. This notation is convenient because for A and B finite,[3] $\#(B^A) = (\#B)^{\#A}$. Moreover, although 2^A seems to admit two interpretations, as $\{0,1\}^A = \{f:A \to 2\}$ and as the power set of A, these two sets are actually isomorphic; yet because of this potential (but fairly harmless) ambiguity, we prefer the notation $\underset{\sim}{P}(A)$ for the power set. (Note also that $A\underset{\sim}{P}$ would be more consistent with our notation for functions!)

For any set B, $B^{\emptyset} = B^0$ has exactly one member $<\emptyset,\emptyset,B>$, because $\emptyset$ is the only (functional) subset of $\emptyset \times B$. This function will be denoted λ_B. As we have seen above, $\lambda_B \neq \lambda_A$ when $A \neq B$, but when it is not too confusing we can identify these empty functions and write $B^0 = A^0 = \{\lambda\}$.

[3] This assumes the perhaps unfamiliar convention that $0^0 = 1$, which is consistent with the usual laws for exponentiation of nonnegative integers, that $(a \cdot b)^c = a^c \cdot b^c$ and $a^{b \cdot c} = (a^b)^c$.

The following special cases of exponential are especially important: if A is a set, A^n is the set of all functions from n to A, or of <u>strings</u> or <u>sequences of length</u> n in A; and A^ω is the set of all functions from ω to A, or of (countably <u>infinite</u>) <u>sequences</u> in A. A^n is isomorphic to the set of all n-tuples in A, and in particular, $A^2 \cong A \times A$, so that the notation $a = \langle a_0, a_1, \cdots, a_{n-1} \rangle$ is reasonable for $a \epsilon A^n$, where $a_i = ia$ for $i \epsilon n$. For $a \epsilon A^\omega$ we similarly write a_i for ia (where $i \epsilon \omega$), and we may write $a = \langle a_i \rangle_{i \epsilon \omega}$, or even $a = \langle a_i \rangle$. Other useful notations are $a_0, a_1, \cdots, a_{n-1}$ or even $a_0 a_1 \cdots a_{n-1}$ for $\langle a_i \rangle_{i \epsilon n}$ and $a_0, a_1, a_2, \cdots$ or $a_0 a_1 a_2 \cdots$ for $\langle a_i \rangle_{i \epsilon \omega}$. For $n = 0$, A^0 consists of the empty function, λ_A, which in this context is called the <u>empty string</u> and denoted λ or even $\langle\ \rangle$.

Several further properties of strings and sequences are discussed in Section I.5 from a categorical point of view. It is also sometimes useful to consider "arbitrary" potentially uncountable sequences in A, as elements of A^I, for I an arbitrary "index set". We still write a_i for ia, when $i \epsilon I$, and even $a = \langle a_i \rangle_{i \epsilon I}$.

1.4 MORE ON SETS AND RELATIONS

Many of the operations given on pairs of sets in Section 1.2 can be extended to potentially infinite families of sets. Thus, let $\underset{\sim}{A}$ be a set each of whose elements is a set, and define: the *union over* $\underset{\sim}{A}$, $\cup\underset{\sim}{A}$, to be $\{a \mid a\in A$ for some $A\in\underset{\sim}{A}\}$; and the *intersection over* $\underset{\sim}{A}$, $\cap\underset{\sim}{A}$, to be $\{a \mid a\in A$ for all $A\in\underset{\sim}{A}\}$. We also speak of *indexed families of sets*, i.e. of functions $A:I \to \underset{\sim}{A}$, where $\underset{\sim}{A}$ is a set of sets. Instead of the notation $\langle A_i\rangle_{i\in I}$ one sometimes sees $\{A_i \mid i\in I\}$, which is literally the image of A in $\underset{\sim}{A}$, or $\{A_i\}_{i\in I}$, or perhaps a phrase like "A_i, for $i\in I$", or "a family A_i of sets". In any case, it is often convenient not to mention $\underset{\sim}{A}$ explicitly, since the relevant part of it can be reconstructed as $\{A_i \mid i\in I\}$. For a family A_i of sets, one defines: the *union* $\cup_{i\in I}A_i$ to be $\{a \mid a\in A_i$ for some $i\in I\}$, i.e., $\cup_{i\in I}A_i = \cup\{A_i \mid i\in I\}$ as defined above; and the *intersection* $\cap_{i\in I}A_i$ to be $\{a \mid a\in A_i$ for all $i\in I\}$. When I is clear from context, we write just $\cup_i A_i$ and $\cap_i A_i$. The *product* $\prod_{i\in I}A_i$ of a family of sets A_i is defined to be $\{a:I \to \cup_{i\in I}A_i \mid a_i\in A_i$ for all $i\in I\}$; that is, $\prod_i A_i$ consists of all sequences over I in $\cup_{i\in I}A_i$ such that a_i is in A_i. We have already seen the special case of the product when all the A_i are the same, say A, as in this case $\prod_{i\in I}A_i = A^I$. The same general sort of laws (associative, commutative, etc.; see Section I.1.5) hold for these infinite operations as for the corresponding finite ones, and we shall not discuss them explicitly here.

One somewhat more esoteric operation we want to consider is the *coproduct*, sometimes called the *disjoint union*, of a family of sets A_i, $\coprod_{i\in I}A_i = \{\langle i,a\rangle \mid a\in A_i$ and $i\in I\}$. Here each element $a\in A_i$ is "tagged" or "labelled" with i, so that one can tell where it came from, even when the family $\langle A_i \rangle_{i \in I}$ is not disjoint. Of course, the family $\langle \{\langle i,a\rangle \mid a \in A_i\}\rangle_{i\in I}$ is disjoint, and its union is $\bigsqcup_i A_i$.

We shall also need some further concepts concerning relations. For this purpose, all relations are on (i.e., have both source and target) a fixed set A, so we need not make much use of the arrow notations. A relation R on A is reflexive iff $1_A \subseteq R$, is symmetric iff $R = R^\vee$, and is transitive iff $RR \subseteq R$. More concretely, these conditions are equivalent to: for all $a \in A$, aRa; for all $a,b \in A$, aRb iff bRa; for all $a,b,c \in A$, if aRb and bRc, then aRc. A relation R on A is an equivalence relation iff it is reflexive, symmetric, and transitive. If E is an equivalence relation on A and $a \in A$, let a/E, or possibly $[a]_E$, denote the set $\{b \in A \mid aEb\}$ of all elements E-related to a; and let A/E denote the set of all such equivalence classes, i.e., $A/E = \{a/E \mid a \in A\}$. A partition of a set A is a set $\underset{\sim}{A}$ of nonempty subsets of A which are pairwise disjoint (i.e., if $A, A' \in \underset{\sim}{A}$ and $A \neq A'$, then $A \cap A' = \emptyset$) and collectively exhaustive (i.e., $\cup \underset{\sim}{A} = A$).

Proposition 1.4. If E is an equivalence relation on A, then A/E is a partition of A. Conversely, if $\underset{\sim}{A}$ is a partition of A, then the relation E defined on A by aEb iff there is some $A \in \underset{\sim}{A}$ such that $a,b \in A$, is an equivalence relation. □

A relation R on A is anti-symmetric iff $R \cap R^\vee = 1_A$, i.e., iff for all $a,b \in A$ aRb and bRa implies $a = b$. A partial ordering relation R on A is an anti-symmetric, reflexive, and transitive relation on A, and the pair $\langle A,R\rangle$, with $R \subseteq A \times A$, is then called a partially ordered set, or poset. Similarly, a quasi-ordering relation R on A is a reflexive, transitive relation on A, and the pair <A,R> is called a quasi-ordered set, or quoset.

Proposition 1.5. Each relation R on A is contained in a least quasi-ordering R^*, and in a least transitive relation R^+, called (respectively) the reflexive and transitive, and the transitive, closures of R. Moreover, any relation is contained in a least equivalence relation.

Proof. For any relation R on A define successive "powers" of R inductively by $R^0 = 1_A$ and $R^{k+1} = R^kR$ (composition). Then $R^+ = \bigcup_{0<k\epsilon\omega} R^k$ is the transitive closure of R; $R^* = R^+ \cup \{1_A\}$ is the reflexive and transitive closure of R; and $(R \cup R^\vee)^*$ is the least equivalence relation containing R. We can show that R^+ is transitive by calculating that $R^+R^+ = \bigcup_{0<j,k\epsilon\omega} R^jR^k \subseteq R^+$. Now we show that if $R \subseteq T$ and T is transitive, then $R^+ \subseteq T$. In fact, we show by induction that $R^k \subseteq T$ for all $0<k\epsilon\omega$. By assumption $R \subseteq T$; now assuming $R^k \subseteq T$, we carry out the inductive step, showing $R^{k+1} \subseteq T$. Since $R^k \subseteq T$ and $R \subseteq T$, we have $R^kR \subseteq TT$, and since T is transitive, $TT \subseteq T$, so that $R^{k+1} \subseteq T$. □

1.5 ALGEBRAIC STRUCTURE

An operation on a set A is a function $\phi : A^n \to A$ for some $n\epsilon\omega$ called the arity of ϕ. A binary (or 2-ary) operation is a function $A^2 \to A$, a unary or monadic (or 1-ary) operation is a function $A \to A$ and a trinary (3-ary) operation is a function $A^3 \to A$. A zeorary (or 0-ary) or nullary operation is a function $A^0 \to A$. 0-ary operations correspond to "distinguished" elements of A because $A^0 = \{\lambda_A\}$, and functions $\phi:\{\lambda_A\} \to A$ are in bijective correspondence with elements of A under the function $\phi \mapsto \lambda_A\phi\epsilon A$. The arity of an operation is thus the number of arguments it needs to be given to produce a value; unary operations need one, while nullary operations need none (and always give the same constant value).

An algebra consists of a set with a family of operations. To consider collections of algebras all of the same species (such as groups, or rings, or lattices), we need a precise notion of "species". This can be done by considering families of operations over a fixed index set, with a fixed arity assigned to each index. Thus a ranked symbol (or operator) set (or sometimes, domain or alphabet) is a sequence $<\Sigma_n>_{n\epsilon\omega}$ of subsets of an alphabet or symbol set Σ; the rank or arity of $\phi\epsilon\Sigma_n$ is n. Often we let just Σ refer to the whole ranked set or sequence. Note that a single symbol $\phi\epsilon \bigcup_i\Sigma_i$ may have more than one arity. This happens even for the usual operations on ω, where "minus" has both rank one and two; e.g., both $-n$ and $m-n$ are allowed.

A Σ-algebra is a pair $<A,\alpha>$, where α is a sequence $<\alpha_n>_{n\epsilon\omega}$ of functions $\alpha_n:\Sigma_n \to A^{A^n}$ assigning n-ary operations on A to the n-ary symbols. A is called the underlying set or carrier of the algebra $<A,\alpha>$, and when it is not confusing we leave the operations implicit and speak of "the algebra A". If there are only a few operations and it is desired to display them with A, say $\Sigma_0 = \{0,1\}$, $\Sigma_2 = \{+,\times\}$, we indicate the algebra in the style $<A,0,1,+,\times>$, or possibly $<A;0,1;+,\times>$ for added emphasis. Note that it is customary to use the same symbol for the operation itself as the corresponding abstract symbol in Σ.

Let $\langle A,\alpha\rangle$, $\langle A',\alpha'\rangle$ be Σ-algebras; then a Σ-*(algebra)* *homomorphism* $h{:}\langle A,\alpha\rangle \to \langle A',\alpha'\rangle$ is a set function $h{:}\ A \to A'$ such that for each n and $\phi\epsilon\Sigma_n$ the following diagram commutes,

$$\begin{tikzcd}
A^n \arrow[r, "\phi\alpha_n"] \arrow[d, "h^n"] & A \arrow[d, "h"] \\
A'^n \arrow[r, "\phi\alpha'_n"] & A'
\end{tikzcd} ,$$

i.e., $\phi\alpha_n h = h^n(\phi\alpha'_n)$, where $h^n{:}A^n \to A'^n$ is the extension of h to A^n, taking $a = \langle a_i\rangle_{i\epsilon n}$ to $ah^n = \langle a_i h\rangle_{i\epsilon n}$. Usually we shall write $\phi\alpha_n$ as just $\phi{:}A^n \to A$ without fear of engendering confusion. $\langle A,\alpha\rangle$ is a *subalgebra* of $\langle A',\alpha'\rangle$ iff $A \subseteq A'$ and for all n and for $\phi\epsilon\Sigma_n$, $\phi\alpha_n = A^n \upharpoonright \phi\alpha'_n$; i.e., iff the inclusion function $A \to A'$ is a Σ-homomorphism.

Partial Σ-*algebras* and their homomorphisms are defined the same way except that for $\phi\epsilon\Sigma_n$, $\phi{:}A^n \to A$ need only be a *partial* function (commutativity of the above square means that the two composite partial functions are equal as partial functions; in particular, they are defined on the same subset of A^n, and give the same values in A' on each point in that set). The underlying set function of a homomorphism is required to be total.

In the above definitions the elements of the carrier of the algebras are all of the same "sort". However, we will want to talk about algebras which have many sorts of elements. For example, we will consider directed graphs as algebras with two sorts of elements, namely vertices and edges. The appropriate definition of many-sorted algebras is unfortunately somewhat technical looking. Let S be a set whose elements denote the sorts; an S-sorted algebra A should have one underlying set A_s for each element

$s\in S$. However, operations on A may involve arguments of an assortment of types, so that their arity cannot be given as a simple integer. Instead, the *arity* of an operation ϕ is described by a *sequence* $q:n \to S$ (for some integer n, the total number of arguments). Then, letting A^q denote the product $\prod_{i\in n} A_{q(i)}$, an operation is a function $\phi:A^q \to A_s$ where $s\in S$ is the *sort* of the operation ϕ. This hopefully motivates the definition of an *S-sorted symbol* or *operator set* as a family of subsets $\Sigma_{q,s}$ of Σ indexed by $S^* \times S$, where S^* denotes the set of all (finite) strings q in S. Then $\phi\in\Sigma_{q,s}$ has *arity* $q:n \to S$, *rank* n, and *sort* $s\in S$. A Σ-*algebra* is a pair $\langle A,\alpha\rangle$, where A is a family $\langle A_s\rangle_{s\in S}$ of sets and α is an indexed (by $S^*\times S$) family of functions, $\alpha_{q,s}:\Sigma_{q,s} \to A_s^{A^q}$ for $\langle q,s\rangle\in S^* \times S$. (Note we omit $\langle , \rangle$ on subscripts). As before, write just A for $\langle A,\alpha\rangle$ when convenient. A Σ-*homomorphism* $h:A \to A'$ of S-sorted Σ-algebras A,A' is a family $\langle h_s:A_s \to A'_s\rangle$ of functions such that for each $\langle q,s\rangle\in S^* \times S$ and $\phi\in\Sigma_{q,s}$,

$$\begin{tikzcd}
A^q \arrow[r, "\phi\alpha_{q,s}"] \arrow[d, "h^q"] & A_s \arrow[d, "h_s"] \\
A'^q \arrow[r, "\phi\alpha'_{q,s}"] & A'_s
\end{tikzcd}$$

commutes, i.e., $(\phi\alpha_{q,s})h_s = h^q(\phi\alpha'_{q,s})$, where $h^q:A^q \to A'^q$ takes $\langle a_i\rangle_{i\in n}$ to $\langle a_i h_{iq}\rangle_{i\in n}$ (where $q:n \to S$ and $a_i\in A_{iq}$). There is also the combination notion of a partial many-sorted algebra, which we shall not explain explicitly here.

If S is a singleton set, say $1 = \{0\}$, then $S^* \times S \cong S^* \cong \omega$, and 1-sorted Σ-algebras correspond to the ordinary Σ-algebras defined earlier.

The most common algebraic structures involve not only a set with some operations on it, but also some equations that the operations must satisfy. Rather than discuss the details of the general theory of such "algebraic theories" in the present context, we defer it to ADJ III and ADJ IV, where Lawvere's (1963) categorical approach to algebraic theories is described. Here we give as examples the equational classes with which we shall be most concerned in the sequal.

Example 1.1. Semigroups have only one binary operation, say $\cdot$ (i.e., $\Sigma_2 = \{\cdot\}$ and $\Sigma_n = \emptyset$ for $n \neq 2$), which is required to be associative, meaning that for all $x_0, x_1, x_2 \epsilon A$, $((x_0, x_1)\cdot, x_2)\cdot = (x_0, (x_1, x_2)\cdot)\cdot$. Actually, binary operations are usually written in "infix" notation rather than "postfix", so that the associative law takes the familiar form $(x_0 \cdot x_1)\cdot x_2 = x_0 \cdot (x_1 \cdot x_2)$. For example, the set ω with the binary operation of addition (i.e., $\cdot = +$) is a semigroup, usually denoted $\underline{N}$. □

Example 1.2. Monoids have the associative binary operation of semigroups, and in addition an identity constant e (i.e., $\Sigma_0 = \{e\}$, $\Sigma_2 = \{\cdot\}$, and $\Sigma_n = \emptyset$ for $n \neq 0,2$) satisfying the equations $e \cdot x_0 = x_0$ and $x_0 \cdot e = x_0$ for all $x_0 \epsilon A$ (in addition to the associative law for $\cdot$). The set ω also has the structure of a monoid $\underline{N}$, with $e = 0$. □

Example 1.3. There is one particular monoid which is of a great importance to us in the following, the free monoid X^* on (or generated by) a set X. Its set of elements is the set of all finite strings in X (i.e., $X^* = \{w{:}n \to X \mid n \epsilon \omega\}$), its identity is the empty string $\lambda{:}0 \to X$ (recall that $n = \{0,1,\cdots,n-1\}$, and $0 = \emptyset$), and its multiplication is concatenation or juxtaposition of strings, defined as follows: for $w{:}n \to X$ and $w'{:}n'$

$ww':n+n' \to X$ has values $k(ww') = kw$ for $0 \leq k < n$, and $k(ww') = (k-n)w'$ for $n \leq k < n+n'$. It is now an exercise (recommended to those who may not have seen concatenation rigorously defined before) to verify the associative and identity laws for X^*. (Incidentally, w is for "word", and elements of X^* are often called "words in X"). □

One property of X^* is basic:

Proposition 1.6. If $i:X \to X^*$ denotes the injection taking $x \in X$ to the string $\langle x \rangle$ of length 1 and value x, and if $f:X \to M$ is any function to a monoid M, then there is one and only one monoid homomorphism $\overline{f}:X^* \to M$ such that

$$\begin{tikzcd} & M \\ X \arrow[ur, "f"] \arrow[r, "i"] & X^* \arrow[u, "\overline{f}"'] \end{tikzcd}$$

commutes, i.e., such that $i\overline{f} = f$.

Proof. In fact, for $w:n \to X$, $w\overline{f}$ is the product $0wf \cdot 1wf \cdot \ldots \cdot (n-1)wf$ in M, and of course $\lambda\overline{f} = e$, the identity in M. For $\langle x \rangle : 1 \to X$, $\langle x \rangle \overline{f} = 0\langle x \rangle f = xf$ by definition, so that indeed $i\overline{f} = f$. We leave the reader to check that the conditions $i\overline{f} = f$, $\lambda\overline{f} = e$, and $(w\langle x \rangle)\overline{f} = w\overline{f} \cdot \langle x \rangle \overline{f}$ actually force this definition. □

The above "extension" property of X^* is called its *universal property*.

