Missing p's

For CHAP 12, see second report, deals with code generation for automatic operations

UNCLASSIFIED

Naval Ordnance Laboratory
White Oak, Silver Spring 19, Maryland

NAVAL ORDNANCE LABORATORY MEMORANDUM 9805

26 January 1949

20 July 1948 (Submitted)

See my 2 memos to DD

From: Haskell B. Curry
To: NOL Files
Via: Chief, Mechanics Division

Subj: On the Composition of Programs for Automatic Computing. (Project NOL 118b, NR-044-049)

Abstract: A study has been made of the planning of computations with a view to shortening the process and to systematizing it so that more work can be done by less technically trained personnel or by machines. In this report, the problem is attacked from the standpoint of composition of computing schedules. For example, if a plan is recorded for each of a group of component processes (a program) by means of a system of symbolization (a code), it is required to form a program for the composite computation. This problem is studied theoretically by use of techniques similar to those used in some phases of mathematical logic.

Foreword: The data and conclusions presented here are for the use of the personnel of the Naval Ordnance Laboratory. They may not represent the final judgment of the Laboratory.

Ref.: (a) Alt, Franz. "A Bell Telephone Laboratories Computing Machine." Mathematical Tables and Other Aids to Computation 3; 1-13, 69-84 (1948).

Sponsored by the Office of Naval Research.

Hand Copy

(b) Burks, Arthur W., Goldstine, Herman H., and von Neumann, J. Preliminary Discussion of the Logical Design of an Electronic Computing Instrument. Part I, Vol. I. Institute for Advanced Study (1946).

(c) Curry, Haskell B. "The Combinatory Foundations of Mathematical Logic". Journal of Symbolic Logic 7, 49-64 (1942).

(d) Curry, H. B. and Lotkin, Max. A Study of Fourth Order Interpolation on the ENIAC. Aberdeen Proving Ground, BRL Report No. 613 (1946).

(e) Curry, H. B. and Wyatt, Willa. A Study of Inverse Interpolation on the ENIAC. Aberdeen Proving Ground, Report No. 615 (1946).

(f) Feys, Robert. "La Technique de la logique combinatoire". Revue Philosophique de Louvain 44, 74-103, 237-270 (1946).

(g) Goldstine, Herman H. and von Neumann, John. Planning and Coding of Problems for an Electronic Computing Instrument. (Part II, Vol. I of Report on the Mathematical and Logical Aspects of an Electronic Computing Instrument.) Institute for Advanced Study. (1947).

(h) Goldstine, Herman H. and von Neumann, John. Planning and Coding of Problems for an Electronic Computing Instrument, Part II, Vol. II. Institute for Advanced Study (1948).

(i) Lotkin, Max. Inversion on the ENIAC Using Osculatory Interpolation. Aberdeen Proving Ground, BRL Report No. 632 (1947).

(j) Goldstine, Herman H. and von Neumann, John Planning and Coding of Problems for an Electronic Computing Instrument, Part II, Vol. III. Institute for Advanced Study (1948).

Encl: (A) Figure 1. Flow Chart for Sample Program.

TABLE OF CONTENTS

p.s. Dr. Missky

computer* (now at Aberdeen Proving Ground) realized the need of "subroutines" and made some provision for dealing with them. Doubtless the same is the case with other machines. The problem of program composition was a major consideration in a study of inverse interpolation on the ENIAC made at Aberdeen Proving Ground**; for, although that study was made under stress, and was directed primarily towards finding at least one practical method of programming a specific problem, yet an effort was made to construct the program by piecing together subprograms in such a way that modifications could be introduced by changing these subprograms. More recently Goldstine and von Neumann have announced*** that they are working on the problem and plan to publish their results in a few months.**** The study of Goldstine and von Neumann just mentioned promises to be a more extensive study than it will be possible to make in this report. Nevertheless, the problem is important enough to justify an independent attack on it.

---

* For an account of this machine see reference (a). The statement in the text is based on information disclosed in a conference held at the Bell Laboratories, July 13, 1945. It was written up in various internal memoranda of the Laboratories before that time.

** See reference (e); cf. also reference (d). One of the modifications mentioned referred to in reference (e), p. 54 §11.2 was later worked out by Lotkin, reference (i).

*** See the prefaces of references (g) and (h).

**** These results are contained in reference (j), which appeared while the present report was in process. (This report was first submitted July 26, 1948.)

5. The present attack, although it owes a great deal to Goldstine and von Neumann, goes back for its fundamental philosophy to the Aberdeen report above mentioned. In fact, it was evolved with reference to inverse interpolation. That problem, which was first suggested as a subject for automatic programming by T. E. Sterne, has shown itself to be well suited for the purpose. It is simple enough so that it is scarcely economical for a big machine; yet it has a structure showing several different kinds of composition. Nevertheless, in view of the shortage of time it seems best to confine attention to the general principles in this report, and to leave the applications to inverse interpolation to a later one.

6. The fact that the illustrations are postponed to this later paper may make the present theory seem somewhat abstruse. However, it is expected that a practical technique will be developed at that time on the basis of the present theory. This technique will differ from that of reference (g) in some respects, although it will agree with it in others. The present theory develops, in fact, a notation for program construction which is more compact than the "flow charts" of reference (g). Flow charts will be used, as in reference (e), primarily as an expository device.* By means of this notation a composite program can be exhibited as a function of its components in such a way that the actual formation of the composite program can be carried out by a suitable machine.

7. Up to the present, nothing has been said about the machine. It might be inferred, from the fact that a modification of a technique for the Princeton machine is derived from a program for the ENIAC, a very different sort of machine, that there are general principles common to all machines. There doubtless are. But in this report it is not feasible to be quite so abstract. It is expedient rather to pick out a single machine which

---

* The flow chart in reference (e), Fig. 44, was added to the report after the work was all done in order to make it easier for the reader. This flow chart is different in principle from those in reference (g); it is more like the flow charts used in ordinary business management.

is more or less typical, to make the study with reference to it, to include such generalizations as seem immediate, and to leave the question of ultimate generalization until later. For this purpose, the Institute for Advanced Study machine has been chosen; largely because, through the courtesy of those in charge of the project, reports concerning it are readily available. However, in order to build up a theory it is necessary to idealize the machine. Accordingly in Section II, below, we consider an idealization of the Princeton machine. The modifications in the theory in order to adapt it to the actual machine will be considered in a later report.

II Fundamental Definitions and Assumptions

8. The following statements are partly definitions and partly assumptions about the machine.

A. Words, Locations, and Programs

9. The machine handles as units certain aggregates of digits called words. These are of two main types, as follows:

a. Quantities, i.e., numbers being calculated.*

b. Orders, i.e., coded instructions govering the operation of the machine.

Words will be denoted by the capital letters G, H, I, J, K, L, M, N with or without subscripts and other diacritical marks. The letter J will indicate a blank word, i.e., one composed exclusively of zeros.

10. The machine has a memory consisting of a certain sequence of locations in each of which a word can be stored. These locations will be distinguished by assigning to each a serial number called its location number. It will be supposed that these locations are infinite in number and that the location numbers are 0, 1, 2, ...

---

* These quantities may include integers necessary to keep track of iterations, etc. They are not necessarily thought of as in the unit interval.

11. An assignment of $n+1$ words to the first $n+1$ locations will be called a program. The number n, which is arbitrary, will be called the length of the program. The letters X, Y, Z, U, V, W will designate programs.

12. Notations for Programs. An equation

$$X = M_0 M_1 \cdots M_n$$

shall mean that X is that program of length n for which the kth word (i.e., the word in the location numbered k) is M_k. Here M_0 will play a perculiar role. It will be called the initating order. Sometimes it will be omitted (this situation will be disclosed by the fact that the subscript of the first word written is not 0); in that case a blank word J is to be supplied. Thus,

$$X = M_1 M_2 \cdots M_n$$

shall mean the same as

$$X = J M_1 M_2 \cdots M_n$$

Finally given two programs

$$X = M_0 M_1 \cdots M_n$$

$$Y = N_0 N_1 \cdots N_n$$

we define their concatenation, XY, thus

$$XY = M_0 M_1 \cdots M_m N_1 N_2 \cdots N_n$$

Thus the kth word in XY is M_k if $k \leq m$, N_{k-m} if $k > m$. Note that N_0 is omitted.

13. A program consisting exclusively of orders will be called an order program; one which consists, except for an initial order J, entirely of quantities will be called a quantity program. Order programs will be designated by the letters A, B; quantity programs by the letters C, D, E, F.

14. The distinction between quantities and orders is

not a distinction of form. Given a word written out in the code, it is impossible to tell from its appearance whether it is of type a or type b. In fact the same word may be interpreted at one time as an order, at another time as a quantity. The machine makes this distinction according to the situation. Making this classification of words in advance is a difficult problem which will concern us later.

15. An order is assumed to have the following structure. It contains two location numbers, which are the numbers for the datum location and exit location, respectively, and a part coding the character of the operation to be performed. The latter will be called the operator. The machine is able to recognize these three parts of an order. The words located at the datum and exit* locations will be called the datum and the exit, respectively; while datum and exit location numbers will be referred to as datum number and exit number, respectively.

B. Operation of the Machine

16. We have been considering the memory as an isolated static phenomenon. We now consider the other organs of the machine and its manner of operation.

