COMPUTER ASSISTED INSTRUCTION

Peter G. Perry, B.Sc (Hons)
Department of Computing Science,
University of Adelaide.

September, 1973

INDEX

INDEX (cont)

INDEX (cont)

FIGURES AND TABLES

SUMMARY.

This thesis considers only two aspects of the whole area of Computer Assisted Instruction.

The first area is the design and implimentation of a practical C.A.I. System. This system was designed to support up to five hundred consoles, and an experimental implimentation was carried out on the Control Data 6400 Computer at the University of Adelaide. The experimental system, which involved some sixteen thousand cards of source program, as well as several thousand more cards in off-line supporting programs, was developed between 1967 and 1969, with minor modifications and extensions since then. The description of this system covers both its external features and the major aspects of the internal organisation; although some of this work has been described in the papers Perry and Lee ([29] , 1969) Perry ([28] , 1969) and Lee ([16] , 1971), much of the internal description appears for the first time in this work.

The second area is the problem of answer comparison which arises when the student is allowed freedom to construct his own responses. These problems do not arise in the simple multiple choice response situation, but it is felt that it is in many situations desirable to ask the student to construct a response, rather than to merely select one of a number of alternatives. This freedom and the resultant requirements for comparison, raise basic philosophical questions in the nature and processing of meaning. It would be all too easy to consider only the philosophy,or to ignore the philosophy and concentrate solely on

the practical aspects of the problem, and it is hoped that a balance between these two extremes has been achieved.

Starting from a discussion of the nature of meaning, a general structure for a language model is proposed, and some theory relating to it is developed. A particular form for this model yields a sub-class which are termed language interpreters, and it is shown that these can lead to a practical method of comparing meaning. It is also shown that language interpreters can be related to the work of Osgood et.al. ([26], 1957) who described in a practical situation some of the language behaviour predicted by the model.

This thesis contains no material that has been accepted for the award of any other degree or diploma in any University.
to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Peter G. Perry.

ACKNOWLEDGEMENTS

I wish to acknowledge the advice and assistance of my supervisors, Professor J.Ovenstone, Dr J.Sanderson, Professor F.Hirst, and Dr C.Barter, and also of Mr J.Weadon, who helped me during some of the transition periods.

It was a privilege to work with Lee Kim Cheng on the design and implementation of the C.A.I. system described in the first section of this thesis.

I am indebted to the staff of S.A. Office Machines Pty Ltd for their assistance with typing and duplicating.

I gratefully acknowledge the support of Control Data Australia Pty Ltd, and the University of Adelaide, for providing the grants which made this work possible.

ERRATA

1. In section 4, both upper and lower case S is used to represent a string.
2. Due to an error in page numbering, there is no page 62.

THE ADELAIDE UNIVERSITY C.A.I. SYSTEM.

1.0 INTRODUCTION

Some information about the Adelaide University computer-assisted instruction (C.A.I.) system has already been published (Perry & Lee, [29], 1969; Perry, [28], 1969; Lee [16], 1971) but these descriptions concentrate mainly on the external features and the author language, and contain little detail of the structure and operation of the system itself. This system represents the first attempt at large scale C.A.I. in Australia, and it is therefore worthwhile to record some of the internal features of it; I shall thus give a brief summary of the overall system, outline some of the reasoning which led to its final form, and describe some of its major mechanisms.

The system design was commenced in 1967; design and initial implimentation were substantially completed by the end of 1968, with minor improvements being added during 1969. As will be pointed out during the description of it, this implimentation does not include all features taken into account in the design; some features are entirely absent, while others are present in a restricted form only.

The design and implimentation of the system has been a joint effort with K.C. Lee. Although we both contributed to all phases of the work, Lee concentrated mainly on the development of the author language assembler, operator display processor, and disc driver, as well as several smaller routines in the system, while I was

mainly responsible for the system loaders, monitor, central program and other equipment drivers.

The system was designed for, and implimented on a Control Data Corporation 6400 computer[6] using both the existing hardware and an interface and C.A.I. consoles designed specifically for the project. At the time implimentation was commenced, the computer was a minimum configuration with 32K of 60 bit memory for the central processor, ten autonomous peripheral processors, each with 4K of 12 bit memory, a mass storage disc with a capacity of approximately a half million bits, a twin screen display console, and six magnetic tape drives; it also included a card reader, card punch incremental plotter, and two line-printers, but these are not directly relevant to the system. It was running under the manufacturers operating system SCOPE 2; this was changed to SCOPE 3 about half-way through the programming of the C.A.I. System.

In the description of the C.A.I. system hardware and software features of the computer will be outlined where relevant; further detail can be found in the manufacturers manuals.

1.1 GENERAL DESIGN PHILOSOPHY

The first major decision in the design was to produce a stand-alone, dedicated system. There were several reasons for this.

First, since the design of the console controller was based on a maximum of 512 student consoles, it was felt that there would be no need to run normal jobs in parallel with C.A.I. courses,

and that the memory space, already barely large enough for effective operation of the manufacturer's system, would not permit it.

The SCOPE 2 operating system made very inefficient use of the disc, in particular permitting only sequential files with forward linking, so that even a simple backspace operation required the whole file up to that point to be searched. Disc accesses were not scheduled to minimise unneccessary track changes, so that paging student work-spaces onto disc would have resulted in a major bottleneck.

Finally, SCOPE is a batch processing system, and not readily adaptable for providing facilities for interactive programming.

It was thus felt that the modifications needed to adapt the SCOPE operating system would require more effort than starting from scratch, with very little benefit to be gained.

The introduction of SCOPE 3 did in fact overcome some of these objections; it included an optimising disc request processor and permitted very efficient use of the disc; it also included an interactive programming package the response of which, however, gives adequate justification to the final point made above. The main advantages that were gained from SCOPE 3, from the point or view of the C.A.I. system, were the more powerful program library maintenance facilities and considerably more powerful assembler.

There are many people involved in the use of a C.A.I.system and the requirements of each class of user must be taken into account, difficult though it is to predict all of the requirements in

advance.

Authors require a powerful, flexible course language, giving full control of the capabilities of the student console and the presentation of the course material, the ability to perform calculations, the ability to offer successively more detailed hints to a student experiencing difficulty with a question, and the ability to set time limits if it is so desired. The ability to branch between parallel segments of a course, and to skip forwards or backwards, depending on the responses of the student, is required, and multimedia presentation facilities are highly desirable. For preparing courses, authors need convenient methods for submitting an initial draft, and on-line facilities for checking and editing. Detailed statistics of student progress are needed to assist in course revision.

Students want data output and computer response to input to be fast, so that they do not spend a lot of time waiting for material to appear. They need the ability to correct typing errors in their responses, the ability to control the presentation of material to some extent - in particular, to be able to stop the programme at any point and receive help, either from prepared material in the course or, if necessary, from a supervisor - and the ability to ask for revision of previous material before commencing a new session. The ability to use the computational power of the computer through the console is also desirable.

Supervisors want to be able to retrieve from the system summaries of individual student's progress. During the time that a

student is interacting with the system, the supervisor, who may be physically distant from the students under his care, should be able to carry on a conversation with any student through the consoles, control devices on the student's console, change the position of the student within the course and, after terminating direct contact with the student, receive a log of the student's position and response to questions as he proceeds.

Administrators and operators want automatic collection of accounting statistics, and a system which requires little intervention during operation. The system should be easy to shutdown at the end of a run, and to restart after a failure.

The system was designed in a modular form, so that, although not all of these features are present in the initial implimentation, most of them can be added without major changes to the rest of the system.

1.2 FROM THE AUTHOR'S VIEWPOINT.

The present implimentation does not have on-line author facilities, although an author can log into the system to check his course, and a command is provided to enable him to reposition himself within the course while checking it.

Courses are prepared and punched on cards for processing onto tape under the normal batch processing system. The course assembler is in two passes, the first being an input formatting pass, and the second the actual assembly; thus adapting the assembler for alterna-

tive input media involves only the writing of a new formatting program and it would be possible to write a pre-processor to handle corrections entered on a console and saved on tape. It would be a major task to adapt the course assembler to run as part of the C.A.I. System, although this could be done.

The author language is unique to the system; although adding to the proliferation of course languages is undesir able from several points of view, the most important being the resulting restrictions on the interchange of programmes between systems, it was felt that COURSEWRITER, (see e.g.[10]) the most widely used C.A.I. language, while being simple to learn and use, did not incorporate all of the features that were mentioned in the previous subsection. The system author language may be briefly characterised as an extension of COURSEWRITER, the major area of extension being the structure of the CUE sequence; this does not, however, imply that the resulting language is superficially similar to COURSEWRITER, as may be seen by referring to Fig. 1.9.2, where a short segment is shown.

The author language is extensively descibed by Lee ([16] 1971, see chapter III and appendix A), and therefore will not be further discussed here; subsection 1.9 gives details of the way in which the resulting object code language is processed by the system.

An important feature of the system is that it can in principle handle more than one C.A.I. language; it would be feasible to have several different courses, in quite different languages, all running at the same time, although in the present implimentation only

one language is provided.

The system dumps raw statistics, as requested by the author; on a magnetic tape; these may then be extracted off-line in any form requested by the author; in particular, it is possible to obtain summaries of the progress of individual students, and also data about individual courses, including unanticipated responses to a question, and the time taken to answer a question.

1.3 FROM THE STUDENT'S VIEWPOINT.

The student interacts with the system through a console. The consoles were designed specifically for the C.A.I. System, and consist of a desk with an IBM Selectric I/O writer built into the top; the design included a slide projector and tape player, but these were not available on the consoles actually built.

Approaching any unused console, the student types the command #LOGIN followed by his identification code; the system responds with a reassuring message while it attempts to identify the student by referring to its master file. When the record is found, it is checked for the number of courses for which the student is enrolled; if he is enrolled for more than one, he is asked to make a selection. On selection of a course for which he is enrolled, and which is available in the system, the login phase is completed. The system pauses, with an appropriate message, until the student types a #GO command.

During most of normal course processing, the console keyboard remains locked; this is a hardware function designed to prevent the accidental insertion of spurious characters into the printout, as the printer, unlike a teletype, prints every character entered on the keyboard. Courses are assembled so that the keyboard is unlocked periodically, and should he wish to halt the presentation, he has merely to rest his hand on the "return" key; when the keyboard unlocks, the null line transmitted will cause the programme to pause.

Student input to the system should always be either an answer to a question or a special command. Any message the student types in is first checked to see whether it is a command (all commands commence with the character#) and then to see whether a response was requested; if neither is the case, then the system pauses if the message is a null line, and otherwise responds with an admonitory message. Commands may be abbreviated to any extent, and the variety available is dependent on the console status.

The student can readily correct typing errors on the keyboard; a backspace removes one character, while an ! deletes the whole line, causing an indication to be printed, and a carriage return so that input starts on the next line. Messages are not recognised by the system until the "return" key is pressed.

When the student completes his session, the command # LOGOUT enables him to sign off from the system; however since the student is always restarted at the beginning of a page on each session, he is invited to continue and complete the current page,

thus avoiding repeating portion of it; The sign off at the end of the page is automatic. Should the student wish to leave the console immediately he can force the logout by repeating the command.

Once the student has logged out, he should wait until the master record is updated before continuing with that course, although he may log in for other courses during the same session. Master record updating is done under the standard batch processing system.

1.4 ADMINISTRATION AND OPERATION.

The system is initialised by a "dead start" procedure similar to that of the standard SCOPE system: the system tape is loaded on the appropriate tape transport, and the hardware automatic load initiated.

It is possible at this stage to initiate any of a variety of programs; currently, the choice is between the system loader, which initiates the entire system from scratch, and a diagnostic program for the remote console multiplexor/interface, but a release version of the system would also contain restarting loaders for recovery after a failure.

When system loading is complete, courses must be copied onto the disc. Part of the system is a course loading program, which reads courses from magnetic tape, and writes them onto the disc, creating at the same time two levels of directories - one for chapters, and one for pages; this program is designed to permit the loading of single named courses, or blocks of courses , from the tape file. ,

It is convenient at this stage to assign the three tapes

that the system requires when running; the first of these is the student master tape, containing the codes and course enrollments for all students recognised by the system; the other two are used for the saving of statistical and accounting data. If these are not assigned at this stage, assignment will be requested when they are needed, but prior assignment saves some time.

The final step is to assign the remote console driver, so that students can access the system.

The system as it currently exists does not include provision for an operator initiated shutdown at the end of the run, but this facility could be added; it would probably involve three stages, the first preventing new students from signing on the the system, the second setting the flag for a logout at the end of the current page and issuing a warning message to each console which is signed on, and the final step, initiated immediately before the system is shutdown, would be to log out any active consoles.

As was mentioned earlier, the restarting procedures have not yet been implimented. The addition of such procedures would not involve any major problems; during the loading procedure, the entire system is written on the disc in parallel with the loading, and thus the central and peripheral programs could be restored from there; also, should the memory tables which define the status of every student and console be destroyed, the disc areas which are used for swapping student data blocks (see 1.7) provide a very recent picture of each console's activity.

The two sets of data tapes are processed off-line, under the normal batch-processing system. One tape gives data on course behaviour and details of individual students progress, intended for use by authors and course supervisors. The second contains accounting information which is used to update the student master tape ready for the next run.

1.5 GENERAL STRUCTURE.

The Control Data 6400 Computer System contains two distinct types of computer; the central processor (C.P.) is very fast, has a large (60 bit) word length, and a wide range of arithmetic and logical instructions, but no access to peripheral processors. Peripheral processors (PPs), of which there ten, are much smaller and slower, with limited arithmetic, but convenient access to peripheral devices through the twelve bi-directional data channels, unlimited access to the memory of the central processor, and complete control over the action of the central processor.

The system thus works on a division of labour between the central and peripheral processors; this division is conceptually similar to that used by SCOPE, but differs somewhat in the details.

The central processor schedules students for attention, processes courses and command words, and controls all messages to and from the C.A.I. consoles. Because of its more efficient shifting and masking instructions, it also handles a bit-table showing which disc sectors are available for use.

Except during the hardware load sequence, it is not possible to distinguish the ten peripheral processors, but it is convenient to follow the SCOPE practise of placing one of them in control. Thus one of them, designated PP0, is loaded with a monitor program; this program maintains the real-time clock, and co-ordinates the work of the other peripheral processors, acting as an interface between them and other segments of the system.

The system status is constantly displayed on the twin-screen operator console by a program running in PP1. This program also enables the operator to control the system by entering commands through the console keyboard.

All data transfers to and from disc are queued and scheduled by a disc driver in PP9.

The remaining seven peripheral processors form a pool upon which the monitor can draw whenever necessary. Each has a small resident program which links it to the monitor through a communication area in central memory; on request from the monitor, this resident program will load as an overlay (by communicating with the disc driver along a data channel) any of the system library routines; when the task is complete, the processor returns to the pool of free units. Examples of these transient tasks are course loading, dumping statistical data from a memory buffer onto magnetic tape, and performing data transfers between central memory and the C.A.I. consoles. The mechanism for assigning pool processors and interlocking their actions does not differ significantly from that used by SCOPE, and hence will

not be further discussed.

1.6 THE C.A.I. CONSOLES.

The C.A.I. Consoles and their multiplexor/interface were designed by Dr. R.J. Potter, of the Adelaide University, and constructed under his supervision.

The design provides for consoles consisting of up to seven independent input or output units, but the four consoles actually constructed contain only the typewriter, which is accounted two units. These typewriters are IBM Selectric I/O Writers; they type at over fifteen characters per second. Data is transferred to and from these in "correspondence code", a six bit code; however, case changes are handled automatically by the console logic, an extra bit being added to the code for this purpose.

A group of up to eight consoles share a "local controller" to which they are connected by a short daisy chained line. The local controller transmits and receives data from the transmission line, acting as a relay between it and the consoles; it also polls consoles for keyboard data if a reject or status response is not to be returned, transmitting either a data character or a "no data" signal, and generates reject signals for consoles which do not respond to the data sent to them.

The multiplexor/interface is designed to send and receive signals over up to sixty four transmission lines; these lines, which link it with local controllers, may be of any length, so that the group

of consoles could be scattered in many different locations; to allow for differences in transmission rates, a status bit is set in the line register if a reply has not been received from the local controller by the time the driving peripheral processor calls for it. So far, only two lines have been built.

The general structure of this equipment is shown in Fig. 1.6,1 and the data format in Fig. 1.6,2. It is described in more detail by Potter ([30], 1967).

There are three programs in the system which communicate with this console complex, and a further program which simulates its behaviour on the operators display screen.

The first of these programs is a hardware diagnostic for the interface itself. Because any adjustment or malfunction of the interface can cause the peripheral processor to hang up with no possibility of recovery without a new dead start, this is a stand-alone program - one of the options of the system preloader is to load this instead of the main system loader - and runs in two peripheral processors; one processor drives the interface, and the second maintains a display of its status, so that when the driver is hung, it is still possible to see the last data transmitted. The program can send functions and data to the interface at various speeds, and input data from it; although designed chiefly to test the interface function limited testing of the consoles is also possible.

The second program is used to check the individual consoles and runs as a transient job in two peripheral processors under the

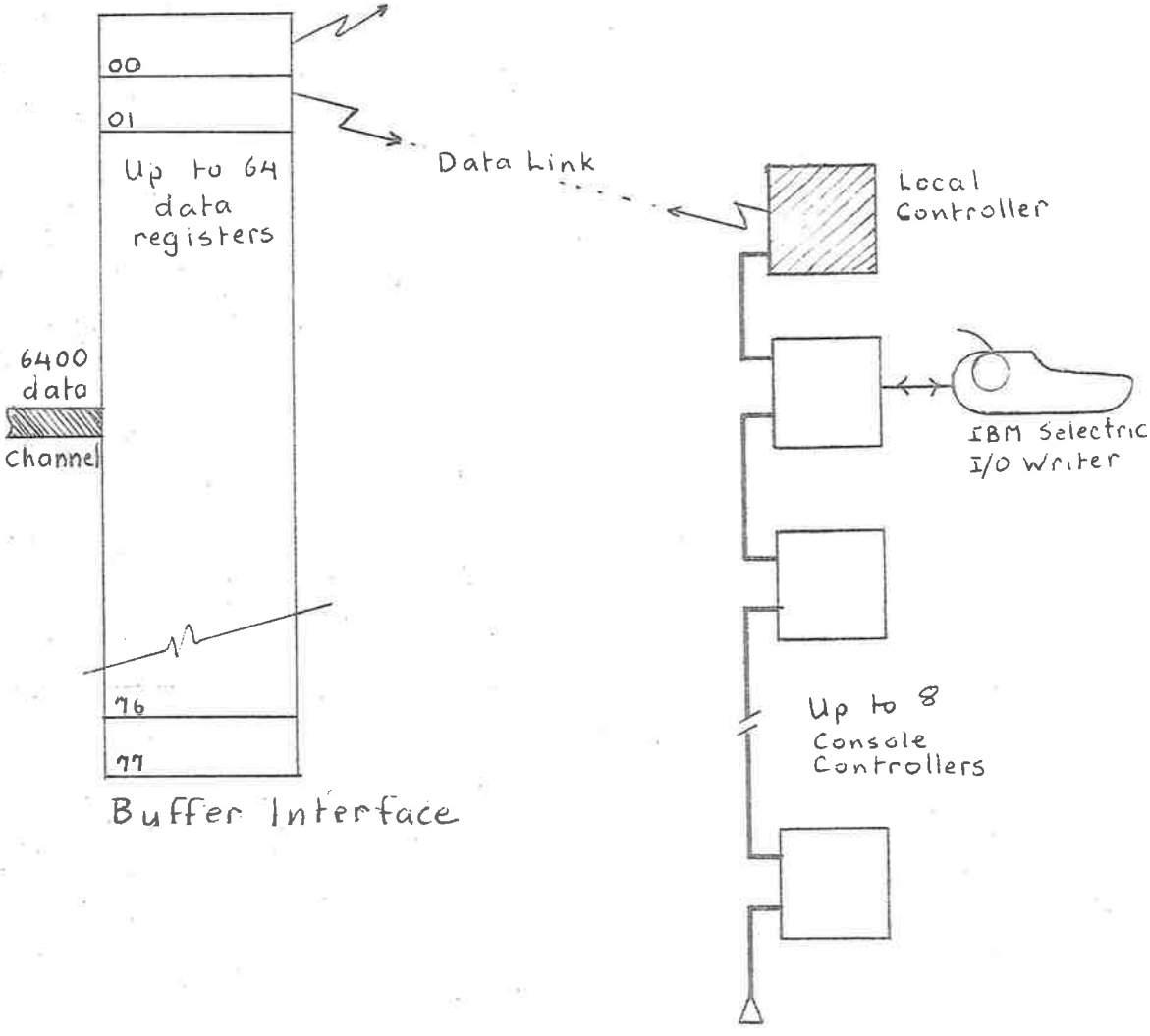
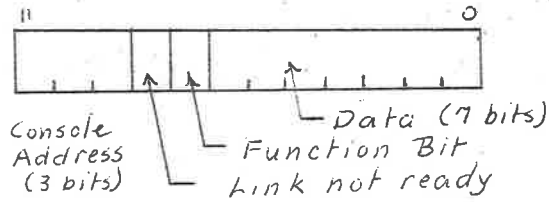
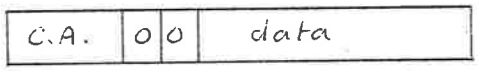


Fig 1.6.1 Remote Console Hardware

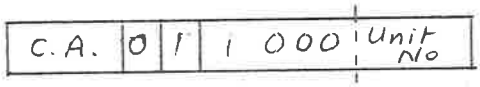


GENERAL DATA WORD FORMAT (12 bits)

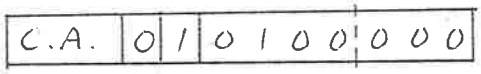
OUTPUT CODES



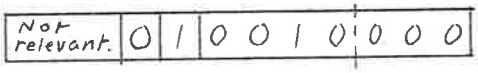
DATA OUTPUT (7 bit char. in correspondence code)



SELECT UNIT (0-7)

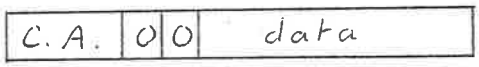


STATUS REQUEST

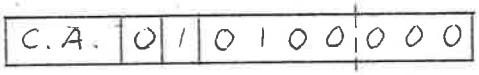


NO OPERATION (Permits any console to return data)

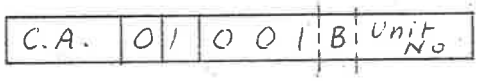
INPUT CODES



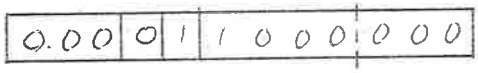
DATA FROM CONSOLE



CONSOLE REJECT



CONSOLE STATUS REPLY (shows selected unit no
B = 1 : console busy)



NO DATA

Fig 1.6.2 Remote Console Data Formats

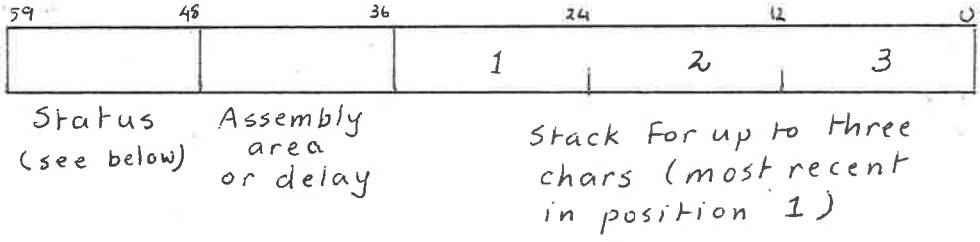
C.A.I. System.

The third program is the remote console driver. This permits the central processor to communicate with up to the maximum complement of 512 consoles, and runs in one peripheral processor. It continually checks the performance of the consoles, reporting any malfunctions to the operator; except in very severe cases, recovery is automatic. Under this program, limited testing of any one data transmission link is possible without affecting normal activity on the other links. In order to allow for maximum transmission time without sacrificing its cycle time, the program processes one set of input data, producing a new set for output, while the previous output data is being transmitted; this avoids the processor being idle during the transmission time, and them delaying the interface while it prepares new output data.

The driver communicates with the central processor through a buffer in central memory. This has one (60 bit) entry for each console; any entry can contain up to three characters (see Fig. 1.6,3) More detailed tables of console status are kept internally by the driver.

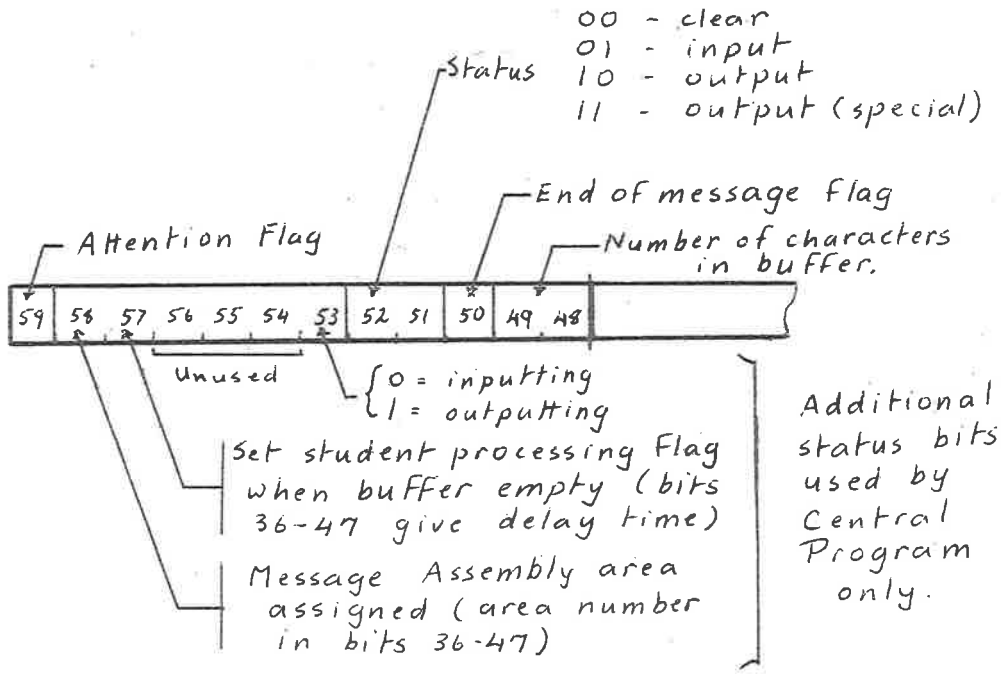
Functions such as unit selection and changing ribbon colour to red for output and black for input are carried out automatically by the driver, which also locks the keyboard should the central memory buffer become full on input.

As well as normal output, the driver supports output in "special mode" allowing the sending of select codes for the control of devices other than the printer. This mode is also used by the



CONSOLE BUFFER FORMAT

(one 60 bit buffer per console)



FORMAT FOR STATUS BITS

Fig 1.6.3 Remote Console I/O Buffer

system for certain messages where the carriage return normally sent at the end of a message before the keyboard is opened is not required.

1.7 THE CENTRAL PROGRAM.

The heart of the system is the program which runs in the central processor, and controls all course presentation. A number of tables are used to keep track of the activities of all students and consoles in the system; the layout of the major tables is shown in figure 1.7.1 .

