TEODOR RUS

DATA STRUCTURES AND OPERATING SYSTEMS

EDITURA ACADEMIEI
Bucureşti

JOHN WILEY & SONS
Chichester • New York • Brisbane • Toronto
A Wiley-Interscience Publication

1979

Library of Congress Cataloging in Publication Data
Rus, Teodor.
Data structures and operating systems.

Updated translation of Structuri de date şi sisteme operative.
"A Wiley-Interscience publication."
Bibliography: p.
1. Operating systems (Computers) 2. Data structures (Computer science) I. Title.

QA76.6.R8613 001.6′4 77-3262
ISBN 0-471-99517-7

This edition is the revised English version of the Romanian book "Structuri de date şi sisteme operative" published in 1974 by **EDITURA ACADEMIEI, Calea Victoriei 125, BUCUREŞTI, 71021**
All rights reserved.

PRINTED IN ROMANIA

DATA STRUCTURES AND OPERATING SYSTEMS

Preface

Recent developments in computer technology have been so rapid that we can often notice a difference between the physical and functional states of real computer systems, on the one hand, and the theory of their design and implementation, on the other. This rapid development is due to the role assigned to computers in the advancement of science and technology. The recent explosion of information can no longer be mastered by man using traditional methods. Nowadays it has become impossible to make correct decisions in any field of social activity without using computer systems. Also, it must be said that the ever increasing degree of complexity produced by progress in science and technology is due, to a large degree, to informational structures. Obviously, one must question the role of computing systems for the future processing of information. Will these techniques be sufficient to handle information or will some other means be devised?

The development of computer systems has produced two results in an attempt to tackle the problems raised by the present development of science and technology. First, the number of computing systems is ever increasing and these have become the basic instruments for solving problems that require mere calculation or mechanical devices for information processing. Second, computing systems have become more and more specialized in order to solve specific types of problem.

The above mentioned facts are closely connected with the present attempts to expand computer structures and to get closer and closer to a kind of simulation of human thought at the highest level of formal logic. In this respect, two basic tendencies are apparent.

(i) The replacement of simple linear structures by non-linear memory structures and generalized addresses, which leads to modified and more complex addressing mechanisms (see the Multics system). In addition, the computer system is entirely virtualized at the level of each of its components.

(ii) A higher degree of virtualization in computer structures, which leads to architectural structures that simulate human formal logic, and the shift to physical real structures by means of wired devices that simplify programming work and make it more profitable.

These two main tendencies in computer science are based on the discovery of several new physical states that can perform the functions of reception, conservation and processing of information. Also, new basic theories for implementing technologies of physical structures and their corresponding software have greatly contributed to this orientation of computers. The theoretical foundation of computer systems has become the main tendency of late; this is due to the lack of balance between practice in this field and the theoretical foundation. Physical equipment is required by prac-

tical necessities. Philosophy and theoretical foundations are necessary not so much for the purpose of constructing a physical computer but mainly for establishing principles that enhance the development and profitability of computers in all respects. However, since computers must first exist and can only afterwards become profitable, the above mentioned lack of balance is not without explanation. At present, quantitative accumulations have become sufficiently large to enable a qualitative leap to be carried out, induced by the foundation of theoretical principles.

Since mathematics has greatly contributed to computers, it is almost natural that a theory of computers should be defined within the scope of mathematics. Mathematics is still a powerful means of reference for computers. However, mathematics was not developed to accomplish the goals of computers, it was devised to formalize human thought. Mathematics serves this purpose on the logical-abstract level whilst computers have been devised to serve this purpose on the logical-concrete level. Hence, computers can formalize thought and also provide a means for the mechanical construction of thought. This is why the objects of computer science are somewhat different from the objects of mathematics, in the same way as their methods and all the consequences thereby induced differ. However, the above mentioned similarities have naturally led to a mutual influence between the two fields.

