SUPERVISION OF TRIGONOMETRIC PROOFS

FOR COMPUTER ASSISTED INSTRUCTION

by

Lee Kim Cheng, B.Sc.

Department of Computing Science

University of Adelaide

20 August 1971

Thesis submitted in fulfillment of the requirements

for the degree of Doctor of Philosophy

TABLE OF CONTENTS

CHAPTER PAGE

I INTRODUCTION 1

Survey of CAI Literature 1

Scope of this Work 13

Brief History of this Work 14

II THE UNIVERSITY OF ADELAIDE COMPUTER ASSISTED INSTRUCTION SYSTEM 21

Basic System Features 21

The CAI Consoles 29

Operating Features 30

Student Working Record Format 32

Author & Student's Views of System 34

Recovery of Statistics 35

III THE AUTHOR LANGUAGE ALFIE 38

Compiling ALFIE Programs 38

Register Arithmetic 45

The Problem Block 46

Specimen Course and Interaction 49

Cueing Feature of ALFIE 51

IV EXPRESSION EQUIVALENCE AND CONSISTENCY SETS 57

Problem Scope 57

Difficulty in Proof-Checking 59

Preliminary Notions and Definitions 60

Expression Equivalence 62

CHAPTER PAGE

The Work of Caviness 63

The Theory of Consistency Sets 66

On Appendix B 70

Trigonometric Substitution 72

C-set of a Step 75

The Representative Point Technique 76

Simple Illustrations 78

V THE STEP-SIZE PROBLEM 81

Discussion 81

Φ-Operations 84

Three Basic Forms 86

Model-A .. A Model for Small Steps 88

Adequacy of Model-A and Model-B 94

VI ASSISTING PROOF CONSTRUCTION 98

Problem-Solving Systems 101

Heuristic Generation of Trigonometric Proofs 104

Substep Resolution 106

C-set Reduction 106

Algebraic Manipulation 108

Latent Structure Probing 110

A Heuristic Proof Constructor 115

Hand Simulation of HPC 119

VII PRACTICAL CONSIDERATIONS 123

C-Set Determination 123

Zero Approximation 125

CHAPTER PAGE

Finding the C-set of a Step 126

Configuration Directed Search 128

The C-set Search Table 129

Substep Resolution 133

Primary and Secondary Resolution 137

Sequential Resolution Procedure 138

Data Structures 139

Index Theory 142

Using Indices in our Data Structures 147

Applying Index Theory to Formula Manipulation in Prefix Polish 149

Super-2 156

VIII CONCLUDING REMARKS AND RECOMMENDATIONS 161

Suggestions for Further Work 164

APPENDIX-A MANUAL AND AUTHOR GUIDE FOR ALFIE

APPENDIX-B THEORETICAL DEVELOPMENT

APPENDIX-C TABLES

APPENDIX-D LISTING OF SUPER-2

BIBLIOGRAPHY

LIST OF TABLES

TABLE		PAGE
4.1	Reference Identities	58
4.2	C-set Table	69
4.3	Identities and their C-sets	80
5.1	Agreement Percentages	96
6.1	Test for Latent Structures	113
6.2	Examples of Latent Structure	114
7.1	Substitution Table	124
7.2	Some Values of S(n,k)	136
7.3	Subexpression Detachment	154
A.1	Card Codes	A-4
C.1	Opinion Data	C-1
C.2	C-set Search Table	C-4
C.3	Step-Size Analysis	C-6
C.4	Latent Structure Analysis	C-13

LIST OF FIGURES

Figure		Page
2.1	Central Memory Utilisation	26
2.2	UACAIS Configuration	31
3.1	Sample Listing from ALFIE	50
3.2	Two Instructional Paradigms	53
7.1	C-set Search Network	127
7.2	A Structured List	149
7.3	Component Detachment	153
7.4	Multiple Component Detachment	155
7.5	A List Cell	158
A.1	Source Card Listing of a Course	A-8
A.2	Assembly Listing of a Course	A-9
A.3	Compiling an ALFIE Course	A-10
A.4	Schematic of a Course	A-35

ABSTRACT

This work consists of two parts. One concerns the development of the University of Adelaide Computer Assisted Instruction System (UACAIS) and the other is an investigation into the problem of supervising trigonometric proofs in CAI.

UACAIS is a dedicated CAI system based on a Control Data 6400 Computer and is designed to support a large number of student consoles. The associated author language ALFIE is cue-oriented. An experimental version of UACAIS has been successfully implemented.

Proof supervision refers to the dual task of checking and of assisting trigonometric proofs. The problems are confined to the single argument case. To simplify treatment, proofs are assumed to have the form $e_0 \rightarrow e_1 \rightarrow \ldots \rightarrow e_n$ where the expressions e_0 and e_n are directly related to the identity being proved and where each $e_i \rightarrow e_{i+1}$ is a step deriving e_{i+1} from e_i.

Proofs are checked step-by-step as they are entered. Each step must be both correct and small. The correctness of a step is a problem in expression equivalence. Step-size is however elusive, being largely dependent on subjective judgement.

The step-by-step checking of a proof can help the student considerably in his proofs by preventing them from going astray. More explicit types of assistance can take

the form of a set of relevant identities for substitution, a next-step to help him continue his proof, or information that no further trigonometric substitution is required. Much of such forms of assistance can be provided with the aid of an automatic proof generator.

Although proof supervision is essentially a symbolic problem, the main emphasis in the solutions proposed is on the use of numeric techniques. The numeric approach is advocated in the belief that it is superior to a purely symbolic one. Several numeric techniques are discussed, including a simple test for deciding the correctness of a step. The most important of these is C-set determination which enables the supervisor to discover a minimum sufficient set of basic identities for proving a given problem identity. An identity requiring no substitution for its proof has an empty C-set. Some theoretical justification for the theory of C-sets is given, using algebraic geometry. Hilbert's Nullstellensatz plays an important role. The relevance of C-sets in proof-checking and in proof construction are discussed.

A schema for defining models of small steps has been developed. Two models based on it have been selected for closer study. A survey of human opinions on step-size was conducted to provide data for gauging the adequacy of these models. Good agreement between both models and the human data was obtained.

The main ideas for proof-checking have been implemented in a program called Super-2. The problem of implementing efficiently the various numeric techniques advocated has also been considered.

DECLARATION

To the best of my knowledge and belief, this thesis contains no material which has been previously published or written by another person, except where due reference is made in the text.

None of the material in this thesis has been accepted for the award of any other degree or diploma in any University.

Lee Kim Cheng

ACKNOWLEDGEMENTS

I am indebted to Professor J.A. Ovenstone and Dr. J.G. Sanderson, my original and current supervisors respectively, for their advice and sympathetic support during my research. Dr. J.L.C. Macaskill, in his capacity as acting head of the Department of Computing Science, has also been very helpful to me.

It was indeed a pleasure to work with Mr. Peter Perry on the CAI project and I have benefitted great~~ed~~ly from this association.

Mrs. Gerry Wellings has undertaken a difficult typing assignment and I am immensely grateful to her for having done a good job of it.

I gratefully acknowledge my indebtedness to the University of Adelaide for supporting my research, since October 1967, with a Postgraduate Scholarship under the University Research Grant.

CHAPTER I

INTRODUCTION

This work is concerned with the problem of supervising trigonometric proofs in computer assisted instruction (CAI). It also involves the development of a CAI system.

Survey of CAI Literature

Since it first came into being some twenty-five years ago, the general purpose digital computer has come to play a very significant role in society. Computer technology has been developing so rapidly that it is now being applied to practically every major aspect of human activity, including education. Today the computer is finding increasing use in teaching and in educational administration.

In the last fifty years vigorous efforts have been made to produce better teaching methods. S.L. Pressey (see [41]) beginning in 1926 devised a series of simple devices for administering and scoring objective tests. He found that these devices could also be used to teach and saw in them the exciting possibility for automating and individualising instruction. However the work of Pressey and his students passed largely unnoticed.