Every student active at a console in the system has a "student working record" which contains most of the information relevant to his activity. This record may be resident in the central memory, or stored on the disc in a fixed place which is a function of the console number. During normal processing, this record consists of a header part, containing pointers and counters, and a body which is a copy of the current segment of the student's course (figure 1.7.2)

There are only a few central memory buffer areas for this data, and in a full scale system a simple scheduling algorithm would determine which data records would be kept in core; however, since only four consoles were actually built, this part of the system has not been implemented, although the main mechanisms are present in the program: every time the working record is altered, it is copied back onto its disc area; most alterations only involve the header, and if this is the case only that part is copied - it is only a change of page which neccessitates copying out the whole record. As a rough estimate, the console would not again need processing for an average of about four seconds (this estimate

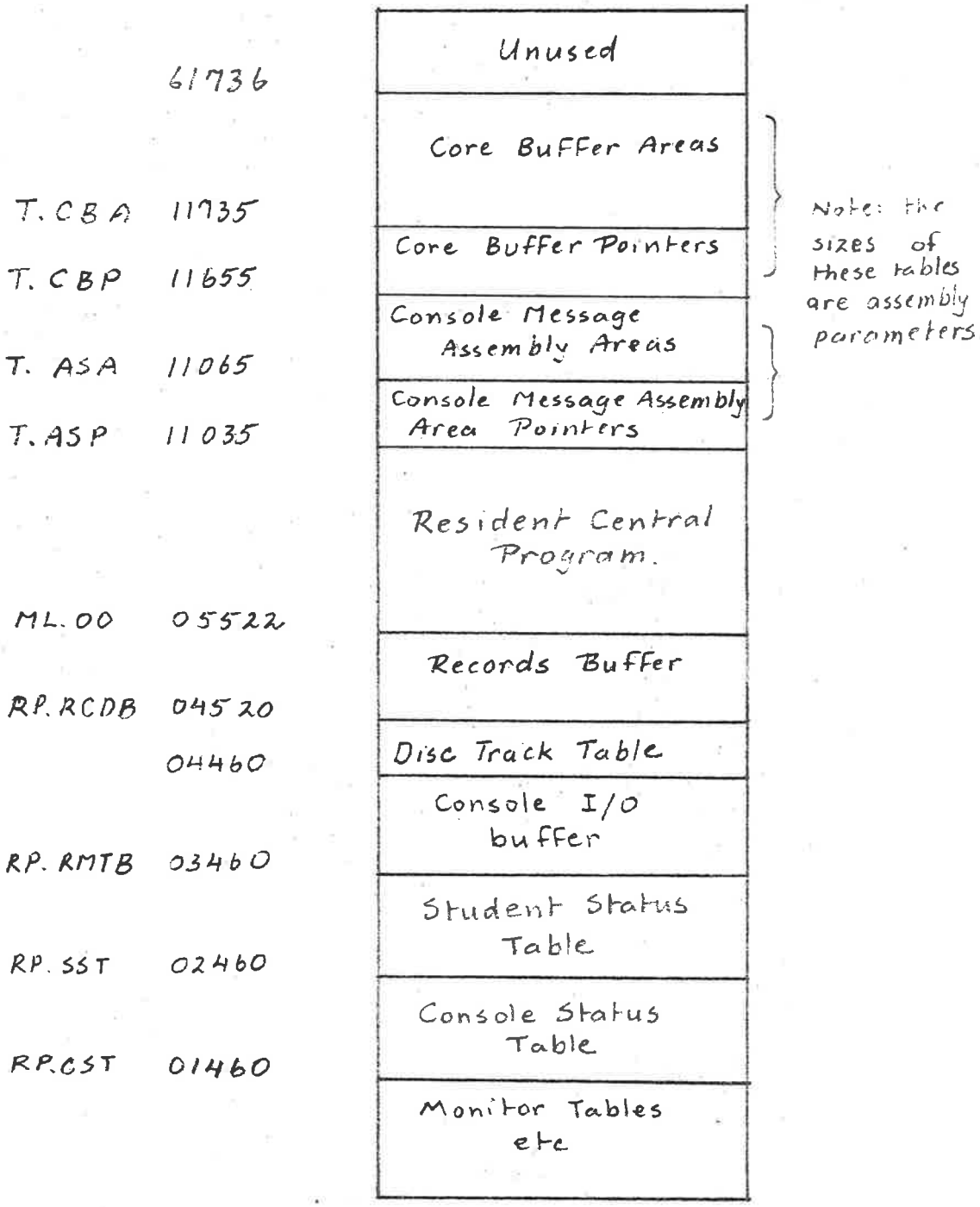


Figure 1.7.1 CENTRAL MEMORY USAGE.

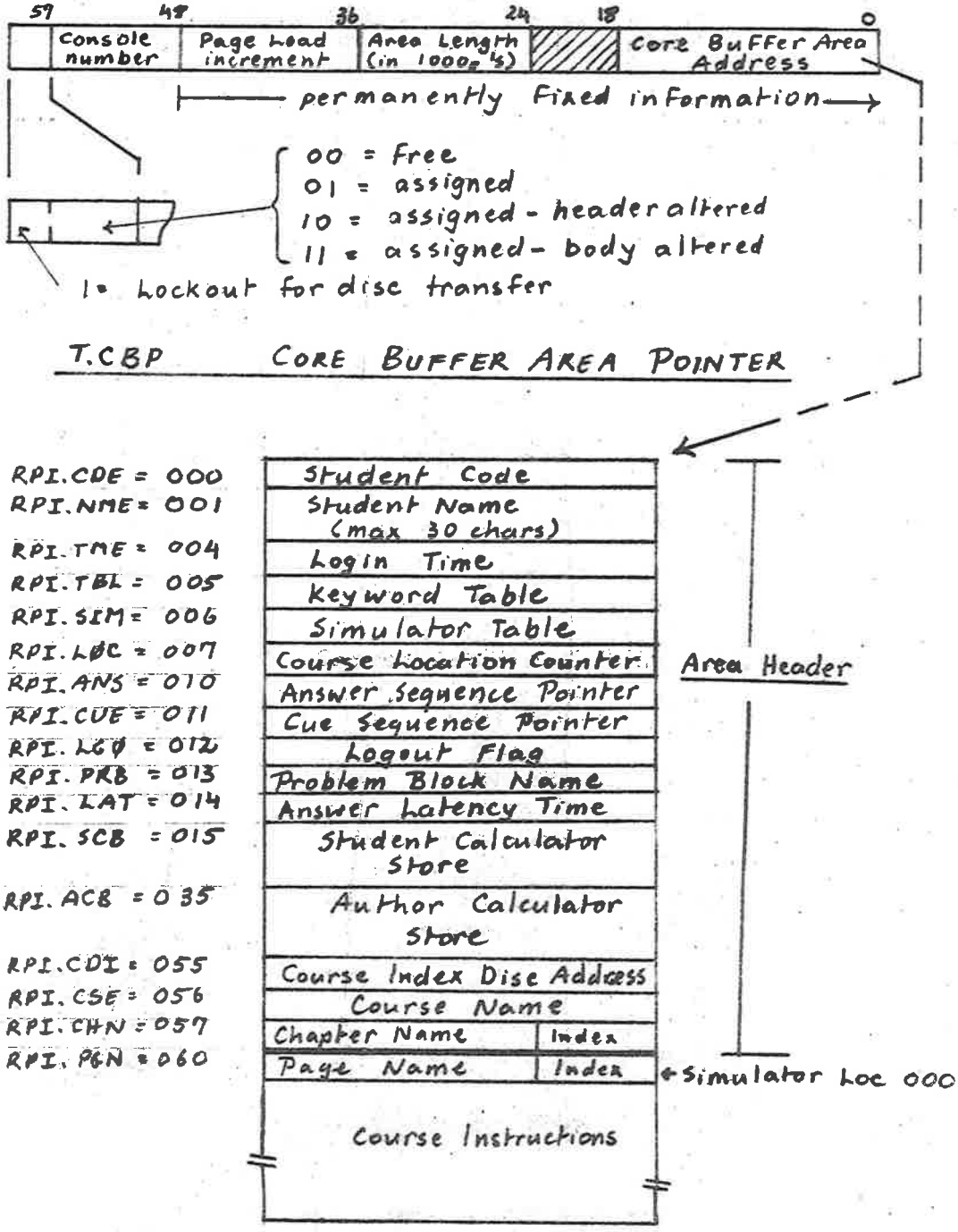


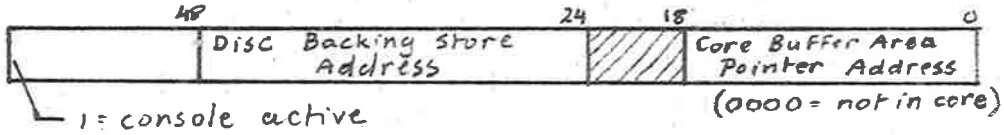
Figure 1.7.2 CORE BUFFER AREA FORMATS.

is based on the assumption that most courses involve mainly data to be printed - a full block takes five seconds to print - and questions, which might average five to ten seconds for an answer; this is counterballanced by branch instructions, which take a negligible amount of time) so that the overhead of possibly unneccessary transfers would not be high. The existence of a very recent copy of this data in a fixed place on disc would also facilitate recovery in the event of a failure.

A computer with the minimum 32K memory could run approximatel fifty consoles without requiring the swapping routine.

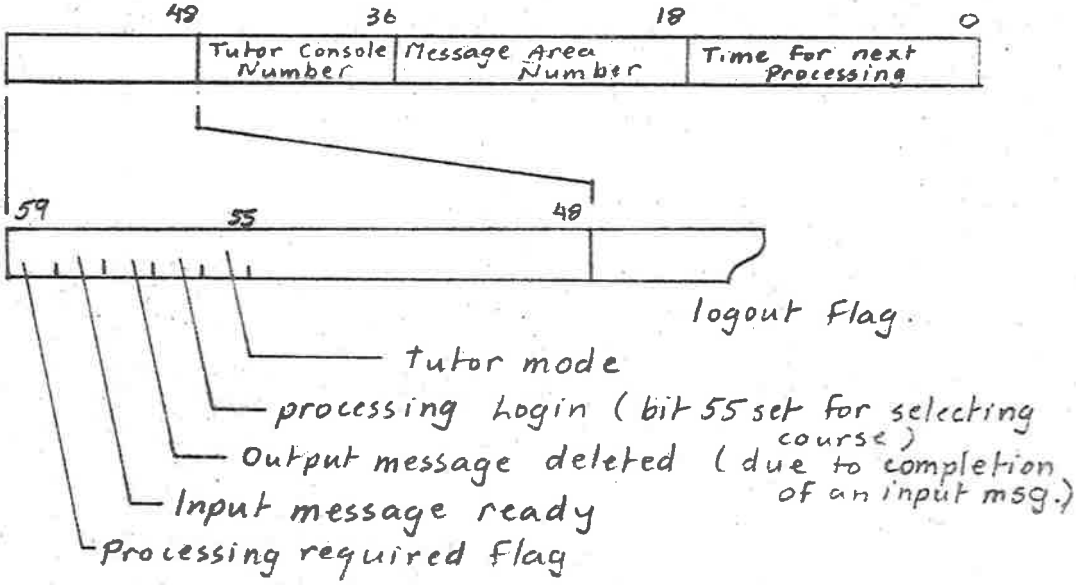
Only consoles with their data records in core are scheduled for processing; each console which requires processing has a bit set in its student status table entry (figure 1.7.3) together with the time (in seconds) at which it should be processed. This allows delay times and maximum times for questions to be set by a course. On each cycle of its processing loop (figure 1.7.4) the program selects from all the consoles requiring processing at the current time the most urgent, and handles it.

A console which is not logged in does not have an associated data record, and so if there is no console to be processed, a second phase is entered, checking for any inactive console with a message ready; such an input message should be a command word - probably #LOGIN, although a few other commands are available to an inactive console, including a request for the console status. The processing sequence is shown in figure 1.7.5 . In a swapping version, this second phase of the master loop would also attempt to schedule a console to have its record read



RP.CST CONSOLE STATUS TABLE

This table shows where information relevant to the console is stored. Note disc addresses are assigned at assembly time.



RP.SST STUDENT STATUS TABLE

This table gives details of the activity of the student at the console.

Figure 1.7.3 MASTER STATUS TABLE FORMATS.

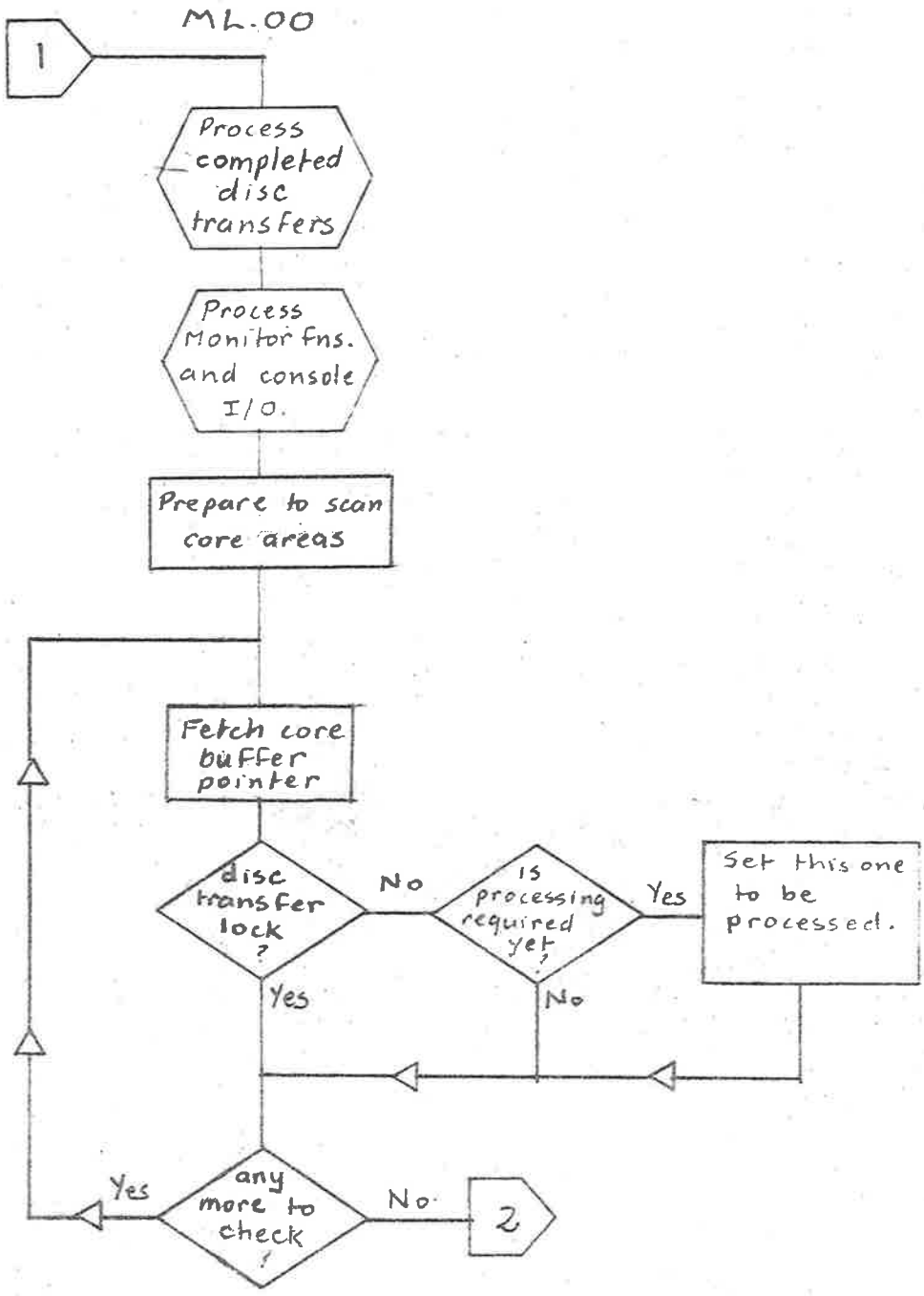


Figure 1.7.4 (a) RESIDENT CENTRAL PROGRAM.

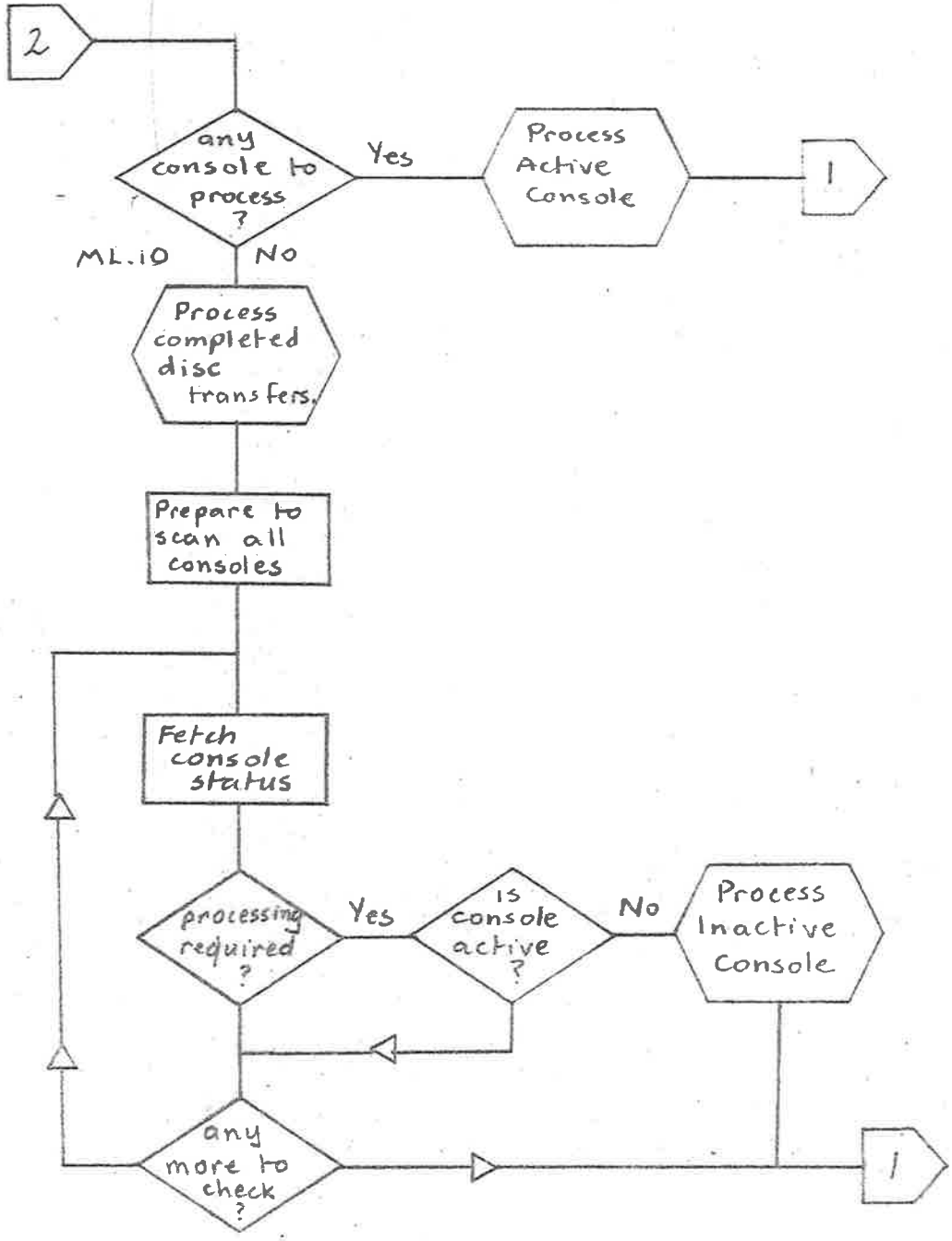


Fig 1.7.4 (b) RESIDENT CENTRAL PROGRAM.

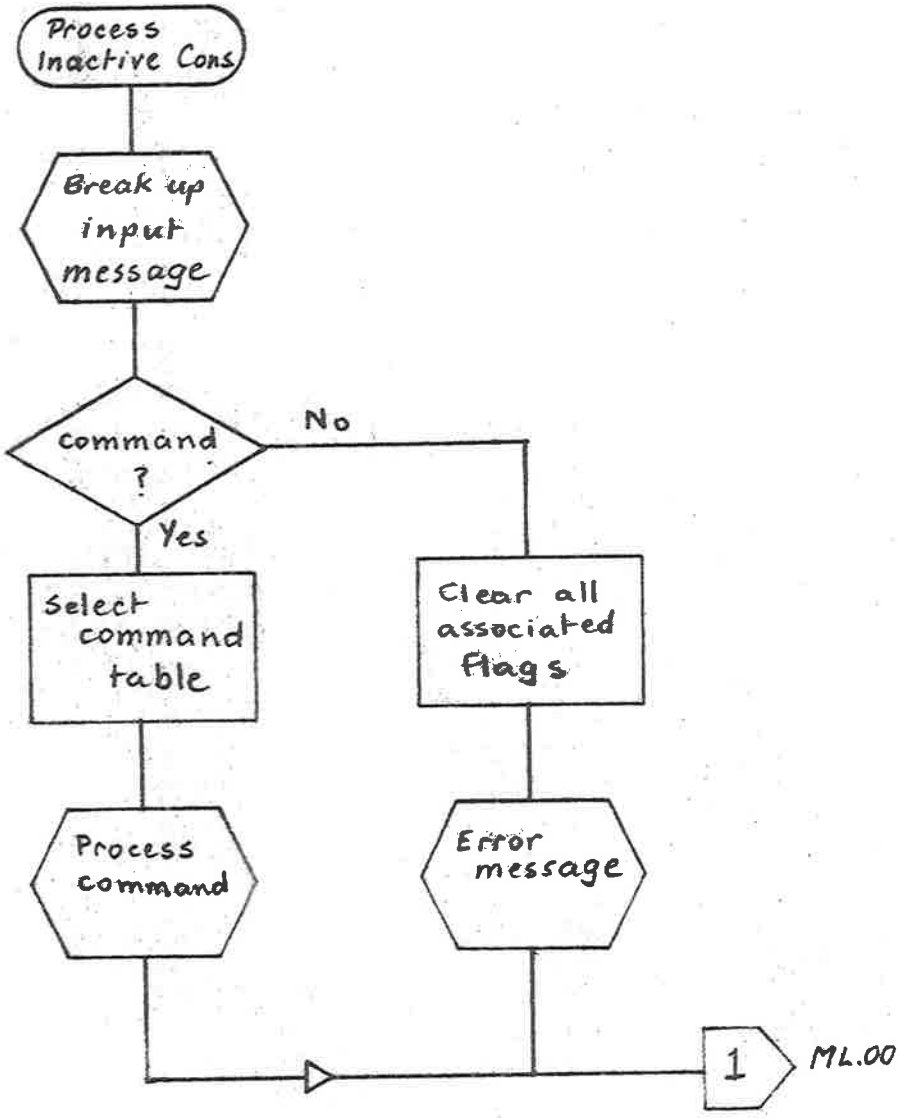


Figure 1.7.5 PROCESSING SEQUENCE - Active Console.

back into memory, where it could then be processed.

As well as course instructions, the completion of an output message can cause a request for processing to be set, thus placing the console in the queue for a core area and eventual processing. The processing sequence is shown in figure 1.7.6 ; note that the term "active console" includes a console whose initial request for a login has been acknowledged, and also one on the verge of being logged out.

Since it is possible for a student to complete an input message after the system has scheduled an output message for the console, but before the message has been sent, special care is taken to restore the status of the console should this happen, so that it does not lead to anomalous results. An input message which is only partially completed, however, is simply deleted when output is scheduled.

The processing sequence also contains a test for "tutor mode", which permits two consoles to carry on a conversation through their typewriters, the central program simply acting as a message exchange. This was the first step towards implementing the supervisor facilities, which are not fully present in the current system.

All console messages are stored in the console code, using one twelve bit byte per character; a message unit may be up to eighty characters long, and consists of a header, showing the length of the message, whether it is to be sent in normal or special mode, and what is to be done on completion, and a body which contains the console characters completion actions may include selecting input mode on the console, and requesting processing of the console with a delay of as much as an hour.

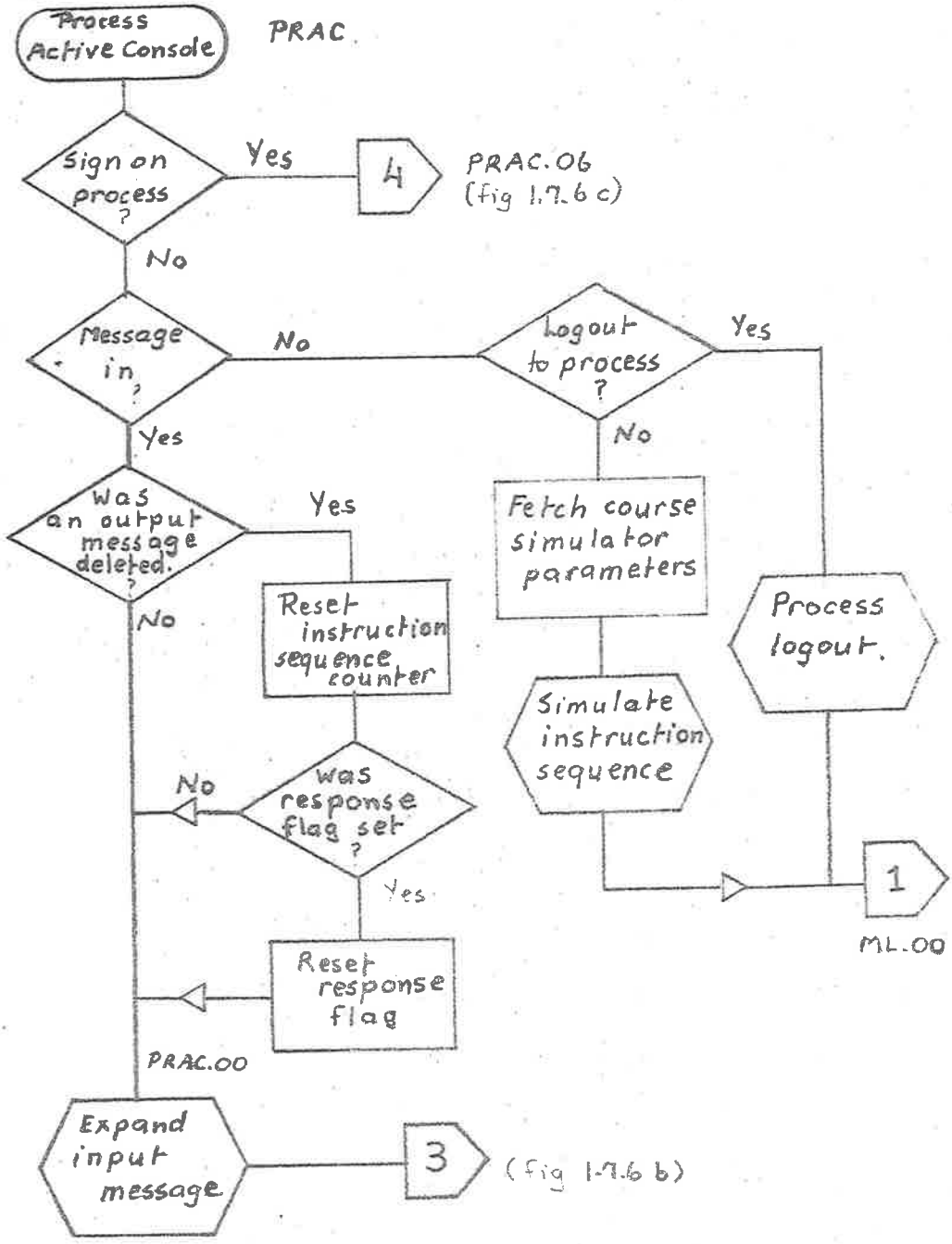


Figure 1.7.6 (a) PROCESSING SEQUENCE - Active Console.