Example 1.4. *Groups* have a unary operation of *inverse* over and above the structure of monoids; thus $\Sigma_0 = \{e\}$, $\Sigma_1 = \{^{-1}\}$, $\Sigma_2 = \{\cdot\}$, and $\Sigma_n = \emptyset$ for $n \geq 3$. The additional equations are $x_0 \cdot x_0^{-1} = e$ and $x_0^{-1} \cdot x_0 = e$ for all $x_0 \in A$. The collection $\underline{Z}$ of all (negative and nonnegative) integers is a group with $x_0^{-1} = -x_0$, "minus". □

Example 1.5. *Abelian* (or *commutative*) groups are groups which satisfy in addition the *commutative law*: for all $x_0, x_1 \epsilon A$, $x_0 \cdot x_1 = x_1 \cdot x_0$. $\underline{Z}$ is an Abelian group. □

Example 1.6. *Rings* have simultaneous group and semigroup structures connected by a "distributive law". Thus $\Sigma_0 = \{0\}$, $\Sigma_1 = \{-\}$, $\Sigma_2 = \{+, \cdot\}$, and $\Sigma_n = \emptyset$ for $n \geq 3$. It is required that A with $0, -, +$ be a group, that A with $\cdot$ be a semigroup, and that for all x_0, x_1, $x_2 \epsilon A$, the *distributive laws* $x_0 \cdot (x_1 + x_2) = (x_0 \cdot x_1) + (x_0 \cdot x_2)$ and $(x_0 + x_1) \cdot x_2 = (x_0 \cdot x_1) + (x_0 \cdot x_2)$ hold. It can then be shown that in addition the commutative law $x_0 + x_1 = x_1 + x_0$ holds. *Unitary rings* have in addition an *identity* 1 for $\cdot$. $\underline{Z}$ is a unitary ring. Usually "ring" really means "unitary ring." □

Example 1.7. *Lattices* have two binary operations, $\vee$ and $\wedge$ ($\Sigma_2 = \{\vee, \wedge\}$, $\Sigma_n = \emptyset$ for $n \neq 2$) connected by a number of laws: $x_0 \wedge (x_1 \wedge x_2) = (x_0 \wedge x_1) \wedge x_2$, $x_0 \vee (x_1 \vee x_2) = (x_0 \vee x_i) \vee x_2$ (associative); $x_0 \wedge x_1 = x_1 \wedge x_0$, $x_0 \vee x_1 = x_1 \vee x_0$ (commutative); $x_0 \wedge x_0 = x_0$, $x_1 \vee x_1 = x_1$ (*idempotent*); and $x_0 \wedge (x_0 \vee x_1) = x_0$, $x_0 \vee (x_0 \wedge x_1) = x_0$ (*absorptive*). ω is a lattice with $x_0 \vee x_1$ the largest and $x_0 \wedge x_1$ the smallest of x_0, x_1.

Lattices have a natural associated partial order, $x_0 \leq x_1$ iff $x_0 \wedge x_1 = x_0$ iff $x_0 \vee x_1 = x_1$ (the last two are easily shown equivalent). With respect to this relation, $x_0 \wedge x_1$ is the largest element x_2 such that $x_2 \leq x_0$ and $x_2 \leq x_1$, called the *greatest lower bound* of x_0 and x_1; similarly, $x_0 \vee x_1$ is the *least upper bound* of x_0 and x_1. In fact, if a poset

$\langle A,\leq\rangle$ has greatest lower and least upper bounds for all pairs of elements, these two operations give A a lattice structure whose order relation is exactly $\leq$ again. The set ω of all nonnegative integers (viewed as sets) has a natural ordering corresponding to inclusion, i.e., $\leq = \subseteq$. The power $\underset{\sim}{P}(X)$ is a lattice with $\wedge = \cap$, $\vee = \cup$, and $\leq = \subseteq$. □

Example 1.8. A lattice is *distributive* iff it satisfies the additional equations $x_0\vee(x_1\wedge x_2) = (x_0\vee x_1)\wedge(x_0\vee x_2)$ and $x_0\wedge(x_1\vee x_2) = (x_0\wedge x_1)\vee(x_0\wedge x_2)$. ω is a distributive lattice, and so is $\underset{\sim}{P}(X)$. □

Example 1.9. A *zero* for a lattice is a constant $\perp$ such that $\perp\wedge x_0 = \perp$ and $\perp\vee x_0 = x_0$ for all x_0. A *unit* for a lattice is a constant $\top$ such that $\top\wedge x_0 = x_0$ and $\top\vee x_0 = \top$ for all x_0. ω has zero 0, but no unit. $\underset{\sim}{P}(X)$ has zero $\emptyset$ and unit X. A zero $\perp$ is a *least* or *minimum* element, in that $\perp \leq X$, for all x_0; similarly, a unit $\top$ is a *largest* or *maximum* element, in that $\top \geq x_0$ for all x_0. A lattice can have at most one zero, and at most one unit; for $\perp \leq \perp'$ and $\perp' \leq \perp$ imply $\perp = \perp'$. □

Example 1.10. A *complement* for a lattice with zero and unit is a unary operation $'$ such that $x_o\wedge x_0' = \perp$ and $x_0\vee x_0' = \top$ for all x_0. ω doesn't have a complement (i.e., no such operation can be defined on it); but $\underset{\sim}{P}(X)$ is complemented, with $X_0' = X-X_0$ for $X_0 \subseteq X$. In general, to say a lattice is *complemented* also implies it has zero and unit. □

Example 1.11. A *Boolean algebra* is a complemented distributive lattice. For any set X, $\underset{\sim}{P}(X)$ is a Boolean algebra. Many new laws follow automatically for Boolean algebras, including $(x_0')' = x_0$ (*involution*) and $(x_0\wedge x_1)' = x_0'\vee x_0'$ and $(x_0\vee x_1)' = x_0'\wedge x_1'$ (the *de Morgan laws*). □

Example 1.12. The notions of greatest lower and least upper bound easily extend to arbitrary (possibly infinite) subsets $S \subseteq P$ of a poset P. An element $x\epsilon P$ is a *lower bound* of S iff $x \leq s$ for all $s\epsilon S$, and is an *upper bound* of S iff $x \geq s$ for all $s\epsilon S$. Then a *greatest lower* bound (or glb) of S is some lower bound $x\in P$ of S such that for any other lower bound y of S, $x \geq y$. Antisymmetry implies any two greatest lower bounds of S are equal, and the unique glb of S, if it exists, is generally denoted $\bigwedge S$. One defines a least upper bound (or lub) for $S \subseteq P$ dually, and the symbol $\bigvee S$ is used. A poset in which every subset has a glb and a lub is called a complete lattice (it is easy to see it is a lattice). A slight extension is to consider glb and lub over families $\underset{\sim}{F} = \{s_i \mid i\in I\}$ of elements of P rather than over subsets $S \subseteq P$. The notations used then are $\bigvee_{i\in I} s_i$ and $\bigvee_{i\in I} s_i$, or possibly $\bigvee_I \underset{\sim}{F}$ and $\bigwedge_I \underset{\sim}{F}$. It is not obvious how to give an algebraic formulation for complete lattices similar to that of Example 1.7 for lattices. In fact, one uses the infinitary operations $\bigvee_I$ over all index sets I, with appropriate equations among them. The notions of infinitary algebras and theories needed to make this precise are treated in ADJ IV. □

Proposition 1.7. If a poset P has greatest lower bounds (or dually, least upper bounds) of all subsets, then it is a complete lattice.

Proof. Assume P has glbs for all subsets, and note that $\bigwedge\emptyset$, the glb of the empty subset, is a unit element $\top$ of P (in the sense that $\top \geq x$ for all $x\in P$). For $S \subseteq P$ let $US = \{x\in P \mid \text{for all } y\in S,\ y \leq x\}$; i.e., US is the set of all upper bounds of S. It is nonempty, for it surely contains $\top$. We claim that $\bigwedge US = \bigvee S$, the least upper bound of S. First, $\bigwedge US$ is an upper bound of S. For, if for all $x\in R \subseteq P$, we have $x \geq y$, then we have $\bigwedge R \geq y$. (Proof: if $y\wedge x = y$ for all $x\in R$, then $y\wedge\bigwedge R = \bigwedge\{y\wedge x \mid x\in R\} = \bigwedge\{y\} = y$, using commutative and idempotent laws for glb's). Thus, since for all $x\in US$, we have $x \geq y$ for any $y\in S$, we conclude that $\bigwedge US \geq y$ for any $y\in S$; i.e., that $\bigwedge US$ is an upper bound of S. Now suppose u is an upper bound of S. Then $u\in US$, so $u \geq \bigwedge US$; i.e., $\bigwedge US$ is the least upper bound, as claimed. □

2. CATEGORIES, FUNCTORS AND MACHINES

This section introduces our first two basic concepts, category and functor, plus some important ancillary concepts at the same level, with a number of applications and examples, primarily of a machine theoretic and algebraic character. The remainder of this report develops certain further computer science and categorical topics in greater detail. Deeper concepts and applications are the subjects of later reports. This order of presentation is motivated by our belief that the reader will be able to approach the more difficult material to follow, particularly adjointness, with greater enthusiasm after seeing these preliminary developments.

Our exposition is somewhat biased by the needs of the computer science material we treat. The reader who wants to see an exposition at the undergraduate mathematics level with applications specifically to algebra is referred to Mac Lane and Birkhoff (1967). Graduate level mathematics texts include Mitchell (1965) and Pareigis (1970), but the interested reader is especially directed to Mac Lane (1971a).

The reader who is going over this material for the first time should probably jump from the end of Subsection 2.2 directly to Section 3 (in Part 2). Even though Subsection 2.3 is not starred, it is also not much used until ADJ II.

2.1 CATEGORIES AND MACHINES

Definition 2.1. A *category* $\underline{\underline{C}}$ consists of:

(1) a class $|\underline{\underline{C}}|$, whose elements are called *objects*;

(2) a class, denoted $\underline{\underline{C}}$, whose elements are called *morphism*;

(3) a pair of functions $\partial_0, \partial_1 : \underline{\underline{C}} \to |\underline{\underline{C}}|$, called *source* and *target* respectively;

(4) a function $1:|\underline{\underline{C}}| \to \underline{\underline{C}}$ called the *identity function*;

(5) a partial binary operation $\circ:\underline{\underline{C}} \times \underline{\underline{C}} \to \underline{\underline{C}}$ called *composition* (we shall write $f \circ g$ for $(f,g)\circ$)

such that the following axioms hold for $f,g,h\in\underline{\underline{C}}$ and $A\in |\underline{\underline{C}}|$:

(1) $A1\partial_0 = A1\partial_1 = A$;

(2) $(f\partial_0 1) \circ f = f \circ (f\partial_1 1) = f$;

(3) $f \circ g$ is defined iff $f\partial_1 = g\partial_0$;

(4) if $f \circ g$ and $g \circ h$ are defined, then both $(f \circ g) \circ h$ and $f \circ (g \circ h)$ are defined, and they are equal;

(5) if $f \circ g$ is defined, then $(f \circ g)\partial_0 = f\partial_0$ and $(f \circ g)\partial_1 = g\partial_1$.

We give the definition of category this way for two specific reasons: it is *algebraic*, in that a category is a two-sorted partial algebra $<|\underline{\underline{C}}|,\ \underline{\underline{C}},\ \partial_0,\ \partial_1,\ 1,\ \circ>$, thus giving clear algebraic motivation for such important concepts as "subcategory" and "functor"; secondly, this definition is related to graphs in a particularly convenient way (see Section 3).

Before giving any examples of categories we shall give an alternative definition of category which is weaker, and therefore more convenient for verifying the examples. The equivalence of the two definitions is proved in Subsection 2.4, at the very end of Section 2.

Definition 2.1A. A *category* $\underline{\underline{C}}$ consists of the same data (1),(2),(3),(4),(5) as in Definition 2.1, satisfying the following axioms for $f,g,h\in\underline{\underline{C}}$ and $A\in|\underline{\underline{C}}|$:

(1) $A1\partial_0 = A1\partial_1 = A$;

(2) $(f\partial_0 1) \circ f = f \circ (f\partial_1 1) = f$;

(3A) if $f \circ g$ is defined, then $f\partial_1 = g\partial_0$;

(4A) if $f\partial_1 = g\partial_0$ and $g\partial_1 = h\partial_0$, then both $(f \circ g) \circ h$ and $f \circ (g \circ h)$ are defined, and they are equal.

Notice that (3A) and (4A) are weaker forms of (3) and (4), and that (5) is omitted. We now describe some notational conventions which are essential for clear exposition:

(I) Objects are generally denoted by capital latin letters; thus $A,B,C \in |\underline{\underline{C}}|$.

(II) For a morphism $f \in \underline{\underline{C}}$, we write $f:A \to B$ or $A \xrightarrow{f} B$ to mean that $f\partial_0 = A$ and $f\partial_1 = B$, and we say that f is a morphism "from A to B".

(III) $\underline{\underline{C}}(A,B)$ denotes the class of all morphism from A to B, and is called a "<u>hom</u> <u>set</u>" of $\underline{\underline{C}}$.

(IV) Composition is indicated by juxtaposition, so that if $f:A \to B$ and $g:B \to C$, then $fg:A \to C$; but sometimes to help parse complex expressions, we may revert to the original "infix" notation $f \circ g$.

(V) We write 1_A for $A1$. Since 1 is injective (if $A1 = A'1$, then $A = A1\partial_0 = A'1\partial_0 = A'$) we can in fact identify 1_A with A, and write $A:A \to A$ for the identity morphism 1_A.

<u>Example 2.1</u>. Perhaps the most familiar category is that of <u>sets</u> and functions, denoted $\underline{\underline{\underline{Set}}}$. Its objects are sets, and its morphisms are functions $f:A \to B$ (recall from Section 1.3 that these are triples $<A,f,B>$ with $f \subseteq A \times B$ satisfying the functional property). We define $<A,f,B>\partial_0 = A$ and $<A,f,B>\partial_1 = B$. Composition is the usual composition of functions, $<A,f,B> \circ <B',g,C> = <A,fg,C>$, defined iff $B = B'$ (as in Section 1.3), and the identity 1_A on a set A is simply the identity function $1_A:A \to A$ on A. We now verify the axioms given in Definition 1A (future examples will leave this as an exercise to the reader, unless there is some hidden difficulty). For axiom (1), since $A1 = <A,1_A,A>$, clearly $A1\partial_0 = A1\partial_1 = A$.

Axiom (2) asserts that for $f:A \to B$, 1_A is a left identity for composition and 1_B a right; this was shown in Section 1.3. We made axiom (3) in effect part of the definition of composition in **Set**. For axiom (4A), say $f\partial_0 = A$, $f\partial_1 = g\partial_0 = B$, $g\partial_1 = h\partial_0 = C$, and $h\partial_1 = D$. Then $f \circ g: A \to C$ and $g \circ h:B \to D$ are defined, and therefore so are $(f \circ g) \circ h$ and $f \circ (g \circ h)$, as morphisms $A \to D$. Their equality is associativity, as shown in Section 1.3. Thus **Set** is indeed a category. □

It is advisable to introduce a note of caution with this example. In many categories, including **Set**, $\mathbf{C}$ and $|\mathbf{C}|$ are not *sets*, but rather "classes" or perhaps "universes". If $|\mathbf{Set}|$ were a set, it would be an element of itself, a phenomenon generally avoided in set theory because of its potential for paradox. Various methods for handling such "large" objects are discussed in Mac Lane (1971b) and Feferman (1969). This report proceeds "naively", i.e., it avoids standing on any particular foundation, but it also avoids doing anything which any reasonable foundation would fail to support.

We shall, however, find use for the following "size" distinctions: a category $\mathbf{C}$ is *small* if both $\mathbf{C}$ and $|\mathbf{C}|$ are sets; and $\mathbf{C}$ is *locally small* iff for all $A,B \in |\mathbf{C}|$, $\mathbf{C}(A,B)$ is a set. Thus, **Set** is not small, but is locally small, as are most of the most useful categories.

Set and the many examples like it which we shall soon encounter (see Example 2.8) often leave the impression that morphisms must be some kind of set maps. In fact, this is far from the case, and there is even a particular tendency for examples of a contrary sort to arise in computer science, as the following may suggest.

Example 2.2. A (deterministic) *automaton* (or *transition system*) A is a two sorted algebra $<X,S,\delta>$, where X and S are sets, with elements called *inputs* and *states* respectively, and $\delta:S \times X \to S$ is a function called the *next state function* of A. A is *finite* iff both X and S are finite. Intuitively, if the device A is in state $s \epsilon S$ at some time and its input is $x \epsilon X$, then its next state is $s' = \langle s,x \rangle \delta \epsilon S$; we also say that the input x has caused a *transition* from state s to state s'. To describe the response of A to sequences of inputs, we extend [1] δ to a *transition function* $\delta^+ : S \times X^* \to S$ (recall from Example 1.3 of Section 1.5 that X^* is the monoid of all finite strings on X) defined for $s \epsilon S$, $n \epsilon \omega$, and $x_0, \cdots, x_{n-1} \epsilon X$ by $(s, x_0 x_1 \cdots x_{n-1})\delta^+ = ((\cdots((s,x_0)\delta, x_1)\delta, \cdots)\delta, x_{n-1})\delta$, and of course $(s,\lambda)\delta^+ = s$, since the null string λ causes no change in state. Intuitively, the input string w causes a transition from $s \epsilon S$ to $(s,w)\delta^+ \epsilon S$.

The state transitions are the morphisms of a category $\mathbf{Tr}_A$ whose objects are the states of A. More precisely, $|\mathbf{Tr}_A| = S$, and $\mathbf{Tr}_A(s_0,s_1) = \{\langle s_0,w,s_1 \rangle | w \epsilon X^*$ and $(s_0,w)\delta^+ = s_1\}$. Then $\langle s_0,w,s_1 \rangle \partial_i = s_i$, $1_s = \langle s,\lambda,s \rangle$, and $\langle s_0,w,s_1 \rangle \circ \langle s_1,w',s_2 \rangle = \langle s_0,ww',s_2 \rangle$. The category axioms are easily verified for $\mathbf{Tr}_A$. For convenience we write $s_0 \overset{w}{\to} s_1$ rather than the triple. This category is very closely related to what is usually called the *transition graph* of the automaton, as we shall see (Example 3.1). For the moment, note that $\mathbf{Tr}_A$ (intuitively) represents the dynamic *flow* of states in response to inputs. Many common automaton concepts are easily expressed in terms of this category. For example, A is *strongly connected* iff $\mathbf{Tr}_A(s_0,s_1) \neq \emptyset$ for all $s_0, s_1 \epsilon S$; and A is *reachable from* $s_0 \epsilon S$ iff $\mathbf{Tr}_A(s_0,s_1) \neq \emptyset$ for all $s_1 \epsilon S$. The subsets of

[1] There is an interesting alternative way to define δ^+, via a function $X^* \to [S,S]$, where $[S,S]$ denotes the monoid of all functions $S \to S$. First define $\delta_a : X \to [S,S]$ by $(s)((x)\delta_a) = (s,x)\delta$, for $s \epsilon S$. Now let $\overline{\delta}_a$ be the unique extension of δ_a to a monoid homomorphism, as given in Proposition 1.6. Then $(s,w)\delta^+ = (s)((w)\overline{\delta}_a)$ for all $w \epsilon X^*$.

X^* corresponding to the sets $\underline{\underline{Tr}}_A(s_0,s_1)$ (and their finite unions) are the sets recognizable or definable by A (see Example 2.11). However, we will use the word "recognizable" for a subset of X^* only when it is recognized by some *finite* automaton.

A *nondeterministic automaton* A is a system $\langle X,S,\delta\rangle$ where X and S are as above, but $\delta : S \times X \to S$ is a relation. The extension of δ to a transition relation $\delta^+ : S \times X^* \to S$ parallels the deterministic case: $(s,\lambda)\delta^+ s$ for all $s \in S$; and $(s,wx)\delta^+ s'$ iff there exists $s'' \in S$ such that $(s,w)\delta^+ s''$ and $(s'',x)\delta s'$. The transition category $\underline{\underline{Tr}}_A$ is constructed in exactly the same way. □

There is another broad class of categories quite useful in computer science whose morphisms are not set maps. We precede the general introduction of this class with some particular instances.

