NAVORD REPORT 4411

AUTOMATIC DIGITAL ENCODING SYSTEM II (ADES II)

29 NOVEMBER 1956

U. S. NAVAL ORDNANCE LABORATORY
WHITE OAK, MARYLAND

NAVORD Report 4411

Aeroballistic Research Report 359

AUTOMATIC DIGITAL ENCODING SYSTEM II (ADES II)

Part 2: The Encoder

Prepared by:

E. K. Blum

ABSTRACT: Automatic Digital Encoding System II (ADES II) is a system for the automatic translation of mathematical formulas into programs of coded instructions for an electronic digital computer. The system has been described in NAVORD Report 4209 as consisting of a formulation language, an Encoder and a digital computer. NAVORD Report 4209 was concerned primarily with the language. The present report is devoted to a description of the Encoder, the device which actually translates the formulas into computer programs.

U. S. NAVAL ORDNANCE LABORATORY
WHITE OAK, MARYLAND

NAVORD Report 4411 29 November 1956

This report is the second half of a two-part report on the results of research in the field of automatic programming. The two parts taken together describe a system designed to automatize the process of preparing mathematical problems for computation on electronic digital computers. The system relieves the mathematician of a major portion of the programming task. In effect, it replaces the human by an automatic device which carries out the programming procedures. The present report gives the design for such a device. An experimental model has already been constructed and is now operative.

This work was performed under NOL Task No. FR-30-1-57.

W. W. WILBOURNE
Captain, USN
Commander

H. H. KURZWEG
By direction

PREFACE

In the following pages, we present the design of a device, the Encoder, for translating mathematical formulations into computer programs. The formulations are written in the language of ADES as described in NAVORD Report 4209 issued earlier this year.

Whereas Report 4209 was directed primarily at the problem formulator, the present report is intended for those who wish to set up ADES for a particular computer. It is assumed that the reader is wise in the ways of digital computers. Therefore, specific machine techniques are not suggested since these will vary with the machine. It is hoped that a general description of what steps have to be taken in the translation process and the ordering of these steps as specified in schematic flow charts will suffice.

An experimental model of the Encoder has been constructed for the 650 Magnetic Drum Calculator at NOL. Several problems have been programmed successfully. A description of this particular Encoder will be found in NAVORD Report 4412. That report contains a complete listing of the actual 650 machine instructions used to carry out the operations outlined in the flow charts of this report. Although the storage limitations of the 650 forced modifications on the model, it was possible to prove the correctness of the main logical design. From the experience gained with this model, it can be said that ADES II is a practical system of automatic programming. However, it requires a machine with magnetic tapes. It is felt that tapes are essential for the efficient handling of the translating subroutines of the Encoder and for the assembly of the library subroutines of the computer.

In connection with library subroutines, we call attention to the Appendix of this report in which an entirely new idea for handling library programs is put forth. One of the demands frequently made of an automatic coding system is that it be possible to write library programs in the language of the system. In the Appendix, we explain how this is possible in ADES. This is particularly significant since the ADES language is mathematical and requires no knowledge of the computer.

We wish to acknowledge the fine work of Mrs. Rose Strompf who assisted in the preparation of the manuscript and flow charts.

CONTENTS

NAVORD Report 4411

AUTOMATIC DIGITAL ENCODING SYSTEM II (ADES II)

Part 2: The Encoder

I - Introduction

This is the second part of a two-part report on ADES, a system of automatic translation of mathematical formulas into computer programs. The first part, contained in NAVORD Report 4209, was devoted primarily to a description of the ADES language. The language is the central component of ADES. The other components are (1) a Digital Computer and (2) an Encoder. This report is devoted to the Encoder.

The Encoder is a device which receives a mathematical formulation as input and produces a complete Computer program as output. In the following pages, the logical design of an Encoder is described. A general account is given first and this is followed by schematic flow charts. Explanatory notes accompany the flow charts. Since the logical design of the Encoder becomes rather intricate in places, minute details are often omitted. It is felt that the mathematician versed in machine techniques will be able to supply these details in a satisfactory form once he has comprehended the basic pattern.

Although the Encoder is for the most part independent of the Computer, the design presented here was constructed with a particular computer in mind. It was unavoidable, therefore, that certain aspects of the design should be influenced by the Computer. For example, the Computer was assumed to be a single-address floating-point machine. It was also supposed to contain one or more index registers. However, these features are believed to be not entirely essential and can, in any case, be achieved by means of interpretive routines.

As was pointed out in NAVORD Report 4209, attention is limited to ADES II, where the Roman numeral, II, indicates a type of incompleteness. Mathematically, the incompleteness rests in the exclusion of general recursions of order greater than 2 and in the restriction of recursion to the definition of variables. This latter restriction is a significant one since most data processing problems require recursions on indexes. However, the design of the Encoder in ADES II is such that it can be appended or modified to eliminate both of these restrictions.

From the machine viewpoint, the incompleteness of ADES II consists in the exclusion of the logistics for handling external storage; e.g., reading tape. This is a more complex problem and to obtain the most efficient solution seems to require a rather deep analysis. If one is willing to settle for a workable solution, however, it is felt that this can be accomplished within the framework of the logical design presented here.

II - General Remarks

Physically, the Encoder consists of a digital computer with certain subroutines loaded into its storage. After a problem has been stated in ADES language, the formulation is punched on cards or tape exactly as written. The cards are then read into the Encoder and the subroutines translate the ADES formulation into a program for the Computer. This program is punched out on cards or tape. At some later time, it can be loaded into the Computer together with data and the actual computation can take place.

In order to distinguish between the Encoder, which does the translating, and the Computer, which does the computing, we shall set up certain conventions.

A storage register in the Computer will be referred to simply as a "storage", and the number specifying its location will be called a "storage address". An address will be denoted by a small Greek letter ($\alpha, \beta, \ldots$). On the other hand, a storage register in the Encoder will be referred to as a "memory cell" and the number specifying its location will be called a "cell number". The cell number of a cell which contains some mathematical symbol in a formulation will be denoted by writing the sign, "#", followed by the particular symbol; e.g., the cell number of the cell which contains the symbol, b_{01}, is denoted by "$\#b_{01}$". (The reader should check Report 4209 for a list of the symbols in the ADES language.) This is to be distinguished from "$\alpha\ b_{01}$", which will denote the address assigned to b_{01} in the Computer. For simplicity, certain standard cells in the Encoder will be denoted by capital Roman letters (A, B, C ...). The contents of cell A will be denoted by "(A)".

The translating subroutines in the Encoder will be referred to as "schemata". This will distinguish them from the library subroutines used in the Computer; e.g., interpretive routines, routines for computing special functions, etc.

One of the most frequent operations in the Encoder is that of transferring the contents of one cell to another cell. We shall abbreviate and designate this operation by using an arrow in several ways. Thus, "A→B" means "transfer the contents of cell A to cell B"; "$\alpha \rightarrow$ Q" means "place the address, α, into cell Q"; i.e., α is actually the contents of some cell. Although we shall use the arrow in different ways, what is intended will be clear from the context.

Another basic operation in the Encoder is that of "scanning". The scanning operation involves two standard cells, S and SC. Standard cell SC contains the cell number of some cell, and S contains the symbol in that cell. The operation, "scan right", causes the cell number in SC to increase by one and copies the symbol in this new cell into S. The operation, "scan left", decreases the cell number in SC by one and copies the new symbol into S. In short, one may think of the Encoder memory as a continuous tape, subdivided

into cells and visualize a reading head which scan the tape. The operation, "scan right", causes the head to move one cell to the right. The operation, "scan left", causes the head to move one cell to the left. The location of the head is recorded in SC and the symbol being read is recorded in S. The following exception should be noted. A "Spacer" cell is one which contains zero. In scanning operations, spacer cells are automatically ignored. For example, each equal sign is loaded into the Encoder memory with two spacer cells to its left. These spacer cells are reserved to record information during the translation process. Until then, they are skipped over by the scanning operation.

III - Summary Description of Logical Design

The Encoder can be subdivided into 7 main schemata which we shall name as follows: The Initial Addressor, the b-schema, the Addressor, the r-schema, the q-schema, the Recursion-schema and the Interpreter.

Let us assume that an entire formulation has been read into the Encoder memory. (Consult Report 4209 for the structure of an ADES formulation.)

Initial Addressor - The translation begins with the Initial Addressor. This schema carries out some preliminary conversions which simplify the final translating process. It will be recalled that in each ADES formulation there is a section called the Computer Table which gives information about the number of data to be supplied for each independent variable as well as the structure of the data if it should happen to be a matrix. The Initial Addressor scans the Computer Table and calculates an initial address for each independent variable listed in the table. It does this on the assumption that the data is to be read into the Computer in the order in which the variables appear in the Computer Table. In fact, the Initial Addressor composes a reading program to load the data in precisely this way. At the same time, it also determines the degree of each independent variable.

The Initial Addressor then scans the equations in the formulation four times. On the first scan, it compiles three tables. The first table contains a list of the cell numbers of the equal signs of the f-equations. The second table contains the cell numbers of the equal signs of the b-equations. The third table contains the cell numbers of the equal signs of the r-equations.

On the second scan, each independent variable is replaced by its initial address and a tag which indicates the degree of the variable. Each dependent variable (denoted by a subscripted b) on the right side of an equation is replaced by the cell number of the equal sign of the b-equation which defines this variable. Each dependent index (denoted by a subscripted r) on the right side of an equation is replaced by the cell number of the r-equation which defines this symbol. Each function (denoted by a subscripted f) on the right side of the equation is replaced either by the cell number of the equal sign of the auxiliary f-equation which defines the function, or if this happens to be a library function (see NAVORD 4209), the symbol is

replaced by the cell number of a translating subschema which will be called upon later to compose the Computer operation code associated with this library function. Finally, on the second scan, all numerical symbols, denoting constants in the formulation, are assigned addresses and are punched out as data to be read into the Computer storages specified by these addresses. In replacing each b, r, and f symbol, an indentifying tag is also attached so that in later scans the different types of symbols can be distinguished.

On the third scan, the Initial Addressor replaces each independent index (denoted by subscripted q) by an address and an identifying tag. This address designates the Computer storage which will be used as an index register for modifying the initial address of the variables associated with this index.

On the fourth scan, any r symbols in the Computer Table are replaced as above. (Recall that these r's specify matrix structures.)

The Initial Addressor also performs a few odd jobs to prepare the way for the main translation. Thus, for example, it records the location of the Master Phase Equation, that is, the equation which defines b_{oo} and signals the start of the formulation. It also records the location of the comma following the last equation of theformulation, and the location of the comma after the Computer Table.

The b-Schema - The heart of the Encoder is the b-schema. This schema controls the order in which the equations in the formulation will be translated. Beginning with the master phase equation, it processes each b-equation as follows.

Starting at the equal sign, it tests the subscript of the equal sign to determine whether this is a recursion equation, a vector equation, or what might be called an ordinary b-equation. If it is an ordinary equation, the b-schema will first translate the left member of the equation, that is, those symbols contained between the equal sign and the first comma to the left of the equal sign. These symbols contain information about quantification, storing, and output. If a quantification is indicated, the b-schema will transfer control to the q-schema. If storing is indicated by the presence of an r, the b-schema will transfer control to the r-schema.

After the left side of the equation has been processed, the b-schema will begin scanning right from the equal sign to process the right member of the equation. It does this by scanning for b's until it reaches the comma which sets off the end of the equation. If no b is found, the cell number of the comma is recorded and the b-schema scans left testing for r's until it reaches the equal sign. If an r is found, this indicates the presence of an index defined by minimization. Control is sent to the Addressor, as will be explained further below. If no r is encountered, then when the equal sign is reached, control is transferred to the Interpreter. The Interpreter will do the actual translating and produce the machine program.

If, in scanning right, the b-schema finds a b symbol, say b_{mn}, it first tests to see whether b_{mn} has been referred to previously. Let us assume for

the moment that this is the first reference to b_{mn}. In that case, it records $\#b_{mn}$ (the cell number of the cell containing b_{mn}) in standard cell B_o. B_o is the first of a sequence of standard cells which we shall denote by B_i. At any particular instant, the B_i cells contain a list of those b's which have been referred to up to that point without being programmed. When b_{mn} is encountered for the first time, the operation $B_i \rightarrow B_{i+1}$ is performed and the cell number of b_{mn} is recorded in B_o. When this has been done, the cell number of the equal sign of the b_{mn} equation is recorded in standard cell E and control is returned to the first point in the b-schema, that is, the point at which the b-schema starts scanning left from the equal sign. Thus, it begins to process the b_{mn} equation. In this way, the b-schema traces a path through the b-equations until it finds one which contains no b's on the right side. Such an equation is ready for translation to a computer program since all symbols have been replaced by addresses and all functions by suitable information for obtaining the corresponding operation codes. In this case, as mentioned before, control will automatically go to the Interpreter.

When the Interpreter has completed the program for b_{mn}, it assigns a storage address, αb_{mn}, to b_{mn} and records αb_{mn} in the second cell to the left of the equal sign of the b_{mn} equation. (In reading in the formulation, the two cells to the left of each equal sign are left blank.) Any subsequent reference to b_{mn} will be treated differently than the first reference, for now, in testing to see whether b_{mn} has previously been referred to, the b-schema will detect the address recorded to the left of the equal sign and unless branching has taken place, it will replace b_{mn} by this address. If branching has taken place, the b-schema will make appropriate tests to determine whether the computer will have the result for b_{mn} available at this juncture. If not, then b_{mn} will be reprogrammed by suitable branching instructions. For details, see the flow chart and brief explanation of branching below.

If the branching situation is such that the address for b_{mn} can be used to replace b_{mn} directly, the b-schema continues to scan right from b_{mn} for the correct number of indexes. It records these indexes in standard cells and transfers control to the Addressor. The Addressor will process the address and the indexes and return control to the b-schema which will resume its scan to the right for other b symbols.

The preceding explanation holds except when b_{mn} is defined by a recursion equation and the b-schema detects that it is in the midst of this very recursion. In this case, although b_{mn} has not yet been programmed, an initial address has already been assigned by the Recursion-Schema. b_{mn} is then given special treatment. Its associated indexes are obtained and processed somewhat differently by the Addressor, (see Recursion-Schema and Flow Charts).

The q-schema - As explained above, when a quantification symbol has been discovered on the left side of a b-equation, control is transferred to the q-schema. The q-schema will scan right from the quantification symbol to obtain three pieces of information: the index being quantified, the lower bound of this index and the upper bound. It uses this information to "set up"

an index register in the Computer. This index register will be used to modify the addresses of all variables associated with the index in question. If the upper and lower bounds are independent variables, this can be carried out in a straightforward manner. The address of the lower bound is easily obtained and is used to compose an instruction which initializes the index register. The address of the upper bound is used to compose a branching instruction which will close the loop corresponding to this quantification. The precise details of setting up the iterative loop which carries out a quantification will vary from computer to computer. Hence, these details will be omitted.

If a bound in a quantification is an r symbol, then the address is not obtainable directly but will generally cause control to be transferred to the r-schema. The r-schema will program the equation which defines the bound and return control to the q-schema which will then complete the instructions for the loop.

The Addressor - When the q-schema has completed its job of setting up the index register, it transfers control to the Addressor. The main function of the Addressor is to provide one or more addresses for each variable in the equations. The Interpreter will compose instructions by combining these addresses with appropriate operation codes.

After each quantification, the Addressor scans all the equations for independent variables. An independent variable of degree zero is replaced directly by a single address, together with an address-identifying tag. An independent variable of degree one has one index associated with it. The Addressor scans right one and obtains the index. If the index is a q, it tests to see whether the loop for this q has been set up. If not, an error alarm is given, indicating improper order of the quantifications in the formulation. If the loop has been set up, the address of the associated index register replaces q in the formulation. The independent variable itself is replaced by its initial address together with a tag indicating that an index register is to be associated with it. If the index associated with independent variable of degree one is an r, then the Addressor checks the equation which defines this r. If the equation has already been programmed, the address assigned to r will have been recorded next to the equal sign. This address is then substituted for r in the formulation and plays the role of an index register as explained above. If the r equation has not been programmed, control is transferred from the Addressor to the r-schema. The r-schema will program the r equation and return control to the Addressor which will proceed as before.

For variables of degree two, there are two associated indexes, called the row and column index respectively. There is also, in the Computer Table, an r which specifies the matrix structure. The two indexes and the equation which defines the structural r are used to set up an index register for this independent variable. This may require a transfer of control to the r-schema. Again, it is impossible to give all the details and the flow charts should be referred to.

It should be noted that if the index associated with an independent variable is an r, and if the equation which defines this r involves other independent variables which also have r's as indexes, then a complicated interaction between the Addressor and the r-schema results. However, this interaction can be kept track of by using two sets of standard cells, the A_i cells and the R_i cells. The A_i cells operate in a manner similar to the B_i cells and the same is true for the R_i.

A part of the Addressor is also used by the b-schema to supply addresses for the dependent variables. (See b-schema.)

When the Addressor has completed its scan for independent variables after a quantification, it returns control to the b-schema. The b-schema then continues to scan the left side of the b-equation.

The r-Schema - As explained previously, the r-schema can be entered in one of several ways. If an r symbol occurs as an index of an independent or dependent variable, then control is sent to the r-schema from the Addressor. If an r symbol occurs as a bound in a quantification, control is sent to the r-schema from the q-schema. If an r symbol occurs on the left side of a b equation to indicate storing, then the r-schema is entered from the b-schema.

The different entries are recorded by placing certain key information in the standard cells T, T_q, and T_s. Briefly, if (T) is not zero, this indicates that the Addressor is working on a dependent variable. If (T_q) is not zero, this indicates that the r-schema has been entered for a minimization involving a quantifier bound. If (T_s) is zero, this means that the r-schema has been entered from the Addressor to obtain an index address for an independent or dependent variable. If (T_s) is not zero, then, depending on the contents, it means that the r-schema has been entered from the q-schema to obtain a bound of a quantification or it has been entered from the b-schema for storing purposes or finally, it has been entered from the Addressor, but this time to obtain a structural r for a variable of degree 2.

The operation of the r-schema is very much like that of the b-schema with obvious modifications. First, there is no need to process the left side of an r equation. Second, in scanning the right side of the equation, the r-schema scans for other r symbols rather than b symbols, and when such r's are found, they are listed in order in the R_i cells. Further, since the r-schema can be entered in the middle of the operation of the Addressor, the right side of an equation may contain independent variables. In that case, before the r equation can be programmed, the independent variable must be replaced by an address. This requires a return to the Addressor and can set up the complicated interaction between the Addressor and the r-schema mentioned before. This interaction is controlled by the A_i cells in conjunction with the R_i cells.

Another difference is that the r-schema must take into account the possible occurrence of the minimization operator. It does this by a special test which sends control to the minimization subschema (see below) if called for.

Owing to the special nature of equations which define those r's which specify storing, the r-schema operates somewhat differently on these equations.

The Recursion-Schema - It will be recalled that when the b-schema discovers a recursion equation, it sends control the Recursion-schema. This schema will test subscripts of the equal sign to determine what type of recursion is called for. For a simple scalar recursion, it will count the number of initial values and allot sufficient storage for these values. If the recursion is of the course-of-values type, it will consult the Computer Table and obtain the total number of values of the recursion index and allot this many storages. It will then record an initial address to the left of the equal sign and will list in the B_i cells the cell numbers of the commas which set off the formulas for the initial values together with certain tags which identify the beginning and end of the recursion. Thus, when the b-schema resumes operation, it will, by consulting the B_i registers, cause the initial values to be programmed in the correct order. For a vector recursion, the number of equations is determined as well and the above process is duplicated for each equation. When all this information has been recorded, control is returned to the b-schema.

For a double recursion, more bookkeeping is required. The parentheses which set off those equations which involve the two recursion indexes must be located as well as the brackets which set off the entire recursion (see NAVORD 4209). The order in which the equations are to be programmed is the order in which they are written and the B_i registers must be set up accordingly. The type of recursion (i.e., preceding-line, course-of-values, etc.) must be determined for each equation and the correct amount of storage allotted. As in simple recursions, initial addresses must be assigned for each of the dependent variables. Again, control is then returned to the b-schema.

The Interpreter - After all operands in an equation have been replaced by addresses, the Interpreter is entered either from the r-schema or the b-schema. The Interpreter has the function of scanning each formula and translating it into a Computer program. It does this by scanning left for functions, starting at the comma. As it scans, it counts the number of operands, (i.e., the number of addresses and addresses with degree-one tags) between successive functions. It compares this number with the degree of the function. The degree of the function is simply the number of arguments of the function and is obtained either from the library subschema associated with each library function, or in the case of auxiliary functions, the degree is simply the number of free variables in the formula which defines the auxiliary function. On the basis of this comparison, the Interpreter determines whether a "store" instruction is required.

For a library function, control is transferred to a library subschema which composes the proper Computer instructions by combining the addresses with the pertinent operation codes. For auxiliary functions, the free variables are replaced by the addresses and the resulting formula is programmed by the Interpreter. For library functions which are to be computed

by a subroutine, the Interpreter will compile the subroutine and insert it into the program with the correct addresses.

In assigning addresses for storing intermediate results, the Interpreter works with a standard cell called the Intermediate Storage Counter. A block of storages in the Computer is reserved for intermediate results and the counter merely keeps track of which storages are available. Since these storages are erased as soon as the result is used, the block need not be very large. In allotting storages for instructions, the Interpreter uses a standard cell called the Instruction Counter (IC). IC is initially set with the first available Computer storage, starting with the lowest possible address. When the Encoder begins to program an equation, it records the contents of IC in the blank cell to the left of the equal sign. Thus, a record is kept of the address of the first instruction in the program for each equation. This address will be used in setting up any branching instructions which may be required.

When the last function in a formula has been translated, the Encoder assumes that the final result will be in a standard storage called the Result Register in the Computer. To complete the program, the Interpreter must determine whether to store this result, and whether to punch it out or print it. The "end-procedure" is governed by the information contained on the left side of the equation. The Interpreter will scan the left side of the equation for an r which indicates storing and for a d which indicates output of some kind. It will then compose the necessary instructions. The Interpreter must also test for a quantification since if one is present, the end-procedure must compose instructions to close the loop corresponding to this quantification. When all these contingencies have been accounted for, the Interpreter will compose a "No Operation" instruction as a final instruction and record the address of this instruction in the cell which contains the equal sign. This address will be used in certain cases where branching is necessary.

In composing storing instructions for each b and r, the Interpreter allots storage addresses by consulting a standard cell called the β -Counter. The β -Counter is initially set with the highest available storage address and is decreased as storage is allocated. Since the Instruction Counter is increased as addresses are assigned to instructions, if there should be an overlap, an error alarm is given.

In reading the flow charts, it will be noted that there are many special cases to consider. These arise mainly in recursion and minimization. In recursions, the storing end-procedure must be slightly modified. Also, initial values receive special treatment. The same is true of minimizations since these interrupt the natural order of the b-schema. Likewise, branching calls for special procedures since, here again, the order is interrupted (see remarks on branching and minimization below).

Finally, when the end-procedure is completed, the Interpreter will transfer control back to either the r-schema or the b-schema.

Branching Subschema - A branch equation has the following form,

$$y = \phi\, x_1 x_2,\ \psi_1, \psi_2$$

where ψ_1 and ψ_2 denote formulas and ϕ denotes one of the branching functions, $<$, $\leq$ or $=$. Thus, y is equal to ψ_1 if the branch condition ϕ holds between the operands x_1 and x_2, and otherwise y is equal to ψ_2. (Note that this is a slight modification of the format described in NAVORD 4209.)

The branch equation is treated like any other b (or r) equation to begin with. The operands x_1 and x_2 are processed and replaced by addresses and the b-schema scans right from the equal sign to the comma. This causes control to go to the Interpreter. The Interpreter, on finding the function, ϕ, sends control to the appropriate library subschema, which, in turn, transfers control to the Branch Subschema. The Branch Subschema scans right for the comma after x_2 and replaces this by an equal sign with the subscript 18. It then obtains a storage address, τ, from the β-Counter. It replaces the function, ϕ, by τ and produces the program which sets up the branch condition. The "transfer" instruction in this program will have τ as its address.

At this point, the fact that a branch has been set up is recorded in standard cell BI. This can be done in several ways. One method is to record a string of 0's, 1's and 2's in BI, the 1's indicating the left part of a branch and the 2's, the right part of a branch. Thus, whenever a branch condition is set up, a digit one will be added to the string. When the corresponding right branch is started, this 1 is changed to a 2. When both parts of the branch have been completed, the 2 is replaced by a zero and the closing of the branch is tallied in another standard cell. The string in BI together with the tally is a sufficient record of the branching structure of the flow chart.

After programming the branch condition, the Branch Subschema returns control to the b-schema (or r-schema) which resumes by scanning the formula, ψ_1, and sending control to the Interpreter which then programs ψ_1. The end-procedure, on finding "$=_{18}$", scans left to the main equal sign. It then composes the storing, output and loop-closing instructions as indicated on the left side of this equal sign. The address, γ, of thefirst instruction composed by the end-procedure is then recorded in place of the symbol, $=_{18}$. A final instruction which transfers control to a storage, τ_1, is then composed. This will have the effect of causing the Computer to jump over the instructions for the right part of the branch in the event that it has taken the left part of the branch, since the instructions for the right part will immediately follow. At this point, another "transfer instruction" is composed and placed in storage τ. This will be acted upon by the Computer when it takes the right branch and will cause it to jump over the instructions in the left branch, except those belonging to the end-procedure.

The formula ψ_2 is programmed and a suitable "transfer" instruction is added to cause control to jump to γ. Finally, another "transfer" instruction is composed and placed in storage τ_1. This completes the branch program.

Arranging the "jumps" in the manner described, has the advantage that it permits the instructions of each branch to be composed and punched out directly with a minimum of bookkeeping.

Minimization Subschema - The minimization operation can occur in two contexts: as a definition of an upper bound in a quantification and in a "table look-up" operation. When an upper bound is defined by a minimization, the independent index involved is the same as the independent index in the quantification. Recalling that the format of a minimization equation is as follows,

$$r = f_{\mu} q \; a \; b,$$

the b is seen to be one of the quantities in the scope of the quantification of q. The minimization subschema replaces the upper bound by b. When the end-procedure of the Interpreter encounters this b on the left side of the equation, it will program it, assign a storage address to it and use this address to compose the required instructions for closing the loop. Note that this programming will interrupt the normal operation of the b-schema. Appropriate records have to be kept of the fact that a minimization is in progress.

If a minimization involves a table look-up operation, then the b in the minimization formula is treated simply as if it were on the right side of a b-equation. For more details, see pages Min 1, Min 2 after the flow charts.

IV - Flow Charts for ADES II Encoder

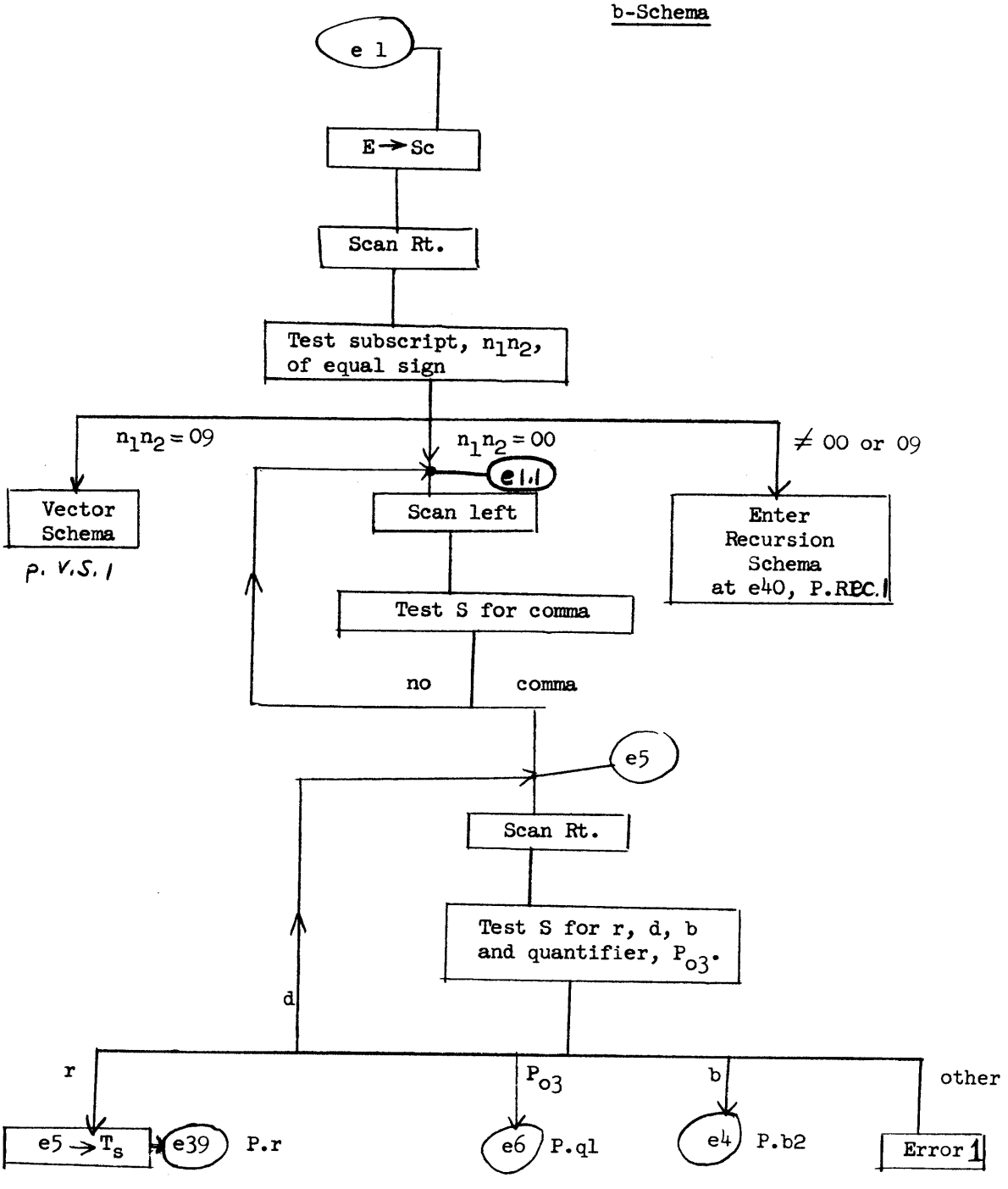
The following pages contain flow charts of the six principal schema described in section III above. These charts are intended to show in some detail how the designs sketched briefly in section III can be carried out in terms of simple operations. However, the main object is to show how the various schemata interlock. Hence, the degree of detail varies. It is believed that too much detail would obscure the main design.

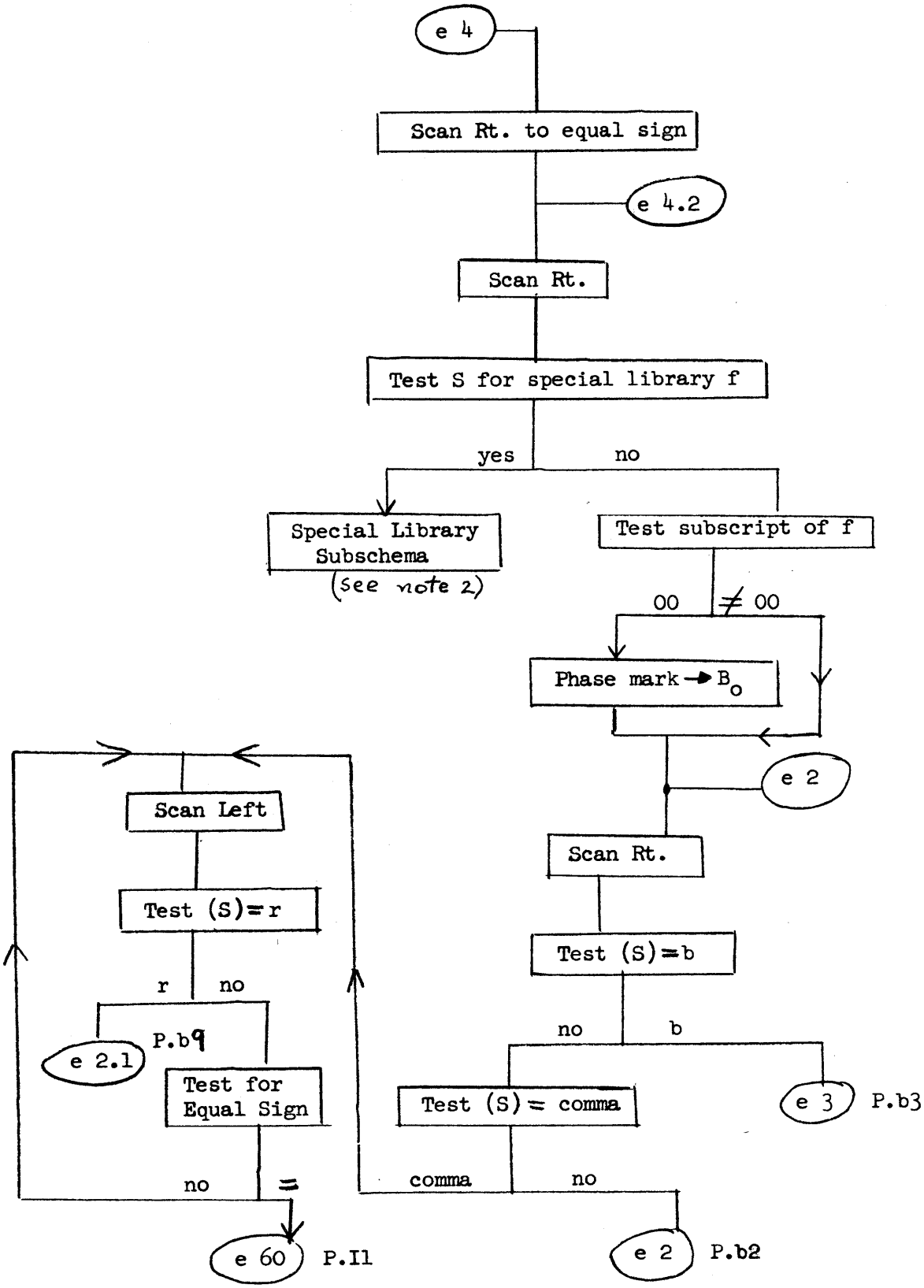
The b-schema is charted on pages b1 to b17; the q-schema on pages q1 to q8; the Addressor on pages 1A to 6A; the r-schema on pages r1 to r5; the Recursion-schema on pages rec.1 to rec.9; the Interpreter on pages I1 to I20. The Minimization Schema is described briefly on pages Min 1 and Min 2, and the Vector schema on pages V.S.1 and V.S.2.

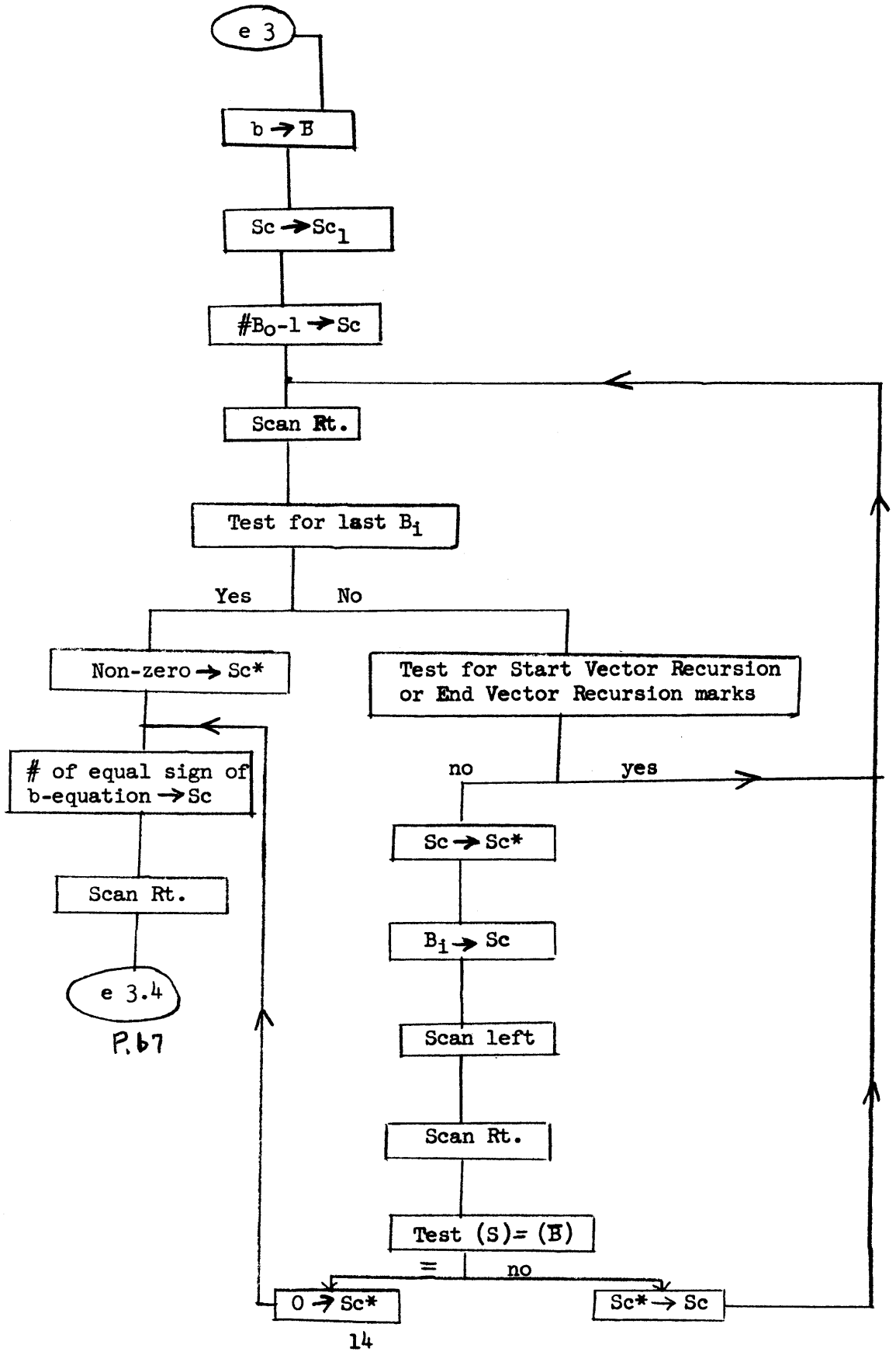
In all charts, the direction of flow is downward unless otherwise indicated by arrows. To continue from page to page, the familiar device of tagging various points by decimal numbers enclosed in a circle is used. For example, on page b1 one of the exits is labeled as "(e4), b2". Turning to page b2, one locates (e4) at the top of the page. The letter, "e", is used to denote "Encoder" and precedes all numbers used as flow chart locations.

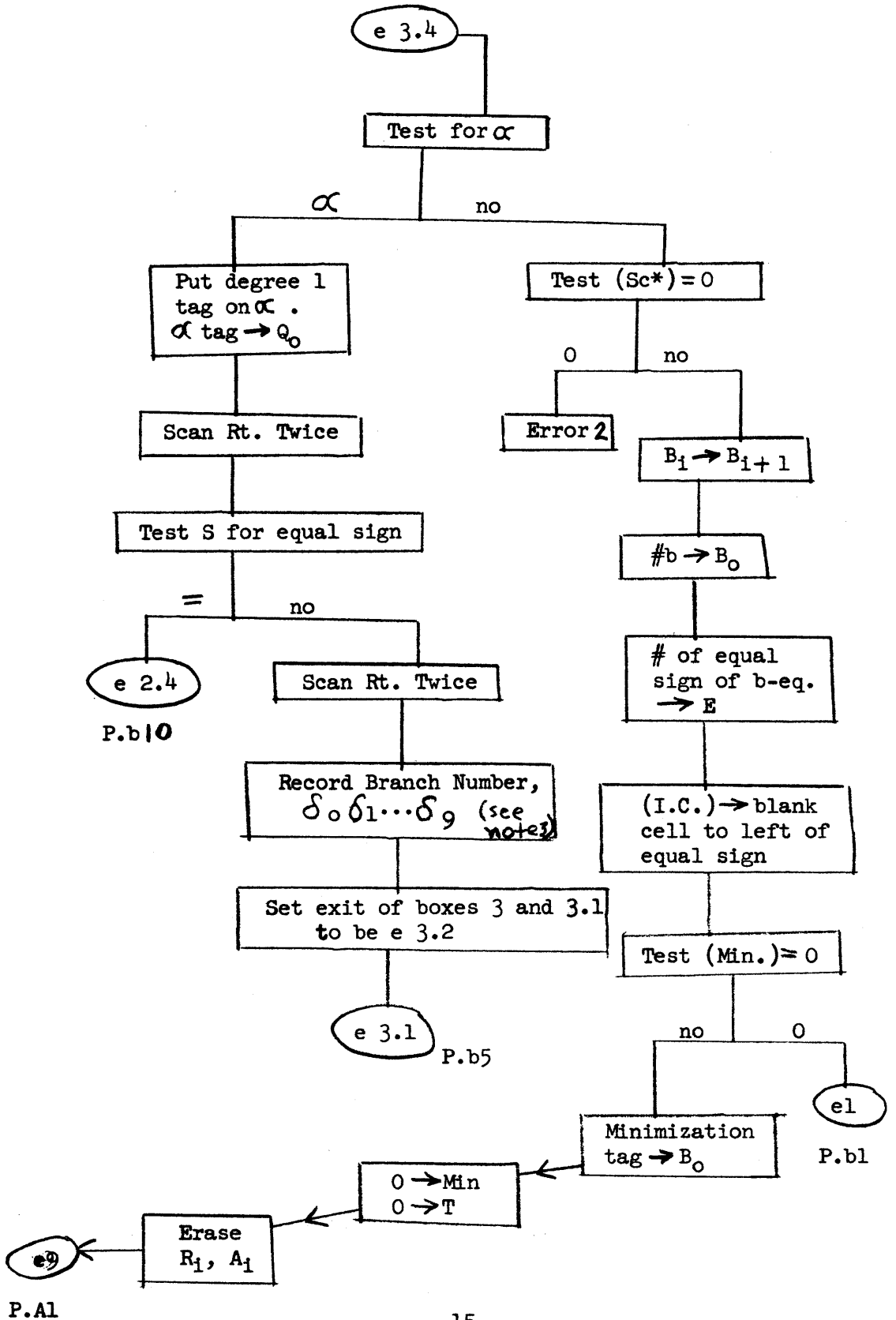
In some cases, it seemed desirable to add explanatory notes. These are collected after the flow charts for each schema. A list of the automatic error checks is given at the end of the flow charts.

