Bestell-Nr:	SUBITO-2010041302724	**Staats- und Universitätsbibliothek Bremen**
PPN:	40815	
Bestelldatum:	2010-04-13 16:51:03	

SUBITO-Bestellung

NORMAL

Warsaw University of Technology
Main Library - ILL

Politechniki 1
PL-00-661 Warsaw

Tel: +48 22 2347401
Mail: wypmb@bg.pw.edu.pl
Fax:

Ben.-Gruppe: USER-GROUP-8

Kontaktperson:
Ms Izabela Fornal
wypmb@bg.pw.edu.pl

Benutzer-Ausweisnummer
SLI07X00293E

Lieferbibliothek: <46>
Staats- und Universitätsbibliothek Bremen
Bibliothekstraße

28359 Bremen

Telefon: +49(0)421/218-2644, -2645
Fax: +49(0)421/218-2040
E-Mail: fernleihe@suub.uni-bremen.de

Leihfrist: Bücher 6 Wochen

Reklamation erbitten wir innerhalb von 10 Tagen

Lieferschein/ delivery note

Rechnung folgt - Bitte veranlassen Sie erst dann eine Zahlung, wenn die Rechnung bei Ihnen eingetroffen ist.
Bills are mailed every three months or according to arrangements.

Datum / date ________

_____ Kopien / copies

Unter Anerkennung der Benutzungsbedingungen wird bestellt:

Verfasser: W. Marek, Z.Pawlak
(Aufsatz)

Standort:
fc 5684

Titel: Computers and Programs
(Aufsatz)

Seiten: 17-19

Band/Heft: 13 **Jahr** 1978

Titel (Monographie/ Zeitschrift)

Computer education
Bowker & Dally
Stafford, Engl.

0010-4590

Lieferform: KOPIE **Lieferart:** EMAIL

Lieferung erwünscht bis:
2010-04-16 16:52:08

SUBITO-2010041302724

Bemerkung 1: 111/10
Bemerkung 2:

COMPUTERS AND PROGRAMS

by Zdzislaw Pawlak, Polish Academy of Sciences

This article considers the concept of the computer and its program in a novel manner, using a common die as a teaching aid!

The digital computer has had a revolutionary impact on many branches of science, technology and administration. At the same time the design and application of computers have given rise to interesting investigations of a purely logical or mathematical nature. In this paper we look at one branch of these investigations into the notions of a computer and of a computer program.

Digital computers are, of course, very complicated devices. None the less the main problems concerning computers can easily be illustrated by means of a simple model.

Memory. The principal part of every computer is its memory. This memory can be thought as a device capable of being at any given moment in one of several possible states. As a very simple example of a computer memory we consider a cube. Each wall of the cube is marked with a different natural number, say one of the numbers 1, 2, 3, 4, 5, 6. If the cube stands on the wall marked 4 then we say that the cube is in the state 4 (similarly for the walls marked 1, 2, 3, 5, 6). If we turn the cube over so that it stands on a different wall then we say that the cube has changed its state.

On each wall of a cube we distinguish arbitrarily four directions N, W, S, E, as illustrated in fig. 1.

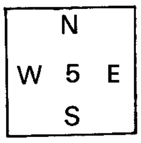

Fig. 1

Each such cube with directions marked on its numbered walls can be represented by a diagram as in figure 2. Walls of the cube are represented by small circles and the numbers assigned to the walls are written into the corresponding circles. If two walls have an edge in common we connect the corresponding circles by a line. Next to each circle and by the appropriate line we write one of the directions N, W, S, E, as shown in fig. 2. Such a

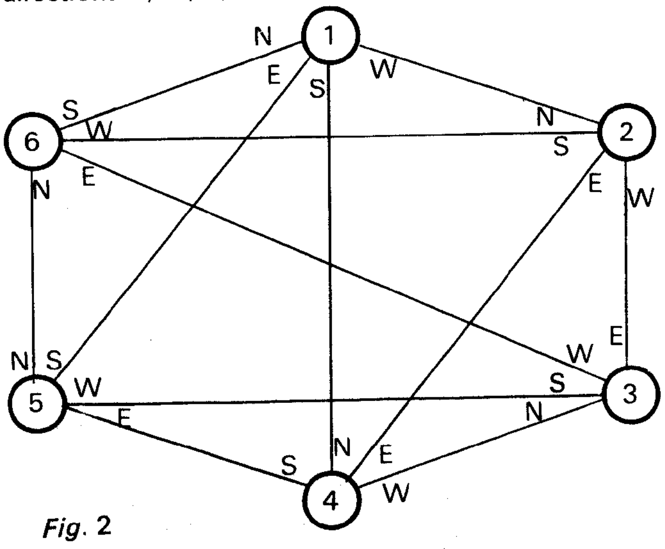

Fig. 2

diagram is easier to use than the ordinary representation of the cube in two dimensions and so we shall use this representation in what follows.