Example 2.3. $\underline{\underline{0}}$ is the empty category (no objects and no morphisms), and more generally, $\underline{\underline{n}}$ is the category with objects the nonnegative integers $k < n$ (i.e., $|\underline{\underline{n}}| = \{0,\cdots,n-1\}$, which is in fact the set n in the notation of Section 1.2) and with exactly one morphism $k_0 \to k_1$ in case $k_0 \leq k_1 < n$, and otherwise none. More set-theoretically, $\underline{\underline{n}} = \{\langle k_0,k_1\rangle | k_0 \leq k_1 < n\}, \langle k_0,k_1\rangle\partial_i = k_i$ for $i \in \{0,1\}, 1_k = \langle k,k\rangle$, and composition is (there is only one possible way to define it now) $\langle k_0,k_1\rangle \circ \langle k_1,k_2\rangle = \langle k_0,k_2\rangle$. It is easy to check validity of the category axioms given in Definition 2.1 or 2.1A. The category structure on $\underline{\underline{n}}$ reflects the natural ordering on the set n, since $k_0 \to k_1$ in $\underline{\underline{n}}$ iff $k_0 \leq k_1$. □

Example 2.4. The above construction can be generalized to any partially ordered set, but actually, it is just as easy and slightly more natural to consider *quasi ordered sets*, or *quosets* for short, i.e., pairs $<Q,R>$ where Q is a set and $R \subseteq Q \times Q$ is a reflexive transitive relation on Q (see also the end of Section 1.4). The associated category $\underline{\underline{Or}}_R$ has the elements of Q as objects, and as morphisms has pairs $<q,q'>$ such that qRq'; i.e., its set of morphisms *is* R. Then the identity on $q \in Q$ is $<q,q>$, an element of R by reflexivity; and the composite $<q,q'> \circ <q',q''>$ is $<q,q''>$, an element of R by transitivity. The axioms given in Definition 2.1 are now easily established. Incidentally, there is also a converse construction: any small category $\underline{\underline{C}}$ such that $\#\underline{\underline{C}}(A,B) \leq 1$ for all $A,B \in |\underline{\underline{C}}|$ is associated with a quasi ordered set $<|\underline{\underline{C}}|,R>$, where $<A,B> \in R$ iff $\#\underline{\underline{C}}(A,B) = 1$. We shall discuss later the sense in which such categories and quasi ordered sets determine each other (Example 2.21).

An extreme form of quasi order (in fact, the minimum quasi order on a set) has each element related *only* to itself, i.e., $R = \{<q,q> | q \in Q\}$, the diagonal relation on Q. The corresponding category has *only* identity morphisms. A category $\underline{\underline{C}}$ such that $\underline{\underline{C}}(A,B) = \emptyset$ unless $A = B$, and $\underline{\underline{C}}(A,A) = \{1_A\}$ for all $A \in |\underline{\underline{C}}|$ is called a *discrete category*. Clearly a small discrete category $\underline{\underline{C}}$ and its underlying set $|\underline{\underline{C}}|$ uniquely determine each other. □

A sort of opposite extreme to categories with only identity morphisms are categories with just one identity morphism.

Example 2.5. Any monoid $<M,\circ,e>$ can be viewed as a category $\underline{\underline{M}}$ withh exactly one object, say $|\underline{\underline{M}}| = \{e\}$, and one morphism $m{:}e \to e$ for each $m \in M$. Thus $m\partial_0 = m\partial_1 = e$, all $m \in \underline{\underline{M}}$. Composition is the monoid operation, and is always defined. The single identity for the single object e is $e{:}e \to e$ (i.e., $1_e = e$). The monoid laws now translate directly into the category axioms. Conversely, any small category $\underline{\underline{C}}$ with one object corresponds in a natural way to a monoid with carrier $\underline{\underline{C}}$. More generally, if A is an object in a locally small category $\underline{\underline{C}}$, then $<\underline{\underline{C}}(A,A),\circ,1_A>$ is a monoid where $\circ$ is composition in $\underline{\underline{C}}$ suitably (bi-) restricted. □

Example 2.6. In much the same way that $\underline{\underline{\underline{\mathrm{Set}}}}$ arises from sets and functions, we can construct the category $\underline{\underline{\underline{\mathrm{Mon}}}}$ whose objects are all monoids (see Example 1.2, Section 1.5) and whose morphisms are monoid homomorphisms, or more precisely, triples $<M_0,h,M_1>$ where M_0,M_1 are monoids and $h:M_0 \to M_1$ is a functional subset of $M_0 \times M_1$ with the homomorphism property. Then $<M_0,h,M_1>\partial_i = M_i$, $1_M = <M,1_M,M>$, and $<M_0,h,M_1> \circ <M_1,h',M_2> = <M_0,hh',M_2>$. The category axioms are easily verified, and as usual we cease to use the triple notation as soon as the precise definition has been given.

In very much the same way, we can define the categories $\underline{\underline{\underline{\mathrm{Sem}}}}$, $\underline{\underline{\underline{\mathrm{Grp}}}}$, $\underline{\underline{\underline{\mathrm{Rng}}}}$, and $\underline{\underline{\underline{\mathrm{Sem}}}}_*$ of (respectively) semigroups (Example 1.1), groups (Example 1.4), rings (Example 1.6), and pointed semigroups (which are semigroups with a distinguished element which satisfies no equations, but is preserved by the homomorphisms). □

This process of constructing a category from a suitable type of mathematical structure is (part of -- see Footnote 7 of Section 0) what the first "doctrine" of Section 0.2 refers to. We now formalize the process of obtaining categories for *algebraic* structures (in the sense of Section 1.5).

Example 2.7. Let Σ be a ranked alphabet (i.e., we are given a "ranking sequence" $\Sigma_0,\Sigma_1,\Sigma_2,\cdots$ of subsets of Σ), and recall that a Σ-algebra is pair $<A,\alpha>$, where α is a sequence $\alpha_0,\alpha_1,\cdots$ of functions $\alpha_n:\Sigma_n \to A^{A^n}$

assigning n-ary operations to n-ary symbols; see Section 1.5 for the definition of a Σ-algebra homomorphism $h:A \to A'$. Let $\underline{\underline{\mathrm{Alg}}}_\Sigma$ be the category with Σ-algebras as objects and Σ-homomorphisms (in the form of triples $<A,h,A'>$) as morphisms, with functional composition, and with the usual identities (namely $<A,1_A,A>$). The category axioms are easily verified. □

The same process can be carried out for many-sorted algebras (again see Section 1.5). We now give an important example of such a category, but in a more naive and convenient notation than that of Section 1.5.

Example 2.8. A *machine* is a six-tuple $M = <X,S,Y,\delta,\sigma,\beta>$, where X,S,Y are sets, called respectively *input*, *state*, and *output sets* (or sometimes *alphabets*), $\delta:S \times X \to S$ is the *next-state function*, $\sigma\in S$ is the *initial state*, and $\beta:S \to Y$ is the *output function*. Just as with automata (Example 2.2), we say a machine M is *finite* iff its X,S, and Y are finite.

Such an M is a three-sorted algebra, with sort set $\{X,S,Y\}$ and operator domain $\Sigma_{S,X;S} = \{\delta\},\Sigma_{\lambda;S} = \{\sigma\}$, $\Sigma_{S;Y} = \{\beta\}$, and $\Sigma_{q;t} = \emptyset$ for all other $<q,t>\in\{X,S,Y\}^*\times \{X,S,Y\}$. Substituting this special operator domain into the general definition of many-sorted algebra homomorphism, we have the definition of a *machine homomorphism* $M \to M'$ as a triple $<a,b,c>$ of set maps, $a:X \to X'$, $b:S \to S'$, $c:Y \to Y'$, such that the following diagrams commute,

$$\begin{tikzcd}
S\times X \arrow[r, "\delta"] \arrow[d, "b\times a"'] & S \arrow[d, "b"] \\
S'\times X' \arrow[r, "\delta'"] & S'
\end{tikzcd}
\qquad
\begin{tikzcd}
 & S \arrow[d, "b"] \\
1 \arrow[ur, "\sigma"] \arrow[r, "\sigma'"'] & S'
\end{tikzcd}
\qquad
\begin{tikzcd}
S \arrow[r, "\beta"] \arrow[d, "b"'] & Y \arrow[d, "c"] \\
S' \arrow[r, "\beta'"] & Y'
\end{tikzcd}$$

where σ and σ' are represented as functions on the one-point set

$1 = \{0\}$ (i.e., σ as a function applied to the element 0 is σ the point in S). It should be clear how to define composition ($\langle a,b,c\rangle \circ \langle a',b',c'\rangle = \langle aa',bb',cc'\rangle$); Proposition 1.1 is used to verify the composite satisfies the necessary diagrams) and identities ($1_M = \langle 1_X, 1_S, 1_Y\rangle$), and it should also be clear that a category of Σ-algebras results. We denote it *Mach*, the category of all machines.

As a simpler example of the same type, we construct the *category of automata* (see Example 2.2) as Alg_Σ, the category of $\{X,S\}$-sorted algebras with $\Sigma_{S,X;S} = \{\delta\}$, and all other $\Sigma_{q;t} = \emptyset$. Then objects are triples $\langle X,S;\delta\rangle$, exactly as in Example 2.2, and morphisms are pairs $\langle a,b\rangle$ with $a:X \to X'$ and $b:S \to S'$ such that

$$\begin{tikzcd}
S\times X \arrow[r, "\delta"] \arrow[d, "b\times a"'] & S \arrow[d, "b"] \\
S'\times X' \arrow[r, "\delta'"] & S'
\end{tikzcd}$$

commutes. Let us denote this category *Aut*. □

Example 2.9. Behaviors. Let $M = \langle X,S,Y,\delta,\sigma,\beta\rangle$ be a machine. Recall that $\delta^+:S \times X^* \to S$ is the transition function of the automaton $\langle X,S,\delta\rangle$ (Example 2.2). The function $(\sigma,-)\delta^+:X^* \to S$ obtained by fixing the first argument of δ^+ at the initial state σ is called the *response function* of M and in the context of machines is also denoted δ^+. Then $\delta^+\beta:X^* \to Y$ is the (external) *behavior* of M. For $w \in X^*, w\delta^+\beta$ is the last output obtained in response to the input string w. Note also that f is the behavior of $\langle X,X^*,Y,\delta,\lambda_X,f\rangle$, where δ is defined by $(w,x)\delta = wx$ (concatenation). Generalizing this, we will call any $f:X^* \to Y$ a *behavior*.

We obtain a category *Beh* of behaviors by defining a morphism from $f:X^* \to Y$ to $f:X'^* \to Y'$ to be a pair of functions, $a:X \to X'$ and $b:Y \to Y'$

such that

$$\begin{tikzcd} X^* \arrow[r, "a^*"] \arrow[d, "f"'] & X'^* \arrow[d, "f'"] \\ Y \arrow[r, "b"] & Y' \end{tikzcd}$$

commutes, i.e., $a^*f' = fb$, where a^* is the extension of $a:X \to X'$ to strings, $(x_0\cdots x_{n-1})a^* = x_0a\cdots x_{n-1}a$, i.e., the i^{th} element of the string $(x_0\cdots x_{n-1})a^*$ is x_ia.

The composite of $<a,b>:f \to f'$ and $<a',b'>:f' \to f''$ is defined to be $<aa',bb'>:f \to f''$, and is shown to satisfy the appropriate square by "composing" the squares for the individual mappings, using Proposition 1.1,

$$\begin{tikzcd} X^* \arrow[r, "a^*"] \arrow[d, "f"'] & X'^* \arrow[r, "a'^*"] \arrow[d, "f'"] & X''^* \arrow[d, "f''"] \\ Y \arrow[r, "b"] & Y' \arrow[r, "b'"] & Y'' \end{tikzcd} \qquad \begin{tikzcd} X^* \arrow[r, "a^*a'^*"] \arrow[d, "f"'] & X''^* \arrow[d, "f''"] \\ Y \arrow[r, "bb'"] & Y'' \end{tikzcd}$$

and the fact (which is easily checked) that $a^*a'^* = (aa')^*$. This composition is certainly associative, $<a,b>(<a',b'><a'',b''>) = <aa'a'',bb'b''> = (<a,b><a',b'>)<a'',b''>$, and the identity on $f:X^* \to Y$ is $1_f = <1_X,1_Y>$ which clearly satisfies the identity laws. Thus we have a category $\underline{\underline{\underline{Beh}}}$ whose objects are all functions $f:X^* \to Y$ (for all sets X,Y) and whose morphisms are the pairs $<a,b>$ as above. □

It is often the case that the objects we most wish to study appear as a subclass of some $|\underline{\underline{\underline{Alg}}}_\Sigma|$. For example, the monoids are a subclass of the Σ-algebras with $\Sigma_0 = \{e\}$, $\Sigma_2 = \{\circ\}$, and $\Sigma_n = \emptyset$ for $n \neq 0,2$. In this case, as many others, the monoid morphisms are also Σ-homomorphisms.

For another example, the finite state machines are a subclass among all machines, with the same morphisms. We shall also see that <u>languages</u>

can be viewed as a special class of behaviors. The relationship which holds in these examples is formalized by the following.

Definition 2.2. A category $\underline{\underline{B}}$ is a subcategory of a category $\underline{\underline{C}}$, written $\underline{\underline{B}} \subseteq \underline{\underline{C}}$, iff: $|\underline{\underline{B}}| \subseteq |\underline{\underline{C}}|$; $\underline{\underline{B}}(A,B) \subseteq \underline{\underline{C}}(A,B)$ for all $A,B \in |\underline{\underline{B}}|$; 1_A in $\underline{\underline{B}}$ equals 1_A in $\underline{\underline{C}}$ for $A \in |\underline{\underline{B}}|$; and finally, composition in $\underline{\underline{B}}$ agrees with (i.e., is the restriction of) that in $\underline{\underline{C}}$. $\underline{\underline{B}}$ is a full subcategory of $\underline{\underline{C}}$ if, in addition, $\underline{\underline{B}}(A,B) = \underline{\underline{C}}(A,B)$ for all $A,B \in |\underline{\underline{B}}|$. $\underline{\underline{B}}$ is a strict subcategory of $\underline{\underline{C}}$ iff $|\underline{\underline{B}}| = |\underline{\underline{C}}|$. (Note that if $\underline{\underline{B}}$ is a full and strict subcategory of $\underline{\underline{C}}$, then $\underline{\underline{B}} = \underline{\underline{C}}$).

This definition is consistent with our two-sorted algebraic definition of category, in that a subcategory is a subalgebra in the appropriate sense.

Now some examples: $\underline{\underline{\mathrm{Mon}}}$ is a full subcategory of $\underline{\underline{\mathrm{Alg}}}_{\Sigma}$ with Σ as above, since the monoid morphisms $M \to M'$ are exactly the Σ-homomorphisms $M \to M'$. The category $\underline{\underline{\mathrm{Ab}}}$ of Abelian groups (see Example 1.5) is a full subcategory of that of all groups, $\underline{\underline{\mathrm{Grp}}}$. The category of all finite machines (those with X,S,Y finite sets) is a full subcategory of $\underline{\underline{\mathrm{Mach}}}$. $\underline{\underline{\mathrm{Mon}}}$ is a full subcategory of $\underline{\underline{\mathrm{Sem}}}_*$, and $\underline{\underline{k}}$ of $\underline{\underline{n}}$ for $k \leq n$.

In terms of the first doctrine (Section 0.2), subcategories often arise through the imposition of some additional restrictions on a type of mathematical structure. Thus $\underline{\underline{\mathrm{Ab}}} \subseteq \underline{\underline{\mathrm{Grp}}}$ arises by imposing the commutativity condition. Full subcategories $\underline{\underline{B}} \subseteq \underline{\underline{C}}$ arise when exactly the same structure is preserved in $\underline{\underline{B}}$ as was in $\underline{\underline{C}}$. A nonfull case can arise by requiring morphisms in $\underline{\underline{B}}$ to preserve some additional structure not required for $\underline{\underline{C}}$.

For an extreme example, given a category $\underline{\underline{C}}$, let $|\underline{\underline{C}}|$ denote the discrete category with objects $|\underline{\underline{C}}|$. Then $|\underline{\underline{C}}|$ is a subcategory of $\underline{\underline{C}}$

(another interpretation of the convention that $|\underline{\underline{C}}| \subseteq \underline{\underline{C}}$) but is very far from full, though it is strict. An amusing full subcategory which can be constructed from any locally small category $\underline{\underline{C}}$ and object $A \in |\underline{\underline{C}}|$ is the monoid $<\underline{\underline{C}}(A,A), \circ, 1_A>$, viewed as a category again.

It is very convenient and common to _define_ a subcategory $\underline{\underline{B}}$ of a category $\underline{\underline{C}}$ by giving a subclass X of $\underline{\underline{C}}$, and letting $\underline{\underline{B}}$ be the least [full] subcategory of $\underline{\underline{C}}$ containing X. The existence of such a subcategory can be shown by considering the intersection $\underline{\underline{S}}$ of the family of all [full] subcategories of $\underline{\underline{C}}$ containing X. First, this family is nonempty (since $\underline{\underline{C}}$ is a full subcategory of $\underline{\underline{C}}$ containing X). Second, $\underline{\underline{S}}$ is itself a [full] subcategory of $\underline{\underline{C}}$ containing X (one easily checks that the intersection of any family of [full] subcategories of $\underline{\underline{C}}$ is again a [full] subcategory). Then finally, $\underline{\underline{S}}$ is the least [full] subcategory of $\underline{\underline{C}}$ containing X (since any such subcategory is in the family defining $\underline{\underline{S}}$, it therefore contains $\underline{\underline{S}}$). One calls this the [full] subcategory _generated by_ X. In the special case where X is the class of identities for some class K of objects $(X = K1)$ we speak of the full subcategory $\underline{\underline{B}}$ of $\underline{\underline{C}}$ generated by K. Here $|\underline{\underline{B}}| = K$. In effect, we define $\underline{\underline{B}} \subseteq \underline{\underline{C}}$ by giving $|\underline{\underline{B}}|$ and requiring fullness.

Example 2.10. Let $\underline{\underline{Fin}}$ be the full subcategory of $\underline{\underline{Set}}$ whose objects are _all_ finite sets (and whose morphisms are then _all_ functions between finite sets). Similarly, $\underline{\underline{Fmach}}$ is the full subcategory of $\underline{\underline{Mach}}$ whose objects are finite machines. Let $\underline{\underline{N}}$ be the full subcategory of $\underline{\underline{Set}}$ whose objects are the natural number $0,1,2,\cdots$ (i.e., $|\underline{\underline{N}}| = \omega$). This category plays a central role as the "base category" for the "algebraic theories" discussed in ADJ III. □

Some interesting and typical nonfull subcategories arise as follows:

Example 2.11. (see Example 2.8). Let us fix an input alphabet X and consider the category $\mathbf{Mach}^X$ of all *X-machines*, with input alphabet X, and with morphisms $\langle a,b,c\rangle$ preserving X, i.e., such that $a = 1_X$. Then $\mathbf{Mach}^X \subseteq \mathbf{Mach}$ is neither full nor strict. Let $\mathbf{Mach}_Y$ be the subcategory of $\mathbf{Mach}$ with fixed output set Y and morphisms $\langle a,b,1_Y\rangle$. Similarly, let $\mathbf{Mach}_Y^X$ be the subcategory of *X,Y-machines*, with both input and output alphabets fixed, and with morphisms $\langle 1_X,b,1_Y\rangle$. Then $\mathbf{Mach}_Y^X \subseteq \mathbf{Mach}^X$. The category $\mathbf{Mach}_2^X$ (where $2 = \{0,1\}$) is the category of *acceptors* over X. For an output function $\beta:S \to \{0,1\}$ can be viewed as designating a set $F = 1\beta^{-1}$ of states as *final states*. An input $w \in X^*$ is *accepted* iff $(\sigma,w)\delta^+ \in F$, i.e., iff $(\sigma,w)\delta^+\beta = 1$. The subset $1(\delta^+\beta)^{-1}$ of X^* is the set of strings *accepted* or *recognized* by M, and we call a set of strings *recognizable* iff there exists a *finite* machine which recognizes that set.