It was not until the mid-1950's that the work of B.F. Skinner and his colleagues (see [65]) at Harvard University generated renewed interest in auto-instructional methods and teaching machines. This was soon to inspire the programmed instruction (PI) movement and its wider implications we now call educational technology.

Skinner argues that his extensive work on operant conditioning has direct relevance for the mechanisation of instruction. He developed teaching programs for machine presentation that were characterised by linearity, small steps, constructed (i.e. overt) responses, low error rate, immediate knowledge of result, and the reinforcement of correct responses. Courses prepared in this way are referred to as linear or skinnerian programs.

Following the development of the linear program, other instructional paradigms have been devised. Crowder [14] introduced the intrinsic programming technique which features branching, larger steps, multiple-choice response and remedial sequences. Two other important instructional methods are the mathetics of Gilbert and structural communication of Bennet et al. (see [42]).

Early auto-instructional materials were presented by mechanical devices. These devices, called teaching machines, can display a segment of a course, and have provisions for accepting answers, usually of the multiple-choice type. Subsequent work has shown that instructional material can often be presented as effectively in the book medium as by teaching machines.

Schramm [62] sums up the essentials of PI as :-

(a) an ordered sequence of stimulus items

(b) to each of which the student responds in some specified way

(c) his responses being reinforced by immediate knowledge of results

(d) so that he moves by small steps

(e) thereby making few errors and practising mostly correct responses

(f) from what he knows, by a process of successively closer approximation towards what he is supposed to learn from the program.

Lumsdaine and Glaser [41] is a standard reference on the early works on teaching machines and programmed learning. The reader may also consult Unwin and Leedham [71], Dunn and Mulroyd [17] and Mann and Brunstrom [43] for more recent efforts in educational technology.

Although PI is potentially a very powerful approach to teaching, its effectiveness and flexibility have been seriously restricted by the traditional media of teaching machines and programmed texts. The digital computer, with its vast information processing and storage capability, and its ability to control multi-media learning stations, alone appears to be able to offer the full potential of programmed learning.

Silvern and Silvern [64] define CAI as a man-machine relationship in which the man is a learner and the machine is a computer system. A two-way communication exists between the human learner and the computer in which there is a stimulus-response-feedback interaction producing learning.

The reader is referred to Coulson [13] and Rath [59] for an account of early CAI development. For a recent review of the field, Hickey [33], Feldhusen and Szabo [22] and Engel [19] may be consulted.

CAI began in 1958 when Rath et al. [60] programmed an IBM 650 to teach binary arithmetic. In the next three years it was investigated at several centres which include Systems Development Corporation, Bolt, Beranek and Newman and the University of Illinois (see [59]). Since then the growth in the number of CAI projects, especially in the United States, has been remarkable. Recent CAI systems have become increasingly large and sophisticated.

Only a few prototype systems were reported by Rath [59] to exist over the 1968-61 period; each had no more than a few simple typewriter terminals. Today there appears to be more than a hundred institutions in the United States engaged in CAI activity. The New York City Board of Education's RCA Instructional 70 System [7] supported 192 terminals serving sixteen schools and there were plans to expand it. More recently, the University of Illinois was reported [18] to be planning for a system with more than 4000 stations using the "Digivue" display device.

Despite a decade of research and development (R & D), results obtained in CAI have been meagre [34]. Nevertheless the works of many centres including those of Hansen and Dick [31] at the Florida State University, and Suppes and his associates [67] at Stanford have clearly demonstrated the technical feasibility of CAI.

There is a small, but rapidly growing, number of studies on the comparative effectiveness of CAI. Their findings, although preliminary in nature, are encouraging and suggest that CAI is at least as effective as the other instructional methods under comparison. We cite a few such studies below.

One of the earliest studies, by Grubb and Selfridge [29] reported very marked superiority of CAI over conventional instruction. They taught six college students a half-semester course on statistics by CAI and another group by conventional lectures. The CAI group took an average of only 5.33 hours to complete the half-semester course and scored an average of 94.3%

in an achievement test compared with an average of 58.4% for the conventional lecture group. The smallness of the CAI group must be borne in mind. Results from similar subsequent studies have been more modest.

Suppes and Jerman [68] at Stanford taught thirty students a computer-based course on elementary Russian. A control group received the same course in a conventional way. The CAI group was taught solely via teletype with Cyrillic keyboard and audio-tapes with earphones, fifty minutes daily, five days per week throughout the academic year. At the end of the year the CAI group was found to perform at a statistically higher level than the control group and to suffer a much lower dropout rate.

Love [40] presented a CAI course on Boolean algebra to groups of students; one group received the lessons individually whereas the other group did so in pairs. Both groups were later found on the average to perform equally well with respect to scores, time and error rate. This finding could mean a significant improvement in the cost-effectiveness of CAI.

Expressed student attitudes towards CAI have been very favourable. Thus Borman [5], Oldehoeft [52], Butler [7], Love [40] and Bell [2] are just a few investigators to report very positive reactions to CAI by their subjects.

With the technical feasibility of CAI well-established, R & D efforts have now been directed towards the wider problem of making CAI more effective and more widely applicable. Areas under investigation include instructional strategies, suitability of subject areas, hardware, software and economics. Needless to say, these problems are generally closely interrelated.

Most early CAI courses are essentially computerised versions of programmed instruction. Since CAI in a sense grew out of PI, this is hardly surprising. The student is presented the course frame-by-frame. Each frame is usually accompanied by one or more tests to ensure understanding. He responds to questions or problems by typing in his answers which are then analysed by the computer. If desired, responses can be recorded for subsequent investigation. Branching based on the student's most recent responses can be incorporated. Reinforcements in the form of encouraging remarks often accompany correct answers.

The drill and the tutorial are two important modes of instruction in CAI. Drills are usually test sequences on which students practise in order to reach a satisfactory level of mastery. The binary arithm~~a~~etic course of Rath et al. [60] is ~~i~~essentially a drill. Among the best known examples of drill in regular instruction are the elementary school courses in mathematics and language skills developed by Suppes and Atkinson [67] for the Palo Alto schools.

Most courses in CAI are programmed in the tutorial mode. In this mode, actual teaching function is undertaken. Learning objectives are carefully identified in terms of the student's entry and desired target behaviours. The latter is attained via a sequence of intermediate behaviours; at the course level these are effected by a carefully prepared program of steps which permit the stimulus-response-reinforcement interaction to occur.

The drill and the tutorial are, however, not the only modes of computerised instruction. Several other modes exist - each exploiting some aspect of the computer's versatility, to facilitate a form of teaching for which it is best suited. Thus we have the inquiry/dialogue, simulation and gaming, information retrieval and the computational-aid modes of computer-assisted teaching.

In the dialogue mode the student is encouraged to undertake greater initiative in the learning; to direct queries on points of difficulty or doubt; in general to engage the system in some form of restricted discussion. It is exemplified by the medical diagnosis exercise of Feurzeig [23] and the work of Taylor [69]. However the dialogue capability is still primitive.

The computer is a powerful medium for conducting gaming and simulation exercises. Unlike the drill, tutorial and dialogue modes, student interactions for these exercises are not programmed step-by-step. Rather, the rules of the game and the model underlying the simulation are incorporated into the program. This mode of CAI is useful in imparting decision-making skills in management sciences and military exercises. Examples of gaming and simulation programs can be found in [38] and [32].

Information retrieval is another way in which the computer can aid learning. With carefully organised data, students can use their own initiative to search for the information they require. One important work in this direction is that of Grubb [28] in his learner-controlled statistics course, which is organised as non-linear files of text and a series of maps with which the students can chart their own paths. An advantage of this approach is that detailed frame-by-frame programming becomes unnecessary.