Instructions. A number of instructions are associated with each computer. These instructions are operations which change the memory from one state to another. For our example we suppose that we have four instructions denoted by the letters N, W, S, E. Instructions α (α = N, W, S, E) means that we have to turn the cube over from the wall it stands on onto the wall pointed to by the direction α. If for example the cube is in state 5 and we apply the instruction S then the resulting state will be 1. We can represent all of the instructions in a table

x	N(x)	W(x)	E(x)	S(x)
1	6	2	5	4
2	1	3	4	6
3	4	6	2	5
4	1	3	2	5
5	6	3	4	1
6	5	2	3	1

Table 1

where N(x), W(x), E(x), S(x) denote the result of applying the corresponding instruction to the state x. Instructions therefore are functions taking states of memory (in our case the numbers 1, 2, 3, 4, 5, 6) as arguments and as values.

Syntactic definition of programs. A program is a finite string of symbols. For the computer under consideration we define the notion of program as follows:

1. The letters N, W, S, E are programs,
2. If A is a program then the expression ?, A is also a program,
3. If A and B are programs then so is A, B.
4. The only programs are the expressions which can be obtained by the rules 1, 2, 3.

The expression ? is called the conditional instruction. The following strings of symbols are programs:
N, E, E, ?, N, ?, S, N
E, S, ?, N, E
N, N, E
E, S, ?, E, E
but the strings as below are not programs
?, ?, E, S, ?
N, ?, S, ?.

The set of all programs is called a programming language. A memory and a programming language together uniquely define a computer. Thus to define a computer is to give its memory and its programming language. More precisely a computer C is a function C:L x M>M where L is a programming language and M is a computer memory.

The semantics (meaning) of programs. With every program we can associate a function whose arguments and values are memory states. This function is called the meaning of the program. The program is an expression representing the function. Let P be a program and let P(x) be the function associated with P. In order to define the meaning of the program (i.e. the function P(x)) we have to show how to obtain the value of P(x) when the program P and the state x are given.

In order to obtain the value P(x) we first apply to the state x the right-most instruction of the program $P = \alpha_0, \alpha_1 \ldots, \alpha_k$. According to the definition of a program this instruction α_k is one of N, W, S, E (not?). Applying α_k to x we obtain a new state x! If we have just applied the instruction α_i ($i<k$) of the program P and the resulting memory state is y and α_{i-1} is an unconditional instruction then we apply to y the instruction α_{i-1}; if α_{i-1} is a conditional instruction (i.e. ?) then we apply to y either the instruction α_{i-2} or α_k. The instruction α_k is applied to y if y is an odd number and α_{i-2} is applied if y is even.

This procedure is terminated if and when we have reached and applied the left-most instruction α_0.

Each unconditional instruction of the program causes a change of state in the memory of the computer (turning over the cube). The conditional instruction denoted by ? investigates whether the present state of the memory satisfies some condition (in our case whether the cube is in an odd or even state). According to the result of this test either the first or the next instruction of the program is applied to the present state of memory. The sequence of states $x_0, x_1, \ldots, x_k, \ldots$ caused by applying the program P to the state x, is called a computation of the program P. If the computation is finite then the last state of the computation is the value of $P(x_0)$. If the computation starting from x_0 is infinite then the value of the $P(x_0)$ is undefined.

Example 1. Consider the program P = E, S, ?, N, E and the memory state 2. According to the definition of the function P associated with the program P, we first apply to the state 2 the instruction E, obtaining E(2) = 4; then to the state 4 we apply the instruction N, which gives the state N(4) = 1. Because the next instruction in the program is ? we have to check whether the state of memory is now odd or even. Because it is odd we apply the first instruction of the program again and obtain E(1) = 5. We apply the next instruction and obtain N(5) = 6. Because the resulting state is even we apply the instruction S which yields S(6) = 1 and then the last instruction E (1) = 6. Because the program contains no additional instructions we have P(2) = 6. This computation may be represented in the form

2
4 = E(2)
1 = N(4)
5 = E(1)
6 = N(5)
1 = S(6)
5 = E(1).

Proceeding similarly for the arguments 1, 3, 4, 5, 6 we obtain the following table representing the function P.

x	1	2	3	4	5	6
P(x)	5	6	6	5	5	4

The function P is the meaning of the program P.

Example 2. Let us consider the program Q = ?, S, E. The computation of this program for the state 1 is as follows:

1
5 = E(1)
1 = S(5)
5 = E(1)
1 = S(5)
. . . . . . etc.

The computation is infinite because after state 1 we always obtain state 5 and after state 5, state 1. Thus the function Q is undefined for the argument 1. Similarly Q is undefined for the arguments 2, 5 and 6. For the arguments 3 and 4 the value of the function Q is 6. The following is therefore the table for the function Q:

x	1	2	3	4	5	6
Q(x)	–	–	6	6	–	–

Problems. We are now in a position to state some of the problems involving computers and their programs.

Halting problem. Given a computer and a program we can ask if the program will terminate for a given initial memory state. If the total number of memory states is finite then this problem is of course solvable. In this case to solve the problem we need only compute the value of the corresponding function for a finite number of cases. This method, however, is not adequate if the set of all memory states is infinite. To solve the problem for this case we need a method which involves only examining the structure of the program and avoids computing the final state of the memory. (This sort of method is also desirable in the finite case when the number of states is large and the computation of the final state is time consuming). This problem has been widely investigated and there are many interesting results concerning different notions of programs and computers. These results are closely related to certain problems in logic.