In the same way, $\mathbf{Aut}^X \subseteq \mathbf{Aut}$ is the subcategory of all X-automata (see Examples 2.2 and 2.8), with the X-component of morphisms fixed at 1_X. □

There are some particularly important categories of which $\mathbf{Set}$ is a subcategory. We shall start being less detailed with these.

Example 2.12. The "category of relations" $\mathbf{Rel}$ has $|\mathbf{Rel}| = |\mathbf{Set}|$ and morphisms relations $R:A \to B$, with composition the usual (Peirce) product of relations, which is associative (see Section 1.3) and has the identity functions $1_A:A \to A$ as identities. The category of partial functions $\mathbf{Pfn}$ has the same objects again, and has morphisms $f:A \to B$ the *partial* functions, with the usual composition and identities. Note that $\mathbf{Set} \subseteq \mathbf{Pfn} \subseteq \mathbf{Rel}$, and that these subcategories are strict but not full.

An interesting category which is closely related to **Set** and **Pfn** is the category $\mathbf{Set}_*$ of pointed sets. Its objects are pairs $\langle A,a\rangle$, where A is a set and $a\epsilon A$ is a "point" of A. A morphism $\langle A,a\rangle \to \langle B,b\rangle$ in $\mathbf{Set}_*$ is a set function $f:A \to B$ which preserves the points, i.e., such that $af = b$. It turns out that **Pfn** and $\mathbf{Set}_*$ are "equivalent" categories (see Part 2). □

Example 2.13. *Languages*. Nonfull subcategories of **Beh** arise by fixing X or Y, as was the case with machines (Example 2.11). An especially important case arises when Y is fixed at $2 = \{0,1\}$, and all second components of morphisms are fixed at the identity on 2. Denote this category **Lng**; the morphisms $f \to f'$ are transliterations (or "length preserving homomorphisms") $a:X \to X'$ of input alphabets, such that

$$\begin{tikzcd}
X^* \arrow[rr, "a^*"] \arrow[dr, "f"'] & & X'^* \arrow[dl, "f'"] \\
& \{0,1\} &
\end{tikzcd}$$

commutes; i.e., such that $(x_0\cdots x_{n-1})f = 1$ iff $(x_0a\cdots x_{n-1}a)f' = 1$. The reason for calling this category **Lng** is that functions $f:X^* \to \{0,1\}$ can be thought of as *languages*, i.e., as subsets of X^*. Namely, $f:X^* \to \{0,1\}$ defines the subset $1f^{-1}$ of all strings $w\epsilon X^*$ such that $wf = 1$; conversely, any subset $L \subseteq X^*$ defines a function $f:X^* \to \{0,1\}$ by $wf = 1$ iff $w\epsilon L$. Such an f is often called a *characteristic function* for the subset. □

It might be noted that there are two terminologies for "languages": (1) elements of X are "words", and strings of words are "sentences";

and (2) elements of X are "letters" of the "alphabet" X, so that strings from X are "words". These two terminologies are used interchangeably in the following. This should cause no confusion because such descriptions are only used for motivation and do not effect formal definitions or proofs. There are also respects in which the word "language" is misleading in the present context: an abstract subset of X^* has neither syntax nor semantics, and only remotely resembles a natural language such as English. However, there are very many examples in computer science of languages in this sense, and the terminology is quite standard.

2.2 FUNCTORS AND MACHINES

From an algebraic point of view, the natural sequel to the definitions of category and subcategory is the definition of a (homo-) morphism of categories.

Definition 2.3. A functor from a category $\underline{\underline{A}}$ to a category $\underline{\underline{B}}$ is a quadruple $<\underline{\underline{A}},|F|,F,\underline{\underline{B}}>$, where $|F|:|\underline{\underline{A}}| \to |\underline{\underline{B}}|$ and $F:\underline{\underline{A}} \to \underline{\underline{B}}$ are functions, such that:

(1) if $f:A \to A'$ in $\underline{\underline{A}}$, then $fF:A|F| \to A'|F|$ in $\underline{\underline{B}}$;

(2) $(fg)F = (fF)(gF)$ whenever the composition fg is defined in $\underline{\underline{A}}$; and

(3) $1_A F = 1_{A|F|}$ for $A \epsilon |\underline{\underline{A}}|$.

We shall call $|F|$ and F the object and arrow parts (or components) of the functor. Because of the condition $1_A F = 1_{A|F|}$, $|F|$ is completely determined by F, and it is usual to suppress $|F|$ and write $F:\underline{\underline{A}} \to \underline{\underline{B}}$ to denote the functor. We also usually write AF rather than $A|F|$, and we call $\underline{\underline{A}}$ the source and $\underline{\underline{B}}$ the target category of F. A functor with equal source and target category is called an endofunctor. A functor $F:\underline{\underline{A}} \to \underline{\underline{B}}$ is injective, surjective, or bijective iff its arrow part is the same; this implies its object part is too. Given $A,A' \epsilon |\underline{\underline{A}}|$, note that

IBM Research

A JUNCTION BETWEEN COMPUTER SCIENCE AND CATEGORY THEORY, I: BASIC CONCEPTS AND EXAMPLES (PART I)

J. A. Goguen/J. W. Thatcher/E. G. Wagner/
J. B. Wright

September 11, 1973

RC 4526

pt. 2

Yorktown Heights, New York
San Jose, California
Zurich, Switzerland

F "birestricts" (see Section 1.3) to $F_{AA'}:\underline{\underline{A}}(A,A') \to \underline{\underline{B}}(AF,A'F)$. We say that F is *full* iff each $F_{AA'}$ is surjective; and that F is *faithful* iff each $F_{AA'}$ is injective. Moreover, F is *strict* iff $|F|$ is an identity function, and F is an *isomorphism* iff F is bijective.

Note that this definition *is exactly* that arising from the notion of homomorphism for categories as partial two-sorted algebras, although it is convenient and reasonable to suppress the object sort. It will also enable us to give a more algebraically satisfying definition of isomorphism of categories later in our development.

Example 2.14. A subcategory relationship $\underline{\underline{A}} \subseteq \underline{\underline{B}}$ induces an *inclusion functor* $F:\underline{\underline{A}} \to \underline{\underline{B}}$ in a natural way, namely $fF = f$ and $A|F| = A$ for $f\in\underline{\underline{A}}$ and $A\in|\underline{\underline{A}}|$. Rather than have to create new special letters for these very common functors, it is convenient to write $\subseteq:\underline{\underline{A}} \to \underline{\underline{B}}$, or even simply $\underline{\underline{A}} \subseteq \underline{\underline{B}}$. □

Proposition 2.1. Any inclusion functor $\underline{\underline{A}} \subseteq \underline{\underline{B}}$ is faithful. If the subcategory is full or strict, so is the functor. Any injective functor $F:\underline{\underline{A}} \to \underline{\underline{B}}$ determines an *image subcategory* $\underline{\underline{C}} \subseteq \underline{\underline{B}}$, defined by $|\underline{\underline{C}}| = |\underline{\underline{A}}|F$ and $\underline{\underline{C}}(AF,A'F) =$ the image $\underline{\underline{A}}(A,A')F$; we usually write $\underline{\underline{A}}F$ for this category. For an injective functor F, $\underline{\underline{A}}F$ is isomorphic to $\underline{\underline{A}}$. More generally, $\underline{\underline{A}}F$ is a category if $|F|$ is injective, though $\underline{\underline{A}}F$ is not in general isomorphic to $\underline{\underline{A}}$. □

But if F is not injective, even if it is faithful, the construction of $\underline{\underline{C}}$ given above may fail to yield a category. For example, define $\underline{\underline{A}}$ by letting $|\underline{\underline{A}}| = \{A,B,C,D\}$ and $\underline{\underline{A}} = \{f:A \to B, g:C \to D, 1_A, 1_B, 1_C, 1_D\}$, with A,B,C,D distinct, and no nontrivial compositions. Define $\underline{\underline{B}}$ by letting

$|\underline{\underline{B}}| = \{A,B,D\}$, $\underline{\underline{B}} = \{f':A \to B, g':B \to D, h:A \to D, 1_A, 1_B, 1_D\}$, with the only nontrivial composition being $f'g' = h$. Define $F:\underline{\underline{A}} \to \underline{\underline{B}}$ by $A \mapsto A$, $B \mapsto B$, $C \mapsto B$, $D \mapsto D$, $f \mapsto f'$, and $g \mapsto g'$. Then $|\underline{\underline{C}}| = \{A,B,D\}$, $\underline{\underline{C}}(AF,BF) = \{f'\}$, $\underline{\underline{C}}(CF,DF) = \{g'\}$, while $\underline{\underline{C}}(AF,DF) = \emptyset$. But if $\underline{\underline{C}}$ were a category, $f'g'$ would have to be defined and in $\underline{\underline{C}}(AF,DF)$, since $f'\partial_1 = g'\partial_0 = B$.

$$\begin{tikzcd}
 & B & C & & & & B & \\
A \arrow[ur, "f"] & & & D \arrow[from=1-3, "g"] \arrow[rr, mapsto] & & A \arrow[ur, "f'"] \arrow[rr, "h"'] & & D \arrow[from=1-7, "g'"]
\end{tikzcd}$$

When $F:\underline{\underline{A}} \to \underline{\underline{B}}$ is injective, we speak of F as an <u>embedding</u> of $\underline{\underline{A}}$ into $\underline{\underline{B}}$, and of $\underline{\underline{A}}F$ as the "embedded copy" of $\underline{\underline{A}}$ in $\underline{\underline{B}}$.

<u>Example 2.15</u>. <u>Power Set Functor</u>. Many natural constructions on sets correspond to functors. The power set (i.e., set of subsets) construction is a good example. Define $\underset{\sim}{P}:\underline{\underline{\underline{Set}}} \to \underline{\underline{\underline{Set}}}$ by letting $A\underset{\sim}{P}$ be the set of subsets of A, and for $f:A \to B$, define $f\underset{\sim}{P}:A\underset{\sim}{P} \to B\underset{\sim}{P}$ for $U \in A\underset{\sim}{P}$ by $U(f\underset{\sim}{P}) = Uf$, the <u>image</u> of U under. f (see Section 1.3). Since it is easily checked that $1_A\underset{\sim}{P} = 1_{A\underset{\sim}{P}}$ and $fg\underset{\sim}{P} = f\underset{\sim}{P}g\underset{\sim}{P}$, we have defined a functor. □

<u>Example 2.16</u>. (see Examples 2.8, 2.9, and 2.13). <u>External Behavior Construction.</u> The natural constructions of computer science also correspond to functors. For example, if $\langle X,S,\{0,1\},\delta,\sigma,\beta\rangle$ is an acceptor (Example 2.11), the set of strings in X^* which it "accepts" is $1(\delta^+\beta)^{-1}$, a language (Example 2.13). This is the object part of a functor. We shall now define the arrow part and prove that it really is a functor in the somewhat more general context of the external behavior functor for machines.

Let $M = \langle X,S,Y,\delta,\sigma,\beta\rangle$ be a machine, and define $ME:X^* \to Y$ to be $\delta^+\beta$. Given $\langle a,b,c\rangle:M \to M'$ in $\underline{\underline{\underline{Mach}}}$, define $\langle a,b,c\rangle E:ME \to M'E$ to be $\langle a,c\rangle$.

We must check it actually is a behavior morphism, i.e., that

(*)
$$\begin{tikzcd}
X^* \arrow[r, "a^*"] \arrow[d, "\delta^+"'] & X'^* \arrow[d, "\delta'^+"] \\
S \arrow[d, "\beta"'] & S' \arrow[d, "\beta'"] \\
Y \arrow[r, "c"] & Y'
\end{tikzcd}$$

commutes. For this purpose it suffices by Proposition 1.1 to show that

(**)
$$\begin{tikzcd}
X^* \arrow[r, "a^*"] \arrow[d, "\delta^+"'] & X'^* \arrow[d, "\delta'^+"] \\
S \arrow[r, "b"] & S'
\end{tikzcd}$$

commutes; for the diagram (*) is the composite of (**) and one of the squares satisfied by $\langle a,b,c\rangle$ being a machine morphism (namely, the third square of Example 2.8). We prove (**) by induction on the length of strings: For length zero, there is only one string, namely λ, and $\lambda a^*\delta'^+ = \lambda\delta'^+ = \sigma'$, while $\lambda\delta^+b = \sigma b = \sigma'$, the last step by the second square in Example 2.8. Now assume (**) for $w \epsilon X^*$ of length n, and consider $wx \epsilon X^*$ for $x \epsilon X$. Then $(wx)\delta^+b = \langle w\delta^+, x\rangle\delta b = \langle w\delta^+b, xa\rangle\delta'$ [by the first square in Example 2.9] $= \langle wa^*\delta', xa\rangle\delta'$ [using the inductive hypothesis] $= (wa^*\cdot xa)\delta'^+ = (wx)a^*\delta'^+$, as desired. This shows $\langle a,b,c\rangle E = \langle a,c\rangle \epsilon\ \underline{\underline{\mathrm{Beh}}}$. To show $E:\underline{\underline{\mathrm{Mach}}} \to \underline{\underline{\mathrm{Beh}}}$ is a functor, we need that $(\langle a,b,c\rangle\langle a',b',c'\rangle)E = \langle a,b,c\rangle E\langle a',b',c'\rangle E$ and that $1_M E = 1_{ME}$. These are clear, since for the first, both equal $\langle aa', cc'\rangle$, while for the second both equal $\langle 1_X, 1_Y\rangle$.

By restricting and corestricting this functor, we obtain $E:\underline{\underline{\mathrm{Mach}}}_2 \to \underline{\underline{\mathrm{Lng}}}$, the functor which describes the construction of the language defined by an acceptor, as indicated in the first paragraph of this example. (It must be verified that E applied to acceptors and acceptor morphisms always yields languages and language morphisms, but this is easy.) □

The functors in the above examples arise from reasonable constructions, with source a category of data for the construction, and target a category of results of the construction, in the manner suggested by the second "doctrine" of Section 0.2. We now give an example to show that not all functors arise from constructions in the most obvious sense.

Example 2.17. (see Example 2.5). Let $h:M_0 \to M_1$ be a monoid homomorphism, and let $\underline{\underline{M}}_0$ and $\underline{\underline{M}}_1$ denote the categories obtained from M_0 and M_1 in the manner of Example 2.5. Then $H:\underline{\underline{M}}_0 \to \underline{\underline{M}}_1$ defined by $mH = mh$ is a functor, for $(mm')h = mhm'h$ and $e_0h = e_1$ show that $(mm')H = mHm'H$ and $1_eH = 1_{eH} = 1_{e'}$, the functorial conditions. Note that $|\underline{\underline{M}}_0| = \{e_0\}$ and $e_0|H| = e_0H = e_0h = e_1$. □

Example 2.18. (see Example 2.2) *Automata and M-transition Systems.* Let $\underline{\underline{M}}$ denote the category obtained from the monoid X^* by the process of Example 2.5. Then a functor $A:\underline{\underline{M}} \to \underline{\underline{\underline{Set}}}$ describes an automaton with X as input alphabet. Because $|\underline{\underline{M}}| = \{\lambda\}$, $\lambda A \epsilon |\underline{\underline{\underline{Set}}}|$ is the only set arising from the objects of $\underline{\underline{M}}$ through A. In fact, this set is the automaton's "state set" and it is hereafter denoted S. We have now only to uncover the transition function $\delta:S \times X \to S$. But $x \epsilon X$ is also a string x in X^*, and therefore a morphism $x: \lambda \to \lambda$ in $\underline{\underline{M}}$, so that we have $xA:S \to S$ in $\underline{\underline{\underline{Set}}}$. Thus, we can define δ by $(s,x)\delta = s(xA)$. Conversely, given an automaton $\langle X,S,\delta\rangle$, we define $A:X^* \to \underline{\underline{\underline{Set}}}$ by $\lambda A = S$ and $wA = (-,w)\delta^+:S \to S$, for $w \epsilon X^*$, where $(-,w)\delta^+$ denotes the function obtained from $\delta^+:S \times X^* \to S$ by fixing the second argument to be w. We shall see later that the class of functors between two categories also forms a category, and when this "functor category" is specialized to $\underline{\underline{M}}$ obtained from X^*, and $\underline{\underline{\mathrm{Set}}}$, it will yield (a category isomorphic to) the category of automata as defined in Example 2.9. See Examples 3.5 and 3.6 (in Part 2).

There is an obvious generalization of the above construction: Let M be any monoid and let $\underline{\underline{M}}$ be its category. Then an *M-automaton* is a functor $A:\underline{\underline{M}} \rightarrow \underline{\underline{\mathrm{Set}}}$. This notion has been studied (Give'on (1967)), and we shall see that many of the known results can be obtained directly and simply from general facts about the corresponding functor categories. □

Our next example, "the category of categories", may at first seem overly abstract, and perhaps even contradictory, but it embodies several basic facts about functors and is important in a wide range of applications to follow. We first give the facts in question.

Proposition 2.2. Given functors $F:\underline{\underline{A}} \rightarrow \underline{\underline{B}}$ and $G:\underline{\underline{B}} \rightarrow \underline{\underline{C}}$, define their *composition* $FG:\underline{\underline{A}} \rightarrow \underline{\underline{C}}$ by $A|FG| = (A|F|)|G|$ for $A \epsilon |\underline{\underline{A}}|$, and $f(FG) = (fF)G$ for $f \epsilon \underline{\underline{A}}$. Then:

(1) the composite FG is a functor;

(2) composition of functors is associative: i.e., if $H:\underline{\underline{C}} \rightarrow \underline{\underline{D}}$ is also a functor, then $(FG)H = F(GH)$; and

(3) given a category $\underline{\underline{B}}$, the functor $1_{\underline{\underline{B}}}:\underline{\underline{B}} \rightarrow \underline{\underline{B}}$ defined by $B|1_{\underline{\underline{B}}}| = B$ for $B \epsilon |\underline{\underline{B}}|$ and $f1_{\underline{\underline{B}}} = f$ for $f \epsilon \underline{\underline{B}}$, is an identity for composition, in the sense that the equations

$$F1_{\underline{\underline{B}}} = F \quad \text{and} \quad 1_{\underline{\underline{B}}}G = G$$

hold; call $1_{\underline{\underline{B}}}$ the *identity functor* on $\underline{\underline{B}}$.

Proof. All these assertions are easy, but we verify (1) for illustration. Let $f,g \epsilon \underline{\underline{A}}$, such that fg is defined. Then $(fg)(FG) = ((fg)F)G = (fFgF)G$ $= (fFG)g(FG)$. Let $A \epsilon |\underline{\underline{A}}|$. Then $1_A(FG) = (1_AF)G = 1_{AF}G = 1_{AFG}$. □

Example 2.19. Let **Cat** denote the category with categories as objects, functors as morphisms, and with composition and identities as in Proposition 2.2. Then Proposition 2.2 provides the necessary associative and identity laws, so that **Cat** is indeed a category. Its "size" is clearly stupendous in some sense, and one might wonder if it is an object in itself. The reader worried about this and related foundational matters is again referred to Mac Lane (1971a, 1971b) or Feferman (1971). One easy way out is to consider instead the category $\mathbf{Cat}_S$ with only *small* categories as objects. Clearly $\mathbf{Cat}_S$ is a full subcategory of **Cat**, and is not itself small. □

Example 2.20. As a first use of **Cat**, we give a functor for the construction of Example 2.5 which "regards" as monoid as a category. Since the results of this construction are categories, the target of the functor will have to be **Cat** (actually small categories will suffice). In fact, for $M\in|\mathbf{Mon}|$, let $MF\in|\mathbf{Cat}|$ have $|MF| = \{e\}$, have morphisms $m:e \to e$ for each $m\in M$, and have composition induced from multiplication in M (see Example 2.5). For $h:M_0 \to M_1$ in **Mon**, let $hF:M_0F \to M_1F$ be the functor described in Example 2.17 above. For $F:\mathbf{Mon} \to \mathbf{Cat}$ to be a functor, we must verify that $(hh')F = hFh'F$ and $1_MF = 1_{MF}$, which is very straightforward.