Desk-calculator facilities are provided in PLANIT [21] and several other systems. A more sophisticated form of computational aid are the systems of Culler and Fried [15] and Oliver and Brooks [53]. These systems, though not designed specifically for CAI usage, can provide a useful on-line display and computation facility for teaching numerical analysis.

CAI systems employ two basic levels of software :-

(1) the operating system that drives the hardware

(2) the teaching programs.

The operating system usually includes programs for presenting instructional materials; accepting, analysing and recording student responses; and various peripheral equipment drivers. There are also softwares for course preparation and validation, on-line or off-line.

To facilitate and simplify the task of preparing courses for computer presentation, author languages have been developed. Zinn [73] is so far the most comprehensive document on languages for instructional programming. In it more than thirty languages have been studied, classified and assessed. He notes that despite their variety, many of their differences are superficial, leaving some user needs still unmet. He notes four classes of instructional languages according to their operational characteristics :-

(1) presentation by successive frames

(2) conversation within a limited context

(3) presentation of a curriculum file by a standard procedure, and

(4) data analysis and revision of materials.

Such a classification is useful but it should be noted that most languages do not fall neatly into a single category.

The majority of CAI languages belong to class 1. Examples are COURSEWRITER (various versions, IBM), PLANIT (SDC), COMPUTEST-II (U of Calif. Medical Centre, San Francisco), and INFORM (Philco-Ford). Of these COURSEWRITER is perhaps the most widely used. Class 1 languages generally have convenient facilities for the display of text, acceptance and analysis of student responses, recording of performance data, and branching based on the student's answers or his response history.

MENTOR (BBN) and ELIZA (MIT) are typical examples of class 2 languages. MENTOR has been used [23] to teach medical diagnosis via conversational interaction. ELIZA has been used [72] by its developer, Joseph Weizenbaum, to simulate a psychiatric interview, which conveys a surprisingly good impression of dialogue and understanding. This has been achieved by analysing input sentences on the basis of decomposition rules associated with keywords in the text. Selected decomposition rules are then used to assemble machine responses in the conversation. ELIZA has subsequently been used for developing teaching programs [69], which again permit the above kind of dialogue.

Two examples of languages for presenting curriculum files by a standard procedure, i.e. class 3 languages, are CATO of the PLATO project of Illinois University and CG-2 of Meadow et al. (see [73]). Such languages are characterised by [flexibility for building ~~the rigidity of their built-in~~ teaching strategies and the consequent ease in which courses can be prepared. Thus ~~in~~ with programs prepared using CATO, the author's task is little more than the insertion of slides for the corresponding slots for questions, hints, answer explanations, and keying in the correct answers at a keyboard.

It is not clear if class 4 forms a useful classification since every language for CAI has some facility for data analysis and course revision and therefore belongs to the class. [Class 4 has been replaced in a later report by Zinn]

Some class 2 languages like MENTOR and FOIL [32] have also been used for programming games and simulation. It is also worth noting that some author languages are extensions of existing standard algorithmic languages which by themselves would be very inconvenient, if not impossible, for writing courses. Thus MENTOR is an extension of LISP and CATO and FOIL are extensions of FORTRAN.

Several author languages permit students to use the computer for computation while taking a course. These include COURSEWRITER which has a service program called DESCAL, and PLANIT which incorporates a fully-integrated CALC mode that can be used interactively by authors during lesson building and by students during lessons. The CALC mode of PLANIT contains rather sophisticated computational routines for functions, matrices and statistical tables. Interactive computing is a basic feature of languages like APL and BASIC, but these are not CAI languages.

Every CAI language has an answer-processing capability which varies from the recognition of multiple-choice responses to the recognition of complex algebraic expressions. Some of the commoner CAI answer-matching routines are [25] :-

Type of Match	Language
Exact string	all languages
keyword	all languages
selected string	CAL, COURSEWRITER
Percentage	COURSEWRITER, LYRIC
partial string	CAL, COURSEWRITER
phonetic	ELIZA, PLANIT
algebraic expression	PLANIT
numeric (within specified limits)	LYRIC, PLANIT
calculated numeric	CATO, PLANIT

Cost has been a severe constraint on the implementation of CAI. Estimates for the cost per student console hour vary from as low as 20¢ (see [74]) to as high as over $80 [9]. These differences in estimates are due to variations in assumptions regarding hardware, course development, operational and other costs. The reader is referred to Chapin [9], and Kopstein and Seidel [36] for their treatments on the problems in estimating costs in CAI and on the economics of CAI generally.

It is widely agreed that CAI cannot at present be justified purely on the basis of cost. Indeed there are some who do not even regard it to be a partial replacement for traditional teaching; rather they see it as an experimental learning and teaching laboratory in which learning and instructional hypotheses can be tested under a fairly realistic, and yet controllable, learning environment. There are also those who see CAI as a powerful means for testing and validating teaching materials, which can then be adapted for cheaper conventional presentation.

Several points have been cited in favour of CAI: it is auto-instructional, cheat-proof; it is said to be capable of providing independent, simultaneous, mass, multi-media instruction with great scope for individualisation both by pre-programming and by dynamic adaptation. It is also capable of recording student performance data, including responses to questions. Not all these advantages have been fully realised or convincingly demonstrated.

Several modes of computer assisted instruction have been mentioned earlier. No single mode can be regarded as being unqualifiedly superior to another; some are more suited than others, for certain teaching objectives. However there is still much scope for the development of new CAI modes.

One promising direction for investigating new instructional modes is that of intelligent CAI systems. By this term we mean a system which simulates some aspect of human intelligence. The vast information processing capability of modern computers seems to have been hardly tapped. In most CAI systems the computer is little more than an information storage and retrieval device, with very little data manipulation required.

Important work is being done in the area of artificial intelligence (see [47, 48]). Slagle [66] has written a LISP program for integrating elementary expressions symbolically. Later, Moses [49] developed a more powerful program which can solve a wide range of textbook integration problems with practically no wrong attempts at all. D.G. Bobrow [4] developed a program for solving high school story algebra problems. These problems, posed in simple English, are converted into their equivalent equations and then solved. Raphael's [58] SIR program is a question-answering system which is able to accept (understand) simple English sentences expressing certain relationships between objects. It is also able to answer simple questions on relationships about objects, given enough information.

Successful automatic theorem proving programs for first-order predicate calculus have also been written (see Robinson [61]).

This author feels that the important results already obtained from research in artificial intelligence would be useful for the development of intelligent CAI systems. In particular, as John McCarthy [45] has pointed out, any program for generating proofs can be incorporated into a proof checker - and this has relevance for proof-supervision in CAI, of which more will be said later.

Scope of this work:

The work to be described in this thesis involves initially the development of a CAI system - the University of Adelaide Computer Assisted Instruction System or UACAIS. The aims of this undertaking were :-

(a) to create a facility for investigating CAI;

(b) to enable the author to acquire practical experience in CAI and an appreciation of some of its problems;

(c) on the basis of experience in CAI gained in (b), to single out problems for further investigation.

This initial work led to the subsequent investigation into the problems associated with supervising proofs in elementary trigonometry.

A Brief History of this Work:

While CAI systems were growing rapidly in number and sophistication overseas, none existed* in Australia in 1966. That year Ovenstone [54] outlined how a large CAI system might be implemented on the University of Adelaide's Control Data 6400 computer. The following year, P.G. Perry and K.C. Lee [55] undertook** to design and implement a CAI system based on the CDC 6400 as part of their graduate research program. This work was under the supervision of Professor J.A. Ovenstone. Terminal and interface hardware was developed by Dr. R.J. Potter [56] and became available at the end of 1968.

The first phase of UACAIS has been completed. It includes the development of the operating system and an author language called ALFIE (Adelaide Language for Instruction & Education). These will be described in the next two chapters.

As the development of UACAIS progressed, a number of problems confronting CAI became obvious. One of these lies in the area of response processing and concerns the analysis of certain types of responses occurring in mathematical instruction.