Mathematicians who have devoted their studies to computer science have often reached highly abstract theoretical results where the physical computer vanishes behind mathematical abstractions. The mathematical theories produced by these studies are only *mathematical theories, although their development has been urged and sometimes induced by real computers. In this respect, it is sufficient to think of the automata theory and algorithms and recursive function theory. These theories can account for some phenomena of real computers but they cannot replace the theory of real computers. Hence, the correlation between these two fields of activity is not only natural but also necessary. This is useful not only for the increase of knowledge for its own sake but also for the efficient development of both mathematics and computers and human thought. In this respect, we shall specify only the consequences of this correlation as an efficient means of computer development. With computers, efficiency means observance of their purpose and higher profitability. More precisely, computers must take good account of the methods of logical construction of thought from mathematics and other sciences, on both the formal and the constructive levels of physical modelling. In addition, the work of computers is production, in the true sense of the word, and production must be profitable in all respects, from initiation and design to implementation and problem solution. Hence, mathematics and other sciences must be able to explain computer facts and especially computer construction and planning.*

Hence, it is necessary to work out a theory for real computers that provides a theoretical foundation for both hardware and software engineering. The engineering used in the production of software systems is a natural requirement considering the fact that computers represent a means of production. We can hardly claim that the engineering for software systems is complete but we are certain that this is one of the fields of computer investigation that has been much invested in, on good account, at this time.

Therefore, the theory of real computers must be related to two aspects of computer activity: (1) the real aspect of the physical hardware whereby operations are *performed within a given logic in the form of programs, and (2) the formal aspect of this logic. The given logic can itself be considered as a virtual computer. The actual fulfilment of the functions of a virtual computer on a real computer, or the implementation of a virtual computer on a real computer, represents one of the key-problems of such a theory.*

Technology is used for designing and implementing a virtual computer on a real computer and for the planning and management of jobs on real computers. Although these problems are clear by now, a theoretical foundation of the principles and methodology for such technological activity is still lacking.

The present volume marks a beginning in this respect. The structure of the book follows the aims given below.

(i) To produce a technique capable of further development. So far, the experience of researchers has proved that such a technique can be obtained by information organization, which can be better done by means of the methods of modern algebra where heterogeneous structures are extremely important, and it appears that this technique is very suitable for this purpose.

(ii) To apply the formal technique of information organization to describe the structure of real computers and the construction of a formal model for the architecture of a computer system that includes both existing structures and future developments.

(iii) To organize the information flow in a computer system by means of the proposed technique.

Considering these stages the material has been divided into eight chapters.

Chapter I presents the mathematical techniques in a graded form, starting from the simplest algebraic structures that are used by computers up to the most complicated ones. The purpose of this chapter is to provide a simple introduction for those who are not well acquainted with abstract algebra.

Chapter II deals with heterogeneous algebraic structures, developed as a mathematical technique for modelling computer systems. The purpose of this chapter is to provide a framework for theories faithful to a given reality and to provide the heterogeneous algebraic structure used as a mathematical basis for the theories of hardware and software.

Chapter III presents a formal model for actual computing systems by means of the technique proposed in Chapter II. In addition, these techniques are used for the formulation of computing structure models that can assist the future development of architectural considerations. In this respect, we have used the system of level hierarchies with computing systems. This principle makes clear the relationship both between real and virtual computer, and the implementation of virtual computers on real computers. A number of examples illustrate both the functions of various components of the computing system and their implementation on hierarchy levels. The flow of information in a real computing system is extremely important for these principles and relations. Hence, computers are conceived on two levels, virtual and real, and then the functional model of the real computer is analysed.

Chapter IV deals with the structure of internal information $I(X_I, \Omega_I)$ associated with the virtual computer. This structure is thought to be similar to the structure associated with the real computer, which is denoted by SRC.

Chapter V presents the structure of the external information $E(X_E, \Omega_E)$. *Thus, the process of problem solving by a computer system is viewed as a sequence of information transformations in a job diagram of the form*

$$E(X_E, \Omega_E) \rightleftarrows I(X_I, \Omega_I) \rightleftarrows SCR \circlearrowleft$$

Chapter VI deals with the operating system as a function for the distributions of jobs in a real computer system. The discussion of the structure and functions of the operating system proceeds from the basic concept of job, which is defined by means of the job diagram presented in Chapter V. Several types of operating system and the methods used for their design and implementation are also discussed.

Chapter VII presents the Multics system as one of the first computing systems whose structure can be formalized on a theoretical basis.

Chapter VIII is devoted to the reliability of the computer system. For the study of reliability we must always bear in mind the human limits *in system design and implementation, whence the need for protection strategies.*

This book is intended for a wide variety of readers: research workers in computer science, professors and students in the computer science field, and also mathematicians, who can contribute so much to this science by their research work.

The author believes that these ideas are sufficient to study the technology of software systems. From this standpoint this book makes a modest beginning, although further developments will, of course, be forthcoming.

I wish to express my profound gratitude to Professors Günter HOLTZ, F. L. BAUER, Charles RATTRAY, and P. LANDIN for their valuable ideas, which have substantially contributed to the present volume.