17. Besides the memory the machine contains an arithmetic unit (Par. 18) and a control (Par. 19).

18. The arithmetic unit (a.u.) is, of course, the organ which makes the calculations. We assume that it contains some auxiliary storage locations and that words can be transferred back and forth between a.u. and the memory. We further assume one such location capable of adding, called the accumulator. In the actual machine there are two locations in the a.u., the accumulator and a "register," each with some special functions. However, the detailed structure of a.u. is irrelevant for our present purpose.

---

* The present theory would have to be but little modified if we admitted the possibility of several data.

19. The control is, for our present point of view, simply a device for keeping track of the location in the memory from which the machine receives its instructions. This location will be called the control location. If the control location is numbered n, we shall say the control is at n.

20. The operation of the machine is as follows. The calculation starts with the control at 0 and the arithmetic unit clear. If the control is at n, the machine interprets the word at location n as an order; it executes the operation indicated by the operator, using the datum and the words in the a.u. as operands; on completion of this operation the control shifts to the exit location, unless the operator orders another control shift* or causes the machine to stop. Each such cycle of operation, from one shifting of control to the next (or from the start to the first control shift) will be called a step.

21. The state of the machine at the beginning or end of a step will be called a configuration. A configuration is completely specified by giving: (a) the program in the memory at that time, (b) the contents of a.u., and (c) the control location. Configurations will be denoted by the letters P, Q, R. The configuration at the end of s steps in a calculation will be called Q_s; its program, control location, and accumulator contents will be called X_s, c_s, a_s, respectively. (For s = o these give the initial configuration, program, etc.)

C. Classification of Orders

22. We consider here the assumptions in regard to the orders more in detail. It is not necessary to require that the orders be as specific as the list given in reference (g), but rather that the orders can be

---

* The possibility is expressly allowed that the operation may order that the control shift to the datum location, rather than to the exit location.

classified into species* according to the scheme given below. Under each species, examples are given from the code of reference (g), Table II. It is assumed that orders ordering different operations can be distinguished in form, viz., by the form of the operator; so that, although there is no distinction in form between a quantity and an order, there is such a distinction between the different species of orders.

23. The scheme of classification is as follows:

A. Arithmetical orders, which order the a.u. to perform some calculation, using the datum and the contents of the a.u. as operands. The result of the calculation remains in the a.u. The program is unchanged. Examples: 2, 3, 4, 5, 6, 7, 8, 11, 12, 20, 21.

B. Transfer orders, which move a wordunchanged from one position to another, erasing the previous content of the new position, but leaving the old position and everything else unchanged. There are three main kinds:

1. Incoming transfers from the memory to the a.u. In these cases the datum shows the location from which the transfer is made. Examples: 1, 9.

2. Internal transfers within the a.u. Here the datum is irrelevant. Example: 10.

---

* The term species hereafter will mean a category of words, viz., one of the smallest categories in the classification obtained by combining the schemes of Par. 9 and 23.

3. Outgoing transfers from the a.u. to the memory. Here the datum is erased and replaced by the word transferred. Example: 17.

c. Control orders, which change location numbers. Control shifts belong here, because these can be regarded as making changes in the exit numbers.

1. Unconditional control shifts. Here the effect of the order is to shift control to the new datum location without making any other change. (In the idealized machine such orders are superfluous; but they are necessary in the actual machine in order to initiate the same action from two different sources.*) Examples: 13, 14.

2. Discrimination orders, which order the a.u. to make a discrimination and then to make a control shift depending on the outcome of the discrimination. There are thus, in effect, two exits; one of these is indicated in the normal fashion by the exit number, the other is indicated by the datum number. Examples: 15, 16.

---

* The presence of such orders allows substitutions to be made in exit numbers while confining C, 3 to datum numbers.

3. Location substitutions, which substitute a location number in the a.u. for one in the datum without affecting anything else. Examples: 18, 19.

D. Stop orders, which cause the machine to stop. It will be supposed here that an order with a blank operator (i.e., one for which all digits are 0) is a stop order. This has the advantage that if a location number is stored as an order with blank operator and the control comes to it in error, the machine will stop. We suppose that a stop order stops the machine so that the configuration remains intact. This is important in case the stoppage is due to an error.

D. A Theorem on Type Determination

24. According to Par. 14 it is not possible to distinguish orders from quantities by their form alone. On the other hand, the theory here developed requires that all the words occurring at all steps of the calculation be assigned a unique type.

25. An assignment of type to each word of a calculation will be called a type determination* when it satisfies the following conditions:

a. Initial condition. The initial order of the initial program is an order; the words initially in the a.u. are quantities.**

---

* The term will also be used for parts of a calculation in an obvious sense.

** Any properly planned calculation will be so arranged that the initial contents of the a.u. will have no effect on the results of the calculation. Thus, it is not actually necessary to clear the a.u. before starting. But the theoretical discussion is a little simpler if we make the assumption mentioned in the text.

b. Exit condition. The exit of an order is always an order.

c. The invariance condition: If a word is not affected in any step, then its type is unchanged also.

d. The transfer condition: When a word is transferred by a transfer order, the word in the new location has the same type as in the old.

e. The arithmetic condition: An arithmetic order never occurs unless the words in the a.u. are quantities. The result then has the same type as the datum; all other words left in a.u. are quantities. In the first case the order concerned will be called a pure arithmetic order; in the second, a mixed arithmetic order. The mixed arithmetic order will be subjected to further conditions below.

f. The type condition on control orders: The datum of a control order which is receiving control is always an order, and so also is the result of a substitution. (In the case of a substitution note that it is a location number which is substituted, and this must come from an order in the a.u.)

g. The discrimination condition: Every discrimination is based on detecting the sign of a quantity.

26. Theorem. Suppose we have a given type-determination for the initial program X_0. If there exists a type-determination for the whole calculation which satisfies the conditions of par. 25 and assigns the given types to X_0, then this determination is unique.

This determination satisfies the condition that the word at the control location is always an order.

27. To prove this we show by induction on s that the type of every word in Q_s is uniquely determined, and that the word in the control location of Q_s is an order. This is true for $s = o$ by the hypothesis and the initial condition. Suppose it true for Q_s; we prove it true for Q_{s+1}. The assertion concerning the control location follows at once from the conditions (b) and (f).* As for the uniqueness, let the order at the control location of Q_s be N_s, then we consider various cases as follows:

a. If N_s is an arithmetic order, the result of the calculation has the same type as the datum; all other words left in a.u. are quantities, while the program is unchanged. Thus the type of every word is uniquely determined.

b. If N_s is a transfer order, nothing is changed except that the transferred word is put in the new location in the place of what was there before. Thus the type of every word is again uniquely determined.

c. If N_s is a control order, then the conditions of par. 25, parts (c) and (f), show that the type of every word is again uniquely determined. In fact, a control order does not change any types.

d. If N_s is a stop order, the machine stops, leaving the configuration unchanged in toto (par. 23, D). We can say that the types are still uniquely determined by the invariance condition.

---

* The control shifts to the exit location and condition (b) applies in all cases except orders of species C_2 and D; in case C either condition (b) or (f) applies while in case D the machine stops.

28. It will be noted that we did not prove the existence of a type determination. It is easy to construct programs such that for the ensuing calculation no type determination exists. (One has only to have the freak situation mentioned as a possibility in Par. 14.) A calculation for which type determination exists will be called typically determinate, and a program which initiates such a calculation is a typically determinate program.

E. Special Kinds of Programs

29. Since an entire calculation is uniquely determined by its initial program, it is proper to characterize programs by properties of the calculations they initiate. In fact, we can apply certain adjectives to either programs or calculations, as we have already done in Par. 28. Some further classes of programs will now be considered.

30. A location such that the type of the word in it is the same throughout the calculation will be said to be fixed as to type. A similar definition will apply for being fixed as to subtype, species, etc. A program will be said to be fixed as to type, subtype, species, etc. if all of its locations are so fixed. It is difficult to imagine a practical program which is not fixed as to type and species; but it is conceivable that such should exist.

31. A primary program is a typically determinate program whose calculations contain no mixed arithmetic order; a secondary program, one in which at least one mixed arithmetic order occurs. In a primary program new location numbers cannot be formed by arithmetic processes. The only location numbers which occur are those already in the initial program.

32. Secondary programs are necessary when one has function tables. Suppose that the values of $f(x_1, \ldots, x_\mu)$ are stored at locations $c + \phi(x_1, \ldots, x_\mu)$, respectively, where $x_1, \ldots, x_\mu$ and $\phi(x_1, \ldots, x_\mu)$ are quantities which, with possible shifting of scale, can be calculated by the machine. In order to "look up" $f(x_1, \ldots, x_\mu)$, it is necessary to calculate the location

number e, $\phi(x_1, \ldots, x_\mu)$. Evidently it is sufficient to calculate $\phi(x_1, \ldots, x_\mu)$ and then add e to it by a mixed arithmetic order.

33. The discussion in Par. 32 leads to the formulation of the following condition, called the table condition, on a program. To formulate this condition we suppose that there are certain groups g_1, g_2, ... of locations with fixed types, the locations in any one group being all of the same type. The table condition then consists of two parts, viz. a restriction on mixed arithmetic orders and a restriction on location substitutions. The former restriction is that a mixed arithmetic operation can occur only when (a) the datum is an order whose datum location is in some g_i, (b) the operation is addition, and (c) the result is an order whose datum location is in the same g_i. The second restriction is that a substitution is only permissible when the original and substituted location numbers are numbers of locations in the same g_i.