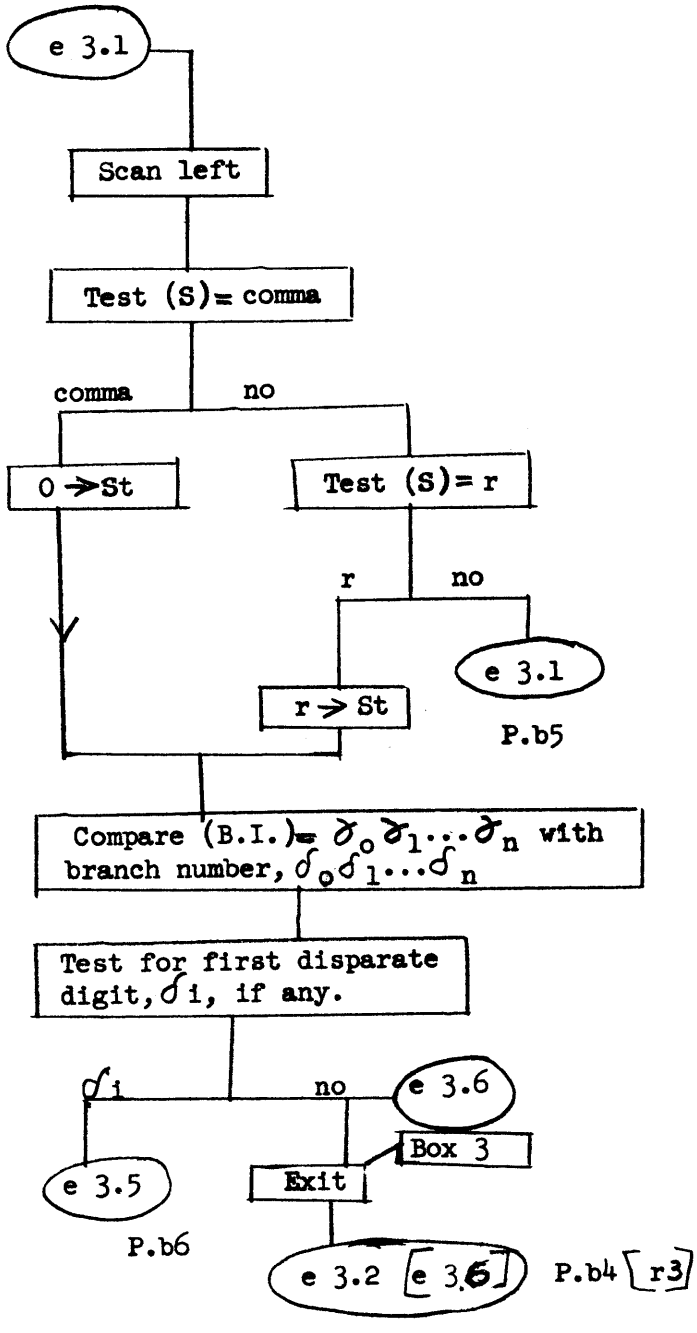
b-Schema

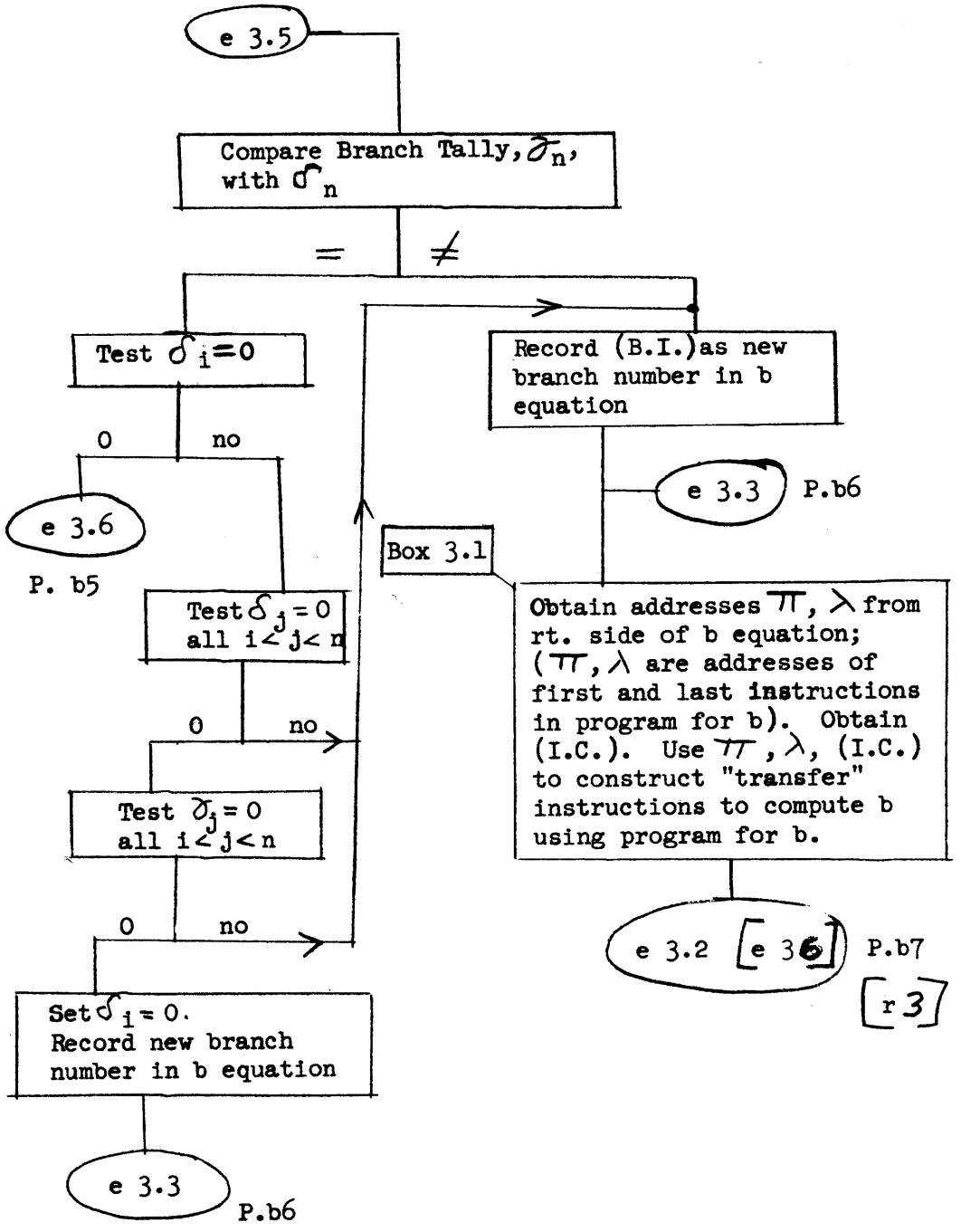

e 1
E → Sc
Scan Rt.
Test subscript, n_1n_2, of equal sign
$n_1n_2 = 09$
$n_1n_2 = 00$
≠ 00 or 09
e1.1
Vector Schema
P. V.S. 1
Scan left
Enter Recursion Schema at e40, P.RPC.1
Test S for comma
no
comma
e5
Scan Rt.
Test S for r, d, b and quantifier, P_{03}.
d
r
P_{03}
b
other
e5 → T_s
e39
P.r
e6
P.q1
e4
P.b2
Error 1

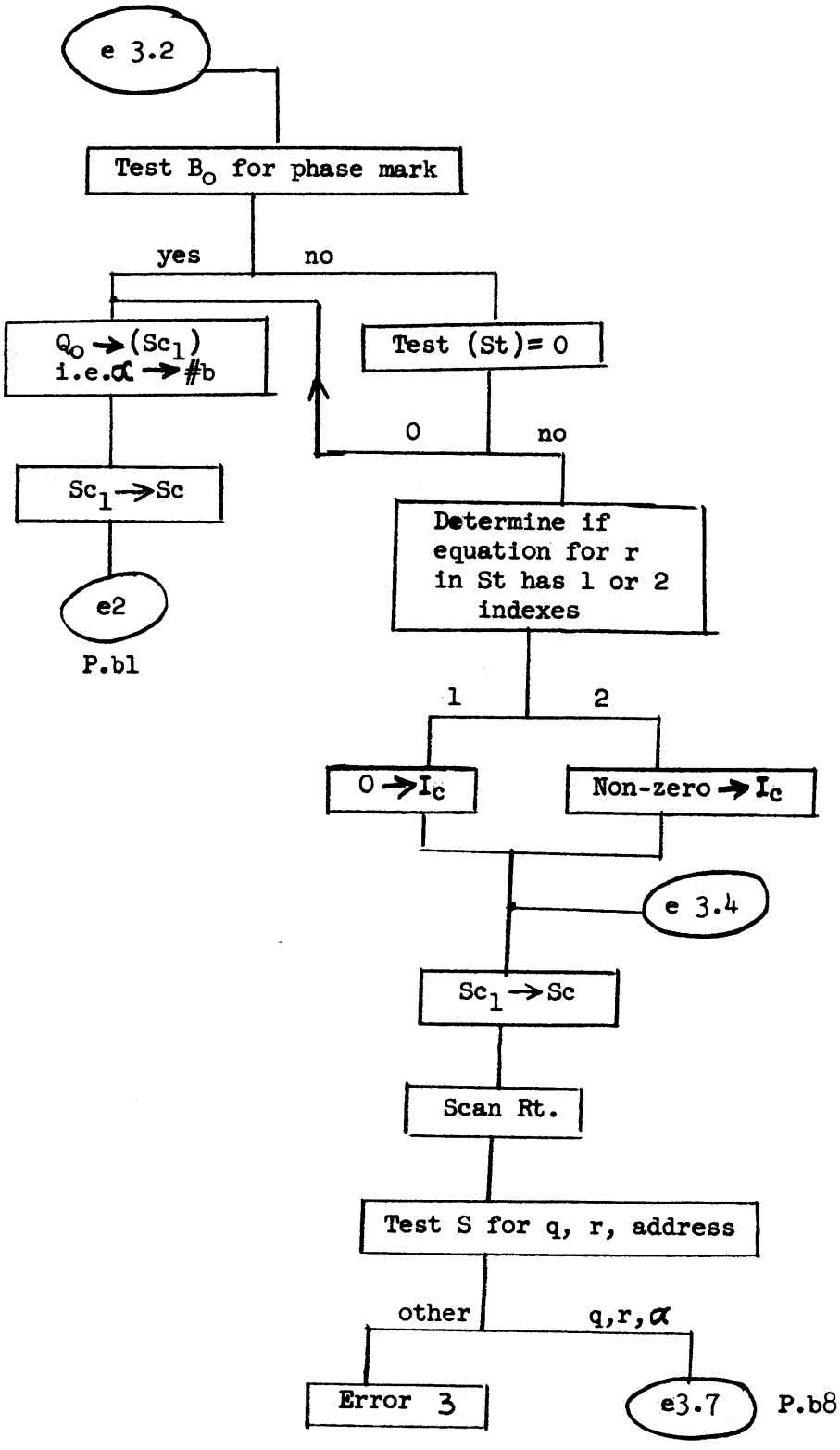

e 4
Scan Rt. to equal sign
e 4.2
Scan Rt.
Test S for special library f
yes
no
Special Library Subschema
(see note 2)
Test subscript of f
00
≠ 00
Phase mark → B0
e 2
Scan Left
Scan Rt.
Test (S)=r
Test (S)=b
r
no
no
b
P.b9
e 2.1
Test for Equal Sign
Test (S) = comma
e 3
P.b3
no
=
comma
no
e 60
P.I1
e 2
P.b2

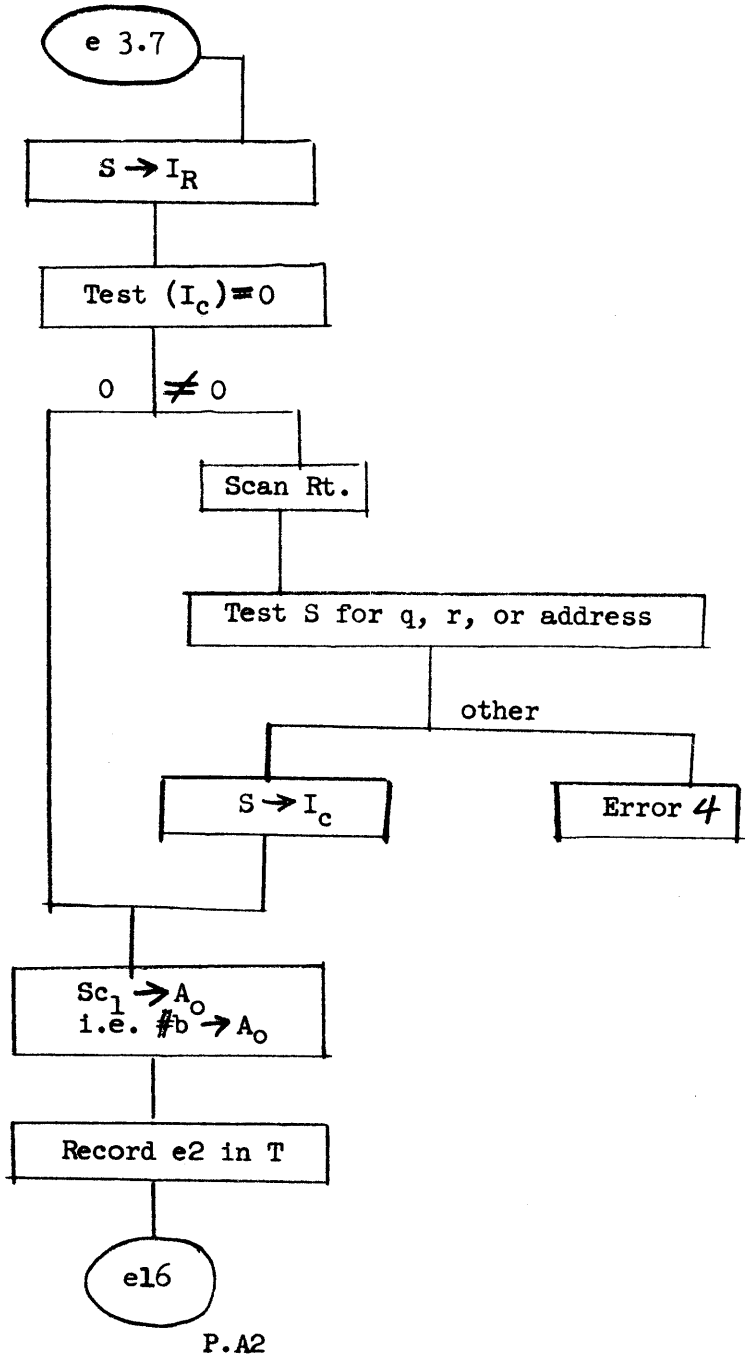

e 3
b → B
Sc → Sc1
#Bo-1 → Sc
Scan Rt.
Test for last Bi
Yes
No
Non-zero → Sc*
# of equal sign of b-equation → Sc
Scan Rt.
e 3.4
P.67
Test for Start Vector Recursion or End Vector Recursion marks
no
yes
Sc → Sc*
Bi → Sc
Scan left
Scan Rt.
Test (S)= (B)
=
no
0 → Sc*
Sc* → Sc

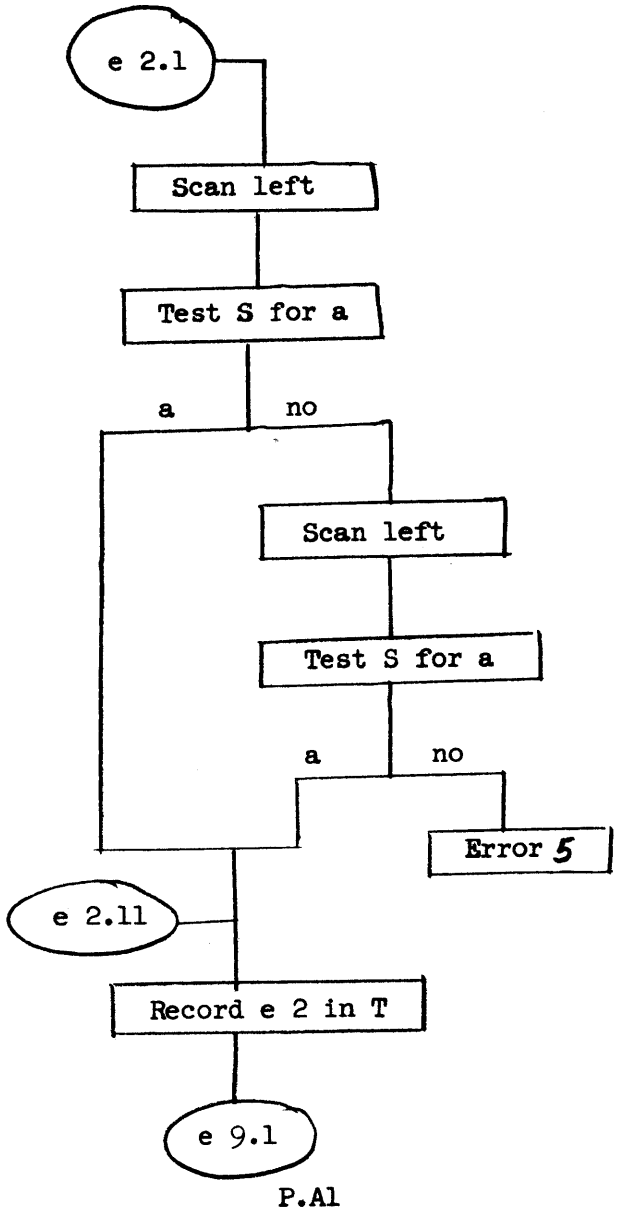

e 3.4
Test for α
α
no
Put degree 1 tag on α . α tag → Q_o
Test (Sc*) = 0
0
no
Error 2
$B_i \rightarrow B_{i+1}$
#b → B_o
Scan Rt. Twice
Test S for equal sign
=
no
e 2.4
P.b10
Scan Rt. Twice
# of equal sign of b-eq. → E
Record Branch Number, $\delta_o \delta_1 \cdots \delta_9$ (see note3)
(I.C.) → blank cell to left of equal sign
Set exit of boxes 3 and 3.1 to be e 3.2
Test (Min.) ≥ 0
e 3.1
P.b5
no
0
e1
P.b1
Minimization tag → B_o
0 → Min
0 → T
Erase R_i, A_i
e9
P.A1


e 3.1
Scan left
Test (S)= comma
comma
no
0 → St
Test (S)= r
r
no
e 3.1
P.b5
r → St
Compare (B.I.)= $\gamma_0\gamma_1\ldots\gamma_n$ with branch number, $\delta_0\delta_1\ldots\delta_n$
Test for first disparate digit, δi, if any.
δi
no
e 3.6
Box 3
e 3.5
Exit
P.b6
e 3.2 [e 3.6]
P.b4 [r3]

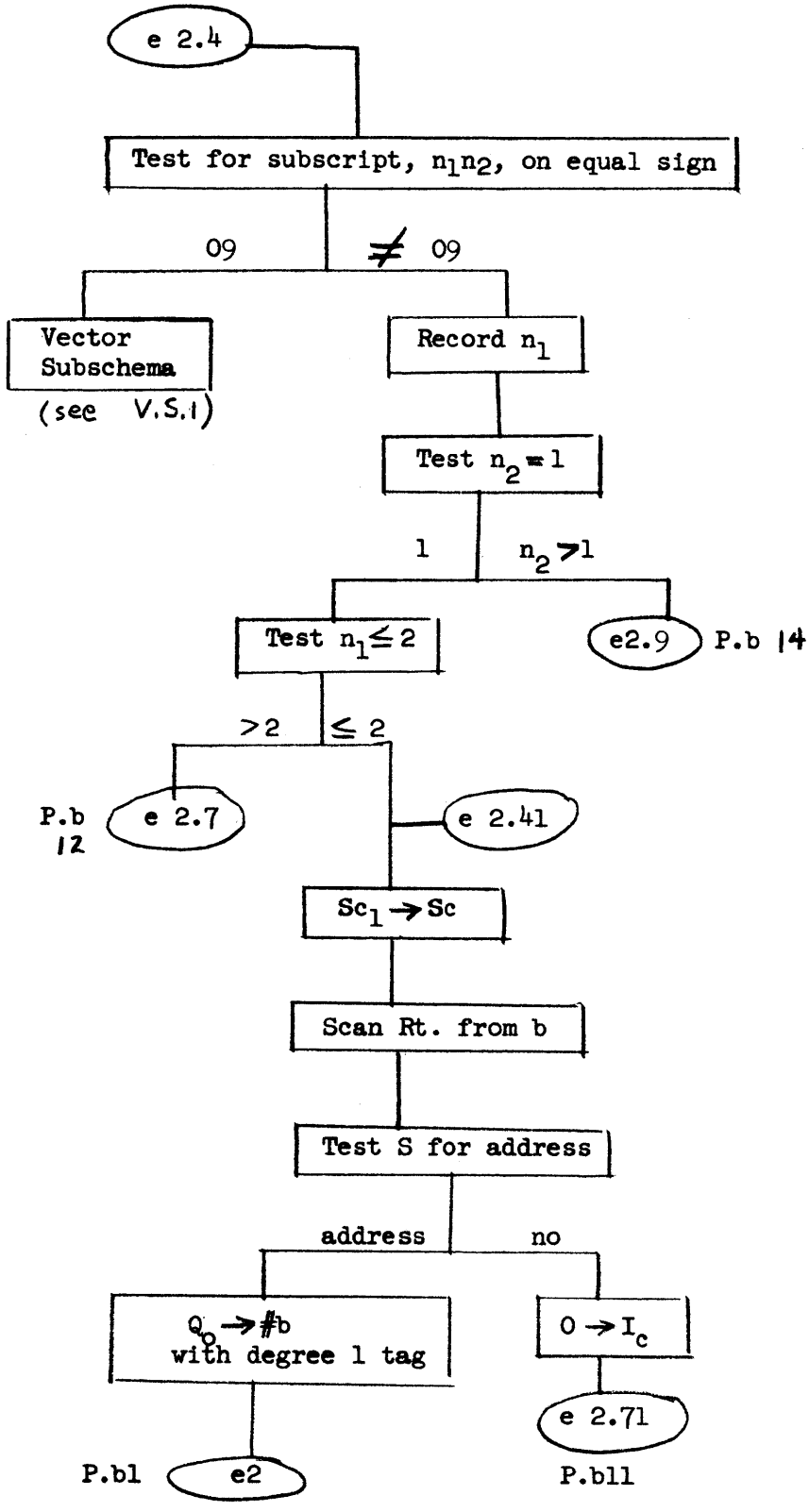

e 3.5
Compare Branch Tally, δ_n, with σ_n
=
≠
Test $\sigma_i = 0$
0
no
e 3.6
P. b5
Test $\delta_j = 0$ all $i < j < n$
0
no
Test $\sigma_j = 0$ all $i < j < n$
0
no
Set $\sigma_i = 0$. Record new branch number in b equation
e 3.3
P.b6
Box 3.1
Record (B.I.) as new branch number in b equation
e 3.3
P.b6
Obtain addresses π, λ from rt. side of b equation; (π, λ are addresses of first and last instructions in program for b). Obtain (I.C.). Use π, λ, (I.C.) to construct "transfer" instructions to compute b using program for b.
e 3.2 [e 36]
P.b7
[r 3]

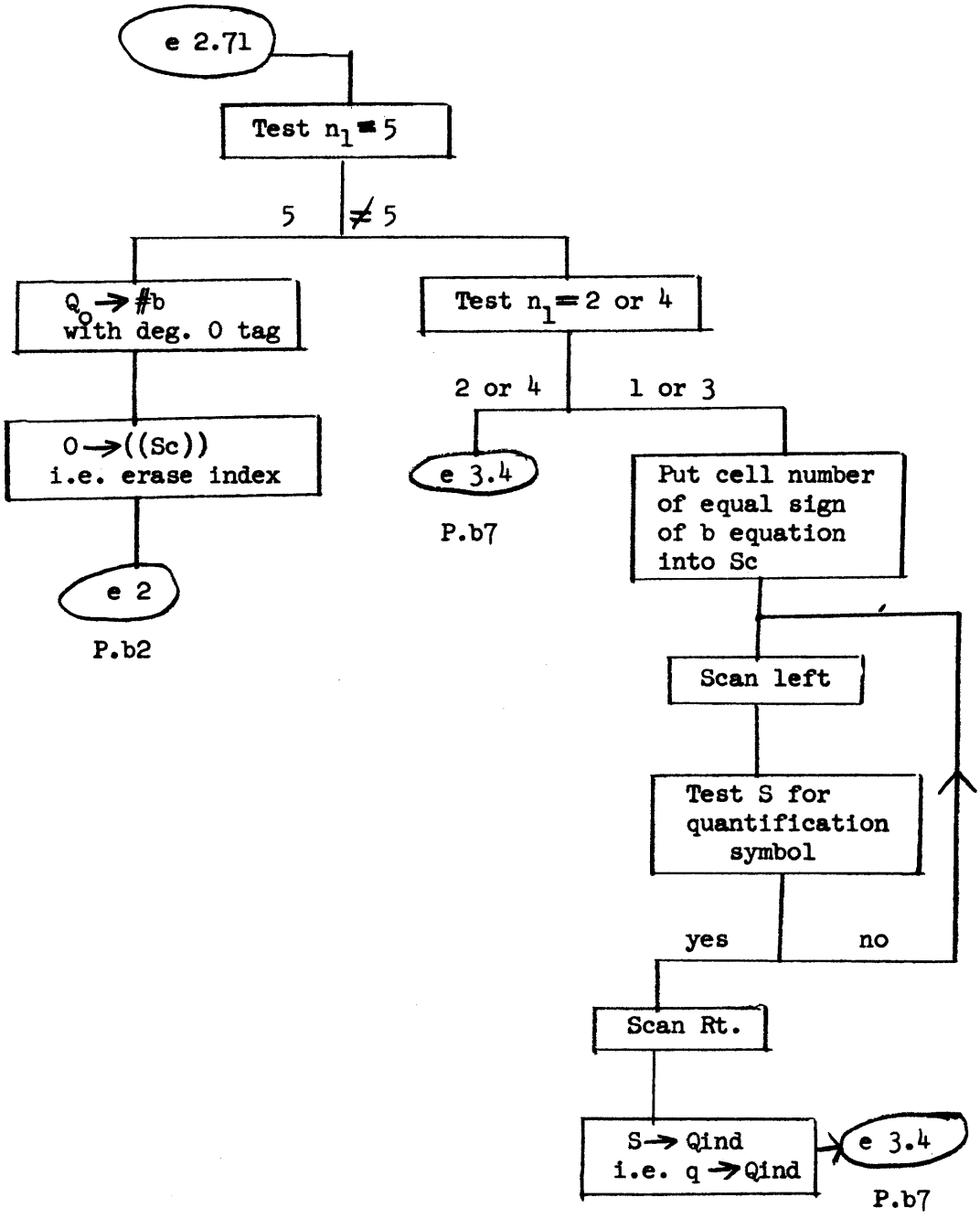

e 3.2
Test B_o for phase mark
yes
no
$Q_o \rightarrow (Sc_1)$
i.e. $\alpha \rightarrow \#b$
Test (St) = 0
0
no
$Sc_1 \rightarrow Sc$
e2
P.b1
Determine if equation for r in St has 1 or 2 indexes
1
2
$0 \rightarrow I_c$
Non-zero $\rightarrow I_c$
e 3.4
$Sc_1 \rightarrow Sc$
Scan Rt.
Test S for q, r, address
other
q,r,α
Error 3
e3.7
P.b8

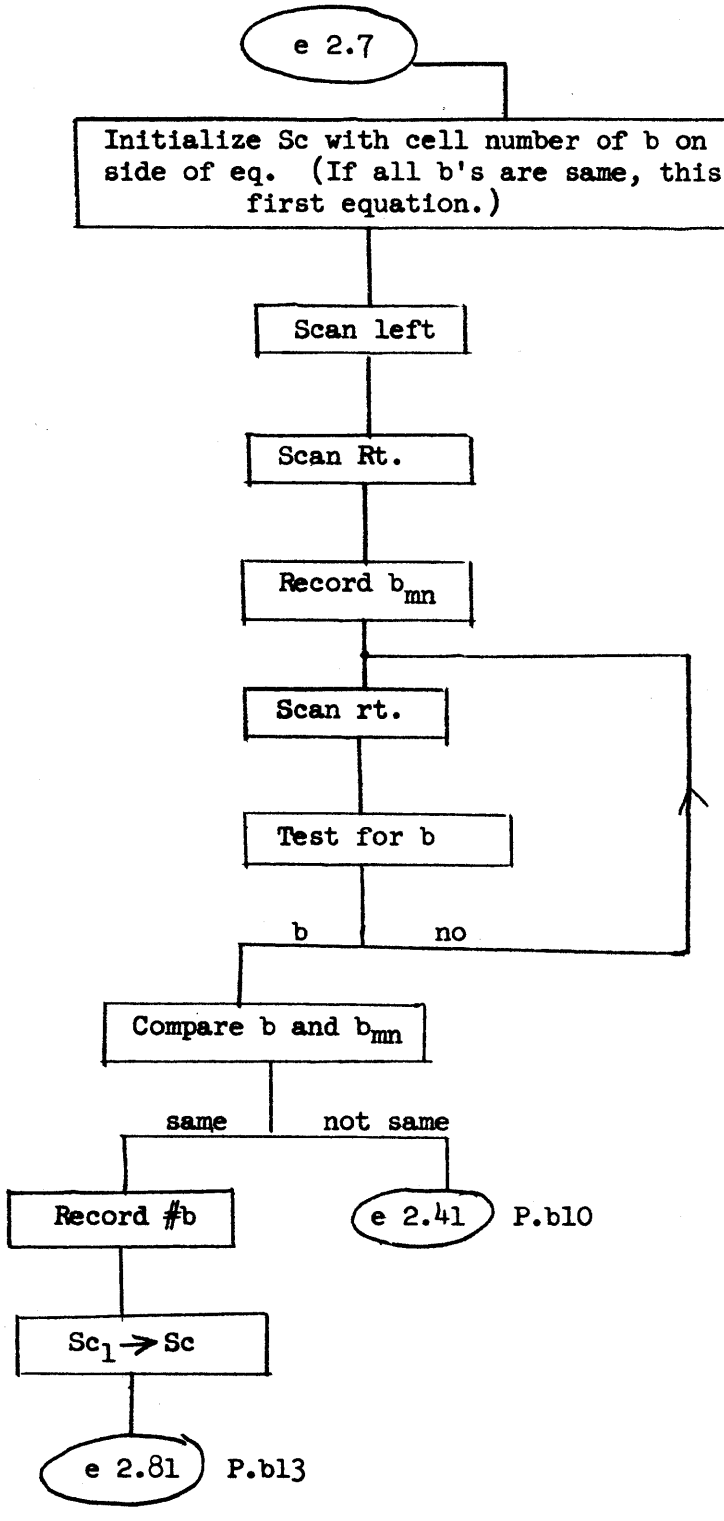

e 3.7
S → I_R
Test (I_c) = 0
0 ≠ 0
Scan Rt.
Test S for q, r, or address
other
S → I_c
Error 4
Sc_1 → A_o
i.e. #b → A_o
Record e2 in T
e16
P.A2

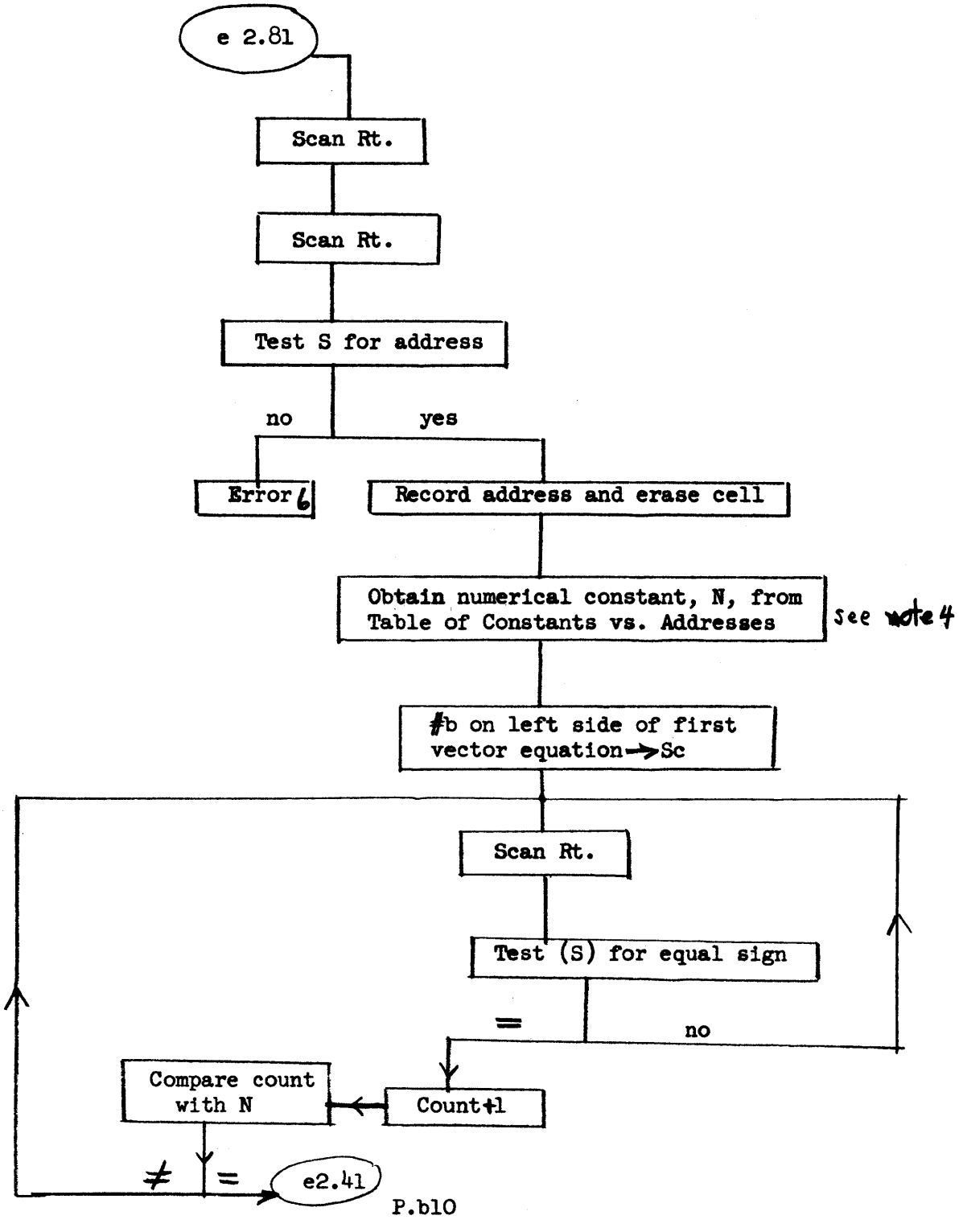

e 2.1
Scan left
Test S for a
a
no
Scan left
Test S for a
a
no
Error 5
e 2.11
Record e 2 in T
e 9.1

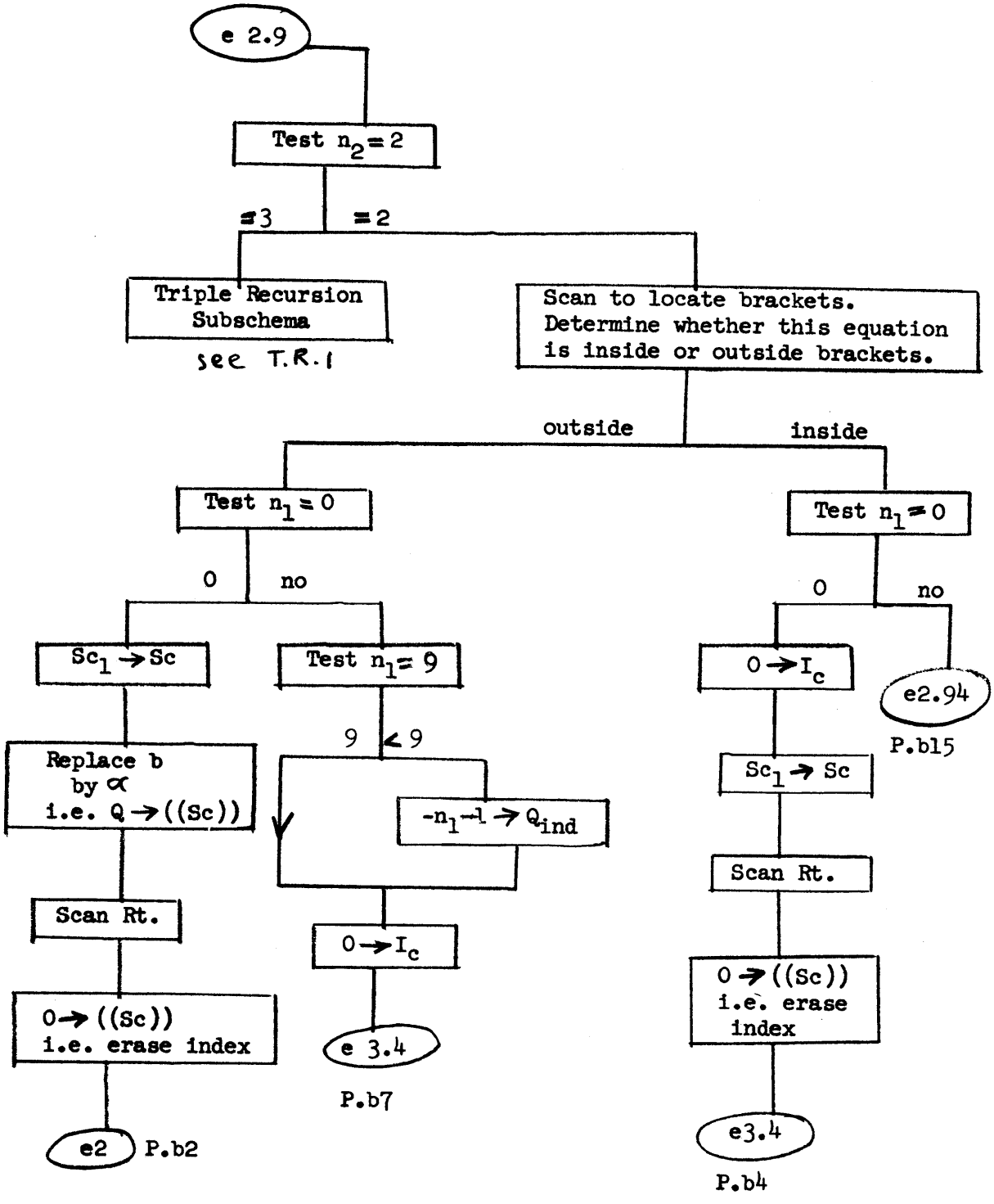
P.A1

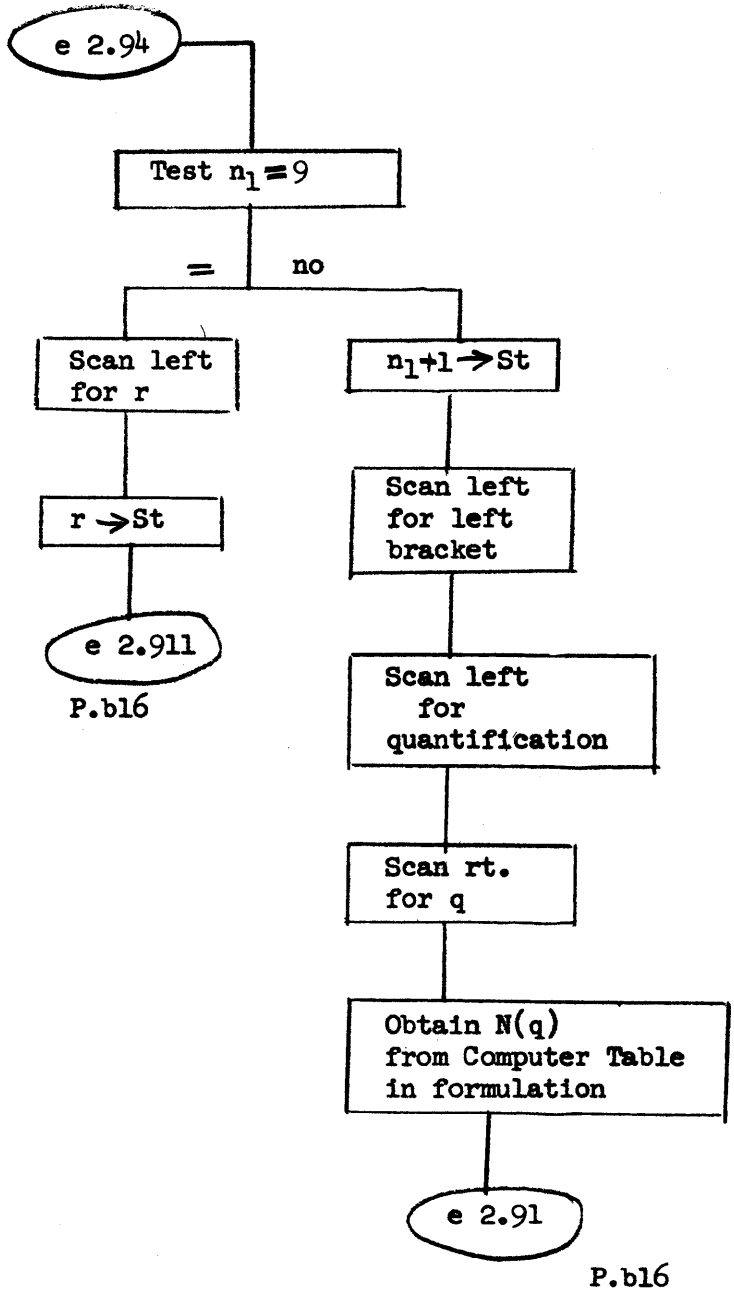

e 2.4
Test for subscript, $n_1 n_2$, on equal sign
09
≠ 09
Vector Subschema
(see V.S.1)
Record n_1
Test $n_2 = 1$
1
$n_2 > 1$
Test $n_1 \leq 2$
e2.9
P.b 14
>2
≤ 2
P.b 12
e 2.7
e 2.41
$Sc_1 \rightarrow Sc$
Scan Rt. from b
Test S for address
address
no
$Q_o \rightarrow$ #b with degree 1 tag
$0 \rightarrow I_c$
e 2.71
P.b1
e2
P.b11