Contents

Chapter I

Abstract structures in computing systems

In this chapter we shall briefly present some of the abstract mathematical structures which are frequently used in the theory of computing systems and which will be also used in the following chapters. To justify the use of such mathematical elements in computing systems, we start with the simple observation that, in practice, computers usually work with signs. These signs have two elements: on the one hand, a physical element which depends on their nature and, on the other hand, an element of information arising from the fact that the signs are symbols which differ from each other.

From the physical point of view, the signs represent physical states of some material quantities such as electric potentials, electric currents, magnetic properties, and so on. By subtracting the physical nature of these signs we are left only with simple symbols which differ from each other and which can be used to build other symbols, and so on. However, these signs, as well as the symbols built on them, can be interpreted as primitive operations of a computer; *i.e.* primitive data to be processed by the computer. Primitive operations or primitive data functions of arbitrary nature are symptomatic of the interpretations given to various types of such elements. However, to be able to speak of such different interpretations, the corresponding signs and constructions performed on them should be structuralized. Structuralization will provide syntactic distinctions which will turn into differences of interpretation. The corresponding structuralization can obviously be performed within the framework of a theory of abstract structures like the usual ones in algebra. This means that algebra will play the role of a structuralization apparatus for signs and constructions performed with these signs, each representing the abstract parts of some physical phenomena, like those mentioned above.

In consideration of what has been said above, we shall briefly discuss in the first two chapters some of the concepts of the usual abstract structures familiar in the theory of computers, like the concepts of semigroup, universal algebra, as well as some concepts which have only recently entered the domain of computers, like heterogeneous universal algebra with state words. Transformations between the structures thus defined will be discussed by means of the concepts of functions, isomorphism, homomorphism, and so on. Since the fundamental elements of the constructions will be sets of signs, we proceed by specifying the language we use, namely that of set theory.

§ 1.1. Concept and Notation in Set Theory

We shall not define the concept of set, since it is intuitively clear. We state only that every time we speak of a set, we have in view one of the following two ways of specifying it.

1. By enumerating the elements of the set.
2. By means of some properties satisfied by all the elements of the set.

The sets will be denoted by Latin capitals, and the first way of specifying a set is

$$A = \{elem\ 1,\ elem\ 2, \ldots\}$$

where *elem 1*, *elem 2*, ... means the sequence of the elements of the set. The second way to specify a set is

$$A = \{a \mid P(a)\}$$

which means that the set A consists of all the elements a which satisfy the property P.

To denote that an element belongs to a set, we use the symbol $\in$, and the expression "a is an element of the set A" will be written as $a \in A$.

A set is void and is denoted by the sign $\emptyset$ if it contains no elements. To avoid confusion between an element and a set containing a single element, the latter will be denoted within brackets: $\{\ \}$. Thus, $\{x\}$ and x are logically distinct. The first is a set and the second is an element of this set.

The following operations are defined for sets.

1. *Union*: if A and B are two given sets, then their union is the set $A \cup B = \{a \mid a \in A \text{ or } a \in B\}$.

2. *Intersection*: the set $A \cap B = \{a \mid a \in A \text{ and } a \in B\}$ is called the intersection of the set A with the set B.

3. *Difference*: the set $A \setminus B = \{a \mid a \in A \text{ and } a \notin B\}$ is called the difference between the sets A and B.

The sign $\notin$ used above is the negation of the sign $\in$, *i.e.*, $a \notin B$ expresses the property "a is not an element of the set B".

Let A and B be given sets. If every element of A belongs to B, we say that A is included in B or that A is a sub-set of B, or that B includes A, and we denote this by $A \subset B$ or $B \supset A$. By means of the implication sign $\Rightarrow$ and of the double implication $\Leftrightarrow$, we have

$$A \subset B \Leftrightarrow (a \in A \Rightarrow a \in B).$$

The sign $\Rightarrow$ is used to denote that the right-hand side follows from the left-hand side, and the sign $\Leftrightarrow$ is used to denote that the left-hand side follows from the right-hand side, and conversely that the right-hand side follows from the left-hand side. Thus, we have $A \subset B \Leftrightarrow B \supset A$.

The previously defined operations with sets satisfy the following properties.