34. A regular program is a typically determinate program which is primary or else satisfies the table condition. This category embraces a wide variety of practical programs. Further restrictions on a regular program will be made in Par. 43.

F. Datum and Exit Locations

35. We define an exit location of a program X as one which receives control at some step of the calculation based on a program X^1 got by changing the quantities in X.* Further we say a datum location of a program X is a location such that there exists an order N of species A,

---

* In any particular calculation there is a unique choice at every discrimination. A change in the quantities of X may change such choices and so bring control to different locations. An exit location is one which can receive control in any such choice. When it is desired to limit considerations to a particular calculation the term actual exit location will be used.

B_1, B_3, or C_3, such that N receives control at a certain step of the calculation based on such an X_1* and the given location is the datum location of N at that step. A location which is either a datum location or an exit location will be called live, one which is neither will be called dead. Then it may be shown by induction on the number of steps that a word at a nondatum location remains unchanged and does not affect the calculations at other locations until it receives control. Consequently a dead location remains isolated from the calculation throughout.**

36. The characterization of exit and datum locations appears to be rather difficult. But it becomes possible if we make the following assumption. An outgoing transfer (species B_3 of Par. 23) is allowed only of the word transferred and the datum are both quantities. This assumption in combination with the type condition for control orders (Par. 25 (f)) entails that the locations of the program are fixed as to species. Conversely if a program is fixed as to species, there is - apart from other restrictions such as the table condition (cf. Par. 33) - no loss of generality in making this assumption, since any allowable transfer of orders can be made by a location substitution.

37. Suppose the assumption of Par. 36 is adopted and that X is a regular program. Then we define two categories of locations of X called E and D locations, by inductive specifications as follows:

a. The initial location of X is an E location.

b. If m is an E location containing an order M of species A, B, C_2, or C_3,

---

* More precisely, in any calculation modified as in the preceding footnote.

** The definition of exit and datum numbers for an order was structural (cf. Par. 15) whereas the concepts for a program are defined functionally. If we were to define them functionally for orders we should have to say that a control shift has no datum numbers - only exit numbers.

then the exit location of M is an E location.

c. If m is an E location containing an order M of species C_1 or C_2 then the datum location of M is an E location.

d. If m is an E location containing an order m of species A, B_1, or B_3, then the datum location of E is a D location.

e. If m is an E location and belongs to a group g_i, then any location belonging to that same g_i is also an E location.

f. If m is a D location and belongs to a group g_i, then any location belonging to that same g_i is also a D location.

With these conventions it can be shown that any exit location of X is an E location, and any datum location of X is a D location.* Furthermore, E locations contain only orders.

38. In orders of species B_2, no datum location is relevant to the operation to be performed. In such a case the datum number part of the order plays no role when the order is executed. A similar statement applies to the exit number part of orders of species C_1 and to both the exit and datum number parts of stop orders (species D). All those digits could, therefore, be used for storage, e.g. of location numbers to be used, possibly in connection with the arithmetic orders, for substitutions. This is a complication. It is simpler to assume that all location numbers stored for such purposes are stored in stop orders. Thus, an order of species B_2 is regarded as having no datum number while one species C_1 is regarded as having no exit number. It will be noted that this

---

* Note that the converse is not asserted.

circumstance is taken into account in making the above definitions.

39. Stop orders require special consideration. Under the present assumptions they are used for two purposes, viz. (a) to stop the machine, and (b) to store location numbers. For simplicity (cf. Par. 38), we agree to separate these two functions. In fact, we shall assume that no stop order is in a location which is both an exit and a datum location. Then there are two kinds of live stop orders, as follows:

a. A stop order in an exit location will be called an output. Then, except for the possibility of malfunctioning, the machine will stop with the control on an output. The different outputs will indicate the different alternatives of the calculation. Such stop orders have, in effect, no datum or exit numbers (cf. Par. 38). We could, without loss of generality, suppose that every output is the word J.

b. A stop order in a datum location will be called a storage order. Then if the control reaches a storage order it is an indication of malfunctioning.

40. It will further be assumed that a substitution can replace a datum number by a datum number, and an exit number by an exit number, but cannot confound the two. In fact, if unconditional control shifts are present, we could suppose that only datum numbers could be changed. (This is the case in the actual machine, in which the exit of every order is the next word.)

41. The following further remarks concern the assumption of Par. 36. In conjunction with the table condition that assumption appears to be stronger than the bald assumption of fixity as to species. Moreover, if it is assumed, the restriction on mixed arithmetic orders in the table condition can be dropped, in fact it is merely necessary to suppose that if the datum of an arithmetic

order and all words not cleared from the a.u. are all quantities, then all new words formed are quantities; if at least one of these constituents is an order, then the new words are orders. Such orders can only be carried into the memory by substitutions. On the other hand, one might go to the other extreme and eliminate location substitutions altogether. In fact such a substitution can be accomplished by forming the difference between the new and old numbers in the accumulator, adding in and transferring back. A substitution enables one to extract a location number from an order; but this could also be done by a mechanism for clearing certain digits in the accumulator, leaving the rest. These possibilities are merely noted here; we shall adhere to the assumptions previously made.

G. Regular Programs

42. A regular program was defined tentatively in Par. 34. The discussion in Par. 35 ff. has shown the desirability of making additional restrictions. These restrictions, and some others of less importance which it is convenient to make, are summarized here.

43. A regular program is one satisfying the conditions:

a. The calculation based on the program is typically determinate in the sense of Par. 24.

b. Either the program is primary or the table condition is satisfied.

c. The conditions of Par. 36-40 are satisfied.

d. The initial location is nondatum, and the initial order is an unconditional control shift directing the control to the locations in the body of the program where the calculation properly begins. (This is a matter of convenience. It is motivated in part by the fact that since the method of starting a calculation has not yet

been planned, it would be advisable to save a place for it.* The restriction affects only certain details. Perhaps it will be dropped later.) The location so referred to will be called the starting location.

e. The location numbers in the program correspond to locations actually present, i.e., these numbers do not exceed the length of the program.

f. The calculation terminates, i.e., the control reaches a stop order after a finite number of steps.

III Transformations

A. Definitions

44. Let T designate a numerical function

$$k' = T(k) \qquad (1)$$

which assigns to each positive integer $k \leq m$ a positive integer $k' \leq n$ with the proviso that $k' = n$ for at least one k. We suppose throughout that $T(0) = 0$. The programs X, Y, Z referred to later shall be as follows:

$$X = M_0 M_1 M_2 \cdots M_p,$$
$$Y = N_0 N_1 N_2 \cdots N_q, \qquad (2)$$
$$Z = L_0 L_1 L_2 \cdots L_r.$$

Finally Θ is a class of natural numbers. We then adopt the definitions in Par. 45-50.

45. Transformations of the first kind. Let T be a given numerical function. We define a program transforma-

---

* Another motivating influence was the connection with combinatory logic mentioned in Par. 50.

tion (T) as follows. Let X be a program such that $p \geq n$ and such that every location number k which occurs in an order of X is in the range $0 < k \leq m$. Then (T) (X) is the Y such that $q = p$ and every N_i is derived from the corresponding M_i by replacing every location number k by T (k). (Note that if M_i is a quantity $N_i = M_i$.)

46. Transformations of the second kind. For a given numerical transformation T we define the program transformation {T} as follows. Let X be a program such that $p = m$. (There is no loss of generality in taking $p = m$. For if $p < m$, T is a fortiori defined over the smaller range; and if $p > m$ we can extend T over the larger range by $T(m + j) = n + j$.) Then {T} (X) is the program Y with $q = n$ such that:

a. $N_0 = M_0$

b. If there exists a k with

$$T(k) = i, \quad (i > 0) \tag{3}$$

we suppose given a method of picking a particular one of these, called k_i, then $N_i = M_{k_i}$.

c. If there exists no such k, then $N_i = J$.

This transformation is equivalent to the following. One goes through X from left to right and places each M_k in the location T (k) in Y; in case two or more M_k contend for the same location then M_{k_i} wins out; if no M_k is assigned to location i it is left blank (J). Note that, if there are multiple solutions of (3) for any i, the transformation is not uniquely defined unless the method of choosing k_i is specified.

47. Replacement. Let X and Y be given and let θ be subclass of the positive integers $\leq p$. Then

$$\underset{\theta}{Q}\, x$$

is the program Z defined as follows. If $p = q$, we have $r = p$, $L_0 = M_0$, and, for $i > 0$,

$$L_i = \begin{cases} M_i \text{ if } i \notin \Theta \\ N_i \text{ if } i \in \Theta \end{cases}$$

If $p<q$ we adjoin $q - p$ J's to X and adjoin the numbers $p+1, p+2, \ldots, q$ to Θ, then apply the definition for $p = q$. If $p>q$ we adjoin $p - q$ J's to Y, but do not make any additions to Θ. The result of all this is that we have for Z:

$$r = \text{the larger of } p \text{ and } q,$$

$$L_0 = M_0,$$

while for $i > 0$

$$L_i = \begin{cases} M_i \text{ if } i \notin \Theta,\ i \leq p, \\ N_i \text{ if } i \leq q, \text{ and } i \in \Theta \text{ or } i > p, \\ J \text{ if } i \in \Theta,\ i > q. \end{cases}$$

When Θ is void, the result will be written $\frac{X}{Y}$. The program transformation carrying X into Z will be indicated $\frac{\Theta}{Y}$, $\frac{O}{Y}$, according to whether Θ is nonnull or null.