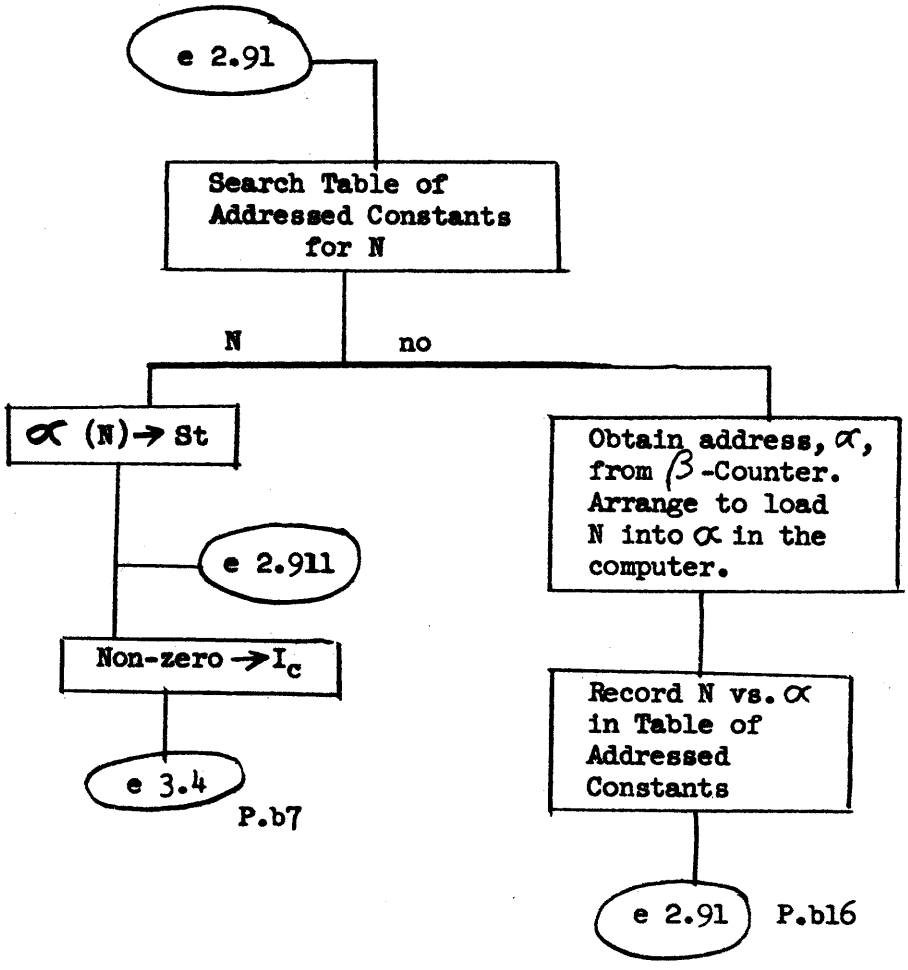

e 2.71
Test $n_1 = 5$
5
$\neq 5$
$Q_o \rightarrow$ #b with deg. 0 tag
$0 \rightarrow ((Sc))$ i.e. erase index
e 2
P.b2
Test $n_1 = 2$ or 4
2 or 4
1 or 3
e 3.4
P.b7
Put cell number of equal sign of b equation into Sc
Scan left
Test S for quantification symbol
yes
no
Scan Rt.
S → Qind i.e. q → Qind
e 3.4
P.b7

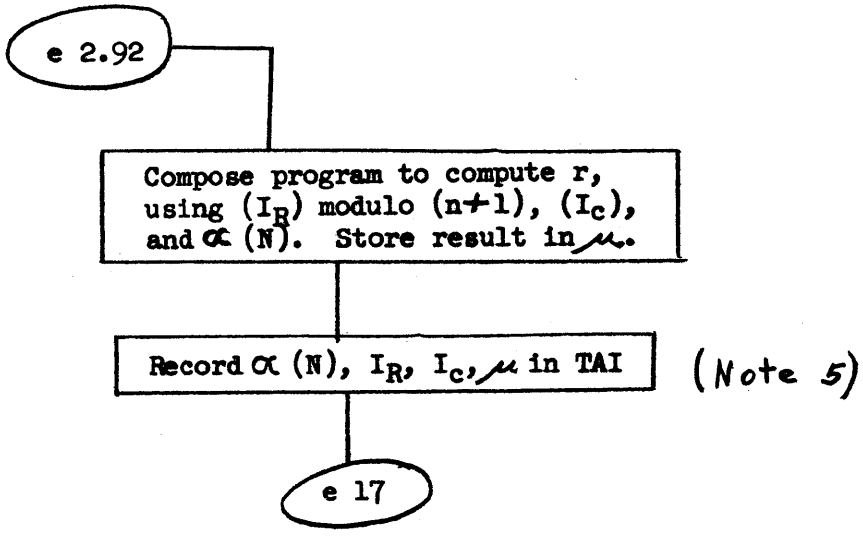

e 2.7
Initialize Sc with cell number of b on left side of eq. (If all b's are same, this is first equation.)
Scan left
Scan Rt.
Record b_{mn}
Scan rt.
Test for b
b
no
Compare b and b_{mn}
same
not same
Record #b
e 2.41
P.b10
$Sc_1 \rightarrow Sc$
e 2.81
P.b13


e 2.81
Scan Rt.
Scan Rt.
Test S for address
no
yes
Error 6
Record address and erase cell
Obtain numerical constant, N, from Table of Constants vs. Addresses
See note 4
#b on left side of first vector equation→Sc
Scan Rt.
Test (S) for equal sign
=
no
Compare count with N
Count+1
≠
=
e2.41
P.b10


e 2.9
Test $n_2 = 2$
=3
=2
Triple Recursion Subschema
see T.R.1
Scan to locate brackets. Determine whether this equation is inside or outside brackets.
outside
inside
Test $n_1 = 0$
Test $n_1 \neq 0$
0
no
0
no
$Sc_1 \rightarrow Sc$
Test $n_1 = 9$
$0 \rightarrow I_c$
e2.94
P.b15
9
< 9
Replace b by α i.e. $Q \rightarrow ((Sc))$
$-n_1 - 1 \rightarrow Q_{ind}$
$Sc_1 \rightarrow Sc$
Scan Rt.
Scan Rt.
$0 \rightarrow I_c$
$0 \rightarrow ((Sc))$ i.e. erase index
e 3.4
P.b7
$0 \rightarrow ((Sc))$ i.e. erase index
e2
P.b2
e3.4
P.b4


e 2.94
Test $n_1 = 9$
=
no
Scan left for r
r → St
e 2.911
P.b16
$n_1+1 \rightarrow St$
Scan left for left bracket
Scan left for quantification
Scan rt. for q
Obtain N(q) from Computer Table in formulation
e 2.91
P.b16


e 2.91
Search Table of Addressed Constants for N
N
no
α (N) → St
e 2.911
Non-zero → I_c
e 3.4
P.b7
Obtain address, α, from β-Counter. Arrange to load N into α in the computer.
Record N vs. α in Table of Addressed Constants
e 2.91
P.b16


e 2.92
Compose program to compute r, using (I_R) modulo (n+1), (I_C), and α (N). Store result in μ.
Record α (N), I_R, I_C, μ in TAI
(Note 5)
e 17

P.5A

Explanatory Notes for b-Schema

1) The following standard registers are used:

Register	Contents
E ...	cell number of b symbol on left side of equation.
C ...	cell number of comma which marks right end of each term.
T_s...	control information; $(T_s) = 0$ if the r-schema is entered to obtain an index for a variable. When the r-equation has been programmed, the Interpreter returns control to el1 (P.I.10). (T_s) = el7 if the r-schema is entered to obtain an index specifying the storage structure of a variable of degree 2. In this case, the Interpreter returns control to el7. (T_s) = e5 if the r-schema is entered for an index specifying that a result is to be stored. The Interpreter will return control to e5. (T_s) = e6.1 if the r-schema is entered to obtain the bound in a quantification. The Interpreter will exit to e6.1.
T ...	If $(T) \neq 0$, the Addressor will send control back to e2. Otherwise, it will resume scanning for a's.
T_q...	$(T_q) \neq 0$ if the r-schema is entered for a table look-up operation.
Sc...	cell number of cell being scanned.
Sc*..	temporary record of contents of Sc; also other information.
Sc_1..	temporary record of contents of Sc.
B_i...	$(i=0,1,\ldots,50.)$ Cell numbers of b-symbols; also control information.

2) On page b2, if a special library function is found, control is sent to the Special Library Subschema. This subschema is not included in ADES II, since it depends to some extent on how library subroutines are treated. Its purpose is to handle special library subroutines of the type mentioned in Appendix I of NAVORD 4209. For example, in a numerical integration subroutine, it is necessary to compute values of a function at points determined by the subroutine. The formula for thefunction must therefore refer to the subroutine. The Special Library Subschema must compile this subroutine into the program and assign addresses to those b-symbols computed by the subroutine. The formula for the function is then programmed by using these addresses.

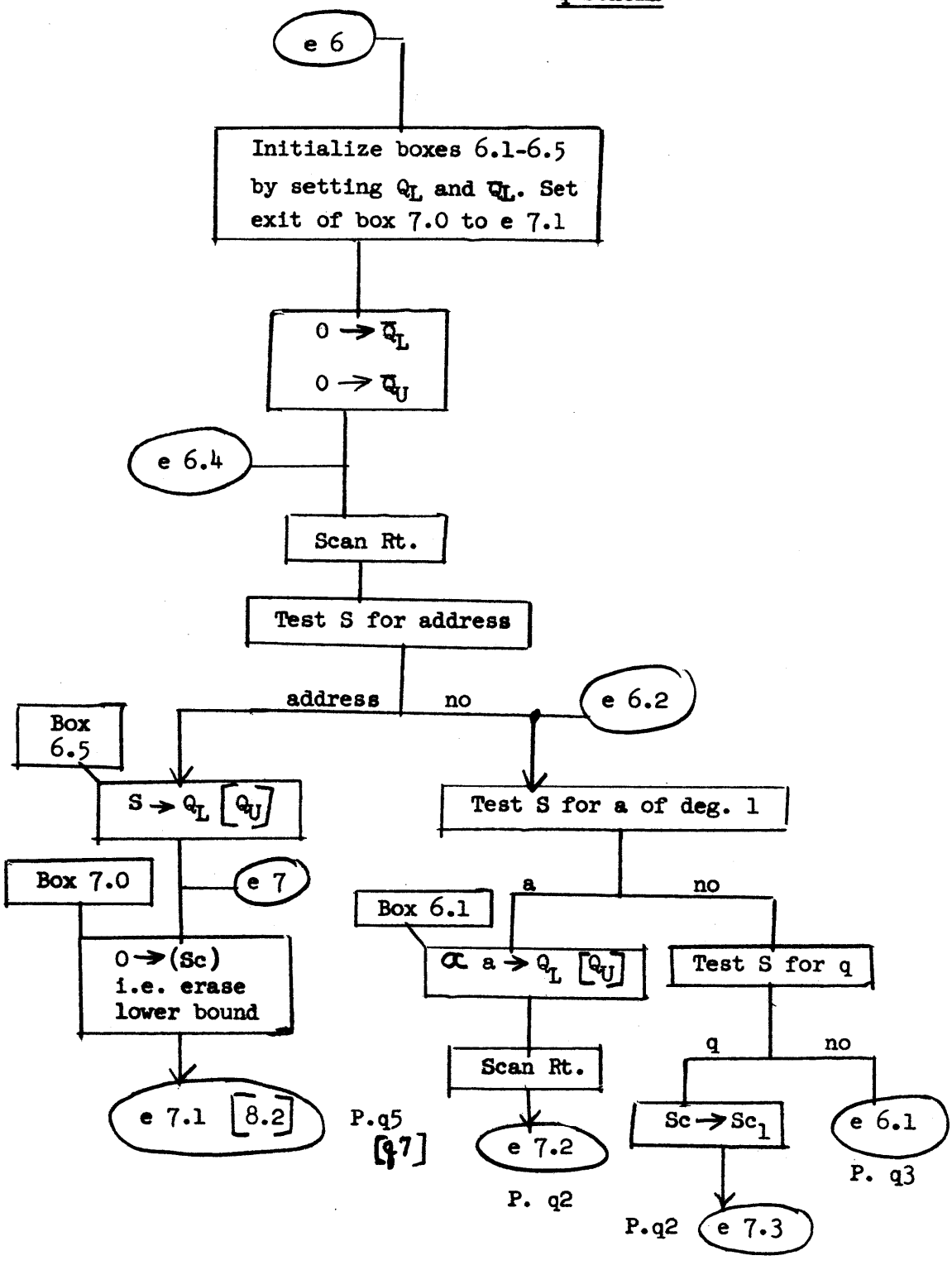
This technique is workable but a better and entirely new method of handling subroutines is presented in the Appendix of this report.

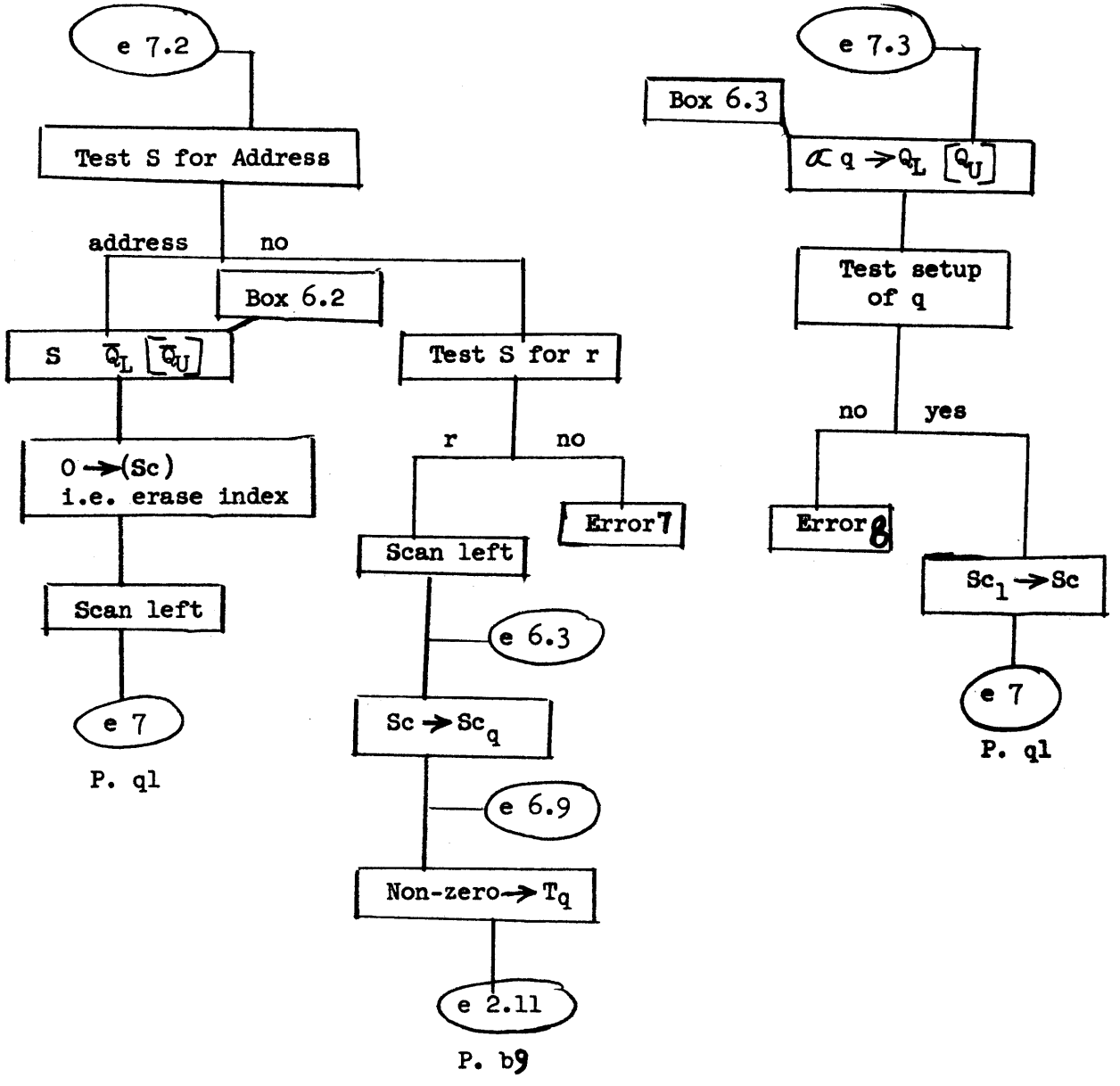
3) As explained in section III, a record of the branching structure is kept in cell B.I. After each equation is programmed, the contents of B.I. are recorded in the two cells to the right of the equal sign. This number is called the "branch number" of the equation and is denoted by δ_0 $\delta_1 \cdots$ δ_9.

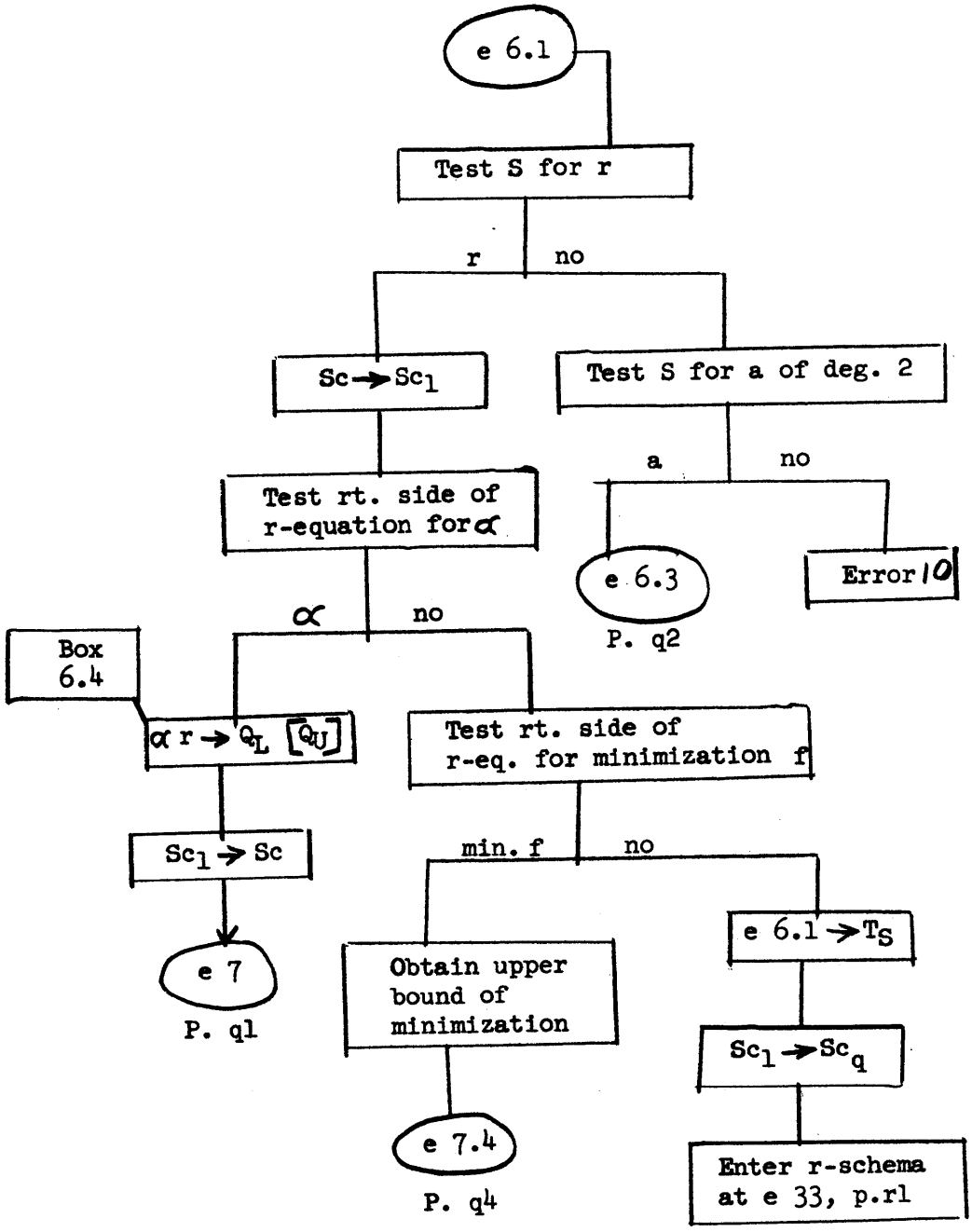
4) The Initial Addressor compiles a table of all numerical constants in the formulation vs. the addresses assigned to these constants. It replaces each constant in the formulation by its address. On page b13, the actual value of the constant, N, must be used in a computation by the Encoder. N is obtained from the Table of Constants.

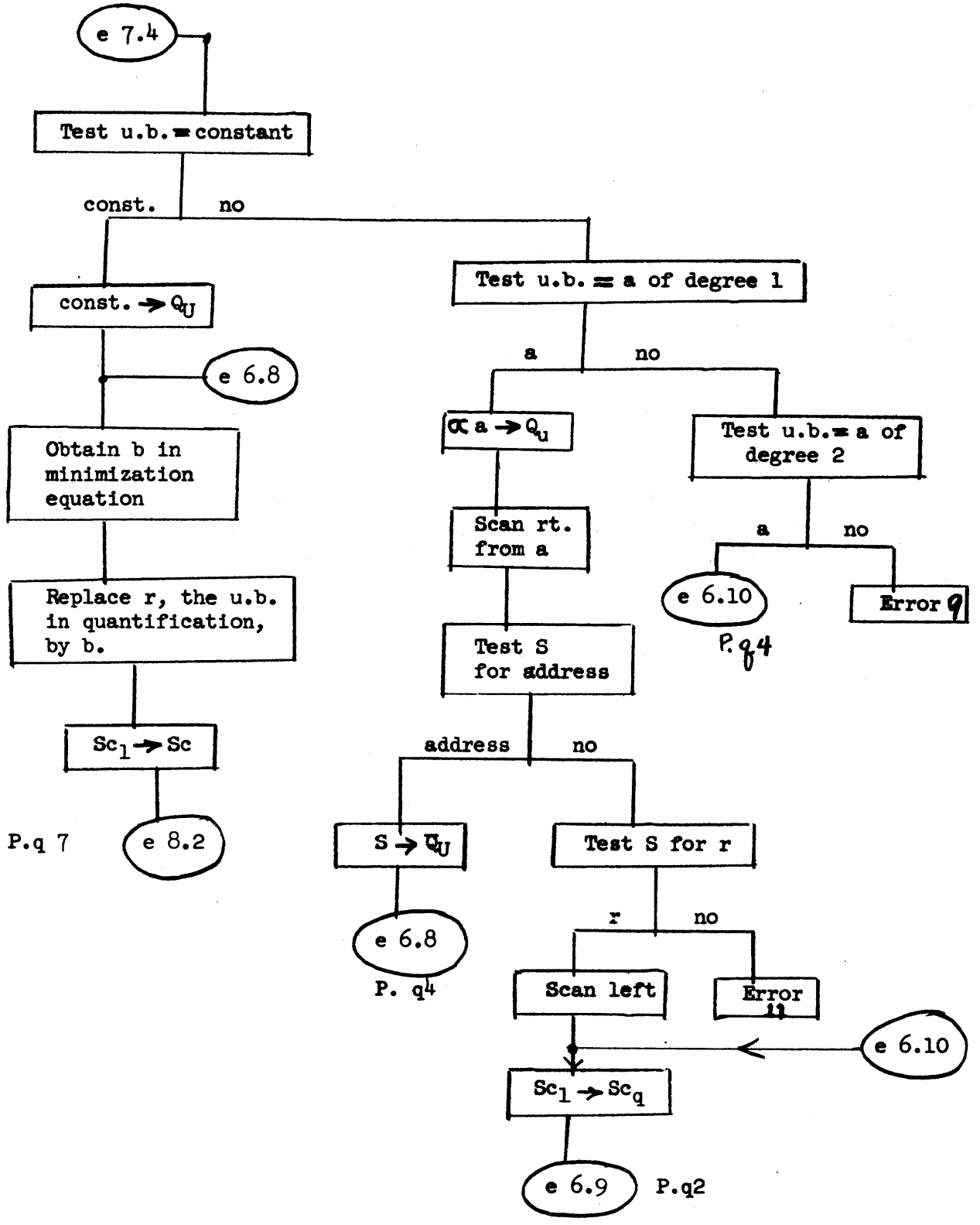
5) On page b17, information about a structural r is stored in TAI. "TAI" stands for "table of addressed indices". TAI is a table which contains a record of the address assigned to an r which specifies matrix structure or storing.

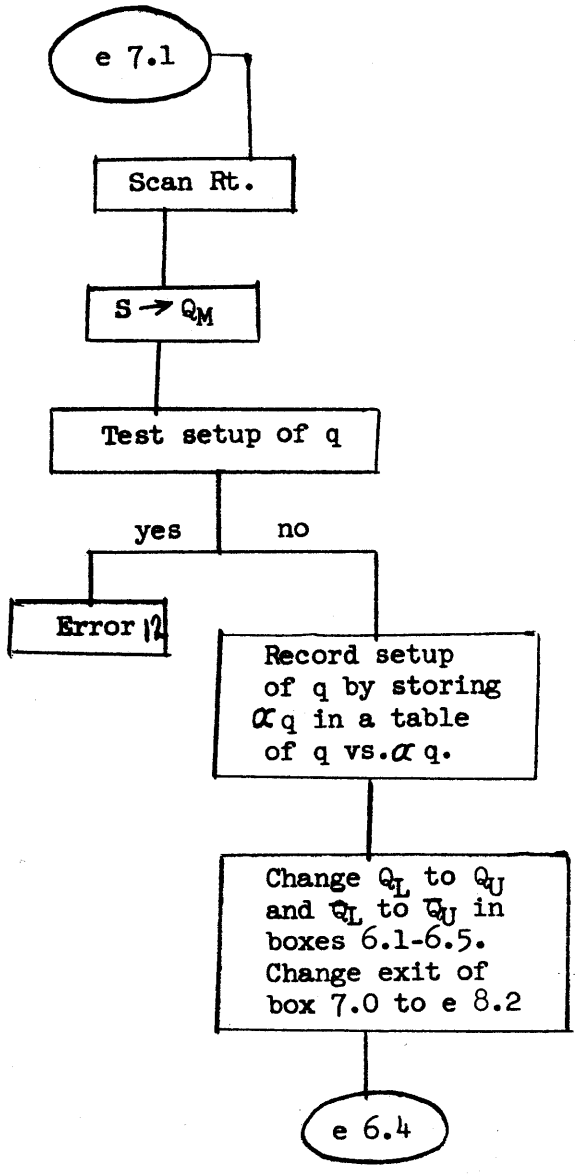
q-Schema


e 6
Initialize boxes 6.1-6.5 by setting Q_L and $\overline{Q}_L$. Set exit of box 7.0 to e 7.1
$0 \rightarrow \overline{Q}_L$
$0 \rightarrow \overline{Q}_U$
e 6.4
Scan Rt.
Test S for address
address
no
e 6.2
Box 6.5
$S \rightarrow Q_L$ [Q_U]
Box 7.0
e 7
$0 \rightarrow$ (Sc) i.e. erase lower bound
e 7.1 [8.2]
P.q5 [q7]
Test S for a of deg. 1
a
no
Box 6.1
α a $\rightarrow Q_L$ [Q_U]
Scan Rt.
e 7.2
P. q2
Test S for q
q
no
$Sc \rightarrow Sc_1$
e 6.1
P. q3
P.q2
e 7.3

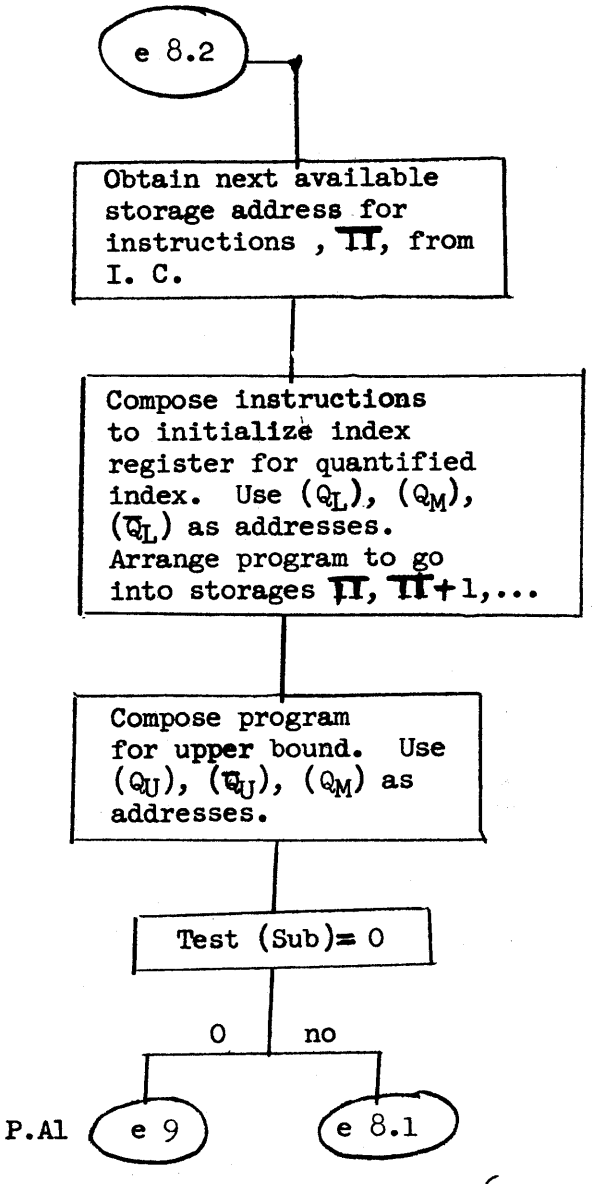

e 7.2
Test S for Address
address
no
Box 6.2
S Q_L [Q_U]
0 → (Sc)
i.e. erase index
Scan left
e 7
P. q1
Test S for r
r
no
Error 7
Scan left
e 6.3
Sc → Sc_q
e 6.9
Non-zero → T_q
e 2.11
P. b9
e 7.3
Box 6.3
α q → Q_L [Q_U]
Test setup of q
no
yes
Error 8
Sc_1 → Sc
e 7
P. q1


e 6.1
Test S for r
r
no
Sc → Sc1
Test S for a of deg. 2
a
no
Test rt. side of r-equation for α
e 6.3
P. q2
Error 10
α
no
Box 6.4
α r → QL [QU]
Test rt. side of r-eq. for minimization f
Sc1 → Sc
min. f
no
e 7
P. q1
Obtain upper bound of minimization
e 6.1 → TS
Sc1 → Scq
e 7.4
P. q4
Enter r-schema at e 33, p.r1

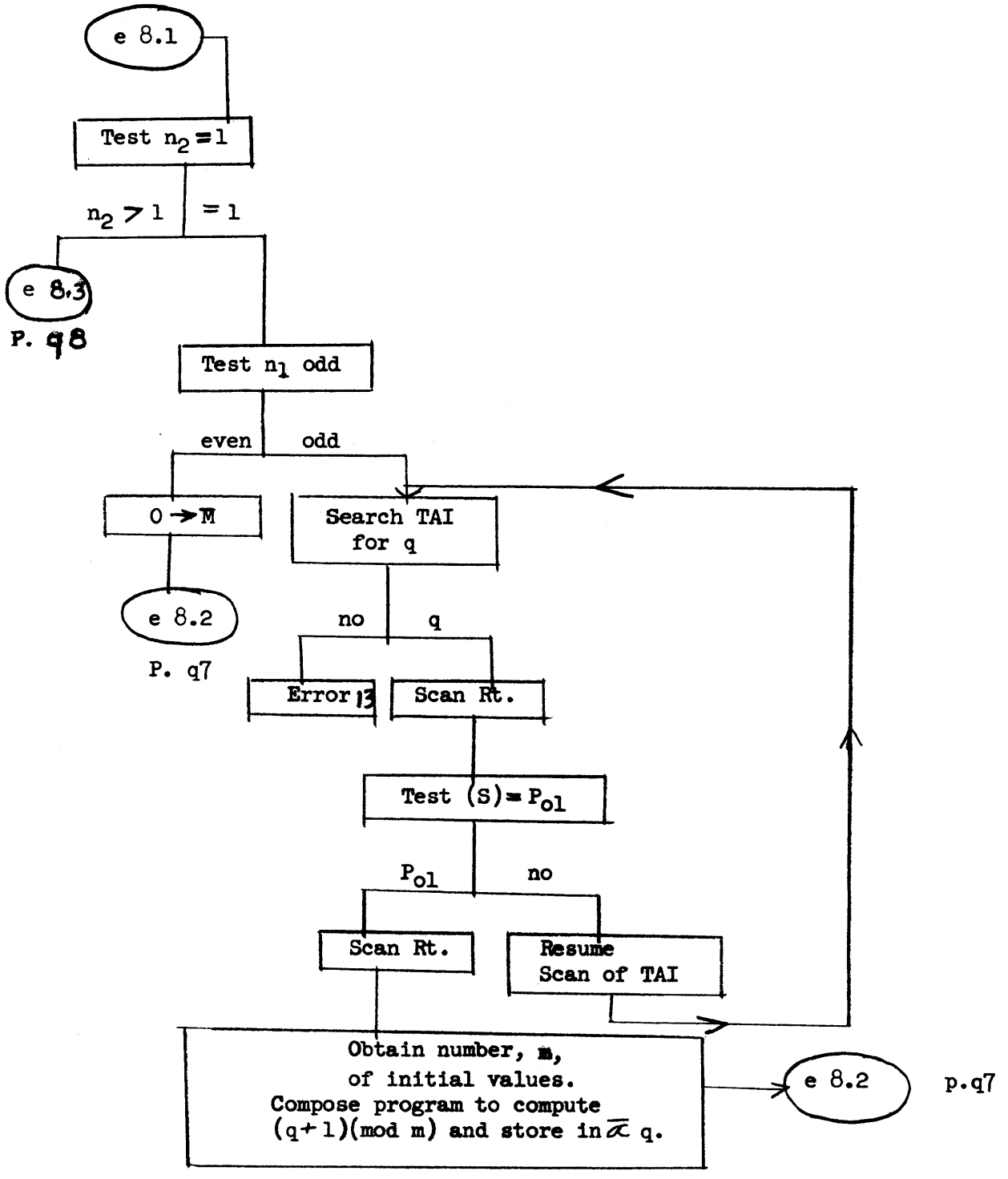

e 7.4
Test u.b. = constant
const.
no
const. → Q_U
e 6.8
Obtain b in minimization equation
Replace r, the u.b. in quantification, by b.
Sc_1 → Sc
P.q 7
e 8.2
Test u.b. = a of degree 1
a
no
α a → Q_u
Test u.b. = a of degree 2
Scan rt. from a
a
no
e 6.10
P.q4
Error 9
Test S for address
address
no
S → Q_U
Test S for r
e 6.8
P. q4
r
no
Scan left
Error 11
e 6.10
Sc_1 → Sc_q
e 6.9
P.q2

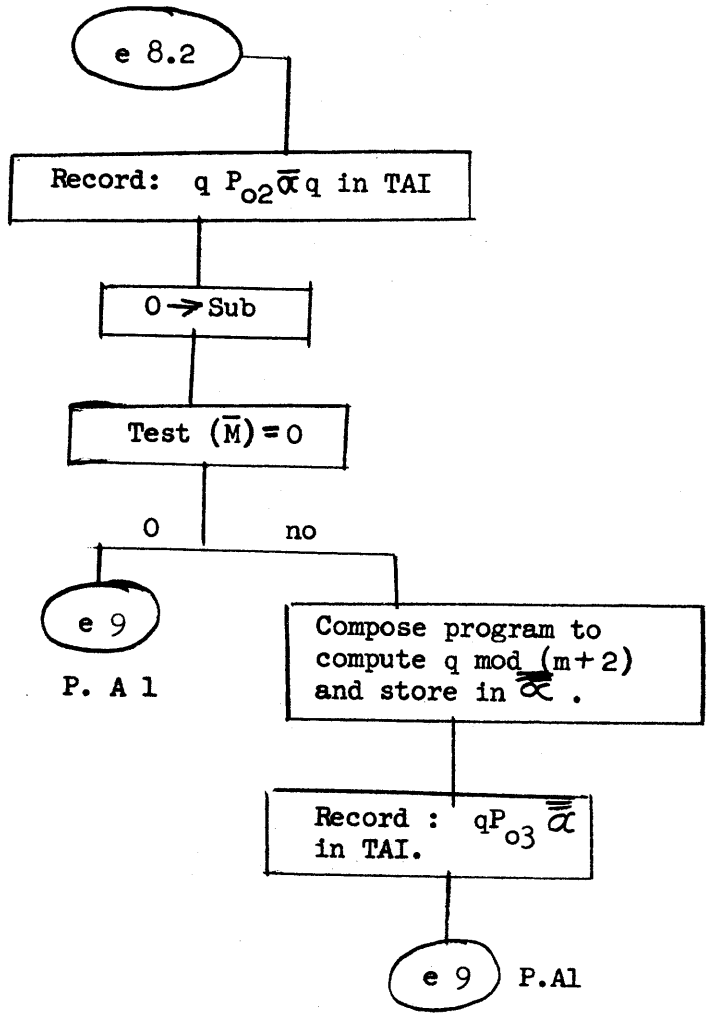

e 7.1
Scan Rt.
S → Q_M
Test setup of q
yes
no
Error 12
Record setup of q by storing α_q in a table of q vs. α q.
Change Q_L to Q_U and $\bar{Q}_L$ to $\bar{Q}_U$ in boxes 6.1-6.5. Change exit of box 7.0 to e 8.2
e 6.4