* *In 1967 a simple CAI capability was incorporated into a time-shared PDP-6 computer at the University of Western Australia.*

** *The development of UACAIS has been a joint, cooperative effort between Peter Perry and the author. It is not easy to attribute any major portion of this development completely to any one of us. Perry played the major role in the overall development and was largely responsible for the implementation of the CAI operation system, including the resident central program, system monitor and interface driver. The author developed the language ALFIE and its associated compiler. He also programmed the resident disc driver and display programs as well as a number of routines in the operating system.*

It has been noted by Glaser [27] that an essential component of the auto-instructional process is performance assessment. The system must continually determine whether the desired learning goals are being attained or not, and this requires response processing. However when we consider the teaching of mathematics, we find that existing capabilities are inadequate for assessing certain kinds of performance. Problem solving and proof construction are skills fundamental in mathematical training. To determine whether a student can solve a problem* or not, it is normally quite sufficient to check only his answer - usually a number, an expression or some similar entity. On the other hand there is no similar simple quantity which a student may type in to indicate that he can prove a proposition. What is required in this case is a supervisory sub-system for following through and validating a proof.

A similar supervisor for checking problem solutions would be both very useful and desirable - especially when incorrect answers are received. This is because in programmed instruction, the knowledge that a student's answer is wrong helps him discover where, why or how his solution has been incorrect. This is made possible by the smallness of a PI frame. However the solving of a problem is generally a large multi-step process and the knowledge of incorrectness of the answer often sheds very little light on the nature and location of the error. An answer can be wrong simply because of a trivial

* *Here we take a problem to be one whose solution yields an 'answer' in contrast to a proposition or a theorem whose truth has to be demonstrated in a proof and yields no 'answer'.*

error in a single solution step. By checking his solution as it is input into the system, a solution supervisor could reveal an error as it occurs, or help the student discover the nature of his error in some similar way.

We have argued briefly the need for a CAI sub-system for supervising mathematical solutions and proofs. While work along this line does not appear to have been done before, the idea of a supervisory system for some formal system is not new. Computer programs for checking proofs have been proposed before, although in a different applicational context. In [45], McCarthy discusses some possible uses of proof-checking programs in mathematics and systems engineering. The solution supervision concept is also implicit in Uhr [70] and Feurzeig and Papert [24]. Actually CAI systems which have interpretive/interactive compilers for teaching programming languages already incorporate a supervisory capacity.

At present very little appears to be known about supervising mathematical proofs. As a modest initial attempt at gaining some understanding of this problem, we have chosen a simple topic, elementary trigonometry, for investigation. This has led to the design considerations for a trigonometric proof supervisor (TPS), capable of :-

(a) checking and validating proofs of trigonometric identities

(b) assisting the student to construct his proofs.

In the actual application of TPS, the following situation is envisaged. The student is asked to prove a problem identity at his console. He inputs his proof step by step via a suitable input device and TPS examines each step as it is input, for correctness and for acceptability of step size. The student may seek help when in difficulty and receive some assistance in the form of a useful identity for substitution, a 'next step' to enable him to continue, suggestions about the method to employ in the proof or some other relevant information. When the student gives up, the system may either give him a proof or complete his proof from where he left off. When his proof is unduly long, the system may bring this to his attention. It should also be appreciated that the student is receiving very important aid from the fact that his steps are analysed immediately on input.

As an illustration, we offer the following hypothetical TPS-student interactions. The replies from TPS are assumed to be based on standard messages stored in the system.

TPS : Show that $\cot^4A-\csc^4A = 1-2*\csc^2A$

STUD : $\cot^4A - \csc^4A$

$$= \frac{\cos^4A}{\sin^4A} - \frac{1}{\sin^4A}$$

$$= \frac{\cos^4A-1}{\sin^4A}$$

$$= \frac{(\cos^2A-1)*(\cos^2A+1)}{\sin^4A}$$

$$= \frac{\sin^2A*(\cos^2A+1)}{\sin^4A}$$

TPS : Incorrect

STUD : *(Oops! Forgot the minus sign - erase last expression)*

$$= \frac{-\sin^2A*(\cos^2A+1)}{\sin^4A}$$

$$= \frac{-(\cos^2A+1)}{\sin^2A}$$

$$= \frac{-(\cos^2A+\sin^2A-\sin^2A+1)}{\sin^2A}$$

$$= \frac{-(2-\sin^2 A)}{\sin^2 A}$$

$$= 1 - \frac{2}{\sin^2 A}$$

$$= 1 - 2*\csc^2 A$$

QED

TPS : Correct. But look at the following proof:

$\cot^4 A-\csc^4 A$

$= (\csc^2 A-1)^2 -\csc A$

$= \csc^4 A-2*\csc^2 A+1-\csc^4 A$

$= 1-2*\csc^2 A$ <u>QED</u>

TPS : Show that $\dfrac{1+\tan A-\sec A}{\sec A+\tan A-1} \equiv \dfrac{1+\sec A-\tan A}{\sec A+\tan A+1}$

STUD : $\dfrac{1+\tan A-\sec A}{\sec A+\tan A-1}$

$$= \frac{1+\sin A/\cos A-1/\cos A}{1/\cos A+\sin A/\cos A-1}$$

$$= \frac{\dfrac{\cos A+\sin A-1}{\cos A}}{\dfrac{1+\sin A-\cos A}{\cos A}}$$

$$= \frac{\cos A+\sin A-1}{1+\sin A-\cos A}$$

#HELP

TPS : It is easier to prove this identity by showing that LS/RS≡1.

STUD : #HELP *(requesting further help)*

TPS : You need only use the identity: $\sec^2 A=\tan^2 A+1$

⋮ ⋮ ⋮

TPS : Prove: $(1-\cot A)*(1-\tan A)=2-\sec A*\csc A$

STUD : $(1-\cot A)*(1-\tan A)$

$= 2-\sec A*\csc A$

QED

TPS : This is not a proof at all!

STUD : *(erasing the previous expression - to renew proof)*

= 1 - cotA-tanA+cotA*tanA

#HELP

TPS : cotA$\equiv$1/tanA

STUD : 1-cotA-tanA+1

= 2-cotA-tanA

= 2-cscA*secA

TPS : Please show last step in greater detail.

STUD : *(erases last expression)*

= 2-cosA/sinA-sinA/cosA

#HELP

TPS : You will need cos^{2}A+sin^{2}A$\equiv$1, cscA$\equiv$1/sinA, secA$\equiv$1/cosA

STUD : *(continuing)*

$$= 2 - \frac{\cos^2 A+\sin^2 A}{\sin A.\cos A}$$

$$= 2 - \frac{1}{\sin A.\cos A}$$

= 2 - secA.cscA

QED

TPS : Good

TPS has only been partially implemented. A program called Super-2 for checking trigonometric proofs off-line has been written. It incorporates a model for small steps and uses a numeric technique for verifying the correctness of steps. The section of TPS for generating proofs and rendering assistance has not been implemented. However the practical and theoretical problems associated with it have been considered in some detail and are described in the thesis. A novel method for determining reference identities 'relevant' to a proof has been developed.

This work on proof supervision constitutes the major part of this thesis and may be regarded as a contribution towards the development of intelligent CAI systems.

To recapitulate, this thesis comprises essentially two parts :-

(i) the development of UACAIS, including ALFIE

(ii) an investigation into the problem of supervising trigonometric proofs.

Although they appear unrelated, it was the pursuance of (i) which led to the problem (ii).

CHAPTER II

THE UNIVERSITY OF ADELAIDE COMPUTER-ASSISTED INSTRUCTION SYSTEM

This chapter is essentially a revised and expanded version of the paper [55] by Perry and Lee. It describes the August 1969 version of the University of Adelaide Computer-Assisted Instruction System (UACAIS). The main emphasis is on external rather than internal features and only a general outline of the internal organisation will be given.