48. Transformation of the second kind with replacement. We define transformation

$$\left\{ \frac{\Theta, T}{y} \right\}$$

as the transformation transforming an X with $p = m$ into

$$Z = \frac{\Theta}{y} \left(\{T\} (X) \right).$$

Where Θ is unexpressed it shall comprise precisely those numbers i for which solutions of (3) do not exist. This composite transformation can be independently defined. The definition would then include both Par. 46 and 47 as special cases. In fact, Z is characterized as follows:

$$r = \text{the larger of } n, q$$

$$L_0 = M_0$$

while for L_i, $i > o$ we have the following cases:

a. If i is not in Θ and there exists a solution of (3) (this implies

$i \leq n$), then we presuppose a method of selecting a principal solution k_i, and $L_i = M_{k_i}$.

b. If i is in Θ or $i > n$, and $i \leq q$, $L_i = N_i$.

c. If i is not in Θ, $i \leq n$, and there exists no solution of (3), then $L_i = J$.

d. If i is in Θ and $i > q$, then $L_i = J$.

The transformation of Par. 46 is the special case where ... and Θ is void, that of Par. 47 is the case where ... the identity.

49. Transformations of the third kind. We define the transformations

$$[T] , \left[\frac{T}{Y}\right] , [\Theta T] , \left[{}^{\Theta}\frac{T}{Y}\right]$$

as the transformations carrying X into the following, respectively,

$$[T](X) = \{T\}(T)(X),$$

$$\left[\frac{T}{Y}\right](X) = \left\{\frac{T}{Y}\right\}(T)(X),$$

$$[\Theta T](X) = \left\{\frac{\Theta T}{O}\right\}(T)(X),$$

where O is a void program

$$\left[{}^{\Theta}\frac{T}{Y}\right](X) = \left\{\frac{\Theta T}{Y}\right\}(T)(X) .$$

That is, a transformation of the third kind is one of the first kind followed by one of the second kind with the same T.

50. Notation for transformations. Since the above transformation is determined by a numerical function T, transformations can be represented, according to the above conventions, by any suitable notation for such functions.

The above definitions also suggest a somewhat similar situation which has arisen in combinatory logic. (See reference (c), which gives further references, also reference (f).) The next few sentences require some technical knowledge of this phase of mathematical logic; but they are not necessary for the rest of the report. In fact, if we consider the transformation of Par. 46 and suppose that for every i all the M_k in X for which (3) holds are identical, then there is a unique normal variator (i.e. combinator corresponding to a variation or "Umwandlung") U such that

$$U N, N_2 \cdots N_q \text{ "}\Rightarrow\text{ "} M, M_2 \cdots M_p ,$$

where the symbol "$\Rightarrow$" indicates a reduction in the sense of combinatory logic, and the words are regarded merely as names for "entities" which can be manipulated according to the rules of the theory of combinators. Otherwise expressed, if $t_k = T(k)$, then

$$U x_0 x_1 \cdots x_q \Rightarrow x_{t_0} x_{t_1} x_{t_2} \cdots x_{t_K} .$$

This relationship to combinatory logic suggests some compact forms of statement which can be used occasionally to supplement the more usual ones. These can be stated nontechnically even though the motivation cannot. Thus if $\alpha_1 < \alpha_2 < \cdots < \alpha_s$, the notation

$$B^{\alpha_s} K^{\beta_s} \cdots B^{\alpha_2} K^{\beta_2} \cdot B^{\alpha_1} K^{\beta_1} \qquad (3a)$$

will denote the transformation T such that

$$T(k) = \begin{cases} k \text{ if } 0 < k \leq \alpha_1 , \\ k+\beta_1 \text{ if } \alpha_1 < k \leq \alpha_2 \\ k+\beta_1+\beta_2 \text{ if } \alpha_2 < k \leq \alpha_3 \\ k+\beta_1+\beta_2 \cdots\cdots + \beta_s \text{ if } \alpha_s < k \end{cases}$$

Also T will be said to be <u>K - free</u> if (3) always has at least one solution; <u>W - free</u> if (3) has no multiple solutions; and <u>C - free</u> if T is monotone increasing.

51. <u>Transformations of configurations</u>. We consider now how to extend a transformation to a configuration. We do this for a very general program transformation

$$S(X) = \left\{ \frac{\Theta T_\alpha}{\gamma} \right\} (T_1)(X)$$

Let Q be a configuration with program X, control location c, and such that the contents of the a.u. is a. Since a transformation of the first kind changes location numbers but not locations, and one of the second kind does the reverse, it is natural to define S(Q) as the configuration whose program is S(X), control location $S(c) = T_2(c)$, and such that the contents of the a.u. is $S(a) = (T)^2(a)$.

B. Homomorphic Transformations.

52. Definition. Let $\{Q_s\}$ be a calculation whose successive steps are $Q_0, Q_1, Q_2, \ldots$ Let S be a transformation on configurations. We say S is a homomorphic transformation for Q_s if, and only if, in the calculation whose initial configuration is $S(Q_0)$ the s'th configuration for every s is $S(Q_s)$.

53. Let X be a regular program, and let T be a numerical transformation which is defined for every location number in X. Then T shall be said to satisfy the difference condition, if and only if, whenever x and y are location numbers of locations in the same group $\mathfrak{S}_i$ (Par. 33)

$$T(x) - T(y) = x - y \qquad (4)$$

in other words, if the differences of location numbers in the same group are invariant.

54. Theorem. Let X be a regular program and T a numerical transformation which is W - free (Par. 51) and satisfies the difference condition (Par. 53). Then, for any Y, $\left[\frac{T}{Y}\right]$ is a homomorphic transformation for the calculation initiated by X.

Proof. Let $X' = [T](X)$. Let $\{Q_s\}$ be the calculation based on X, $\{Q'_s\}$ that based on X'. Let Q_s have the program X_s, control location c_s, and accumulator contents a_s; while these three entities for Q'_s are X'_s, C'_s, a'_s. Let N_s be the order in location c_s of X_s; N'_s that in location c'_s of X'_s, and let d_s, d'_s be the datum numbers, e_s, e'_s the exit numbers, and M_s, M'_s the data of N_s, N'_s respectively. To prove the theorem we show by induction on s that $Q'_s = \left[\frac{T}{Y}\right](Q_s)$ i.e. (by Par. 51),

$$X'_s = \left[\frac{T}{Y}\right](X) \; ; \; c'_s = T(c_s) \; ; \; a'_s = (T)(a_s) \; . \qquad (5)$$

The proof is completed in Par. 55-64.

55. The relations (5) hold for s = 0. For by definition

$$X'_0 = X' = \left[\frac{T}{V}\right](X) = \left[\frac{T}{V}\right](X_0) .$$

Since a_0 and a'_0 are equal*

$$T(a_0) = a_0 = a'_0 .$$

Since $c_0 = c'_0 = 0$

$$T(c_0) = c'_0 .$$

56. From now on suppose (5) holds for a given s. We show first the induction for the second equation in (5). Now c'_s is the exit number of N'_s (except in the case of a control shift or stop order, for this see Par. 61-64). The equation $T(k) = c'_s$ has the solution $k = c_s$; since T is W-free, this solution is unique. Hence, by definition of $\left[\frac{T}{V}\right]$

$$N'_s = (T)\ (N_s).$$

Therefore, by definition of (T) and $c_s \neq 1$

$$c'_{s+1} = T(c_{s+1}). \qquad (6)$$

57. Let (5) hold and let N_s be an arithmetic order. Then

$$X'_{s+1} = X'_s \ ; \ X_{s+1} = X_s .$$

Hence, by the hp. of the induction

$$X'_{s+1} = \left[\frac{T}{V}\right](X_{s+1}) . \qquad (7)$$

Further since $N'_s = (T)(N_s)$ and d_s, d'_s are corresponding datum numbers,

$$d'_s = T(d_s). \qquad (8)$$

---

* This remark follows by the footnote to Par. 25 (a). Alternatively, it could be made part of the definition of homomorphic transformation.

Now the solution k of $d'_s = T(k)$ is unique; hence, by the definition of $\left[\frac{T}{Y}\right]$,

$$M'_s = (T)(M_s). \tag{9}$$

We also have, since a.u. has only quantities,

$$a'_s = (T)(a_s) = a_s. \tag{10}$$

If N_s is pure, then $M'_s = M_s$; further $(T)(a_s \neq 1) = a_s \neq 1$. Since $a'_s \neq 1$ is formed by the same operation from a_s and M_s as was $a_s \neq 1$, we have

$$a'_{s+1} = (T)(a_{s+1}). \tag{11}$$

On the other hand, if N_s is mixed, M_s is an order containing a location number x belonging to some group g_i (Par. 33). The effect of N_s is to add x to a quantity z in the accumulator so as to form a second location number y belonging to the same group. In the transformed calculation M'_s is, by (9), an order containing T(x), while accumulator holds T(s) = s. The effect of N'_s is to form in the accumulator the location number $T(x) \neq s = T(x) \neq y - x$. By (4) this is T(y). Since the rest of the a.u. is unaffected by N_s or N'_s and (5) holds, we have (11) in this case also.