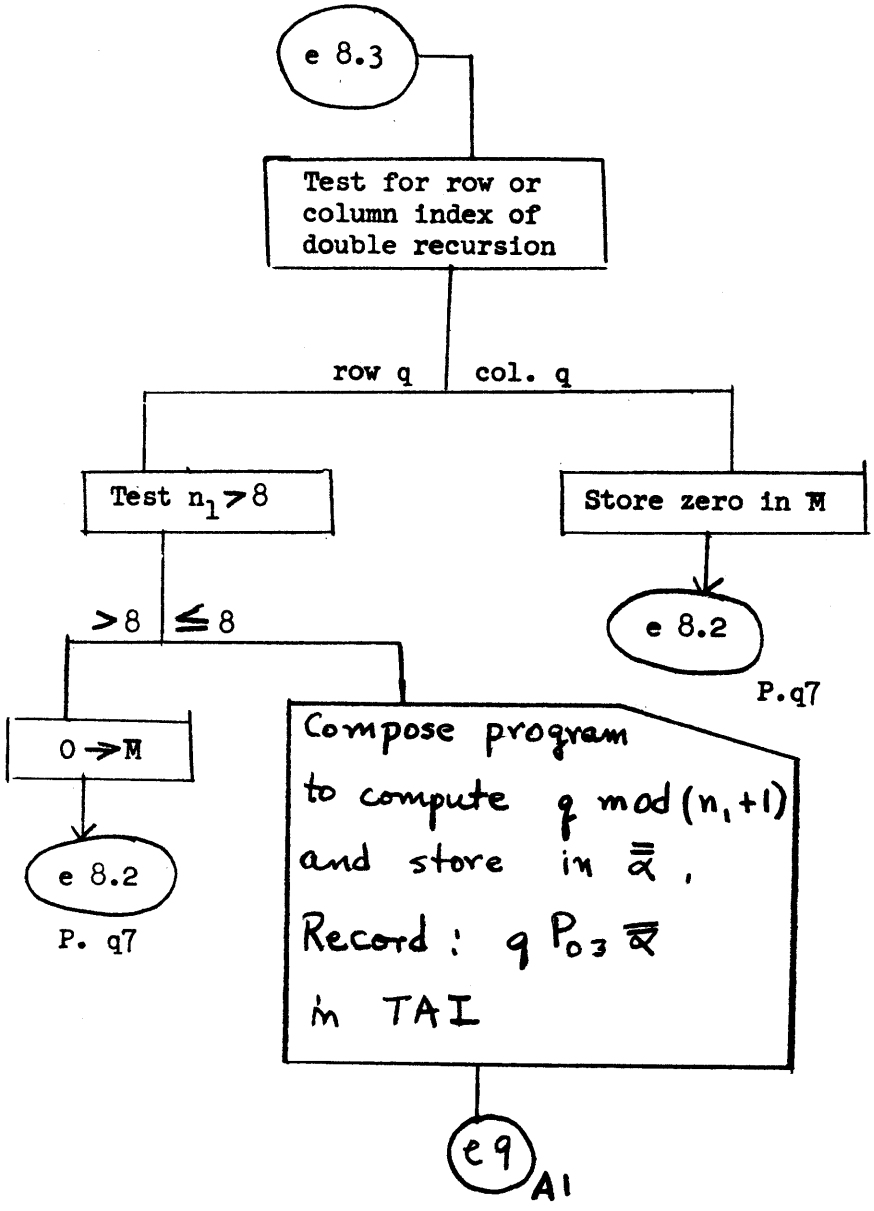
P. q1


e 8.2
Obtain next available storage address for instructions, Π, from I. C.
Compose instructions to initialize index register for quantified index. Use (Q_L), (Q_M), ($\bar{Q}_L$) as addresses. Arrange program to go into storages Π, Π+1,...
Compose program for upper bound. Use (Q_U), ($\bar{Q}_U$), (Q_M) as addresses.
Test (Sub)= 0
0
no
P.A1
e 9
e 8.1

p.q6


e 8.1
Test $n_2 = 1$
$n_2 > 1$
$= 1$
e 8.3
P. q8
Test n_1 odd
even
odd
0 → M
e 8.2
P. q7
Search TAI for q
no
q
Error 13
Scan Rt.
Test (S) = P_{01}
P_{01}
no
Scan Rt.
Resume Scan of TAI
Obtain number, m, of initial values. Compose program to compute (q+1)(mod m) and store in q.
e 8.2
p.q7


e 8.2
Record: q $P_{o2}\bar{\alpha}$ q in TAI
0 → Sub
Test $(\bar{M}) = 0$
0
no
e 9
P. A 1
Compose program to compute q mod (m+2) and store in $\bar{\alpha}$.
Record : $qP_{o3}\bar{\alpha}$ in TAI.
e 9
P.A1

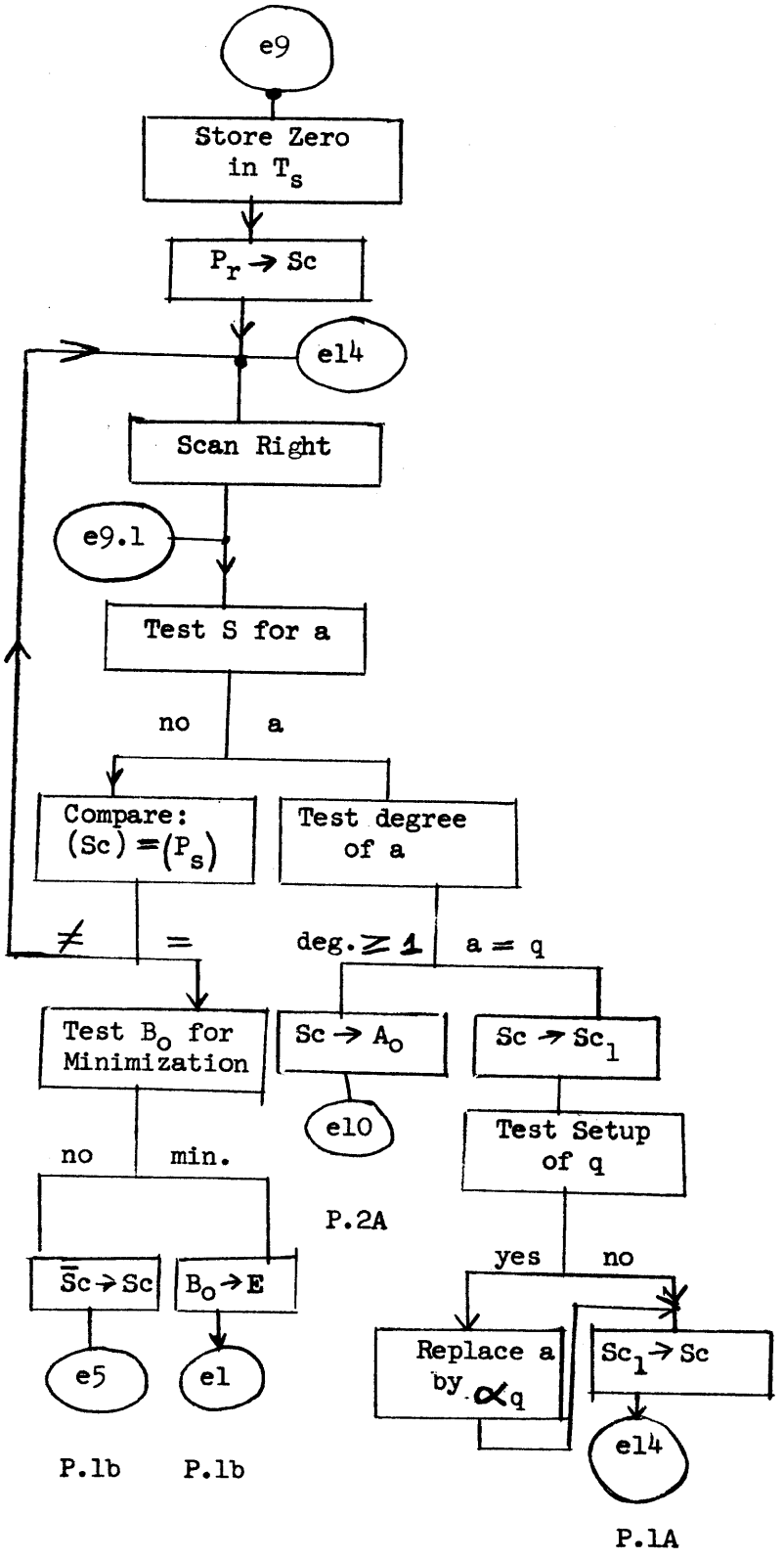

e 8.3
Test for row or column index of double recursion
row q
col. q
Test $n_1 > 8$
Store zero in M
e 8.2
P.q7
> 8
≤ 8
0 → M
e 8.2
P. q7
Compose program to compute q mod (n_1+1) and store in $\overline{\overline{a}}$.
Record : q P_{03} $\overline{\overline{a}}$ in TAI
e 9
A1

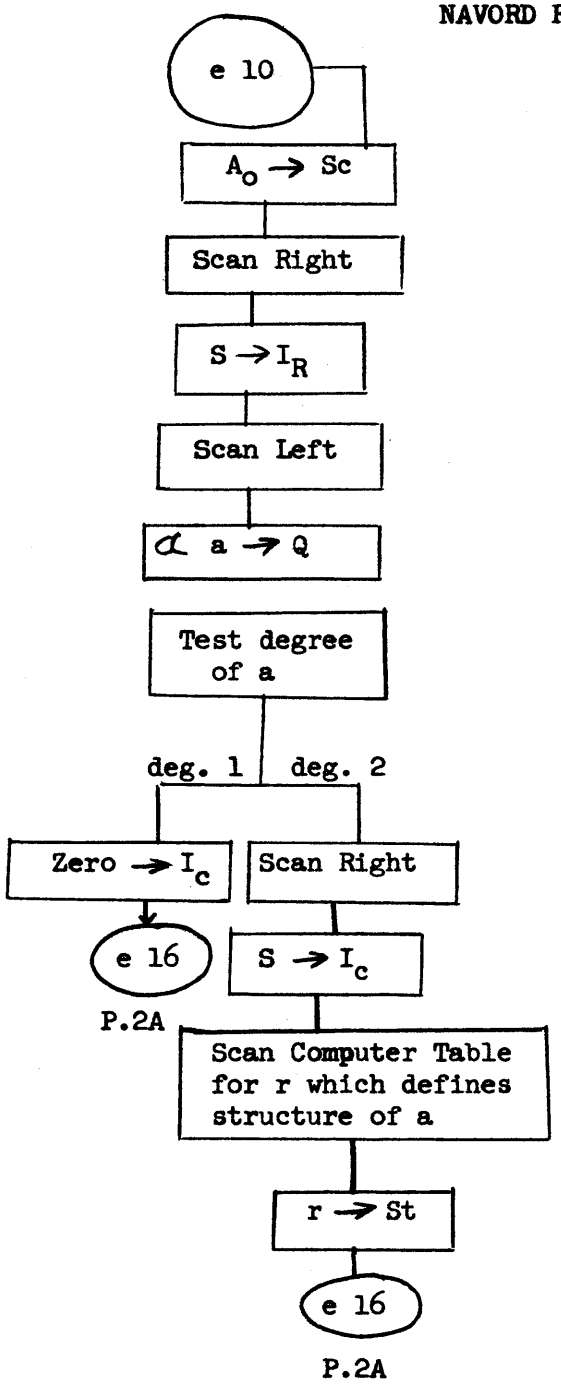
Explanatory Notes; q-Schema

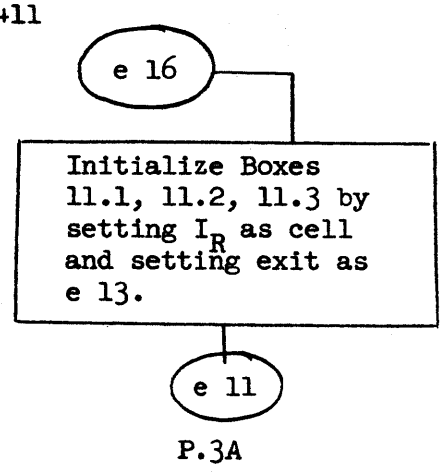
1. The following standard cells are used:

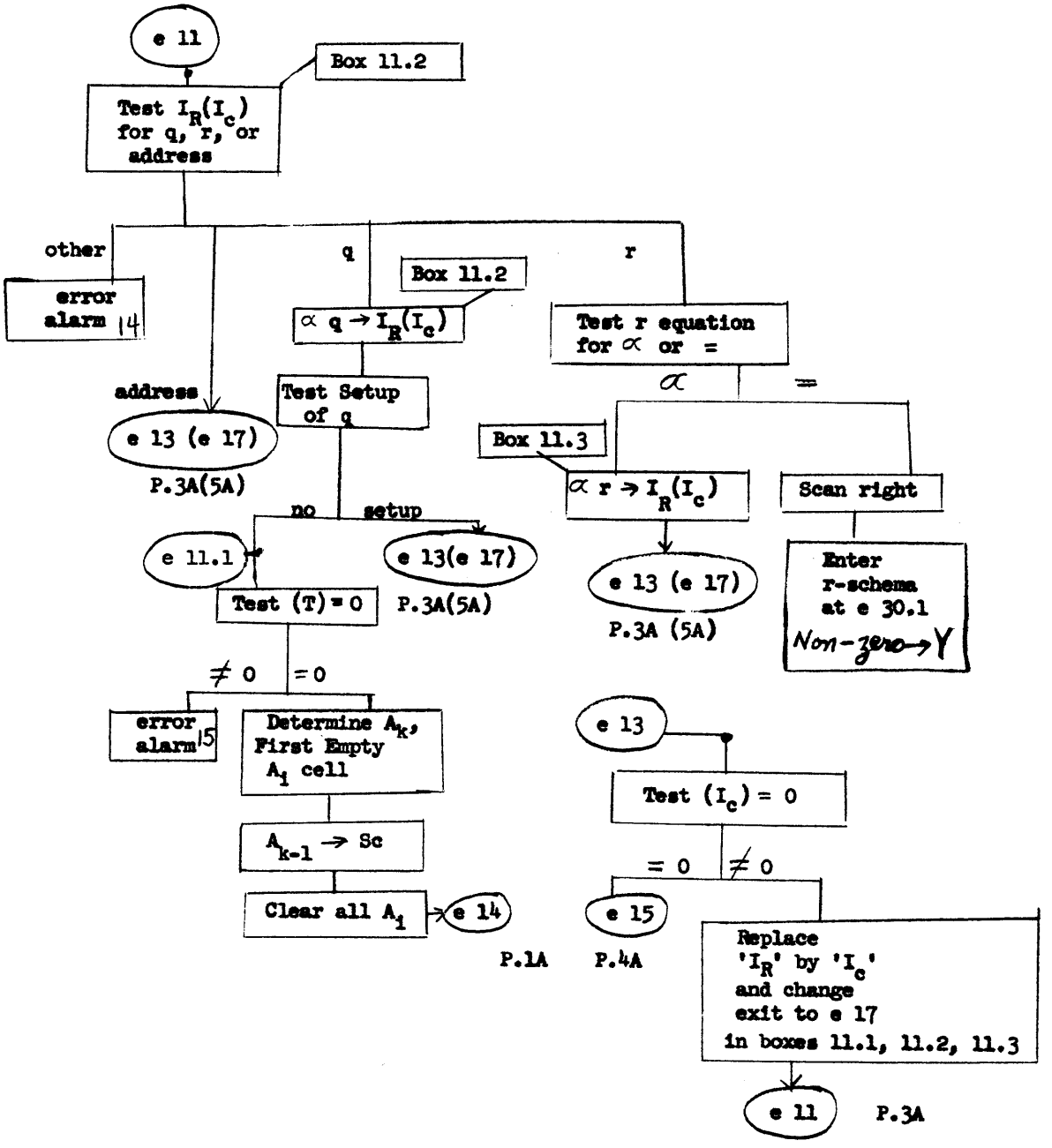
Cell	Contents
Q_L	Initial address of lower bound of quantifier.
$\overline{Q}_L$	Address of Index for the lower bound of quantifier.
Q_U	Initial address of upper bound of quantifier.
$\overline{Q}_U$	Address of index for the upper bound of quantifier.
Q_M	Address of independent index being quantified.

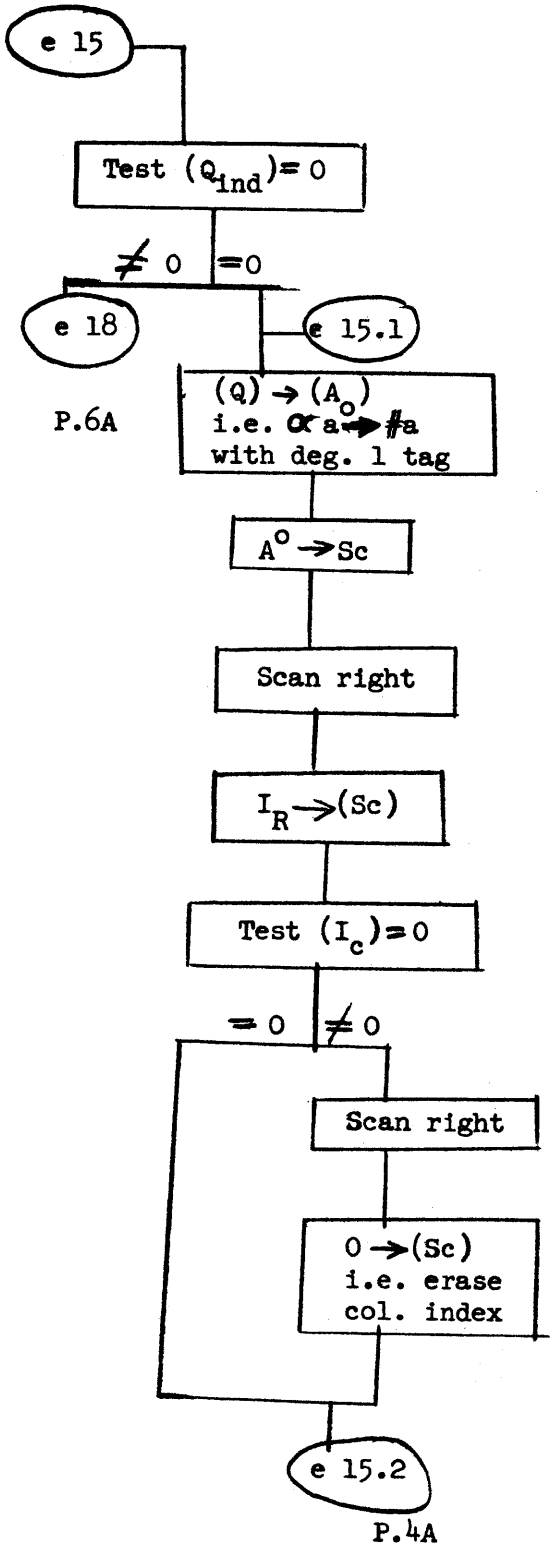
ADDRESSOR

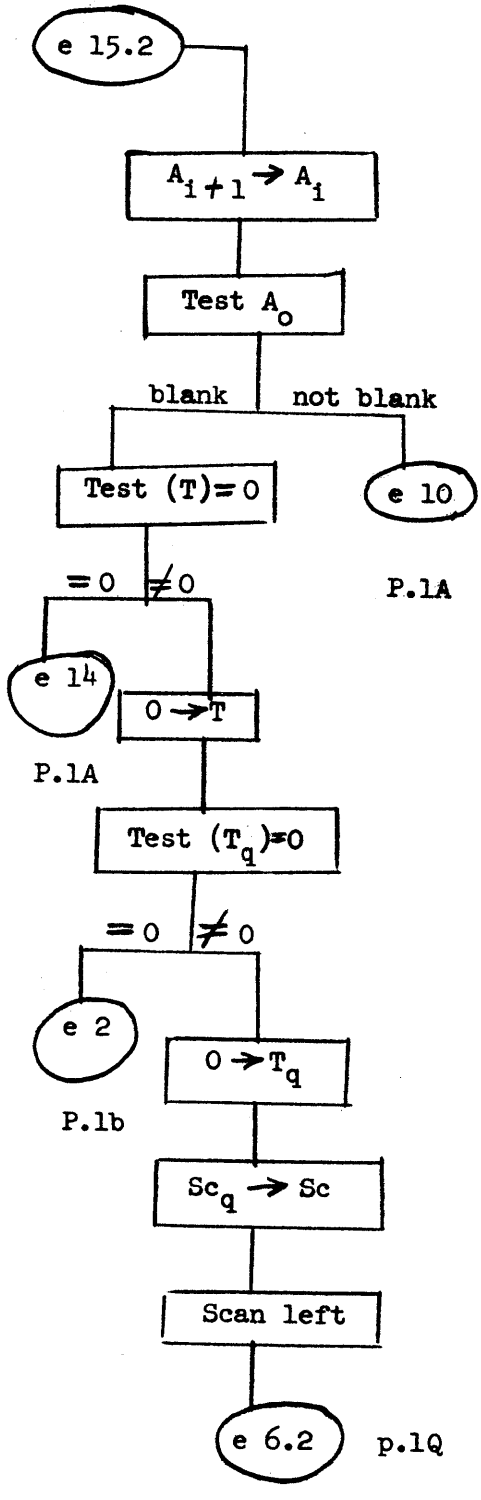

e9
Store Zero in T_s
$P_r \rightarrow Sc$
e14
Scan Right
e9.1
Test S for a
no
a
Compare: $(Sc) = (P_s)$
Test degree of a
$\neq$
=
deg. ≥ 1
a = q
Test B_o for Minimization
$Sc \rightarrow A_o$
$Sc \rightarrow Sc_1$
e10
Test Setup of q
no
min.
P.2A
yes
no
$\overline{Sc} \rightarrow Sc$
$B_o \rightarrow E$
Replace a by α_q
$Sc_1 \rightarrow Sc$
e5
e1
e14
P.1b
P.1b
P.1A

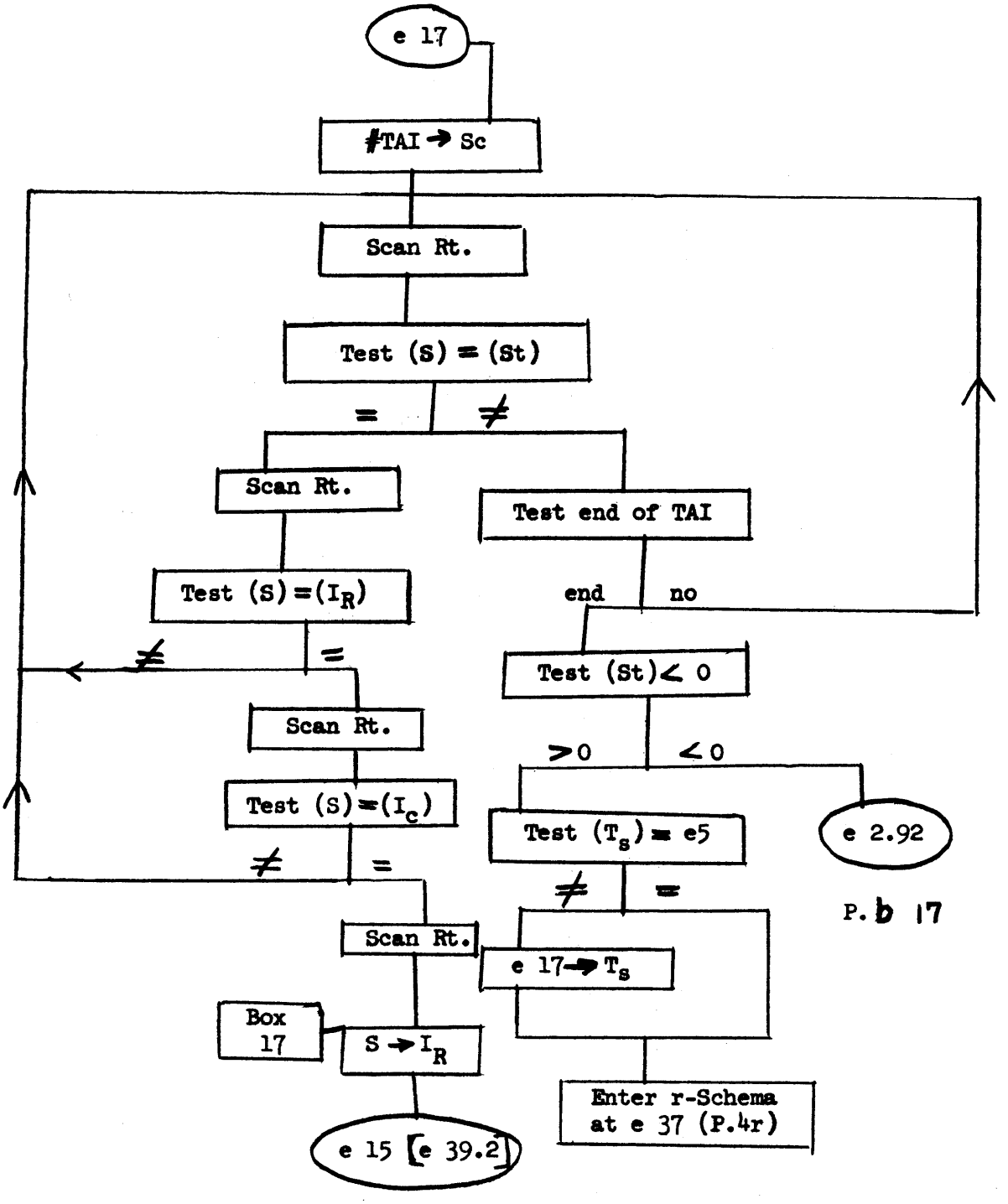

e 10
$A_o \rightarrow Sc$
Scan Right
$S \rightarrow I_R$
Scan Left
α a $\rightarrow$ Q
Test degree of a
deg. 1
deg. 2
Zero $\rightarrow I_c$
e 16
P.2A
Scan Right
$S \rightarrow I_c$
Scan Computer Table for r which defines structure of a
r $\rightarrow$ St
e 16
P.2A

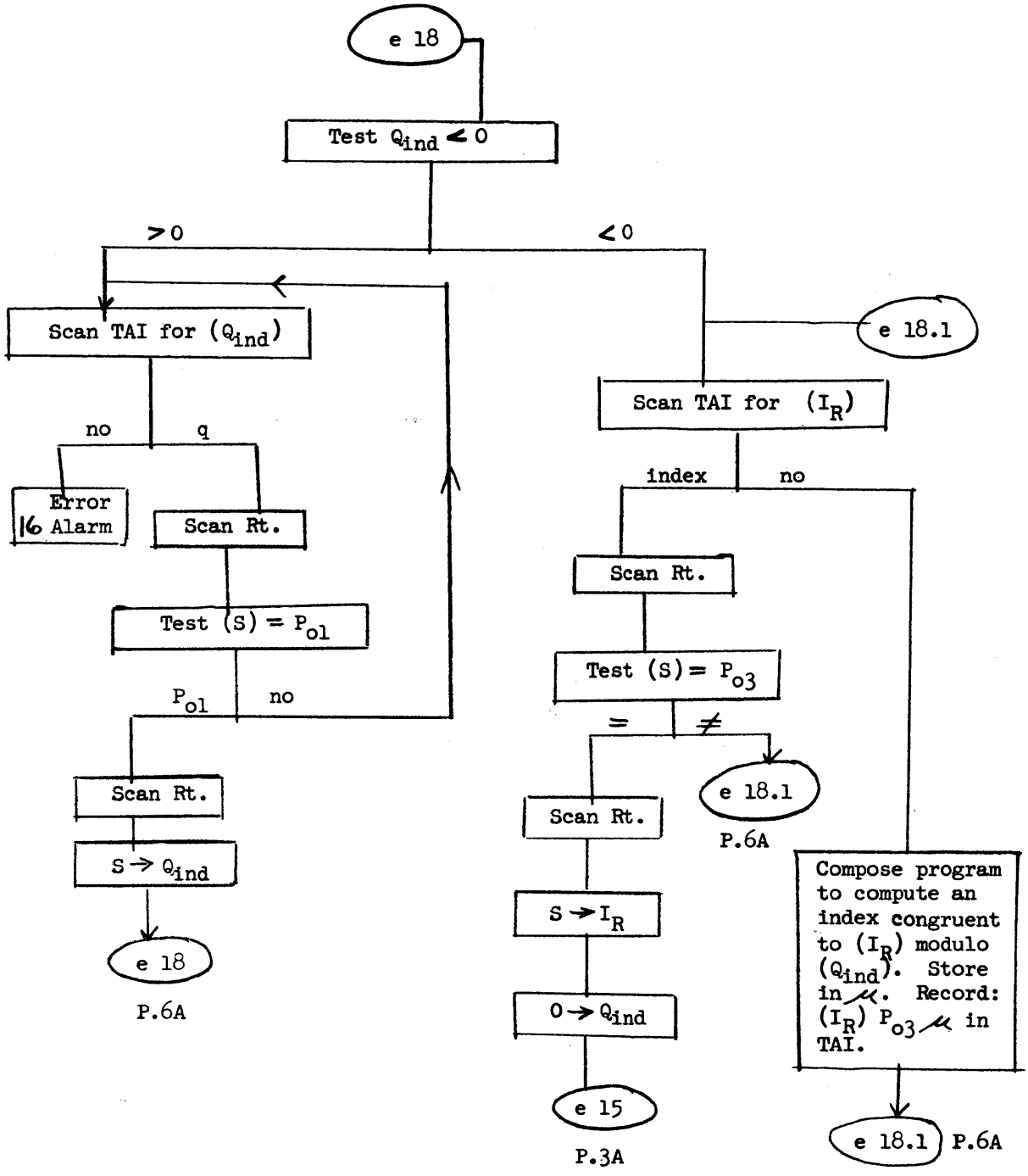

e 16
Initialize Boxes 11.1, 11.2, 11.3 by setting I_R as cell and setting exit as e 13.
e 11
P.3A


e 11
Box 11.2
Test $I_R(I_c)$ for q, r, or address
other
q
r
error alarm 14
Box 11.2
$\propto$ q → $I_R(I_c)$
Test r equation for $\propto$ or =
address
Test Setup of q
$\propto$
=
e 13 (e 17)
P.3A(5A)
Box 11.3
$\propto$ r → $I_R(I_c)$
Scan right
no setup
e 11.1
e 13(e 17) P.3A(5A)
e 13 (e 17)
P.3A (5A)
Enter r-schema at e 30.1
Non-zero → Y
Test (T) = 0
≠ 0
= 0
error alarm 15
Determine A_k, First Empty A_i cell
A_{k-1} → Sc
Clear all A_i
e 14
P.1A
e 13
Test (I_c) = 0
= 0
≠ 0
e 15
P.4A
Replace 'I_R' by 'I_c' and change exit to e 17 in boxes 11.1, 11.2, 11.3
e 11
P.3A


e 15
Test (Q_{ind})= 0
≠ 0
=0
e 18
e 15.1
P.6A
(Q) → (A_o)
i.e. α a → #a
with deg. 1 tag
A^o → Sc
Scan right
I_R → (Sc)
Test (I_c)= 0
= 0
≠ 0
Scan right
0 → (Sc)
i.e. erase
col. index
e 15.2
P.4A


e 15.2
A_{i+1} → A_i
Test A_o
blank
not blank
Test (T)= 0
e 10
P.1A
= 0
≠ 0
e 14
P.1A
0 → T
Test (T_q)=0
= 0
≠ 0
e 2
P.1b
0 → T_q
Sc_q → Sc
Scan left
e 6.2
p.1Q


e 17
#TAI → Sc
Scan Rt.
Test (S) = (St)
=
≠
Scan Rt.
Test end of TAI
Test (S) = (I_R)
end
no
≠
=
Test (St) < 0
Scan Rt.
>0
<0
Test (S) = (I_c)
Test (T_s) = e5
e 2.92
≠
=
P. b 17
Scan Rt.
e 17 → T_s
Box 17
S → I_R
Enter r-Schema at e 37 (P.4r)
e 15 [e 39.2]
P.4A [r 5]

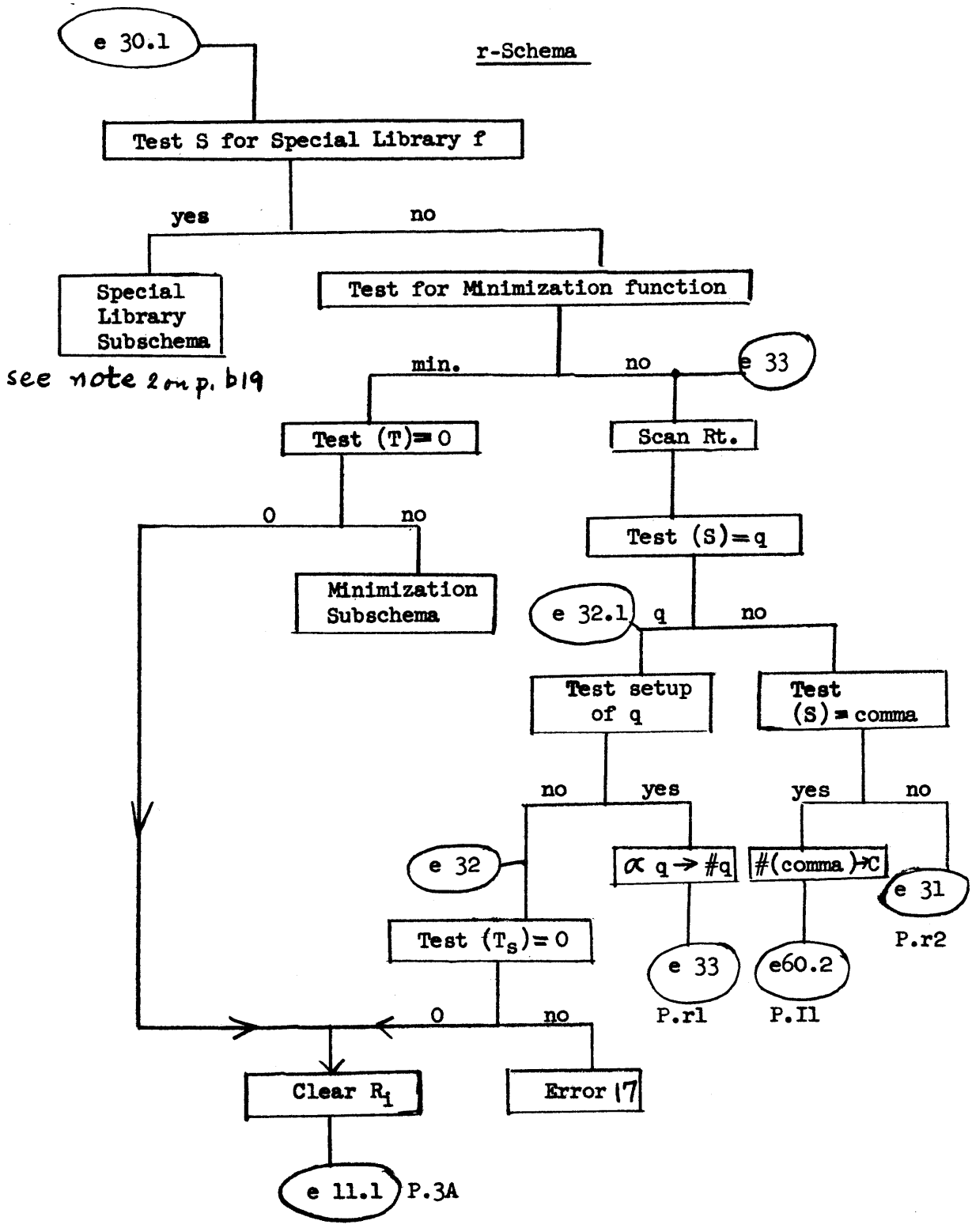

e 18
Test Q_{ind} < 0
> 0
< 0
Scan TAI for (Q_{ind})
no
q
Error
16 Alarm
Scan Rt.
Test (S) = P_{o1}
P_{o1}
no
Scan Rt.
S → Q_{ind}
e 18
P.6A
e 18.1
Scan TAI for (I_R)
index
no
Scan Rt.
Test (S) = P_{o3}
=
≠
e 18.1
P.6A
Scan Rt.
S → I_R
0 → Q_{ind}
e 15
P.3A
Compose program to compute an index congruent to (I_R) modulo (Q_{ind}). Store in . Record: (I_R) P_{o3} in TAI.
e 18.1 P.6A

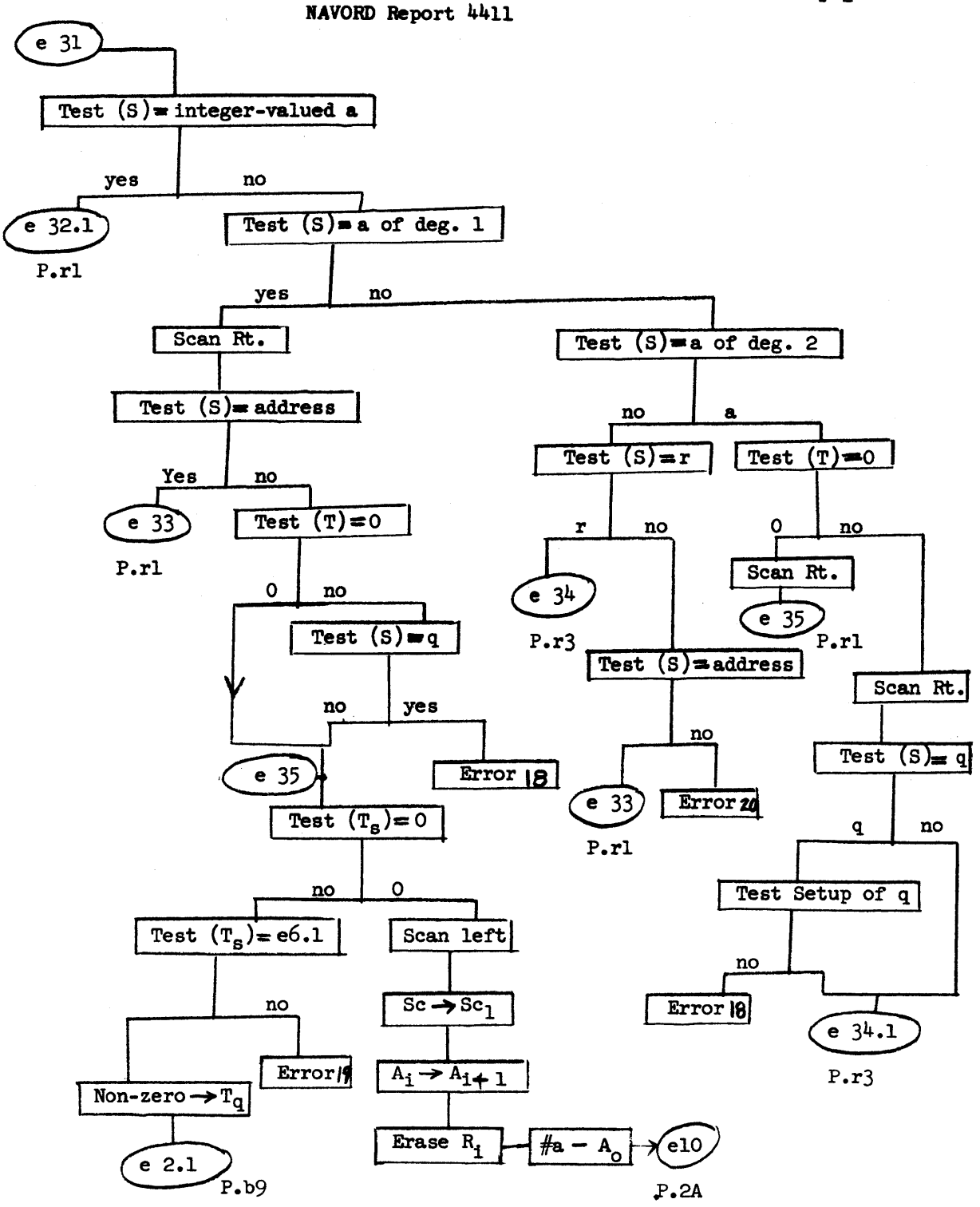
Explanatory Notes; Addressor

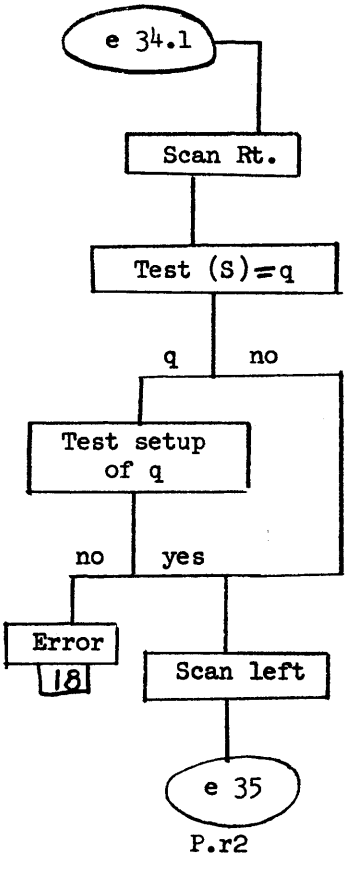
1. The following standard cells are used:

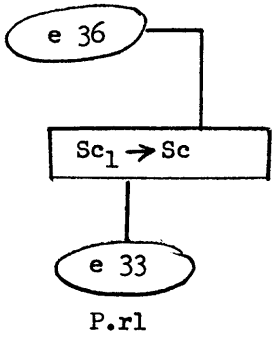
Cell		Contents
P_r	...	Cell number of comma after the Computer Table.
P_s	...	Cell number of comma after last equation.
A_i	...	(i = o,1,...50) Cell numbers of a-symbols being scanned.
Q	...	Initial address of an a-symbol.
I_R	...	Address of row index of an a-symbol.
I_C	...	Address of column index of an a-symbol.
$\overline{Sc}$	...	Cell number of upper bound in quantification.
Q_{ind}	...	Index in a recursion.
St	...	Index which specifies structure of matrix.
#TAI	...	Cell number of first cell before the Table of Addressed Indexes.