This functor is clearly injective, and its image is the full subcategory of **Cat** whose objects are all the small categories with exactly one object. In fact, when corestricted to this subcategory of **Cat**, F is *bijective*, thus giving an example of an isomorphism of categories. We let $\mathbf{Mon}_C$ denote this full subcategory of **Cat** (small one object categories as objects). Then F *embeds* **Mon** into **Cat**, with image $\mathbf{Mon}_C$. □

Example 2.21. A case very similar to the above is quosets (see Example 2.4). First, the category $\underline{\underline{\mathrm{Quo}}}$ of quosets has quosets as objects, and has order-preserving morphisms; i.e., a function $h:Q \to Q'$ is in $\underline{\underline{\mathrm{Quo}}}\,(Q,Q')$ iff $q_0 \leq q_1$ in Q implies $q_0 h \leq q_1 h$ in Q'. Now consider the functor $F:\underline{\underline{\mathrm{Quo}}} \to \underline{\underline{\mathrm{Cat}}}$ which constructs from $Q \epsilon |\underline{\underline{\mathrm{Quo}}}|$ a category as described in Example 2.3, and which takes a morphism $h:Q \to Q'$ in $\underline{\underline{\mathrm{Quo}}}$ to a functor $hF:QF \to Q'F$ (in $\underline{\underline{\mathrm{Cat}}}$) defined by $q|hF| = qh \epsilon |QF|$, and $<q_0,q_1>hF = <q_0 h, q_1 h> \epsilon Q'F$. It is now not difficult to see that F is indeed an injective functor whose image is contained in the full subcategory of all small categories $\underline{\underline{C}}$ with at most one morphism in each $\underline{\underline{C}}(A,B)$. But the image is properly contained in this subcategory, because such $\underline{\underline{C}}$ may contain morphisms $f:A \to B$ not of the form $<A,B>$. Thus we do not have isomorphism here; the relationship is the more subtle one of *equivalence*, which we shall discuss in Part 2. □

Some rather peculiar but important functors are called "forgetful functors" because they "construct" new objects by "forgetting" part of the structure of given objects ("destruction" might be a better description of this process than "construction"). The letter U is often used for such functors. It stands for "underlying", as one thinks of $U:\underline{\underline{A}} \to \underline{\underline{B}}$ as extracting the "underlying $\underline{\underline{B}}$ structure" from an object with $\underline{\underline{A}}$ structure. Often forgetful functors extract an underlying set.

Example 2.22. (see Examples 2.1 and 2.7). Define $U:\underline{\underline{\mathrm{Mon}}} \to \underline{\underline{\mathrm{Set}}}$ by sending $<M,\circ,e>$ to the set M and $h:<M,\circ,e> \to <M',\circ',e'>$ to the function $h:M \to M'$. Since composition in $\underline{\underline{\mathrm{Mon}}}$ *is* composition of functions and the identity homomorphism is the identity function, U preserves composition and identities, and is thus a functor. □

Example 2.23. (see Example 2.8). Define $U:\underline{\underline{Mach}} \to \underline{\underline{Aut}}$ by sending $\langle X,S,Y,\delta,\sigma,\beta\rangle$ to $\langle X,S,\delta\rangle$, and $\langle a,b,c\rangle:\langle X,S,Y,\delta,\sigma,\beta\rangle \to \langle X',S',Y',\delta',\sigma',\beta'\rangle$ to $\langle a,b\rangle:\langle X,S,\delta\rangle \to \langle X',S',\delta'\rangle$. U extracts the underlying automaton from a machine. □

Notice we now feel free to be less detailed in explaining and verifying examples. Both the above are instances of functors $\underline{\underline{Alg}}_{\Sigma} \to \underline{\underline{Alg}}_{\Sigma'}$, where $\Sigma' \subseteq \Sigma$ (see Example 2.7), in the sense that for each index i, $\Sigma_i' \subseteq \Sigma_i$. We shall not give here a precise general description of this process of forgetting structure, sometimes known as "reduction" in universal algebra, but we do wish to define its simplest general subcase.

Example 2.24. (see Example 2.7). Define $U:\underline{\underline{Alg}}_{\Sigma} \to \underline{\underline{Set}}$ by sending $\langle A,\alpha\rangle$ to A and $h:\langle A,\alpha\rangle \to \langle A',\alpha'\rangle$ in $\underline{\underline{Alg}}_{\Sigma}$ to $h:A \to A'$ in $\underline{\underline{Set}}$. This is the general underlying set functor for Σ-algebras. □

Example 2.25. Free Monoids. There is a functor $F:\underline{\underline{Set}} \to \underline{\underline{Mon}}$ corresponding to the free construction of a monoid X^* from a set X which we have been using so much. Thus, $XF = X^*$ and for $a:X \to X'$, $aF = a^*:X^* \to X'^*$. This is easily seen to be a functor (that is, $(1_X)^* = 1_{X*}$ and $(ab)^* = a^*b^*$), and is more conveniently denoted $^*:\underline{\underline{Set}} \to \underline{\underline{Mon}}$. □

There is a vague intuitive sense in which the above functor and $U:\underline{\underline{Mon}} \to \underline{\underline{Set}}$ are "complementary", in that one "freely does" what other "freely undoes". This kind of relationship holds in general for "free constructions", and is considered in detail in ADJ II under the name of adjointness.

Example 2.26. The monoid of an automaton. (see Example 2.18). Let $A:\underline{\underline{M}} \to \underline{\underline{Set}}$ be an M-automaton (as in Example 2.18), and let us consider $\underline{\underline{M}}A$, the image of $\underline{\underline{M}}$ under the functor A. Since $\underline{\underline{M}}$ has just one object, so does $\underline{\underline{M}}A$, and it is obvious that as a subset of $\underline{\underline{Set}}$, $\underline{\underline{M}}A$ is closed under composition (in fact, $mA \circ m'A = mm'A$). Therefore $\underline{\underline{M}}A$ is a one object small category (see also Proposition 2.1), i.e., a monoid. Viewed as such, it is called the monoid of A (see Medvedev (1956), Myhill (1957)). □

2.3 OPPOSITE CATEGORIES AND MULTIFUNCTORS

This subsection shows how to handle constructions which take several arguments as data, some of them possibly "contravariantly" (see later) while still using exactly the same definition of functor. We feel it is superfluous and confusing to introduce more than one basic notion of "functor", and we shall always understand this word in the sense of Definition 2.3. However, we do need to make use of some auxiliary constructions on categories to achieve this effect. The first of these in effect reverses all arrows.

Definition 2.4. Given a category $\underline{\underline{C}}$, given by the sextuple $\langle |\underline{\underline{C}}|, \underline{\underline{C}}, \partial_0, \partial_1, 1, \circ \rangle$, its opposite category $\underline{\underline{C}}^{op}$ is given by the sextuple $\langle |\underline{\underline{C}}|, \underline{\underline{C}}, \partial_1, \partial_0, 1, \circ^{op} \rangle$, where $f \circ^{op} g$ is defined iff $g \circ f$ is defined (in $\underline{\underline{C}}$), and is then defined to be $g \circ f$. Note the reversal of ∂_0 and ∂_1 in going from $\underline{\underline{C}}$ to $\underline{\underline{C}}^{op}$. We often write $f^{op}: B \to A$ for the morphism in $\underline{\underline{C}}^{op}(B,A)$ corresponding to $f: A \to B$ in $\underline{\underline{C}}(A,B)$. The function 1 remains unchanged.

It is straightforward to verify that $\underline{\underline{C}}^{op}$ is a category; associativity and identity in $\underline{\underline{C}}^{op}$ come from those in $\underline{\underline{C}}$, but one must keep clear in which category one is working. Note that $f^{op}g^{op} = (gf)^{op}$. $\underline{\underline{C}}^{op}$ is often called the dual of $\underline{\underline{C}}$ because it allows one to dualize concepts by reversing arrows. This will be abundantly illustrated in Section 4, but for the moment we hope what we mean will be adequately suggested by noting that "source" is dual to "target".

Example 2.27. (see Example 2.10). Recall that $\underline{\underline{N}}$ is the full subcategory of $\underline{\underline{Set}}$ with elements of ω as objects. We now give a direct intuitive interpretation of $\underline{\underline{N}}^{op}$. Its objects are of course integers. But a morphism $n \to m$ in $\underline{\underline{N}}^{op}$ is a morphism $m \to n$ in $\underline{\underline{Set}}$ (i.e., a function). Let us now imagine that we are given a countable number of copies of a "register" R in which some sort of "data" can be stored, and that we wish to describe succinctly the various multiple simultaneous (finitary) transfer operations which are possible, i.e., those which involve transfer from some finite number n of registers to some finite number m of registers, and leave all others unchanged. In fact, such a transfer can be described by a set function $f{:}m \to n$, in which $f{:}\ i \mapsto j$ (i.e., $if = j$) means that the i^{th} register is given the (former) contents of the j^{th} register. (It is of course possible that $m = n$ and f is the identity function, in which case there is no change in any register.) That is, the transfer $n \to m$ is described by $f{:}m \to n$ in $\underline{\underline{Set}}$. Moreover, the algebra of the transfers is the algebra of such set functions. In particular, composition of transfers is given by the composition of the set functions, but in the opposite order. In other words, it is the category $\underline{\underline{N}}^{op}$ rather than $\underline{\underline{N}}$ which describes the transfers. Other operations which are available in this context, such as tupling, will be discussed later.

The above intuitive discussion can be given a precise mathematical character as follows: Let R be a fixed set (the "register" whose elements are items of "data" to be "transferred"). Then $f{:}m \to n$ in $\underline{\underline{Set}}$ describes a projection function $R^n \to R^m$, sending $<r_0,\ldots,r_{n-1}>\in R^n$ to $<r_{0f},r_{1f},\ldots,r_{(m-1)f}>\in R^m$ (i.e., sending $r{:}n \to R$ in R^n to the composite $fr{:}m \to R$, in R^m). Denote this function $R^f{:}R^n \to R^m$. In fact, R^- gives a functor $\underline{\underline{N}}^{op} \to \underline{\underline{Set}}$, and $\underline{\underline{N}}^{op}$ contains the algebra of projections among the powers of R. □

Example 2.28. As a flourish of categorical dexterity, we define the functor $^{op}:\underline{\underline{\mathrm{Cat}}} \to \underline{\underline{\mathrm{Cat}}}$ corresponding to the construction of Definition 2.4. Since we know ^{op} for the objects, we need only define it for the morphisms in $\underline{\underline{\mathrm{Cat}}}$, i.e., the functors. Given $F:\underline{\underline{A}} \to \underline{\underline{B}}$, define $F^{op}:\underline{\underline{A}}^{op} \to \underline{\underline{B}}^{op}$ by $A|F^{op}| = A|F|$ for $A\epsilon|\underline{\underline{A}}^{op}| = |\underline{\underline{A}}|$ and $f^{op}F^{op} = (fF)^{op}$ for $f^{op}\epsilon\underline{\underline{A}}^{op}$. We check that F^{op} preserves composition. $(f^{op}g^{op})F^{op} = ((gf)^{op})F^{op} = ((gf)F)^{op} = (gFfF)^{op} = (fF)^{op}(gF)^{op} = (f^{op}F^{op})(g^{op}F^{op})$. □

*Fact 2.3. $^{op\ op} = \underline{\underline{\mathrm{Cat}}}$; i.e., the composite of the functor $^{op}:\underline{\underline{\mathrm{Cat}}} \to \underline{\underline{\mathrm{Cat}}}$ with itself is the identity functor $\underline{\underline{\mathrm{Cat}}}$: $\underline{\underline{\mathrm{Cat}}} \to \underline{\underline{\mathrm{Cat}}}$. In particular, if $\underline{\underline{C}}$ is a category, $(\underline{\underline{C}}^{op})^{op} = \underline{\underline{C}}$.

Proof. $|(\underline{\underline{C}}^{op})^{op}| = |\underline{\underline{C}}^{op}| = |\underline{\underline{C}}|$ and $(\underline{\underline{C}}^{op})^{op}(A,B) = \underline{\underline{C}}^{op}(B,A,) = \underline{\underline{C}}(A,B)$ give that $(\underline{\underline{C}}^{op})^{op} = \underline{\underline{C}}$. Now we need only show the same for a functor $F:\underline{\underline{A}} \to B$. In fact, $A|F^{op\ op}| = A|F^{op}| = A|F|$ and $fF^{op\ op} = f^{op\ op}F^{op\ op} = (f^{op}F^{op})^{op} = (fF)^{op\ op} = fF$. □

Perhaps the most important use of opposite categories is in discussing so called "contravarient functors", i.e., "functors" which "reverse the order of composition". They are not really a different kind of functor at all, but rather a phenomenon which we treat using ordinary functors and opposite categories.

Example 2.29. It would seem that the assignments $A \mapsto 2^A$ (recall $2 = \{0,1\}$) and $(f:A \to B) \mapsto 2^f:2^B \to 2^A$ by $h2^f = fh$, for $h\epsilon 2^B$ (i.e., for $h:B \to 2$) should define a functor $2^-:\underline{\underline{\mathrm{Set}}} \to \underline{\underline{\mathrm{Set}}}$. But this is not the case. If $g:B \to C$, we should have $2^{fg} = 2^f 2^g$, but $2^f 2^g$ is not even defined, since $2^f:2^B \to 2^A$ while $2^g:2^C \to 2^B$. However, $2^g 2^f$ is defined, and in fact, it equals 2^{fg}: for $h:C \to 2$, we have $h2^g 2^f = (h2^g)2^f = (gh)2^f = (fg)h = h2^{fg}$. Thus 2^- is contravariant, since it "reverses the order of composition", or more sophisticatedly, "reverses arrows". In fact, we can define $2^{-}:\underline{\underline{Set}}^{op} \to \underline{\underline{Set}}$ by $A \mapsto 2^{A}$ and $(f^{op}:B \to A) \mapsto 2^{f}:2^{B} \to 2^{A}$, where this 2^{f} is defined exactly as above, noting that $f^{op}:B \to A$ in $\underline{\underline{Set}}^{op}$ means $f:A \to B$ in $\underline{\underline{Set}}$. The fact that $2^{fg} = 2^{g}2^{f}$ proved above now gives functorality (it is easy to check out the identities).

$$\begin{tikzcd}
A \arrow[r, "f"] \arrow[dr, "fgh"'] & B \arrow[r, "g"] \arrow[d, "gh"] & C \arrow[dl, "h"] \\
 & 2 &
\end{tikzcd}$$

Because $A\underset{\sim}{P}$ is isomorphic to 2^{A}, the above construction can be viewed as giving us another "power set functor", this time a "contravariant power set functor" $\underset{\sim}{P}^{*}$ rather than the earlier *covariant* one. In fact, when looked at this way, for $f:A \to B$, $f\underset{\sim}{P}^{*}:B\underset{\sim}{P} \to A\underset{\sim}{P}$ is most reasonably thought of as the *inverse image*, taking $B_0 \subseteq B$ to $B_0 f^{-1} \subseteq A$. We leave the reader to check that $\underset{\sim}{P}^{*}:\underline{\underline{Set}}^{op} \to \underline{\underline{Set}}$ is indeed a functor, either directly from the above definition, or through the correspondence with 2^{-}. Later we shall define a notion of *natural isomorphism* for functors, and 2^{-} with $\underset{\sim}{P}^{*}$ will be an instance. □

Example 2.30. *Hom Functors.* A slightly different notation for the functor 2^{-} suggests an important generalization. In fact, $2^{A} = \underline{\underline{Set}}(A,2)$, and for $f:A \to A'$ it is reasonable to write 2^{f} as $\underline{\underline{Set}}(f,2):\underline{\underline{Set}}(A',2) \to \underline{\underline{Set}}(A,2)$. We can then replace 2 by any set B, to obtain a functor $\underline{\underline{Set}}(\_,B):\underline{\underline{Set}}^{op} \to \underline{\underline{Set}}$. However, there is nothing about the properties we have used which is special to the category of sets. Thus, let $\underline{\underline{C}}$ be any (locally small) category, and let $A,B \in |\underline{\underline{C}}|$. Then $\underline{\underline{C}}(A,B) \in |\underline{\underline{Set}}|$. Now let $f:A \to A'$ in $\underline{\underline{C}}$, and define $\underline{\underline{C}}(f,B):\underline{\underline{C}}(A',B) \to \underline{\underline{C}}(A,B)$ in $\underline{\underline{Set}}$ by sending $h:A' \to B$ in $\underline{\underline{C}}(A',B)$ to $fh:A \to B$ in $\underline{\underline{C}}(A,B)$. That this defines a functor

$\mathbf{C}(\_,B):\mathbf{C}^{op} \to \mathbf{Set}$ is verified exactly as in the case $\mathbf{C} = \mathbf{Set}$ above. Such a functor is sometimes called a *contravariant hom functor*.

A *covariant hom functor* is obtained by fixing the first component instead of the second. Thus again let $\mathbf{C}$ be a (locally small) category, let $A\epsilon|\mathbf{C}|$, and let $f:B \to B'$ in $\mathbf{C}$. Then define $\mathbf{C}(A,f):\mathbf{C}(A,B) \to \mathbf{C}(A,B')$ by sending $h:A \to B$ in $\mathbf{C}(A,B)$ to $hf:A \to B'$ in $\mathbf{C}(A,B')$. It is once again easy to verify that this defines a functor $\mathbf{C}(A,\_):\mathbf{C} \to \mathbf{Set}$. □

Evidently, $\mathbf{C}(A,B)$ is a set-valued construction depending on both its arguments, "contravariantly" in the first and "covariantly" in the second. It should therefore be possible to regard $\mathbf{C}(\_,\_)$ as some kind of a functor. This is accomplished in essentially the same way that multivariable functions are regarded as plain ordinary functions in ordinary set theory: they are functions on the product of the various sets of allowable arguments for the "variables." For multivariable functors, we similarly use the product of the various categories involved as arguments.

Definition 2.5. Let $\mathbf{A}$ and $\mathbf{B}$ be categories. Then the *product category* $\mathbf{A} \times \mathbf{B}$ has object class $|\mathbf{A} \times \mathbf{B}| = |\mathbf{A}| \times |\mathbf{B}|$, and morphism class $\mathbf{A} \times \mathbf{B}$; $\partial_i:\mathbf{A} \times \mathbf{B} \to |\mathbf{A}| \times |\mathbf{B}|$ is defined by $<a,b> \mapsto <a\partial_i,b\partial_i>$ for $i = 0,1$; $1:|\mathbf{A} \times \mathbf{B}| \to \mathbf{A} \times \mathbf{B}$ is defined by $<A,B> \mapsto <1_A,1_B>$; and composition is defined by $<a,b><a',b'> = <aa',bb'>$.

We shall leave to the reader the task of verifying that $\mathbf{A} \times \mathbf{B}$ is a category. Validity of the axioms for pairs follows from validity for each component separately. There is also an appropriate notion of product for an arbitrary family of categories. It might be noted that this definition is again a specialization to categories of the corresponding general definition of product for many sorted algebras; although we did not give this definition in Section 1.5, the interested reader can presumably reconstruct it if he wishes (or see Section 4 in Part 2). A *multifunctor* is then a functor whose source is a product category.