Equivalence of programs. In our example of a computer the set of memory states is finite. There are therefore only a finite number of functions whose arguments and values range over the set of memory states. There are, however, infinitely many programs. So some programs must have the same meaning, i.e. compute the same functions. Such programs are said to be equivalent. The problem arises of determining whether two programs P and Q are equivalent, i.e. whether P(x) = Q(x) for all x.

Again we are not interested in solving this problem by actually computing all the possible values of the functions P and Q (even when the set of memory states is finite). We are interested instead in solving the problem by a method which investigates the structure of the expressions P and Q. Alternately we may attempt to find a finite or infinite set of axioms and a set of inference rules such that two programs are equivalent if and only if they can be shown to be so by derivation from the given set of axioms by means of the inference rules. Not many results have been obtained in these areas. Solutions have been found only for some simple cases of programs.

Equivalence of programming languages. The computer in our example has five instructions N, W, S, E, ?. It is easy to see that we can remove the instruction E, for example, and replace it by a program causing the same changes of states as the instruction E. From this it follows that both programming languages, one with and the other without the instruction E, define the same class of functions (meanings of programs). We say that two programming languages are equivalent if they define the same class of functions. Given a computer memory and two sets of instructions are the corresponding programming languages equivalent ? A general method for answering this question has not been found. We can also ask for a minimal set of instructions equivalent to a given set of instructions, or equivalently for a minimal set of instructions defining a given programming language. This is yet another unsolved problem.

Semantics of programs. In the examples 1 and 2 we have associated with each program its meaning by computing for all possible initial memory states the value of the corresponding function (when the value was defined). This method is not adequate for practical purposes. We are interested instead in assigning meaning to programs in the following way: to each instruction we assign a function and then extend the assignment to the whole program, consisting of instructions, obtaining the function assigned to the program. If we have some program we are interested in knowing what function the program is computing. But at present there are no methods known for dealing with this problem, and for example we are unable to prove in general that a given program does or not compute sin x.

Complexity of programs. If two programs are equivalent we might be interested in determining which of them is in some sense the simpler of the two. For example as a measure of the simplicity of a program for a given argument we can take the length of the computation for the argument. By the length of computation for a program P and state x we mean the number of changes of memory states caused by the program P beginning with the initial state x and ending with the final state P (x). The program P is simpler than the program Q if and only if for all states x the length of the computation of P(x) is less than the length of the computation Q(x). Here too investigations are only in their initial stage and not many results have been obtained.

Simulation and equivalence of computers. We say that computer C′ simulates computer C if for each computation $x_0, x_1, \ldots, x_k$ in C there exists a computation $x_0', x_1', \ldots, x_1'$ in C′ such that for all i $(0<i<k)$ there exists j $(0<j<m)$ such that $x_i = x_j'$ and $x_1 = x_1'$ and $x_k = x_m$ and if for j_1 and i_2, x_{j_1} corresponds to x_{i_2} then $j_1 \geqslant i_2$. This means that in order to simulate computer C by C′ we have to simulate by a computation in C′ the whole of each computation in C step by step in such a way that each state of memory in the computer to be simulated is represented in the corresponding computation of the computer which simulates it. When such a relation holds between computers each action in the first computer C can also be performed in the second computer C′ If the computer C simulates the computer C′ and vice versa we say that the computer C and C′ are equivalent.

The topics of simulation and equivalence of computers have not been investigated very intensively even though they are of great practical importance.

Polish Academy of Sciences
Computation Centre
Warsaw 1.XI.1971 r.

COURSES

The Middlesex Polytechnic
COMPUTERS IN EDUCATION
The advent of the computer is having, and will continue to have, a profound influence on our way of life. The Middlesex Polytechnic has been involved for over six years with providing practical assistance to teachers who wish to introduce computing into their school curriculum, and the short full-time 'Computers in Education' courses have arisen as a natural consequence. This is their fifth year of operation.

'An introduction to Computer Education in School' is intended for teachers wishing to introduce computing into the middle school curriculum. It covers all aspects of computing, including elementary programming and the elements of data processing. The remaining courses cover various aspects of computing in more detail, enabling the teacher to acquire enough familiarity with the subject to teach it in the upper school or Technical College. Despite this specialisation, each course, except Further Fortran, is designed to be suitable for those with no previous knowledge of the subject, or even any specialist mathematical background.

Fee £6·00, including course material (subject to confirmation). Five days full-time, from 9.30 – 4.30 p.m.

MC1C	Basic Fortran, or
MC2	Practical approach to Logic. 25th to 29th June 1973.
MC3	An Introduction to Computer Education in School. 25th to 29th June 1973.
MC4	Further Fortran or
MC5	Electronic Data Processing, or
MC6	City & Guilds Code. 2nd to 6th July 1973.

For further information, write to:
N. Bowker,
Course Organiser,
Computers in Education,
Middlesex Polytechnic,
Queensway, Enfield, Middlesex.