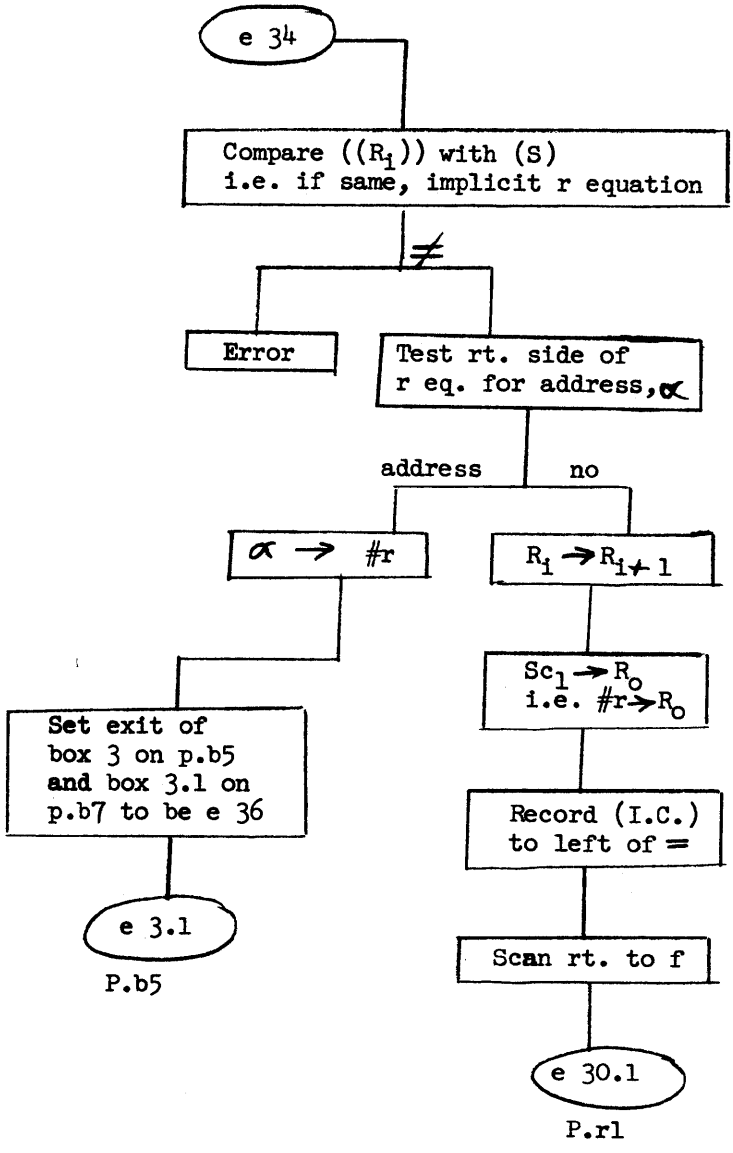
2. The symbol P_{o3} denotes punctuation.

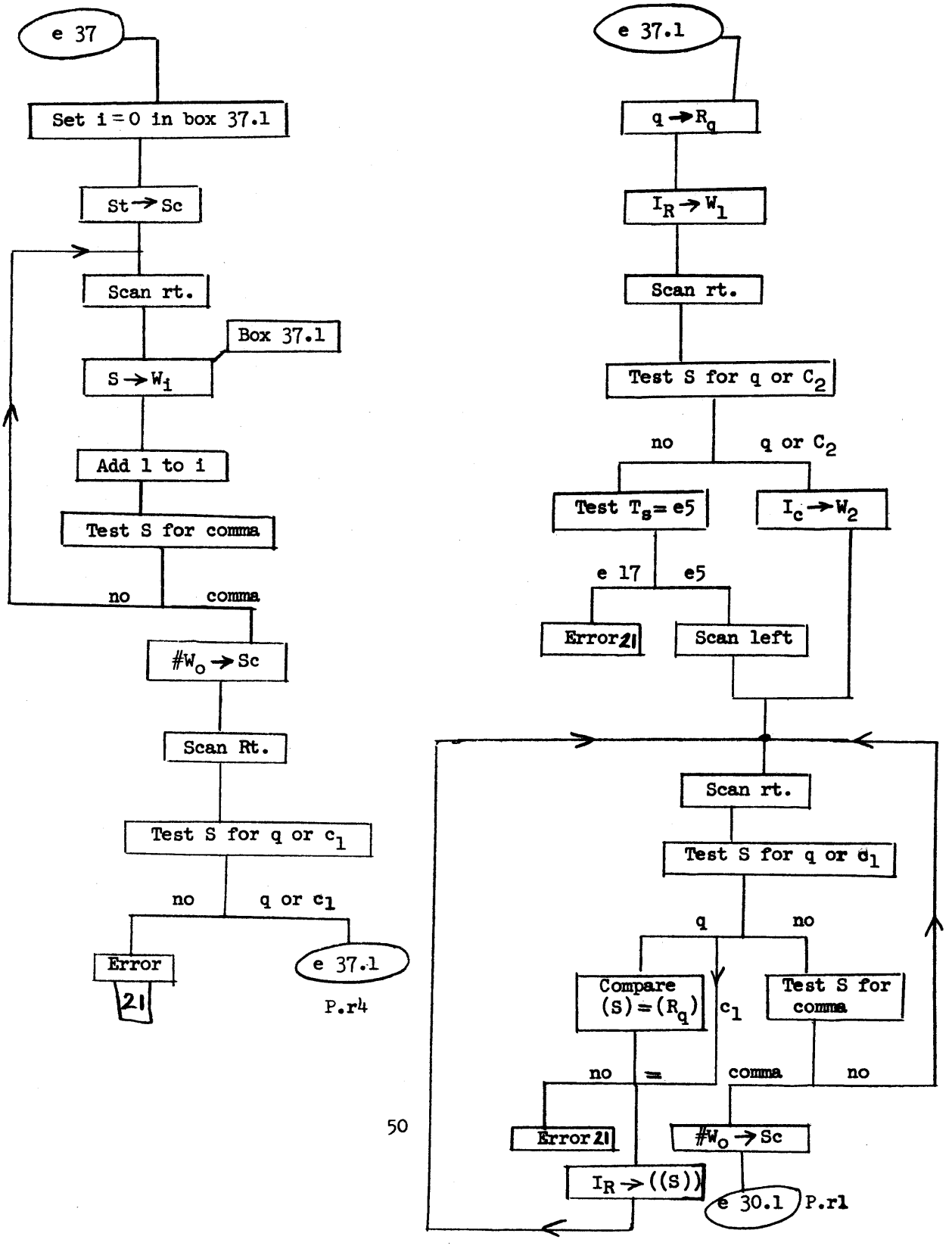

e 30.1
r-Schema
Test S for Special Library f
yes
no
Special Library Subschema
see note 2 on p. b19
Test for Minimization function
min.
no
e 33
Test (T)= 0
Scan Rt.
0
no
Test (S)=q
Minimization Subschema
e 32.1
q
no
Test setup of q
Test (S) = comma
no
yes
yes
no
e 32
∝ q → #q
#(comma)→C
e 31
P.r2
Test (Ts)= 0
e 33
P.r1
e60.2
P.I1
0
no
Clear Ri
Error 17
e 11.1
P.3A

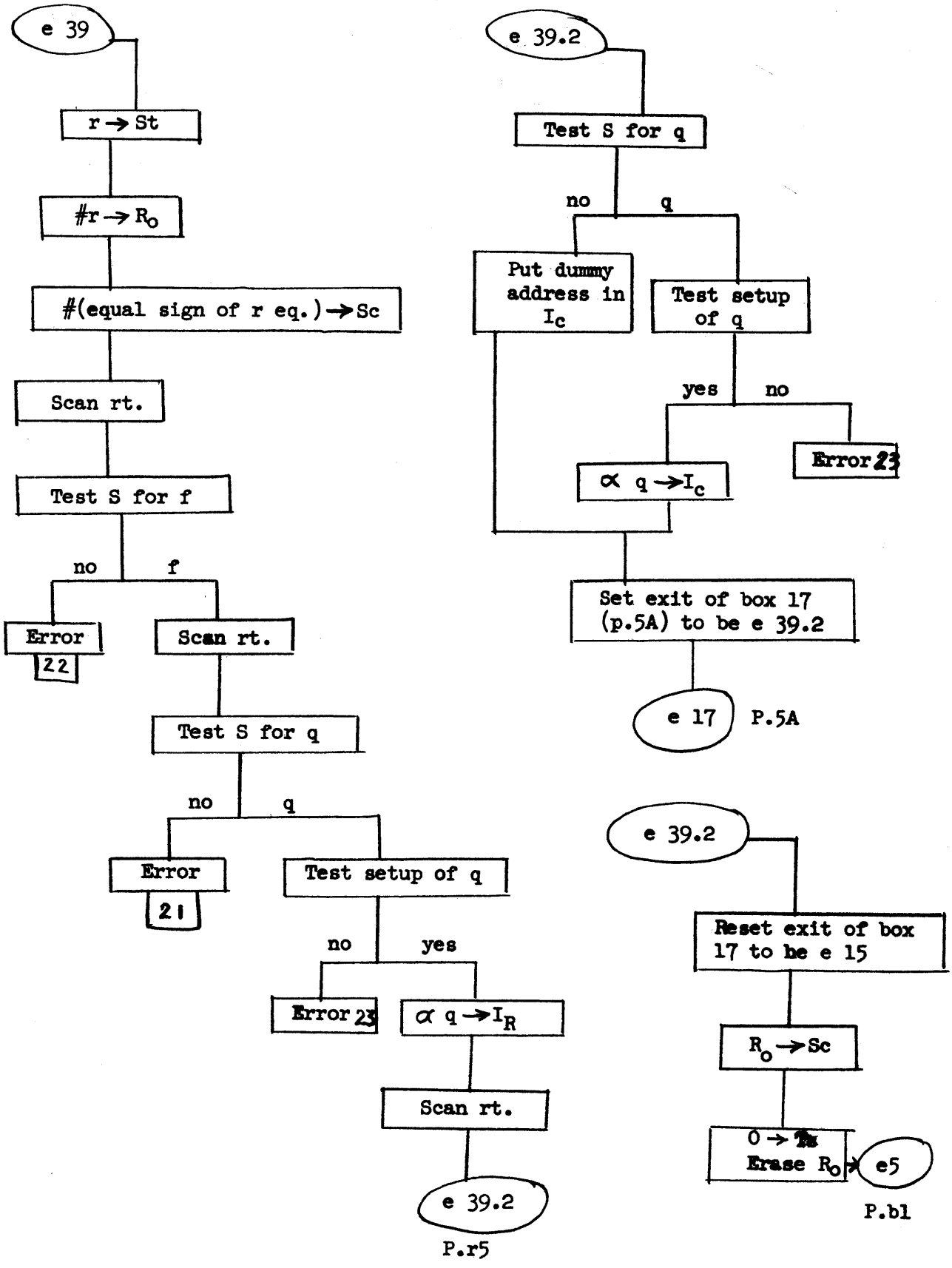

e 31
Test (S) = integer-valued a
yes
no
e 32.1
P.r1
Test (S) = a of deg. 1
yes
no
Scan Rt.
Test (S) = address
Yes
no
e 33
P.r1
Test (T) = 0
0
no
Test (S) = q
no
yes
e 35
Error 18
Test (T_s) = 0
no
0
Test (T_s) = e6.1
no
Non-zero → T_q
e 2.1
P.b9
Error 19
Scan left
Sc → Sc_1
A_i → A_i + 1
Erase R_i
#a − A_o
e10
P.2A
Test (S) = a of deg. 2
no
a
Test (S) = r
Test (T) = 0
r
no
e 34
P.r3
0
no
Scan Rt.
e 35
P.r1
Test (S) = address
no
e 33
P.r1
Error 20
Scan Rt.
Test (S) = q
q
no
Test Setup of q
no
Error 18
e 34.1
P.r3


e 34.1
Scan Rt.
Test (S)=q
q
no
Test setup of q
no
yes
Error
18
Scan left
e 35
P.r2


e 36
Sc1 → Sc
e 33
P.r1


e 34
Compare ((Ri)) with (S)
i.e. if same, implicit r equation
≠
Error
Test rt. side of r eq. for address, α
address
no
α → #r
Ri → Ri+1
Set exit of box 3 on p.b5 and box 3.1 on p.b7 to be e 36
Sc1 → Ro
i.e. #r → Ro
Record (I.C.) to left of =
e 3.1
P.b5
Scan rt. to f
e 30.1
P.r1


e 37
Set i = 0 in box 37.1
St → Sc
Scan rt.
Box 37.1
S → W_i
Add 1 to i
Test S for comma
no
comma
#W_o → Sc
Scan Rt.
Test S for q or c_1
no
q or c_1
Error
21
e 37.1
P.r4
e 37.1
q → R_q
I_R → W_1
Scan rt.
Test S for q or C_2
no
q or C_2
Test T_s = e5
I_c → W_2
e 17
e5
Error 21
Scan left
Scan rt.
Test S for q or c_1
q
no
Compare (S) = (R_q)
c_1
Test S for comma
no
=
comma
no
Error 21
#W_o → Sc
I_R → ((S))
e 30.1
P.r1

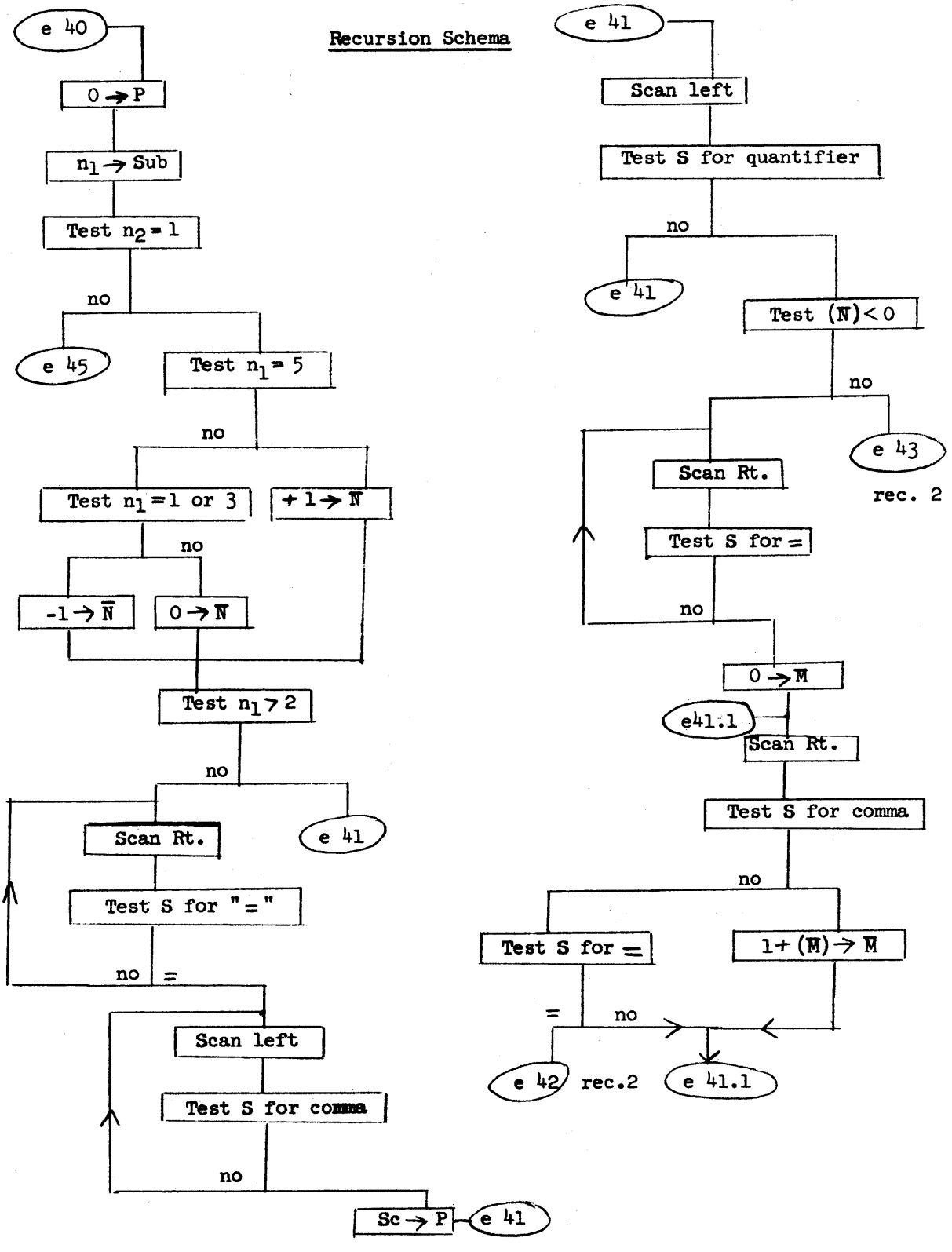

e 39
r → St
#r → R0
#(equal sign of r eq.) → Sc
Scan rt.
Test S for f
no
f
Error
22
Scan rt.
Test S for q
no
q
Error
21
Test setup of q
no
yes
Error 23
∝ q → IR
Scan rt.
e 39.2
P.r5
e 39.2
Test S for q
no
q
Put dummy address in Ic
Test setup of q
yes
no
Error 23
∝ q → Ic
Set exit of box 17 (p.5A) to be e 39.2
e 17
P.5A
e 39.2
Reset exit of box 17 to be e 15
R0 → Sc
0 → Erase R0
e5
P.b1

Explanatory Notes; r-Schema

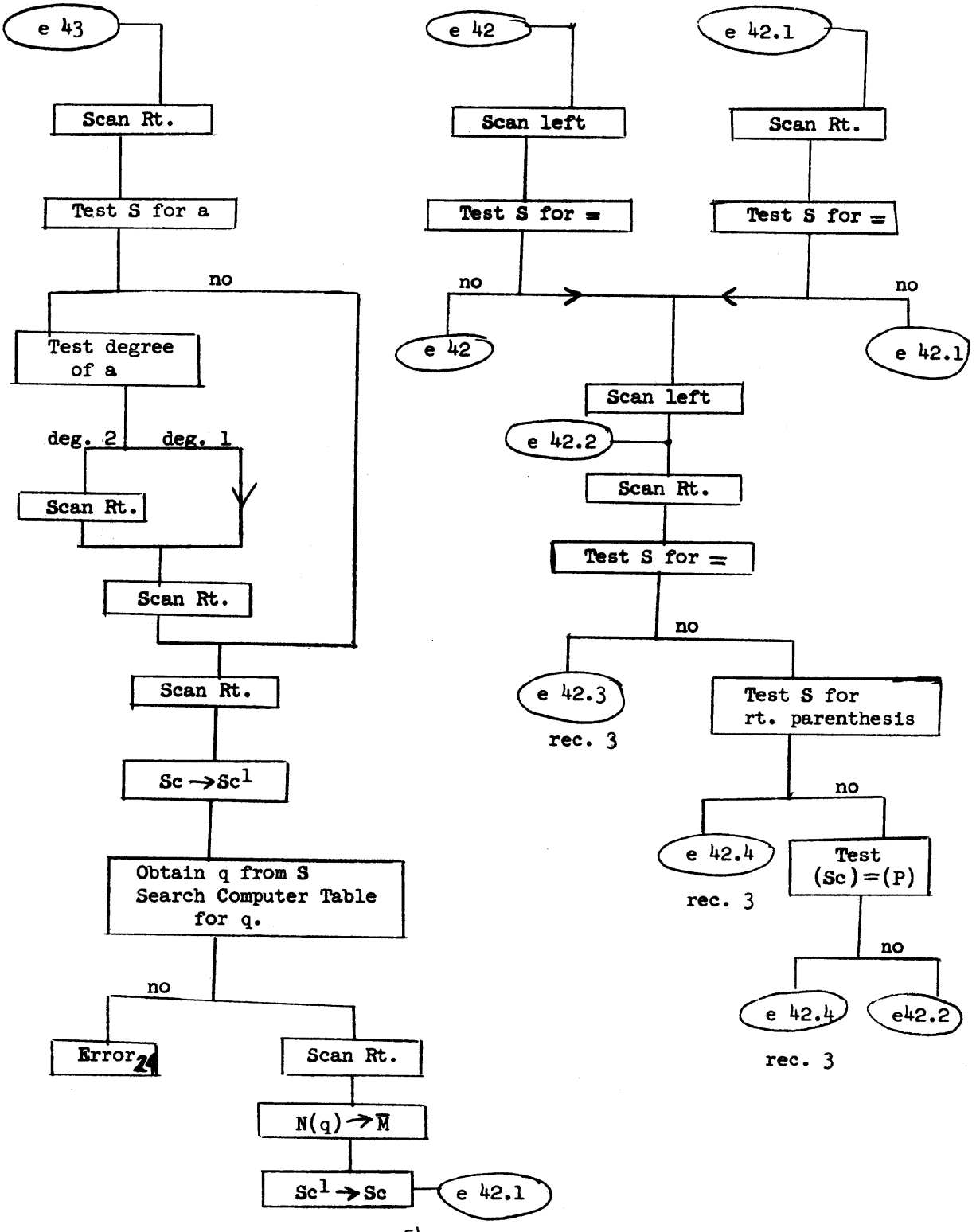
1. The following standard cells are used:

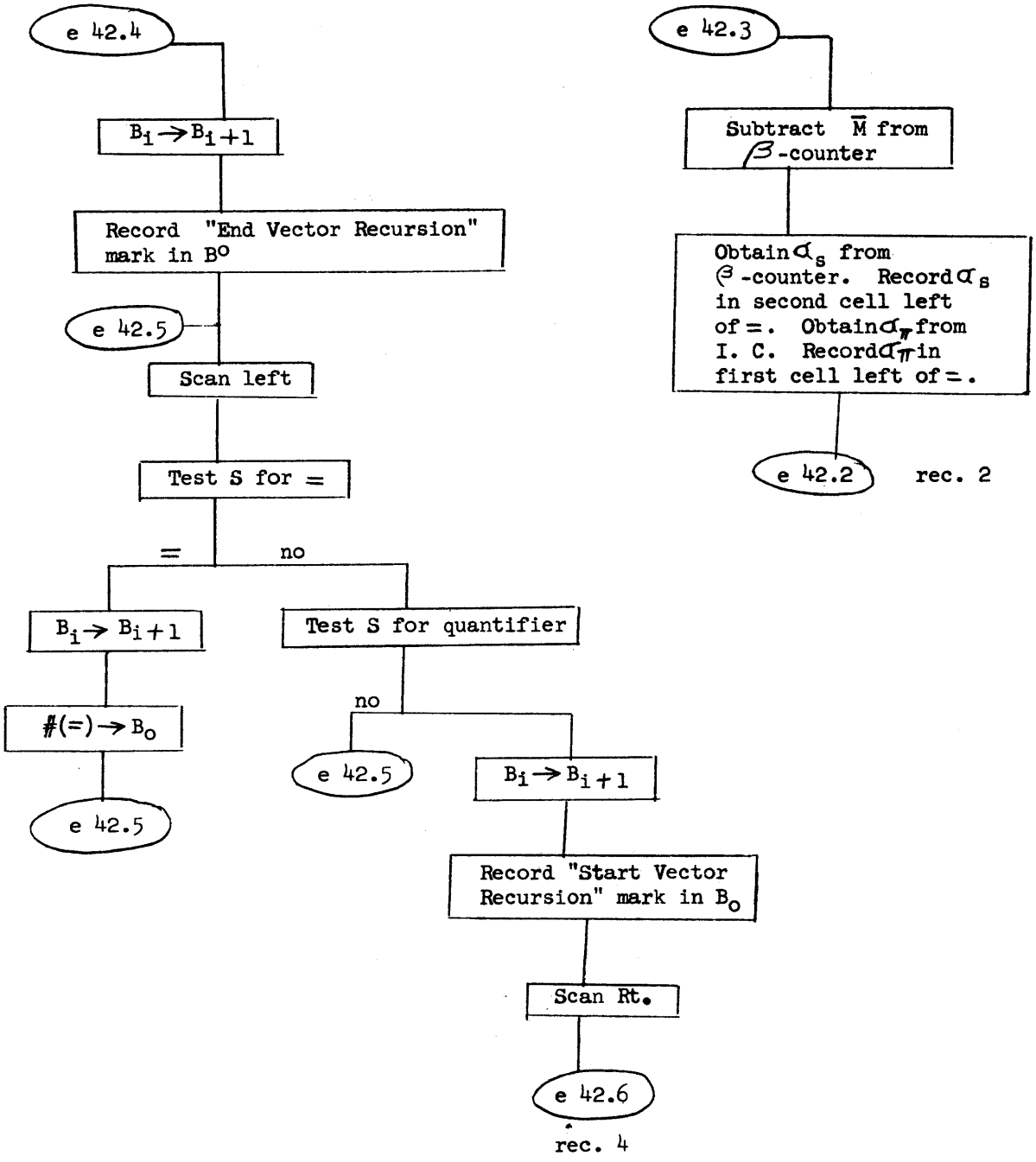
Cell		Contents
R_i	...	Cell numbers of r-symbols, $(i = 0,1,...,50)$.
W_i	...	These are cells into which the formula for each storage r-symbol is placed before being programmed. This preserves the original formula for possible later use, $(i = 0,1,...,50)$.

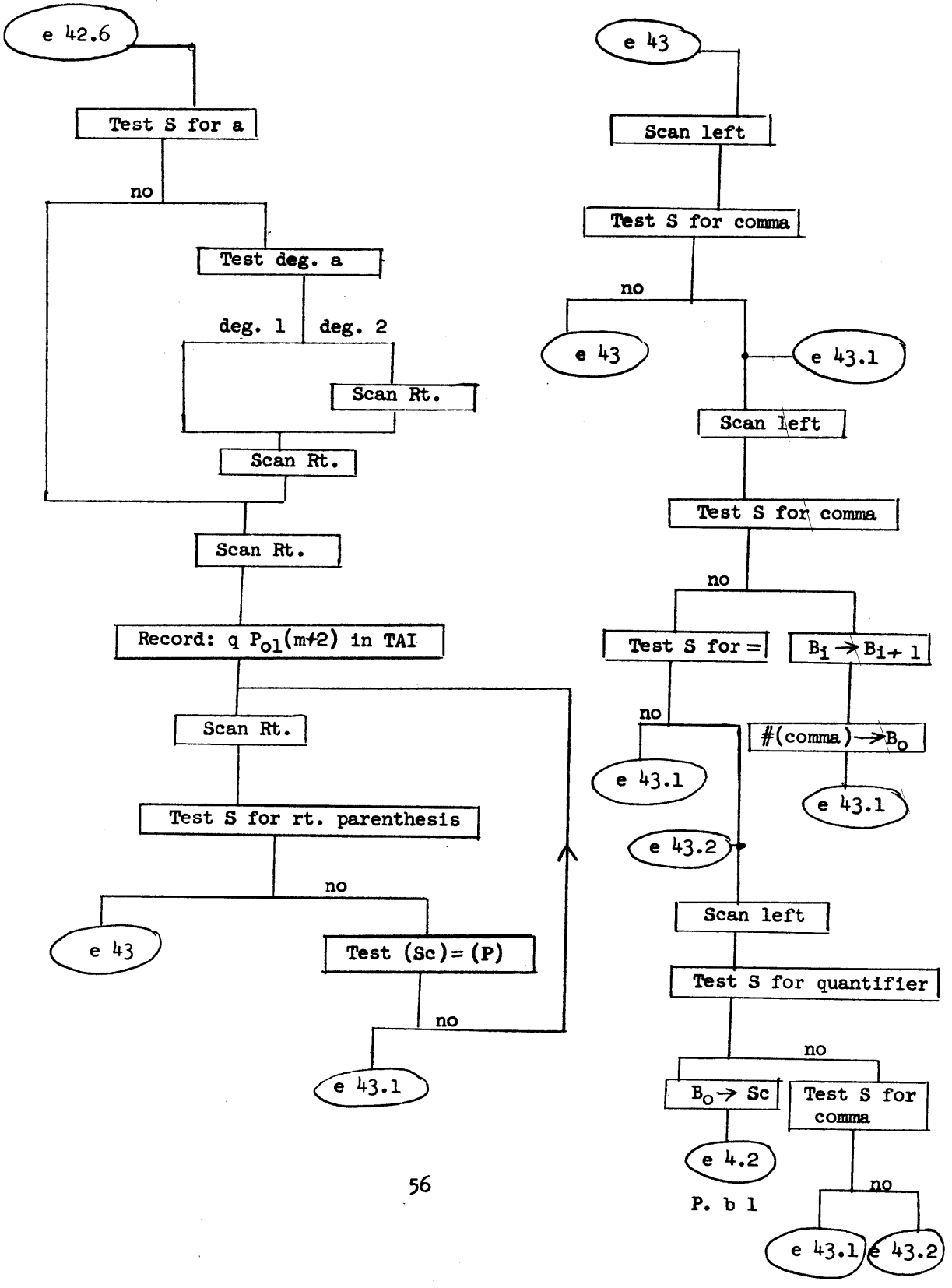
Recursion Schema

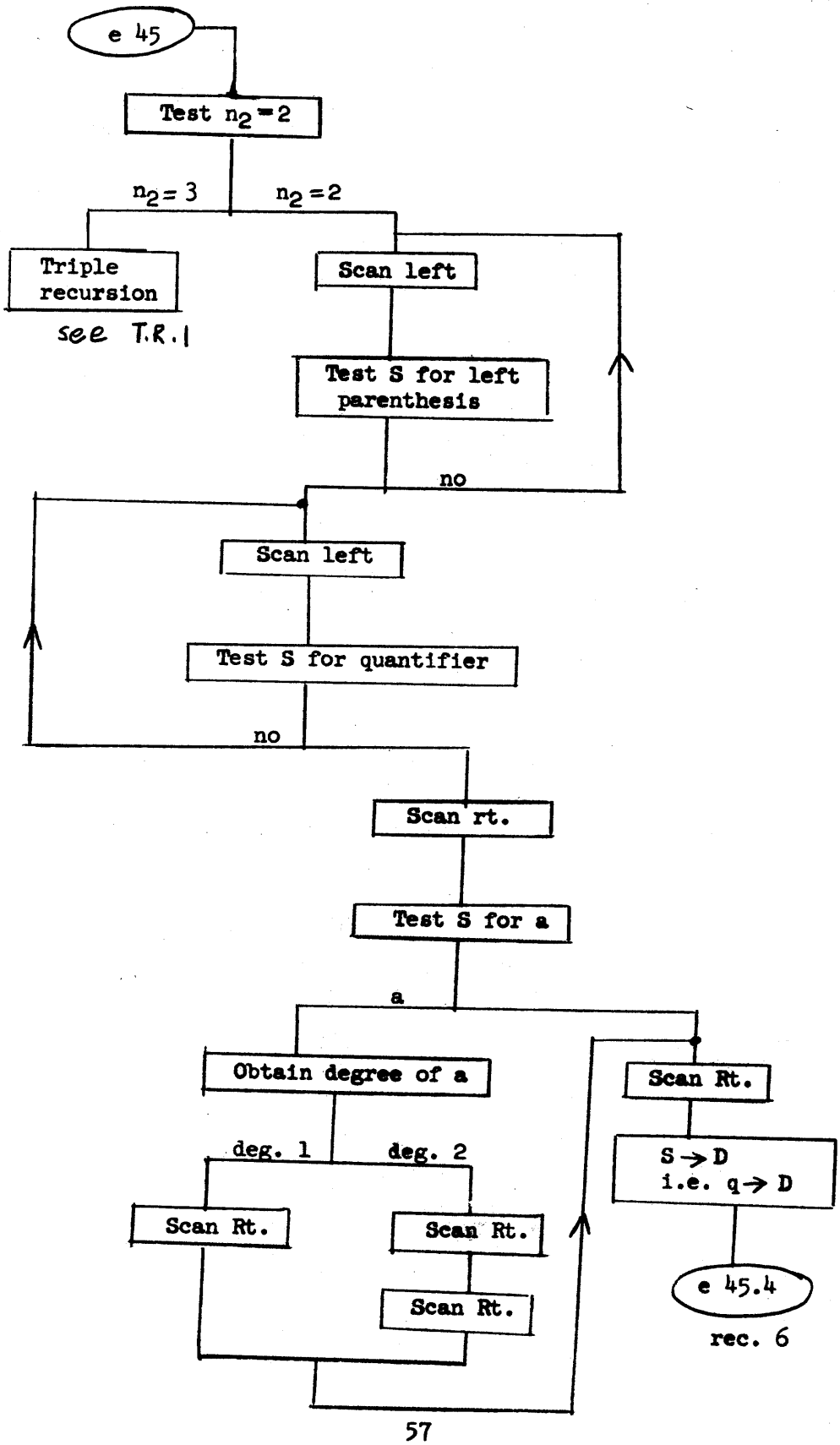


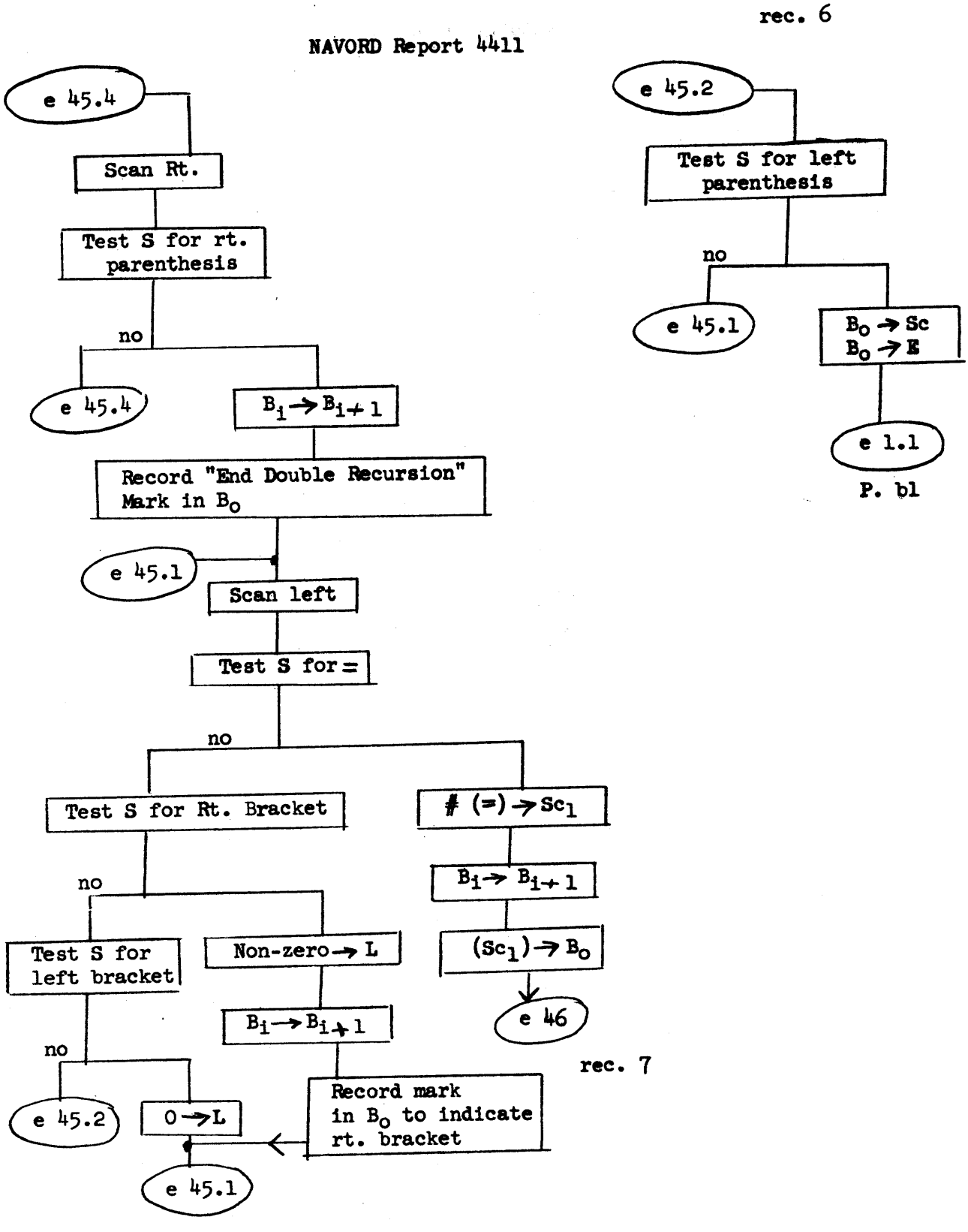
rec. 2

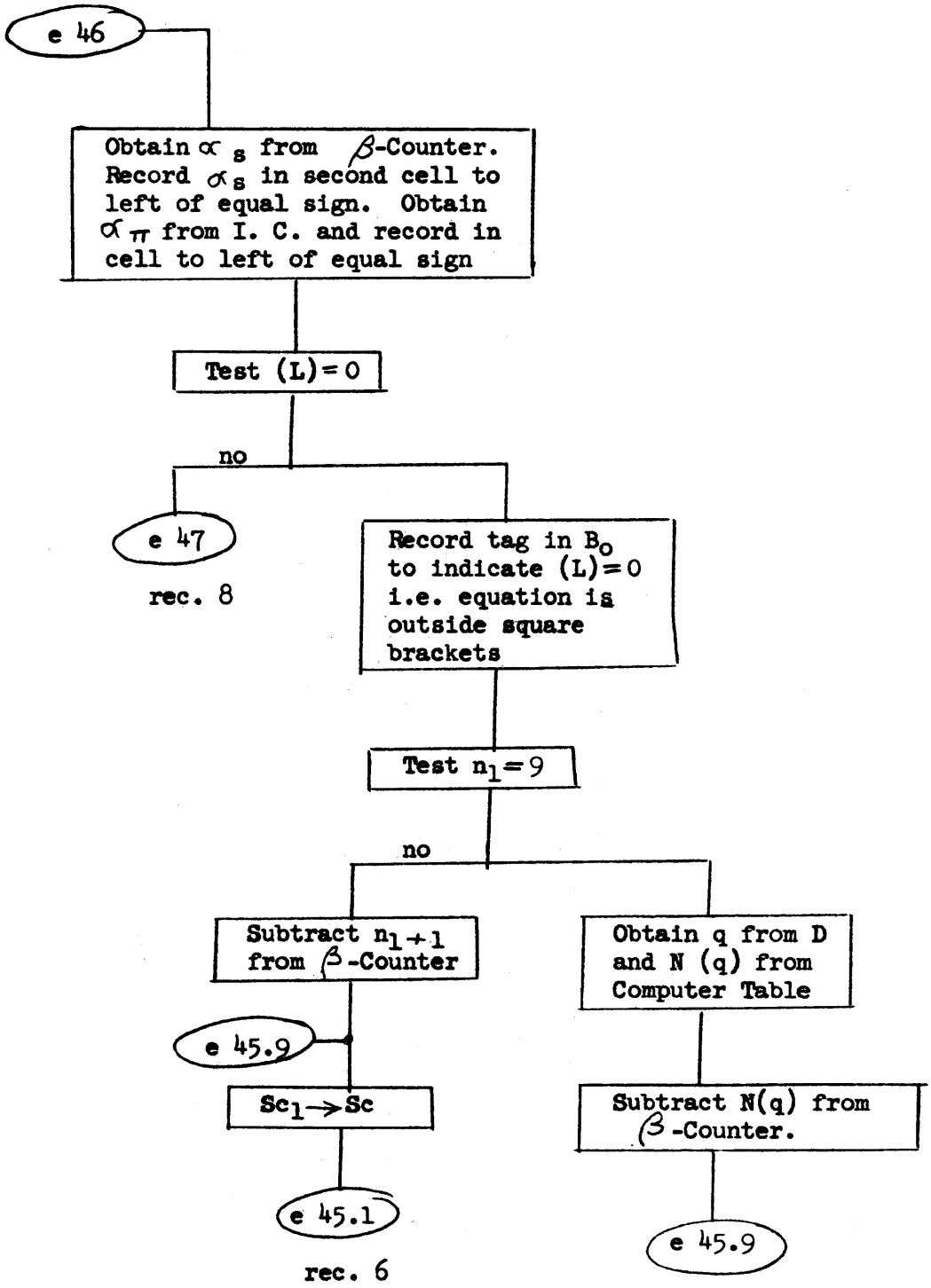

e 43
Scan Rt.
Test S for a
no
Test degree of a
deg. 2
deg. 1
Scan Rt.
Scan Rt.
Scan Rt.
Sc → Sc1
Obtain q from S
Search Computer Table
for q.
no
Error 24
Scan Rt.
N(q) → M
Sc1 → Sc
e 42.1
e 42
Scan left
Test S for =
no
e 42
Scan left
e 42.2
Scan Rt.
Test S for =
no
e 42.3
rec. 3
e 42.1
Scan Rt.
Test S for =
no
e 42.1
Test S for
rt. parenthesis
no
e 42.4
rec. 3
Test
(Sc)=(P)
no
e 42.4
e42.2
rec. 3