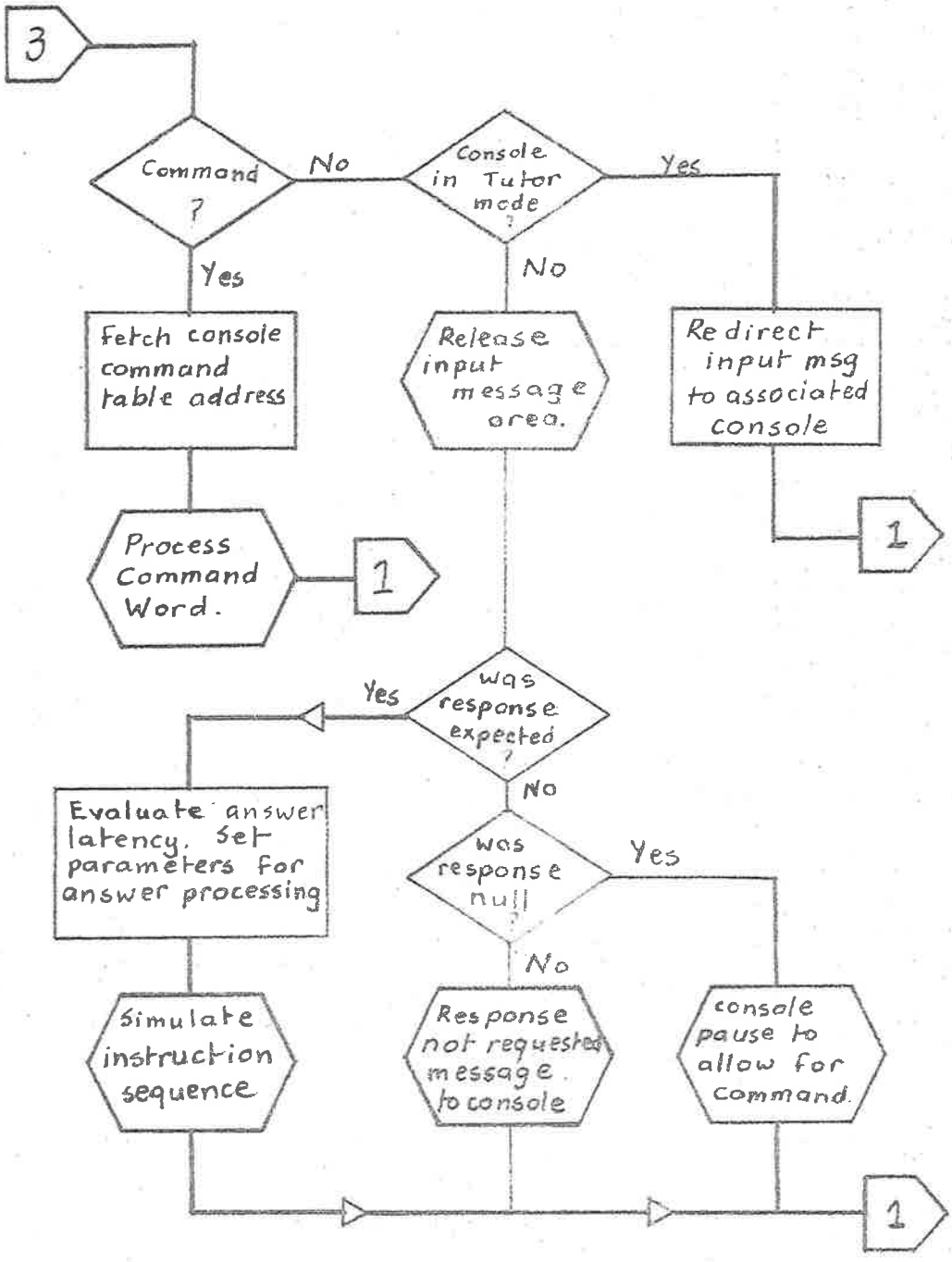


Figure 1.7.6 (b) PROCESSING SEQUENCE - Active Console (cont).

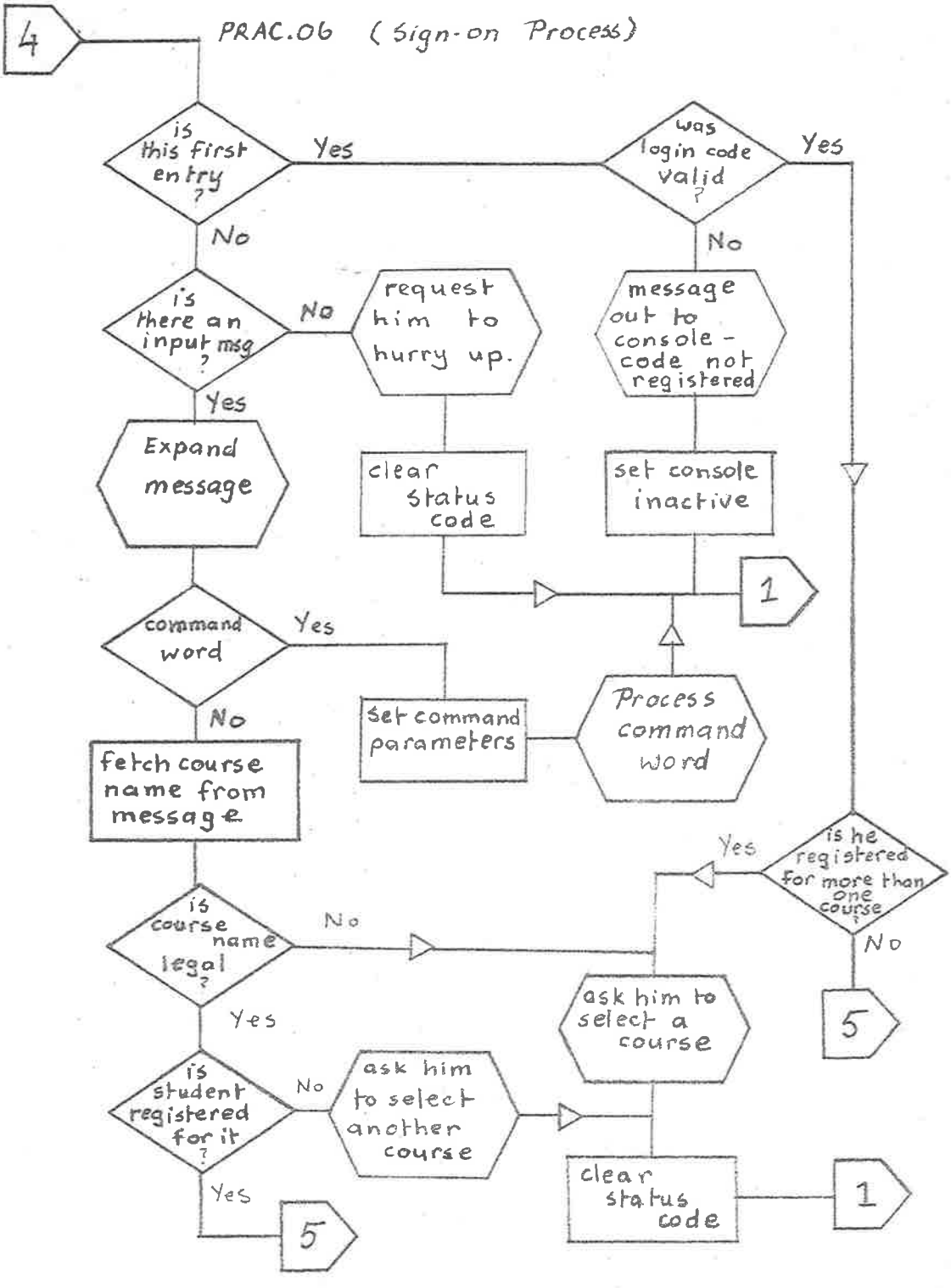


Figure 1.7.6 (c) PROCESSING SEQUENCE - Active Console (cont).

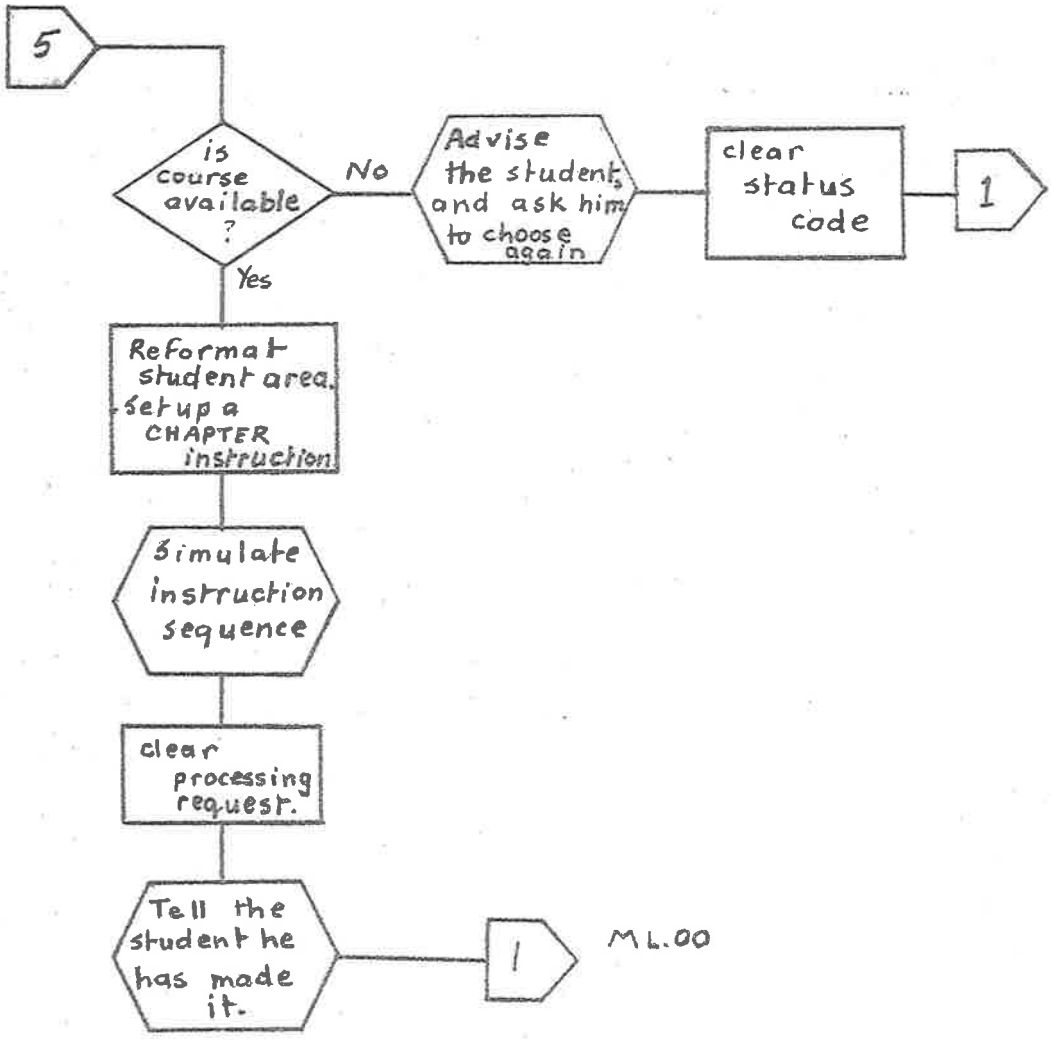


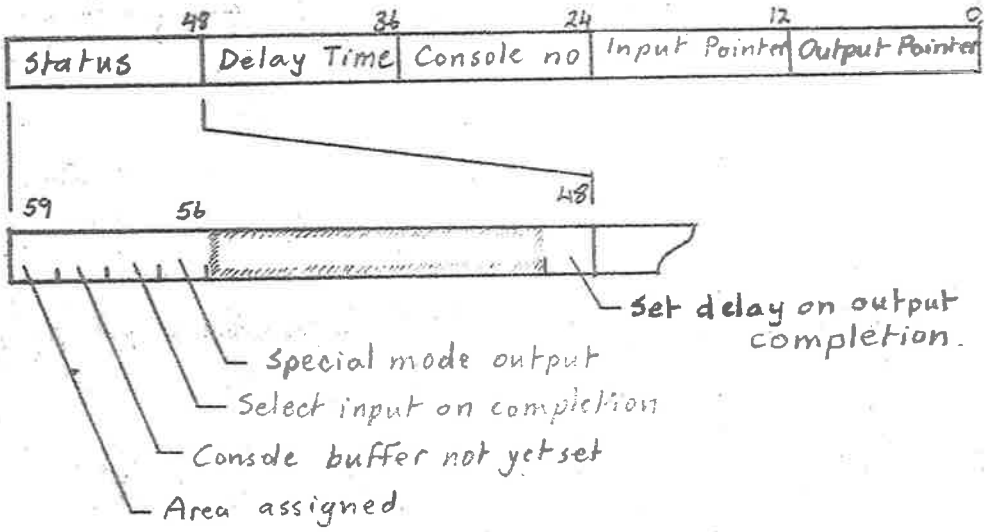
Figure 1.7.6 (d) PROCESSING SEQUENCE - Active Console (cont).

An output message may consist of several units, but input messages are automatically terminated on the eightieth character.

All messages, both input and output, are stored in the message assembly areas (figure 1.7.7). When a new input message is started, it is assigned one of these assembly areas, and characters are moved from the I/O buffer into it; if no area is available, the console buffer can hold the first three characters, after which the keyboard will lock until an assembly area is found. Editing of the input line is done by the central program as it copies characters from the buffer to the assembly area. On completion of the input message, the console is scheduled for immediate processing.

The formats of input and output messages are identical; by simply setting a few bits in the header, an input message immediately becomes an output messagedestined for another console. System messages are stored in the same format, and are simply copied into an assembly area with the appropriate console number preset in the header; the same is true of text from any course.

When input messages are being processed, they are first copied into a dual buffer area, where they are stored one character per word; one area holds the original console code, and the second holds a display code (the standard 6400 internal character code) translation. The display code version also includes some additional bits to indicate whether the code is alphabetic, numeric, or punctuation; most processing is done using this code because it does not involve case changes.



T.ASP MESSAGE ASSEMBLY AREA POINTER

Note: Input & Output pointers give word and shift counts relative to the start of the sixteen word message area (with a capacity of 16*5 = 80 chars). Format is such that if the console no. is reset in bits 24-35, and bit 58 set, an input message may be automatically redirected to another console.

Fig. 1.7.7. MESSAGE ASSEMBLY AREA.

1.8 COMMAND PROCESSING.

Commands are recognised by the initial character # ; this overcomes possible problems due to either conflict between command words and responses to questions, or misspelling of a command. The set of commands available to any console isdefined by a table whose address is stored in the header of the student working record; thus author language statements to restrict the range of available commands could be readily implemented.

The table entries are to the addresses of subroutines; each subroutine contains the command name and the code to process it. The input line, from the character after the # up to (but not including) the first non-alphanumeric, is compared with the command name, and the routine is entered if they match; this mechanism permits commands to be abbreviated to any desired extent, the order of the table entries defining which will be used if the resulting abbreviation is not unique.

Most commands leave the console idle - that is with no further processing scheduled - so that it is up to the student to initiate further action; the major exception to this is the #GO command. Other commands available include:

#STOP	which is a do-nothing command.
#STATUS	which is in fact three separate commands (one for each of console free, being logged in, and active.) and returns the appropriate staus message.

#LOCATE enabling named chapters and pages within a course to be selected, as an author aid and for debugging purposes.

#COMPUTE giving the student access to a powerful desk calculator with his own set of sixteen registers; the first register, named $, contains the result of the last calculation, and the others, named A to Ø, may be used freely to hold results. Calculation is all in floating point, with an INT function permitting integer values to be obtained.

```
A=B+2*(INT(C/D)+3.24)
A=B=C=D=E+F
3+4*(B=C)
```

are typical expressions for evaluation. The registers are saved from session to session.

as well as, of course, #LOGOUT and #LOGIN.

1.9 COURSE PROCESSING.

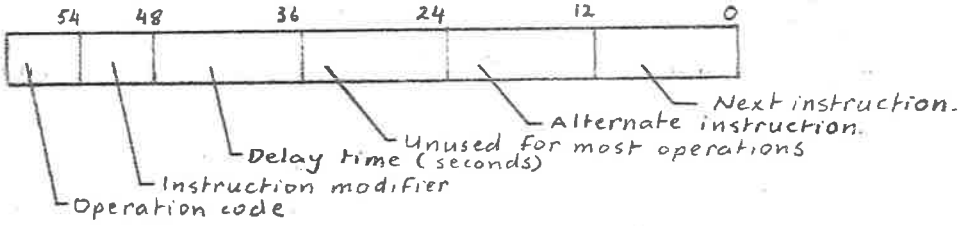
A course is divided into a maximum of sixtyfour chapters, each with up to sixtyfour pages. A page may, in its assembled form, be as long as four hundred and sixty (decimal) words; a page consisting solely of text could contain up to two thousand characters of data for typing on a console.

The course assembler codes the instructions and writes the

result on tape; this process is essentially one-for-one - that is, one author language instruction results in one course simulator instruction - with the exception of text, where a single block of text in the source may become several units in the coded form. The tape is copied by the system into a random file on the disc; this file is given the name of the course; each chapter has an index giving the name and location of every page within it, while every chapter is named, and the location of its index is given, in an overall course directory. Memory space being limited, these indices are stored with the course on disc.

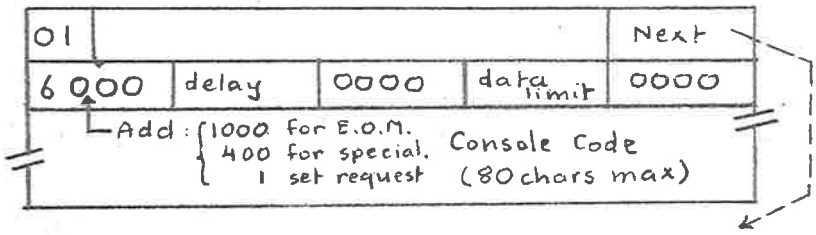
The course instructions are processed by a simulator which breaks up each instruction into its constituent fields (figure 1.9.1) and branches to the operating code. The address of the branching table to be used is a parameter to the simulator, and is stored in the header of the student working record, thus permitting the use of several quite distinct languages within the system, although only one has been implemented.

The conventions for the use of the instruction fields are shown at the top of figure 1.9.1 ; a non-zero modifier usually indicates that a request for further processing is to be set, with a delay as given in the delay field, while a zero modifier indicates that further processing is to be undertaken only on a response from the console. The simulator uses a location counterstored in the header of the student working record; this is normally set from the "next instruction" field. All addressing is done relative to the start of the page; the first word contains the page name, and the first actual instruction in the

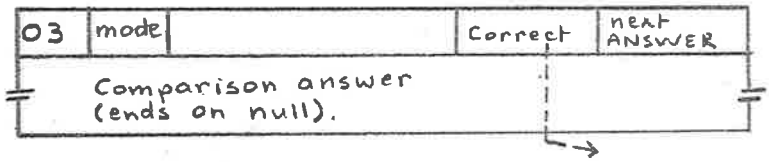


TYPICAL COURSE OPERATION FORMAT

TEXT

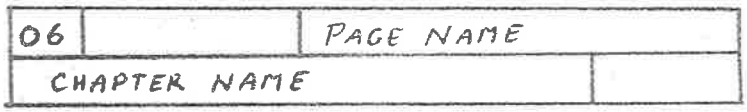


ANSWER



PAGE (note: same format used for NEXT CHAPTER, BLOCK, RECORD)

CHAPTER



COMPUTE

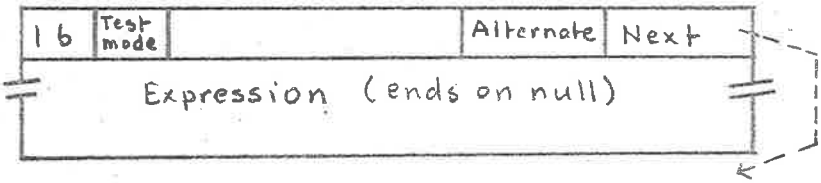


Fig 1.9.1 COURSE INSTRUCTION FORMATS.

page is at address 0001 .

The instructions and their actions during normal processing are:

00 SIGN OFF

The console is immediately logged out. At the next session, the student will recommence at the start of the current page.

01 TEXT

Immediately after the instruction is a message header word, followed by a block of console code. This is copied into a message assembly area, and the console number set into the header. No processing request is set, although the message header may cause one to be set after the data has been typed.

02 CUE

A question mark is typed at the start of a new line on the console, and a flag is set to show that a response has been requested. The "next instruction" field points to the first answer supplied by the author. The location counter is incremented by one.

03 ANSWER

The location counter is set from the "alternate instruction" field, and the next instruction is processed immediately. This is equivalent to ignoring the instruction.

04 BRANCH

This is a branch within the current page. If the "spare"

field is non-zero, a record is made of the student's current position. This instruction may be a genuine branch, a pause, or simply a no-operation.

05 PAGE

This is a branch to a page within the current chapter. The low order thirtysix bits of the instruction contain the page name, left justified and in display code; if this is zero, the next page after the current one is required. Processing of the console will continue with the location counter reset to the start of the page as soon as the disc transfer is completed.

06 CHAPTER

Similar to the PAGE instruction; the chapter name is in the word immediately after the instruction. A zero page name indicates the first page of the chapter.

07 GIVECUE

This instruction is only valid after a question has been asked and answered. If the modifier is zero, it causes a branch back to the last CUE instruction issued; a non-zero modifier causes a branch to the instruction immediately after the CUE. The next instruction is processed immediately.

10 TUTOR

This instruction was intended to enable a course author to assign the student to a tutor or supervisor, and to send the

tutor an explanatory message. It currently acts as a TEXT instruction.

11 DEVICE

This is currently a dummy, as consoles do not contain any special devices. Note that text instructions could also be used to control special devices; the additional instruction would have been a more compact way of storing the commands.

12 BLOCK

This instruction sets a block name (from its lower thirty-six bits) and recording mode (from the modifier field) for the recording of answers. The location counter is incremented by one, and the next instruction obeyed immediately.

13 RECORD

A record of the student's position and time, including an identifier from the low thirtysix bits of the instruction, is made. The location counter is incremented by one.

14 CUE

This is equivalent to the 02 CUE instruction, except that a W is printed instead of a question mark; it is used in the WAIT instruction in the author language.

15 NEXT CHAPTER

Similar in format to a PAGE instruction, the given page is

taken from the chapter following the current one.

16 COMPUTE

This instruction permits an expression to be evaluated, and a branch to be made on the result. The expression is stored in display code, ending on a zero (null) code, immediately after the instruction. The author has a set of sixteen registers (see student command #COMPUTE) which are saved from session to session. The result can be compared with zero in any of the six possible ways, depending on the contents of the modifier field, and either the next or alternate branch taken; a zero modifier means that no test is to be made. The calculator routine uses an algorithm suggested by Sanderson (private communication), and permits operations of assignment, addition and subtraction, and multiplication and division, as well as functions of one or two variables.

17 END

The student has completed the course; he is signed off, with an indication in the sign-off record that he should be deregistered for this course.

20 ERROR MESSAGE

This code is used by the disc driver to indicate a non-existent course segment; it sets the logout flag and acts as a text.

All other instruction codes are illegal, and result in a request to the student to logout of the system. Note that in instruction execution, normally only one instruction is executed at a time; even a simple branch with zero delay does not cause the next instruction to be obeyed immediately; the request flag is set, and the console re-enters the queue for processing. Exceptions to this rule are the ANSWER, BLOCK, and GIVECUE instructions.

When the student responds to a question, the instruction codes have a different significance (achieved quite simply by using a different instruction table). The current location counter is saved to permit possible later processing of a GIVECUE instruction, and a new value set which was the "next instruction" field of the CUE. At this stage the student response has been expanded out and converted into display code. Each answer points to the next one in sequence via its "next instruction" field, and to the next instruction to be executed when a match is made via its "alternate instruction" field; the answer for comparison (if there is one) is stored after the ANSWER instruction.

Simulation proceeds with any instruction except an ANSWER being illegal; each ANSWER in turn is compared with the student's response, the method of comparison being determined by the modifier field; if there is no match, the instruction addressed by the "next instruction" field is processed immediately. The last ANSWER instruction has a zero "next instruction" field, and if that one does not match, the system responds with the message "incorrect.", and simulates a GIVECUE instruction; the message is suppressed if the student response was a null line,

```
PAGE,T34A.
            Let us find out how much you know about the structure
      of English sentences.
PROBLEM (M=3) NAME
            What is the subject of the sentence
            "The old man sat on the bench."

CUE (T=40)
            The subject of a sentence is found by asking who? or
      what? is spoken about.
CUE (T=30)
            Who sat on the bench?
EXPLAIN.
            The subject of the sentence is "The old man", because
      this answers the question "Who sat on the bench?"
      Let us try some more examples on this.
GO TO PAGE (T27A)
```

Cue Sequence

```
ANSWER (M=2) *THE OLD MAN*
      That is correct.
GOTO PAGE (*)
ANSWER (M=2) *MAN*
ANSWER (M=2) *THEMAN*OLDMAN*
            You are nearly correct, but that is not the complete
      subject.  Now tell me the whole subject.
WAITCUE.
ANSWER (M=2) $THEBENCH$BENCH$
      No, that is part of what is said about the subject.
ANSWER, #NO#
GIVECUE.
            :
ENDANS (M=0)
      How about a little hint to help you.
GIVECUE.
```

Answer Sequence

Figure 1.9.2 A SAMPLE PAGE FROM A COURSE.

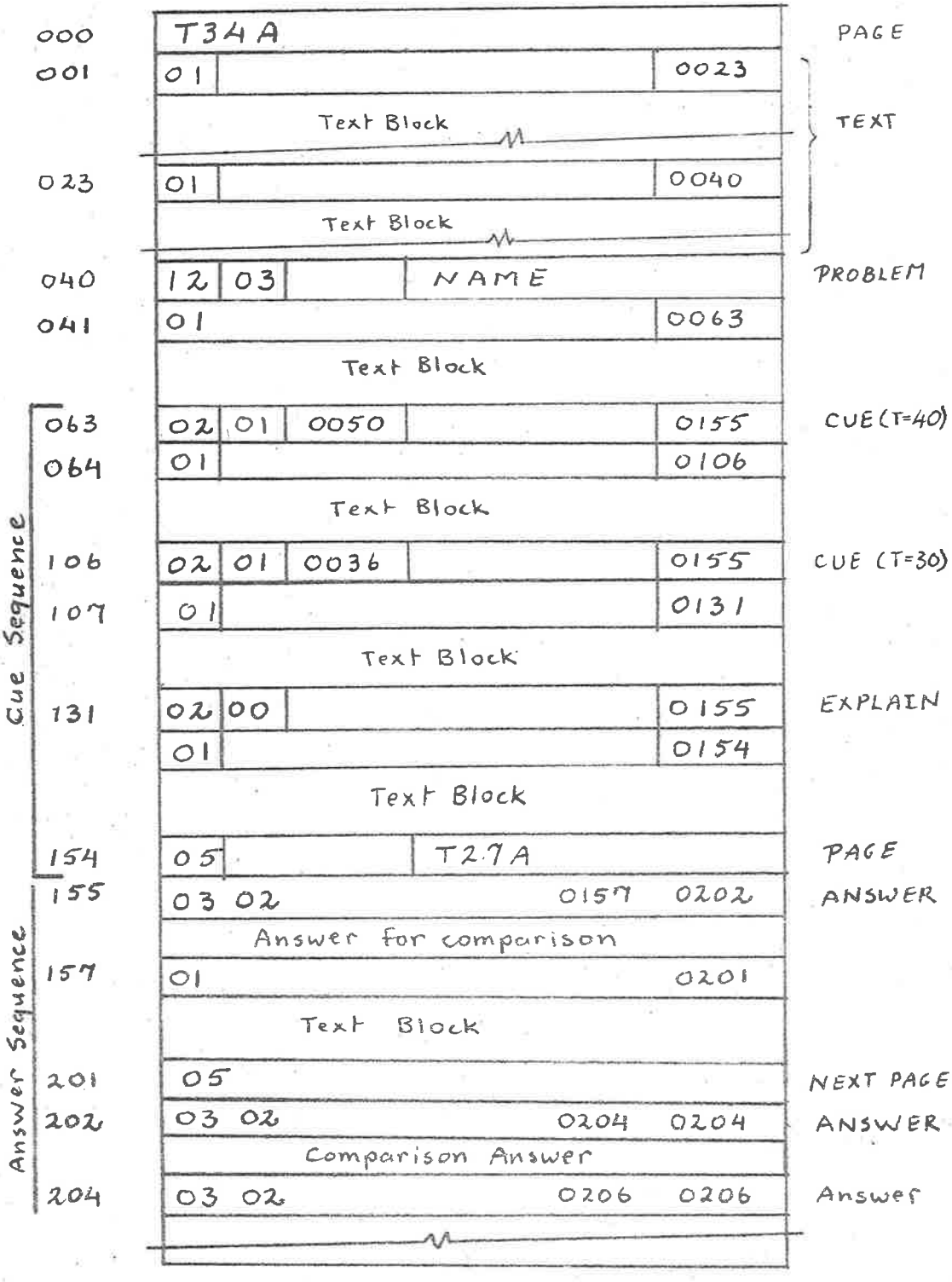


Fig 1.9.3 AN ASSEMBLED PAGE.

(see Fig 1.9.2 for source code.)

thus giving him a convenient means of asking for any help that the author has provided. If an answer is matched, the location counter is set from the "alternate instruction" field, and a request for (normal) processing entered.

Various forms of recording are possible, including recording of the student's complete answer, or simply the number of the answer that was matched, depending on the mode set up by the last BLOCK instruction.