UACAIS is implemented on a Control Data 6400 computer [11]. The aims of the system are to provide :-

(a) a flexible medium for the presentation of teaching programs;

(b) authors and instructional researchers with useful statistics on the effectiveness of their teaching programs;

(c) a general experimental tool for the investigation of teaching and learning processes.

Basic System Features

UACAIS is based on existing equipment and its available software together with locally-developed interface and terminal hardware and its supporting software.

In the initial stages of the implementation, the CDC 6400 installed at the University had the minimum 32K of 60-bit central memory, a central processor (CP), and ten peripheral and control processors (PP), each of which is an autonomous computer with its own store of 4096 12-bit words and an unlimited access to the central memory by the use of read and write instructions. The 6400 also had twelve bi-directional 12-bit data channels, common to all

PPs and connected to the various peripheral equipment.

The main peripheral I/O devices were :-

(a) a dual-screen CRT display console with manual keyboard. Display modes are dot and character; characters comprising 26 alphabetics, 10 numerics and 11 specials, in three sizes;

(b) a mass storage discfile providing nominal storage of 500 million bits;

(c) six magnetic tape units, with three recording densities on half-inch tapes of up to 2400 feet;

(d) one card reader with a maximum speed of 1200 cpm;

(e) one card punch;

(f) two 800-lpm line-printers;

(g) one 10-inch CALCOMP incremental plotter.

The CDC 6400 has since been upgraded by the addition of :-

(a) 32K of 60-bit central memory.

(b) three remote terminals each with a CRT display console, a card reader and a line-printer.

(c) four removable disc-pack units.

(d) a 30-inch CALCOMP plotter.

Some of the peripheral I/O devices were not required in the CAI system, although the card reader and the line-printers were used for certain off-line CAI applications such as course preparation. The remote CRT terminals and the disc-packs were acquired too late to be incorporated into the system, but should prove extremely important for future versions of UACAIS. The CRTs would be a useful replacement for the noisier and slower typewriters now in use. The use of removable disc-packs would enable a virtually unlimited number of courses to be available on-line very conveniently and rapidly. At present, required courses have first to be loaded onto the discfile from tapes.

The major peripheral device in UACAIS is the discfile which serves as a random access, high-speed transfer, mass storage unit. It is used in the CAI system in three main ways:-

(a) The complete operating system is copied onto the discfile from tape as it is loaded. This allows the system to be recovered without a complete reload in the event of a hang-up. It also facilitates rapid access to PP subprograms when they are needed.

(b) All teaching programs are stored on the discfile when needed. A system of directories gives access to any course segment by name.

(c) The disc is used as a backing store for student operating information. Each console is allocated the required amount of space by the system.

The implementation of the CAI system represents considerable programming effort.* The entire software has been developed by P.G. Perry and the author, and has been coded completely in the central and the peripheral assembly languages [12] of the CDC 6400. The size of the system precludes a detailed description of the programs in the system; only an outline of the more important ones will be given.

* *As an indication of this, the system took more than two years to develop. In terms of card volume, the software, including the compiler for ALFIE, occupies about 25,000 cards.*

The system includes the monitor, the display and the disc driver programs, each residing in a separate PP. The remaining seven PPs form a pool on which the monitor can draw, for the execution of any of a suite of utility subprograms, including equipment drivers and general routines for performing jobs which cannot be carried out conveniently by the CP. The main programs in the system are outlined below.

MTR MTR is the monitor program and resides in PP-0. It controls the assignment and release of pool PPs, data channels, disc space and peripheral equipment, and coordinates their activities. It also maintains a real-time clock and services CP and PP requests for the loading and execution of utility programs.

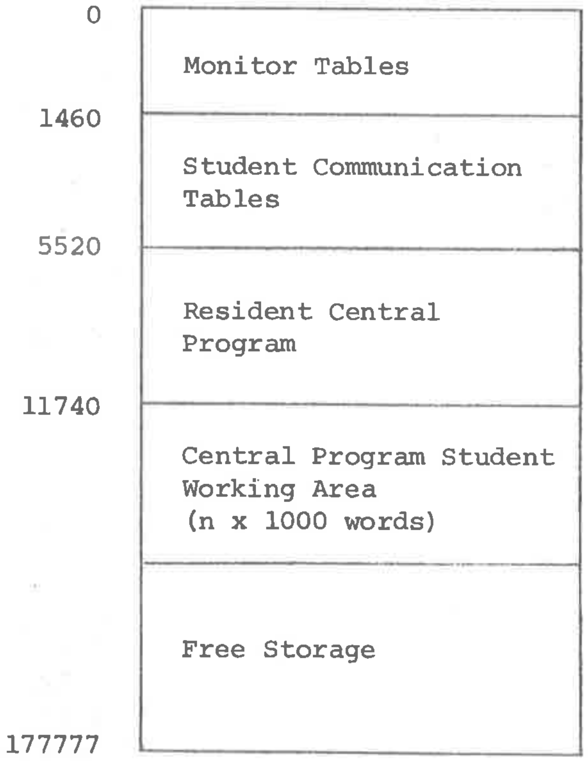
RDD This is the disc driver program. It resides in PP-9 and communicates with the discfile via channel-00. RDD queues and processes all access to the discfile. PP requests to RDD are for data to be read from the disc, for data to be written on the disc, for the loading of PP overlays and for the release of disc tracks. CP requests concern mainly the loading of course segments on central memory, and the transfer to and from the disc, of student record information. RDD maintains a request stack, 1 for PP and up to 8 for CP, servicing them on the basis of their required disc access times, rather than on the first-come-first-served principle. Thus for two competing requests, the one entailing less track select and disc group-head switching time, will be serviced first. The reason for adopting such a priority system is to ensure optimum throughput between the disc and central memory.

DSP Resident in PP-1, DSP is the system display program, driving the 6400 CRT scope via channel-10. It monitors system status and activity and permits operator control of the system from the keyboard. Among the operations it can handle are the activation and deactivation of channels, the assignment and the switching on and off of equipment, the modification of central memory and the despatching of equipment function codes. These control functions of DSP had proved invaluable in the debugging of the CAI system - during the various phases of its implementation. The display console has also been used to simulate the student terminals before they became operational.

RCP RCP is the major resident central memory program. It contains various monitor and student communication tables as well as student working areas. Figure 2.1 is an outline of the central memory in RCP and gives an insight into the internal structure of the program.

RCP is also responsible for the presentation of courses. The system was designed to have eight student areas to be time-shared by up to 512 students through swapping. Course segments not immediately required in core are rolled out to the disc, while segments that require processing are loaded from the disc. Since we do not expect to have more than a few dozen consoles for some considerable time, this version of UACAIS has been modified to have arbitrarily, forty student areas. This means that for up to a maximum of forty consoles, no swapping of student working records will be necessary. RCP accepts and processes, using an interpreter subroutine, the instructions of the teaching program. It analyses student answers, performs the appropriate branching, records on tape student responses and performance data when so directed, and also handles the sign-on and sign-off protocol. RCP also maintains a desk calculator facility for servicing simple computational needs. The student uses this facility through the #COMPUTE command; the syntax for the expressions to be evaluated resembles the one for the register expression in ALFIE. (See Appendix A)

The system software also contains many other programs. These include those for driving the login tape, the logout tape and the student response-performance tape. Also included are the remote console/interface driver-scheduler and the course and system loaders.


0
Monitor Tables
1460
Student Communication Tables
5520
Resident Central Program
11740
Central Program Student Working Area (n x 1000 words)
Free Storage
177777

(The memory locations refer to a recent version of RCP; n is the number of student areas.)