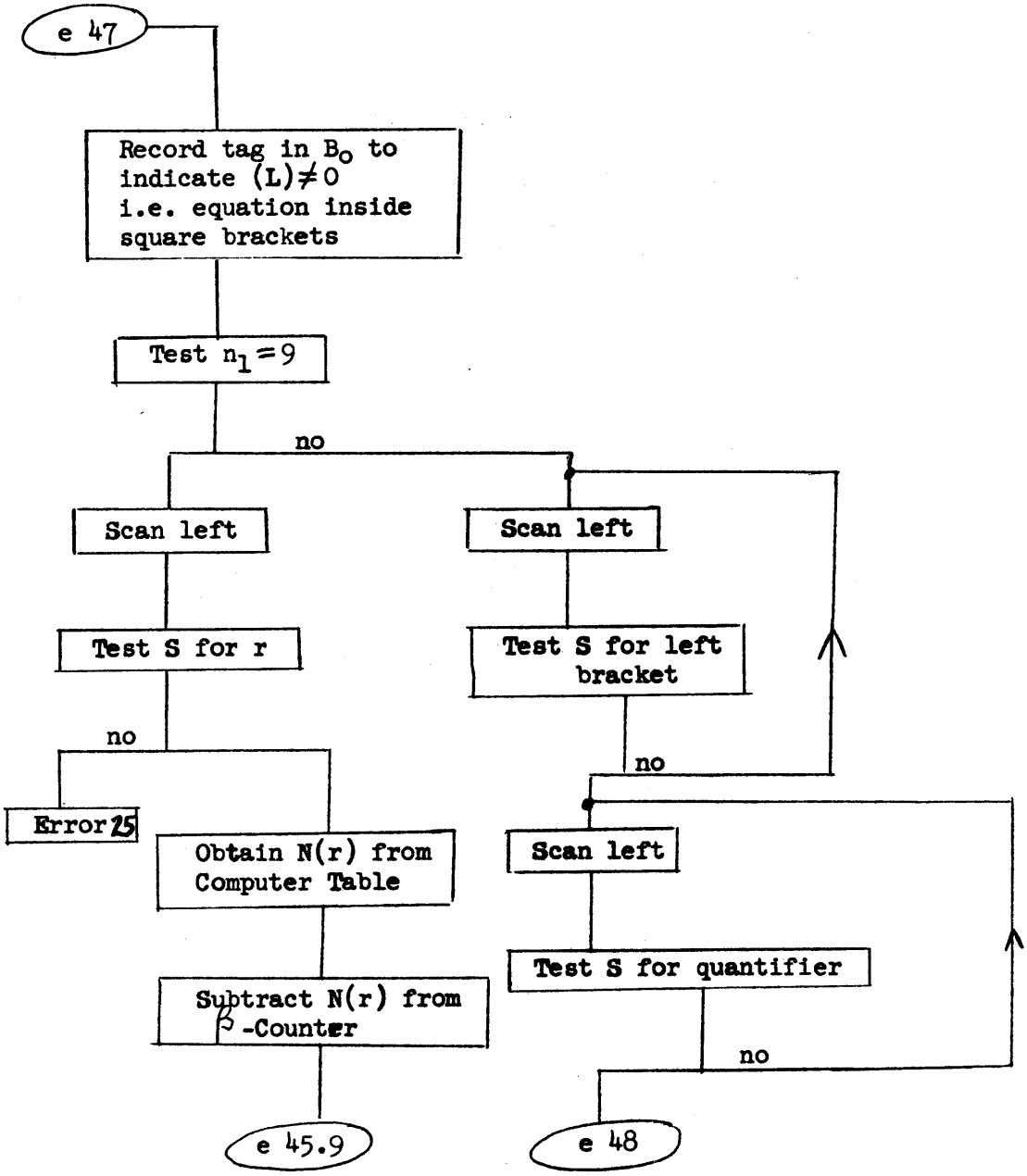

e 42.4
$B_i \rightarrow B_{i+1}$
Record "End Vector Recursion" mark in B^o
e 42.5
Scan left
Test S for =
=
no
$B_i \rightarrow B_{i+1}$
#(=) $\rightarrow B_o$
e 42.5
Test S for quantifier
no
e 42.5
$B_i \rightarrow B_{i+1}$
Record "Start Vector Recursion" mark in B_o
Scan Rt.
e 42.6
rec. 4
e 42.3
Subtract $\overline{M}$ from β-counter
Obtain α_s from β-counter. Record α_s in second cell left of =. Obtain α_π from I. C. Record α_π in first cell left of =.
e 42.2
rec. 2

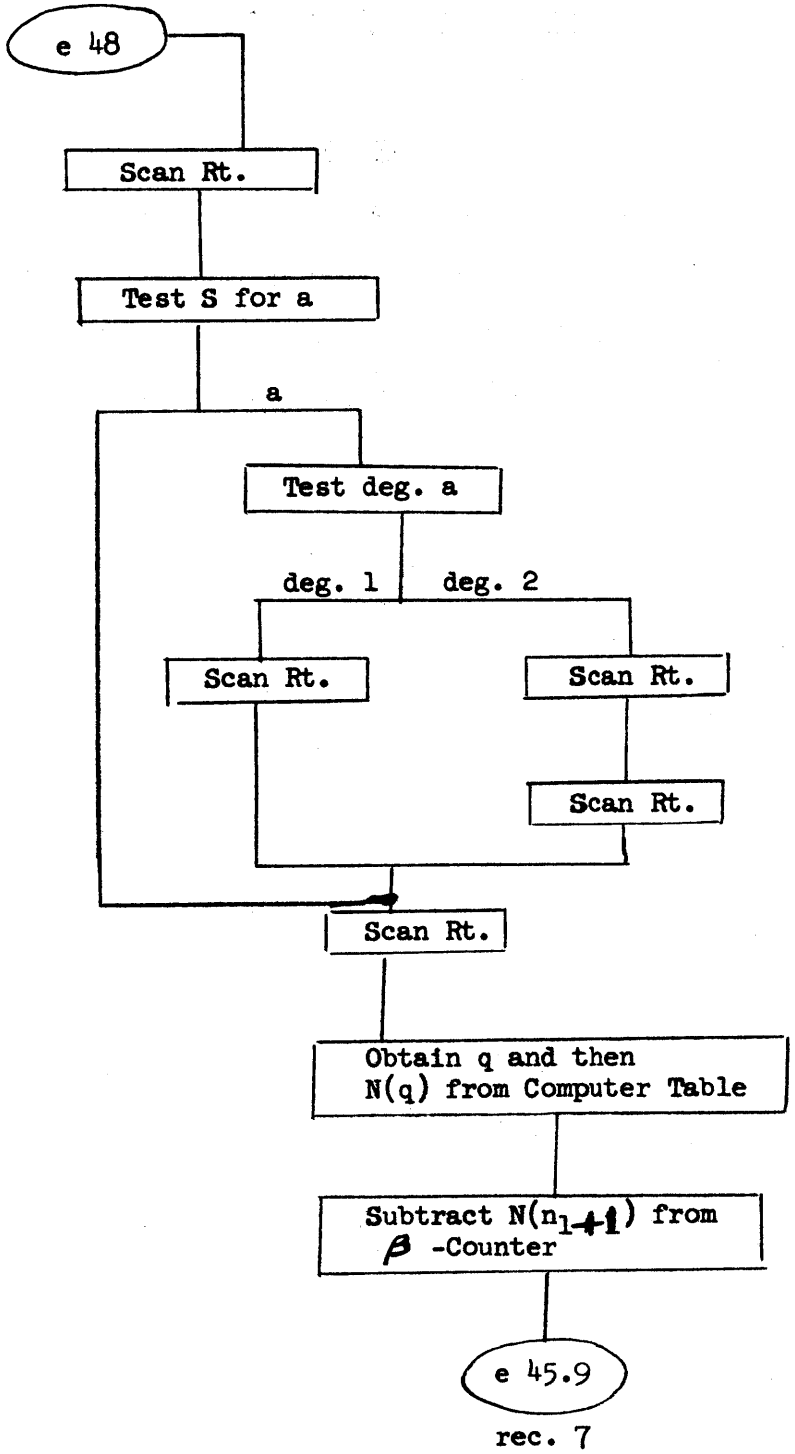

e 42.6
Test S for a
no
Test deg. a
deg. 1
deg. 2
Scan Rt.
Scan Rt.
Scan Rt.
Record: q P_{o1}(m+2) in TAI
Scan Rt.
Test S for rt. parenthesis
no
e 43
Test (Sc)=(P)
no
e 43.1
e 43
Scan left
Test S for comma
no
e 43
e 43.1
Scan left
Test S for comma
no
Test S for =
$B_i \rightarrow B_{i+1}$
no
e 43.1
#(comma) $\rightarrow B_o$
e 43.1
e 43.2
Scan left
Test S for quantifier
no
$B_o \rightarrow$ Sc
Test S for comma
e 4.2
P. b 1
no
e 43.1
e 43.2


e 45
Test n_2=2
n_2=3
n_2=2
Triple recursion
see T.R.1
Scan left
Test S for left parenthesis
no
Scan left
Test S for quantifier
no
Scan rt.
Test S for a
a
Obtain degree of a
deg. 1
deg. 2
Scan Rt.
Scan Rt.
Scan Rt.
Scan Rt.
S → D
i.e. q → D
e 45.4
rec. 6


rec. 6
e 45.4
Scan Rt.
Test S for rt. parenthesis
no
e 45.4
$B_i \rightarrow B_{i+1}$
Record "End Double Recursion" Mark in B_o
e 45.1
Scan left
Test S for =
no
Test S for Rt. Bracket
no
Test S for left bracket
no
e 45.2
0 → L
e 45.1
Non-zero → L
$B_i \rightarrow B_{i+1}$
Record mark in B_o to indicate rt. bracket
(=) → Sc_1
$B_i \rightarrow B_{i+1}$
$(Sc_1) \rightarrow B_o$
e 46
rec. 7
e 45.2
Test S for left parenthesis
no
e 45.1
$B_o \rightarrow$ Sc
$B_o \rightarrow$ E
e 1.1
P. b1


e 46
Obtain α_s from β-Counter. Record α_s in second cell to left of equal sign. Obtain α_π from I. C. and record in cell to left of equal sign
Test (L)=0
no
e 47
rec. 8
Record tag in B_o to indicate (L)=0 i.e. equation is outside square brackets
Test n_1=9
no
Subtract n_1+1 from β-Counter
e 45.9
$Sc_1 \rightarrow Sc$
e 45.1
rec. 6
Obtain q from D and N (q) from Computer Table
Subtract N(q) from β-Counter.
e 45.9


e 47
Record tag in B_o to indicate $(L) \neq 0$ i.e. equation inside square brackets
Test $n_1 = 9$
no
Scan left
Test S for r
no
Error 25
Obtain N(r) from Computer Table
Subtract N(r) from β-Counter
e 45.9
rec. 7
Scan left
Test S for left bracket
no
Scan left
Test S for quantifier
no
e 48
rec. 9

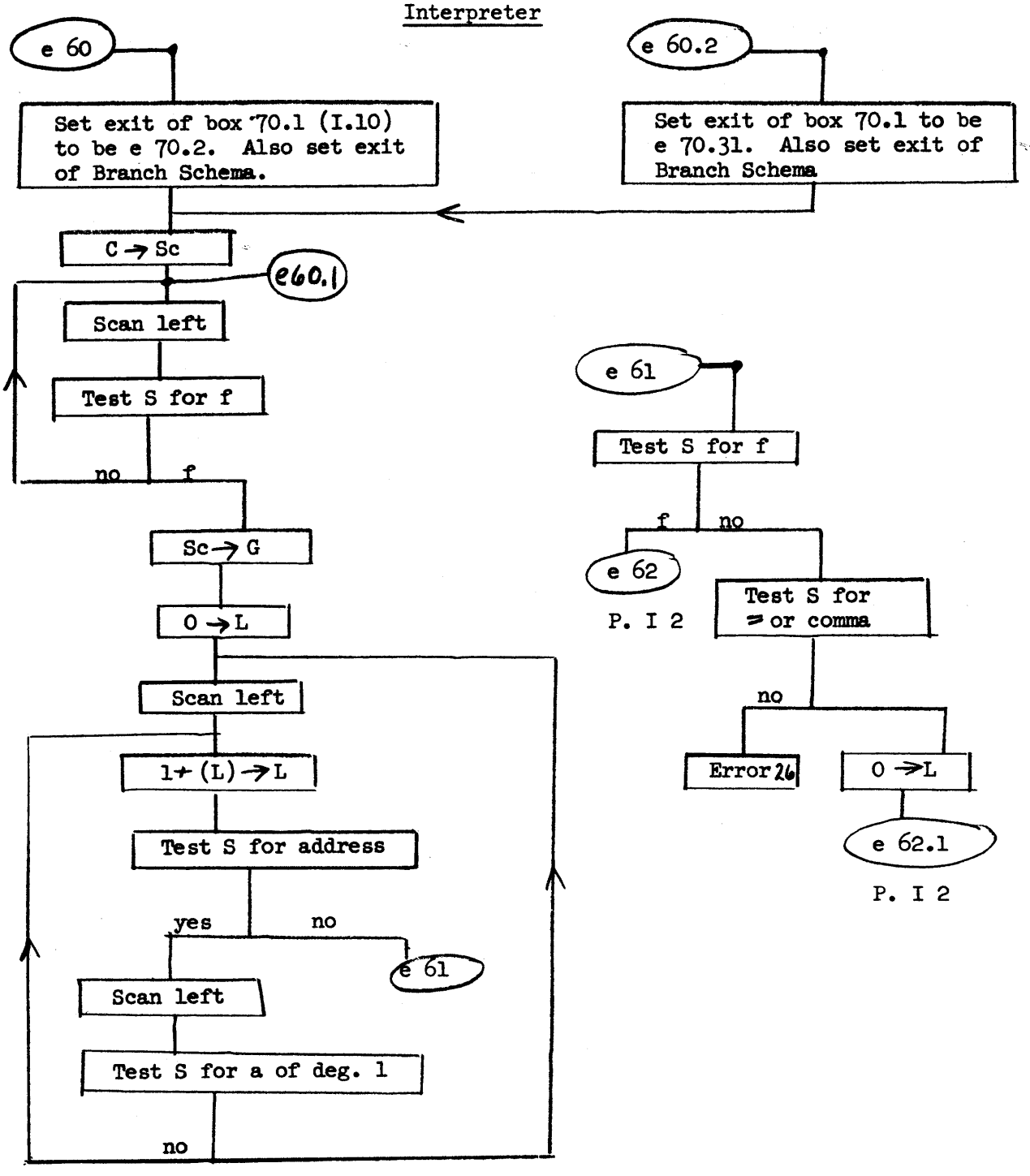

e 48
Scan Rt.
Test S for a
a
Test deg. a
deg. 1
deg. 2
Scan Rt.
Scan Rt.
Scan Rt.
Scan Rt.
Obtain q and then N(q) from Computer Table
Subtract N(n1+1) from β -Counter
e 45.9
rec. 7

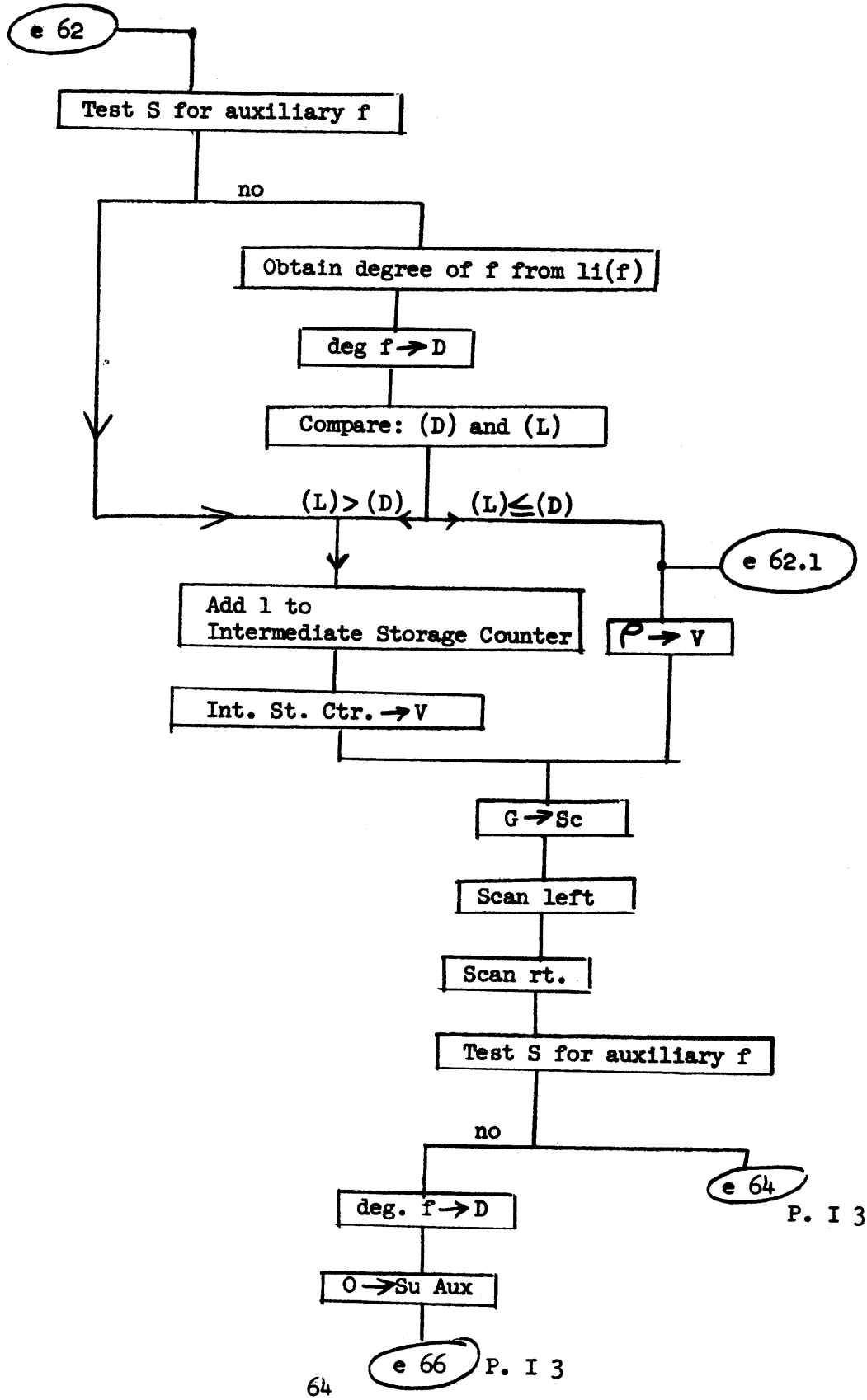
Explanatory Notes; Recursion Schema

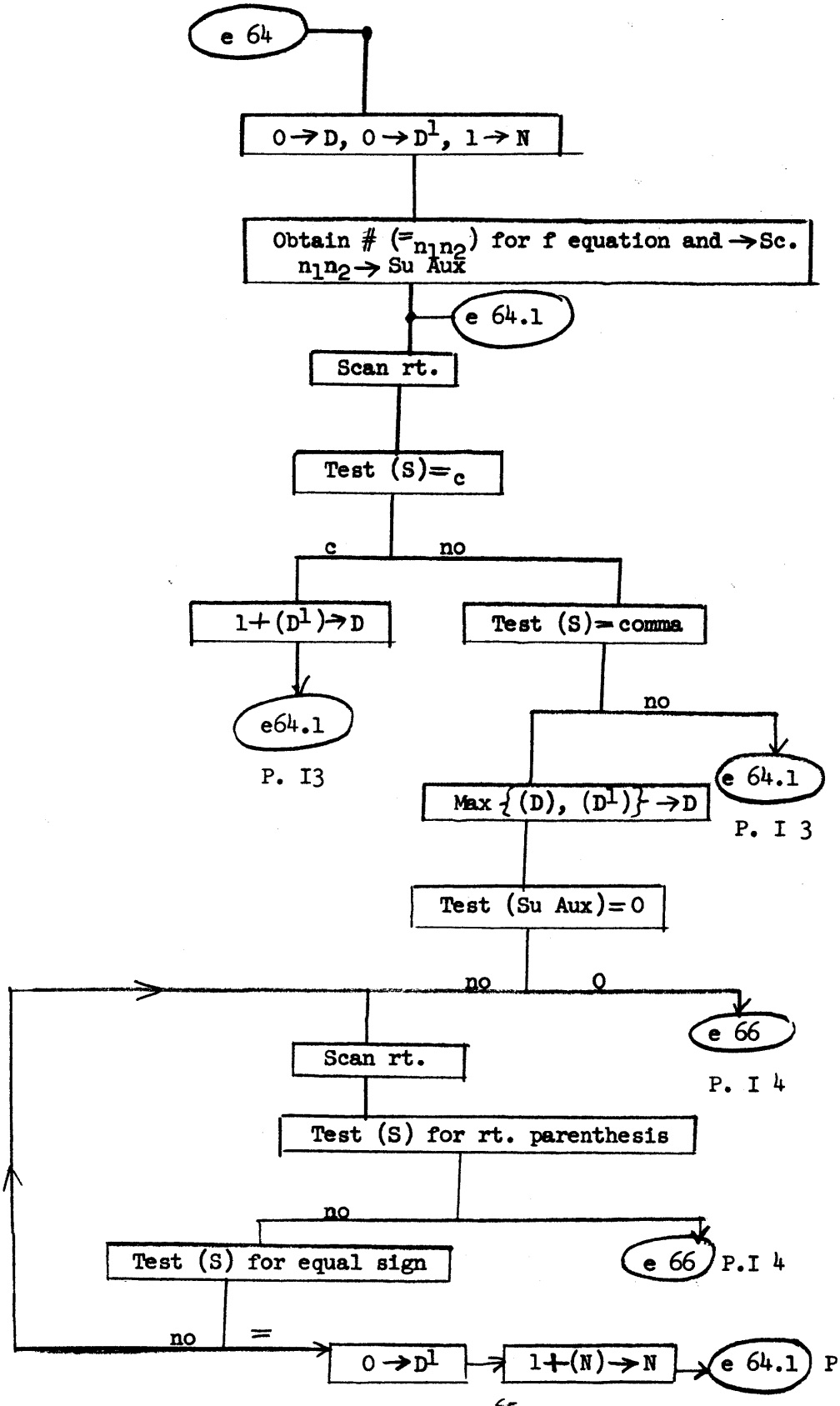
1. The following standard cells are used:

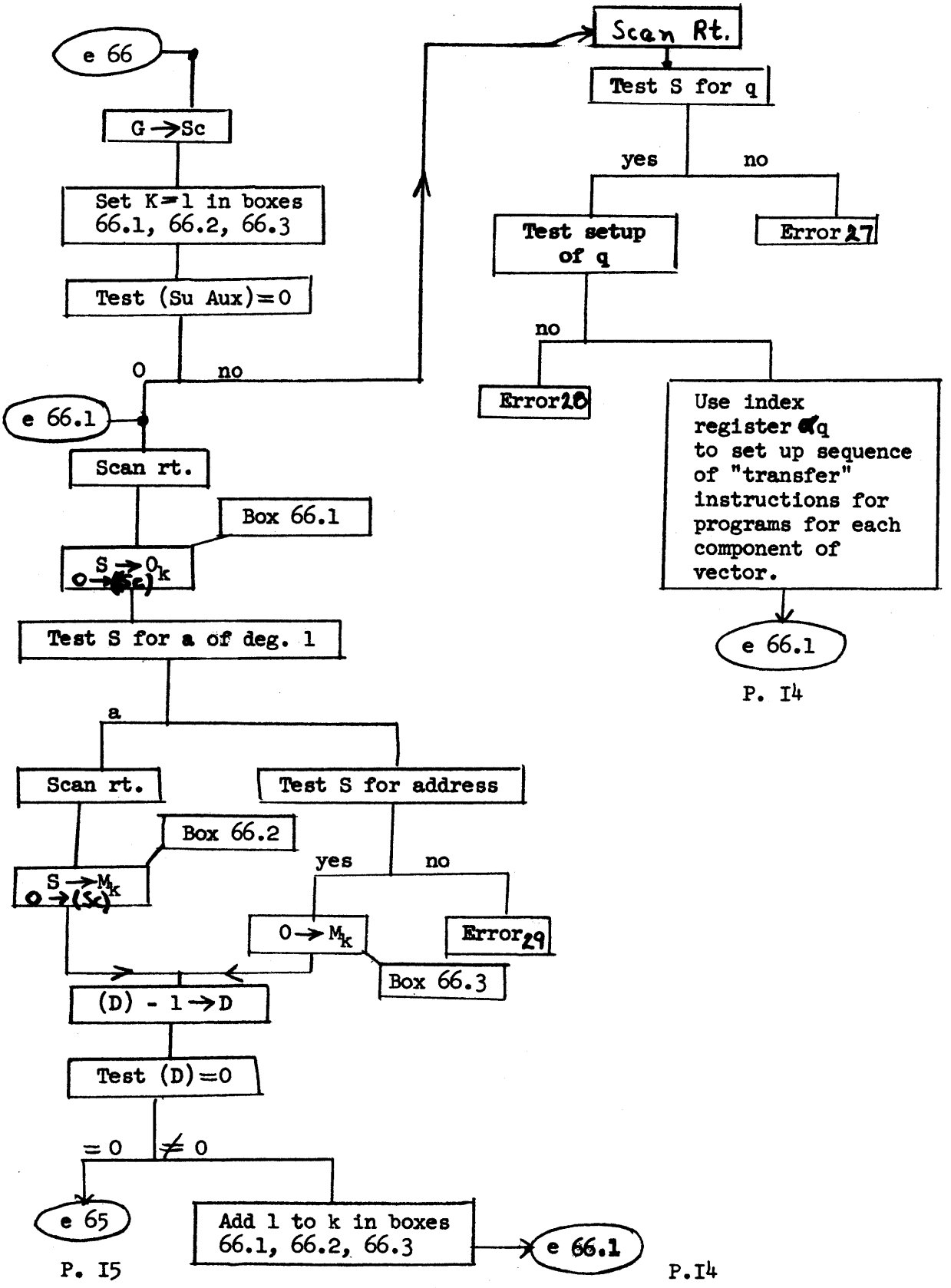
Cell		Contents
P	...	Cell number of last comma in a scalar recursion.
Sub	...	Subscript of equal sign in recursion equation.
M	...	Number of initial values plus one, or N (q) where q is recursion index in course-of-values recursion and N(q) is number of values which q will take on.

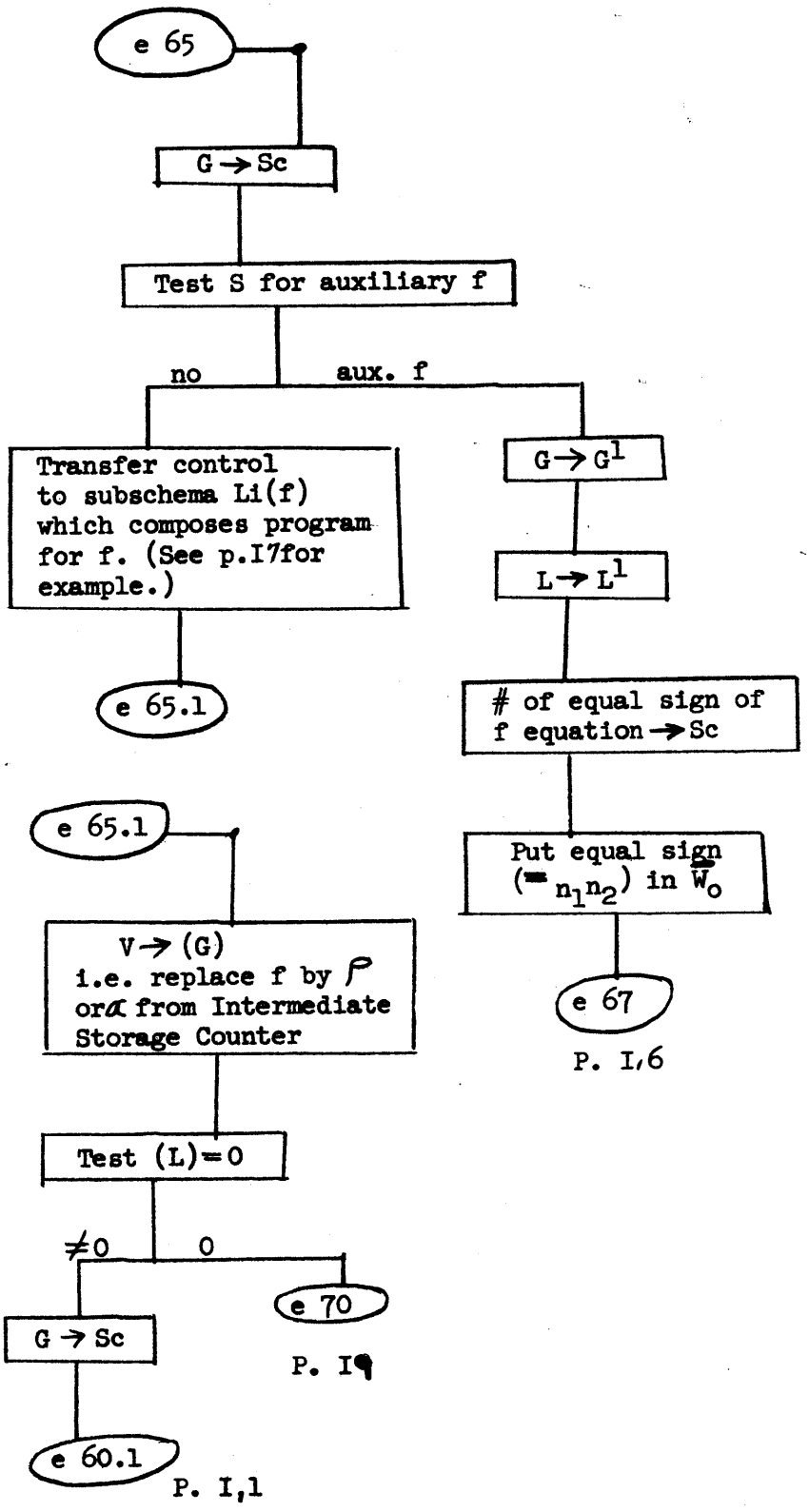
Interpreter

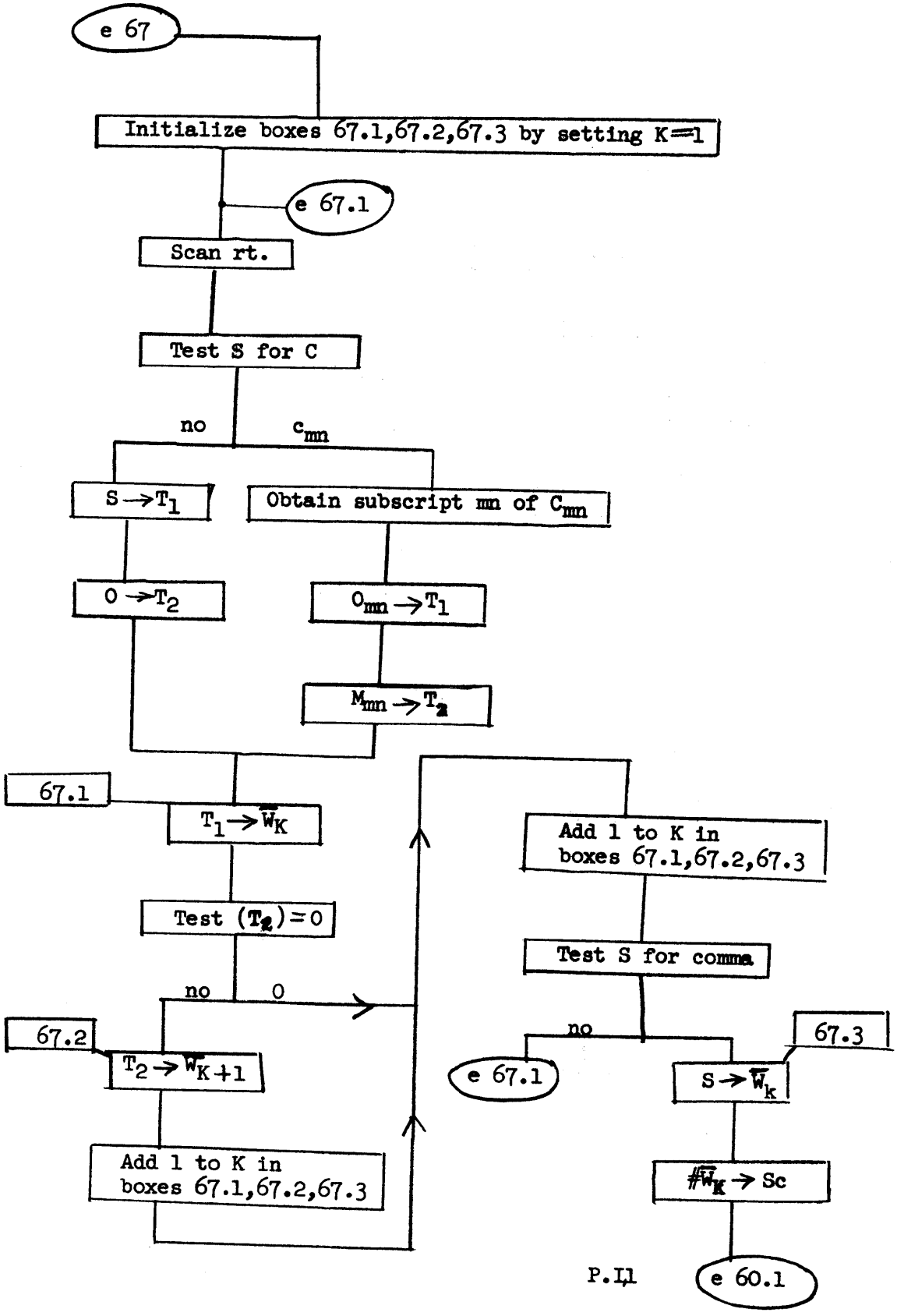



e 62
Test S for auxiliary f
no
Obtain degree of f from li(f)
deg f → D
Compare: (D) and (L)
(L) > (D)
(L) ≤ (D)
e 62.1
Add 1 to
Intermediate Storage Counter
P → V
Int. St. Ctr. → V
G → Sc
Scan left
Scan rt.
Test S for auxiliary f
no
e 64
P. I 3
deg. f → D
0 → Su Aux
e 66
P. I 3


e 64
0→D, 0→D1, 1→N
Obtain # (=n1n2) for f equation and →Sc.
n1n2→ Su Aux
e 64.1
Scan rt.
Test (S)=c
c
no
1+(D1)→D
Test (S)=comma
e64.1
P. I3
no
e 64.1
Max{(D), (D1)}→D
P. I 3
Test (Su Aux)=0
no
0
e 66
P. I 4
Scan rt.
Test (S) for rt. parenthesis
no
Test (S) for equal sign
e 66
P.I 4
no
=
0→D1
1+(N)→N
e 64.1
P. I 3

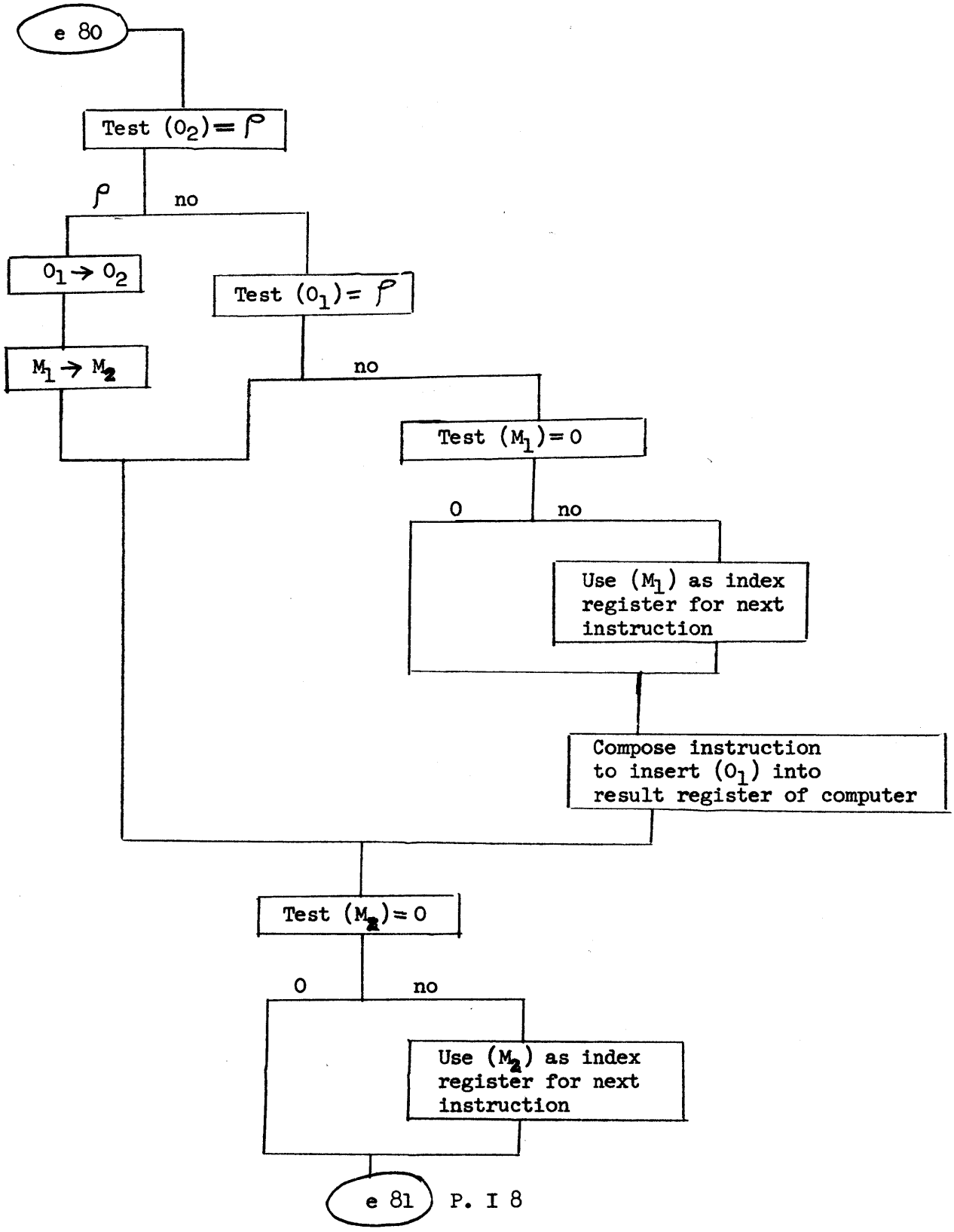

e 66
G → Sc
Set K=1 in boxes 66.1, 66.2, 66.3
Test (Su Aux)=0
0
no
e 66.1
Scan rt.
Box 66.1
S → 0k
0 → (Sc)
Test S for a of deg. 1
a
Scan rt.
Test S for address
Box 66.2
S → Mk
0 → (Sc)
yes
no
0 → Mk
Error29
Box 66.3
(D) - 1 → D
Test (D)=0
= 0
≠ 0
e 65
P. I5
Add 1 to k in boxes 66.1, 66.2, 66.3
e 66.1
P. I4
Scan Rt.
Test S for q
yes
no
Test setup of q
Error27
no
Error28
Use index register q to set up sequence of "transfer" instructions for programs for each component of vector.
e 66.1
P. I4

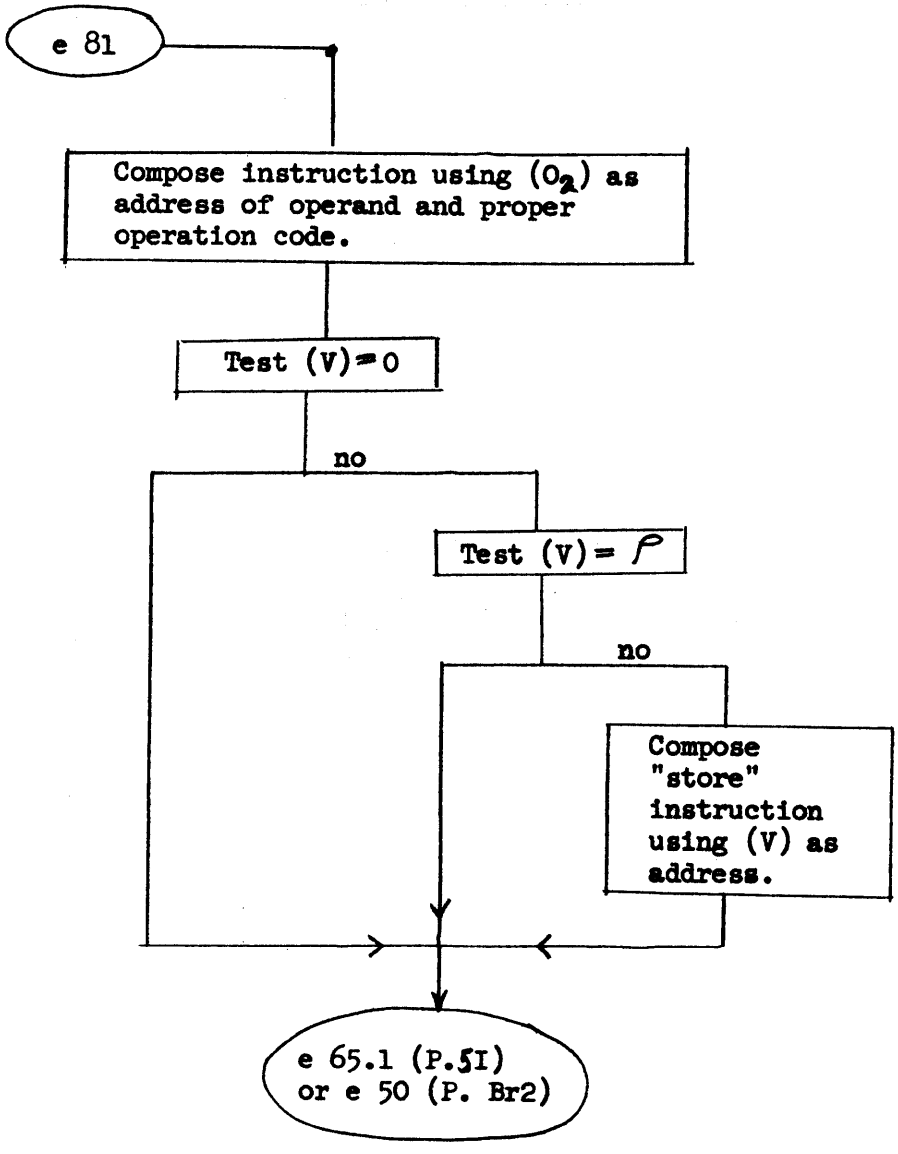

e 65
G → Sc
Test S for auxiliary f
no
aux. f
Transfer control to subschema Li(f) which composes program for f. (See p.I7for example.)
e 65.1
G → G1
L → L1
# of equal sign of f equation → Sc
Put equal sign (= n1n2) in W̄o
e 67
P. I,6
e 65.1
V → (G) i.e. replace f by ρ or α from Intermediate Storage Counter
Test (L)=0
≠0
0
e 70
P. I9
G → Sc
e 60.1
P. I,1