Currently, only a few modes af answer comparison have been implemented; mode zero will match any input, and other modes are based on character-by-character comparisons, with various classes of characters being ignored. There is provision, however, for over fifty further answer comparison modes; it would be particularly simple, for example, to evaluate the student's answer as an expression, and compare the result within specified limits, using the desk calculator, and many further possibilities could be implementedwith varying degrees of difficulty. Further types of instructions, for example to allow editing or transforming the student's response, could also be permitted to appear in the ANSWER sequence.

1.10 LOGIN, LOGOUT, AND RECORDS.

When a student uses the #LOGIN command, he is assigned a core area, and the login processor is called into a PP to search the student master tape.

The student master tape consists of one or more files, each commencing with an index record and ending on a file mark. The second character of the student's five character registration code designates

the file in which the entry should appear. This format permits the use of the hardware "sense for file mark" operation available on the tape transport, thus enabling the record to be located very quickly within a large number of records. As a rough estimate, a single tape could be used for approximately four thousand student records. Although there could be up to thirtysix files, with up to five hundred and ten records per file, a reasonable compromise might be twenty files of up to two hundred records each.

The format of the student master record is shown in figure 1.10.1 . The short header contains the student's code, his name, two words which might be total time used and time not yet paid for (they are reserved for accounting purposes, and not referenced by the system), the student's block of sixteen calculator registers, and a status word which contains the number of courses for which the student is registered. A student may not be registered for more than twentythree courses; this limitation is required by the restricted length of the core area into which the record is copied. The status word would also indicate whether the record was that of a student, author, or supervisor. Following the header is a block for each course, containing course name, current positio and the contents of the author calculator registers.

When the central program makes the request to copy this material, it sets the lockout flag on the core buffer area pointer (see figure 1.7.2); on completion of the transfer, this is cleared, enabling the console to enter the queue for servicing; note that it is now considered to be "active" in the system.

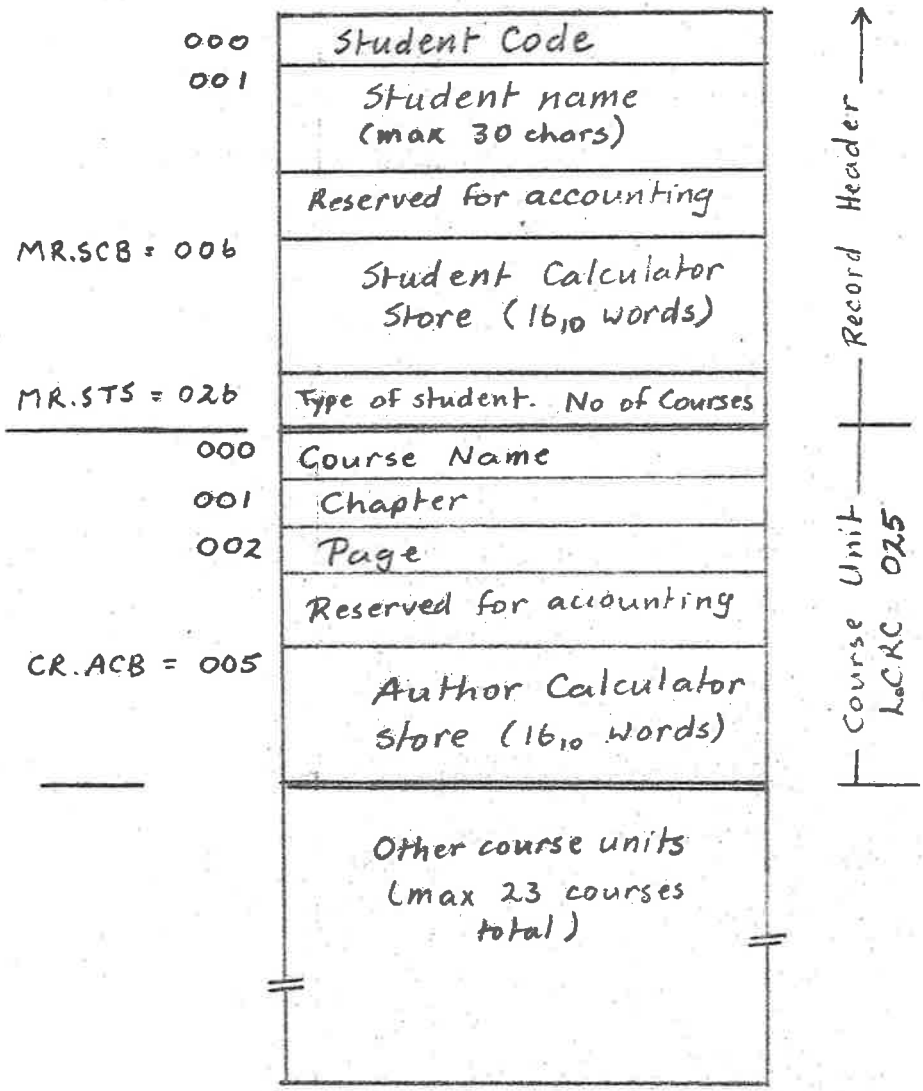
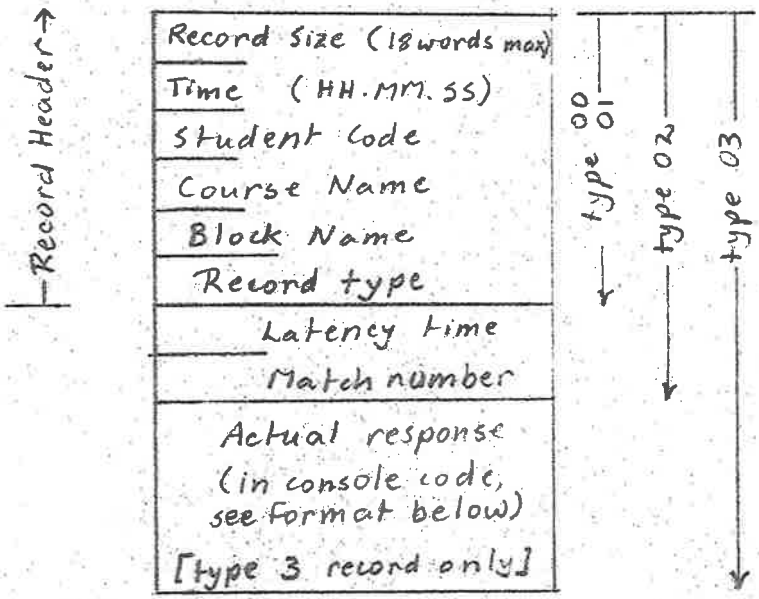


Fig 1.10.1 STUDENT MASTER RECORD FORMAT.

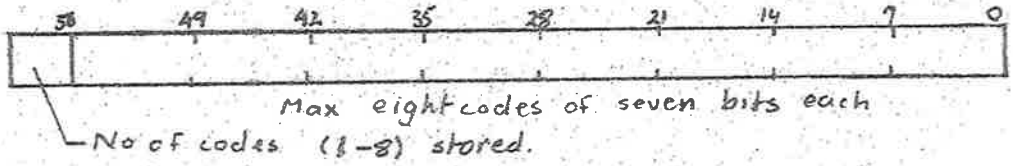
If the student code is not present on the tape, the peripheral program clears the first word of the core area, and clears the lockout; sensing this, the central program responds to the console with the appropriate message, and sets it idle again.

When a course has been selected, either because it is the only one for which the student is registered, or because the student makes a choice of those for which he is registered, the record is rearranged into the format shown in the buffer area in figure 1.7.2 . Note that the course selection is checked for availability before acceptance, and that the student may logout if the course he wants is not available in the system. A PAGE instruction is inserted in the empty body of the buffer area, and the course simulator entered; on return from the simulator, the normal request for immediate processing (which would take effect as soon as the transfer were complete) is cleared, and a message indicating successful completion of the login is sent; this pause gives the student a chance to request help before continuing with the course. For accounting purposes, the time spent on the console from the insertion of the PAGE instruction is recorded; no record is thus made of a student who logs in but does not select a course.

Whenever a request for recording is encountered, the record is entered into a wrap-around buffer in central memory. The format of an entry is shown in figure 1.10.2 ; for simple position records (see section 1.9, operations 04 and 13) the record is six words long; if an answer match number (the position of the matching answer in the ANSWER sequence , zero indicating no match) is recorded, the record is eight



Record Types: 00 = position (from GØTØ)
01 = position (from RECORD)
02 = match number only
03 = answer record.



FORMAT FOR STORING RESPONSE

Fig 1.10:2 STATISTICS RECORD FORMAT.

words long; for recording of an entire response, the record could be up to eighteen words long, the response being recordedin the original console code packed eight characters to a word.

A peripheral program periodically empties the buffer onto magnetic tape; no interlocking with the central program is needed, as the output and input pointers to the buffer are stored separately, and the central program only updates its input pointer after the record has been stored. The peripheral program is called by the central program if the buffer becomes more than half full, and on completion of its task it requests to monitor to reload it after approximately 15 seconds; if this is not adequate to keep the buffer relatively clear, the central program will call it again, thus giving an effective clearance of eight times per minute, and so on.

A logout can be initiated by the student using the #LOGOUT command, by aninstruction in the course, or by an error in the course. It will have been noticed that when a student signs on, he is restarted at the beginning of a page; this enables courses to be modified, with very few limitations, without interfering with students part way through it; locations within the page will be changed, but the page and chapter names are allways referred to symbollically.

Because the student will recommence at the start of a page, it is desireable that he sign-off at the end of a page, to avoid repeating a portion of it. Thus when the student requests a logout, it is not processed immediately, but a flag is set, and he is invited to continue with the current page. Should he not wish to do so, he has the option

of forcing the logout by repeating the command (or of changing his mind by using an #UNLOGOUT command). When any instruction which changes the page is encountered, the page access is processed, and the logout bit set in the student status table entry; this results in the logout being processed when the access is completed.

Course logout instructions are processed immediately; the student recommences at the start of the current page. Course errors cause the logout flag to be set, and a message requesting the student to logout is sent; because the flag is set, the logout will be processed immediately. Course errors are also displayed on the operator console; they should only occur if warning messages from the course assembler are ignored.

The logout is processed by clearing the status tables and calling in a PP program which dumps the header of the core buffer area onto a tape. The format of the resulting record is shown in figure 1.10.3 . The program then clears the reservation of the buffer area, and the logout is complete.

Since logout records and statistics records may be readily distinguished by the contents of their first word, the two may be interspersed on a single tape; this could be improved by having the central program process the logout by entering the relevant data directly into the records buffer. Several interchangeable disc packs have since been added to the computer, and an alternative would be to have the student records on disc packs, updated immediately on logout; the present method could be readily adapted to this.

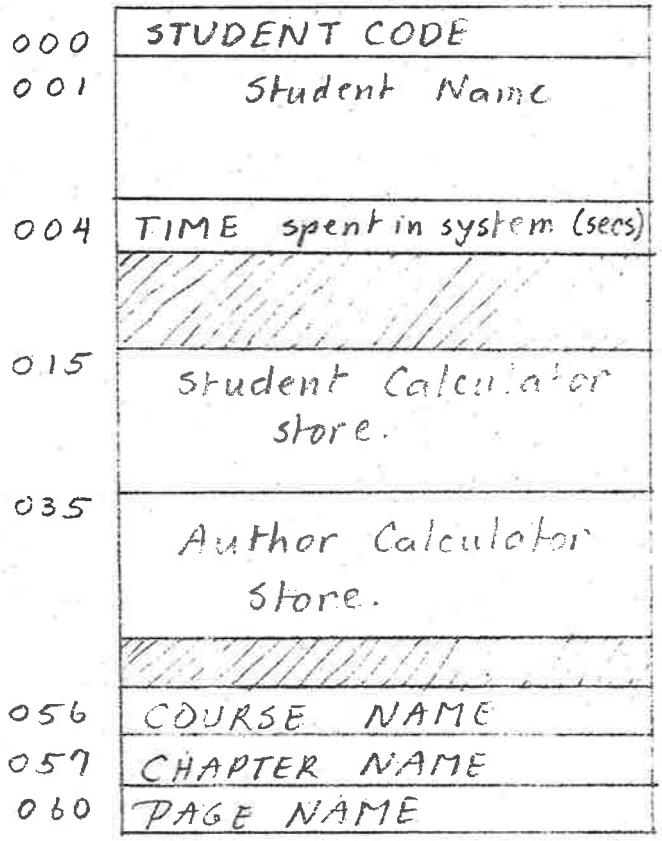


Fig 1.10.3 LOGOUT RECORD FORMAT.

1.11 SUMMARY.

The Adelaide University C.A.I. system is by current standards a medium scale system offering limited facilities at considerable expense. However, at the time of its design and implementation, it was, by comparison with other available systems, quite large.

It was estimated (by Ovenstone, the project supervisor) that the system could be run for approximately $1 per student-console hour, although this figure does not include the not inconsiderable cost of course preparation.

Many of the decisions made during the design phase have inevitably imposed restrictions on the resulting systme, but it is nevertheless a powerful and flexible approach to C.A.I.

THE ANSWER COMPARISON PROBLEM.

2.0 INTRODUCTION.

The correct matching of an actual student response to a question with one of the answers specified by the course author is clearly the central problem in free response Computer-Assisted-Instruction. In order to deal with this problem it is first necessary to state it clearly and in a general form.

In this section, we will examine briefly some methods currently in use, and propose a general statement of the problem, together with some examples to illustrate it.

2.1 CURRENT METHODS OF ANSWER COMPARISON.

Teaching programmes have traditionally taken one of two forms, the linear or the branching programme.

In the linear teaching programme, the student is permitted to construct his own response to the question given, and then compare this with the correct one supplied by the course author. The result of this comparison has no effect on the subsequent presentation of the programme. Because of this, answer matching in computer implementations of linear teaching sequences is not critical, especially as a well constructed frame will usually permit only on correct answer.

The branching programme traditionally uses multiple choice questions, and once again there is little difficulty in a computer system; it is only necessary to decide whether the student responded with A, B, C, or D.

However, it is often desireable to allow the student freedom in contructing his own answer, and it is then necessary to decide which of several branches provided by the course author should be followed. In this situation, it is reasonable to give the author the responsibility of anticipating the import of all relevant responses, but it is not always reasonable to require him to anticipate the exact form which the responses will take, and this poses the major problem in the matching process.

The simplest, and almost certainly the most widely used method is that of character-by-character comparison. In its basic form even a single unexpected space or comma will cause an unwarranted failure in comparison and so the method is usually modified so that only the "important" characters are compared, the others being passed over: thus, for example, by ignoring punctuation, vowels, and the second letter of a double consonant, some measure of independence from misspellings can be attained.

An elaboration of this allows for possible missing or erroneously introduced characters: the two character strings being compared are scanned to find the maximum possible number of matching characters, and a score which is the ratio of this to the number of characters in the longer string is compared with a threshold value to determine acceptance. This method, which is about as sophisticated as a character-by-character comparison can be, will be discussed in more detail in a later section.

A paper by C. N. Alberga ([1] , 1967) discusses these and related methods of comparison of character strings.

However, while it is possible to improve the technique of character-by-character matching, it is not possible to alter the basic requirements that the student must have attempted to reply with the same words or symbols in the same order for matching to succeed, whereas the course author is usually more interested in the meaning of the response than its form.

Some work has been done on the equivalence of mathematical formulae, and in particular the work of K.C. Lee ([16], 1971) on trigonometric proof supervision goes well beyond the minimum requirement of being based on meaning. For natural language answers, however, the selection of keywords from the response is the best available method. When several classes of words are permitted, this method can cope with many cases of synonymy, antonymy and negation and so is to a large extent based on meaning.

It can be seen that answer comparison has used ad hoc methods in the past, with little attempt to form a unified basis for the development of new methods; as a new situation arises, a special method which gives satisfactory results in most cases is developed.

2.2 A STATEMENT OF THE PROBLEM.

An experimental system designed by R.D. Smallwood ([34] 1962, see in particular page 89) used a method which highlights the nature of the problem. The student was presented with a question, and

permitted to constuct a response. When this was completed, he was presented with a set of alternatives, and asked to choose the one which most nearly matched the meaning of his answer. Since one of the alternatives was always a "catch all", an explicit choice was always possible, and the student them entered the number of the matching alternative.

This system thus used a human to carry out the answer matching process; the fact that the human so used was the actual student is irrelevant, as it would be possible to refer the comparison to an independent person. It is apparent that, if this arbitrator is honest, accurate, and familiar with any special terms or symbols used, any answer could be processed in this way.

Using this as a basis, we may state the problem of answer comparison in general terms:

> Given the finite set $S = \{S_1, S_2, \ldots S_k\}$
> (the reference set) of strings in some language,
> the answer comparison problem is, for an arbitrary
> string s (the response), to determine;
>
> 1. to which of the strings of S the string s is closest in meaning.
> 2. whether the degree of closeness in meaning indicates a significant relationship between the strings.

A string may, in the most general sense be considered to be the ordered sequence of actions, as perceived by the computer, which constitute the student's response; in a more restricted sense, it may

be convenient to consider it to be a sequence of characters, or even words, being processed. More precise definitions of some of the terms used in this statement will be given in later sections; at the present stage, intuitive notions of such terms as "language" or "meaning" will suffice.

If the string s is closest in meaning to some string S_1 of the reference set, and the degree of closeness is significant, we shall say that the response matches S_1. If it is not significant, we shall say that s has no match in the reference set. If the response is not in the language from which the reference set is chosen, and in which we assume the arbitrator to be proficient, it would normally be defined to have no match.

The current work will be mainly concerned with the first part of the problem, namely the selection of the string "closest in meaning" to the given string.

2.3 EXAMPLES AND DISCUSSION.

We take a first example, a very simple case where the difference in meaning may be explicitly defined. The language is the set of real numbers, and we can define the difference in meaning between two real numbers x and y to be $|x-y|$ Two numbers will be considered to be significantly related if this does not exceed 0.1.

Let the reference set be

$$\{1.5, 1.7, 2.1, 2.3\}$$

and consider first the response 1.63 which is clearly closest in meaning to 1.7 and since

$$|1.63 - 1.7| = 0.07 \leq 0.1$$

we see that this is significant so that the response 1.63 matches 1.7

A response of 1.85 is also closest to 1.7, but

$$|1.85 - 1.7| = 0.15 > 0.1$$

so that 1.85 has no match in the given reference set. Similarly,a response of -231, although closest to 1.5 has no match. This illustrates the need for the second part of the general statement.

A response such as 'x + y/z' poses a problem in that the definitions of difference in meaning and the significance of the relationship no longer apply. However, the problem is easily dealt with by rejection - the response is not in the language being dealt with.

Another problem case is the response 1.6, which is seen to match both 1.5 and 1.7. In a practical situation, this would usually be resolved by taking the one which appears first in the reference set.

The work of K.C. Lee ([16] , 1971, see especially chapters IV and V) on trigonometric proofs can also be related to this statement of the answer comparison problem; in this case the reference set contains only one string, the result of the previous step in the proof, and the response is the proposed next step. His concept of a small step correspond to the idea of significance of the relationship. This could be extended by including in the reference set anticipated incorrect steps to give the student help in the nature of his error on an incorrect step.

As a last example, we turn to the English language, and consider possible answers to the question "What is an enzyme?". For

A. An enzyme is a catalyst in a biological system.

B. An enzyme is a catalyst in a biochemical reaction.

C. An enzyme acts as a catalyst.

D. An enzyme speeds up a biochemical reaction.

E. Enzymes are formed in biochemical reactions.

F. Many reactions would not take place without enzymes.

TABLE 2.3,1 A reference set for the Question - "What is an Enzyme?"

our reference set, we take the strings shown in table 2.3,1.

Let us first suppose that the question is presented as a multiple choice one, with the six choices in the table presented to the student and labelled A, B, C, D, E and F. In this case a response of A clearly matches the first string, while a response of G has no match.

However, presenting the question in that form would perhaps give the student too much assistance in forming his answer and it might be considered desirable to allow the student to construct his answer and then match it against the original strings. Then if his response is

An enzyme is a catalyst.

it would normally be considered closest in meaning to the third string in the reference set, with a significant degree of closeness, while a response of

A catalyst acts as an enzyme

might also be closest in meaning to the third string but this would not be considered significant.

2.4 SUMMARY

We have seen in this section how it is possible to formulate the answer comparison problem in general terms. The statement is adequate to cover both the simple multiple choice situation and the more complex case of a freely constructed response.

We are now faced with the more formidable task of formulating the solution to the problem.

WHAT IS MEANING

3.0 INTRODUCTION.

Everyone knows what the word "meaning" means, but a closer examination shows that the situation is not as simple as it may appear at first; in fact there are so many definitions of meaning that to attempt to list them all would be unpractical. The classic study of the subject, Ogden and Richards The Meaning of Meaning, lists at least sixteen fundamentally distinct definitions ([23], 1923, p306) and more recent avenues of research have almost certainly added several more.

Osgood, Succi and Tannenbaum in discussing the approach used in The Measurement of Meaning state

> There are at least as many meanings of meaning" as there are diciplines to deal with language and of course many more than this. ([26], 1957, p2)

This does point to one major aspect of the problem; language is a very complex form of behaviour and there are many ways of approaching its vario aspects; proponents of each approach have tended to use familiar terms and give them more specialised meanings, as has occurred in most other areas of scientific endeavour. However, with many related avenues of approach to language, each with its own terminology, there has been an almost irresistable temptation to link the distinct meanings of one word, and eventually to confuse them. This accounts for much of the confusion in the use of the term "meaning" and in order to clarify the problem we will examine briefly some of the different approaches

that can be made to the study of language with a view to selecting an appropriate working definition of "meaning".

There is another aspect of the problem which stems from a quite different source, and it is worth a brief mention here. An utterance is processed by a listener at several levels; thus Miller ([18], 1968, pp72-74) differentiates hearing, matching, accepting, interpreting, understanding and believing. Many of the approaches to language are distinct because they are primarily concerned with different levels of processing, and certainly when discussing meaning it is possible to discuss it at several distinct levels in this hierarchy (irrespective of whether such hierarchy is real in the sense of being distinct and distinguishable processes in the human listener or not). The study by Osgood, Succi and Tannenbaum (op. cit) used factor analysis and found that "meaning" could be accounted for by three independent factors and in commenting on this, Osgood states:

> It should be stressed that these regularly reproducible factors seem to represent the most generalisable and gross representational processes of the human animal... not the precise denotative discriminations of which this animal is capable. The aspect of the meaning of signs indexed by this technique is thus more their affect, connotation or "feeling tone" than their denotative or referential properties.
>
> ([25], 1963, p271)

Thus we may also anticipate that some definitions of meaning will differ chiefly because they place major emphasis on different facets of the same idea. This corresponds to the different criteria which a person might use in comparing different groups of statements; for one group, the chief criterion might be the refernential meaning, for another, the structure, for a third the metre, for a fourth the implications and so on. To claim that the problem is solely one of different disciplines working in the same general area would be a gross oversimplification.

With this reservation in mind, we consider some of the approaches which have been made to the study of language.

3.1 STRUCTURAL LINGUISTICS.

> Linguistics is a descriptive science. It describes the code of any language, the set of distinctions that differentiate alternative messages... The linguist ... is never concerned with what things mean in a referential sense.
>
> (Osgood, [25], 1963, p261)

For many linguists, the "meaning" of a unit is simply the amount of information it gives about the structure as a whole; thus Martin Joos defines the meaning of a morpheme as;

> the set of conditional probabilities of its occurrence in context with all other morphemes.
>
> ([14], 1958)

and this has been the basis of a number of statistical studies

(Rubenstein and Goodenough, [31], 1965; Lewis, Baxendale and Bennett [17] 1967) which have shown correlations between these probabilities and the synonymy/antonymy relationship which is one aspect of referential meaning. Such studies illustrate the tendency to generalise very precise definitions of the term "meaning" outside of their field of applicability; Joos definition, which is essentially based on the concepts of the Mathematical Theory of Communication (Information Theory), is quite acceptable within the frameworks of structural linguistics and it is only when "amount of information" in the communication theory sense is equated with "meaning" in a referential rather than a structural sense, that confusion results. This point is extensively discussed by Bar - Hillel ([3], 1964, ch 16; op. cit, ch 2)

It is not, however, practical to entirely divorce the study of structure from semantic considerations; thus a structural description of a language would not in general be considered adequate unless the structural units (for example, the separation into phrases and clauses) correspond closely to semantic units. Also, many syntactic catagories are based on semantic considerations - for example, the subdivision of the class "noun" into (among others) "mass nouns" and "collective nouns" to explain why some do not take a plural form after the quantifier "some" thus ;

some families (collective)

but

some butter (mass)

Thus while Chomsky agrees that

> The fundamental aim of linguistic analysis
> ... is to separate the grammatical sequences
> ... from the ungrammatical sequences
> ... and to study the structure of the grammatical sequences.
>
> (in Allen and Van Buren[2], 1971 p18)

he also states that

> a linguistic description which treats the grammer and the lexicon as two separate entities without rules to inter-relate them cannot serve as a descriptively adequate specification of the facts of language.
>
> (op.cit. p102)

However, while it must be accepted that there is a close link between the study of structure (syntax) and reference (semantics) structural definitions of meaning will not in general form an acceptable basis for the comparison of answers on C.A.I.

3.2 BEHAVIOURAL PSYCHOLOGY.

The behaviourists and in particular B.F. Skinner, have made a considerable contribution to the study of language. The central theme of their approach is that only behaviour that can be observed and measured is valid data for the consideration of the psychologist. Thus Osgood describes Skinner as

> Rejecting meanings and all other non-observables
>
> ([25], 1963 p250)

A behaviourist does not neccessarily reject the concept of meaning but defines it operationally as the response evoked from the listener.

This approach to psychology is an enthusiastic use of the principle of William of Occam, and is undoubtedly one of the factors which has helped lift pschology from its former unscientific state to its present position (which may be termed pre-scientific or scientific, depending on the point of view). However, the application is over-enthusiastic when applied to the study of language, and Chomsky has criticised Skinner's approach in a penetrating review, stating that

> One would naturally expect that the prediction of the behaviour of a complex organism... would require, in addition to information about external stimulation, knowledge of the internal structure of the organism, the ways in which it processes in-put information and organises its own behaviour.
>
> (in De Cecco, [8], 1968, p325)

Osgood (op. cit. pp249 ff) suggests that the one stage model must be supplanted by a two stage one, and in some cases even a three stage model is neccessary to explain language behaviour; such models neccessarily incorporate the postulation of processes which cannot be directly measured but whose behaviour must be inferred as part of the model.

Thus a purely behaviourist model of language is not an adequate basis for a definition of meaning.

3.3 TRADITIONAL LINGUISTICS

This title might be used to cover the rest of the field, but instead of trying to survey the whole area we shall look at only a few points which are more directly relevant to our topic; a more detailed survey can be found, for example in Ullmann Semantics ([37], 1962)

The traditional linguists have tended to have a more philosophical bent than either of the two previous categories; both the structuralists and behaviouralists are chiefly concerned with describing linguistic behaviour as it exists, whereas the traditionalists prefer explanation to description.

A suitable starting point is the work of C.S. Peirce on "Semiotics", his views are largely reflected by C.W. Morris who divides meaning into several related areas including syntactic meaning - the relation of signs to other signs - and semantic meaning - the relation of signs to their significates ([20], 1946). However, as early as 1923 Ogden and Richards has criticised the "sign - significate" definition of meaning as inadequate ([23], 1923; see also Ullmann,[37] , 1962, ch. 3 for a more modern interpretation); this criticism has a striking modern parallel in the criticism of the behaviourist model by Chomsky and by Osgood, cited in the previous subsection: in essence, while it is theoretically possible to simply relate the sign to either its "evoked response" (behaviourist) or "significate" (traditionalist) unless an intermediate stage is postulated, the model does not explain learning transfer phenomena which are basic to human language behaviour. Such an intermediate stage might be termed "understanding by the listener"

- it is the "thought or referen ce" of the Ogden and Richards model, and the "representational-mediational process" of the Osgood, Succi and Tannenbaum model.

A summary of this position might be that "meaning is something that happens in a listener". One might suppose that eventually it will be possible to measure the neural changes involved, and thus reconcile behaviourists with this point of view; at present, any attempt to measure this directly (as opposed to the indirect methods of Osgood et.al) would result in irreparable damage to the listener, so that the processes must for the present remain as theoretical constructs.