Figure 2.1a Central Memory Utilisation

0	Pointers and Channel Status Table	Locn 0 is set to zero; contains pointers to library directory, display buffer area, status of each of 12 data channels.
10	Message Areas for PP-1 to PP-9	8 word block per PP. Each block has a 6-word message buffer area in which PP and monitor can insert a message.
120	PP Control Area	4 word block per PP for PP to communicate with monitor regarding request to execute programs etc,
170	Equipment Status Table	Status of equipment: disc file, display console, interface, card punch, card reader, plotter, printer and MTU plus dummy entries.
210	Display Buffer Area	For the insertion of message for A-display by DSP.
554	File name and Program name tables (Library Directory)	Directory of system routines like CLR, BKP, EXJ, SIM, ... and directory of courses loaded in the system.
1400	Link Status Table	Status of each link (active, inactive).
1420	Monitor Request Table	Area which MTR examines for systems requests, inserts/updates clock, and CP disc I/O requests.
1440	Program Exchange Jump Area	Exchange Jump Package for Idling program.
1460		

Figure 2.1b Monitor Tables

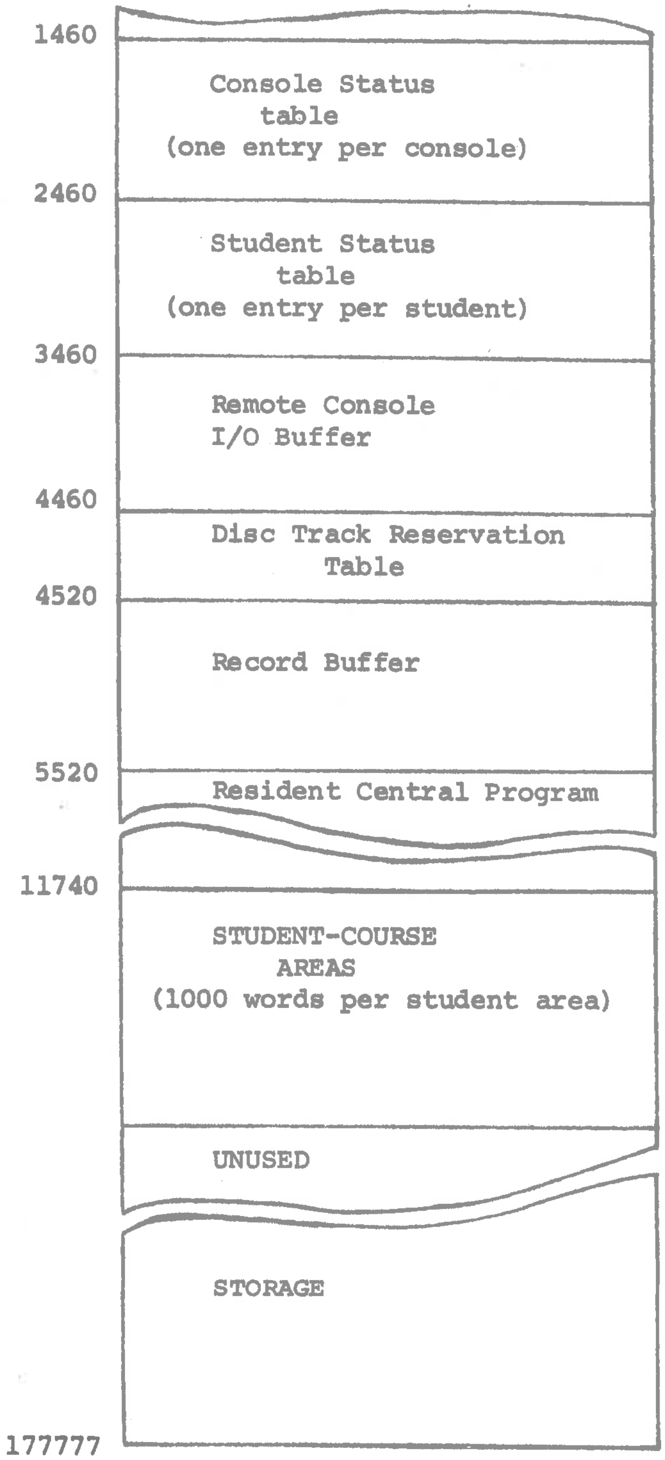

1460
Console Status table (one entry per console)
2460
Student Status table (one entry per student)
3460
Remote Console I/O Buffer
4460
Disc Track Reservation Table
4520
Record Buffer
5520
Resident Central Program
11740
STUDENT-COURSE AREAS (1000 words per student area)
UNUSED
STORAGE
177777

Figure 2.1c Other Tables

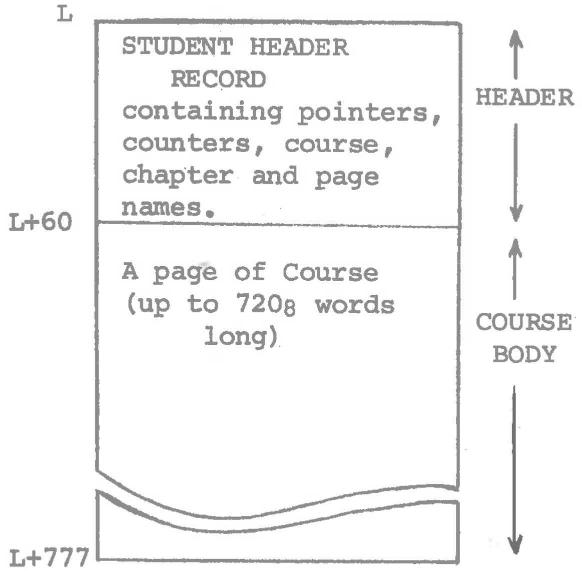

L
STUDENT HEADER RECORD containing pointers, counters, course, chapter and page names.
HEADER
L+60
A page of Course (up to 720_8 words long)
COURSE BODY
L+777

Figure 2.1d A Student Area

Note: All numbers in octal

The CAI Consoles

The CAI consoles and their interface with the computer have been designed and built within the University [56]. The interface is designed to have up to 64 external links, two of which have now been implemented. Each link is capable of controlling 8 consoles, giving a maximum of 512 consoles.

A console is a unit which can contain up to seven pieces of equipment, only one of which can be operating at a time. The consoles available at present have only a typewriter (for input and output) but it is hoped that future consoles will incorporate a slide projector, a computer-controlled tape-recorder, for the playback of pre-recorded messages only, and a CRT display in lieu of the typewriter.

The consoles work on a basic seven bit code, with an eighth bit for indicating control functions, such as status response and unit select, and three bits for the console address. The seven bit format allows for an implicit case bit in the typewriter code (see Appendix A).

All consoles are run by a single driver subprogram which schedules the console to be processed on each link at each cycle and performs certain automatic functions such as deactivating faulty consoles and links.

Although only four consoles have been built, the system is theoretically capable of processing the full complement of 512 consoles.

Input messages are terminated by a carriage return (c/r) and are not recognised by the system until this is received. The student can delete single characters by backspacing, and erase a whole message by using a character reserved for this purpose. If a message exceeds the arbitrary limit of 80 characters, it is automatically terminated and a c/r sent back.

When messages from the student are being interpreted, the first test is for the first character being "#" indicating a command word. If not the system next checks if an answer to a question is expected. Failing this the message is rejected.

Operating Features

The general layout of the configuration of UACAIS is given in Figure 2.2.

Although a powerful computer could be expected to time-share CAI with normal job processing, as is being done elsewhere, this system has been designed as a dedicated system. The modifications needed in the standard operating system of the CDC 6400, SCOPE, would have taken at least as long to make as the preparation of an independent system from the start. Also a more compelling reason is that there would not have been enough core space left in the then minimum 6400 configuration, to run background jobs as well. Furthermore, such a design would have been even more sensitive to revisions in the operating system than to those in the independent system; it should be appreciated that SCOPE was undergoing numerous revisions at the time.