58. Let (5) hold, and let N_s be a transfer order. Let L, L' be the words transferred in the original and transformed calculations, respectively. Then

$$L' = (T)(L). \tag{12}$$

For if L originates in the a.u., this follows by the third equation in (5); and if L is the datum, then we have (9) as in Par. 57 and hence (12).

59. If the transfer is into the a.u., then (7) holds as in Par. 57. Further from (12) and from the fact that all other locations in the a.u. are unchanged, we have (11).

60. If the transfer is into the memory, then L, L' replace M_s, M'_s, in d_s, d'_s, respectively. Now from the definition of $\left[\frac{T}{Y}\right]$ in Par. 49, $\left[\frac{T}{Y}\right](X_s \neq 1)$ differs from $\left[\frac{T}{Y}\right](X_s)$ at most in the location d'_s. In that location,

since d_s is the unique k satisfying $T(k) = d'_s$ and since L occupies location d_s in $X_s \not\perp 1$, $\left[\frac{T}{Y}\right](X_s \not\perp 1)$ has (T)(L), i.e., L'. But $X'_s \not\perp 1$ is formed from $X'_s = \left[\frac{T}{Y}\right](X_s)$ by putting L' in location d'_s. Hence (7) holds. Since a.u. is unchanged, (11) is clear.

61. Let (5) hold and let N_s be a control order. If N_s is an unconditional control shift, the case is trivial because such an order does not change either the program or the a.u., and the control can be taken care of as in Par. 56 (cf. also Par. 62).

62. For a discrimination order, the case is different. These orders likewise do not change X or a, so that (7) and (11) hold as before. But the proof of (6) has to be revised. In fact c_s is either d_s or the exit number e_s of N_s; and c'_s is similarly either d'_s or e'_s. The choice between these alternatives depends on the sign of a quantity in the a.u. This quantity is the same in both cases by the hp. of the induction (since it is a quantity and the third equation (5) holds). Hence we have either $c_s \not\perp 1 = d_s$, $c'_s \not\perp 1 = d'_s$ or $c_s \not\perp 1 = e_s$, $c'_s \not\perp 1 = e'_s$. In either case the argument of Par. 56 proves (6).

63. If N_s is a substitution order, the case is similar to an outgoing transfer. The a.u. is not affected, and the effect in X is to replace M_s by L, where L is obtained from M_s by the substitution. It is only necessary to show (12). This, however, is immediate from the hp. of the induction.

64. If N_s is an output, the machine stops and preserves Q_s. Then N'_s also stops the machine and preserves Q'_s. There is then nothing to prove.

65. We consider now the question of weakening the hypotheses so as to admit some cases where T is not W - free. We should not expect T to be homomorphic, in general, in such cases; and simple examples show the expectation is correct. But we consider here <u>some special cases where the theorem of Par. 54 remains true even though T is not W - free.</u> In these cases, for a given i, the solutions of (3) are a certain set of locations called the <u>i - set</u>. The cases are important because they allow the dropping out of certain

superfluous words to be subsumed under the notion of transformation.

66. The first case is that where every i - set contains at most one live location, and k_i is the unique live location if it exists and is otherwise arbitrary. For we used W - freeness in Par. 54 ff. only to know that the word in location c_s^i, d_s^i, c_s^i of X_s^i must be the image of the word in location c_s, d_s, c_s of X. But these locations are live. Hence, the argument is equally sound under the weaker hypothesis.

67. Another case of interest is that where among the live locations of the i - set all except at most one is an unconditional control shift whose datum is another word in the same i - set. We then take k_i to be the unique exception.* This case we can handle by modifying the definition of a step. In fact let a step (in the old sense) where the control is at one of the words of the i - set be called a substep, the one where the control is k_i being the principal substep, and let a maximal set of consecutive substeps in the same i - set be called a full step. Then a full step in the original calculation corresponds to a step in the transformed. We use an induction where s is the number of full steps. The orders giving the substeps of the (s + 1)st step, in the order in which they receive control, will then have the following form:

Location No.	Operator**	Datum No.	Exit No.
c_s	O_s	d_s	$e_s = j_o$
j_o	C	j_1	j_1
j_1	C	j_2	j_2
. . . . .			
$j_p - 1$	C	j_p	j_p
$j_p = c_s + 1$	$O_s + 1$	j_p	$e_s + 1$

* Unless there are no exit locations in the i - set such a k_i exists by Par. 43(f).

** The code in reference (g), Table 2, is used; O_s, $O_s + 1$ are unspecified operators.

Here the order above the dashed-line is the last order of the sth full step. The Q_s which results from this full step is the same as that of the last substep (above), alone, provided we change e_s to $c_s \neq 1$; the latter change has no effect in $\left[\frac{T}{Y}\right]$ (Q_s). This reduces the present case to Par. 54. It is assumed that all locations in such an i-set are non-datum locations.

C. Normal Programs

68. Permutations which move the various groups (Par. 34) as rigid units (i.e., without disturbing the differences of location numbers of locations in the same group) are special cases of homomorphic transformations. Since the locations in the same group are all of the same type, we can segregate the types of an arbitrary regular program. If the orders are at the beginning, we shall have a program of the form X = A C, where A consists of orders only, C of quantities. In such a case X will be said to be normal; A will be called the order program or order part of X, C the quantity program or quantity part. We discuss here some matters connected with this normal form.

69. The order part of X gives the general kind of calculation. If we change the quantity program, we shall get the same kind of calculation with a different set of data. In actual practice these two programs are likely to be stored separately, and we are likely to use the same order program with many different quantity programs. But it is necessary, in such cases, that the order and quantity programs correspond. In practical work, safeguards must be provided to insure that this will be so.*

70. In the more elaborate combinations of programs, transformations which alter the locations of the quantity program only are of some importance. This leads us to adopt the following notation. Let the quantity program of X be of length n and the order program of length m. Let $x_1, \ldots, x_n$ be integers. Then $X(x_1, x_2, \ldots, x_n)$ shall mean the program $[T]$ (X) where

* This much program composition was provided for in the Torc machine (see Par. 4); furthermore, this was so done that the order and quantity programs could be stored separately on tapes and used as often as desired.

$$T(K) = K \text{ if } 0 < K \leq m$$

$$m + x_{K-m} \; m < K \leq m+n.$$

To illustrate this concept, let X be a program for adding two numbers. In the code of the Princeton machine, modified to conform with Section II, the order program would be as follows:

Location No.	Operation	Datum	Exit
1	∤ *	5	2
2	h	6	3
3	S	7	4
4	Stop		

The quantity program is:

Location No. in X	Location No. in C	Initial Quantity	Final Quantity
5	1	x	x
6	2	y	y
7	3	-	x ∤ y

Then X (1, 2, 2) would be the program

1	∤	5	2
2	h	6	3
3	s	6	4
4	Stop		
5	1	x	x
6	2	y	x ∤ y

* i.e., add and clear. The code in reference (g) does not provide a symbol for this operator.

This differs from X = X (1, 2, 3) in that y is replaced by x $\not$ y instead of being retained. On the other hand X (1, 1, 2) is

1	$\not$	5	2
2	h	5	3
3	S	6	4
4	Stop		

5	1	x	x
6	2	-	2x

This requires x and y to be equal. (Note that the transformations used are not homomorphic and also that C contains spaces not only for the given quantities, but for calculated ones.)

71. In later sections we shall allow $x_1, \ldots, x_n$ to be symbols for the quantities of some larger program, which can be translated into location numbers by a table of the quantity part of that program.

72. Characteristic of a Program. The following three integers will be important concerning any program: (1) the length of A called the order length, (2) the length of C or quantity length, and (3) the number of outputs. If these three numbers have the values α, β, and τ respectively (these symbols will be the ones generally used hereafter), the program will be said to have the characteristic (α, β, τ).

IV Program Composition

73. We now embark on our plan of studying the combination of programs. The fundamental concept here is that of substitution. A program Z will be said to be formed by substitution of Y for a certain output in X, when Z carries on a calculation homomorphic to X until

the control reaches that output, then starts a calculation homomorphic to Y using the quantities calculated by X as quantity program. We start with the simplest cases of substitution then build up to more complex ones. The latter must include the formation of loop compositions, where some output of Y restarts X or some part of it.

74. The developments here are so carried out as to preserve consecutiveness of orders as much as possible. This causes a little more complexity, perhaps, than is strictly necessary. But it is thought it will have advantages when the theory is applied to the actual machine.

75. All programs discussed in this chapter are assumed to be regular programs.*

A. Simple Substitution

76. Suppose first that

$$X = AC \qquad Y = BC. \tag{13}$$

Here A and B are order programs; the quantity program C is supposed to be the same in both X and Y. Let α, β, γ be the lengths of A, B, C, respectively. Let m be the location number of a word M in A for which Y is to be substituted.** Let the starting location (Par. 43 d) of B have location number n.

77. First we define transformations T_1 and T_2 as follows:

$$T_1(k) = \begin{cases} k & \text{for } 0 < k < m, \\ m+n-1 & \text{for } k = m, \\ k+\beta-1 & \text{for } m < k \leq \alpha+\gamma. \end{cases} \tag{14}$$

---

* This will require a restriction on the transformation T_0 below. The outputs m_i must not separate any of the groups g_i into two separate parts.