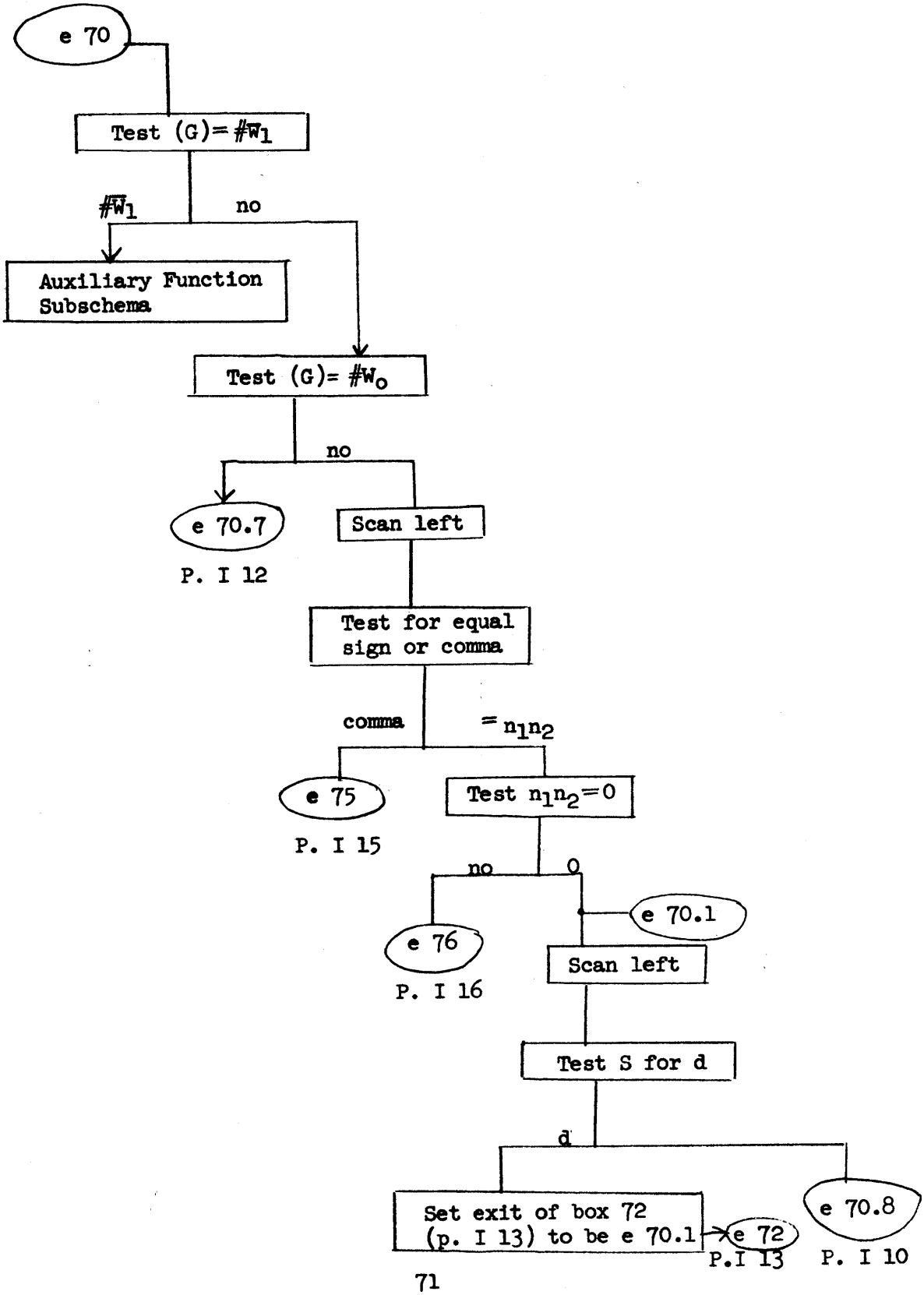

e 67
Initialize boxes 67.1,67.2,67.3 by setting K=1
e 67.1
Scan rt.
Test S for C
no
c_{mn}
S→T_1
Obtain subscript mn of C_{mn}
0→T_2
0_{mn}→T_1
M_{mn}→T_2
67.1
T_1→$\overline{W}_K$
Test (T_2)=0
no
0
67.2
T_2→$\overline{W}_{K+1}$
Add 1 to K in boxes 67.1,67.2,67.3
Add 1 to K in boxes 67.1,67.2,67.3
Test S for comma
no
67.3
e 67.1
S→$\overline{W}_k$
#$\overline{W}_K$→Sc
P.I1
e 60.1

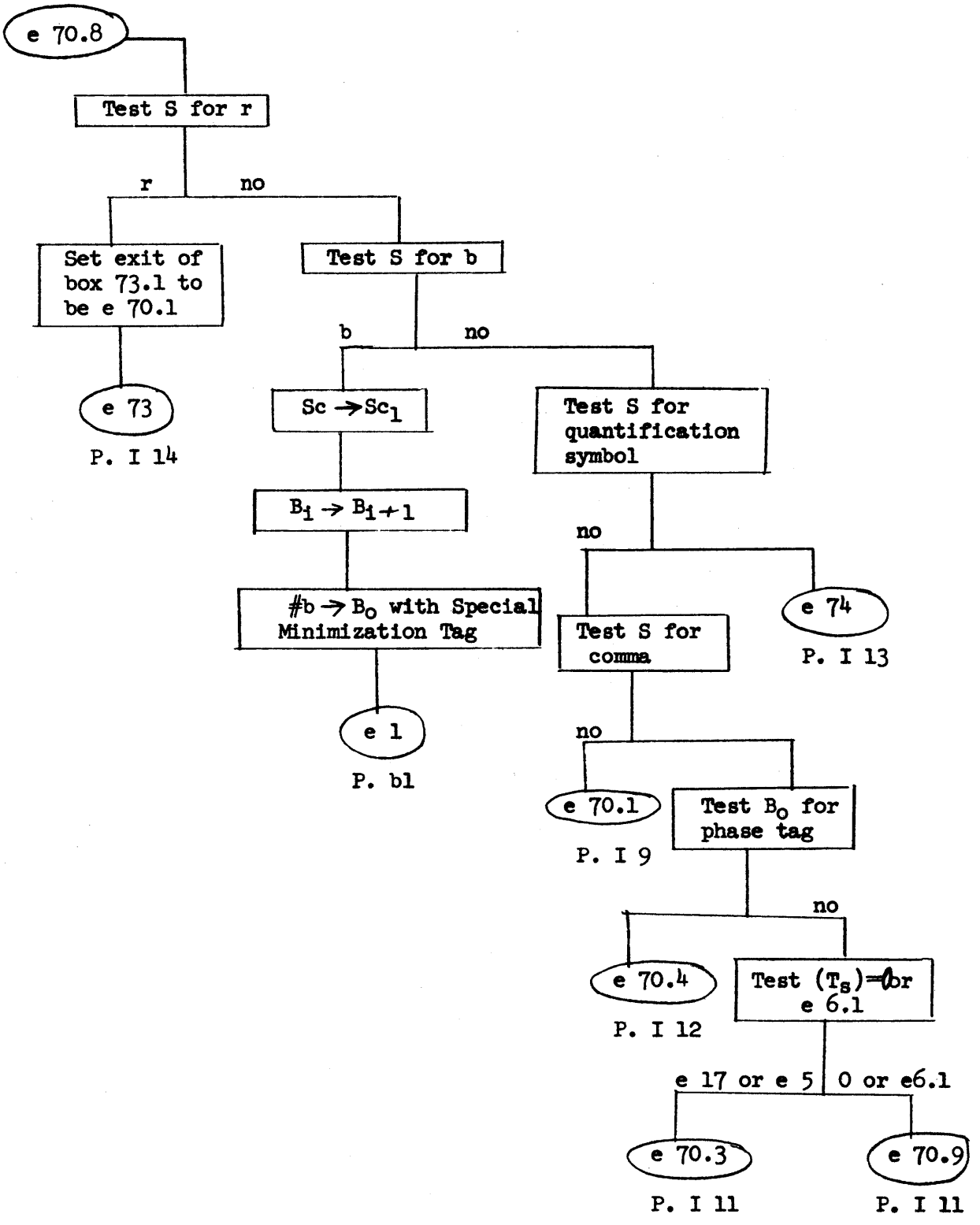
NAVORD Report 4411

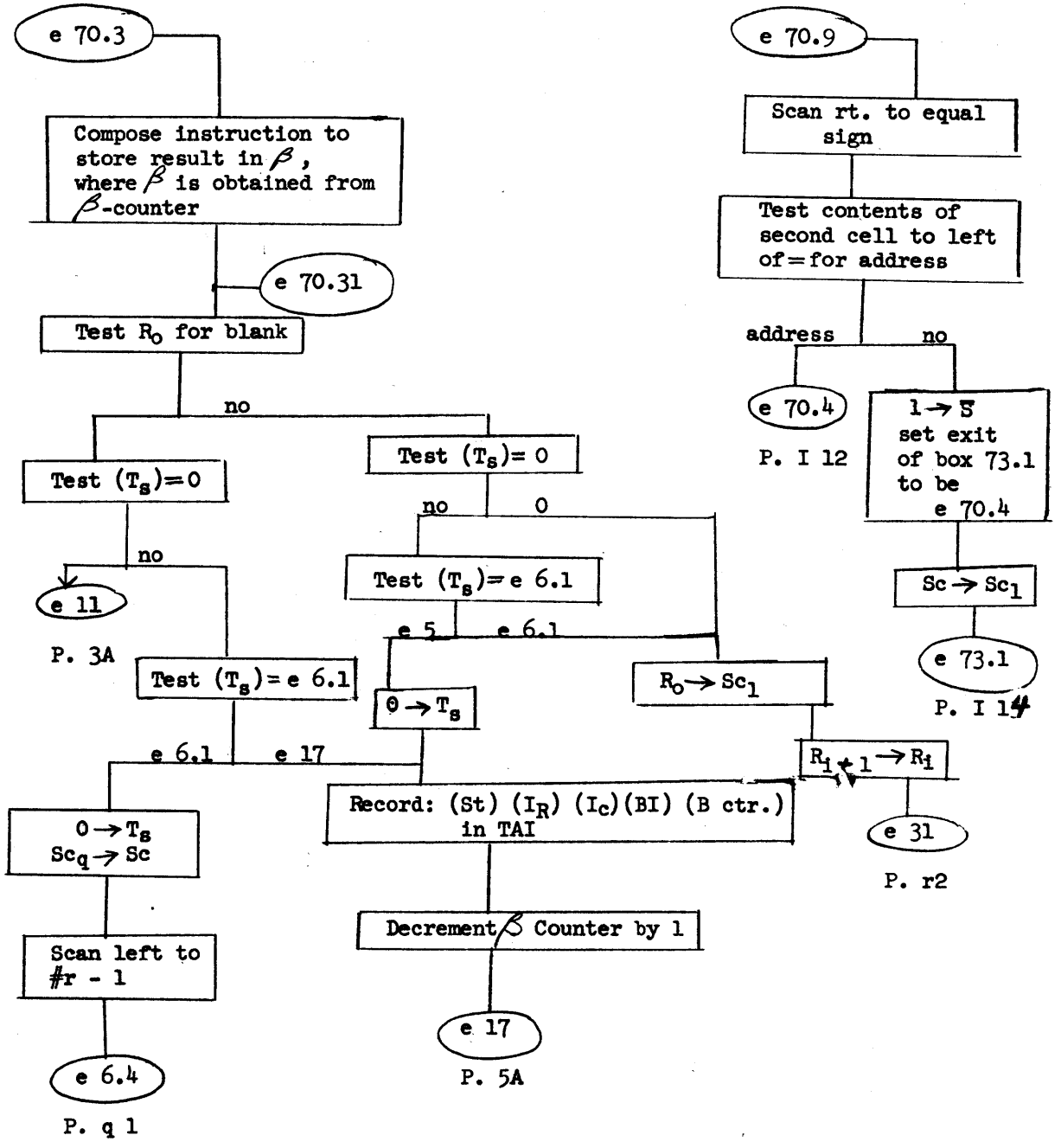
Li(f) for Arithmetic Operations

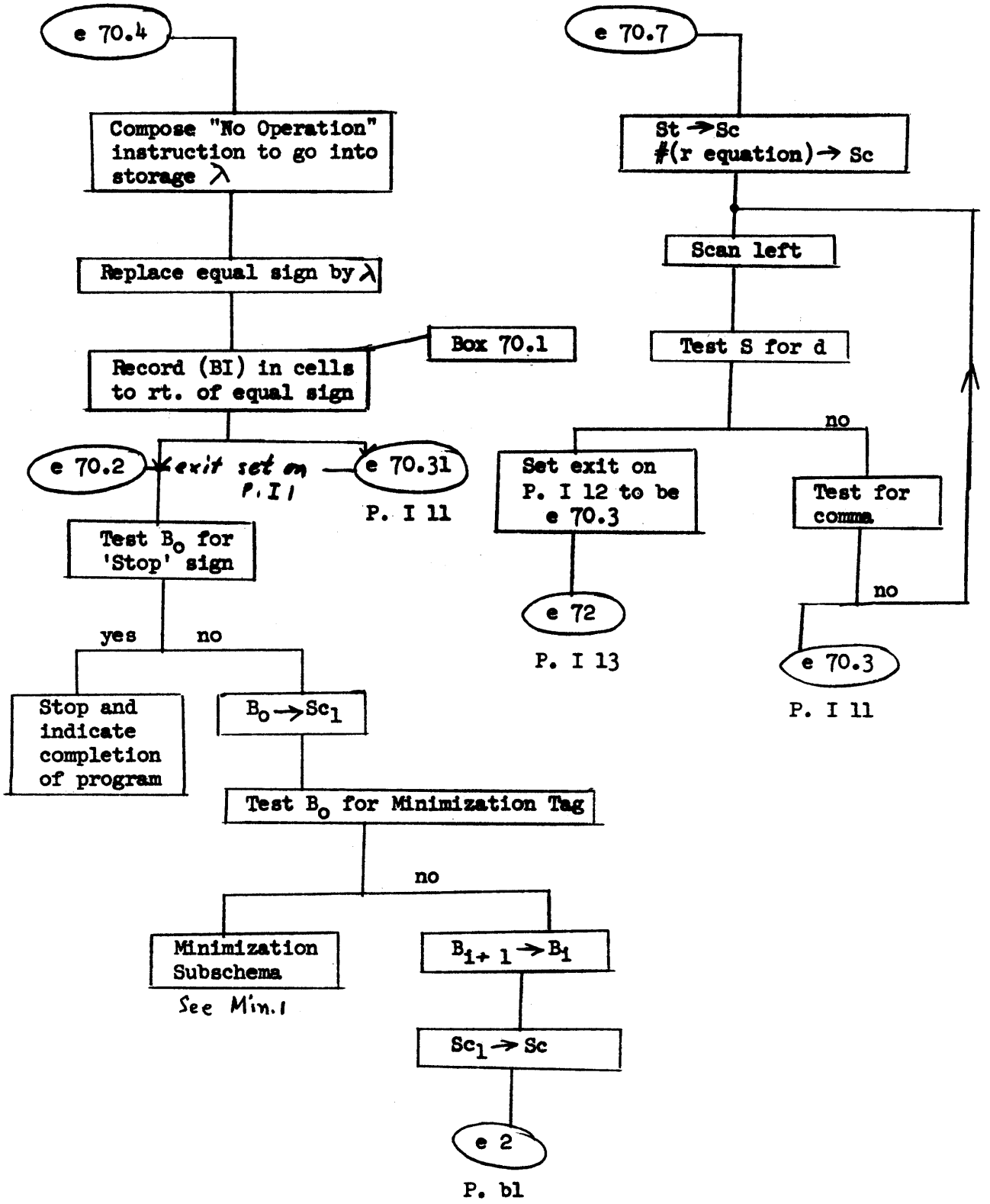


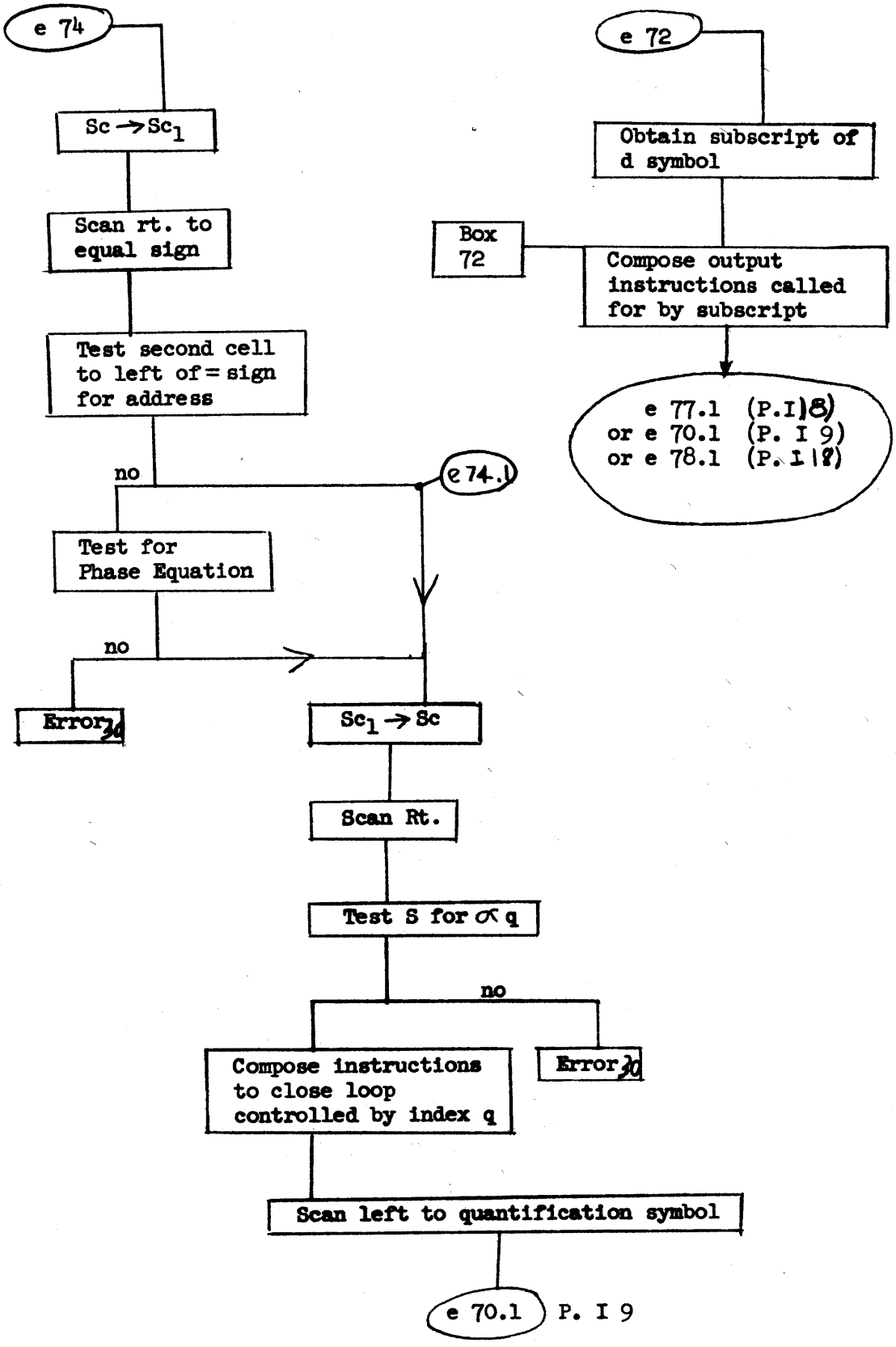

e 81
Compose instruction using (O_2) as address of operand and proper operation code.
Test (V)=0
no
Test (V)= P
no
Compose "store" instruction using (V) as address.
e 65.1 (P.5I) or e 50 (P. Br2)

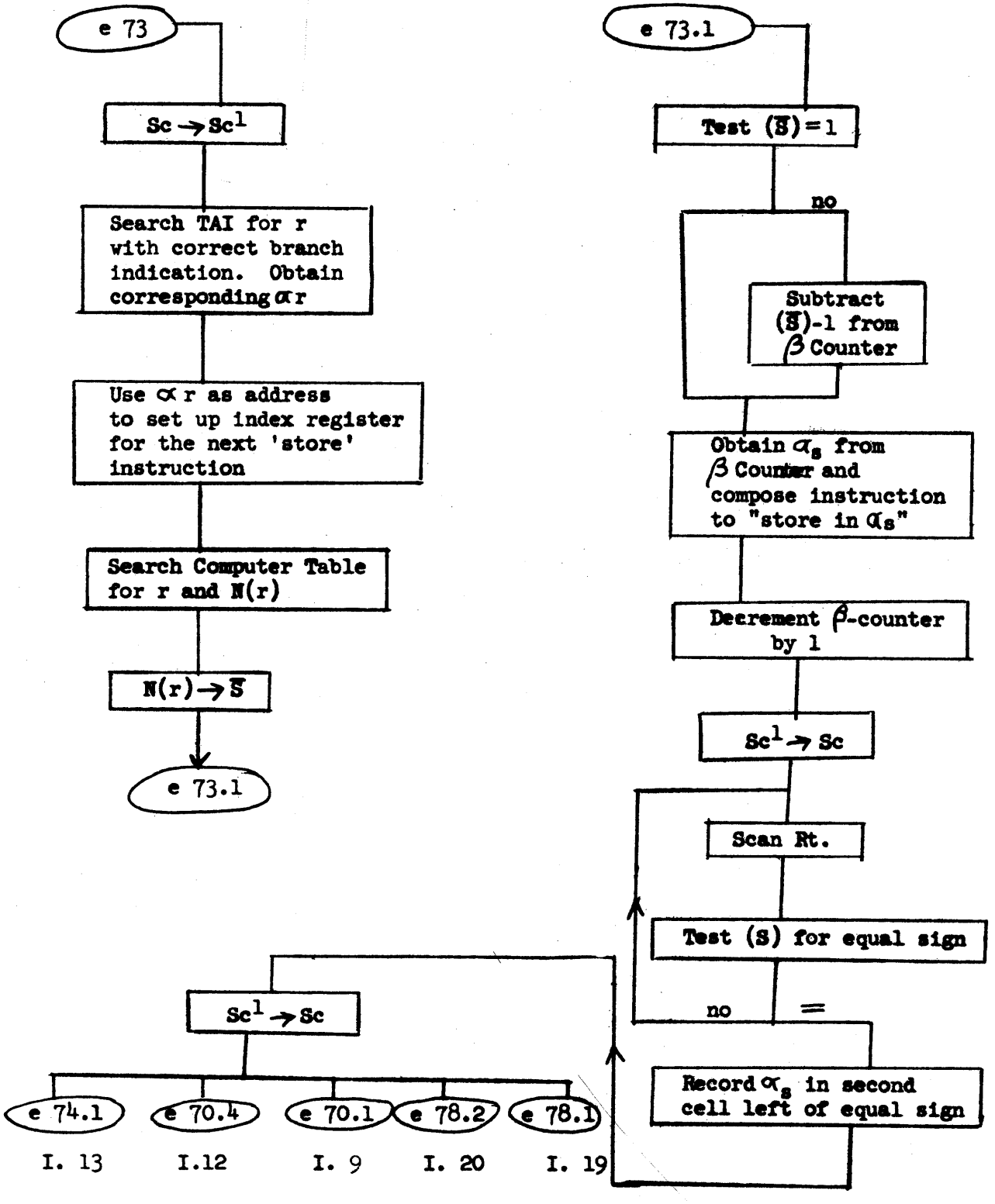

e 70
Test (G)= #W1
#W1
no
Auxiliary Function Subschema
Test (G)= #W0
no
e 70.7
P. I 12
Scan left
Test for equal sign or comma
comma
= n1n2
e 75
P. I 15
Test n1n2=0
no
0
e 76
P. I 16
e 70.1
Scan left
Test S for d
d
Set exit of box 72 (p. I 13) to be e 70.1
e 72
P.I 13
e 70.8
P. I 10

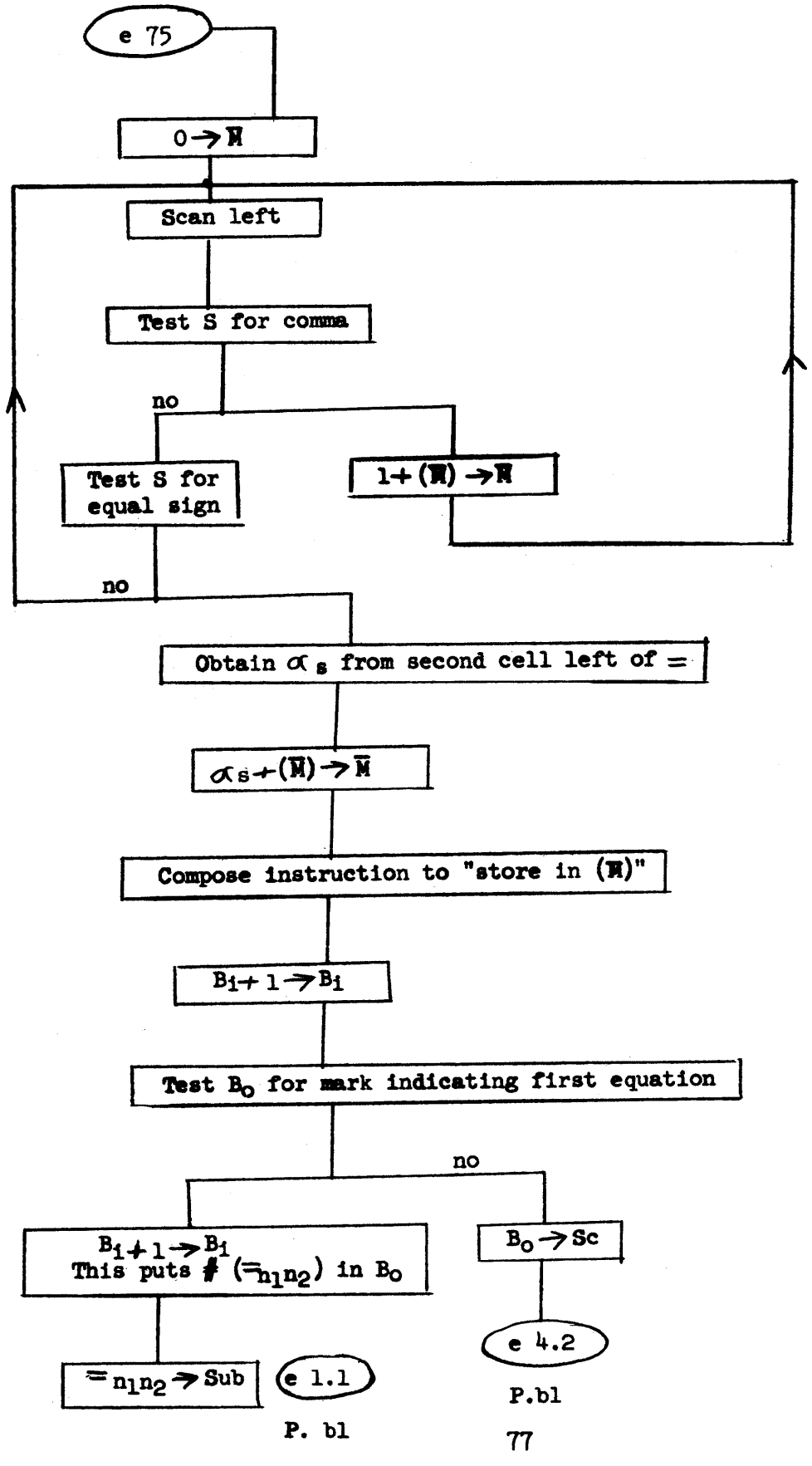

e 70.8
Test S for r
r
no
Set exit of box 73.1 to be e 70.1
e 73
P. I 14
Test S for b
b
no
Sc → Sc1
Bi → Bi+1
#b → B0 with Special Minimization Tag
e 1
P. bl
Test S for quantification symbol
no
Test S for comma
e 74
P. I 13
no
e 70.1
P. I 9
Test B0 for phase tag
no
e 70.4
P. I 12
Test (Ts)=0 or e 6.1
e 17 or e 5
0 or e6.1
e 70.3
P. I 11
e 70.9
P. I 11

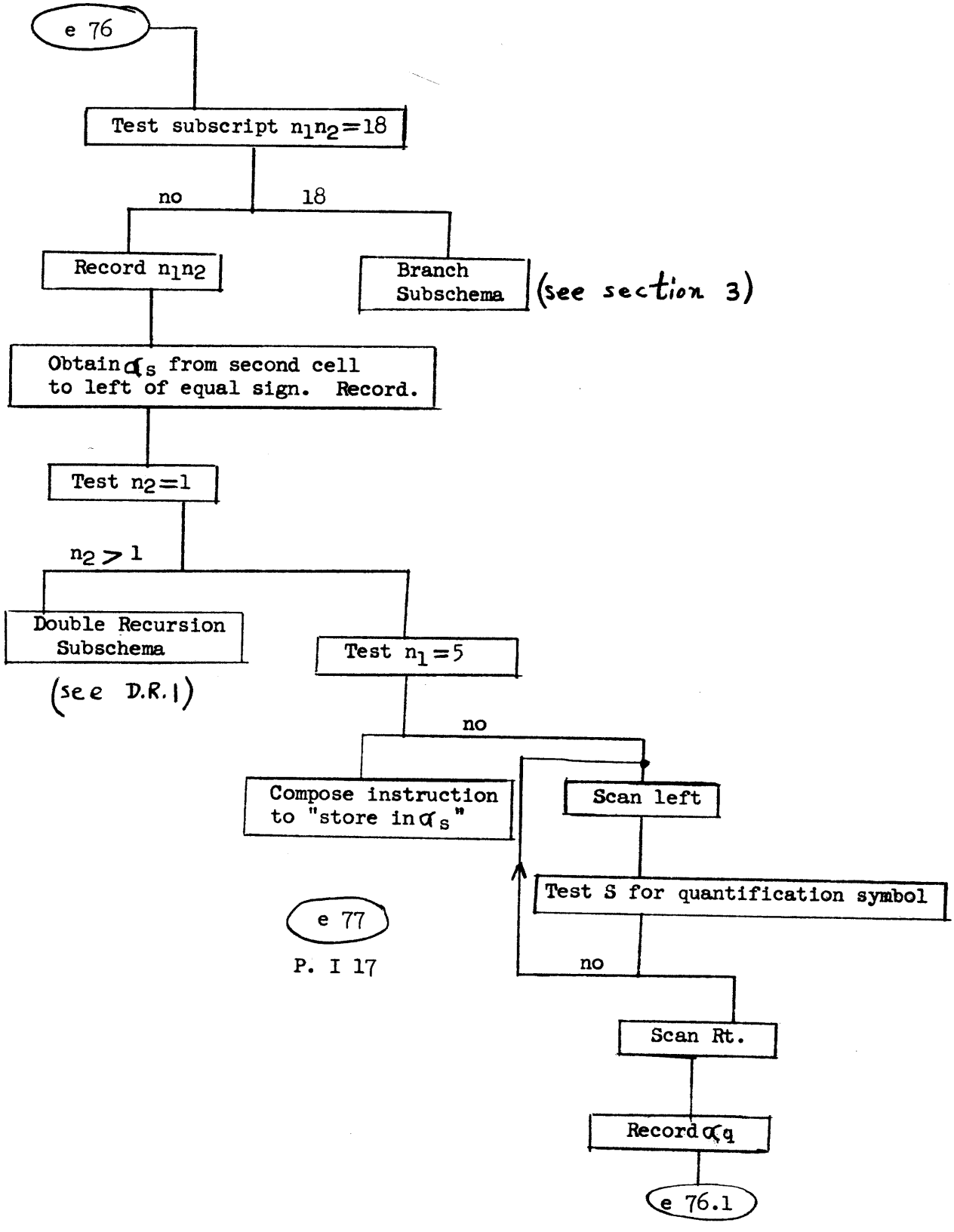

e 70.3
Compose instruction to store result in β, where β is obtained from β-counter
e 70.31
Test R_0 for blank
no
Test $(T_s)=0$
no
e 11
P. 3A
Test $(T_s)=e$ 6.1
e 6.1
e 17
0 → T_s
Sc_q → Sc
Scan left to #r - 1
e 6.4
P. q 1
Test $(T_s)=0$
no
0
Test $(T_s)=e$ 6.1
e 5
e 6.1
0 → T_s
Record: (St) (I_R) (I_c)(BI) (B ctr.) in TAI
Decrement β Counter by 1
e 17
P. 5A
e 70.9
Scan rt. to equal sign
Test contents of second cell to left of = for address
address
no
e 70.4
P. I 12
R_0 → Sc_1
$R_i + 1$ → R_i
e 31
P. r2
1 → S set exit of box 73.1 to be e 70.4
Sc → Sc_1
e 73.1
P. I 14

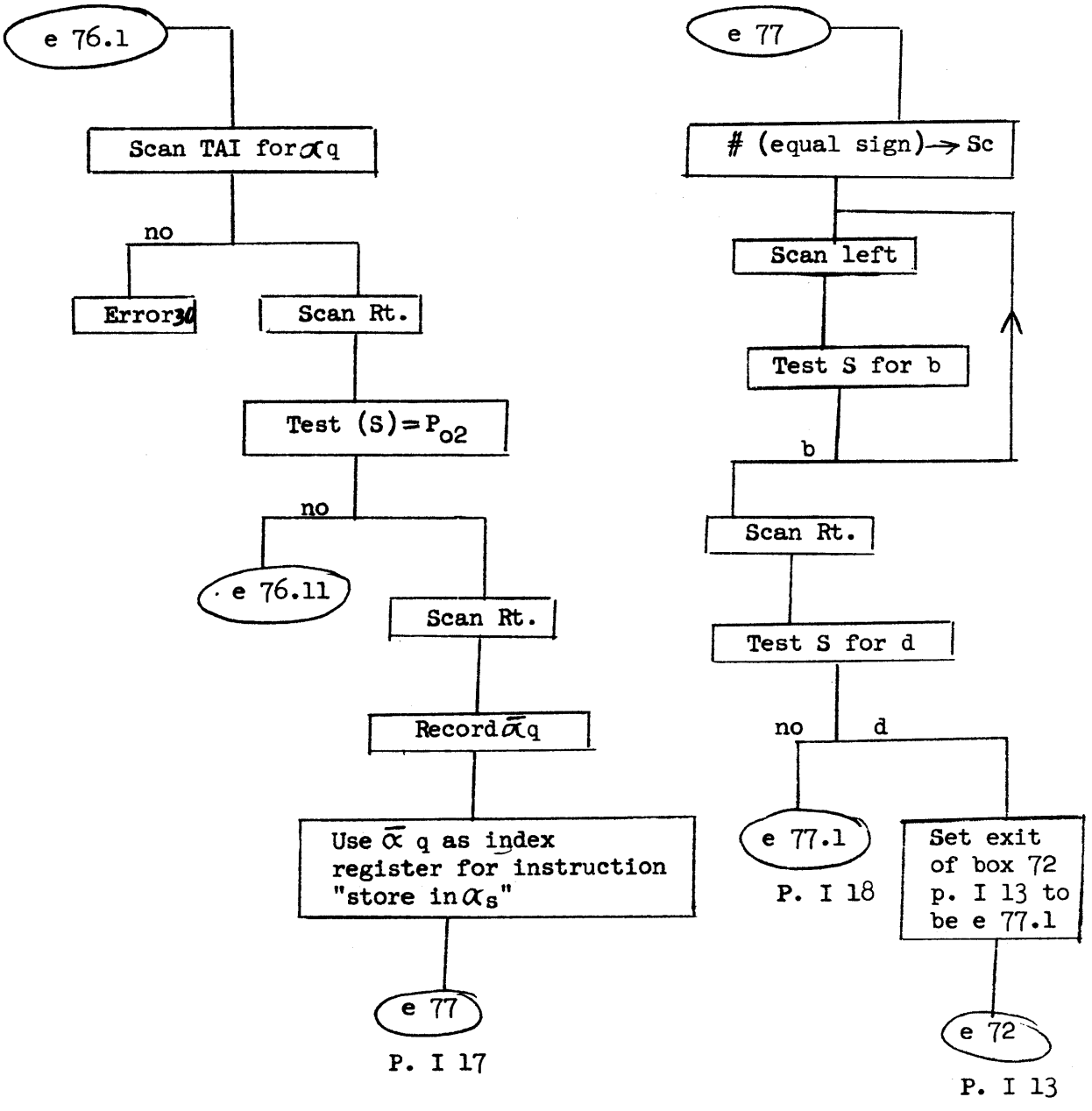

e 70.4
Compose "No Operation" instruction to go into storage λ
Replace equal sign by λ
Record (BI) in cells to rt. of equal sign
Box 70.1
e 70.2
exit set on P. I 1
e 70.31
P. I 11
Test B_0 for 'Stop' sign
yes
no
Stop and indicate completion of program
$B_0 \rightarrow Sc_1$
Test B_0 for Minimization Tag
no
Minimization Subschema
See Min. 1
$B_{i+1} \rightarrow B_i$
$Sc_1 \rightarrow Sc$
e 2
P. b1
e 70.7
St → Sc
#(r equation) → Sc
Scan left
Test S for d
no
Set exit on P. I 12 to be e 70.3
e 72
P. I 13
Test for comma
no
e 70.3
P. I 11

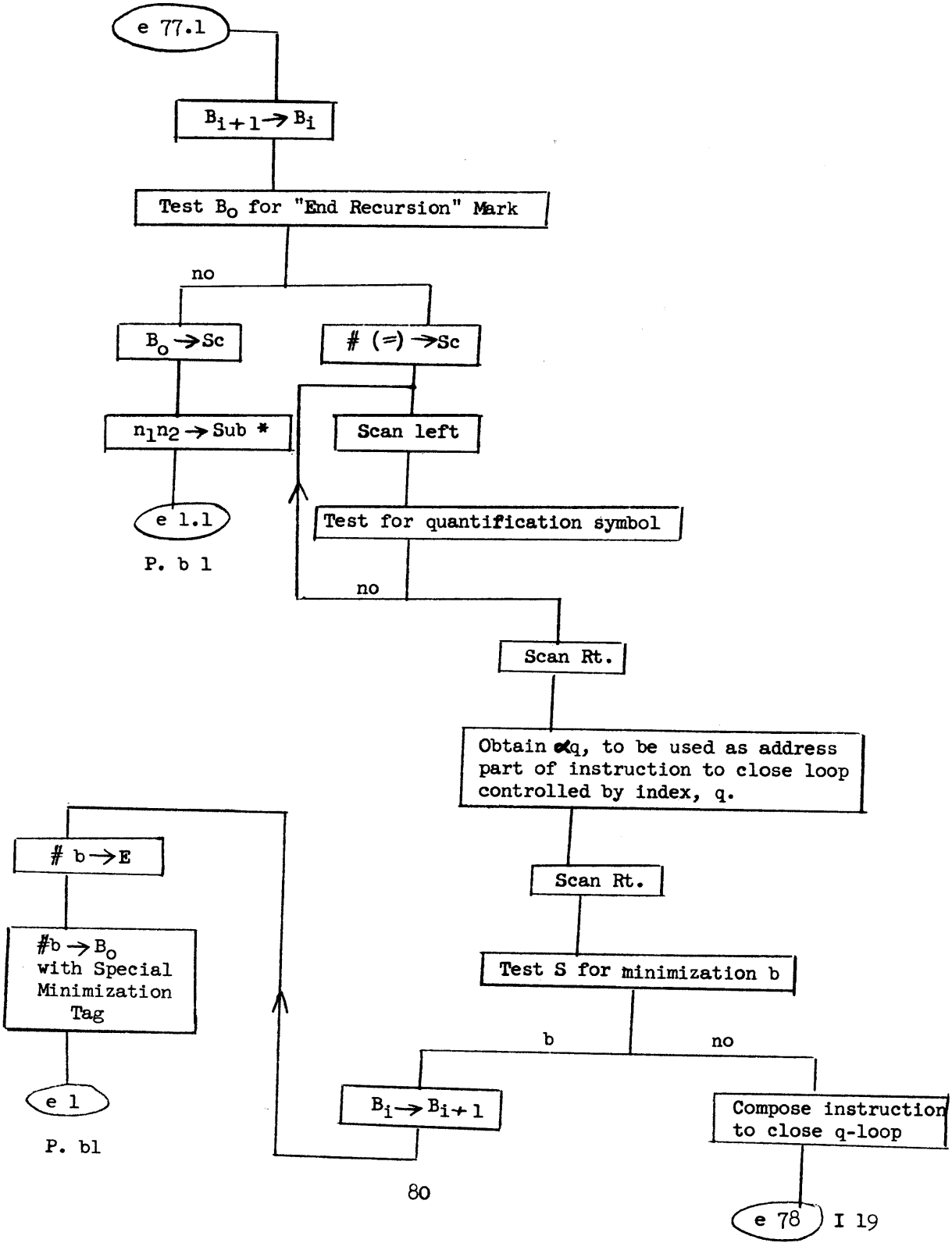

e 74
Sc → Sc1
Scan rt. to equal sign
Test second cell to left of = sign for address
no
e 74.1
Test for Phase Equation
no
Error 30
Sc1 → Sc
Scan Rt.
Test S for σ q
no
Compose instructions to close loop controlled by index q
Error 30
Scan left to quantification symbol
e 70.1
P. I 9
e 72
Obtain subscript of d symbol
Box 72
Compose output instructions called for by subscript
e 77.1 (P.I 8)
or e 70.1 (P. I 9)
or e 78.1 (P. I 8)


e 73
Sc → Sc1
Search TAI for r with correct branch indication. Obtain corresponding αr
Use αr as address to set up index register for the next 'store' instruction
Search Computer Table for r and N(r)
N(r) → S̄
e 73.1
e 73.1
Test (S̄) = 1
no
Subtract (S̄)-1 from β Counter
Obtain αs from β Counter and compose instruction to "store in αs"
Decrement β-counter by 1
Sc1 → Sc
Scan Rt.
Test (S) for equal sign
no
=
Record αs in second cell left of equal sign
Sc1 → Sc
e 74.1
e 70.4
e 70.1
e 78.2
e 78.1
I. 13
I.12
I. 9
I. 20
I. 19