In the event of a software hang-up, the system cannot at present be recovered, as it is an experimental rather than a production version. However, the loader saves a copy of the system on the discfile at load time and the major tables can be replaced from central memory. Even if the tables in memory were lost the portion of the disc which is used as a backing store would contain the positions of all students from only a few seconds before;

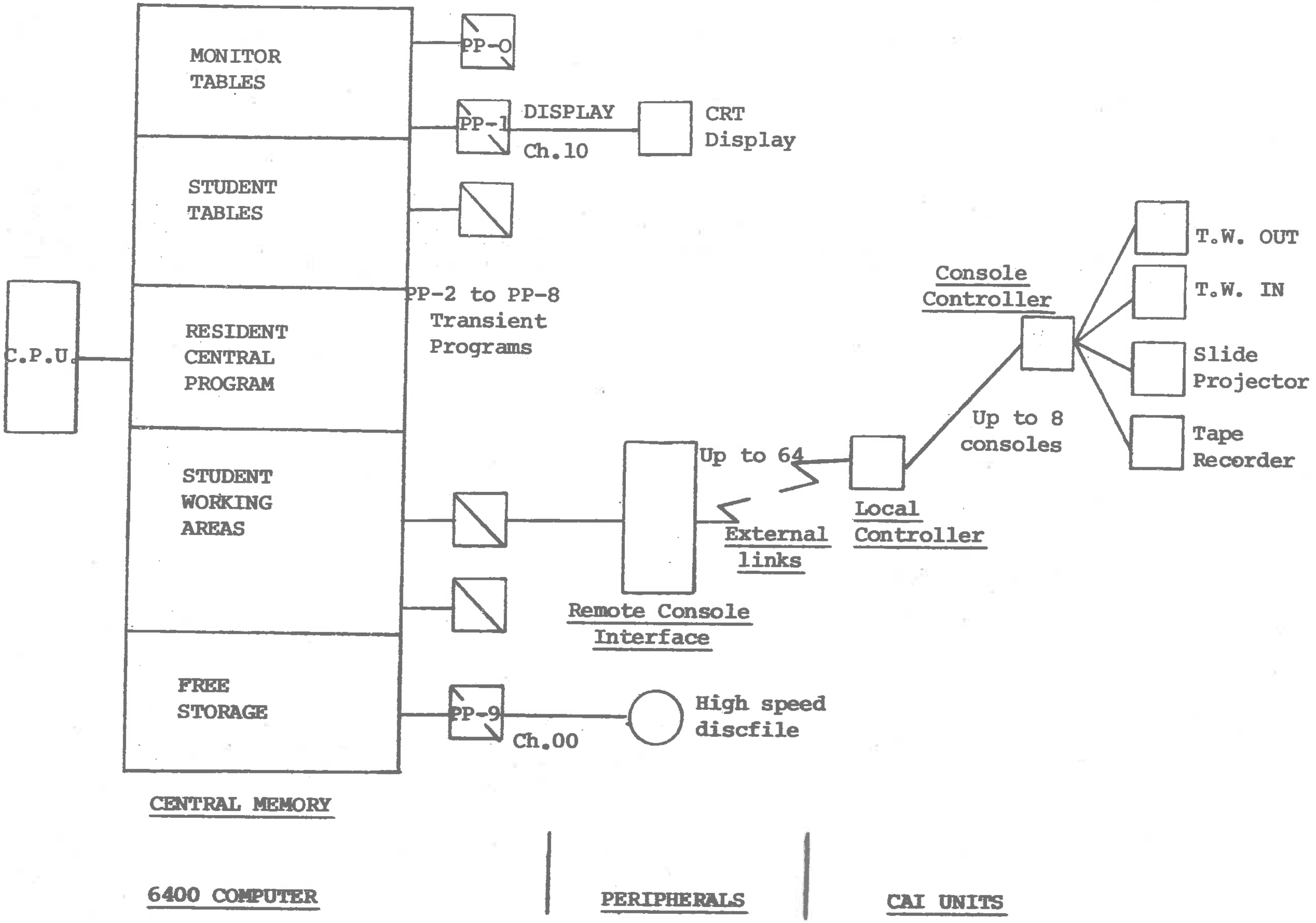


Figure 2.2 UACAIS Configuration

only a list of the active consoles would be needed to effect a recovery.

Student Working Record Format

Some further insight into the internal arrangement of the system can be gained from the format of the information used to process an individual student. (See figure 2.1d.) This is best described in three separate phases:

(a) At Sign-on Time

When the student types in his registration code, a record with that code name is searched for on the login tape. This record should contain the student's name, his mode (whether he is a student, author or a proctor), the contents of various counters and a list of all the courses for which he is enrolled. Each item in this list includes the course, chapter and page names, the total time spent on the course so far, and the content of counters for the course.

(b) During Operation

The student working record during operation can be divided into two main parts:

1. Header, containing the student's name, various counters and flags, the course name and the position in the course. This section is derived initially from the information found in (a) above, and is altered as the student progresses.
2. Body, which is a copy of the current segment of the course. By keeping this copy with the information concerning the particular student, it is not necessary to search for it each time the student is processed.

With a total record length of 512 60-bit words and the header taking up 48 words, a maximum segment length of 464 words is left. A full segment containing only textual data and the necessary data transmission commands could contain up to 2000 characters, and these would take about 2½ minutes to present. During the swapping to and from the disc of the student working record, the full record need be written on the disc only once; subsequent transfers to the disc, for a given page of the course, need involve only the first one or more disc segments (64 words each) of the course required to cover the header. This is because only the header portion is variable during operation, and once a copy is on the disc, only the header segments need be altered. (Disc transfers are carried out in segments.) Since in a fully implemented 512-console system a great deal of disc swapping operation can be expected, valuable time would be saved by not retransferring the non-varying segments of the record to the disc.

(c) After Sign-off

When the student signs-off the record header information is dumped on another tape, the logout tape, in its final form. The information it now contains is the student's code, time spent during the session, the new counter values, the name of the course and the new position in it. This information is used to produce an updated master record (or login) tape so that when the student next signs on, the new course position is known. However this updating is made only after the CAI system has finished running; by using disc packs instead of two magnetic tapes in future, immediate updating of the student master record can be made.

The Author and the Student's Views of the System

To the author, the system is essentially a means of presenting a course. This he can achieve through the author language ALFIE. At present courses are prepared off-line on cards and assembled by the ALFIE course compiler onto tape.

A student signs-on at the console by typing in his registration code. He then selects from among the courses for which he is enrolled the one he wishes to pursue. This course is presented to him according to the author's instructions: he may be offered informative material explaining a concept, then presented with questions to probe his grasp of the concept and additional, remedial, material if necessary.

The student has some control over the presentation by use of command words, such as LOGOUT, STOP, GO, LOCATE, COMPUTE and HELP. These may be abbreviated to any extent by dropping the end characters. For instance, HELP could be shortened to HEL, HE or simply H; the requirement that the abbreviation should uniquely identify the command is obvious - failure to do this will cause the use of the first command word in the table which could satisfy the message.

The student uses LOGOUT to sign off from the session. The STOP suspends the presentation of the course, until a GO is received. The GO also overrides any system/course interrupts, causing the lesson to continue. The

COMPUTE is used for simple calculations; the student supplies the expressions to be evaluated. HELP causes the system to present any cueing material that may be associated with the question presented. LOCATE enables the student to jump to any course segment by specifying its chapter and page names as necessary.

The student may also be directed to a proctor at another console and they can converse through the medium of the typewriter. This feature has not been fully implemented in the current version.

Recovery of Statistics:

The CAI system has a wealth of statistics available to it and many of these may be recorded quite simply. The selection of statistics will be largely the responsibility of the authors, but typical data would include the time to reach a certain position in the course, the latency time for a response, the number of attempts at a question, and unanticipated answers.

The data is recorded in raw form on tape for off-line processing where it can be summarised in various forms, both from the point of view of the course and the individual students.