Example 2.30 continued. Define $\mathbf{C}(\_,\_):\mathbf{C}^{op} \times \mathbf{C} \to \mathbf{Set}$, for $a:A' \to A$ and $b:B \to B'$, by defining $\mathbf{C}(a,b):\mathbf{C}(A,B) \to \mathbf{C}(A',B')$ by sending $f:A \to B$ in $\mathbf{C}(A,B)$ to $afb:A' \to B'$ in $\mathbf{C}(A',B')$. It is clear that $\mathbf{C}(1_A,1_B)$ is an identity function. We now check that $\mathbf{C}(\_,\_)$ preserves composition. Let $a':A'' \to A'$ and $b':B' \to B''$ in $\mathbf{C}$ also. Then, with a,b,f as before, $f\mathbf{C}(a,b)\mathbf{C}(a',b') = afb\mathbf{C}(a',b') = a'afbb' = f\mathbf{C}(a'a,bb')$. $\mathbf{C}(\_,\_)$ is called *the hom functor* for $\mathbf{C}$. □

Example 2.31. We show that binary Cartesian product for sets gives a functor $\times:\mathbf{Set} \times \mathbf{Set} \to \mathbf{Set}$. In fact, $(A,B)\times$, for A,B sets, is the set $A\times B$, and $(f,g)\times$ for $f:A \to A'$, $g:B \to B'$ functions, is the function $f\times g:A\times B \to A'\times B'$ defined by $\langle a,b\rangle \mapsto \langle af,bg\rangle$. Then clearly $(1_A,1_B)\times = 1_A\times 1_B = 1_{A\times B}:A\times B \to A\times B$, and for $f':A' \to A''$ and $g':B' \to B''$, $(ff',gg')\times = (f,g)\times\circ(f',g')\times:A\times B \to A''\times B''$ (i.e., $(ff')\times(gg') = (f\times g)\circ(f'\times g')$). □

It should be reasonably clear from the above how we treat functors of three or more arguments. For example, suppose $F(X,Y,Z)$ arises from a construction having data items of three different types as inputs, the first "contravariant"; specifically, perhaps $F(X,Y,Z)$ is $\mathbf{Set}(X,Y\times Z)$ for X,Y,Z sets. Let the categories corresponding to the data items be $\mathbf{X},\mathbf{Y},\mathbf{Z}$, respectively. Then the construction should be represented by a functor $F:\mathbf{X}^{op} \times \mathbf{Y} \times \mathbf{Z} \to \mathbf{W}$, where $\mathbf{W}$ is the category corresponding to the structure of the construction's result. In the specific case above, we have $\mathbf{Set}(\_,\_\times\_):\mathbf{Set}^{op}\times \mathbf{Set} \times \mathbf{Set} \to \mathbf{Set}$ In general, one forms the product of the data categories, attaching "op" to them as appropriate.

(The preceding discussion is of course not intended to be technically precise; it is, in a sense, an elaboration of the second doctrine of Section 0.2.).

It is worth pointing out explicitly that, just as with functions, functors of fewer arguments can be obtained from multifunctors by fixing some of the arguments to be constants. If $F:\underline{\underline{A}}\times\underline{\underline{B}} \to \underline{\underline{C}}$ is a functor and $A\in|\underline{\underline{A}}|$, then $(A,\_)F:\underline{\underline{B}} \to \underline{\underline{C}}$ defined for $B\in|\underline{\underline{B}}|$ by $B(A,\_)F = (A,B)F$, and for $g\in\underline{\underline{B}}$ by $g(A,\_)F = (1_A,g)F$, is also a functor. Similarly for $(\_,B)F:\underline{\underline{A}} \to \underline{\underline{C}}$. These facts are easily verified from the definitions involved, and they readily extend to functors of three or more arguments.

Example 2.31 continued. Since $\times:\underline{\underline{\underline{\mathrm{Set}}}} \times \underline{\underline{\underline{\mathrm{Set}}}} \to \underline{\underline{\underline{\mathrm{Set}}}}$ is a functor, so are $\_{}^{\times}S:\underline{\underline{\underline{\mathrm{Set}}}} \to \underline{\underline{\underline{\mathrm{Set}}}}$ and $S^{\times}\_:\underline{\underline{\underline{\mathrm{Set}}}} \to \underline{\underline{\underline{\mathrm{Set}}}}$, for S a fixed Set. □

Example 2.30 continued. From $A,B\in|\underline{\underline{C}}|$ and $\underline{\underline{C}}(\_,\_):\underline{\underline{C}}^{op} \times \underline{\underline{C}} \to \underline{\underline{\underline{\mathrm{Set}}}}$, we obtain $\underline{\underline{C}}(\_,B):\underline{\underline{C}}^{op} \to \underline{\underline{C}}$ and $\underline{\underline{C}}(A,\_):\underline{\underline{C}} \to \underline{\underline{C}}$ as given in Example 2.30. Letting $\underline{\underline{C}} = \underline{\underline{\underline{\mathrm{Set}}}}$, we obtain the functors conventionally denoted $B^{-}:\underline{\underline{\underline{\mathrm{Set}}}}^{op} \to \underline{\underline{\underline{\mathrm{Set}}}}$ and $\_{}^{A}:\underline{\underline{\underline{\mathrm{Set}}}} \to \underline{\underline{\underline{\mathrm{Set}}}}$, as $\underline{\underline{\underline{\mathrm{Set}}}}(\_,B)$ and $\underline{\underline{\underline{\mathrm{Set}}}}(A,\_)$ respectively. For $B = 2 = \{0,1\}$, we get the power set functor of Example 2.29 again. □

*2.4 THE WEAKER DEFINITION OF CATEGORY

This subsection contains the rather pedantic proofs involved in showing the equivalence of the two definitions of category given in Section 2.1. These were deferred so that the examples of categories could immediately follow the definition. There is also some closely related material.

Proposition 2.4. Definitions 1 and 1A are equivalent.

Proof. It is immediate that Definition 1A is weaker than Definition 1, since axioms (1) and (2) appear identically in both, axioms (3A) and (4A) are evidently weaker forms of axioms (3) and (4), and axiom (5) appears only in Definition 1. It remains to show that (5) and the additional parts of (3) and (4) follow from just (1), (2), (3A), and (4A).

We first show the converse of (3A). Assume that $f\partial_1 = g\partial_0$, and let $h = g\partial_1 1$. Then by (4A), $(f \circ g) \circ h$ is defined, and so in particular is $(f \circ g)$.

We now prove the axiom (4). Assume $f \circ g$ and $g \circ h$ are defined. Then by (3A), $f\partial_1 = g\partial_0$ and $g\partial_1 = h\partial_0$. Now (4A) applies, to give both $(f \circ g) \circ h$ and $f \circ (g \circ h)$ defined, and equal, as desired.

Finally, we prove axiom (5). Assume $f \circ g$ is defined. Then by axiom (3A), $f\partial_1 = g\partial_0$. Let $A = g\partial_1$ and $h = A1 = g\partial_1 1$. Then by axiom (1), $h\partial_0 = A1\partial_0 = A = g\partial_1$, so that by axiom (4A), $(f \circ g) \circ h$ is defined. Therefore, by axiom (3A), $(f \circ g)\partial_1 = h\partial_0 = A = g\partial_1$, as desired. The other claim is proved similarly. □

It is perhaps also worthwhile to mention the following simple results while we are on this subject.

Fact 2.5. Let f be a morphism in a category $\underline{\underline{C}}$. Then $f\partial_i 1\partial_j = f\partial_i$, for all $i,j = 0,1$.

Proof. Let $f\partial_i = A$. Then $f\partial_i 1\partial_j = A1\partial_j = A = f\partial_i$, by axiom (1.). □

Call $e\epsilon\underline{\underline{C}}$ an *identity morphism* iff $f \circ e = f$ and $e \circ g = g$, whenever such compositions are defined.

Fact 2.6. For each $A\epsilon|\underline{\underline{C}}|$, $A1$ is an identity morphism. If e is an identity morphism in $\underline{\underline{C}}$, then $e\partial_0 = e\partial_1$. If e and e' are identity morphisms in $\underline{\underline{C}}$ such that either $e\partial_0 = e'\partial_0$ or $e\partial_1 = e'\partial_1$, then $e = e'$. In particular, if e is an identity morphism, then $e = e\partial_0 1$.

Proof. Suppose $f \circ A1$ is defined. Then by Axiom (3A), $f\partial_1 = (A1)\partial_0$. By Axiom (1), $(A1)\partial_0 = A$, so that $f\partial_1 = A$ also. Therefore $f \circ A1 = f \circ (f\partial_1 1)$.

But this is equal to f by Axiom (2). The proof that $A1 \circ g = g$, if defined, is similar. Thus $A1$ is an identity morphism.

Let $e \in \underline{\underline{C}}$ be an identity morphism, and suppose $f \circ e$ is defined (some such f always exists; for example $f = e\partial_0 1$). By assumption $f \circ e = f$. Therefore $f \circ e = (f \circ e) \circ e$, and the latter composition is defined. Then by Axiom (3), $(f \circ e)\partial_1 = e\partial_0$; and by Axiom (5), $(f \circ e)\partial_1 = e\partial_1$. Thus $e\partial_0 = e\partial_1$ as claimed.

Now suppose e,e' identity morphisms with $e\partial_0 = e'\partial_0$. Then $e\partial_1 = e\partial_0 = e'\partial_0$, so that $e \circ e'$ is defined. Since both e,e' are identities, $e \circ e' = e$ and $e \circ e' = e'$. Thus $e = e'$. On the other hand, if $e\partial_1 = e'\partial_1$, since $e\partial_1 = e\partial_0$ and $e'\partial_1 = e'\partial_0$, we have again $e\partial_0 = e'\partial_0$, and therefore $e = e'$.

Since $e\partial_0 1$ is an identity morphism, and $e\partial_0 1\partial_0 = e\partial_0$ by Fact 2.5, we conclude that $e = e\partial_0 1$. □

It follows that the image of the identity $1:|\underline{\underline{C}}| \to \underline{\underline{C}}$ is the class of identities in $\underline{\underline{C}}$, and that the function 1 (co)restricted to this class is a bijection. This suggests that perhaps one could formulate a definition of category in which the role of the objects is played instead by the identities. This would be a one-sorted or "morphisms only" (also called "nonobjective") definition, and can indeed be found in the literature (e.g., see Mitchell (1965), p. 2).

ACKNOWLEDGEMENTS

The authors acknowledge with warmth and gratitude the sustaining and irreplaceable love and support of special people very close to them (each his own); the encouragement and support of institution and the people therin, IBM (for all of us), UCLA (Joseph Goguen), Edinburgh University and U.K. Science Research Council (James Thatcher); and, the understanding and patience of those helping to prepare this manuscript, Eleanor Kerse of Edinburgh University and especially Martha Pierce of IBM Yorktown.

BIBLIOGRAPHY

This is a collection of work related to the junction between computer science and category theory. It therefore contains material from each partner, as well as the fruit of their union. We cannot hope to include all possible applicable algebra or algebracizable computer science, nor even all applicable categorical algebra and categoricizable computer science. We do hope for completeness in the area of the junction itself, but know we are unlikely to be able to achieve it, in part because of the rapid growth, and the intellectual and geographical diversity of the subject. Suggestions from readers for additions would be greatly appreciated.

In the belief that categorical algebra is only algebra in a more extreme or pure form, so that most algebra is categoricizable, we have included here material which is algebraic in character, style, or form, but not explicitly categorical. This includes both pure algebra and its applications to computer science. Moreover, we have also included applications to certain fields closely connected to computer science, particularly system science. Finally, we have included a fair amount of applied material which is not explicitly algebraic in character, but which seems to us to be particularly "algebraicizable" or which is necessary background for applications which are algebraic.

Works are listed alphabetically by author, and then by year. The order of publications with the same author(s) and year is arbitrary, but fixed by a small letter of the alphabet. Following most citations is a descriptor, to be interpreted by the following scheme: capitals for subject, C for computer science, M for mathematics, S for system science, and 0 for other (such as biology, psychology, etc.); small letters for method and style, f for fairly algebraic (or very algebraicizable), a for algebraic, and c for categorical (default is algebraicizable); and (optionally) numbers for level, 1 for introductory text, survey, or exposition, 2 for advanced level text, survey or exposition, and 3 for research level (sometimes we may interpolate, e.g., 2 1/2 means both advanced expository and research). The default is normally 3. For example CSc2 1/2 means the work concerns both computer and system science, is categorical, and is both research and expository. Our classifications are very superficial and subjective, sometimes outright guesses. We in no way intend to pass judgement on the works involved, but only to aid the reader in making selections for further exploration.

Aho, A. V.

(1973) (Ed.) *Currents in Computing*, Prentice Hall, New Jersey (1973). C2.

Aho, A. V. and Ullman, J. D.

(1968) "Translating on a context-free grammar," *Proceedings* ACM Symposium on Theory of Computing (1968) 93-112. C.

Alagić, S.

(1973) "Natural state transformation," Report 73B-2, Computer and Information Sciences Department, Univ. of Mass. (1973). Cc.

Arbib, M. A.

(1966) "Categories of (M,R)-systems," *Bull. of Math. Bio-physics 28* (1966). Oc.

(1967) See Kalman, Falb and Arbib.

(1968) See Give'on and Arbib.

(1968a) (Ed.) *Algebraic Theory of Machines, Languages and Semigroups*. Academic Press, New York (1968). CSa.

(1969) *Theories of Abstract Automata*, Prentice Hall, New Jersey (1969). Cfl 1/2.

Arbib, M.A. and Give'on, Y.

(1968) "Algebra automata I: Parallel programming as a prolegoma to the categorical approach," *Information Control 12* (1968) 331-345. Cc.

Arbib, M. A. and Manes, E.G.

(1972) "Decomposable machines and simple recursion," Computer and Information Sciences Technical Report 72B-2, Univ. of Mass. (1972). Cc.

(1972a) "Machines in a category," Report 73B-1, Computer and Information Sciences Dept., Univ. of Mass., June (1972). CSc2 1/2.

(1973) "Adjoint machines, state behavior machines, and duality," Report 73B-1, Computer and Information Sciences, Univ. of Mass., February (1973). CSc.

Arbib, M.A. and Zeiger, H.P.

(1969) "On the relevance of abstract algebra to control theory," *Automatica 5* (1969) 589-606. Sc2 1/2.

Bǎianu, I.

(1969) See Comorasan and Bǎianu.

Bǎianu, I. and Marinescu, M.

(1968) "Organismic supercategories: I. Proposals for a general unitary theory of systems," *Bull. of Math. Bio-physics 30* (1968) 625-635. Oc.

Bainbridge, E.S.

(1972) "A unified minimal realization theory with duality," Ph.D. Dissertation, Dept. of Computer and Communication Sciences, Univ. of Michigan, November (1972). CSMc.

Bar-Hillel, Y., Perles, M. and Shamir, E.

(1961) "On formal properties of phrase structure grammars," Z. Phonetik, Sprach. Kummunikationsforsch 14 (1961) 143-172. CM.

Beckman, F.S.

(1970) "Categorical notions and duality in automata theory," IBM T. J. Watson Res. Ctr. Report RC 2977, July (1970). Cc2.

Bekic, H.

(1969) "Definable operations in general algebra, and the theory of automata and flowcharts," Report IBM Laboratory Vienna, (1969). CM.

Benson, D.B.

(1970) "Syntax and semantics: A categorical view," Information and Control 17 (1970) 145-160. Cc.

(1971) "Semantic preserving translations," Manuscript, Computer Science Dept., Washington State Univ., Pullman, Washington (1971). Cc.

(1972) "An abstract machine theory for formal language parsers," Manuscript, Computer Science Dept., Washington State Univ., Pullman, Washington (1972). Cc.

(1973) "The basic algebraic structures in categories of derivations," to appear. Cc.

Birkhoff, G.

(1938) "On the structure of abstract algebras," Proceedings Cambridge Phil. Soc. 31 (1938) 433-454. Ma.

(1960) Lattice Theory, Amer. Math. Soc. Colloq. Pub. 25 New York (1948). (revised edition) (1960). Ma2.

(1967) See Mac Lane and Birkhoff.

Birkhoff, G. and Lipson, D.

(1970) "Hetrogeneous algebras," J. Combinatorial Theory 8 (1970) 115-133. Ma.

Brainerd, W.S.

(1967) "Tree generating systems and tree automata," Ph.D. Dissertation, Purdue University, June (1967). Cf.

(1968) "The minimization of tree automata," Information and Control 13 (1968) 484-491. Ca.

Büchi, J.R.

(1959) "Weak second-order arithmetic and finite automata," Univ. of Michigan, Logic of Computer Group Technical Report, September (1959;: Z. Math. Logik Grundlagen Math. 6 (1960) 66-92. MC.

(1962) "Mathematische theorie des verhaltens endlicher automaten," Z. Angewandte Math. und Mech. 42 (1962) 9-16. MC.

(1964) "Regular canonical systems," Arch. Math. Logik Grundlagenforschung 6 (1964) 91-111. CM.

Büchi, J.R. and Elgot, C.C.

(1958) "Decision problems of weak second-order arithmetics and finite automata," Abstract 553-112, Notices Amer. Math. Soc. 5 (1958) 834. MC.

Büchi, J.R. and Wright, J.B.

(1960) "Mathematical theory of automata," Notes on material presented by J.R. Buchi and J.B. Wright, Communication Sciences 403 Fall (1960) The Univ. of Michigan. CMa.

Bucur, J. and Deleanu, A.

(1969) Theory of Categories, Academic Press, New York (1969). Mc2.

Burks, A.W. and Wright, J.B.

(1962) "Sequence generators and digital computers," in Recursive Function Theory, (Ed. J.C.E. Dekker) Proceedings Symposia in Pure Math. V, Amer. Math. Soc., Providence, R. I. (1962) 139-199. CM.

Burstall, R.M.

(1969) "Proving properties of programs by structural induction," Computer Journal, February (1969). Cf.

(1972) "An algebraic description of programs with assertions, verification and simulation," Proceedings ACM Conference on Proving Assertions about Programs, Las Cruces, New Mexico (1972) 7-14. Cc.

(1972a) "Some techniques for proving correctness of programs which alter data structures," Machine Intelligence 7 (Eds. B. Meltzer and D. Michie) Edinburgh University Press (1972) 23-50. Cf.

Burstall, R.M. and Landin, P.J.

(1969) "Programs and their proofs: An algebraic approach," Machine Intelligence 4 (Eds. B. Meltzer and D. Michie), Edinburgh University Press (1969) 17-43. Ca.

Cadiou, J.M.

(1972) See Manna and Cadiou

Chomsky, N.

(1961) "On the notion of 'rule of grammar'," Structures of Language and its Mathematical Aspects, Proceedings Symposia Appl. Math, XII, Amer.Math. Soc., Providence, R.I. (1961) 6-24. OCM.

(1965) Aspects of the Theory of Syntax, The M.I.T. Press, Cambridge, Massachusetts (1965). OCM.

Chomsky, N. and Schutzenberger, M.P.

(1963) "The algebraic theory of context-free languages," Computer Programming and Formal Systems (Ed. P. Braffort and D. Hirschberg), North-Holland Co., Amsterdam (1963) 118-161. Ca.

Cohen, P.M.

(1966) Set Theory and the Continuum Hypothesis, W. A. Benjamin, New York (1966). Ml.

Cole, J.C.

(1971) "Categories of sets and models of set theory," Aarhus University, Matematisk Institut, Preprint Series, 52 (1971). Mc2.

Comorasan, S. and Bǎianu, I.

(1969) "Abstract representations of biological systems in supercategories," Bull. of Math. Bio-physics 31 (1969) 59-70. Oc.

Culik, K.

(1967) "On some transformations in context-free grammars and languages," Czechoslovak Math. J. 17 (1967) 278-311. C.

Davis, R.

(1969) "Universal coalgebra and categories of transition systems," Math. Sys. Th. 4 (1969) 91-95. MCc.

de Bakker, J.W.

(1969) See Scott and de Bakker.

(1971) "Recursive procedures," Mathematical Centre Tracts 24, Mathematical Centre, Amsterdam (1971). CM2.

(1971a) "Recursion induction and symbol manipulation," Proceedings MC-25 Informatica Symposium, Mathematical Centre Tracts 37, Mathematical Centre, Amsterdam (1971). CM.

de Bakker, J.W. and de Roever, W.P.

(1973) "A calculus for recursive program schemes," to appear in Proceedings, IRIA Symposium on Automata, Formal Languages and Programming, North-Holland, Amsterdam (1973). CM.

de Bakker, J. W. and Meertens, L.G.L. th.

(1972) "Simple recursive program schemes and inductive assertions," Mathematical Centre Tracts 142 Amsterdam (1972). CM.

Deleanu, A.

(1969) See Bucur and Deleanu.

Desoer, C.A.