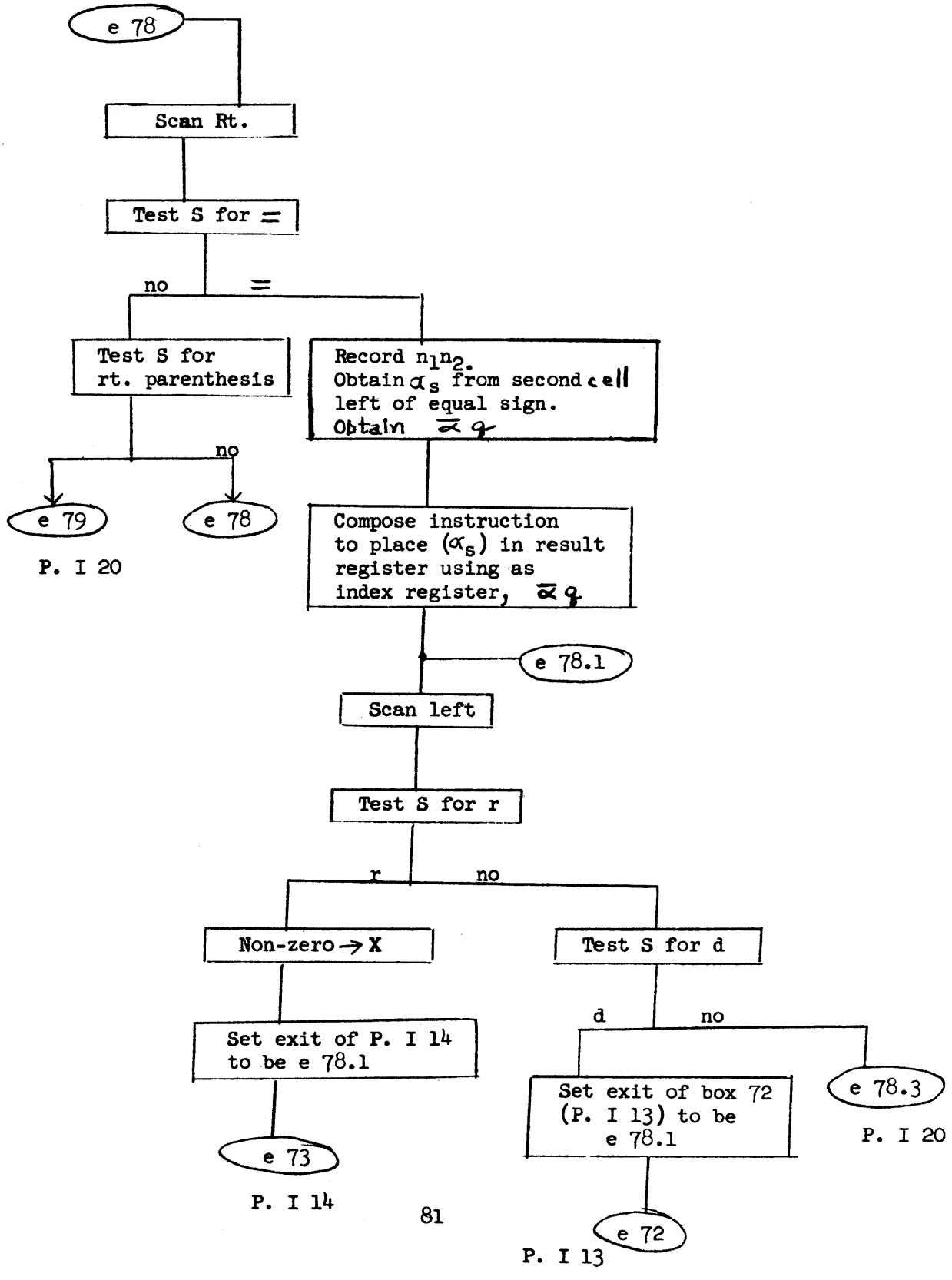

e 75
0 → M
Scan left
Test S for comma
no
Test S for equal sign
1+(M) → M
no
Obtain α_s from second cell left of =
α_s+(M) → M
Compose instruction to "store in (M)"
B_i+1 → B_i
Test B_o for mark indicating first equation
no
B_i+1 → B_i
This puts # ($=n_1n_2$) in B_o
B_o → Sc
$=n_1n_2$ → Sub
e 1.1
P. b1
e 4.2
P.b1

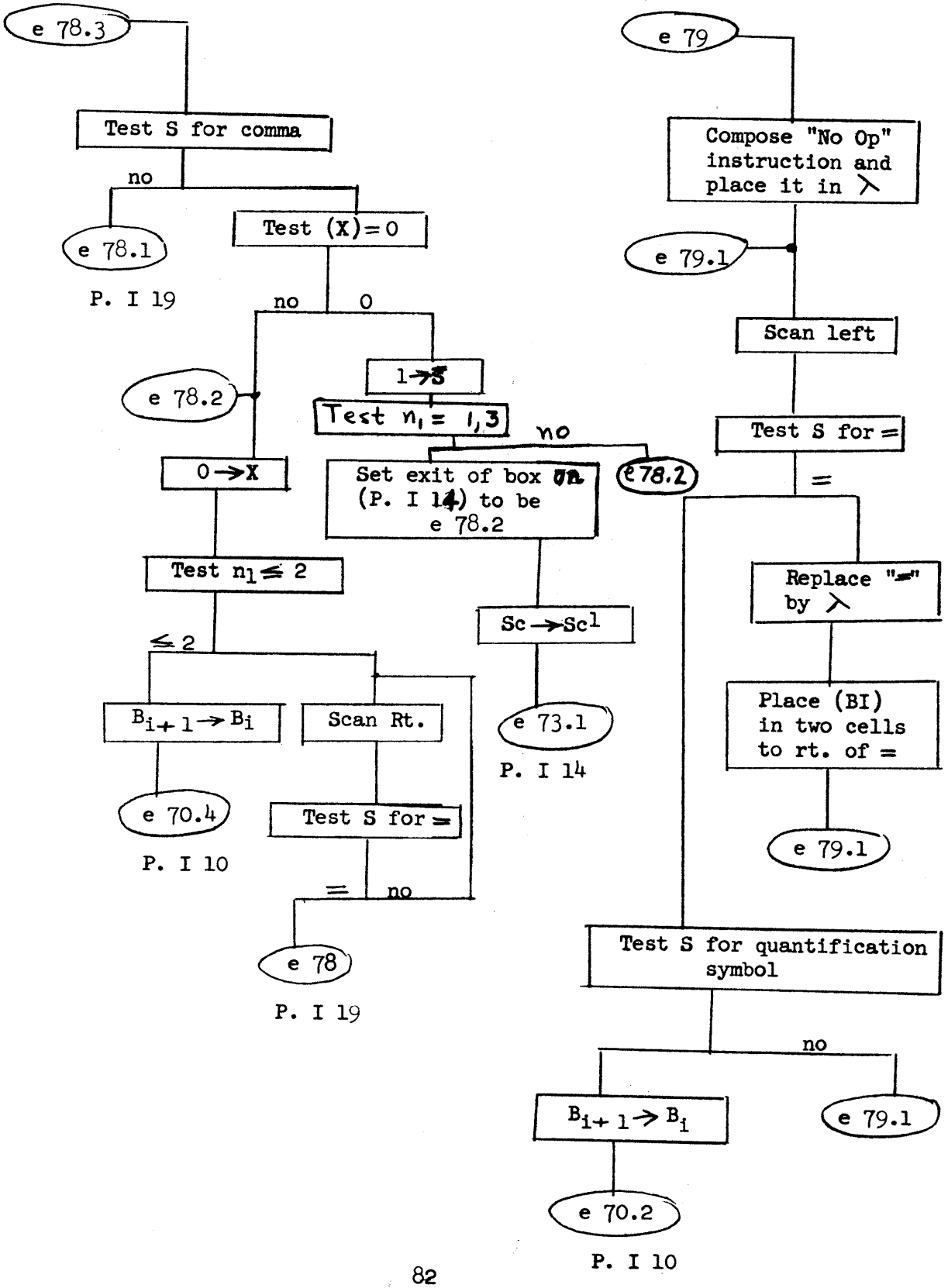

e 76
Test subscript n_1n_2=18
no
18
Record n_1n_2
Branch Subschema
(see section 3)
Obtain α_s from second cell to left of equal sign. Record.
Test n_2=1
$n_2 > 1$
Double Recursion Subschema
(see D.R.1)
Test n_1=5
no
Compose instruction to "store in α_s"
Scan left
Test S for quantification symbol
e 77
P. I 17
no
Scan Rt.
Record α_q
e 76.1
P. I 17


e 76.1
Scan TAI for α_q
no
Error 30
Scan Rt.
Test (S) = P_{o2}
no
e 76.11
Scan Rt.
Record $\bar{\alpha}_q$
Use $\bar{\alpha}$ q as index register for instruction "store in α_s"
e 77
P. I 17
e 77
(equal sign) → Sc
Scan left
Test S for b
b
Scan Rt.
Test S for d
no
d
e 77.1
P. I 18
Set exit of box 72 p. I 13 to be e 77.1
e 72
P. I 13


e 77.1
$B_{i+1} \rightarrow B_i$
Test B_o for "End Recursion" Mark
no
$B_o \rightarrow Sc$
(=) →Sc
$n_1 n_2 \rightarrow$ Sub *
Scan left
e 1.1
P. b 1
Test for quantification symbol
no
Scan Rt.
Obtain αq, to be used as address part of instruction to close loop controlled by index, q.
Scan Rt.
Test S for minimization b
b
no
b→E
#b → B_o with Special Minimization Tag
e 1
P. b1
$B_i \rightarrow B_{i+1}$
Compose instruction to close q-loop
e 78

I 19


e 78
Scan Rt.
Test S for =
no
=
Test S for rt. parenthesis
Record n_1n_2. Obtain α_s from second cell left of equal sign. Obtain α_q
no
e 79
e 78
P. I 20
Compose instruction to place (α_s) in result register using as index register, α_q
e 78.1
Scan left
Test S for r
r
no
Non-zero → X
Test S for d
Set exit of P. I 14 to be e 78.1
d
no
Set exit of box 72 (P. I 13) to be e 78.1
e 78.3
P. I 20
e 73
P. I 14
e 72
P. I 13


e 78.3
Test S for comma
no
e 78.1
P. I 19
Test (X)=0
no
0
e 78.2
1→S
Test n_1 = 1,3
no
0→X
Set exit of box 0a (P. I 14) to be e 78.2
Test $n_1 \leq 2$
≤ 2
$B_{i+1} \rightarrow B_i$
Scan Rt.
e 70.4
P. I 10
Test S for =
= no
e 78
P. I 19
Sc → Sc^1
e 73.1
P. I 14
e 79
Compose "No Op" instruction and place it in ⋋
e 79.1
Scan left
Test S for =
=
e 78.2
Replace "=" by ⋋
Place (BI) in two cells to rt. of =
e 79.1
Test S for quantification symbol
no
$B_{i+1} \rightarrow B_i$
e 79.1
e 70.2
P. I 10

Explanatory Notes; Interpreter

1. The following standard cells are used:

Cell	Contents
G	Cell number of function being programmed.
L	Number of operands between function being programmed and next function to the left.
D	Degree of function.
Li(f) ...	Directions for library subroutine for f.
V	Address in which result is to be stored.
Su Aux ..	Subscript of equal sign of f-equation.
W_i	Formula for auxiliary function is placed in these cells for translation. This preserves the original formula.
O_k	Address of operand to replace c_k in formula for auxiliary function, $(k = 0,1,\ldots,99)$.
M_k	Address of index associated with operand in O_k, $(k = 0,1,\ldots,99)$.

List of Error Indications Given by Encoder

Error Number	Meaning
1	Improper symbol on left side of a b-equation.
2	A non-recursive implicit definition of a b-symbol.
3	Improper index used with a b-symbol.
4	Improper column index used with a b-symbol.
5	Improper operand in a b-equation.
6	Improper indexing of b-symbol defined by vector recursion.
7	Improper index used with lower bound of quantifier.
8	Order of quantifications incorrect.
9	Improper upper bound in minimization.
10	Improper bound in a quantification.
11	Improper index for upper bound in minimization.
12	Independent index quantified more than once.
13	Improper quantification in a recursion.
14	Improper indexing of some variable.
15	Order of quantifications is incorrect (i.e. an index, q, is called for before it has been quantified).
16	Improper quantification in a recursion.
17	An r used as a quantifier bound involves some q which has not yet been quantified.
18	An independent variable in some r-equation has as index a q-symbol which has not yet been quantified.
19	Improper operand in r-equation defining a storage r-symbol.
20	Improper operand in some r-equation.
21	Incorrect format of storage r-equation.

Error Number	Meaning
22	Right side of storage r-equation does not begin with an f-symbol.
23	Independent index in storage r-equation is not properly quantified.
24	Recursion index for course-of-values recursion is not listed in Computer Table.
25	No storage r in course-of-values double recursion.
26	Right side of equation is not well-formed formula. (i.e. incorrect number of operands for some function.)
27	Auxiliary vector function not indexed.
28	Index of auxiliary vector function improperly quantified.
29	Improper operand used with auxiliary vector function.
30	Error in operation of Encoder.

Minimization Subschema

We recall that the minimization function can occur in two contexts: (1) to define an index of a variable in a "table look-up" type of operation; (2) to define an index as the bound in a quantification.

Let us consider the first instance. Suppose that the indexed variable, $a_1 r_1$, appears in the formulation and that r_1 is defined by the equation,

$$r_1 = f_{min}\ q_2\ 25\ b_3, \ .$$

This means that r_1 is the minimum value of q_2 less than or equal to 25 and such that $b_3 \leq 0$, where b_3 is defined by a b-equation.

The indexed variable, $a_1 r_1$, is not processed in the normal operation of the Addressor. When r_1 is discovered by the b-schema (see p. b9, e 2.1), the initial address for the variable a_1 is obtained and placed in Q_o and a non-zero quantity is placed in T. The Addressor then attempts to operate on $a_1 r_1$ and is led into the r-Schema at e30. The r-Schema proceeds as usual except that now when it encounters the Minimization equation, it transfers control to the Minimization Subschema.

The Minimization Subschema obtains the independent index, q_2, and its upper bound, 25, from the minimization equation. It sets up an index register for q_2 with zero as the lower bound. It then stores a non-zero quantity in the register, Min. It records $\#b_3$ in Sc, places b_3 in S, and transfers control to e3 of the b-Schema. Since (Min) $\neq$ 0, the b-Schema records $\#b_3$ in B_o together with a special minimization tag. After resetting all A_i and R_i registers and putting zero in T, control passes to the beginning of the Addressor as after a quantification.

The Addressor processes all a's which have q_2 as an index. However, the Addressor does not transfer back to e5 as after a quantification. Instead, it tests B_o for a minimization tag, finds one, and transfers control to e1 of the b-Schema. This causes b_3 to be programmed.

After b_3 is programmed and assigned a storage address, αb_3, the Interpreter tests B_o for a minimization tag. The presence of the tag causes control to go to "Minimization End-Procedure", which is a subschema which completes the program for r_1 by composing instructions to test $b_3 \leq 0$ and to close the loop corresponding to q_2. The final value of q_2 which will be computed in this loop is the value for r_1. It will be stored in αq_2. The right side of the r_1 equation is erased and αq_2 is recorded in its place. The Interpreter then transfers (B_{i+1}) to B_i, puts the cell number of the equal sign of the equation containing $a_1 r_1$ into Sc and control passes to e2 of the b-Schema.

If r_1 occurs as the bound of an independent index, q_1 say, in a quantification, this is detected in the q-Schema (p. q3). The q-Schema replaces r_1 by b_3 in the quantification. Later, the Interpreter encounters b_3 (p.I 18)

and causes it to be programmed. Then the End-Procedure composes the instructions for closing the q_1-loop, using αb_3 and the upper bound. Finally, putting (B_{i+1}) into B_i, control passes to e78 (p. I 19).

The final value of q_1 is the value for r_1. Since it is stored in αq_1, the right side of the r_1 equation is erased and αq_1 is recorded in its place. Thus, to call for the final value of $b_2 q_1$ say, the formulator would write $b_2 r_1$. The Addressor would replace b_2 by its initial address, αb_2, and r_1 by αq_1 to be used as an index register by the Computer.

Vector Schema

On page b1, if the subscript of an equal sign is found to be 09, control is sent to the Vector Schema. In lieu of flow charts, the following verbal description of the Vector Schema is given.

It will be recalled from NAVORD 4209 (p. 17), that a vector equation is a set of consecutive b-equations of the form,

$$
\begin{aligned}
(\; y &= {}_{09}\, \phi_o \; , \\
&= {}_{09}\, \phi_1 \; , \\
&\;\; \vdots \qquad \vdots \\
&= {}_{09}\, \phi_n \; , \;)
\end{aligned}
$$

where y denotes the dependent variable which represents the vector and ϕ_i denotes the b-term which defines the component y(i), i= 0,1,...,n. The first reference to y, causes all components to be programmed by the Vector Schema (V.S.).

V.S. scans to to the left parenthesis. It then scans right for equal signs and records a storage address, α_s, (obtained from the β-counter) in the second cell to the left of each equal sign. This insures that the components will be stored in consecutive storages. When the right parenthesis is reached, V.S. scans left for equal signs. This time, as each equal sign is encountered, the operation "$B_i \rightarrow B_{i+1}$" takes place and the cell number of the equal sign is recorded in B_o. The cell number of the last equation is recorded with an "End Vector" mark.

Control is returned to el.1 of the b-Schema, which then causes the components y(0),..., y(n) to be programmed in that order. The Interpreter (p. I 16) will detect the subscript 09 and send control to an end-procedure which causes $B_{i+1} \rightarrow B_i$ after each component is programmed and stored.

Since the formula for a component can refer to prior components, this is tested for on page b10. The index indicates which component is required and the address is obtained from the corresponding equation.

The above procedure should not be confused with the treatment of vector auxiliary equations (see NAVORD 4209, p. 17). These equations are processed by the Interpreter (see e64, p. I3) which composes the necessary transfer instructions which will direct the computer to the programs for each component of the vector. On p. I9, after each component is programmed, control goes to the Auxiliary Function Subschema. This subschema tests for a vector auxiliary function by examining the subscript of the equal sign as recorded in $\overline{W}_o$ (page I5). If the subscript is 09, this indicates a vector auxiliary

function is being programmed. By testing register N (p. I3), the subschema determines which component of the vector is being programmed. An appropriate "transfer" instruction is composed to permit the Computer to jump to the program for this component when called for. (N) is then reduced by 1 and the equation for the next component is located. Control is returned to e67 (I6) and the next component is programmed.

Double Recursion Subschema

On page I 16, there is an exit to the Double Recursion Subschema. This subschema is not charted and will be explained briefly in the following paragraphs.

The subschema performs what might be called the end-procedure of the programming of an equation in a double recursion. As in the end-procedure for a vector recursion, this involves the composition of storing instructions, output instructions and instructions for closing loops corresponding to the row index and column index.

The information which controls this end-procedure is obtained from the equation itself and from the B_i registers. On pages rec. 5 through rec. 9, certain control information is placed in the B_i registers in preparation for the end-procedure. Thus, for example, the Interpreter can tell from the contents of B_0 whether this particular equation is within the square brackets (see NAVORD 4209 for double recursion format). By moving (B_{i+1}) to B_i, it controls the sequence of programming. By testing for the 'End Recursion' mark in B_0, it can tell when to close the outside loop and complete the recursion program.

Triple Recursion Subschema

The format of a triple recursion is given in NAVORD 4209. The subschema for translating a triple recursion will not be charted since the main ideas are the same as those in vector recursion and double recursion. Thus, on page b14, when control is sent to this subschema, there is a scanning for the brackets and parentheses which set off the equations controlled by the three quantifiers. The B_i registers must be loaded with the necessary control information and storage addresses must be allotted for the results.

APPENDIX

One of the demands frequently made of an automatic programming system is that it be possible to write library programs in the language of the system. In a system like ADES, in which the formulator need know almost nothing about the machine operations, this would mean that anyone familiar with the ADES language could write library programs quite independently of the Computer.

A rather novel technique has been invented to make this possible in the ADES system. It introduces an entirely new idea in the handling of library subroutines, and we shall outline it briefly in this appendix.

It will be recalled that those functions in an ADES formulation which are not defined by f-equations are assumed to be in the "library"; i.e. it is assumed that the Encoder "understands" which machine instructions should be composed to carry out the function evaluation.

For the simple library functions like "add", "multiply", etc., the Encoder merely uses the basic machine instructions. For a function like "cosine" or "logarithm", it is assumed that a subroutine has previously been prepared and stored on a library tape. The Encoder searches the library tape, identifies the subroutine and compiles it into the program as a closed subroutine. We shall continue to assume the existence of a tape of library subroutines for the elementary functions (trigonometric, hyperbolic, square root, etc.). These subroutines are programs in machine language. Therefore, they should be coded by a programmer who is an expert with the particular computer being used. This is as it should be since these subroutines are used most frequently and should be extremely efficient.

Now, for other mathematical functions which are more complicated and less frequently used (e.g. numerical procedures for solving differential equations), we propose another method. These functions should be regarded not as library subroutines in machine language, but as "library formulations" in ADES language. Thus, the Encoder should have access to a second library tape on which are stored those formulations which are expected to recur occasionally. Such a system could operate as follows:

First, the ADES alphabet must be extended to include the "library independent variables" $x_1, x_2, \ldots$, the "library dependent variables" $y_1, y_2, \ldots$, the "library independent indexes" $i_1, i_2, \ldots$, the "library dependent indexes" $j_1, j_2, \ldots$, and the "arbitrary function symbols" $g_1, g_2, \ldots$.

A library formulation must be written in ADES language as explained in NAVORD Report 4209 except that the variables are x's and y's, the indexes are i's and j's and the arbitrary functions are g's. A library formulation is identified simply by a subscripted f, that is, it is just another function.

Assume that a library formulation has been stored on the formulation tape together with its identifying function symbol; e.g. f_{86}. To use this library

formulation in a problem, the formulator would write the identifying function, f_{86}, followed by the parameters on which the function depends. These parameters can be independent or dependent variables, indexes, and other functions; i.e. a's, b's, q's, r's and f's. The parameters must be written in some conventional order. For example, we might agree to write first those a's and b's to be used as input, then the f's which are to be substituted for the arbitrary functions, and finally the b's which are to be assigned to the computed results. Thus, if f_{86} is a formulation which operates on two independent variables and one dependent variable and if there is one arbitrary function involved, the formulator might write

$$b_5 = f_{86}\ a_3\ a_7\ b_2\ f_{51},$$

as one of the equations in his formulation. The Encoder would operate on f_{86} as follows:

Before any part of the formulation is translated, a new schema called the Assembly Schema would scan the equations for library functions. On finding f_{86}, the tape containing the library formulations would be scanned for "f_{86}". The formulation which follows "f_{86}" on the tape should contain three independent variables denoted by x_1, x_2, and x_3, and one arbitrary function denoted by g_1. The final result in the library formulation should be denoted by y_1.

The Assembly Schema would copy the library formulation from the tape and append it to the main formulation, substituting a_3, a_7 and b_2 for the arguments x_1, x_2, and x_3 respectively and substituting f_{51} for g_1. The dependent variable, y_1, would be replaced by b_5. Any other y's would be replaced by b's with subscripts which did not conflict with others in the main formulation. This permits the formulator to call for the result as b_5 in other parts of his formulation.

If a library formulation computes more than one final result, the following equation might be written:

$$b_5 = f_{93}\ a_1\ a_4\ a_9\ b_3\ f_{61}\ f_{58}\ b_6\ b_7, \ .$$

This would call for library formulation f_{93}, in which there are four input variables, x_1, x_2, x_3, and x_4 to be replaced by a_1, a_4, a_9 and b_3 respectively; there are two arbitrary functions, g_1 and g_2, to be replaced by f_{61} and f_{58} respectively; and there are three final results, y_1, y_2, and y_3 to be replaced by b_5, b_6, and b_7 respectively. (Of course, the significance of y_1, y_2 and y_3 must be indicated in a catalog of library formulations so that any formulator will know how to distinguish them.)

It should be understood that an arbitrary function in a library formulation is always to be replaced by an auxiliary function, that is, by an f which is defined by an f-equation in the main formulation. All other functions in a library formulation are denoted by f's. In particular, one library formulation may contain an f which refers to another library formulation. In that event, the Assembly Schema would have to scan the copy of the library formulation. It would then operate as before to copy the second library

formulation. This has the effect that each library formulation becomes an open subroutine when finally translated by the Encoder into a computer program.

The use of arbitrary functions in a library formulation allows the formulation to be completely general. In fact, this is the usual way of describing a general mathematical procedure. It will be found that library formulations for most of the standard numerical methods can be written merely by copying the formulas from some textbook (subject always to the rules and syntax of ADES of course). The syntax of the f-equations in the ADES language should be broadened somewhat to allow for branching and referrals to other f-equations.

Output specifications for printing, punching, etc. of results computed by library formulations would have to be given in separate equations. This leads to the suggestion that the ADES language be modified somewhat so that a general output table can be given for all results, rather than have the output specified for each equation.

Besides making it possible for "anybody" to write library subroutines, the ideas outlined above have other advantages. (1) The problem of cross-referencing subroutines is eliminated. (2) Since the information in formulation form is exceedingly compact, the amount of tape needed for library formulations will be much less than would be required by the machine program. (3) The assembly process is quite simple.

Aeroballistic Research Department
External Distribution List for Applied Mathematics (X3)

No. of Copies	
	Chief, Bureau of Ordnance Department of the Navy Washington 25, D. C.
1	Attn: Ad3
1	Attn: Ad6
1	Attn: Ree
1	Attn: Re9a
	Chief, BuAer Washington 25, D. C.
3	Attn: TD-414
	Commander, U. S. NOTS Inyokern, China Lake, Calif.
1	Attn: Technical Library
1	Attn: Code 503
	Commander, NAMTC Point Mugu, California
2	Attn: Technical Library
	Superintendent U. S. Naval Postgraduate School Monterey, California
1	Attn: Tech Rpts Section
	Director, NRL Washington 25, D. C.
1	Attn: Code 2021
	Officer in Charge, NPG Dahlgren, Virginia
2	Attn: Technical Library
	Office, Chief of Ordnance Washington 25, D. C.
1	Attn: ORDTU
	Chief, AFSWP Washington 25, D. C.
1	Attn: Document Library Br.
	Commander, WADC Wright-Patterson AF Base Ohio
3	Attn: WCORC

No. of Copies	
	Commanding General Aberdeen Proving Ground, Md.
2	Attn: Tech Info Br.
2	Attn: Director, BRL
	Commanding General Redstone Arsenal Huntsville, Alabama
1	Attn: Aero. Lab, GMDD
5	ASTIA Document Service Center Knott Building Dayton 2, Ohio
	NACA High Speed Flight Station Box 273 Edwards AF Base, Calif.
1	Attn: Mr. W. C. Williams
	NACA Ames Aeronautical Laboratory Moffett Field, California
1	Attn: Librarian
	NACA Langley Aeronautical Laboratory Langley Field, Virginia
1	Attn: Librarian
1	Attn: Adolf Busemann
2	Attn: John Stack
	NACA Lewis Flight Propulsion Lab. 21000 Brookpark Rd. Cleveland 11, Ohio
1	Attn: Librarian
1	NACA 1512 H Street, N. W. Washington 25, D. C.
	Commanding Officer, DOFL Washington 25, D. C.
1	Attn: Lib., Rm 211, Bldg. 92

No. of Copies

Office of Naval Research
Room 2709, T-3 Building
Washington 25, D. C.
1 Attn: Head, Mechanics Br.

Director of Intelligence
Headquarters, USAF
Washington 25, D. C.
1 Attn: AOIN-3B

Director, DTMB
Aerodynamics Laboratory
Washington 7, D. C.
1 Attn: Library

University of California
Berkeley 4, California
2 Attn: Dr. S. A. Schaaf
Via: ONR

The University of Texas
P. O. Box 8029
Austin 12, Texas
1 Attn: Defense Research Lab.
Via: ONR

1 Applied Math and Statistics Lab.
Stanford University
Stanford, California
Via: InsMat

University of Michigan
Willow Run Research Center
Willow Run Airport
Ypsilanti, Michigan
1 Attn: Librarian
Via: ONR

APL/JHU
8621 Georgia Ave.
Silver Spring, Maryland
2 Attn: Technical Rpts Group
Via: InsOrd

The Ohio State University
Research Foundation
Nineteenth Avenue
Columbus 10, Ohio
1 Attn: Security Officer
Via: ONR

No. of Copies

CIT
Pasadena 4, California
1 Attn: Aeronautics Dept.
1 Attn: Jet Propulsion Lab.
Via: ONR

University of Minnesota
Minneapolis 14, Minn.
1 Attn: Mechanical Eng. Dept.
Via: ONR

RAND Corp.
1700 Main St.
Santa Monica, California
1 Attn: Lib., USAF Project RAND
Via: InsMat

Douglas Aircraft Co., Inc.
Santa Monica Division
3000 Ocean Park Blvd.
Santa Monica, California
1 Attn: Chief Engineer
Via: InsMat

1 CONVAIR Corp.
A Div. of Gen. Dynamics Corp.
Daingerfield, Texas
Via: InsMat

United Aircraft Corporation
400 Main Street
East Hartford 8, Connecticut
1 Attn: Chief Librarian
Via: InsMat

Guggenheim Aeronautical Lab.
California Inst. of Technology
Pasadena 4, California
1 Attn: Aeronautics Library
Via: ONR

Cornell Aeronautical Lab., Inc.
P. O. Box 235, 4455 Genessee St.
Buffalo 21, New York
1 Attn: Librarian
Via: InsMat

No. of
Copies

Lewis Flight Propulsion Lab.
21000 Brookpark Road
Cleveland 11, Ohio
1 Attn: Chief, Supersonic
Propulsion Div.
Via: InsMat

Hughes Aircraft Corp.
Culver City, California
1 Attn: Assistant Director
GMRD Division
Via: InsMat

1 McDonnell Aircraft Corp.
P. O. Box 516
St. Louis 3, Missouri
Via: InsMat

General Electric Company
2900 Campbell Avenue
Schenectady 5, New York
1 Attn: Library
Guided Missiles Dept.
Via: InsMat

Eastman Kodak Company
Navy Ordnance Division
50 West Main Street
Rochester 14, New York
2 Attn: Mr. W. B. Forman
Via: InsOrd

1 The Avco Manufacturing Corporation
Research Labs.,
2385 Revere Beach Parkway
Everett 49, Massachusetts
Via: InsMat

Aeroballistic Research Department
External Distribution List for Applied Mathematics (X3a)

No. of Copies	
1	Chief, Fluid Mechanics Section National Bureau of Standards Washington 25, D. C.
1	National Bureal of Standards Washington 25, D. C. Attn: Applied Math Div.
6	Commanding Officer Office of Naval Research Br. Off. Box 39, Navy 100 Fleet Post Office New York, New York
1	Langley Aeronautical Laboratory Langley Field, Virginia Attn: Theoretical Aerodynam. Division
1	Naval Research Laboratory Washington 25, D. C. Attn: Dr. H. M. Trent Code 6230
1	Case Institute of Technology Cleveland 6, Ohio Attn: G. Kuerti
1	Massachusetts Institute of Tech. Cambridge 39, Mass. Attn: Prof. Joseph Kaye Room 1-212
1	The Johns Hopkins University Charles and 34th Streets Baltimore 18, Maryland Attn: Dr. Francis H. Clauser
2	Director Inst. for Fluid Dynamics and Applied Mathematics University of Maryland College Park, Maryland
1	Cornell University Graduate School of Aero. Eng. Ithaca, New York Attn: Prof. W. R. Sears
1	AERCON, INC. 560 Punahou St. Altadena, California
1	University of Michigan Randall Laboratory Ann Arbor, Michigan Attn: Prof. Otto Laporte
1	Brown University Providence 12, Rhode Island Attn: Prof. William Prager
1	University of Michigan Engineering Research Institute Ann Arbor, Michigan Attn: Mr. H. E. Stubbs Research Associate

No. of Copies

1 Oak Ridge National Laboratory
Oak Ridge, Tennessee
Attn: Dr. A. S. Householder

1 University of Michigan
Engineering Research Institute
Willow Run Airport
Ypsilanti, Michigan
Attn: Mr. C. C. Elgot

1 Los Alamos Scientific Laboratory
Los Alamos, New Mexico
Attn: Dr. S. M. Ulam

1 University of Wisconsin
Madison 6, Wisconsin
Attn: Dr. S. C. Kleene

1 National Bureau of Standards
Washington 25, D. C.
Attn: Dr. H. Antosiewicz

1 National Bureau of Standards
Washington 25, D. C.
Attn: Dr. F. L. Alt

1 Pennsylvania State College
State College, Pa.
Attn: Dr. H. B. Curry

1 Electronic Computer Project
Institute for Advanced Study
Princeton, N. J.
Attn: Dr. H. H. Goldstine

1 Applied Mathematics Laboratory
David Taylor Model Basin
Washington 7, D. C.
Attn: Dr. H. Polachek

1 University of California
Berkeley 4, California
Attn: Dr. D. H. Lehmer

1 Raytheon Company
Waltham 54, Mass.
Attn: Dr. R. F. Clippinger

No. of Copies

1 Division of Computing Services
Institute of Math. Sciences
New York University
New York 3, New York
Attn: Dr. E. Bromberg

1 Westinghouse Electric Corp.
Pittsburgh 30, Pa.
Attn: Dr. C. W. Adams

1 Digital Computer Laboratory
Massachusetts Institute of Technology
Cambridge 39, Mass.
Attn: Dr. Jay Forrester

1 Sperry Rand Corporation
Philadelphia, Pa.
Attn: Dr. Grace Hopper

1 University of Pennsylvania
Philadelphia, Pa.
Attn: Dr. S. Gorn

1 Data Processing Department
Computing Devices of Canada
P. O. Box 508
Ottawa 4, Ontario
Attn: Mrs. M. Larmour

1 Applied Mathematics
University of Illinois
Graduate College
168 Engineering Research Laboratory
Urbana, Illinois
Attn: Mr. J. P. Nash

1 University of Michigan
Engineering Research Institute
Willow Run Airport
Ypsilanti, Michigan
Attn: Mr. John W. Carr III

1 University of Michigan
Engineering Research Institute
Willow Run Airport
Ypsilanti, Michigan
Attn: Mr. J. B. Wright

No. of Copies

1 Logistics Branch
Office of Naval Research
Washington, D. C.
Attn: Dr. M. E. Rose

1 Melpar, Inc.
3000 Arlington Boulevard
Falls Church, Va.
Mr. Aaron Halperin

1 United Merchants & Manufacturers
1407 Broadway
New York 18, New York
Attn: Mr. S. Greshin

1 Computer Services
Lockheed Aircraft Corp.
Missile Systems Division
Van Nuys, California
Mr. W. W. Leutert

1 Remington-Rand
Eckert-Mauchly Division
1900 West Allegheny Ave.,
Philadelphia 29, Pennsylvania
Attn: Mr. Charles Kats
Box 5616, Room 701

1 Naval Research Laboratory
Code 6232
Washington 25, D. C.
Attn: Dr. Ben Lepson

1 I.B.M.
590 Madison Avenue
New York 22, New York
Attn: Mr. Harlan Herrick

1 Bell Telephone Laboratories
Murray Hill, New Jersey
Attn: Dr. R. W. Hamming

1 Mathematical Analysis Department
Lockheed Aircraft Corp.
Burbank, California
Attn: Mr. Harvey Bratman

No. of Copies

1 Vallecitos Atomic Laboratory
General Electric Co.
1763 South First St.,
San Jose, California
Attn: Mr. R. H. Stark

1 Northrop Aircraft
2417 N. 165th St.,
Gardena, California
Attn: Mr. Seymour Ginsburg

1 Computing Operation
Bldg. 305 FPLD
General Electric AGT
Cincinnati 15, Ohio
Attn: Mr. Donald J. Hahn

1 Plant II
Physical Research Staff
Boeing Airplane Co.,
Seattle, Washington
Attn: Miss Mandalay Grems

1 Department of Mathematics
Darmouth College
Hanover, New Hampshire
Attn: Prof. John McCarthy

1 The Rand Corporation
1452 Fourth St.,
Santa Monica, California
Attn: Mr. Harry A. Pappo

1 Computing Center
Ramo-Wooldridge Corp.
5730 Arbor Vitae St.,
Los Angeles 45, Calif.
Attn: Dr. Walter F. Bauer

1 Mathematics Division
Stanford Research Institute
Menlo Park, California
Attn: Dr. George Evans

No. of Copies

1 Methods Division
The Prudential Insurance Co. of America
Newark 1, New Jersey
Attn: Mr. J. E. Coachman

1 Director
National Security Agency
3801 Nebraska Ave., N. W.
Washington 16, D. C.
Attn: LIB Acquisitions

1 Electronic Data Processing Dev.
General Electric Company
One River Road
Schenectady 5, New York
Attn: Mr. J. W. Pontius

1 Air Force Cambridge Research Center
L. G. Hanscom Field
Bedford, Massachusetts
Attn: Mr. Bert F. Krauss, CRRI

1 Computer Control Co.,
10966 LeConte St.,
W. Los Angeles, California
Attn: Mr. Frank Stockmal

1 The Equitable Life Assurance Society
393 Seventh Avenue
New York 1, New York
Attn: Mr. Walter L. DeVries

1 Metropolitan Life Insurance Co.
One Madison Avenue
New York 10, New York
Attn: Mr. Robert D. Acker

1 Digital Computer Laboratory
Convair
San Diego 12, California
Attn: Mr. Ben Ferber

No. of Copies

1 Computer Techniques Development
Investigations Section
AGT Development Department
Building 300
General Electric Co.,
Cincinnati, 15, Ohio
Attn: Mr. D. L. Shell

1 New York University
College of Engineering
University Heights
New York 53, NewYork
Attn: Mr. Emanuel Mehr

1 Burroughs Corp. Research Center
Paoli, Pennsylvania
Attn: Mr. Jos. Deutsch

1 Programming and Operation Research
Hughes Aircraft Co.,
Culver City, California
Attn: Mr. Leon Gainen

1 Systems Analysis Dept.,
Remington Rand
Engineering Research Associates Division
1902 West Minnehaha Avenue
St. Paul W4, Minnesota
Attn: Mr. B. F. Cheydleur

1 American-Standard Atomic Energy Division
1682 Broadway
Redwood City, California
Attn: Mr. Edward J. Leshan

1 General Kinetics Inc.
555 - 23rd St., South
Arlington 2, Virginia
Attn: Mr. A. E. Roberts

1 Carnegie Institute of Technology
Schenley Park
Pittsburgh 13, Pennsylvania
Attn: Prof. A. J. Perlis

No. of Copies

1 Computational Services Section
Babcock & Wilcox
1201 Kemper St.,
Lynchburg, Virginia
-Attn: Mr. R. F. Reiss

1 Computing Center
Ramo-Wooldridge Corp.
5730 Arbor Vitae St.,
Los Angeles 45, Calif.
Attn: Mr. Robert Perkins

1 I.B.M.
590 Madison Avenue
New York 22, New York
Attn: Mr. J. C. McPherson

1 I.B.M.
1111 Connecticut Ave. N. W.
Washington 6, D. C.
Attn: Mr. T. Horton

1 Servomechanisms Inc.
23821 Madison St.,
Torrance, California
Attn: Mr. Neil Block

1 Computing Center
Ramo-Wooldridge Corp.
5730 Arbor Vitae St.,
Los Angeles 45, California
Attn: Dr. E. K. Blum

1 Computing Center
Ramo-Wooldridge Corp.
5730 Arbor Vitae St.,
Los Angeles 45, California
Attn: Dr. David Young

1 National Bureau of Standards
Room 108A
West Building
Washington 25, D. C.
Attn: Mr. Max Klein

No. of Copies

1 Michelson Laboratory
Code 507
China Lake, California
Attn: Mr. H. Hauer

1 Mechanical Engineering Dept.
Columbia University
Broadway & 16th St.,
New York, New York
Attn: Prof. J. H. Weiner

1 Mathematics Dept.
Pennsylvania State College
State College, Pennsylvania
Attn: Prof. George Raney