** This location is to be a nondatum location.

Then the programs T_1 (X) and T_2 (Y) both have the lengths $\alpha + \beta + \gamma - 1$.

78. Now let Θ consist of the numbers k for which

$$m \leqq k < m + \beta. \tag{15}$$

Then let

$$Z = \left[\frac{\Theta T_1}{[T_2](Y)} \right](X). \tag{16}$$

The Z given by (16) is the program formed by substituting Y in X at M.

79. By the definitions in Par. 44 ff., it follows that if we set

$$X = K_0 K_1 K_2 \cdots K_{\alpha + \gamma}$$
$$Y = L_0 L_1 L_2 \cdots L_{\beta + \gamma},$$

then

$$(a) \quad Z = N_0 N_1 N_2 \cdots N_{\alpha+\beta+\gamma-1},$$

where

$$N_0 = K_0$$

and

$$(\alpha). \text{ If } 0 < k < m, \quad N_k = (T_1)(K_k)$$
$$(\beta). \; m \leqq k \leqq m + \beta - 1, \quad N_k = (T_2)(L_{k-m+1})$$
$$(\gamma). \; m + \beta - 1 < k, \quad N_k = (T_1)(K_{k-\beta+1}).$$

80. Consider now the effect of this program Z. By Par. 45, the transformation $\left[\frac{T_1}{[T_2](Y)} \right]$ is a homomorphic transformation of X(X being a regular program). The locations in Θ are dead locations in $\left[\frac{T_1}{[T_2](Y)} \right](X)$ except the location of M (i.e., $m \neq n - 1$) which is a nondatum location. If the words in these locations are replaced by the corresponding words of $[T_2]Y$, there is no change in the calculation until the control reaches $m \neq n - 1$. On the other hand, the calculation

$$\left[\frac{T_2}{[T_1](X)} \right](Y) \tag{17}$$

differs from Z only in the initial order. The calculation is homomorphic to Y, the locations corresponding to those in A are dead, and the calculation starts at location m / n - 1. Thus when the control in Z reaches m / n - 1, the situation is the same, except for change in C, as though it had reached that location in (17). From then on the calculation is homomorphic to Y with the changed C. Thus Z has the effect stated in Par. 73.

81. We shall be interested in this form of composition primarily in the case where M is an output of X. If we suppose that, as part of the specification of X, the outputs are specified and assigned numbers, say O_1, O_2, ..., O_p, then Z is uniquely defined as soon as the number of the output is given. With the understanding that the output is O_1, Z is determined by X and Y alone and can be symbolized, $X \rightarrow Y$. The way in which the outputs are numbered in $X \rightarrow Y$ will be determined later. Note that the order length of $X \rightarrow Y$ is $\alpha / \beta - 1$; its quantity length is γ, the same as that of X and Y. If the numbers of outputs in X and Y, respectively, are t_1 and t_2, that of Z is $t_1 / t_2 - 1$.

82. A degenerate case of $X \rightarrow Y$ will be useful later. In case B, except for the initial order, consists solely of a single stop order, we write Y as O. Then it is clear that, except for the numbering of outputs, $X \rightarrow O = X$.

B. Multiple Substitution

83. Suppose that X is a program with p outputs, O_1, O_2, ..., O_p *, and that X, $Y_1, Y_2 \ldots, Y_p$ are regular programs of the form

$$X = A_0 C \qquad Y_i = A_i C \qquad i = 1, 2, \ldots, p \tag{18}$$

Then we define the program

$$Z = X \rightarrow Y_1 \beta Y_2 \beta \cdots \beta Y_p \tag{19}$$

as the program obtained by substituting Y_1 for O_1, then

---

* These need not exhaust the outputs of X.

Y_2 for O_2, then Y_3 for O_3, etc. and finally Y_p for O_p, as in Par. 76. With proper renumbering of outputs we have

$$Z = ((\ldots(X \to Y_1) \to Y_2) \to Y_p)$$

84. The program Z can be characterized in the following manner. Let M_i be the ith output O_i and let m_i be its location number; the m_i are then all distinct from one another. Let n_i be the starting location of Y_i. Form the program Z' by inserting each A_i in the place of the corresponding M_i and renumbering; this Z' is independent of the order of making these insertions. Let $T_i(k)$ be the number of the location occupied by the kth word of Y_i in Z'; and let $T_0(k)$, for k not any of the m_i, be the number of the location occupied by the kth word of X in Z', while $T_0(m_i) = T_i(n_i)$. The transformations T_0 and T_i so defined are both W — free and C — free; in fact they are of the form (3a) in Par. 50. Let Θ_i be the class of integers j for which there exists a k such that $T_i(k) = j$ and such that k is the number of a location in A_i. Let the kth word in X be K_k, in Y_i be L^i_k. Then the kth word N_k in Z is as follows:

a. If k is not in any of the Θ_i, let h be the unique solution of $T_0(h) = k$; then $N_k = (T_0)(K_h)$.

b. If k is in Θ_i, then i is unique; furthermore, there is a unique h such that $T_i(h) = k$. Then $N_k = (T_i)(L^i_h)$.

The proof of these statements may be obtained by an induction on P. This is a straightforward but rather tedious process. It will be omitted here.

85. It follows from the characterization of Par. 84 that

$$X \to Y_1 \,\&\, Y_2 \,\&\, \cdots \,\&\, Y_p$$

is independent of the order in which the substitutions are made, because the result in Par. 84 depends only on the correlation between the m_i and the Y_i. We can make this

explicit by defining T_0, T_i thus: let τ of the m_i be $< k$, and let the sum of the τ corresponding α_i be ρ. Let m_j be the greatest of the m_i which are $\leq k$. Then we have

$$T_0(K) = \begin{cases} K \text{ if no } m_i \leq k, \\ K + \rho - \tau \text{ if } m_j < K, \\ m_j + \rho - \tau + n_j - 1 \text{ if } m_j = k. \end{cases}$$

As for $T_i(k)$ let ρ, τ be as above for $k = m_j$; then

$$T_i(K) = \rho - \tau + m_i + k - 1.$$

86. As in Par. 84 we have the possibility of degenerate cases where one or more of the Y_i is 0.

C. Reduction to a Single Quantity Program

87. We consider now the composition by multiple substitution of programs

$$X = A_0 C_0$$
$$Y_i = A_i C_i \qquad i = 1, 2, \ldots, P \qquad (20)$$

This can be reduced to the substitution of Par. 83 by transformations. The result is to be a program

$$Z = BD. \qquad (21)$$

88. When such a composition is contemplated, it willpresumably be the case that we wish to identify each quantity in every C_i with some quantity in D. We assume this and let the number of the location in D which is assigned to the kth location in C_i be $S_i(k)$ ($i = 0, 1, \ldots, P$). Let α_i be the length of A_i, γ_i that of C_i, β that of B, and δ that of D. Then form the transformation

$$T_i'(k) = \alpha_i + S_i(k - \alpha_i).$$

In the notation of combinatory logic (cf. Par. 51), this is the same as

$$T_i' = B^{\alpha_i} S_i \qquad (22)$$

Now set

$$X' = [T_0'](X)$$
$$Y_i' = [T_i'](Y_i) \qquad i = 1, 2, \ldots, P. \qquad (23)$$

Then we shall have

$$x' = A_0' C_0'$$
$$y_i' = A_i' C_i' \qquad i = 1, 2, \cdots, p, \qquad (24)$$

where

$$A_i' = (T_i')(A_i) \qquad i = 0, 1, 2, \cdots, p$$

and C_i is a program of length δ.*

89. Now replace C_i by D thus obtaining

$$x_i'' = A_0' D$$
$$y_i'' = A_i' D \qquad (25)$$

Then we adopt as definition of Z

$$z = x'' \longrightarrow y_1'' \,\&\, y_2'' \,\&\, \cdots \,\&\, y_p'' . \qquad (26)$$

90. If the locations in the various C_i and D which are identified by the S_i actually contain the same words, then

$$x'' = \left[\frac{T_0'}{\{K^{\alpha_0}\}D}\right](x),$$
$$y_i'' = \left[\frac{T_i'}{\{K^{\alpha_i}\}D}\right](y_i), \qquad (27)$$

where $\{K^{\alpha_i}\}D$ is explained in Par. 50. The assumption is fulfilled in case the C_i and D consist exclusively of blanks. As stated in Par. 69, this is likely to be the

---

* Note that C_i differs from D in that the kth location in C_i is blank if the location is not one of those assigned to a location in C_i by S_i; if it is one so assigned, the kth word is the same as in one of the locations in C_i. The ambiguity in case two or more locations in the same C_i are identified will make no difference in the definition of Z below.

case when we are compounding programs practically. Under these circumstances any ambiguities there may be due to identification of different locations in the same C_i are irrelevant.