1. Commutativity: $A \cup B = B \cup A$ and $A \cap B = B \cap A$.
2. Associativity: $A \cup (B \cup C) = (A \cup B) \cup C$ and
$$A \cap (B \cap C) = (A \cap B) \cap C.$$
3. Distributivity: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ and
$$A \cup (B \cap C) = (A \cup B) \cap (A \cup C).$$
4. Morgan formulas: $X \setminus (A \cup B) = (X \setminus A) \cap (X \setminus B)$,
$$X \setminus (A \cap B) = (X \setminus A) \cup (X \setminus B).$$

If M is a set, then we denote the set of the sub-sets of M by PM. (PM is sometimes also denoted by 2^M). By means of the notation already defined we have $PM = \{A \mid A \subset M\}$ and it can be seen immediately that PM is closed with respect to the operations $\cup$, $\cap$ and $\setminus$.

§ 1.2. Concept of Application

For two given sets, A and B, the set $A \times B = \{(a, b) \mid a \in A, b \in B\}$ is called the cartesian product of the sets A and B.

If $R \subset A \times B$, then the triplet $f = (A, B, R)$ is a binary relation or a correspondence of A in B.

If $(a, b) \in R$ and $f = (A, B, R)$ is a binary relation, then we say that a is related to b by the relation R and this is often denoted by aRb.

If the sets A and B coincide, that is, if $R \subset A \times A$, then the relation R can have the following properties.

1. Reflexivity: the relation $R \subset A \times A$ is called reflexive if $(a, a) \in R$ for any $a \in A$, or if aRa for any $a \in A$.

2. Symmetry: the relation $R \subset A \times A$ is called symmetric if from $(a, b) \in R$ it follows that $(b, a) \in R$, or $aRb \Rightarrow bRa$.

3. Transitivity: the relation $R \subset A \times A$ is said to be transitive if from $(a, b) \in R$ and $(b, c) \in R$ it follows that $(a, c) \in R$, or aRb and $bRc \Rightarrow aRc$.

An equivalence relation in a set A signifies a relation $R \subset A \times A$ with the properties of reflexivity, symmetry and transitivity. An equivalence relation is usually denoted by the symbol $\sim$.

Let therefore $\sim$ be a given equivalence relation in the set A. Two elements $a, b \in A$ are said to be equivalent if $a \sim b$. For each element $a \in A$, we denote by $C(a)$ the sub-set of A which consists of all the elements equivalent to a; i.e. $C(a) = \{b \in A \mid a \sim b\}$. Since the relation is reflexive, it follows that $a \in C(a)$.

PROPOSITION I.1. *For any two elements $a, b \in A$ with the property $a \notin C(b)$ and $b \notin C(a)$, the equality $C(a) \cap C(b) = \emptyset$ holds.*

Proof. Suppose $C(a) \cap C(b) \neq \emptyset$. Then there exists an element $c \in A$ such that $c \in C(a) \cap C(b)$. Therefore, $a \sim c$ and $c \sim b$. From the transitivity of the equivalence relation it follows that $a \sim b$; *i.e.* $C(a) = C(b)$.

From Proposition I.1 it follows that the equivalence relation divides the set A into disjoint sub-sets which will be denoted by $\{C(a) \mid a \in A\}$. The elements of the set are called equivalence classes. $C(a)$ is the equivalence class of the element $a \in A$ with respect to the equivalence relation.

Let us define now the notion of partition of a set A: by partition of a set A we mean a family $\mathscr{P}$ of disjoint sub-sets of A such that the union of the elements of $\mathscr{P}$ is the set A itself.

From the definition of the partition of a set it follows that the family Q of equivalence classes defined by an equivalence relation $\sim$ in the set A is a partition of the set A. The family Q is called the quotient set of A and is denoted by $Q = A/\sim$.

A relation $f = (A, B, R)$ is called a function (or application) if the following two conditions hold.

(i) For every $a \in A$ there exists a $b \in B$ such that $(a, b) \in R$.

(ii) If $(a, b) \in R$ and $(a, b') \in R$, then $b = b'$.

When the relation $f = (A, B, R)$ is a function, we use the notation $f: A \rightarrow B$, and for the pair $(a, b) \in R$ we have $f(a) = b$. To denote a point-like correspondence of the two sets in a function, we sometimes use the notation $a \overset{f}{\mapsto} b$. The set A is called the domain of the function f, and B is called the co-domain of f. For each sub-set $X \subset A$, the sub-set $Y \subset B$ of all the elements $f(x) \in B$ for $x \in X$ is called the image of X in B of the function f and is denoted by $f(X)$. In other words,

$$f(X) = \{f(x) \in Y \mid x \in X\}.$$

The image of the set A which represents the domain of f is denoted by $\mathrm{Im}(f)$ and called the image of f.