3.4 A WORKING DEFINITION.

> Any definition of meaning should be regarded as no more than a working hypothesis. Its value will depend on how it works: On the help it can give in the description, interpretation, and classification of semantic phenomena.
>
> (Ullmann, [37], 1962, p66)

The discussion in the previous subsections probably implies considerably more order than actually exists in a wide and contentious field; the conclusion however does lead us in a useful direction - if we wish to know what meaning is, we must look inside the listener.

We thus define that the meaning of any utterance to a particular listener is measured by the effect that the utterance has on

the listener.

While the use of the word "effect" would be for too vague to use with a human listener, unless it were interpreted in behaviourist terms, we shall be constructing "listeners" which are data structures, and the problem then becomes the much simpler one of how two structures should be compared.

Before simply accepting this working definition, there are a few points that are worth further consideration. The first is that an utterance,whether vocal or written, has no intrinsic property of meaning; thus this sentence is only a sequence of meaningless marks on paper until interpreted by someone or something, at which stage we can measure its meaning (to that person or thing) in terms of its effect on the interpreter. A second point is that the meaning may differ from person to person; this is a well known phenomenon although,if the individuals concerned are all competent in the language used, the variation should be slight.

3.5 SUMMARY

In this section we have examined some of the different approac to the problem of language and meaning. Of the appraoches discussed, only one, that of the "traditional linguist" is relevant to the problem at hand and since it has been shown that an adequate description of linguistic competence must take cognisance of hypothesised changes within the listener, it is natural to difine meaning in terms of these changes

Before pursuing the implications of this we shall develop

some notation for the language elements under discussion and define some of the terms,including "language" itself, which have up to this point been used rather loosely.

PRELIMINARY NOTATION AND RESULTS.

4.0 INTRODUCTION.

In this section, we develop a notation for describing language elements and define some of the associated terms, such as "language" and "syntax".

The definition of language is a standard one; it is used, for example by Bar-Hillel and Shamir ([3], 1964 ch 7) in their discussion of finite state languages, and also by Berztiss ([4], 1971, ch 4) The notation to be used differs in detail from that used by these authors, but follows substantially the same lines.

4.1 BASIC ELEMENTS.

Definition 4.1,1

An Alphabet is a finite set of symbols.

$$W = \{W_1, W_2, \ldots W_n\}$$

Definition 4.1,2

A string of length M is an ordered M-tuple.

$$S = (W_1, W_2, \ldots W_m)$$

of elements of W.

Definition 4.1,3

Let

$$S = (S_1, S_2, \ldots S_k)$$

be a string of length k, and

$$S^1 = (S^1_1, S^1_2, \ldots S^1_l)$$

be a string of length l.

The Catenation of S and S^1 is the (k+l) tuple.

$$S.S^1 = (S_1, S_2, \ldots S_k, S^1_1, S^1_2, \ldots S^1_l)$$

The set of all strings of a given fixed length m is denoted $\mathcal{S}_m$. Of particular interest are the sets $\mathcal{S}_0$ which contains just one string, the null string Λ, and $\mathcal{S}_1$ whose elements are the strings of unit length. These we identify with the single elements of W which comprise them. The set of all strings, $\mathcal{S}$, is simply the union of all of these sets

$$\mathcal{S} = \bigcup_m \mathcal{S}_m.$$

$\mathcal{S}$ is seen to be countably infinite, and to form a semigroup, with identity Λ under the operation of catenation.

if a string S has length m we write

$$m = l(S)$$

and note that $l(s.s_1) = l(s)+l(s_1)$

We make no assumptions about the alphabet other than that it is finite, and that there is no need to break its elements down into subelements.

While the symbols of the alphabet will sometimes be referred to as "words" this is not intended to imply that they are necessarily what we ordinarily call words; a suitable basis for describing written English, for example, is the set of alphabetic characters, together with a few punctuation characters; it might however, be more convenient to use larger units such as the set of morphemes which also have the useful property of meaning. The definitions and notation can deal with either.

Although it is perhaps not immediately obvious that the set of all words, in the usual sense of that term, in the English language is in fact finite, since natural languages are dynamic, it is certainly clear that a finite set of words is adequate to describe all written English up to the present time. It is not even certain that the number is increasing, as words of words pass out of use (how many people for example know the meaning of the word "mundungus"?) while it is more common to find a new meaning attached to an old word than to find a completely new word introduced (consider, as an example, the new meaning physicists have given the word "plasma")

4.2 LANGUAGE.

Definition 4.2,1

A Language L is any non-finite subset of $\mathcal{S}$.

Definition 4.2,2

A Sublanguage L_1, of a language L is any non-empty subset of L

Definition 4.2,3

Let L_1 be a sublanguage of L.

The (Binary) Expansion L_1^2 of L_1 in L is the set constructed by the following procedure:

(1) Every string in L_1 is in L_1^2

(2) If S_1 and S_2 are both in L_1^2 , and $S_1.S_2$ is in L_1 then $S_1.S_2$ is in L_1^2

We restrict languages to the non-finite case since any finite language can in principle be completely described, and the answer

recognition problem solved by means of a table. Sublanguages, however are allowed to be finite.

The expansion of a sublanguage is defined recursively; The notation L_1^2 is based on the fact that the second rule of 4.2,3 allows for strings to be concatenated in pairs, and provides for the definition of, for example, L_1^3,in which strings can be taken singly, in twos, or in threes, and also higher order expansions. We would have L_1^0 being the empty set, L_1^1 being simply L_1 itself, and the obvious result

$$L_1^0 \subset L_1^1 \subset L_1^2 \subset L_1^3 \ldots . \subset L_1 \subset L$$

However we will use only binary expansions, and the unqualified term expansion can be taken to mean the binary expansion.

Theorem 4.2,4

For an arbitrary string S, it is possible to decide whether S is in L_1^2 by making a finite number of decisions of the form

$$S_i \in L_1$$

or

$$S_j \in L$$

Proof:

(1) Clearly, if $S \in L_1$ then $S \in L_1^2$ and if $S \notin L$ then $S \notin L_1^2$.

(2) If $S \notin L_1$ and length $(S) \leq 1$, then, since a string of length less than one could only be in L_1^2 by virtue of the first rule

of 4.2,3, $S \notin L_1^2$

(3) Suppose now that $S \notin L_1$, and it is of length two or more. Clearly, if S is in L_1^2, it must be because of the second rule. We note that taking either S_1 or S_2 as the null string in this rule does not get anywhere and so we can assume neither is null. Thus if S is a member of L_1^2 it must be the catenation of two strings, both of which must be in L_1^2 and both shorter than S.

If $\ell(S) = m$, then there are (m-1) ways in which it can be split into two substrings, each of length at most (m- 1). Thus the decision for S is reduced to 2(m-1) decisions for strings of length at most (m-1).

This reduction process can be repeated until either S is accepted as a member of L_1^2 or, after a number of operations seen to be considerably less than

$$2^{m-1}\ (m-1)!$$

only strings of length one remain, in which case it follows that S is not in L_1^2

Definition 4.2,5

Let $S \in L_1^2$, and suppose the process described in the above proof results in

$$S = S_1 \cdot S_2$$

$$S_1 = S_{11} \cdot S_{12} \; ; \; S_2 = S_{21} \cdot S_{22}$$

$$S_{11} = S_{111} \cdot S_{112}$$

etc.

where each of the strings is in L_1^2, and the breakdown process results finally in strings in L_1. The steps in this breakdown constitute a Binary Parse of S with respect to L_1

Let us consider an example:

Suppose the alphabet is

{a,b,c,d,}

and the relevant portion of L is

{a,b,ab,ac,bd,abc,cac,bbd,acbbd,abcac,cacbbd,abcacbbd}

while L_1 is

{b,ab,ac,bd,abc,cac,}

Consider the string abcacbbd which is in L but not in L_1. There are seven ways in which this string can be split in two; of these, only three result in pairs of strings in L. The three are

ab	cacbbd
abc	acbbd
abcac	bbd

Taking the first one, the first part is in L_1 and the second part may be split in five ways, but only one,

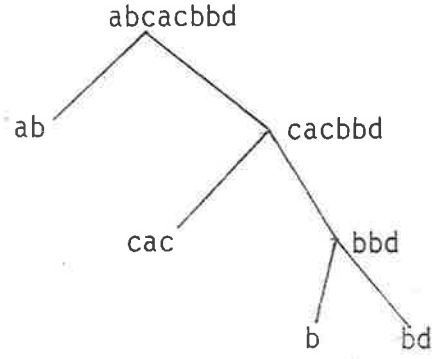
cac bbd

results in two strings in L. The first part again in L_1, and the second part may be split into

b bd

both of which are in L . Hence abcacbbd is in L_1^2

This parse may be represented by a binary tree



The other possibilities result also in parses these are

From the second: (ab̄c), (ac,(b,bd))

From the third: (ab, cac), (b.bd)

and also: (abc, ac), (b,bd)

We now introduce some simple results about expansions.

Theorem 4.2,6

Let L_1 be a sublanguage of L, and L_2 a subset of L_1^2. Then L_2^2 is also a subset of L_1^2.

Proof:

Let S_1 and S_2 be in L_2. Then if $S_1.S_2$ is in L, it is also in L_2^2. However, since S_1 and S_2 are also in L_1^2, if $S_1.S_2$ is in L it is also in L_1^2.

Corollaries

(1) $(L_1^{\,2})^2 = L_1^{\,2}$

(2) if $L_1 \subset L_2$, then $L_1^{\,2} \subset L_2^{\,2}$

(3) if $L_1 \subset L_2 \subset L_1^{\,2}$, then $L_2^{\,2} = L_1^{\,2}$

Definition 4.2,7

A Core Element of L_1 is a string

$$S \in L_1^{\,2}$$

which cannot be written as the catenation of two non-null strings both in $L_1^{\,2}$.

The set of all core elements is denoted C_{L_1}

Theorem 4.2,8

(1) C_{L_1} is a subset of L_1

(2) $(C_{L_1})^2 = L_1^{\,2}$, and every string S in $L_1^{\,2}$ has a binary parse into elements of C_{L_1}

Proof:

(1) Any string $S \in C_{L_1}$ is also in $L_1^{\,2}$. However, since it is not the catenation of two non-null strings in $L_1^{\,2}$, it is only be in $L_1^{\,2}$ because it is also in L_1

(2) By Theorem 4.2,6, $C_{L_1}^{\,2}$ is a subset of $L_1^{\,2}$.

Let S be any string in $L_1^{\,2}$. Then if S is a core element of L_1, it is in $C_{L_1}^{\,2}$. If S is not a core element, it is the catenation of two shorter strings, which are themselves either core elements in which case $S \in C_{L_1}^{\,2}$, or can be further subdivided.

Since the string is of finite length, this process cannot continue indefinitely, and so S can be parsed into core elements, and is in $C_{L1}{}^2$

Theorem 4.2.9

If L_1 and L_2 are sublanguages of L such that

$$L_2{}^2 = L_1{}^2$$

then

$$(L_1 \cap L_2)^2 = L_1{}^2$$

Proof:

From the definition, it follows that

$$C_{L_1} = C_{L2}$$

and $C_{L1} \subset L_1$

$C_{L2} \subset L_2$

Thus

$$C_{L1} = C_{L2} \subset (L_1 \cap L_2)$$

and so

$$C_{L1}{}^2 = L_1{}^2 \subset (L_1 \cap L_2)^2$$

However, since

$$(L_1 \cap L_2) \subset L_1$$

$$(L_1 \cap L_2)^2 \subset L_1{}^2$$

and the result follows.

Theorem 4.2. 10

Let L be a sublanguage of L. Then

$$C_{L1} = \cap L_2$$

where the intersection is taken over all sublanguages L_2 which satisfy

$$L_2^2 = L_1^2$$

Proof:

The result is obvious, since C_{L1} is a subset of any of the L_2, and is also a possible choice for L_2.

4.3 SYNTAX AND SEMANTICS

Definition 4.3.1

A Syntax or Grammarof a language L is a finite set of rules which specify exactly which strings of S are in L.

DEFINITION 4.3.2

A Semantic Model (or the semantics) of a language L is a relation from L to M where M is a set of "meanings" M is the Semantic Range of L

One of the fundamental problems of computational linguistics is that of defining the syntax of the language being dealt with. It is quite simple to define an artificial language which, while intuitively well defined, is not computable, in the sense that it is not always possible to decide whether an arbitrary string is or is not in the language within a finite time. An example is the language A defined by:

String s is in A · if and only if

(1) s is a valid ALGOL program (block)

(2) in execution program s always stops in a finite time, irrespective of the data supplied to it.

(Davis, [7], 1958, p78; Nelson[22] 1968, p130)

Progress so far on the syntax of natural languages does not indicate that they are any better behaved, and indeed there is no reason to expect that they should be. Thus it may not be natural language in a form which is computer programmable, even if agreement could be reached on whether such forms as "colourless green ideas sleep furiously" are sentences or not.

It is this consideration which led to the inclusion of Theorem 4.2.4, which may be stated as:

If L_1 is recursive in L then so is L_1^2

Strings in the language L are termed grammatical (with respect to L) and are called sentences or utterances. They are generally assumed to have the property of meaning, irrespective of whether the individual elements (words) have this property. We have already argued that this meaning is not an intrinsic property, but is an arbitrarily assigned value which can only be measured by reference to a "listener" - some person or machine competent in the language.

An example of the way in which meaning is associated with strings can be found in computer languages: the meaning of a given computer program may be defined as the function which the program evaluates (cf. the semantics of a programming language as defined by Sanderson ([32], 1967)). Two programs are thus equivalent if they cause the computer to evaluate the same function - i.e. to always produce the same output from the same input. It is well known that the same program run on two different computers may produce two

different results (for example, not all FORTRAN compilers compile the DO statement in the same way); this is an illustration of our working definition of meaning. Since the definition of language does not say anything about meaning, such a program has more than one meaning.

However, syntax and semantics are not quite independent: consider, for example, the four B.N.F. grammers for simple arithmetic statements given in Figure 4.3.3. Although each defines exactly the same language (i.e. set of strings) they will generally give a string a different structure; for instance, the string

$$a * b + c * d$$

is given the structure

$$((a * b) + c) * d$$

by the first. This corresponds to the meaning given by some assemble which do not use an operator hierarchy. The second gives the structure

$$(a * b) + (c * d)$$

as is used in simple algebra. The third implies a reversal of the operator hierarchy, giving

$$a * (b + c) * d$$

while the fourth evaluates from right to left in the same manner as the APL language

$$a *(b + (c * d))$$

While each as a syntax for the language is equally valid in each

```
<expression> :: = <item> | < expression> <op> <item>
<op>         :: =   + | *
<item>       :: =   a|b|c|d
```

```
<expression> :: =  <term> | <expression> + <term>
<term>       :: =  <item> | <term> * <item>
<item>       :: =   a|b|c|d
```

```
<expression>:: = <term> | <expression> * <term>
<term>       :: =  <item> | <term> + <item>
<item>       :: =   a|b|c|d
```

```
<expression>:: =<item> | <item> <op> <expression>
<op>         :: =  +| *
<item>       :: =   a|b|c|d
```

Figure 4.3.3 Four grammers for a simple language.

case, it is obvious that the second one is "more natural" than the other three when dealing with algebra. This concept of a "natural" syntax which corresponds to a certain semantic interpretation of a language is useful in practise, but rather difficult to define explicitly.

Just as the structure (syntax) may be defined independently of the meaning (semantics), the converse is also true. The meaning of a string is some value or values arbitrarily associated with it in terms of our working definition of meaning, this association is done by the listener.

4.4 SUMMARY

In this section, we have defined certain terms and notation, for use in subsequent sections, and also discussed some of the background for them. While most of this is aimed at formalising certain intuitive notions, and providing a framework for immediate use, the work on expansions in section 4.2 will not be used until section 7.3; it is however, more convenient to include it in this section with the other notation for character strings.

A GENERAL MODEL FOR LANGUAGE BEHAVIOUR.

5.0 INTRODUCTION

Reasons for basing a model of language on human behaviour have already been discussed; language is not only a very central part of human activity, but also probably the most complex form of behaviour that we exhibit. Thus a model which could satisfactorily explain and reproduce language behaviour could almost certainly reproduce behaviour in other areas such as learning and logical reasoning; conversely, no model of human behaviour would be complete if it did not include language behaviour.

In this section, an abstract model for describing human behaviour is proposed which, while limited in scope, is adequate for the purpose of developing a model of language suitable for solving the answer comparison problem. A model of language is then derived from this.

Such models clearly rely on many assumptions, and have many philosophical implications. These will be dealt with in more detail in the next section.

5.1 A BASIC MODEL FOR HUMAN BEHAVIOUR.

In forming a model for human behaviour, the first step is clearly to form a description of the particular person (or a representative or "average" person) being modelled. There are many forms which such a description might take, but while the choice of the manner of describing persons would be

an early and crucial step in synthesising such a model, we do not need to be prescriptive; we simply postulate that such a description is possible.

The description of any particular person will vary with time, and we shall denote the description of a particular person at time t by P_t. Such a description is one of a set of possible descriptions of persons, and we denote this set $\mathcal{P}$. There will in most cases be no confusion if the words "description of" are omitted, and we say that P_t is a person.

At time instant t, person P_t exists in some environment; we postulate that it is possible to describe this environment, and denote the description E_t. The set of possible (descriptions of) environments is denoted $\mathcal{E}$.

As a result of the mutual interaction of P_t and E_t, both are modified, so that at some later time t + 1, P_t has been changed to P_{t+1} ; for our purposes, we do not need to consider the corresponding change in the environment. It is necessary for digital simulation to assume that if this change is not on a discrete time scale, then it can at least be closely approximated by one. The interaction can thus be symbolised by

$$(P_t, E_t) \rightarrow P_{t+1}$$

and is, in its most general form, a <u>relation</u> from the set $\mathcal{P} \times \mathcal{E}$ to $\mathcal{P}$.

It is clear, however, that if this is to be of practical

use, the outcome must be assumed to be uniquely determined, in which case the relation is a function, and we may write

$$P_{t+1} = F(P_t, E_t)$$

5.2 A BASIC LANGUAGE MODEL

We now consider the case of an environment which essentially contains only linguistic elements. If the environment of a person P_t consists of the single utterance s, we may write

$$P_{t+1} = F(P_t, s)$$

and the "difference" between P_{t+1} and P_t is, by definition, the meaning of s to P_t. In this context, the subscripted t becomes superfluous, and will be omitted.

We assume that the null-statement is meaningless, so that

$$F(P,\Lambda) = P$$

While such a model does provide a framework for the discussion of language models, it does not appear to help in solving the answer comparison problem: the question

"is string s closer to s_1 than to s_2?"

has simply been transformed into a similar question about descriptions of people, namely

"is person F (P,s) closer to $F(P,s_1)$ than to $F(P, s_2)$?"

It is therefore necessary to include in the model some means of comparing persons. The minimum requirement would be simply an oracle which could provide answers to this type of question on demand; however, a slightly more powerful device,

which will not only provide the answer but also give a numerical estimate of by how much, will be more useful. We thus postulate as part of the model a function M which maps pairs of persons into non-negative real numbers, with the following properties:

(1) If P_1 and P_2 are persons, M (P_1, P_2) is a non-negative real number

(2) For all P_1 and P_2

$$M(P_1, P_2) = M(P_2, P_1)$$

(3) $M(P_1, P_2) = 0$ if and only of $P_1 = P_2$.

It would be convenient if this mapping also obeyed the triangle inequality, in which case it would be a metric over P, and even more so if the metric were Euclidean, as this is the form usually sought for the representation of psychological data; thes, however, are not essential requirements.

We can now define two basic functions for the comparison of strings;

Definition 5.2.1

The potency of a string s to a person P is

$$\mu_p(s) = M(F(P_1 s)\ P)$$

Definition 5.2.2

The separation of two strings s_1 and s_2 to a person P is

$$\delta_p(s_1, s_2) = M(F(P_1 s_1), F(P_1 s_2))$$

It is readily shown that

(1) $\delta p\ (s_1, s_2) \geq 0$

(2) $\delta p\ (s_1, s_2) = \delta p\ (s_2, s_1)$

(3) $\delta p\ (s_1, s_2) = 0$ if and only if s_1 and s_2 are synonymous (ie, have the same meaning so that $F(P, s_1) = F(P, s_2)$

(4) $\mu_p(s) = \delta p\ (s, \Lambda)$

With regard to our earlier definition of meaning, potency is a measure of the meaningfulness of the string, and separation a measure of the difference in meaning between two strings, as perceived by the particular person P.

5.3 THE RELATIONSHIP BETWEEN M AND F.

We have set up a general language model consisting of three separate parts; an "effect procedure", denoted by F, which calculates the effect of a string on a person, a comparison mapping, denoted by M, which enables persons to be compared, and the "person", P, who is the basis for the comparison.

It is clear, however, that the point at which F completes its task and M takes over is a matter of convenience; two simple examples will illustrate this.

Example 1:

Suppose that the description of P is a character string of some form, and that the function F simply appends the input string s to this description. In this case the

actual comparison of the strings is made by M, which would obviously be the most complex part of the model.

Example 2:

Suppose that a person is represented by a set of numerical values representing scores on a set of independent factor scales, and that F calculates changes in these values and adjusts them accordingly, due to the effect of the input string s. The obvious choice for M is simply the square root of the sum of the squares of the differences between corresponding factors (this is "obvious" because of the geometrical interpretation of the factor scores as coordinates of a point in space when M is simply the Euclidean distance between points) Clearly, in such a model, F would be very complex, and M relatively simple.

We note that the actual results, in terms of the comparison functions μand δ, could be exactly the same for the two models. This illustrates the arbitrary nature of the point of division between F and M.

The concept of two models which give the same results will be explored more fully in section 7.1

5.4 Summary

In this section, we have proposed a structure for a general

language model; a model of this form is capable of resolving the first part of the answer comparison problem, which is to select from a given reference set the string closest in meaning to a given string. We shall see later that many processes which have been used for answer comparison can be expressed as a model of this type.

Before considering the models in more detail, we shall look more closely at some of the underlying assumptions, several of which have been deliberately glossed over in this section for clarity in the exposition.

ASSUMPTIONS UNDERLYING THE MODEL.

6.0 Introduction

We now have constructed a frame work within which the first part of the answer comparison can be resolved. However, this is based on several assumptions, many of which were not discussed during the development of the model; some of these assumptions will be discussed in more detail in this section.

The assumptions may be divided into two broad categories; some of them are general, in that they apply to any attempt to automate the language process, while others are more specific to our particular model.

6.1 General Assumptions.

The much debated problems of syntax versus semantics, which we discussed earlier (see section 3.1) and the computability of a language grammer (see section 4.3) both fall in this category, but will not be further discussed.

Perhaps less fundamental, but more sweeping in its implications, is the claim that natural language processing of the order that we require is impossible. One exponent of this view is Bar-Hillel ([3], 1964, chapter 12) who claims that high-quality machine translation is not possible because the computer is not capable of resolving some types of semantic ambiguities. Among the examples he gives are

The box was in the pen.

where the meaning of "pen" is "obviously" an enclosure, but could in some peculiar contexts be a writing instrument, and

The pen was in the inkstand.

The inkstand was in the pen.

where the same problem arises twice over, with probably two separate resolutions, unless the work being translated is a fantasy. Bar-Hillel points out that the information on which a correct selection can be based is non-linguistic, and that it is very difficult to explain exactly what additional information would enable a computer to correctly interpret the word "pen" in all cases.

We can distinguish perhaps four distinct cases. The first case is that of a deliberate play on words; in such a case, the result is almost certainly untranslatable, as it relies on a specific ambiguity of the source language which would probably not be present in any target language; in our context, the computer presumably should think

"meaning ambiguous - deliberate- play on words"

and respond

"very punny"

Natural language data processing has a long way to go before it reaches that level.

A second case is when the obvious interpretation is correct; in the second example above, this would be

"The (writing instrument) was in the inkstand.

The inkstand was in the (enclosure)

and despite Bar-Hillel's misgivings, this poses no insuperable problems; a program of the level of sophistication of Winograd's ([40], 1972) could handle this. We note in passing that Winograd's program includes a quite definite picture of the "universe", which Bar-Hillel would claim was non-linguistic, but is, as we shall see the very basis of a language model.

The third case is when the "obvious" interpretation is incorrect, but previous information is adequate to allow a correct interpretation; this is a classical case of context, and is in fact not significantly different from the previous case if "context" is interpreted in a wide sense.

The final case is when the obvious interpretation is wrong, and there is no previous information to indicate this; in such a case, even a human translator would translate incorrectly, and we might even conclude that high-quality (human) translation is not possible. Naturally we must assume that the human will later realise his error (if it can be legitimately called such) and go back to correct it; since computers also can backspace and alter their output files, we can presume that when the problems associated with the first four cases have been solved this one will present no major difficulties.

The problem is thus essentially one of context; Simmons

33 1962) states

Words may have many meanings...
How is a computer to select the
appropriate meaning or action?
The answer tends to define context.

Unfortunately, many exponents of machine translation seem to define context as "that other information needed to resolve ambiguity", and then claim that ambiguity can always be overcome by reference to the context.

An interesting view of the topic may be obtained by replacing the word "listener" by "context" in our working definition of meaning, thus obtaining:

The meaning of an utterance in a particular
context is the effect that the utterance has
on the context.

Language is a very complex system of behaviour, and it is quite reasonable to try to simplify its description by the mathematical appraoch of abstaction; however, abstact descriptions of language omit the one essential ingredient, namely, the human being who is the language user. While this process of abstaction has unquestionably led to great advances in linguistics the time has come when it must be realised that one of the factors that makes humans the only satisfactory language handling devices so far created is that they bring to the task not only their knowledge of vocabulary and grammer, but also the wealth of background information which Bar-Hillel labels "non-linguistic"; it is non-

linguistic in the same sense that the wheel is not an electronic device - but who can imagine a civilisation with complex electronic technology but no wheel?

Language is not an end in itself, but a tool;we use it to describe and interact with a real world and it is only by using our built up knowledge of that world that we can understand and use language. It is this built in knowledge of the world which constitutes the context in which any particular statement is interpreted, and of coursethe statement leaves its mark on the person who interprets it, and thus becomes part of the context in which future statements are dealt with.

Context, then, is a state of mind; it is the sum total of past experience that the listener brings to his understanding of a particular linguistic utterance. Any computer program which is to deal with language at a level of competence comparable to that of a human must have built into it (either by program, or more likely by having the ability to accumulate its experiences) its own picture of the universe in which it exists.

The question that we are now faced by is whether this is feasible - can an adequate model of the world of experience be stored and accessed by a computer quickly enough to give it a reasonable language facility? At this stage, no definite answer can be given, but allowing for the current rate of development of technology, and supposing that much of human mental capacity is not essential to linguistic ability, it seems reasonable to be

more hopeful than Bar-Hillel.

6.2 Specific Assumptions

In developing our model, we assumed a discrete time scale, and pointed out that in a digital computer it is not possible to deal with a truly continuous scale. The fact that humans are not able to distinguish stimuli above quite low frequencies as distinct - for example pictures flashed at over about sixteen frames a second are seen as continuous motion, rather than discrete images - indicates that the discrete model can be expected to be at worst a very good approximation.

The other major set of assumptions relates to the description of a person and environment at some time instant, and the possibility of predicting the outcome of their interaction one time step later. This is essentially a philosophical problem, and is not unrelated to the discussion of the previous subsection. If it is possible to measure every aspect of a person and his environment at an instant, and to compute their interaction, then it must follow that the person had no "free will" during the time span for which the computation was carried out; on the other hand, if people do have free will, then it is not possible to describe completely the state of a person and his environment and the mapping which determines their interaction. There is a parallel here with the uncertainty principle of quantum mechanics; one explanation of this principle is that any attempt to measure the state of a system interferes with the system, and this would certainly apply to an attempt to measure the state of mind of

a person. It thus seems unlikely that an exact description of this for a particular person is or will ever be possible; on the other hand, the description of a "typical" person, and a "typical" interaction with the environment may well be within the bounds of possibility.

6.3 Summary

The fact that linguistic ability is so central and basic a part of human behaviour means that the possibility of stimulating it on a computer raises considerations which are of a quite deep philosophical nature. It is not within our scope to attempt to answer all of these problems; we note that they exist, and point out that, if our task is to have any possibility of success, the assumptions must be at least approximately satisfied.

TYPES OF LANGUAGE MODEL.