91. In order to get a notation for this Z, we develop a little further the ideas of Par. 70. Let z_1, z_2, ..., and z_δ be a notation for the locations of D. Let

$$x_j = z_{s_0(j)}$$
$$y_{ij} = z_{s_i(j)} \qquad (28)$$

then we can set

$$x'' = X(x_1, x_2, \cdots, x_{\alpha_0})$$
$$y_i'' = Y_i(y_{i1}, y_{i2}, \cdots, y_{i\alpha_i}).$$

(These notations are ambiguous, unless D is given.) Then the definition of Z can be stated as follows:

$$Z(z_1, z_2, \cdots, z_\delta) = X(x_1, x_2, \cdots, x_{\alpha_0}) \longrightarrow$$
$$Y_1(y_{11}, \cdots, y_{1\alpha_1}) \& \cdots\cdots \& Y_\rho(y_{\rho 1}, \cdots, y_{\rho\alpha_\rho}). \qquad (29)$$

92. We now consider the characterization of Z directly in terms of X and the Y_1, Y_2, ..., Y_ρ. Let T_0', T_1', ..., T_ρ' be as in Par. 88. Let T_0'', T_1'', ..., T_ρ'' be the T_0, T_1, ..., T_ρ of Par. 84 defined with reference to X'', Y_1'', ..., Y_ρ''. Then set

$$T_i = T_i'' T_i' \qquad i = 0, 1, 2, \ldots, \rho.$$

Let the Θ_i be defined as in Par. 84 with reference to the T_i''. Let the kth words of X, X'', Y_i, Y_i'', Z, respectively, be K_k, K_k'', L_k^i, $L_k^{i''}$, N_k. Then we have:

a. If k is in Θ_i, there exists a unique h in the range

$$0 < h \leq \alpha_i \text{ such that } T_i'(h) = T_i''(h) = k;$$

then

$$N_k = (T_i'')(L_k^{i''}) = (T_i'')(T_i')(L_h^i) = (T_i)(L_h^i).$$

b. If k is not in any of the Θ_i and

$$0 < k < \beta = \alpha_0 + \alpha_1 + \cdots + \alpha_p - p ;$$

then there exists a unique h in the range $0 < h < \alpha_0$ such that

$$T_0(h) = T''_0(h) = k .$$

For this h

$$N_k = (T''_0)(K''_h) = (T''_0)(T'_0)(K_h) = (T_0)(K_h) .$$

c. If $k > \beta$, then k is not in any of the Θ_i. There exists a unique h in the range $\alpha_0 < h \leq \alpha_0 + \delta$ viz., $h = k - \beta + \alpha_0$, such that $T''_0(h) = k$. Then N_k is the word of index h in X", and this is precisely the word of index $h - \alpha_0 = k - \beta$ in D. If the condition of Par. 90 holds and if $h - \alpha_0 = \delta_0 (j - \alpha_0)$, $h = T'_0(j)$, then N_k is the same as K_j; if $h - \alpha_0 = \delta_i (j - \alpha_i)$ then N_k is the same as L^i_j.

93. The characterization of Par. 91 leads to the following practical procedure for constructing the program Z. In the first column of a sheet of columnar paper we write the successive locations of Z, the total numbering $\beta + \delta$. In the second column we write the transformation T_0 by going through X and writing h in the kth row whenever $T_0(h) = k$; if h is one of the m_i, we put it in parentheses. In the $(i + 2)$nd column we do the same for T_i. It is then easy to write out Z. Given any $k \leq \beta$ there will be only one column in which there is an unparenthesized h. That column shows which program, X or Y_i, should be used. We now look up this order in the given program and make the appropriate transformation. This can be done by finding h in the appropriate column and replacing it by the corresponding k. It is intended to give examples of this technique and modifications in the future report cited in Section I. Evidently the process could be carried out with a suitable machine; considerable memory

might be involved, but not extensive calculation; however, this question is not gone into at present.

94. As to the effect of the program, it is evident that Z makes a transformation homomorphic to X", and hence to X with the quantities of C_0 taking the values prescribed by D, until the control reaches some $T_i(m_i)$. The calculation then proceeds according to a Y_i, in which C_i has been replaced by the appropriate quantities from the D as it has been calculated up to that time.

D. Loop Programs

95. In order to carry out iterative calculations, it is necessary to have programs which double back on themselves, repeating certain parts as prescribed a number of times, or until certain conditions have been fulfilled. We consider here the formation of such programs.

96. If X is a program with an output O_i in location m_i, we can form a loop program by requiring that when the control reaches m_i the calculation will start all over again at n_i. The order located at n_i will then be called the input. This can be accomplished simply by replacing O_i with an unconditional control shift which transfers control to n_i. There is thus no particular difficulty about it. In our idealized machine we can accomplish the same end by means of a transformation such that m_i and n_i both correspond to the same j, say

$$T(m_i) = T(n_i) = j,$$

the transformation being otherwise W - free (Par. 50), and K - free and C - free to boot if we want it. In fact, if $n_i < m_i$ we can set

$$T(K) = \begin{cases} K \text{ for } K < m_i \\ n_i \text{ for } K = m_i \\ K-1 \text{ for } K > m_i. \end{cases}$$

We must regard n_i as the principal solution of T(k) = j.

97. The formation of the loop program is thus a kind of substitution of part of X in itself. Evidently we can indicate such a formation by the same sort of notation as for substitution; we have only to replace the appropriate Y_i by an indication of the input. We may suppose that in the specification of X there is designated a certain number of locations indicated by $I_1, I_2, \ldots, I_q$ which can function as inputs. Then the program formed from X by joining the second output to the third input could be indicated: $X \rightarrow O_1 \& I_3$. But actually this is not likely to be necessary. The inputs are likely to be either the start of X or that of some component program. One can use X in the first case and the symbol for the component in the second. Thus the program formed from X by (1) substituting a program $Y \rightarrow Z_1 \& Z_2$ for the first output, (2) leaving the second output to become the O_1 of the new program, (3) going back to the beginning of X in the third output, (4) going to the beginning of Z_2 from the fourth output, and (5) combining the fifth output with the second could be written:

$$X \rightarrow (Y \rightarrow Z_1 \& Z_2), O_1 \& X \& Z_2 \& O_5$$

The repeated appearance of the symbol for a component program is thus sufficient indication of a loop program.

98. The combination of outputs can be regarded as the formation of a loop. See example in Par. 97.

99. Presumably, theorems analogous to those proved in Par. 83 and 87 will hold in this case also. This question is left open for the present.

E. Complex Programs

100. We consider now the formation of more complex programs by repeated applications of the above processes. We take as fundamental composition mode

$$X \rightarrow Y_1 \& Y_2 \& \cdots \& Y_p \quad (30)$$

and consider how to keep the notation straight when X, $Y_1, \ldots, Y_p$ are formed by (possibly iterated) composition

from certain ultimate components U_1, U_2, ..., U_r. It will be supposed that these are regular programs whose programming is completely known, and that on those known programs, we have a definite numeration of the outputs, which is regarded as a part of the program. For each U_i it is also supposed that we know the starting location, the order length, the quantity length, and any inputs which may be necessary.

101. In an expression of the form (30), the "X", "Y_1", ..., "Y_p", or whatever symbols take their place, will be called the constituents; the constituent in the place of "X" will be called the functional constituent; those in the places of "Y_1", ..., "Y_p" the argument constituents. A constituent which does not contain parts of the form (30) will be called an ultimate constituent. It is supposed that the ultimate constituents are symbols for the U_i, or programs derived from them by transformation, and for outputs. An ultimate constituent which is an argument constituent will be called an argument; if it is not part of any functional constituent it will be called an ultimate argument. It will be supposed at first that the arguments are all symbols for outputs.

102. In order to take care of loop programs, it will be necessary to modify this scheme, so as to have notations for inputs and to allow them to appear as arguments. A possible notation, which will be adopted tentatively, is to allow a constituent, which appears in a unique position as ultimate functional constituent, to appear as argument, with the understanding that when this happens the output at that point is to be joined to the input at the point where the constituent occurs as functional constituent. When it is desired to be explicit, these input indications will be enclosed in angular brackets.

103. Additional modifications may be necessary to take care of transformations. These will not be considered here. No particular trouble is anticipated in regard to them.

104. The ultimate arguments are the arguments of the composite program; their numbers are to indicate the numbering which they are to have in the final programming. When the same number appears two or more times the outputs are to be identified.

105. A functional constituent which is a symbol for an ultimate component, with or without a transformation, will be supposed to have the outputs stated on the programs for those constituents. Otherwise the outputs of every constituent will be explicitly indicated. In either case whenever an expression of form (30) occurs, the number of argument constituents is to be exactly equal to the number of outputs in the functional constituent, and the expression is to indicate the program formed as in Par. 83 by substituting Y_i for the ith output of X . Note that if an "O_i" occurs as part of a functional constituent it is not an output of the whole program.