It can easily be shown that the following relations hold for the operations $\cup$ and $\cap$ and a function f:

$$f(A \cup B) = f(A) \cup f(B)$$

$$f(A \cap B) \subset f(A) \cap f(B)$$

If $f(A) = B$, then we say that the function $f: A \rightarrow B$ is a function from A into B. This means that for each $b \in B$ there is an $a \in A$ such that $f(a) = b$.

If $f(A)$ consists of only one element $b \in B$, then we say that $f: A \rightarrow B$ is a constant function from A to B. If A is a non-empty set, then for each $b \in B$ there exists only one constant function $f_b: A \rightarrow B$ such that for each $X \subset A$, $f_b(X) = b$.

For each sub-set Y of B, the sub-set X of A, which consists of all the elements of A, $x \in A$, such that $f(x) \in Y$, is called the inverse image of Y in the function $f: A \rightarrow B$ and is denoted by $f^{-1}(Y)$. In other words, $f^{-1}(Y) = \{x \in A \mid f(x) \in Y\}$. If Y has only one element, $Y = \{y\}$, then $f^{-1}(Y)$ is called the inverse image of the element y in the function f and is denoted $f^{-1}(y)$. The following proposition then holds.

PROPOSITION I.2. *An element $y \in B$ belongs to the image $f(A)$ of the function f if and only if $f^{-1}(y)$ is non-empty.*

For two sub-sets of the co-domain B of the function f, X and Y, the following relations are satisfied for the operations $\cup$, $\cap$ and $\setminus$ and the function $f: A \to B$:

$$f^{-1}(X \cup Y) = f^{-1}(X) \cup f^{-1}(Y)$$

$$f^{-1}(X \cap Y) = f^{-1}(X) \cap f^{-1}(Y)$$

$$f^{-1}(X \setminus Y) = f^{-1}(X) \setminus f^{-1}(Y)$$

From these equalities it can be seen that the inverse image behaves better with respect to the operations performed on the sub-sets of the co-domain than does the image with respect to the operations performed on the sub-sets of the domain. Thus, if X and Y are two disjoint sub-sets of B, then their inverse images in f, $f^{-1}(X)$ and $f^{-1}(Y)$, are disjoint sub-sets of the domain A of the function. In particular, the inverse image of distinct elements in B gives distinct elements in A.

A function $f: A \to B$ is called injective if and only if the inverse image $f^{-1}(b)$ of every element $b \in B$ is empty or has only one element. Thus, it follows that f is injective if and only if the image of distinct elements in A are distinct elements in B: symbolically, this can be written as: $f(a) = f(b) \Rightarrow a = b$.

An example of an injective function is inclusion. If $A \subset B$, then $i: A \to B$ defined by $i(a) = a \in B$ for $a \in A$ is called the inclusion function of A into B. This is sometimes denoted by $i: A \subset B$.

A function $f: A \to B$ is called surjective if for every $b \in B$, $f^{-1}(b)$ is non-empty; i.e. for every $b \in B$ there exists an $a \in A$ such that $f(a) = b$.

A function $f: A \to B$ is called bijective if it is surjective and injective. Two properties follow by this definition.

(i) Since f is injective, for every $b \in B$, $f^{-1}(b)$ has only one element.

(ii) Since f is surjective, for every $b \in B$, $f^{-1}(b)$ has at least one element.

(i) and *(ii)* show that $f^{-1}(b)$ is an element of A; i.e. $b \mapsto f^{-1}(b)$. In other words, $f^{-1} = (B, A, R^{-1})$ where $R^{-1} = \{(b, a) \mid (a, b) \in R\}$ is also a function, called the inverse function of function f.

The function f^{-1} is bijective. This follows immediately from the definition of the function f and from the construction of the inverse function f^{-1} for the given function f.

An example of an inverse function can be obtained from the inclusion function $i: A \subset B$. This is called the identical function of the set A, and for every $a \in A$ we have $i^{-1}(a) = i(a)$. For this reason, the function is denoted by 1_A.

Let $f: A \to B$ and $g: B \to C$ be two given functions. The operation of composition of the functions f and g is defined by the following conventions: for every $a \in A$ we denote by $h(a)$ the element $g(f(a))$ of the set C. This means that $h: A \to C$ defined by $h(a) = g(f(a))$ is a new function called the composite function of f and g. If h is denoted by $g \circ f$ then we have $h = g \circ f: A \to C$.