7.0 Introduction

The result of comparing two strings by a language model of the type described will depend, in general on all three parts - the effect procedure F, the comparison mapping M, and the person or context P. However, if such a model has any general properties, these would normally be associated with the combination of F and M, which define the interaction of a string S and the person P. Thus in this section, we will be concentrating on the F and M combination and investigating some of the properties that they might produce in a language model.

We note at this stage that defining F and M will specify the form of P - for example, whether it consists of an array of numbers, or a character string, or some other form - but not the actual value. We might compare this to a human, who appear to be born with some language facility (the F and M part of the model) and learn an actual language by interaction with other people. In an analogous manner, it might be possible to start out with quite general values for P, and then condition it by applying to it a suitable body or text. This could be particularly appropriate for answer comparison, where often words change their meaning as the context changes through a course; an example might be the word "deviation", which could be used at one point in its statistical sense - standard deviation - and another in its pyschological sense - deviation from the norm -

within the same course.

With these ideas in mind, we consider some of the properties which might be found.

7.1 Equivalence of Language Models.

We noted in section 5.3 that it is quite possible for two distinctly different language models to give exactly the same result. Such models will be called strictly equivalent.

Definition 7.1.1

Two language models (F_1, M_1, P_1) and (F_2, M_2, P_2) are strictly equivalent over a language L if, for any strings s_1 and s_2 in L

$$M_1(F_1(P_1, s_1), F_1(P_1, s_2)) = M_2(F_2(P_2, s_1), F_2(P_2, s_1))$$

(i.e. $\delta P_1(s_1, s_2) = \delta P_2(s_1, s_2)$)

Strict equivalence between language models is easily seen to be an equivalence relation, and is also easily seen to guarantee that the two models will produce exactly the same string to match a given string in an answer comparison. However, if these are to be the criteria the requirements may be considerably relaxed.

Definition 7.1.2

Two language models are equivalent over a language L if there exists a non-negative monotone increasing real valued function of a non-negative real variable, such that for any two strings s_1 and s_2 in L,

$$M_1(F_1(P_1, s_1), F_1(P_1, s_2)) = f(M_2(F_2(P_2, s_1), F_2(P_2, s_2)))$$

(i.e. $\delta P_1(s_1, s_2) = f(\delta P_2(s_1, s_2))$)

A further relaxation is possible if one model is permitted to give an ambiguous answer when the other does not.

Definition 7.1.3

Two language models are weakly equivalent over a language L if, for any strings s_1, s_2, and s_3 in L,

if $\delta_{p1}(s_1,s_2) > \delta_{p1}(s_1,s_3)$ then $\delta_{p2}(s_1,s_2) \geq \delta_{p_2}(s_1,s_3)$

and

if $\delta_{p_1}(s_1,s_2) < \delta_{p1}(s_1,s_3)$ then $\delta_{p_2}(s_1,s_2) \leq \delta_{p2}(s_1,s_3)$

(where δ_{p_1} and δ_{p_2} refer to the separation functions of the two models respectively).

Weak equivalence is not an equivalence relation, being reflexive and symmetric, but not transitive; in fact we note that

Theorem 7.1.4

Every language model is weakly equivalent to

(i) The trivial model for which

$$\delta_p(s_1,s_2) = 0$$

For all s_1 and s_2

(ii) The strict equality model, for which

$$\delta_p(s_1,s_2) = 0 \text{ if } s_1 = s_2$$
$$= 1 \text{ if } s_1 \neq s_2$$

Proof:

(i) This is obvious, since, taking the trivial model as the first one in the definition, the restrictions do not apply as

$$\delta_{p1}(s_1,s_2) = \delta_{p1}(s_1,s_3)$$

For all s_1, s_2, s_3.

(ii) Again taking the given model as the first one in the definition, if

$$\delta_{p1}(s_1,s_2) > \delta_{p1}(s_1,s_3)$$

then this can only occur if

$$\delta_{p1}(s_1,s_2) = 1$$

and

$$\delta_{p1}(s_1,s_3) = 0$$

i.e if

$$s_1 \neq s_2, \quad \text{and } s_1 = s_3$$

then clearly we have

$$\delta_{p2}(s_1,s_3) = 0$$

while, by definition,

$$\delta_{p2}(s_1,s_2) \geq 0$$

giving

$$\delta_{p2}(s_1,s_2) \geq \delta_{p2}(s_1,s_3)$$

The other contingancy leads to a similar situation, with

$$s_1 = s_2 \quad \text{and} \quad s_1 \neq s_3$$

resulting in a similar conclusion, and the theorem is proved.

It is quite clear that the trivial model would be simple to construct: if we take the description P to be a single number initially zero, and define

$$F(P,s) = P$$

for any s, then with the obvious definition

$$M(P_1,P_2) = |P_1 - P_2|$$

all of the requirements are fulfilled.

The strict equality model is perhaps not so obviously possible within the constraints of the models, but it turns out to be again quite simple to construct : because strings are of finite length and over a finite alphabet each string has a Gödel number which is unique; if we now modify the above (trivial) model by defining

$$F(P,s) = P + gn(s)$$

where gn (s) denotes the Godel number of string s, and define

$$M(P_1,P_2) = 1 \text{ if } P_1 \neq P_2$$
$$= 0 \text{ if } P_1 = P_2$$

then it is easy to see that the requirements are all satisfied.

We thus see that weak equivalence is not a very stringent requirement. Equivalence, however, is quite a strong requirement and in fact two models are equivalent if and only if they always produce the same result for the first part of an answer comparison. The hierarchical nature of the three categories is illustrated by the following theorem

Theorem7.1.5

If two language models are strictly equivalent then they are equivalent; if two language models are equivalent, then they are weakly equivalent.

Proof:

(i) We simply take f(x) = x in the difinition of equivalence.

(ii) If the models are equivalent

$$\delta_{P_1}(s_1,s_2) = f(\delta_{P_2}(s_1,s_2))$$
$$\delta_{P_1}(s_1,s_3) = f(\delta_{P_2}(s_1,s_3))$$

Now if

$$\delta_{P_1}(s_1,s_2) > \delta_{P_1}(s_1,s_3)$$

because f is monotone increasing,

$$\delta_{P_2}(s_1,s_2) > \delta_{P_2}(s_1,s_3)$$

and a similar result follows for the case when

$$\delta_{P_1}(s_1,s_2) < \delta_{P_1}(s_1,s_3)$$

Thus the requirement for weak equivalence is met.

7.2 The Context Independent Model.

We have associated the word context with the person P who is the basis of comparison, and there are several senses in which we might consider a language to be independent of P, the most straightforward being the case in which the result of comparing two strings does not depend on P.

Definition 7.2.1

Two descriptions P_1 and P_2 are strictly equivalent over a language L under (F,M) if, for every pair of strings s_1 and s_2 in L, $\delta_{P_1}(s_1,s_2) = \delta_{P_2}(s_1, s_2)$

Definition 7.2.2

A language model is strictly context independent over a language L if all descriptions P in the set P are strictly equivalent over L.

Another way of wording these definitions is to say that

(i) P_1 and P_2 are strictly equivalent if (F,M,P_2) and (F,M,P_1) are strictly equivalent.

(ii) The model is strictly context independent if, for any P_1 and P_2, in P, (F,M,P_1) and (F,M,P_2) are strictly equivalent.

By analogy with the discussion of the previous subsection, we can define context independent and weakly context independent models:

Definition 7.2.3

A language model is context independent over the language L if, for all P_1 and P_2 in P, (F,M,P_1) and (F,M,P_2) are equivalent.

Definition 7.2.4

A language model is weakly context independent over L if for all P_1 and P_2 in P, (F,M,P_1) and (F,M,P_2) are weakly equivalent.

In these definitions, P must be understood to be the set of all descriptions which are of the form required by F and M. A context independent model will always give the same closest string in an answer comparison, regardless of the initial P chosen, while a weakly context independent model will give the same answer except that some choices of P may give more "closest" strings than others.

The trivial and strict equality models described in the previous subsection are both examples of strictly context independent language models.

It is quite clear, however, that the meaning of natural language statements may vary considerably with context, and so we would not expect context independent models to be adequate for comparing strings in natural languages.

7.3 The Recursive Model

We would expect that in many cases, the result of placing a person in an environment consisting of a string

$$S = S_1 \,.\, S_2$$

would be the same as placing the person in an environment consisting of S_1, and then in an environment consisting of S_2;i.e.

$$F(P,S_1.S_2) = F(F(P,S_1), S_2)$$

The requirement for this to be true might be that S_1 and S_2 are both complete statements which are in some way independent. If such a formula were to hold, the value of F could be found by a recursive process, and so we term such a model a recursive language model.

Definition 7.3.1

Let L_1 be a sublanguage of L. A language model (F,M,P) is semi-recursive over L_1 if for any two strings S_1 and S_2 in L_1^2 such that $S_1.S_2$ is in L,

$$F(P, S_1.S_2) = F(F(P,S_1), S_2)$$

We call such a model semi-recursive because the recursive formula does not apply in all cases. From the results of section 4.2 we can show that

Theorem 7.3.2

If a language model is semi-recursive over any non-empty subset of L_1^2 then it is semi-recursive over L_1

Proof:

This follows immediately from Theorem 4.2.6

Theorem 7.3.3

If a language model is semi-recursive over L_1, then for any string S in L_1^2, it is possible to evaluate F(P,S) by evaluating F only over strings in C_{L1}.

Proof:

The string S has a binary parse into elements of C_{L1} (Theorem 4.2.8)

Suppose that the first step of this is

$$S = S_1.S_2$$

Then applying the recursive formula,

$$F(P,S) = F\ (F\ (P,S_1),\ S_2)$$

If S_1 and S_2 are not in C_{L_1} then they may be further decomposed into

$$S_1 = S_{11}.S_{12} \quad ; \quad S_2 = S_{21}.S_{22}$$

and, once again applying the formula,

$$F(P,S) = F(F(F(F(P,S_{11}),S_{12})S_{21})S_{22})$$

This process can be repeated until, after a finite number of steps, only strings in C_{L_1} remain in the expression.

It can be seen that the final evaluation starts with P, and consecutively applies the terminal strings from the parse, proceeding from left to right.

Definition 7.3.4

A language model is recursive over L if there exists a sublanguage L_1 of L such that

$$L_1^2 = L$$

and the model is semi-recursive over L_1.

It follows from the definition that a recursive model is semi-recursive over any subset of L, and that it is necessary to define F only for strings in C_L, as the definition is then extended automatically to cover all other strings in L.

We note the following property of a (semi-) recursive strictly context independent model:

Theorem 7.3.5

If a language model is both semi-recursive over L_1 and strictly context independent over L_1^2, then for any strings S_1 and S_2 in L_1^2, such that $S_1.S_2$ is also in L_1^2,

$$\delta_p(S_1.S_2, S_1) = \mu_p(S_2)$$

Proof:

$$\delta_p(S_1.S_2, S_1)$$

$$= M(F(P, S_1, S_2), F(P, S_1)) \qquad \text{(definition)}$$

$$= M(F(F(P, S_1), S_2), F(P, S_1)) \qquad \text{(recursive)}$$

$$= M(F(P^1, S_2), P^1)$$

where $P^1 = F(P, S_1)$

$$= M(F(P, S_2), P) \qquad \text{(context independent)}$$

$$= \mu_p(S_2)$$

which is the required result.

7.4 The Metric Model.

We noted earlier that M is almost a metric over P; if the final requirement, the triangle inequality, is met, then a variety of interesting results follow. Before developing these, we define

the term "synonym" and prove a result that is true for any language model of the class we are considering.

Definition 7.4.1

The strings S_1 and S_2 are synonymous in a language model if

$$\delta_p (S_1,S_2) = 0$$

Theorem 7.4.2

The relation of synonymy is an equivalence relation.

Proof:

(i) Clearly $\delta_p (S,S) = 0$ for any S; the relation is reflexive

(ii) Clearly $\delta_p (S_1,S_2) = \delta_p (S_2, S_1)$ so that if $\delta_p(S_1,S_2) = 0$ then $\delta_p(S_2,S_1) = 0$; the relation is symmetric.

(iii) Suppose

$$\delta_p (S_1,S_2) = 0 \text{ and } \delta_p(S_2,S_3) = 0$$

Then by the definition of the mapping M, we must have

$$F (P, S_1) = F(P, S_2)$$

and

$$F (P, S_2) = F (P, S_3)$$

thus giving

$$F (P, S_1) = F (P, S_3)$$

so that

$$\delta_p (S_1,S_3) = 0$$

and the relation is transitive.

An important fact about equivalence relations is that they divide a set up into equivalence classes. We will denote the set of equivalence classes of language L under the relation of synonymy by the symbol $\overline{L}$.

Definition 7.4.3

A language model is metric over the language L if, for any strings S_1, S_2, and S_3 in L,

$$\delta_p (S_1,S_3) \leq \delta_p(S_1,S_2) + \delta_p(S_2,S_3)$$

This is a slightly weaker requirement than the requirement that M satisfy the triangle inequality, but it is adequate for the proof of all of the results we require.

Theorem 7.4.4

In a metric language model, the separation function δ_p is a metric over the set $\overline{L}$ of classes of synonymous strings.

This requires no proof, as the triangle inequality was the only missing property. It is necessary to introduce $\overline{L}$ to satisfy the requirement that a metric be non-zero for distinct elements

Theorem 7.4.5

In a metric language model,

$$\delta_p (S_1, S_2) \leq \mu_p (S_1) + \mu_p (S_2)$$

Proof:

$$\delta_p (S_1, S_2) \leq \delta_p(S_1,\Lambda) + \delta_p (\Lambda, S_2)$$

$$\mu_p (S_1) + \mu_p (S_1)$$

Theorem 7.4.6

In a metric language model, if S_1 and S_2 are synonymous,

$$\mu_p (S_1) = \mu_p (S_2)$$

and

$$\delta_p (S , S_1) = \delta_p (S, S_2)$$

for any S.

Proof:

$$\text{(i)} \quad \mu_p (S_1) = \delta_p (S_1, \Lambda)$$
$$\leq \delta_p (S_1, S_2) + \delta_p (S_2, \Lambda)$$
$$\leq 0 + \mu_p (S_2)$$

Similarly

$$\mu_p (S_2) \leq \mu_p (S_1)$$

and so equality must hold

$$\text{(ii)} \quad \delta_p (S,S_1) \leq \delta_p(S,S_2) + \delta_p (S_2, S_1)$$
$$\leq \delta_p(S,S_2) + 0$$

Similarly

$$\delta_p (S, S_2) \leq \delta_p (S,S_1)$$

and equlaity must hold.

Since the inequality

$$\delta_p (S_1, S_2) \leq \mu_p (S_1) + \mu_p(S_2)$$

holds in a metric language model, it is possible to give a sensible definition of antonymy: the two strings must have equal meaningfulness and be separated by the maximum possible amount.

Definition 7.4.7

Two strings S_1 and S_2 are antonyms in a metric language model if

$$\delta_p(S_1,S_2) = 2\mu_p(S_1) = 2\mu_p(S_2)$$

An interesting point is that the null string is both a synonym and antonym for itself.

The following theorem relates to a model which falls in all three of the categories we have considered.

Theorem 7.4.8

If a language model is semi-recursive over L_1 and strictly context independent and metric over L_1^2, then for any strings S_1 and S_2 in L_1^2 such that $S_1.S_2$ is also in L_1^2,

$$\mu_p(S_1.S_2) \leq \mu_p(S_1) + \mu_p(S_2)$$

and

$$\delta_p(S_1.S_2,S) \leq \delta_p(S_1,S) + \mu_p(S_2)$$

Proof:

We prove only the second part, since the first is obtained by putting $S = \Lambda$

$$\begin{aligned}\delta_p(S_1.S_2,S) &\leq \delta_p(S_1.S_2, S_1) + \delta_p(S_1,S) \\ &\leq \mu_p(S_2) + \delta_p(S_1,S)\end{aligned}$$

(using theorem 7.3.5) which is the required result.

7.5 THE ADDITIVE MODEL.

Definition 7.5.1

A language model is additive if the person P is represented by an array of numbers only, and the mapping M depends only on the difference between corresponding elements of the two P-arrays.

It will in fact be true that all of the specific models that we consider will be both metric and additive.

Definition 7.5.2

In an additive language model, the effect of a string S is

$$E_p(S) = F(P,S) - P$$

(where the subtraction sign implies finding the difference between corresponding elements in the two arrays F(P,S) and P)

Then we can see that

$$F(P,S) = P + E_p(S)$$

which is the reason for the term "additive". In an additive language model,

$$\delta_p(S_1,S_2) = M(E_p(S_1), E_p(S_2))$$
$$= M(E_p(S_1) - E_p(S_2),0)$$
$$\mu_p(S) = M(E_p(S), 0)$$

and the mapping M can be replaced by an appropriate mapping which measures one person only; this would usually be a norm of some form, so that we may write

$$\mu_p(S) = ||E_p(S)||$$
$$\delta_p(S_1,S_2) = ||E_p(S_1) - E_p(S_2)||$$

When an additive language model is context independent, we have $||E_p(S)||$ is independent of P; this will often be because $E_p(S)$ is independent of P, and if this is the case, we will call the model functually context independent.

Definition 7.5.3

An additive language model is functionally context independent over the language L, if for any P_1 and P_2 in P, for all S in L

$$E_{p_1}(S) = E_{p_2}(S)$$

Theorem 7.5.4

An additive, functionally context independent language model which is semi-recursive over L_1 is order free over L_1, where, by order free, we mean that, for any S_1 and S_2 in L_1^2 such that $S_1.S_2$ and $S_2.S_1$ are also in L_1^2

$$E_p\ (S_1.S_2) = E_p\ (S_2.S_1)$$

Proof:

$$\begin{aligned} E_p\ (S_1.S_2) &= F\ (P, S_1.S_2) - P \\ &= F\ (F\ (P, S_1), S_2) - P \\ &= F\ (F\ (P, S_1), S_2) - F(P, S_1) \\ &\quad + F\ (P,S_1) - P \\ &= E_{F(P,S_1)}(S_2) + E_p(S_1) \\ &= E_p\ (S_2) + E_p\ (S_1) \end{aligned}$$

Similarly

$$\begin{aligned} E_p\ (S_2.S_1) &= E_p\ (S_1) + E_p\ (S_2) \\ &= E_p\ (S_2) + E_p\ (S_1) \\ &= E_p\ (S_1.S_2) \end{aligned}$$

Corrolaries:

(i) $\delta_p\ (S_1.S_2, S_2.S_1) = 0$

(ii) For an appropriate string

$S = S_1.S_2.S_3 \ldots S_n$

any string S^1 obtained by permuting the substrings of S is synonymous to S

7.6 Summary

We have discussed several types of language model which fall

into the general class that was defined in section 5.2; of these by far the most important are the additive metric models, and we now look at some of these as they relate to previous work in related areas.

EXAMPLES AND DISCUSSION

8.0 Introduction

We have developed a structure for a language model, and discussed various types of model that can arise within this structure; the structure that we used was based on the observation that humans are so far the only competent handlers of natural languages.

In this section, we shall first of all show that the structure can handle languages other than natural languages, by considering work that has been done on the comparison of mathematical expressions. We shall then discuss the work of Osgood, Succi and Tannenbaum on the empirical measurement of meaning.

8.1 Comparison of Mathematical Expressions.

The case where the language under consideration consists of well formed algebraic expressions is a particularly interesting and useful one, both on account of its applications in C.A.I., and because, unlike the corresponding natural language problem, precise definition is possible.

Let us consider initially expressions which involve only one variable and which are defined, continuous and integrable over an interval [a,b] ; such expressions can be considered as functions of one argument.

Two such functions are equivalent over the interval [a,b] if their values are the same for all possible assignments of the variable within that interval. Because of the restrictions that

have been placed on the functions, an equivalent definition is that the functions f and g are equivalent if

$$\int_a^b (f(x) - g(x))^2 \, dx = 0$$

(it can be shown that this integral exists for the functions considered)

Furthermore, this integral is a measure of the least squares goodness of fit of f as an approximation to g (or vice-versa) and hence is a good distance measure for the language under consideration.

The numerical approach to testing for expression equivalence has been found to be quite fruitful; Lee ([16] , 1971, p65ff) has shown that it is possible to test for equivalence of trigonometric formulae, and also to determine the set of identities required to transform one formula into another equivalent one, and cites the work of Oldehoeft, who showed that, for a wide class of functions, comparing the values at one random point is enough to show equivalence with zero probability of error ([24] , 1970)

It thus seems reasonable to approximate the above integral numerically, and we shall show that it is possible to do this with a language model of the type we have been considering, and that this leads to the methods used in the works discussed in the previous paragraph.

We take for the description P a vector of n elements.

$$P = (P_1, P_2, \ldots, P_n)$$

all initially set to zero (or in fact any other arbitrary value, as the model we shall construct is context independent)

The effect procedure consists of evaluating the input expression at n arbitrarily chosen points, $x_1, x_2, \ldots x_n$, and adding each result to the corresponding element of P. The mapping m is based on the standard Euclidean distance; we sum the squares of the differences of corresponding elements, multiply this total by (b-a), and divide by n.

The result of comparing f with g is clearly

$$\frac{b-a}{n} \sum_{i=1}^{n} (f(x_i) - g(x_i))^2$$

which is an approximation to

$$\int_a^b (f(x) - g(x))^2 \, dx$$

although it is probably not the numerically best approximation. Thus the language model that we have set up provides for the expressions of the restricted class under consideration, a comparison function which satisfies

(i) $\delta(f,g) \geq 0$

(ii) $\delta(f,g) = 0$ if f is equivalent ot g.

(iii) The value of $\delta(f,g)$ is a least squares measure of the difference between f and g.

The case when only one point of evaluation is used gives the methods of Oldehoeft and Lee which were cited above, and we see that the probability of $\delta(f,g) = 0$ when f and g are not equivalent is, even for this case, zero (it must be appreciated that, given any set $x_1, x_2, \ldots x_n$ of points of evaluation, there are infinitely many non-equivalent functions which coincide at all of these points

and thus give a δ of zero; however, if these points are fixed at random, the probability of two randomly chosen functions coinciding on them and yet being non-equivalent is zero).

The generalisation to expressions involving more than one variable and with possible points within the interval at which they are undefined, is now obvious. Care must, however, be taken, as the method cannot be generalised to functions which have finite discontinuities; as a particular example, take

$$f(x) = 0 \quad \text{if } x = 0$$
$$= 1 \quad \text{if } x \neq 0$$
$$g(x) = 0 \quad \text{if } x = 1$$
$$= 1 \quad \text{if } x \neq 1$$

where we see that f and g are not equivalent on the interval $[-2,2]$, although

$$\int_{-2}^{2} (f(x) - g(x))^2 \, dx = 0$$

and the probability that a randomly chosen set of points in the interval will show the difference between f and g is zero

The method use by Lee to find the set of identities (the consistency set) needed to transform one expression into another, by treating the trigonometric functions as further variables, also falls easily within the scope of this generalisation, and his concept of step size could be readily related to the second part of our statement of the comparison problem.

We thus see that a simple language model of the type we are considering can be used to compare mathematical expressions, and

that this comparison is closely related to methods used by others working in this field.

8.2 The Work of Osgood, Succi and Tannenbaum.

The work of Osgood et. al ([26], 1957) marked the first steps in a new direction for the psycholinguists. These workers showed that the emotive meaning of words could be specified by three scores on three independent scales; an evaluative (good - bad) scale, a potency (strong - weak) scale, and an activity (active - passive) scale. This was done by collecting a large amount of data from people, and using the techniques of factor analysis to process their results. From their results, they devised a test for evaluating common words on these scales; the results can then be illustrated by making a three-dimensional model, using the three scales as axes, and, among other things they suggested that such models could have applications in the diagnosis of mental illness. They also showed that, (as we might expect) this model does not vary a great deal from person to person for normal people, and that repetition of phrases (an example they used was "shy secretary") causes changes in the position of the words which can be at least approximately predicted from their initial positions.

The relevance of this work to our language model is apparent. the person P is represented by a set of points, representing the words in the three dimensional space, termed the semantic space, by Osgood et. al; the mapping M is simply the root mean square

distance between corresponding points, a measure very similar to that used in the previous subsection; the major problem is as we might expect the definition of F, for which Osgood et.al. give only a very rough formula applicable to strings of two words.

One of the major problems in experimenting with the Osgood model in order to find more information on the F function is the difficulty of accurately evaluating small changes in P and also the fact that because the measurement is carried out by a linguistic tool, the measurement tends to drown out the effect being sought; thus they had to repeat any phrase many times in order to achieve a measurable effect. However, their results are still encouraging evidence in support of the structure of the present models.

It must be remembered when we are considering the work of Osgood et.al. that the type of meaning being measured is emotive meaning and not the semantic meaning with which we are primarily concerned, although at no stage do they make this clear. Thus their results are probably not directly applicable to the problem in hand.

8.3 Summary

In this section, we have discussed previous work in two quite distinct areas and we have seen that in both cases the concept of a language model, as we have constructed it, fits in quite naturally with the work that has been done, and offers a convenient basis for its discussion.

LANGUAGE INTERPRETERS.

9.0 Introduction

The developement of the language models up to this point has been completely abstract: we have not given any indication of the form that the description of a person might take or of the way in which such descriptions might be obtained. We now consider in more detail the two portions, P and F, of such a model, and derive from the general class of lan guage models a more restricted class which, for want of a better name, we shall call the class of language interpreters.

Language interpreters are additive and metric, giving in fact a representation of strings in a Euclidean space.

9.1 The Context Network.

As was stated in the introduction to section 7, humans appear to have an inbuilt language facility which is readily adapt to handle languages from Japanese to German. This facility we might identify with the F portion of a language model; this implies that F should be at least to a large extent, a general purpose procedure, which further implies that the meanings of specific words should be stored, not as part of the F section of the model, but in the description P; in practice, function words such as prepositions and conjunctions, which are principally concerned with indicating the relationship between the other words which carry the bulk of the semantic content, might well be exceptions to this general rule.

We are thus faced with the problem of selecting a form for

P which will implicitly contain all of the semantic data for the words of the language. In order to see how this might be done, consider the problem faced by a human who encounters an unknown word for the first time; the meaning of the word may be clear from the context in which it is used, or the person may refer to a dictionary, where its meaning will be related to that of other words. The person does not, of course, update his whole language processing facilities, but simply fits the new word into its proper place in relation to all other words in his memory.

Thus we shall assume that the meaning of a word is implicit in its relationships with all other words; the non-absolute nature of this is comparable to the non-absolute nature of dictionary definitions, and may indeed reflect a similar feature in human language behaviour. We shall restrict ourselves to the case where the relationship between two words can be expressed as a single real number; such a relation need not, of course, be symmetric.

The structure of P will thus be represented as a weighted network, the context network, with the nodes being associated with elements of the alphabet, and each arc, which may be directed or undirected, having a weighting which represents the relationship between the two nodes which it joins. Since we would expect that in general nearly all possible arcs would be significant, we may take this network to be complete, and this means that in practice it would be stored as an array, either square for the non-symmetric case, or triangular for the symmetric case.

9.2 The Comparison Mapping

The root mean square difference between two networks provides a convenient means of comparing them. This is calculated by summing the squares of the differences between corresponding weightings, dividing by the total number of weightings, which is n(n-1) for the non-symmetric case, and n(n-1)/2 for the symmetric case, where n is the number of symbols in the alphabet, and then taking the square root. This procedure has the advantage of producing a representation which is Euclidean, as we prove in the following theorem.

Theorem 9.2.1

A language interpreter is equivalent to a mapping of strings into a subspace of R^k where k is the number of arcs in the context network, with the separation of two strings being the usual Euclidean distance between them.

Proof:

Let $E_p(S)$ be the effect of string s; then by taking the k elements of $E_p(S)$ in some well defined order, a k-vector is obtained. Each element of this k-vector is divided by the square root of k, and the result is taken as the coordinates of the image of the string S in a k-space. Thenull string of course is mapped onto the origin.

It is clear from the definition of M that it corresponds to the Euclidean distance between the image points of two strings.

Since there are only a countable number of strings in any language, it is clear that for any given language interpreter,

not every point in the k-space is the image of a string, and in fact the result may be representable in a space of lower dimension.

The converse of this theorem is also true:

Theorem 9.2.2

Any language model which represents strings as points in a Euclidean space of finite dimension is equivalent to a language interpreter.

Proof:

We simply reverse the above procedure, and use the coordinates of the points as a means of defining the effect of any string.