106. Repeated use of the notation (30) leads to a large number of parentheses. Various devices may be employed to make the structure of the program more perspicuous. The following methods will be considered here and illustrated with reference to the program:

$$U_1 \rightarrow (U_2 \rightarrow (U_4 \rightarrow O_1 \,\&\, \langle U_1 \rangle) \,\&\, (U_5 \rightarrow \langle U_3 \rangle \,\&\, O_3) \,\&\, \langle U_3 \rangle) \,\&\, (U_3 \rightarrow O_2 \,\&\, O_1) .$$

107. The Peanese dot notation is said to have been used by Leibniz; it has been extensively used by Peano and many writers in modern symbolic logic. In this notation, brackets are replaced by groups of dots. They are placed beside an operational sign to indicate a bracket which extends away from that sign until it meets an equal or larger number of dots. This scheme will here be modified so that a group of dots on the right of "$\rightarrow$" takes precedence over one attached to "&". Then the above program becomes

$$U_1 \rightarrow : U_2 \rightarrow \cdot U_4 \rightarrow O_1 \,\&\, \langle U_1 \rangle \cdot \& \cdot U_5 \rightarrow \langle U_3 \rangle \,\&\, O_3 \cdot \& \cdot \langle U_3 \rangle : \& : U_3 \rightarrow O_2 \,\&\, \langle O_1 \rangle \,.$$

108. The Polish prefixed operator notation. This notation invented by Lukasiewicz and used extensively by modern Polish logicians depends on the fact that if operational signs are sharply distinguished from other signs

and if a definite number of operands is assigned to each, then parentheses can be inserted in only one way. Thus, if we agree that "$\rightarrow_p$" shall be an operational sign with exactly $p \neq 1$ operands, and that (30) be synonymous with

$$\rightarrow_p \times \gamma_1 \gamma_2 \cdots \gamma_p ,$$

then the above sample program becomes

$$\rightarrow_2 U_1 \rightarrow_3 U_2 \rightarrow_2 U_4 \, O_1 \langle U_1 \rangle \rightarrow_2 U_5 \langle U_3 \rangle \, O_3 \langle U_3 \rangle \rightarrow_2 U_3 \, O_2 \langle O_1 \rangle .$$

Although this notation is compact and can be read with some practice, yet it is not particularly perspicuous.

109. A notation similar to that used by Frege can be devised by writing (30) in the form

$$\times \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \vdots \\ \gamma_n \end{bmatrix}$$

Then the sample program becomes

$$U_1 \begin{bmatrix} U_2 \begin{bmatrix} U_4 \begin{bmatrix} O_1 \\ \langle U_1 \rangle \end{bmatrix} \\ U_5 \begin{bmatrix} \langle U_3 \rangle \\ O_3 \end{bmatrix} \\ \langle U_3 \rangle \end{bmatrix} \\ U_3 \begin{bmatrix} O_2 \\ \langle O_1 \rangle \end{bmatrix} \end{bmatrix}$$

Although this is perspicuous it is not so compact as the Peano notation. Frege's notation, it must be remembered, died with him.

110. Flow chart. This is a quasi-graphical representation of the flow of the control. A flow chart for the sample program is shown in Figure I. Note that this notion of flow chart, which is the same as that in reference (e), is not the same as the "flow diagram" in reference (g). The purpose there is to show all the details of the logical aspect of the program; here merely how the program is compounded from its "ultimate components," which may be quite complicated programs already setup.

111. In dealing with composition of programs theoretically, where we use expressions involving symbols for unspecified programs, we naturally cannot adhere to the requirement that the outputs be indicated. As in Par. 76 to 95, we have to leave the numeration of the outputs unspecified, but we suppose it is done in the same fashion.

F. Associativity Properties

112. The notations in Par. 100 ff. give essentially a unique method of constructing a program from its ultimate components. But it is evident that the same program can be constructed in different ways. In other words, there are equivalences among our programs, so that we have in effect a calculus of program composition. Although a thorough study of that calculus will have to be left until later, there are certain properties which can well be mentioned here.

113. One such property was considered in Par. 85, viz., that in forming the program (30), the Y_i can be substituted in any order. Thus we have the property

$$X \rightarrow Y_1 \,\&\, Y_2 = (X \rightarrow Y_1 \,\&\, O_1) \rightarrow Y_2 = (X \rightarrow O_1 \,\&\, Y_2) \rightarrow Y_1 ,$$

provided O_1 does not occur in X, Y_1, Y_2.

114. Another property is

$$(X \rightarrow (Y \rightarrow O_1)) \rightarrow Z = X \rightarrow (Y \rightarrow Z) .$$

This means the same as

$$(X \rightarrow Y) \rightarrow Z = X \rightarrow (Y \rightarrow Z) \qquad (31)$$

115. In combination with Par. 113, this yields a rather general associativity property. The property (31) may be proved by working out a characterization of both sides, as in Par. 84 and 92, and showing they are equal. The details, which are rather involved, are omitted here.

116. Another property of a similar nature, which is similarly proved, is

$$X(x_1, \ldots, x_m) \rightarrow Y(y_1, \cdots, y_n) = (X \rightarrow Y)(y_1, \cdots, y_n)(x_1, \ldots, x_m),$$

where, with reference to some $D = z_1, z_2, \ldots, z_s$, $x_k = z_{i_k}$, $y_k = x_{h_k} = z_{j_k}$.

V Concluding Remarks

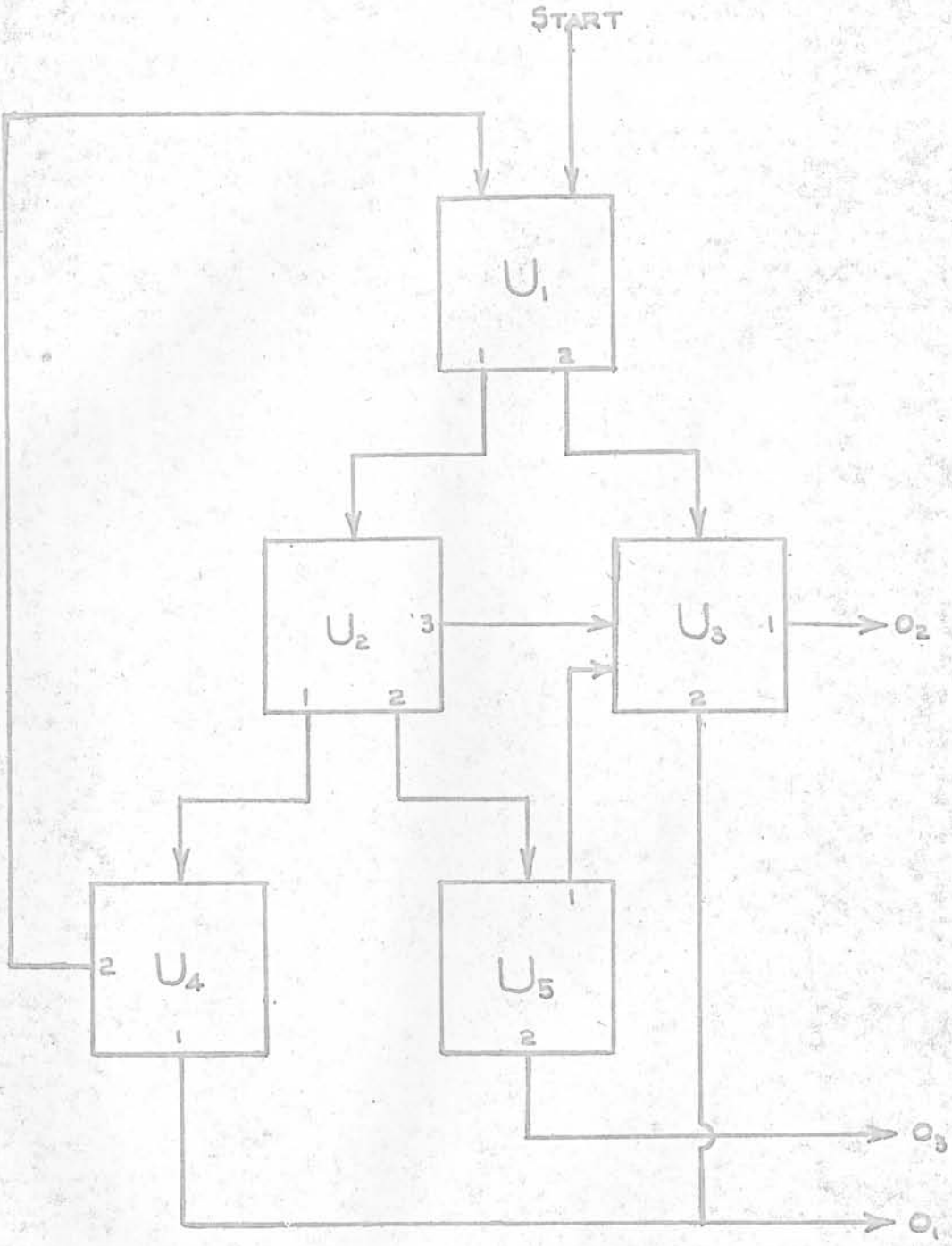
117. The theory of composition of which the basic portions are developed here has reference to a restricted class of programs called regular programs, defined in Par. 42 and 43. It is not known how general this type of program is. But it seems likely that a great variety of problems can be solved by using programs of that character.

118. This class of programs may or may not be as general as that to which the methods of Goldstine and von Neumann (reference (g)) apply. The relationship between the two methods has not been investigated; probably it would be premature to do so until the study of subroutines, which the authors of reference (g) have promised, appears. But several persons have noticed that the technique in reference (g) involves a lot more fuss than is really necessary. The procedures suggested by the theory of program composition, at least insofar as they have been tested on simple problems, appear to lead to a less formidable technique, even when the problems are planned in detail from scratch.

119. The machine with regard to which reference (g) was written has, besides the finite size which it has in common with any machine, two deviations from the idealized machine considered here viz., (1) the fact that orders are stored in the memory in pairs, and (2) the fact that the exit locations of every order except a stop order or a control shift are determined to be the next position in the memory. The modifications due to these peculiarities, and the working out of a technique illustrated by examples, are left for a later paper.

H. B. CURRY

HBC:hra:lsa



FLOW CHART FOR SAMPLE PROGRAM

Fig. 1