(1963) See Zadeh and Desoer.

de Roever, W.P.

(1973) See de Bakker and de Roever.

Dieudoune, J.

(1971) See Grothendieck and Dieudoune.

Doner, J.E.

(1965) "Decidability of the weak second-order theory of two successors," Abstract No. 6ST-468, Notices American Math. Soc. 12 (1965) 819. MCa.

(1970) "Tree acceptors and some of their applications," J. Comp. and Sys. Sci. 4 (1970) 406-451. Ca.

Ehrig, H.

(1973) "Axiomatic theory of systems and systematics," Report 73-05, Technische Universitat Berlin, Kybernetic Fachbereich, March (1973). CSMc2 3/4.

Ehrig, H. and Pfender, M.

(1972) Kategorien und Automaten, de Gruytes, Berlin (1972). Cc2.

Eilenberg, S.

(1969) "The algebraicization of mathematics," The Mathematical Sciences Essays for COSRIMS, MIT Press, Cambridge, Mass. (1969) 153-160. Cacl.

Eilenberg, S. and Elgot, C.C.

(1969) "Iteration and recursion," Proceedings Nat. Acad. of Sci. 61 (1968) 378-379. Cc.

(1970) Recursiveness, Academic Press, New York (1970). MCc2.

Eilenberg, S. and Kelly, G.M.

(1966) "Closed categories," Proceedings, Conference on Categorical Algebra, Springer-Verlag (1966) 421-562. Mc.

Eilenberg, S. and Mac Lane, S.

(1942) "Group extensions and homology," Ann. Math. 43 (1942) 757-831. Mc.

(1945) "General theory of natural equivalence," Trans. Amer. Math. Soc. 58 (1945) 231-294. Mc.

Eilenberg, S. and Schützenberger, M.

(1969) "Rational sets in commutative monoids," Journal of Algebra 13 (1969) 173-191. MCc.

Eilenberg, S. and Wright, J.

(1967) "Automata in general algebras," Information and Control 11 (1967) 52-70. Cc.

Elgot, C.C.

(1958) See Bűchi and Elgot.

(1961) "Decision problems of finite automaton design and related arithmetics," Trans. Amer. Math. Soc. 98 (1961) 21-51. CM.

(1968) See Eilenberg and Elgot.

(1970) See Eilenberg and Elgot.

(1970a) "The common algebraic structure of exit-automata and machines," Computing 6 (1970) 349-370. Cc.

(1971) "Algebraic theories and program schemes," Symp. on Semantics of Algorithmic Languages, (Ed. E. Engeler), Springer-Verlag (1971) 71-88. Cc.

(1972) "Remarks on one-argument program schemes," Formal Semantics of Programming Languages, (Ed. R. Rustin), Prentice-Hall, New Jersey (1972) 59-64. Cc2.

Engeler, E.

(1968) Formal Languages: Automata and Structures, Markham, Chicago (1968). CMc1 3/4.

Engelfriet, J.

(1971) "Bottomup and topdown tree transformations," Memorandum 19, Technische Hogeschool Turente, July (1971). Ca2 1/2.

Falb, D.L.

(1967) See Kalman, Falb and Arbib.

Feferman, S.

(1969) "Set-theoretical foundations of category theory," Reports of the Midwest Category Seminar III (ed. by S. Mac Lane), Lecture Notes in Mathematics, 106 Springer-Verlag (1969) 201-247. Mca.

Fischer, J.

(1968) "Grammars with macro-like productions," Proceedings, 9th Ann. Symp. on Switching and Automata Theory, (1968). C.

Floyd, R.W.

(1967) "Assigning meanings to programs," Proceedings, Symp. on Appl. Math, 19, Amer. Math. Soc., Providence, Rhode Island (1967) 19-32. C.

Freyd, P.

(1964) Abelian Categories, Harper and Row (1964). Mc2.

(1972) "Aspects of topoi," Bull. Austral. Math. Soc. 7 (1972) 1-76. Mc2 3/4.

Gabriel, P. and Zisman, M.

(1967) Calculus of Fractions and Homotopy Theory, Springer-Verlag (1967). Mc.

Gallaire, H.

(1969) "Decomposition of linear sequential machines II," Math. Sys. Th. 4 (1969) 168-190. SCa.

Gallaire, H. and Harrison, M.A.

(1969) "Decomposition of linear sequential machines," Math. Sys. Th. 3 (1969) 246-287. SCa.

Ginsburg, S.

(1966) The Mathematical Theory of Context-Free Languages, McGraw-Hill, New York (1966). CM1 3/4.

Ginsburg, S. and Rice, H.G.

(1962) "Two families of languages related to ALGOL," J.ACM 9 (1962) 350-371. C.

Ginzberg, A.

(1968) Algebraic Theory of Automata, Academic Press, New York. Ca 1 1/2.

Give'on, Y.

(1966) "On some categorical algebra aspects of automata theory: The categorical properties of transition systems," Ph.D. Thesis, Communication Sciences Program, University of Michigan, February (1966). CMc2 1/2.

(1967) "Categories of semimodules: The categorical structural properties of transition systems," Math. Sys. Th. 1 (1967) 67-78. Cc.

(1967a) "On some properties of the free monoids with applications to automata theory," J. Comp. Sys. Sci. 1 (1967) 137-154. Cc.

(1968) See Arbib and Give'on.

(1970) "A categorical review of algebra automata and system theories," Symposia Mathematica 4 Instituto Nazionale di Alta Matematica, Academic Press, New York (1970). CSc2 1/2.

Give'on, Y. and Arbib, M.A.

(1968) "Algebra automata II: The categorical framework for dynamic analysis," Information and Control 12 (1968) 346-370. Cc.

Give'on, Y. and Zalcstein, Y.

(1970) "Algebraic structures in linear systems theory," J. of Comp. and Sys. Sci. 4 (1970) 539-556. Sc.

Goguen, J.A.

(1967) "L-fuzzy sets," J. Math. Anal. and Appl. 18 (1967) 145-174. CSc.

(1968) "Categories of fuzzy sets," Ph.D. Thesis, Dept. of Math., Univ. of Cal. at Berkeley, May (1968). MCSc2 1/2.

Goguen, J.A. (cont'd)

(1969) "Categories of L-sets," Bull. Amer. Math. Soc. 75 (1969) 622-624. MSCc.

(1969a) "The logic of inexact concepts," Synthese 19 (1969) 325-373. SOac2 1/2.

(1970) "Mathematical representation of hierarchically organized systems," Global Systems Dynamics, (Ed. E.D. Attinger), S. Karger (1970) 111-129. SOc.

(1971) "Systems and minimal realization," Proceedings, 1971 IEEE Conference on Decision and Control, Miami Beach (1971) 42-46. CSc.

(1971a) "Discrete-time machines in closed monoidal categories, I," Institute for Computer Research, Quarterly Report No. 30, August (1971) The Univ. of Chicago, to appear in J. of Comp. and Sys. Sci. CSMc.

(1972) "Minimal realization of machines in closed categories," Bull.Amer. Math. Soc. (1972) 777-783. CSMc.

(1972a) "On homomorphisms, simulations, correctness, subroutines, and termination for programs and program schemes," Proceedings, 13th IEEE Symp. on Switching and Automata Theory (1972) 52-60. Cc.

(1972b) "Hierarchical inexact data structures in artificial intelligence problems," Proceedings, Fifth Hawaii Int'l Conference on System Sciences (1972) 343-347. COc.

(1972c) "On mathematics in computer science education," IBM T. J. Watson Res. Ctr. Technical Report RC 3899 (1972). Ccl 3/4.

(1972d) See Goguen and Yacobellis.

(1973) "Realization is universal," Math. Sys. Th. 6 (1973) 359-374. CSc2 1/2.

(1973a) "On homomorphisms, correctness, termination, unfoldments, and equivalence of flow diagram programs," to appear as UCLA Computer Science Department Technical Report (1973). Cc.

(1973b) "System theory concepts in computer science," Proceedings, Sixth Hawaii Int'l Conference on System Sciences (1973) 77-80. CSc.

Goguen, J..A. and Yacobellis, R.

(1972d) "The Myhill functor, input-reduced machines, and generalized Krohn-Rhodes theory," Proceedings, Fifth Princeton Conference on Information Sciences and Systems (1972) 574-478. CSc.

Griffiths, T.V.

(1968) "Some remarks on derivations in general rewriting systems," Information and Control 12 (1968) 27-54. Ca.

Gray, J.

(1965) "Sheaves with values in a category," Topology 3 (1965) 1-18. Mc.

Grothendieck, A. and Dieudoune, J.

(1971) Elements de Geometrie Algebrique, Springer-Verlag (1971). Mc2 1/2.

Halmos, P.R.

(1960) *Naive Set Theory*, Van Nostrand, Princeton, New Jersey (1960). Ml.

Harrison, M.A.

(1969) See Gallaire and Harrison.

Hartmanis, J. and Stearns, R.E.

(1966) *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, New Jersey (1966). Ca 1 3/4.

Hautus, M.L.J.

(1971) See Kalman and Hautus.

Heller, A.

(1967) "Probabilistic automata and stochastic transformations," *Math. Sys. Th.* *1* (1967) 197-208. MSc.

Herman, H.T. and Kotelly, J.C.

(1967) "An approach to formal psychiatry," *Perspectives in Biology and Medicine* *10* (1967). Oc.

Higgins, P.J.

(1963) "Algebras with a scheme of operators," *Math. Nachr*. *27* (1963) 115-132. Ma.

Hitchcock, P. and Park, D.M.R.

(1972) "Induction rules and proofs of termination," to appear in *Proceedings*, IRIA Symposium on Automata, Formal Languages and Programming, North-Holland, Amsterdam (1972). CM.

Hoare, C.A.R.

(1969) "An axiomatic basis for computer programming," *C.ACM*, October (1969). Cf.

Hopcroft, J.E. and Ullman, J.D.

(1969) *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, Mass. (1969). Cl 1/2.

Ianov, I.I.

(1960) "The logical schemes of automata," English translation in *Problems of Cybernetics* *1*, Pergamon Press (1960) 82-140. Cf.

Irons, E.T.

(1961) "A syntax directed compiler for ALGOL 60," *C.ACM* *4* (1961) 51-55. C.

Kalman, R.E.

(1969) "Introduction to the algebraic theory of linear dynamical systems," *Lecture Notes in Operations Research and Math. Economics* *11*, Springer-Verlag (1969). Sa.

(1972) See Roucheleau, Wyman and Kalman.

Kalman, R.E., Falb, D.L. and Arbib, M.A.

(1967) Topics in Modern System Theory, McGraw-Hill, New York (1967). SCa.

Kalman, R.E. and Hautus, M.L.J.

(1971) "Realization of continuous time linear dynamical systems: Rigorous theory in the style of Schwartz," Proceedings, Conf. on Differential Equations, NRL Math. Research Center (1971) 14-23. Sc.

Kan, D.

(1958) "Adjoint functors," Trans. Amer. Math. Soc. 87 (1958) 294-329. Mc.

Karp, R.M.

(1959) "Some applications of logical syntax to digital computer programming," Harvard University Thesis (1959). Cf.

Karp, R.M. and Miller, R.E.

(1969) "Parallel program schemata," J. Comp. and Sys. Sci. 3 (1969) 147-195. C.

Kelly, G.M.

(1966) See Eilenberg and Kelly.

Kelly, G.M. and Mac Lane, S.

(1971) "Coherence in closed categories," J. Pure and Applied Algebra 1 (1971) 97-140. Mc.

Knuth, D.E.

(1968) The Art of Computer Programming, Vol. I, Fundamental Algorithms, Addison-Wesley, Reading, Mass. (1968). Cl.

(1968a) "Semantics of context-free languages," Math. Sys. Th. 2 (1968) 127-145. Cf.

(1971) "Examples of formal semantics," Springer Lecture Notes in Mathematics 188 (Ed. E. Engeler) (1971) 212-235. Cf.

Kotelly, J.C.

(1967) See Herman and Kotelly.

Krohn, K. and Rhodes, J.

(1962) "Algebraic theory of automata," Symp. on the Math. Theory of Automata, Polytechnic Institute of Brooklyn (1962). Ca.

(1965) "Algebraic theory of machines, I: Prime decomposition for finite semigroups and machines," Trans. Amer. Math. Soc. 116 (1965) 450-464. Ca.

Lallement, G.

(1969) "On the prime decomposition theorem for finite monoids," Math. Sys. Th. 5 (1969) 8-12. Ca.

Lambek, J.

(1968) "Deductive systems and categories, I," Math. Sys. Th. 2 (1968). Mc.

Landin, P.J.

(1969) See Burstall and Landin.

(1970) "A program machine symmetric automata theory," Machine Intelligence 5 (Eds. B. Meltzer and D. Michie), Edinburgh Univ. Press (1970). Ca.

Lawvere, F.W.

(1963) "Functional semantics of algebraic theories," Proceedings, Nat'l Acad. Sci. 50 (1963) 869-872. Mc.

(1964) "An elementary theory of the category of sets," Proceedings, Nat'l Acad. Sci. 52 (1964) 1506-1510. Mc.

(1969) "Diagonal arguments and Cartesian closed categories," Category Theory, Homology Theory, and Applications II, Lecture Notes in Math. 92, Springer-Verlag, New York (1969). Mc.

(1969a) "Adjointness in foundations," Dialectica 23, (1969). MOc.

(1970) Equality in hyperdoctrines and comprehension schemas as an adjoint functor," in Applications of Categorical Algebra, Proceedings, Symp. Pure Math. XVII, American Mathematical Society (1970). Mc.

(1971) "Quantifiers and sheaves," Actes du Congres Int'l des Mathematiciens, T. 1, Nice 1970, Gauthier-Villars, Paris (1971) 329-334. Mc.

Lewis, P.M. and Stearns, R.E.

(1968) "Syntax directed transduction," J.ACM 15 (1968) 465-488. C.

Linton, F.E.J.

(1966) "Some aspects of equational categories," Proceedings, Conf. on Categorical Algebra, LaJolla, California 1965, Springer, New York (1966) 84-95. Mc.

Lipson, D.

(1970) See Birkhoff and Lipson.

Luckham, D.C., Park, D.M.R. and Paterson, M.S.

(1970) "On formalized computer programs," J. Comp. and Sys. Sci. (1970). Cf.

Mac Lane, S.

(1942) See Eilenberg and Mac Lane.

(1945) See Eilenberg and Mac Lane.

(1971) See Kelly and Mac Lane.

(1971a) Category Theory for the Working Mathematician, Springer-Verlag (1971). Mc2.

(1971b) "One universe as a foundation for category theory," Reports of the Midwest Category Seminar III (Ed. by S. Mac Lane), Lecture Notes in Mathematics, 106 Springer-Verlag (1969) 192-200. Mca.

Mac Lane, S. and Birkhoff, G.

(1967) Algebra, MacMillan, New York (1967). Macl.

Maibaum, T.S.E.

(1972) "The characterization of the derivation trees of context-free sets of terms as regular sets," Proceedings, 13th Annual Symp. on Switching and Automata Theory, College Park, Maryland (1972) 224-230. Ca.

Manes, E.G.

(1972) See Arbib and Manes.

(1972a) See Arbib and Manes.

(1973) See Arbib and Manes.

Manna, Z. and Cadiou, J.M.

(1972) "Recursive definitions of partial functions and their computations," in Proceedings, ACM Conf. on Proving Assertions about Programs, (1972) 58-65. C.

Manna, Z., Ness, S. and Vuillemin, J.

(1972) "Inductive methods for proving properties of programs," in Proceedings, ACM Conference on Proving Assertions about Programs, (1972) 27-50. C.

Manna, Z. and Pnueli, A.

(1970) "Formalization of properties of functional programs," J.ACM 17 (1970) 555-569. C.

Manna, Z. and Vuillemin, J.

(1972) "Fixpoint approach to the theory of computation," C.ACM 15 (1972) 528-536. Cl 1/2.

Marinescu, M.

(1969) See Băianu and Marinescu.

McCarthy, J.

(1963) "A basis for a mathematical theory of computation," Computer Programming and Formal Languages (Eds. P. Braffort and D. Hirschberg) North-Holland, Amsterdam (1963) 33-69. Cf.

(1963a) "Towards a mathematical science of computation," Proceedings, 1962 IFIP Congress, North-Holland, Amsterdam (1963). Cf.

McNaughton, R.

(1973) "Algebraic decision procedures for local testability," to appear in Math. Sys. Th. CMa.

Medvedev, Yu T.

(1956) "On the class of events representable in a finite automaton," Avtomaty, Moscow, Ixdatel'stvo Inostrannoi Literary (1956) 385-401. Translated in Sequential Machines: Selected Papers (Ed. E.F. Moore) Addison-Wesley, Reading, Mass. (1964). C.

Meertens, L.G.L. th.

(1972) See de Bakker and Meertens.

Mezei, J. and Wright, J.B.

(1967) "Algebraic automata and context-free sets," *Information and Control* *11* (1967) 3-29. Ca.

Miller, R.E.

(1969) See Karp and Miller.

Miller, R.E. and Thatcher, J.W.

(1972) (Eds.) *Complexity of Computer Computations*, Plenum Press (1972). C.

Milner, R.

(1972) "Implementation and applications of Scott's logic for computable functions," in *Proceedings*, ACM Conference on Proving Assertions about Programs (1972) 1-6. C.

Mitchell, B.

(1965) *Theory of Categories*, Academic Press, New York (1965). Mc2.

Morris, F.L.

(1972) "Correctness of translations of programming languages -- an algebraic approach," Stanford Artificial Intelligence Project Memo AIM-174 (1972). Cc.

Morris, J.H. Jr.

(1971) "Another recursion induction principle," *C.ACM* *14* (1971) 351-354. C.

Myhill, J.

(1957) "Finite automata and the representation of events," WADC Tech. Report (1957) Wright-Patterson AFB, Ohio 57-624. Ca.

Nerode, A.

(1958) "Linear automaton transformations," *Proceedings*, Amer. Math. Soc. *9* (1958) 541-544. CMa.

(1959) "Some stone spaces and recursion theory," *Duke Math. J.* *26* (1959) 397-406. Ma.

Ness, S.

(1972) See Manna, Ness and Vuillemin.

Nivat, M.

(1968) "Transductions des langages de Chomsky," Annales Institut Fourier (Grenoble) 18 (1968) 339-456. Ca.

Pareigis, B.

(1970) Categories and Functors, Academic Press, New York (1970). Mc2.

Park, D.M.R.

(1969) "Fixpoint induction and proofs of program properties," Machine Intelligence 5 (Eds. B. Meltzer and D. Michie) Edinburgh Univ. Press (1969) 59-78. Ca.

(1970) See Luckham, Park and Paterson.

(1972) See Hitchcock and Park.

Paterson, M.S.

(1970) See Luckham, Park and Paterson.

Peirce, C.S.

(1931) Collected Papers, Harvard University Press, Cambridge, Mass. Six Volumes (1931-1936). M.

Perles, E.

(1961) See Bar-Hillel, Perles and Shamir.

Perrot, J-F.

(1970) "On the relationship between finite automata, finite monoids and prefix codes," Proceedings 2nd ACM Symposium on Theory of Computing (1970) 217-220. CMa.

(1971) "Groups and automata," in Theory of Machines and Computations (Eds. Z. Kohavi and A. Paz) Academic Press (1971) 287-293. CMa.

Peters, S. and Ritchie, R.W.

(1967) "On the generative power of transformational grammars," unpublished manuscript (1967). OCM.

Petrone, L.

(1968) "Syntax directed mapping of context-free languages," Proceedings, 9th Ann. Symp. on Switching and Automata Theory, October (1968). C.

Pfender, M.

(1972) See Ehrig and Pfender.

Pierce, R.S.

(1970) "Classification problems," Math. Sys. Th. 4 (1970) 65-80. MSc.

Pnueli, A.

(1970) See Manna and Pnueli.

Post, E.

(1943) "Formal reduction of the general combinatorial decision problem," Amer. J. Math. 65 (1943) 197-215. M.