The requirement is clearly that if k is the dimension of the space, we must have

$$n \geq (1 + \sqrt{4k + 1})/2$$

for the non-symmetric model or

$$n \geq (1 + \sqrt{8k + 1})/2$$

for the symmetric case, where n is the size of the alphabet.

We note, for example, that the semantic space of Osgood et.al. can be modelled by a language interpreter; the requirement is that the network have at least three nodes. However, this theorem, although useful from a theoretical point of view, does not provide much practical assistance as unless the original model provides a practical means of calculating the coordinates of any given string, it would be almost impossible to actually define an effect procedure F which would result in the required model.

9.3 The Effect Procedure

The chief purpose of the effect procedure is to enable calculation of the effect $E_p(S)$ of any string S. Naturally, we would expect that, for a language interpreter intended to deal with the semantic meaning of strings in a natural language - and we must remember that, as the example in section 8.1 illustrates, our definitions do not necessarily imply any restriction to such languages - it would be extremely difficult to derive such a procedure, and we shall not attempt to do so here.

We have already seen that language interpreters are a subclass of the additive metric language models. Particular examples may be further classified according to whether they are or are not semi-recursive, and are or are not context independent, and all of the work of section 7 carries over to them.

9.4 Summary

The subclass of language models that we have called language interpreters have several desirable properties: their form is convenient to implement, and permits a simple comparison mapping which is inherently Euclidean and the class in principle includes all language models which give such representations.

In the next section, we show how language interpreters can offer a practical means of comparing character strings.

EXAMPLES OF LANGUAGE INTERPRETERS.

10.0 Introduction

So far we have been mainly concerned with the theoretical aspects of answer comparison; it is now necessary to show that the language models can produce acceptable results.

Rather than attempt to set up a full-scale model for the entire English language, we shall concentrate on a single example. The reference set is given in Figure 10.0.1 and consists of the same six statements used in the second example of section 2.3. The three additional strings together with those in the reference set, will be compared in turn with each member of the reference set; the exact match that must occur when a string is compared with itself will be ignored.

It is necessary to have a standard of comparison. One possibility would be to ask a sample of people to rank each string in the reference set with respect to each other string; however, the selection of appropriate distances from the resulting data poses some problems. Instead, we shall use a comparison technique proposed by Morrison [21] and described by Algebra in his paper on "String Similarity and Misspellings" ([1], 1967). This is a quite sophisticated method of comparison, although it does not rely on any specific properties of the English language (such as the relatively small information content of vowels). It also is convenient as it leads directly to a measure of similarity which can be readily converted to a distance measure comparable with the separation measure produced by a language interpreter.

1 An enzyme is a catalyst in a biological system

2 An enzyme is a catalyst in a biochemical reaction

3 An enzyme acts as a catalyst

4 An enzyme speeds up a biochemical reaction

5 Enzymes are formed in biochemical reactions

6 Many reactions would not take place without enzymes

Reference Set

7 A catalyst acts as an enzyme

8 Enzymes catalyse biochemical reactions

9* An enzime is a catallist in a biological sttem

Other Strings

Figure 10.0.1 STRINGS USED TO COMPARE LANGUAGE MODELS

The results of this method will be compared with three language interpreters; the first two will be based on simple character by character comparison, and the third one will use words as the basic unit.

10.1 The Standard Method.

The method proposed by Morrison is perhaps best illustrated by an example:

let S_1 = "An-enzyme-acts-as-a-catalyst"

S_2 = "An-enzyme-speeds-up-a-biochemical reaction"

where we have

$l(S_1) = 28$; $l(S_2) = 42$

and S_1 and S_2 have a common selection

S_{12} = "An-enzyme-s--a-alat"

$l(S_{12}) = 19$

By a common selection we mean that the elements of S_{12} may be found in the same order in both S_1 and S_2 . It may be verified that, although S_1 and S_2 have several common selections of length 19, there are none of length more than 19; thus S_{12} is a longest common selection.

Morrison defined the degree of match between two strings as the ratio of the length of the longest common selection to the length of the longer of the two strings; in the example given,

$$\text{degree of match} = {}^{19}/_{42} \simeq 0.452$$

Clearly the degree of match is a rational number between 0 and 1, being 1 only for identical strings. It is more convenient for our purposes to convert this to a distance measure by subtracting it from 1. Thus we would have in the above case

String	Λ	1	2	3	4	5	6
1	1	0					
2	1	.2245	0				
3	1	.4783	.5511	0			
4	1	.4566	.2654	.5477	0		
5	1	.5000	.3062	.6512	.2559	0	
6	1	.6275	.6275	.6863	.6667	.6275	0
7	1	.6087	.6327	.4643	.7143	.7442	.6471
8	1	.4566	.2654	.5527	.2858	.2559	.6667
9	1	.0870	.2654	.5218	.4783	.5218	.5883

Distances Between Strings

1	1 2 4 3 5 6 Λ
2	2 1 4 5 3 6 Λ
3	3 1 4 2 5 6 Λ
4	4 5 2 1 3 6 Λ
5	5 4 2 1 6 3 Λ
6	6(1 2 5)4 3 Λ
7	3 1 2 6 4 5 Λ
8	5 2 4 1 3 6 Λ
9	1 2 4(3 5)6 Λ
Λ	all same dist.

Reference Strings in Order of Closeness

Figure 10.1.1 RESULTS FOR LONGEST SELECTION METHOD.

$$d\ (S_1,S_2) = 1 - \frac{19}{42} = \frac{23}{42} \simeq 0.548$$

This distance does have quite convenient properties, as we shall now pro

Theorem 10.1.1

The distance between two strings, as defined above, is a metric over the set of all strings.

Proof:

(i) Clearly $d(S_1,S_2) \geq 0$ being zero only for identical strings we note that $d(\Lambda,\Lambda)$ is undefined, but it is consistent to define this to be zero.

(2) The distance is clearly symmetric, as the longest common selection does not depend on the order in which the two strings are given.

$$d(S_1,S_2) = d(S_2,S_1)$$

(3) The proof of the triangle inequality,

$$d\ (S_1,S_3) \leq d(S_1,S_2) + d\ (S_2,S_3)$$

is more difficult and is given in full in Appendix A.

Thus the distance does satisfy the requirements of a metric over the set of strings.

The general procedure for evaluating this distance is quite slow; it is in essence necessary to try to match all possible common selections, and this is a highly recursive process. Because of its nature, the process was coded in assembly language, and, running on a small mini-computer, averaged approximately twenty minutes per comparison. The process is fastest when the two strings are identical and rapidly deteriorates as the strings become more and more dissimilar.

Details of the program are given in Appendix B.

Because of its slow speed, this method would not as it stands be suitable for C.A.I. work, and Morrison suggests several alternative methods for approximating the results with less calculation. It is possible that the method could be used as it stands if a "threshold look ahead" were incorporated in the algorithm; since the match would in any case be rejected if the degree of match fell below some fixed value (e.g. 0.6), it would be possible at each step to look ahead to see if this value could possibly be achieved and thus avoid following up any fruitlessbranches; this would in fact improve the speed remarkably as it is precisely the unfruitful avenues which require so much calculation time.

The table of figure 10.1.1 gives the results of the comparisons for the test data using the longest common selection method. This shows both the distances between the pairs of strings, and the resulting ranking of the reference strings in order of closeness to each one. Note that as could easily have been predicted the distance of any string from the null string is unity, and that

$$d\ (S_1,S_2) \leq d\ (S_1,\Lambda)$$

in all cases.

An interesting method of representing this data graphically is to mark the strings on a plane with the distance between the points in proportion to the distance between the strings; such a representation can only be an approximation, as exact representation in two dimensions is not in general possible, and further, as the plotting procedure

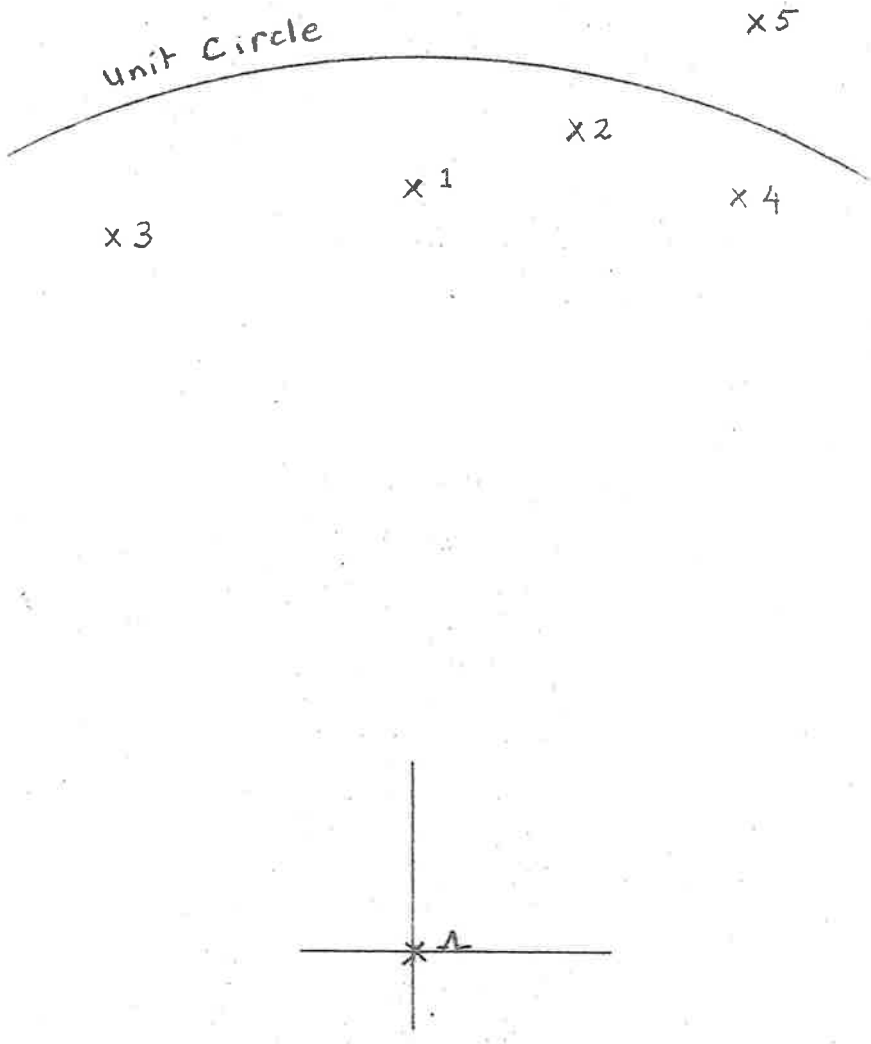


Figure 10.1.2 REPRESENTATION OF THE REFERENCE SET IN TWO DIMENSIONS - Comparison Method.

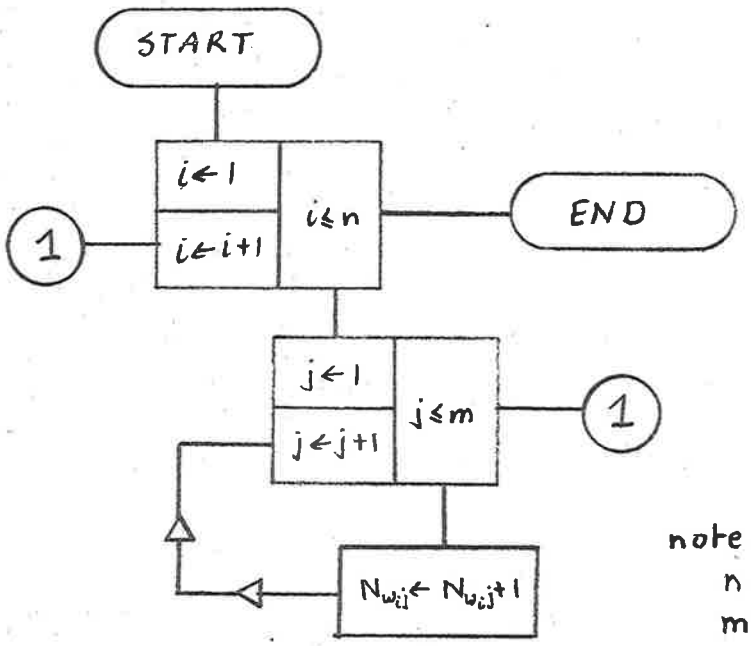
involves a minimisation (of a function of twelve variables), there is no general method of ensuring a best representation as opposed to a local minimum. This relationship is shown for the reference strings in Figure 10.1.2; the error in the representation is approximately three percent.

10.2 A Fully Recursive, Context Independent Language Interpreter.

This is the simplest possible form of language interpreter. Suppose that the input string is

$$S = W_1 W_2 \ldots W_n$$

(where the elements are chosen from a set consisting of the alphabet and the space character.) then the effect procedure takes the following form



From this it is clear that the result will depend only on the elements present in the string, and not on their order; this is well illustrated by the comparison of the results for the third and seventh strings, which are anagrams of each other. In fact this model would treat the string

"aaaaaa cceel mnn sss ttt yyz"

as a synonym for either of them, whereas the reference method treats them as being quite distinct.

The simplicity of this interpreter makes it considerably faster than the common selection method; the program was written in the BASIC language and run interpretively on a mini-computer, taking less than one minute per comparison. Unlike the common selection method the time for a comparison is proportional to the string length, and independent of the actual maount of similarity between the strings. The time is, however, proportional to the square of the network size.

A comparison of the thirtyfive distinct distances given by the separations of the eight strings from the six reference strings and the null string, disregarding repetitions or the distance of a string from itself, yields a correlation coefficient of 0.845. This is infact the highest correlation of the three interpreters being considered.

The actual separations obtained are given in Figure 10.2.1, and a graph of the relationship, scaled to be approximatley the same size as the corresponding plot for the reference method, is given in Figure 10.2.2

String	Λ	1	2	3	4	5	6
1	4.439	0					
2	4.737	1.112	0				
3	3.035	1.942	2.306	0			
4	3.873	1.783	1.622	2.064	0		
5	3.931	2.056	1.838	2.446	1.035	0	
6	4.537	1.770	1.773	2.419	1.650	1.898	0
7	3.035	1.942	2.306	0	2.064	2.446	2.419
8	3.477	1.860	1.884	1.770	1.270	1.252	2.028
9	4.581	1.068	1.233	2.401	2.009	2.190	1.997

Distances Between Strings

1	1 2 6 4 3 5 Λ
2	2 1 4 6 5 3 Λ
3	3 1 4 2 6 5 Λ
4	4 5 2 6 1 3 Λ
5	5 4 2 6 2 3 Λ
6	6 4 1 2 5 3 Λ
7	3 1 4 2 6 5 Λ
8	5 4 3 1 2 6 Λ
9	1 2 6 4 5 3 Λ
Λ	Λ 3 4 5 1 6 2

Reference Strings in Order of Closeness

Figure 10.2.1 RESULTS FOR LANGUAGE INTERPRETER #1.

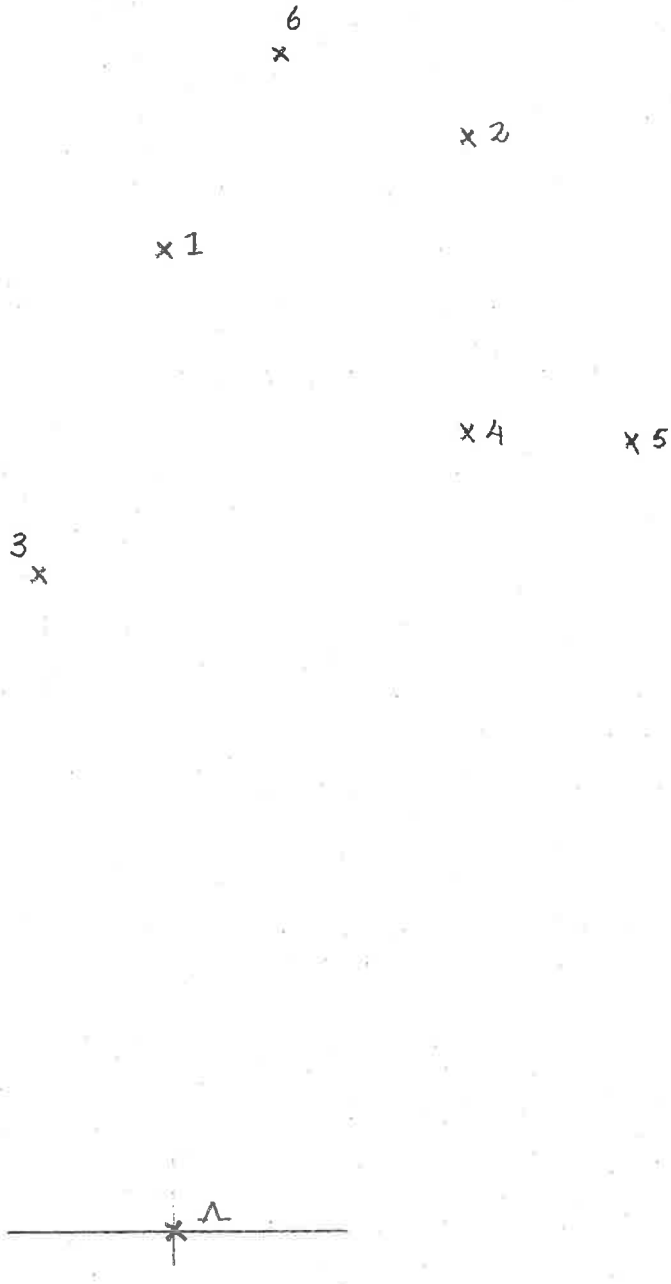


Figure 10.2.2 REPRESENTATION OF THE REFERENCE SET IN TWO DIMENSIONS - Model #1

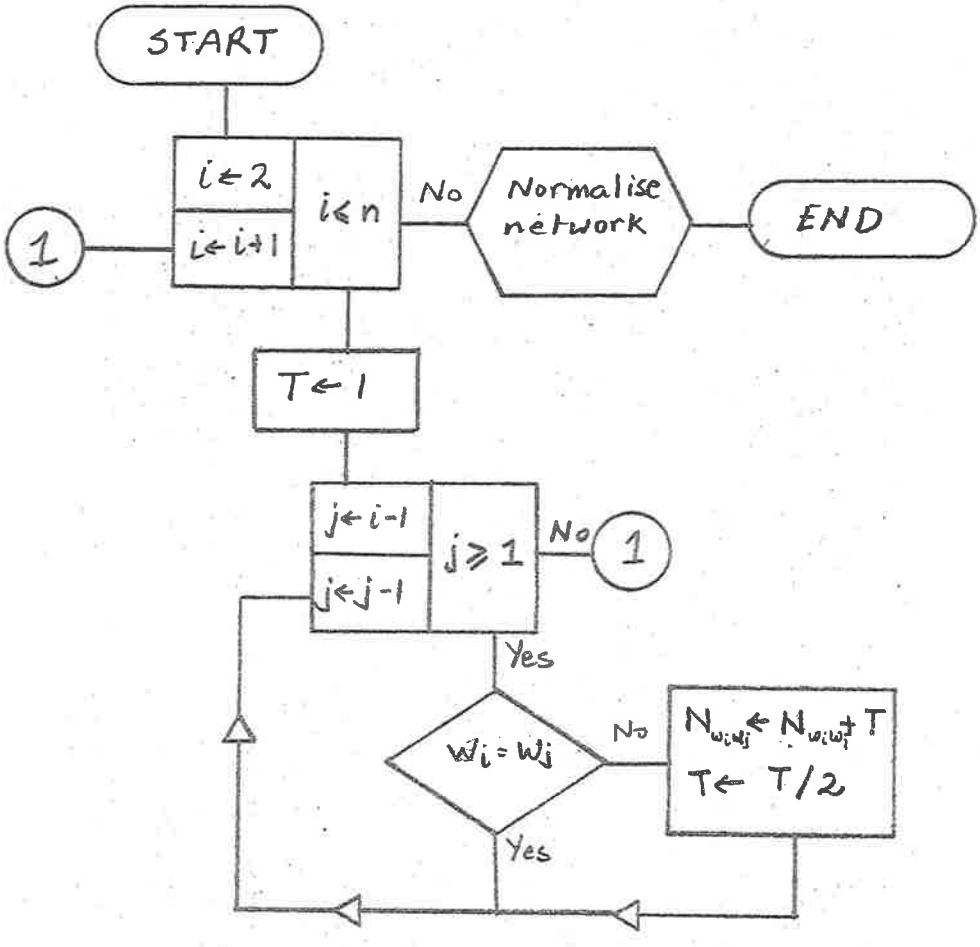
10.3 A More Complex Interpreter.

The second language interpreter has been specifically designed to have the property

$$\delta_N (S,\Lambda) = 1 \qquad S \neq \Lambda$$

in common with the comparison method.

The effect procedure is based on a "normalising" process so that the sum of squares of all the elements should be unity; the network is initially set up with all zero elements. The procedure is:



Although the comparison time for this method clearly increases as the square of the string length, the actual time taken for the comparisons made was not very different from that of the previous method; this is essentially because dependence on the size of the network has been eliminated.

The correlation with the reference method is not very high, being only 0.692; this is partly attributable to the fact that in this model, some separations are greater than the separation from the null string, and in fact the correlation does rise if only non-null strings are considered.

The results are given in Figures 10.3.1 and 10.3.2.

10.4 A Language Interpreter Based on Words.

Whereas the previous two language interpeters used the letters of the alphabet and the space character, we shall now consider one based on words.

The words in the dictionary are given in Figure 10.4.1. The weightings in the context network were chosen as estimated differences in meaning between the pairs; this is, as in both earlier examples, a symmetric relationship. The chief features of the weightings chosen are that "catalyse" and "catalyst" are considered to be synonyms (in a more extensive version they would be recognised as instances of the same root word and only one dictionary entry made), "biological" and "biochemical" are close to being synonyms, and the words "in","up", and "as" are almost meaningless. It will have been noted that the words "a", "an", "are", "is" and "would" do not appear in the dictionary;

Strings	Λ	1	2	3	4	5	6
1	1	0					
2	1	.522	0				
3	1	.711	.711	0			
4	1	.884	.668	.977	0		
5	1	.949	.693	1.063	.668	0	
6	1	1.068	.966	1.049	.965	.966	0
7	1	.722	.741	.245	1.019	1.112	1.048
8	1	.834	.597	.897	.703	.639	1.007
9	1	.431	.607	.836	.933	.977	1.099

Distances Between Strings

1	1 2 3 4 5 Λ 6
2	2 1 4 5 3 6 Λ
3	3(1 2)4 Λ 6 5
4	4(2 5)6 3 1 Λ
5	5 4 2 1 6 Λ 3
6	6 4(5 2)Λ 3 1
7	3 1 2 Λ 4 6 5
8	2 5 4 1 3 Λ 6
9	1 2 3 4 5 Λ 6
Λ	all same dist.

Reference Strings in Order of Closeness

Figure 10.3.1 RESULTS FOR LANGUAGE INTERPRETER #2.

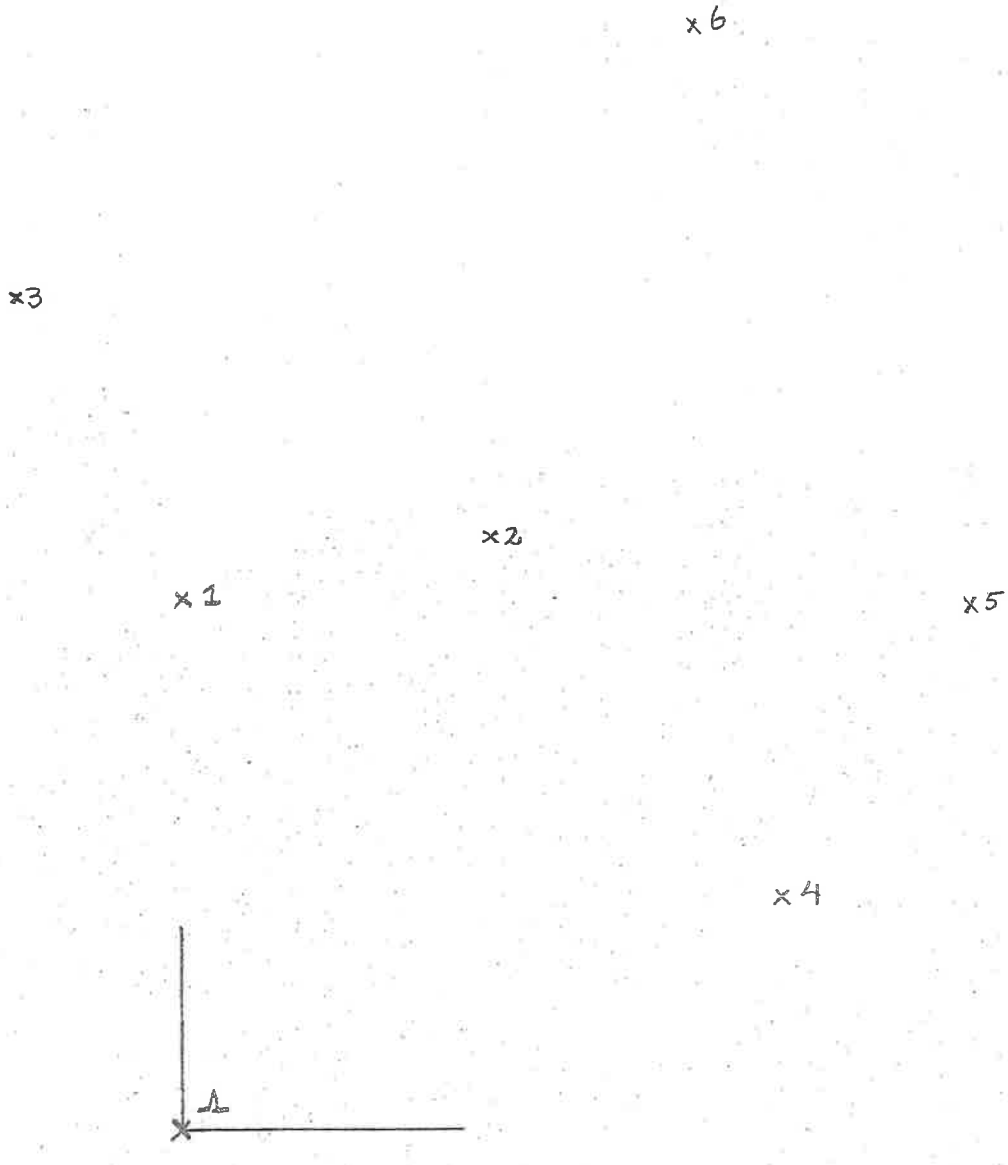
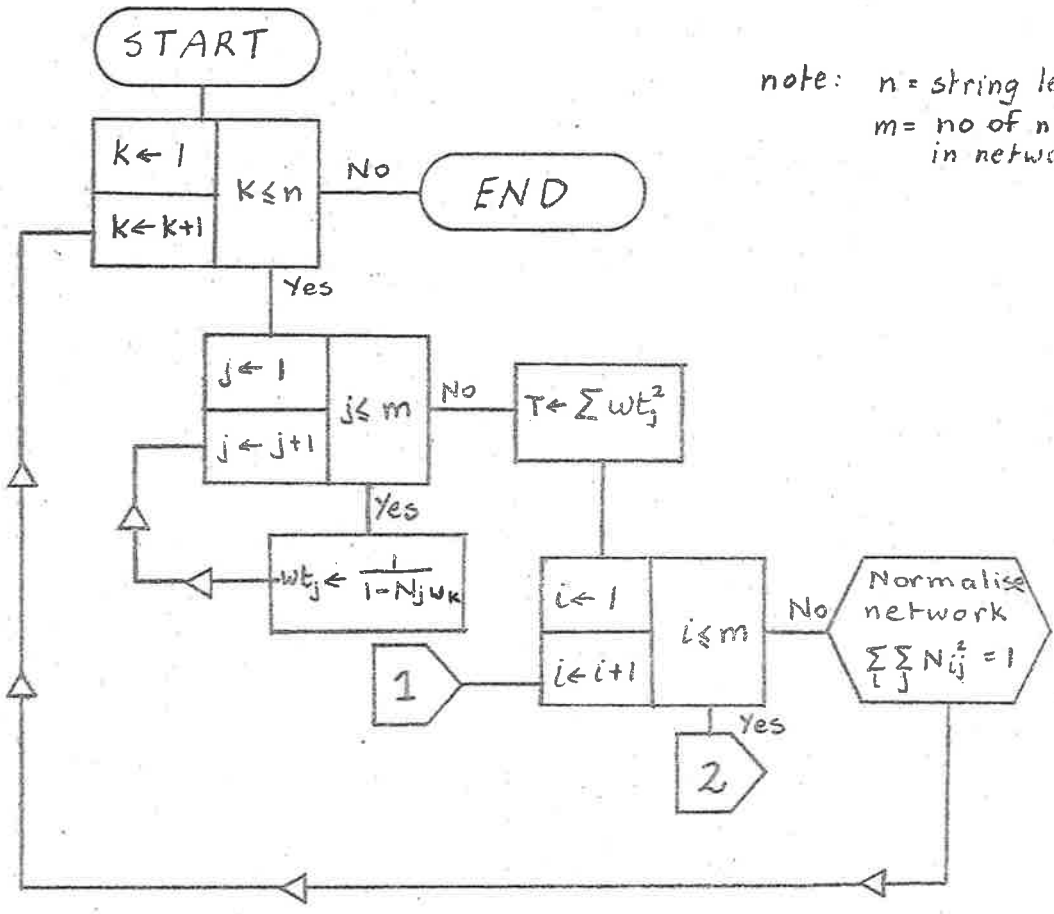


Figure 10.3.2 REPRESENTATION OF THE REFERENCE SET IN TWO DIMENSIONS - Model #2

these words have very high frequency and very low meaning content, and were eliminated from the input strings.