(1947) "Recursive unsolvability of a problem of Thue," J. Symbolic Logic 12 (1947) 1-11. M.

Prather, R.E.

(1970) "On categories of infinite automata," Math. Sys. Th. 4 (1970) 295-305. Cc.

Rabin, M.O.

(1967) "Mathematical theory of automata," Proceedings, Symp. on Applied Mathematics 19 (1967) 153-175. C2.

Rhodes, J.

(1962) See Krohn and Rhodes.

(1965) See Krohn and Rhodes.

Rice, H.G.

(1962) See Ginsburg and Rice.

Ritchie, R.W.

(1967) See Peters and Ritchie.

Rose, G.F.

(1964) "An extension of ALGOL-like languages," C.ACM 7 (1964) 52-61; Also, SDC TM-738/003/00 May (1963). C.

Rosenbloom, P.

(1950) Elements of Mathematical Logic, Dover Publications (1950). M1.

Rota, G.C.

(1969) "Baxter algebras and combinatorial identities I,II," Bull. Amer. Math. Soc. 75 (1969) 325-334. Mc.

Roucheleau, Y.

(1972) "Finite-dimensional, constant, discrete-time linear dynamical systems over some classes of commutative rings," Ph.D Dissertation, Operations Research Dept., Stanford University (1972). Sa2 1/2.

Roucheleau, Y., Wyman, B.F., and Kalman, R.E.

(1972) "Algebraic structure of linear dynamical systems, III. Realization theory over a commutative ring," Proceedings, Nat'l Acad. of Sciences 69 (1972) 3404-3406. Sa.

Rounds, W.C.

(1968) "Trees, transducers, and transformations," Ph.D. Dissertation, Stanford University, August (1968). Cf2 1/2.

(1969) "Context-free grammars on trees," Proceedings, ACM Symp. on Theory of Computation, Marina del Rey (1969) 143. Ca.

Rutledge, J.D.

(1964) "On Ianov's program schemata," J.ACM 11 (1964) 1-9. Cf.

Schnorr, C.D.

(1969) "Transformational classes of grammars," Information and Control 14 (1969) 252-277. Cc.

Schoenfield, J.R.

(1967) Mathematical Logic, Addison-Wesley, Reading, Mass. (1967). Ml.

Schützenberger, M.

(1961) "On the definition of a family of automata," Information and Control 4 (1961) 245-270. CMd.

(1963) See Chomsky and Schützenberger.

(1965) "On finite monoids having only trivial subgroups," Information and Control 8 (1965) 190-194. MCSa.

(1969) See Eilenberg and Schützenberger.

Scott, D.

(1959) See Rabin and Scott.

(1970) "Outline of a mathematical theory of computation," Proceedings, 4th Princeton Conf. on Inform. Science and Systems (1970). CMa.

(1971) "The lattice of flow diagrams," Symposium on Semantics of Algorithmic Languages, (Ed. E. Engeler), Lecture Notes in Math. 188 Springer-Verlag, Berlin (1971). Ca.

(1972) "Lattice theory, data types and semantics," Formal Semantics of Programming Languages, (Ed. R. Rustin), Prentice-Hall, New Jersey (1972) 65-106. Ca.

(1972a) "Continuous lattices," Proceedings, 1971 Dalhoisie Conf., Springer Lecture Note Series, 274, Springer-Verlag (1972) 97-136. Cac.

(1973) "Lattice-theoretic models for the λ-calculus," to appear. Ca.

Scott, D. and de Bakker, J.W.

(1969) "A theory of programs," unpublished notes, IBM Seminar, Vienna (1969). CMfa2.

Scott, D. and Strachey, C.

(1971) "Toward a mathematical semantics for computer languages," Proceedings, Symp. on Computers and Automata Microwave Res. Inst. 21, Polytechnic Institute of Brooklyn (1971). Ca.

Semandeni, Z.

(1972) "On logical educational materials, categories, and automata," Queen's Univ. at Kingston, Ontario, Canada, Dept. of Math., Reprint No. 1972-38 (1972). CMOc.

Shamir, E.

(1961) See Bar-Hillel, Perles and Shamir.

(1968) "Algebraic, rational and context-free power series in non-commuting variables," Algebraic Theory of Machines, Languages, and Semigroups (Ed. M.A. Arbib), Academic Press, New York (1968) 329-341. Ca.

Shepard, C.D.

(1969) "Languages in general algebras," Proceedings, ACM Symp. Theory of Computing (1969) 155-163. Ca.

Sipusic, T.L.

(1972) "Two automata decomposition theorems generalized to machines in a closed monoidal category," Institute for Computer Research, Univ. of Chicago, Quarterly Report No. 35, November (1972). Cc.

Stearns, R.E.

(1966). See Hartmanis and Stearns.

(1968) See Lewis and Stearns.

Strachey, C.

(1971) See Scott and Strachey.

Strong, H.R.

(1968) "Algebraically generalized recursive function theory," IBM J. Res. Devel. 12 (1968) 465-475. CMa.

Tarski, A.

(1955) "A lattice theoretical fixpoint theorem and its applications," Pacific J. Math. 5 (1955) 285-390. Mf.

Thatcher, J.W.

(1967) "Characterizing derivation trees of context-free grammars through a generalization of finite automata theory," J. Comp. and Sys. Sci. 1 (1967) 317-322. Ca.

(1969) "Transformations and translations from the point of view of generalized finite automata theory," Proceedings, ACM Symp. Th. Comp., Marina del Rey (1969) 129-142. Ca.

(1970) "Generalized sequential machine maps," J. Comp. and Sys. Sci. 4 (1970). Ca.

(1972) See Thatcher and Miller.

(1973) "Tree automata - an informal survey," Currents in Computing, (Ed. A.V. Aho), Prentice-Hall, New Jersey (1973). Cal.

Thatcher, J.W. and Wright, J.B.

(1968) "Generalized finite automata with an application to a decision problem of second-order logic," Math. Sys. Th. 2 (1968) 57-81. Ca.

Thue, A.

(1914) "Probleme über vanderungen vin zeichenreihen nach gegebenen Regeln," Skrifter utgit av Videnskapsselskapet i Kristiania, I. Matematisk naturvidenskabelig klasse, No. 10, (1914) 34. Mc.

Ullman, J.D.

(1968) See Aho and Ullman.

(1969) See Hopcroft and Ullman.

Vuillemin, J.

(1972) See Manna, Ness and Vuillemin.

(1972a) See Manna and Vuillemin.

Wagner, E.

(1969) "Uniformly reflexive structures: On the nature of Godelizations and relative computability," Trans. Amer. Math. Soc. 144 (1969) 1-41. Ca.

(1971) "Languages for defining sets in arbitrary algebras," Proceedings, 12th IEEE Symp. on Switching and Automata Th., East Lansing, Mich. (1971). Ca.

(1971a) "An algebraic theory of recursive definitions and recursive languages," Proceedings, 3rd Annual ACM Symposium on Theory of Computing, Shaker Heights, Ohio (1971). Ca.

(1973) "From algebras to programming languages," Proceedings, 5th ACM Symp. on Theory of Computing, Austin, Texas (1973). Cac.

Wegner, P.

(1972) "A view of computer science education," Amer. Math. Monthly, February (1972) 168-172. OCl.

Wand, M.

(1971) "Theories, pretheories, and finite state transformations on trees," MIT, Artificial Intell. Lab. Memo. No. 216 (1971). Ca.

(1972) "A concrete approach to abstract recursive definitions," MIT, Artificial Intell. Lab. Memo. No. 262 (1972). Cac.

(1972a) "Closure under program schemes and reflective subcategories," MIT, Artificial Intell. Lab. Memo. (1972). Cc.

(1972b) "Uninterpreted schemes are natural transformations," MIT, Artificial Intell. Lab. (1972). Cc.

(1972c) "Models of higher type," MIT, Artificial Intell. Lab. (1972). Cac.

Weber, H.

(1966) See Wirth and Weber.

Windeknecht, T.G.

(1967) "Mathematical systems theory: Casuality," Math. Sys. Th. 1 (1967). Sa.

Wirth, N. and Weber, H.

(1966) "EULER: A generalization of ALGOL and its formal definitions: Part I," C.ACM 9 (1966) 13-23. C.

Wright, J.W.

(1960) See Büchi and Wright.

(1962) See Burks and Wright.

(1965) See Thatcher and Wright.

(1967) See Eilenberg and Wright.

(1967a) See Mezei and Wright.

(1968) See Thatcher and Wright.

Wyman, D.F.

(1972) See Roucheleau, Wyman and Kalman.

Yacobellis, R.

(1973) "Generalized Krohn-Rhodes theory," Ph.D. Dissertation, Univ. of Chicago, Committee on Information Sciences (1973). Cc.

(1972) See Goguen and Yacobellis.

Zadeh, L. and Desoer, C.A.

(1963) Linear Systems Theory, McGraw-Hill, New York (1963). Sfl.

Zalcstein, Y.

(1970) See Give'on and Zalcstein.

(1972) "Locally testable languages," J. Comp. and Sys. Sci. 6 (1972) 151-167. CMa.

(1973) "Syntactic semigroups of some classes of star-free languages," Automata, Languages and Programming (Ed. M. Nivat), North-Holland (1973). CMa.

(1973) "On the semigroups of linear sequential machines," J. Comp. and Sys. Sci. 2 (1973) 25-28. SCa.

Zeiger, H.P.

(1967) "Ho's algorithm, commutative diagrams, and the uniqueness of minimal linear systems," Information and Control 11 (1967) 71-79. SCac.

(1969) See Arbib and Zeiger.

Zeigler, B.

(1971) "A note on series-parallel irreducibility," Math. Sys. Th. 5 1-3. Ca.

Zisman, M.

(1967) See Gabriel and Zisman.

INDEX OF DEFINITIONS, CONCEPTS AND EXAMPLES

- **Ab** 50
- acceptors Ex. 2.11; 52
- adjointness 62
- **Alg** 47, 49, 50, 62
- algebras 31
  - carrier of ___ 31
  - category of ___ Ex. 2.7; 46
  - homomorphism of ___ 32
  - partial ___ 32
  - subalgebra of ___ 32
  - underlying set of ___ 31, 61
- algebraic structure 30, 46
- arity of operations 30, 33
- **Aut** 48, 52, 62
- $\mathbf{Aut}^X$ 52
- automata Ex. 2.2; 43
  - category of state transitions of ___ 43
  - M- ___ Ex. 2.18; 59
  - monoid of ___ Ex. 2.26; 62
  - nondeterministic ___ 44
- **Beh** 48, 52, 57
- behaviors 48, 56
  - category of Ex. 2.9; 48
  - construction of Ex. 2.16; 56

Boolean algebra Ex. 1.11; 37

cardinality 23

$\underline{\underline{\mathrm{Cat}}}$ 60, 65

$\underline{\underline{\mathrm{Cat}}}_S$ 60

category 11, 39, 40

composition in a ___ 40

discrete ___ 45

identity function of a ___ 40

locally small ___ 42

monoid as a ___ Ex. 2.5; 45, Ex. 2.17; 58, Ex. 2.20; 60

morphisms of a ___ 11, 39

objects of a ___ 11, 39

opposite ___ 63

product ___ 68

small ___ 42

__of acceptors Ex. 2.11; 52

__of algebras Ex. 2.7; 46

__of automata Ex. 2.8; 48

__of behaviors Ex. 2.9; 48

__of categories Ex. 2.8; 47, Ex. 2.19; 60

__of finite machines Ex. 2.10; 51

__of languages Ex. 2.13; 53

__of machines Ex. 2.8; 48

__of monoids Ex. 2.6; 46, Ex. 2.20; 60

__of partial functions Ex. 2.12; 52

__of relations Ex. 2.12; 52

__of sets Ex. 2.1; 41

__of small categories Ex. 2.20; 60

__of state transitions Ex. 2.2; 43

__of X,Y-machines Ex. 2.11; 52

closure 30

commuting diagram 23

concatenation 34

contravarient functor 65

coproduct 28

covariant functor 66

diagonal 20

diagram 23

dual 63

embedding 56

empty string 27

endofunctor 54

equivalence 22

logical ___ 16

__classes 29

__of categories 61

equivalent

__sets 22

Example

1.1 semigroups 34

1.2 monoids 34

1.3 free monoid 34

1.4 groups 35

1.5 Abelian groups 36

1.6 rings 36

1.7 lattices 36

1.8 distributive lattices 37

1.9 zero of a lattice 37

1.10 complemented lattice 37

1.11 Boolean algebra 37

1.12 lower and upper bounds 37

2.1 category of sets 41

2.2 category of transitions 42

2.3 ordinals 44

2.4 quosets 44

2.5 monoids 45

2.6 category of monoids 46

2.7 category of algebras 46

2.8 category of machines 47

2.9 category of behaviors 48

2.10 category of finite sets 51

2.11 category of X-machines 52

2.12 category of relations and partial functions 52

2.13 category of languages 53

2.14 inclusion functor 55

2.15 power set functor 56

2.16 behavior functor 56

2.17 monoid as a category functor 58

2.18 M-automata 58

2.19 category of categories 60

2.20 monoids as categories 60

2.21 quosets as categories 61

2.22 carrier of monoid functor 61

2.23 reduction of algebras functor 62

2.24 underlying set functor 61

2.25 free monoids 62

2.26 monoid of an automaton 62

2.27 nonnegative integers and register transfers 64

2.28 ^{op} as a functor 65

2.29 power set functor 65

2.30 hom functor 66, 68, 69

2.31 Cartesian product functor 68, 69

Fin 51

Fmach 51

forgetful functor 61

free

___ monoids Ex. 2.25; 62

full

- __subcategory 50
- __functor 55

functions 19

- bijective ___ 22
- characteristic ___ 53
- injective ___ 22
- inverse of ___ 26
- partial ___ 21
- preimage of ___ 21
- projection ___ 64
- set of definition of ___ 21
- surjective ___ 22

functor 12, 54

- adjoint ___ 13
- arrow part of ___ 54
- bijective ___ 54
- composition of ___ 59
- contravariant ___ 65
- covariant ___ 66
- faithful ___ 55
- forgetful ___ 61
- full ___ 55
- identity ___ 59
- inclusion ___ 55

hom ___ 66

injective ___ 54

object part of ___ 54

powerset ___ Ex. 2.15; 56, Ex. 2.29; 66

strict ___ 55

surjective ___ 54

underlying set ___ Ex. 2.24; 62

graph

transition ___ 43

__of a relation 19

groups Ex. 1.4; 35

Abelian ___ Ex. 1.5; 36, 50

Grp 46, 50

hom functor 66

homomorphism

machine ___ 47

__of algebras 32, 33

__of categories 54

image

__of a relation 22

__subcategory 55

induction 26

isomorphic

__sets 22

isomorphism 22, 55

__of categories 55, 61

natural ___ 66

- languages 49
  - category of ___ Ex. 2.13; 53
- lattices Ex. 1.7; 36
  - complemented ___ Ex. 1.10; 37
  - complete ___ Ex. 1.12; 37
  - distributive ___ Ex. 1.8; 37
  - zero in ___ Ex. 1.9; 37
- length of a string 27
- **Lng** 53,57
- logic 16
- lower bound 37
  - greatest ___ 37
- **M** 45, 58, 60
- M-automata Ex. 2.18; 58
- **Mach** 48, 50, 51, 56, 57, 62
- **Mach**$^{X}_{Y}$ 52
- machines Ex. 2.8; 47, Ex. 2.10; 51
  - behavior of ___ Ex. 2.9; 48
  - category of ___ Ex. 2.8; 47
  - finite ___ 50, Ex. 2.10; 51
- many sorted algebras 33
- **Mon** 46, 50, 60, 61, 62
- **Mon**$_{C}$ 60

monoids Ex. 1.2; 34

category of ___ Ex. 2.5; 45, Ex. 2.20; 60

free ___ Ex. 1.3; 34, Ex. 2.25; 62

___as a category Ex. 2.6; 46, Ex. 2.17; 58, Ex. 2.20; 60

___of an automaton Ex. 2.26; 62

$\underline{\underline{n}}$ 44,50

$\underline{\underline{N}}$ 51, 64

natural

___equivalence 13

___isomorphism 13

___transformation 13

operations 30

absorptive ___ 36

arity of ___ 30

associative ___ 34

commutative ___ 36

distributive ___ 36

idempotent ___ 36

many sorted ___ 33

monadic ___ 30

sort of ___ 33

opposite

___category 63

$\underline{\underline{Or}}_R$ 45, 61

ordinal notation 19

partial functions 21

category of ___ Ex. 2.1; 52

partial ordering 29, Ex. 2.4; 44

partition 29

Pfn 52

Pierce product 23

poset 30

product

__of categories 67

__of sets 18

quasi-ordering 30, Ex. 2.14; 44

Quo 61

quoset 30

ranked operator set 31

Rel 52

relations 19

antisymmetric ___ 29

birestriction of ___ 22

category of ___ Ex. 2.12; 52

composition of ___ 23

converse of ___ 23

corestriction of ___ 22

equivalence ___ 29

graph of ___ 19

identity ___ 20

image of ___ 22

inclusion ___ 20

inverse image of ___ 23

Pierce product of ___ 23

reflexive ___ 29

restriction of ___ 22

symmetric ___ 29

rings Ex. 1.6; 36, 46

Rng 46

Sem 46

Sem$_*$ 46, 50

semigroups Ex. 1.1; 34, 46

pointed ___ 46, 50

sequences 27

Set 41, 46, 51, 53, 56, 58, 61, 62, 64, 65, 66, 68, 69

Set$_*$ 53

sets 17

category of ___ Ex. 2.1; 42

families of ___ 28

finite ___ Ex. 2.10; 51

pointed ___ 53

recognizable ___ 52

source

__of functors 54

__of morphisms 39

__of relations 19

strict

__functor 55

__subcategory 50

strings 27

subcategory 50

image ___ 55

target

__of functors 54

__of morphisms 39

__of relations 19

$\underline{\underline{\mathrm{Tr}}}_A$ 43

transitions

__of an automaton 43

underlying set functor Ex. 2.24; 62

universal property 35

upper bound 37

least ___ 38

INDEX OF SYMBOLS AND NOTATION WITH FIRST OCCURRENCE

Logical Symbols

∨	and (16)
⊥	false (16)
∀	for all (16)
===>	implication (16)
<===>	logical equivalence (16)
¬	not (16)
∧	or (16)
∃	there exists (16)
∃!	there exists a unique (16)
⊤	true (16)

Sets

#	cardinality (23)
•	composition (23)
∨	converse (23)
∐	coproduct (28)
↾	corestriction (22)
−	difference (18)
∅	empty set (18)
λ	empty function (26)
$\underset{\sim}{P}_F$	finite subsets (18)
1_S	identity on S (20)
∩	intersection (18)
⋂	intersection (28)
-1	inverse (26)
∈	membership (17)
∉	nonmembership (17)
ω	nonnegative integers (19)
$\underset{\sim}{P}$	powerset (18)
X	product (18)
∏	product (28)
/	quotient (29)
*	reflexive transitive closure (30)
↿	restriction (22)
⊆	subset (18)
+	transitive closure (30)
∪	union (18)
⋃	union (28)

Miscellaneous

+	addition (14
=	equals (16)
/	division (16)
$\bigwedge$	greatest lower bound (38)
$\bigvee$	least upper bound (38)
≤	order (37)
≥	order (38)
∑	sum (16)
____	boldface (5)
- - -	italic (5)
∿ ∿ ∿	script (5)
=====	special (5)
□	end of proof (16)

Arrow notations

$\rightarrow$	(19)
$\mapsto$	(20)
$\rightarrowtail$	(22)
$\twoheadrightarrow$	(22)
$\hookrightarrow$	(22)
$\cong$	(22)

Categories - general

$\circ$	composition (40)
1_A	identity (41)
$\lvert\underline{\underline{C}}\rvert$	objects (39)
^{op}	opposite (63)
$\underline{\underline{C}}$	morphisms (39)
∂_0	source (39)
∂_1	target (39)