Naturally, the ninth string in the sample, because it involves only misspellings, does not fit into the framework of this model; in a practical situation, the spelling errors would have to be resolved at the time of looking up the words in the dictionary, and would not enter into the main part of the model. For this reason, the last string has not been used in calculation of the correlations for either of the two previous models.

The effect procedure takes the following form:

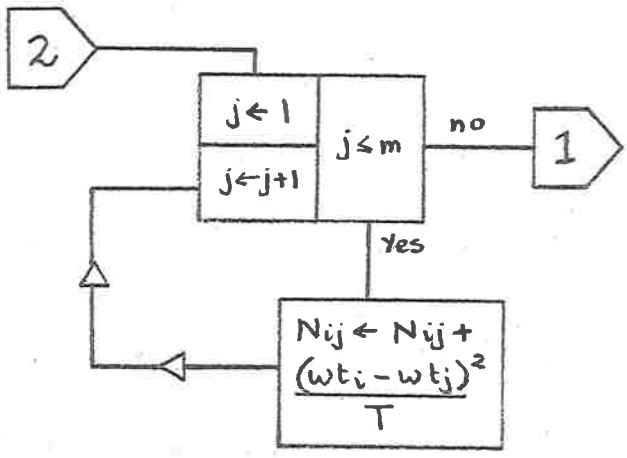


	Sequence Number	Word	Frequencies Ref set	Other
(a)	1	ACTS	1	1
	2	AS	1	1
	3	BIOCHEMICAL	3	1
	4	BIOLOGICAL	1	
	5	CATALYSE		1
	6	CATALYST	3	1
	7	ENZYME(S)	6	2
	8	FORMED	1	
	9	IN	3	
	10	MANY	1	
	11	NOT	1	
	12	PLACE	1	
	13	REACTION(S)	4	1
	14	SPEEDS	1	
	15	SYSTEM	1	
	16	TAKE	1	
	17	UP	1	
	18	WITHOUT	1	
(b)		A	6	1
		AN	4	1
		ARE	1	
		IS	2	
		WOULD		1

Figure 10.4.1 DICTIONARY FOR INTERPRETER #3.

(a) Words in dictionary

(b) Words deleted from input text



This is in form a fully recursive model; for each input word, a weighting is distributed to each node, so that the weightings are the same for two words that are exact synonyms, and decrease with difference from the input word; these weightings are then used to modify the network values, and the network is then renormalised so that the sum of squares of all its values is unity.

The results of this process have a correlation of 0.790 with the reference method. The results of the run are shown in figures 10.4.2 and 10.4.3.

It is quite clear that the time taken will be proportional to string length and the square of the number of network nodes; with the given data, total time per comparison was approximately one minute.

10.5 SUMMARY.

The three language interpreters demonstrated show quite

String	Λ	1	2	3	4	5	6
1	.0875	0					
2	.0906	.0059	0				
3	.0532	.0413	.0435	0			
4	.0927	.0142	.0134	.0488	0		
5	.0613	.0344	.0372	.0236	.0396	0	
6	.0724	.0475	.0486	.0405	.0563	.0409	0
7	.0530	.0414	.0436	.0003	.0489	.0237	.0405
8	.0913	.0142	.0127	.0478	.0185	.0426	.0484

Distances Between Strings

1	1 2 4 5 3 6 Λ
2	2 1 4 5 3 6 Λ
3	3 5 6 1 2 4 Λ
4	4 2 1 5 3 6 Λ
5	5 3 1 2 4 6 Λ
6	6 3 5 1 2 4 Λ
7	3 5 6 1 2 4 Λ
8	2 1 4 5 3 6 Λ
Λ	Λ 3 5 6 1 2 4

Reference Strings in Order of Closeness

Figure 10.4.2 RESULTS FOR LANGUAGE INTERPRETER #3

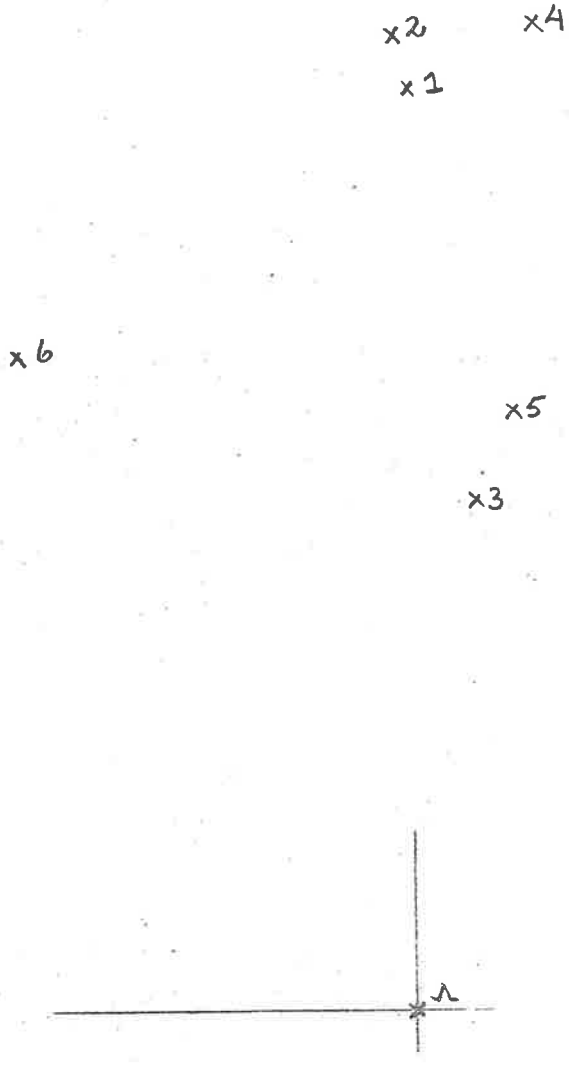


Figure 10.4.3 REPRESENTATION OF THE REFERENCE SET IN TWO DIMENSIONS - Model #3

reasonable agreement with the method chosen as a basis for comparison, and are considerably faster than it.

One advantage that the language interpreters have is that the effect of the input string need only be calculated once, whereas the reference method has to start from scratch for each individual comparison.

On the basis of the results given here, the language interpreters are seen to be an acceptable model for resolving the first part of the answer comparison problem.

CONCLUSION

In this work, we have considered two separate aspects of Computer Assisted Instruction: first, the design of a practical C.A.I. system, and second, the problem of answer comparison.

The Adelaide University system is still the largest scale C.A.I. system which has been designed and implemented in Australia, and offers many powerful facilities. Its implementation, even if only to an experimental stage, shows that Australia does have the capacity to make significant contributions to research in this area. It does appear likely, however, that practical application of C.A.I. techniques in this country will be centred more on the development of smaller scale systems involving less initial cost; even in this area, the design of the Adelaide system could well be taken as a guide for the kind of facilities that could be offered by such systems.

Answer comparison has been discussed, and some of the associated problems pointed out. The history of natural language data processing is a brief one, and it is possible that the renewed emphasis on the treatment of meaning, seen for example in the work of Wilks ([39], 1972) and Winograd ([40], 1972), will be more fruitful than the earlier work in this area. We have investigated some of the philosophical and practical problems involved in dealing with meaning. A class of language models adequate for the solution of the answer comparison problem has been proposed, and its relationship with some existing models for both natural and artificial languages investigated; the results of this indicate that the class of models is a useful frame-

work for the discussion of language models in general. It has also been shown that the language models proposed can produce results acceptable in a practical situation.

There are of course many other aspects of Computer Assisted Instruction; the design of a practical system, and the problem of answer comparison are certainly two of the most central, but other areas, such as the preparation of course material and the design of author languages, to name just two, are also important, and need further investigation. It appears inevitable that C.A.I. will eventually be a part of the normal educational process, but there is a need for considerable work in the area before this becomes possible.

APPENDIX A

PROOF OF THE TRIANGULAR PROPERTIES OF THE COMMON SELECTION PROCESS.

Notation : Let s_1, s_2, and s_3 be strings, and denote the maximum common selections by s_{12}, s_{13}, and s_{23}.

Lemma 1

$$l(s_{13}) \geq l(s_{12}) + l(s_{23}) - l(s_2)$$

Proof :

Both s_{12} and s_{23} are substrings of s_2.

Let $k = l(s_2)$, and

$$s_2 = w_1 w_2 w_3 \cdots w_k$$

Then s_{12} is form ed by choosing $l(s_{12})$ elements from this sequence, leaving $l(s_2) - l(s_{12})$ elements unused. Now s_{23} may be chosen from these elements, but if its length $l(s_{23})$ exceeds $l(s_2) - l(s_{12})$ then every element by which it exceeds this must also be an element of s_{12}.

Thus s_{12} and s_{23} have at least

$$l(s_{23}) - (l(s_2) - l(s_{12})) = l(s_{12}) + l(s_{23}) - l(s_2)$$

common elements which must be in the same order.

However, these are selections of s_1 and s_3 respectively, and hence s_1 and s_3 must have at least this many common elements.

i.e.

$$l(s_{13}) \geq l(s_{12}) + l(s_{23}) - l(s_2)$$

as was to be shown.

Theorem 1

If s_1, s_2, and s_3 are of the same length,

$$d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$$

Proof :

This is straight from the lemma. Let $l(s_1) = l(s_2) = l(s_3) = l$.

Then

$$l(s_{13}) \geq l(s_{12}) + l(s_{23}) - l$$

$$\frac{l(s_{13})}{l} \geq \frac{l(s_{12})}{l} + \frac{l(s_{23})}{l} - 1$$

$$1 - \frac{l(s_{13})}{l} \leq 1 - \frac{l(s_{12})}{l} + 1 - \frac{l(s_{23})}{l}$$

$$d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$$

as required.

The general case is more difficult to handle, and we break it up into several cases. We note that because of symmetry between s_1 and s_3 in the required formula, we may assume $l(s_1) \geq l(s_3)$, and consider only three cases.

Lemma 2

If $l(s_2) \geq l(s_1)$ then the triangle inequality holds.

Proof :

Fromm lemma 1,

$$l(s_{13}) \geq l(s_{12}) + l(s_{23}) - l(s_2)$$

$$l(s_2) - l(s_{13}) \leq l(s_2) - l(s_{12}) + l(s_2) - l(s_{23})$$

$$1 - \frac{l(s_{13})}{l(s_2)} \leq 1 - \frac{l(s_{12})}{l(s_2)} + 1 - \frac{l(s_{23})}{l(s_2)}$$

But $d(s_1,s_3) = 1 - \frac{l(s_{13})}{l(s_1)}$

and, since $l(s_1) \leq l(s_2)$

we have $d(s_1, s_3) \leq 1 - \frac{l(s_{13})}{l(s_2)}$

and the result follows.

Lemma 3

If $l(s_1) > l(s_2) \geq l(s_3)$ then the triangle inequality holds.

Proof :

As before,

$$l(s_{13}) \geq l(s_{12}) + l(s_{23}) - l(s_2)$$

$$l(s_1) - l(s_{13}) \leq l(s_1) - l(s_{12}) + l(s_2) - l(s_{23})$$

and dividing through by $l(s_1) \geq 0$

$$d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)\ \frac{l(s_2)}{l(s_1)}$$

Since $\frac{l(s_2)}{l(s_1)} < 1$ the result follows.

Lemma 4

If $l(s_1) \geq l(s_3) > l(s_2)$ then the triangle inequality holds.

Proof :

As before,

$$l(s_{13}) \geq l(s_{12}) + l(s_{23}) - l(s_2)$$

$$l(s_1) - l(s_{13}) \leq l(s_1) - l(s_{12}) + l(s_2) - l(s_{23})$$

$$\leq l(s_1) - l(s_{12}) + l(s_3) - l(s_{23})$$

and, dividing through by $l(s_1) \geq 0$

we obtain

$$d(s_1, s_3) \leq d(s_1, s_2) + \frac{l(s_3)}{l(s_1)} d(s_2, s_3)$$

and since $\frac{l(s_3)}{l(s_1)} \leq 1$

$$d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$$

as required.

Theorem 2

For any three strings s_1, s_2, and s_3,

$$d(s_1, s_3) \leq d(s_1, s_2) + d(s_2, s_3)$$

Proof :

The above lemmas have showed the result to hold in each possible case. The result is therefore generally true.

APPENDIX B

THE LONGEST COMMON SELECTION PROCEDURE.

The process of finding a longest common selection from two given strings s1 and s2 is a recursive one. The simplest (although rather inefficient) method can be readily described by the ALGOL like formula:

```
integer procedure MATCH (p1, p2);
MATCH := if p1 > l1 ∨ p2 > l2 then 0
    else if s1[p1] = s2[p2] then 1+MATCH(p1+1, p2+1)
        else MAX(MATCH(p1+1, p2), MATCH(p1, p2+1));
```

where the characters are assumed to be coded in the arrays

```
s1[1:l1], s2[1:l2]
```

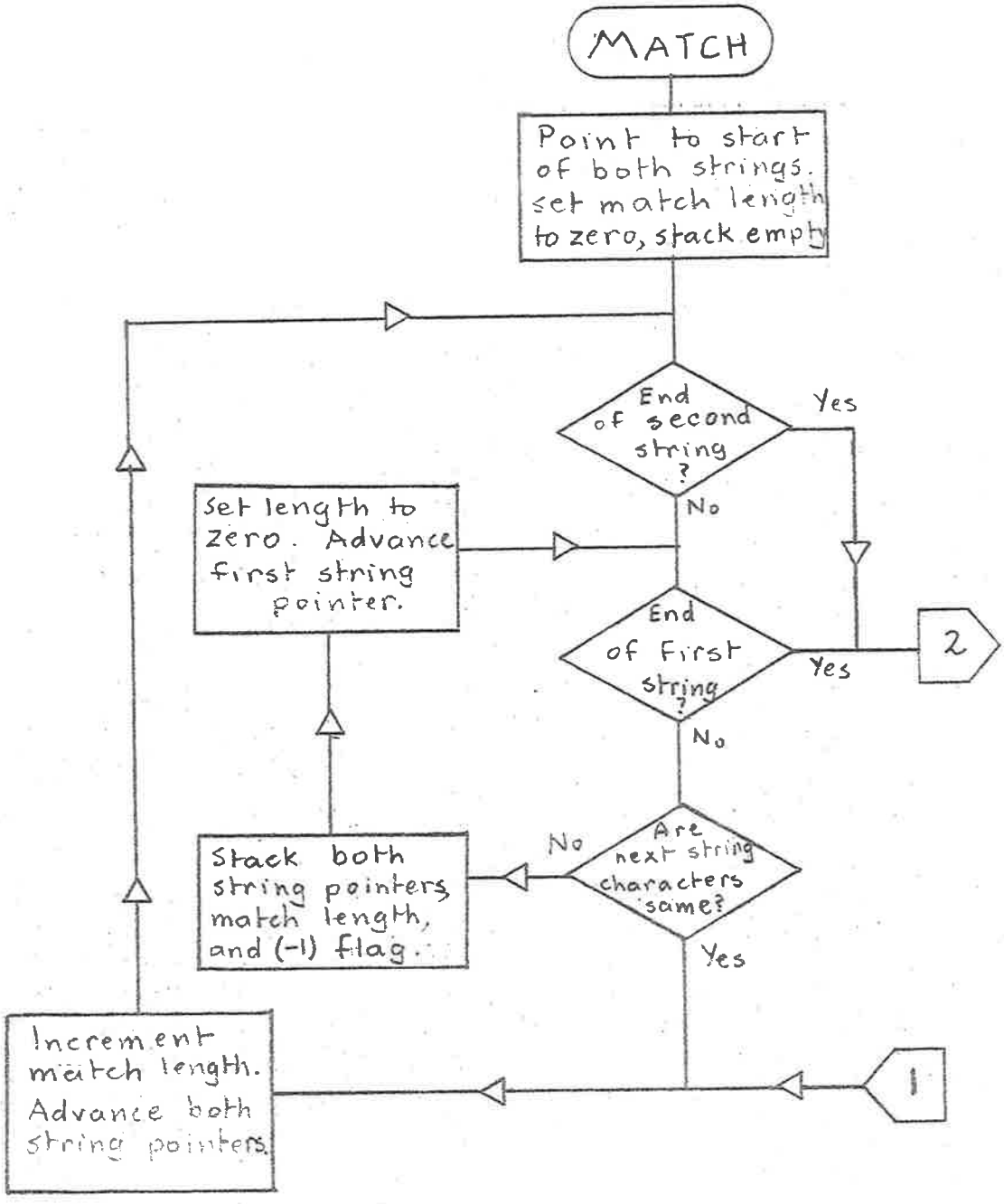
and the match length is found by calling

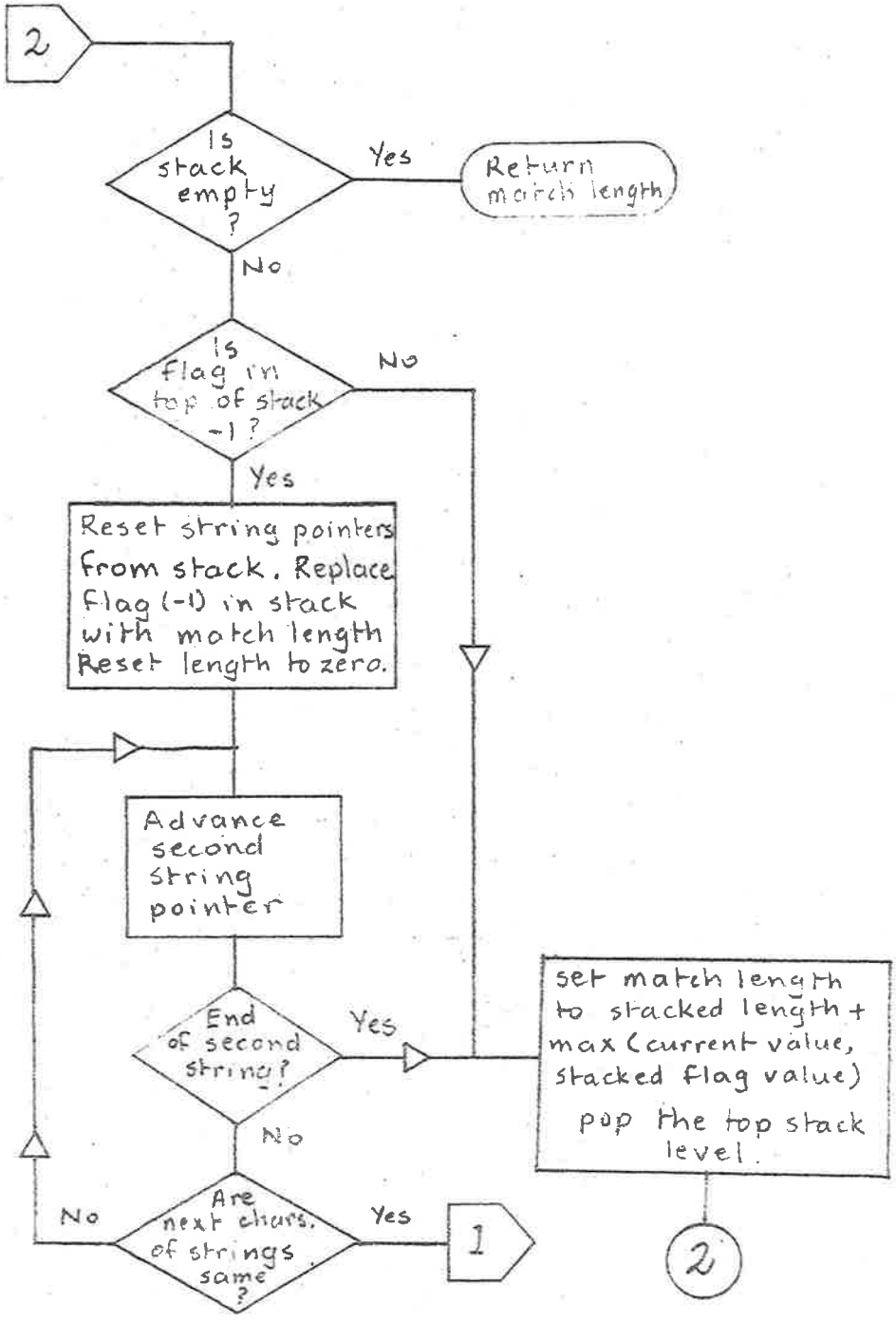
```
MATCH(1, 1);
```

This process, however, checks many of the possibilities twice, and so the method actually used is a modification of this which overcomes the inefficiencies in the calculation of

```
MAX(MATCH(p1+1, p2), MATCH(p1, p2+1))
```

by evaluating only the first part of this and then scanning along the second string for a character equal to the character broken out of the first string. The method was coded in assembly language, in order to increase the speed, and a stack with four entries in each level was used; these entries were the current values of the pointers p1 and p2, the current match length, and a flag set to -1 while evaluating the first part, and to the result of this while breaking characters out from the second string.


MATCH
Point to start of both strings. set match length to zero, stack empty
End of second string ?
Yes
No
Set length to zero. Advance first string pointer.
End of First string ?
Yes
2
No
Are next string characters same?
No
Stack both string pointers, match length, and (-1) flag.
Yes
Increment match length. Advance both string pointers.
1


2
Is stack empty ?
Yes
Return match length
No
Is flag in top of stack -1 ?
No
Yes
Reset string pointers from stack. Replace flag (-1) in stack with match length Reset length to zero.
Advance second string pointer
End of second string?
Yes
set match length to stacked length + max (current value, stacked flag value)
pop the top stack level.
No
Are next chars. of strings same ?
No
Yes
1
2

The resulting procedure is quite rapid for short strings, or for strings with a high degree of matching (and hence relatively little genuine recursion), but deteriorates rapidly with increasing string length and difference.

BIBLIOGRAPHY

1 Alberga,C.N. 1967 "String Similarity and Misspellings"
Communications of the A.C.M. Vol 10, no 5; May 1967.

2 Allen,J.P.B. & VanBuren,P. (Eds) 1971
Chomsky: Selected Readings Oxford U.P., London

3 Bar-Hillel,Y. 1964
Language and Information Addison-Wesley, Reading(Mass)

4 Berztiss,A.T. 1971
Data Structures Academic Press, New York

5 Borko (Ed) 1962
Computer Applications in the Behavioural Sciences
Prentice-Hall, New York

6 Control Data Corp 1966
6400/6600 Computer Systems Reference Manual Pub 60100000

7 Davis,M. 1958
Computability and Unsolvability McGraw-Hill, New York

8 DeCecco,J.P. 1968
The Psychology of Learning and Instruction
Prentice-Hall, Englewood Cliffs (N.J.)

9 Feigenbaum & Feldman (Eds) 1963
Computers and Thought McGraw-Hill, New York

10 Frye,C.H. 1968 "C.A.I. Languages: Capabilities and Applications"
Datamation Sept 1968, pp34-37.

11 Green,B.F., Wolf,A.K., Chomsky,C., & Laughery,K. 1963
"Baseball: An Automated Question Answerer"
in Feigenbaum & Feldman [9]

12 Hays,D.G. 1962 "Automatic Language Data Processing"
in Borko [5]

BIBLIOGRAPHY (cont)

13 Holloway,J. 1951
Language and Intelligence MacMillan, London

14 Joos,M 1958 "Description of Language Design"
in Joos (Ed) Readings in Linguistics
American Council of Learned Societies, New York

15 Koch,S (Ed) 1963
Psychology: A Study of a Science,Vol VI McGraw-Hill, New York

16 Lee,K.C. 1971
Supervision of Trigonometric Proofs for C.A.I.
Doctoral Thesis, University of Adelaide, Adelaide, South Australia

17 Lewis,P.A.W., Baxendale,P.B., & Bennett,J.L. 1967
"Statistical Determination of the Synonymy/Antonymy Relationship Between Words"
Journal of the A.C.M. Vol 14, no 1; Jan 1967; pp 20-44

18 Miller,G.A. 1968
The Psychology of Communication Allan lane-Penguin Press, London

19 Minsky,M.L. (Ed) 1968
Semantic Information Processing M.I.T Press, Cambridge (Mass)

20 Morris,C.W. 1946
Signs, Language, and Behaviour Prentice-Hall, New York

21 Morrison,H.W.
Computer Processing of Responses in Verbal Training
I.B.M Research Report RC1133

22 Nelson,R.J. 1968
Introduction to Automata Wiley, New York

23 Ogden,C.K. & Richards,I.A. 1923
The Meaning of Meaning Kegan-Paul, London

BIBLIOGRAPHY (cont)

24 Oldehoeft,A.E. 1970
Computer Assisted Instruction in Teaching Numerical Methods
Doctoral Thesis, Purdue University, Lafayette, Indiana

25 Osgood,C.E. 1963 "Psycholinguistics"
in Koch [15]

26 Osgood,C.E., Succi,G.J., & Tannenbaum,P.H. 1957
The Measurement of Meaning University of Illinois Press, Urbana

27 Ovenstone,J.A. 1966 "C.A.I. in Undergraduate and Postgraduate Medicine" The Medical Journal of Australia
Vol 2, no 10; Sept 1966; pp 487-492

28 Perry,P.G. 1969 "The Adelaide University C.A.I. Project"
in Smith [35]

29 Perry,P.G. & Lee,K.C. 1969 "A Computer Assisted Instruction System" Proceedings of the Fourth Australian Computer Conference
Vol 1, pp 401-406 Griffin Press, Adelaide

30 Potter,R.J. 1967
Computer Assisted Instruction Unpublished report, Dept of Computing Science, University of Adelaide, South Australia

31 Rubenstein,H. & Goodenough,J.B. 1965
"Contextual Correlates of Synonymy"
Communications of the A.C.M. Vol 8, no 10; Oct 1965; p627

32 Sanderson,J.G. 1967 "A Basis For a Theory of Programming Languages" Australian Computer Journal Vol 1, Nov 1967, pp 21-27

33 Simmons,R.F. 1962 "Synthex: Computer Synthesis of Human Language Behaviour" in Borko [5]

34 Smallwood,R.D. 1962
A Decision Structure for Teaching Machines
M.I.T. Press, Cambridge (Mass)

35 Smith (Ed) 1969
The Role of the Computer in the Secondary School
Proceeding of a weekend seminar in Adelaide.
Australian Computer Society, Canberra

36 Turner,C.W. 1962 "The Passive Construction in English Scientific Writing" Aumla No 18; Nov 1962; pp 181-197

37 Ullmann,S. 1962
Semantics: An Introduction to the Science of Meaning
Basil Blackwell, Oxford

38 Weizenbaum,J. 1966 "Eliza - A Computer Program for the Study of Natural Language Communication Between Man and Machine"
Communications of the A.C.M. Vol 9, no 1; Jan 1966; pp 36-45

39 Wilks,Y.A. 1972
Grammar, Meaning, and the Machine Analysis of Language
Routledge & Kegan Paul, London

40 Winograd,T.J. 1972 "Understanding Natural Language"
Cognitive Psychology Vol 3, no 1; Jan 1972