The current version of the system is operating with the four typewriter consoles so far installed. Several course segments have been developed to debug the system and to test out various language features. Most of the system features that have been described have been implemented and they are now working correctly.

Our experience with the present version indicates that the system can be expanded to support additional consoles without difficulty. Only a slight expansion to certain hardware driver routines would be required. Also additional commands can be added in a modular fashion without adversely affecting the overall system.

Although the CAI system described cannot claim to compete with overseas systems in terms of size and complexity, for systems are now being built with several thousand consoles, we feel that apart from being the first experiment with large scale CAI in Australia it does have some novel features.

The course preparation language, ALFIE, is naturally enough unique. While this is not necessarily desirable in terms of standardisation, some of its features, especially the control which the author has over the presentation of cueing materials are, we believe, new.

Apart from the provision of more consoles, and of more sophisticated design, some of the hardware additions to the computer which would most facilitate CAI operation are:

(a) Extended Core Storage: This is a fast access core backing store, addressable by block transfers of any length. This could be used to reduce the memory space used by programs and tables, allowing the use of more extensive programming features than would otherwise be possible. The use of E.C.S. as a backing store would also reduce the load on the discfile and significantly cut down disc swapping time.

(b) Disc Packs: These are fast random access, replaceable storage devices which could be used for storing CAI courses, thereby eliminating the need for copying them from tape to the discfile prior to use. Faster sign-on would also be possible with the student master records stored on disc packs and an updated record could be produced immediately on sign-off.

The development of the preliminary version of UACAIS has also led to ideas for useful additions to the software. This includes expansion and improvements to the author language, the development of an on-line course compilation, testing and revision capability. The set of keyboard commands should be expanded. Application packages for extracting student performance statistics should be developed; preferably these should be accessible from the keyboard. Also basic statistics like time spent in the current session, performance level and so on, should be immediately available to the student when he signs-off.

CHAPTER III

THE AUTHOR LANGUAGE ALFIE

This chapter describes and discusses the main features of ALFIE, the author language for the Adelaide CAI system. A detailed specification of the language is given in Appendix A.

At present courses may be prepared only on cards. They are compiled onto magnetic tape for subsequent use under UACAIS. During CAI operation, required courses are loaded from tape onto the discfile ready for use. Courses are presented to the students at the remote consoles on their typewriters. The students communicate with the CAI system via the typewriter keyboard. They progress independently of one another, each proceeding at his own pace. Different students may be doing different courses, or different segments of a common course simultaneously. This independent presentation of courses is done automatically by the operating system. The author, in preparing a course, may imagine that he is teaching only one student, even though in reality, a single course may be used by more than one student during a CAI session.

While making ALFIE as flexible and powerful as we can, we have also attempted to make it easy to learn and convenient to use. This we believe we have achieved to a large extent, judging from the speed with which beginners have learnt to program courses in the language.

Compiling ALFIE Programs

A course on cards is actually compiled in two stages. The first stage is performed by LKC, a preprocessor program for translating the course in the card format into a common intermediate code called CONCODE. The actual compiler (or assembler) ALF, then converts the course in CONCODE into the object codes. It can be seen that we have in this way, made provisions for the implementation of other course-preparation media. All that is required for a new medium is a corresponding preprocessor for converting a course in that medium into CONCODE. Figure A.3 (Appendix A) illustrates the approach just discussed.

In the compilation, two listings are also produced:-

(1) the source card listing produced by LKC (see Figure A.1); and

(2) the assembly listing produced by ALF (see Figure A.2).

The preprocessor LKC distinguishes between five kinds of course cards by examining their first columns. They are:-

(1) Listing Control Card (LCC): This has a 2-8 (D̄ on IBM 029) punch on column 1. It is used for producing and suppressing source card listing.

(2) Comment Card (CC): A CC is characterised by an '*' on column 1. The characters on the remaining 79 columns are treated as comments, and appear on the source card and assembly listings.

(3) Edit Card (EC): Carries a '=' on column 1 and 'EDIT' on the next four columns. The EC is used as an aid for line truncation and text justification.

(4) Keyword Card (KWC): A KWC has a '$' or '→' (w̄ on IBM 029) on column 1, or a 'c' if it is a continuation KWC. KWCs are for carrying instructions, each of which is identified by a keyword.

(5) Textbody Card (TBC): A TBC is normally one with column 1 blank; however any card which is not an LCC, CC, EC or KWC is a TBC. TBCs are used for coding typewriter texts streams. A text stream is a string of

typewriter codes to be output on the typewriter, and consists of a set of successive TBCs bonded by two KWCs.

The characters available on the console typewriters are the usual:-

(1) Alphabetic: A B C ... X Y Z a b c ... x y z

(2) Numeric: 1 2 3 ... 7 8 9 0

(3) Special: + - * / ( ) $ = blank , . [ ] # : ' °
& _ " % @ ¢ ? ! ;

Available control functions are: carriage return, index*, tab, backspace and the black and red ribbon select.

Textual material must be composed from this set of characters and functions only. Since many of these are not available on a standard card punch, a card code for representing them is required. Table A.1 gives the list of all available typewriter characters and functions, their card codes based on the IBM 029 punch, as well as other codes.

A course in ALFIE is essentially a sequence of directives or instructions and texts. The directives are KWCs while texts are made up of TBCs; CCs, ECs and LCCs may be included as comments and to control source and listing and text editing.

A course is organised into an arbitrary number of named chapters, each of which may have up to sixtyfour named pages. Figure A.4 in Appendix A shows schematically the organisation of a typical course into chapters and pages. The arrows indicate the various kinds of branching permitted in ALFIE.

*Depressing the index key positions the typewriter carriage on the same column on the next line.

Each course directive is associated with a keyword and belongs to one of the three categories:

(1) listing control

(2) course organisational

(3) command.

The listing control directives are the LIST and NOLIST instructions. These are respectively directives to the assembler to produce and to suppress assembly listing. By placing the LIST and NOLIST cards appropriately in the deck, a selective listing of a course is obtained.

The organisational instructions are the COURSE, CHAPTER, PAGE and END cards. They direct the assembler in the organisation of a course into chapters and pages. Every course begins with a COURSE card, which initiates and names it. Each chapter is started and named by a CHAPTER card, which also terminates the previous page and chapter if any. Each page begins with a PAGE card which names it, and terminates the previous page if any. The END card terminates the current page, chapter and course.

The command directives enable an author to present texts, ask questions, accept and analyse student answers, take the appropriate action on the basis of such analyses, make either conditional or unconditional branches to almost any point in a course, perform register arithmetic (q.v.), sequence and control the presentation of a course and record student responses.

Command directives may be labelled. A label is any alphanumeric string not exceeding six characters, followed by a colon as in '$L2:CUE.'. A label may occur by itself, as in '$LX3:' and serves to define a page location.

The '→' used instead of the '$' on column 1 of a KWC defines an implicit label (q.v.). All non-implicit labels in a page must be unique.

A command directs the course interpreter to perform certain operations. Commands are normally executed one by one down the page, continuing on the next page. This strictly sequential execution is interrupted by branch instructions. Certain critical (q.v.) commands like the CUE, also have special meaning which will be explained later.

When a command is executed, the next one will be executed immediately unless the command contains a delay - whether explicit or implicit. A delay is a period that must elapse before the next command is executed, but it can be overriden by an interrupt from the student, typing in an answer or a '#GO'.

The various commands in ALFIE will now be briefly described. The reader may refer to Appendix A for further details.

The TEXT command instructs the system to present the immediately accompanying text stream. A TEXT without an accompanying text stream is ignored. However a text stream need not be preceded by a TEXT card, in which case a '$TEXT.' card is assumed. A delay of up to 999 seconds can be specified via the T-parameter. If not specified, as in '$TEXT.', a 1 second delay is assumed, while T=* implies an infinite delay.

The SIGNOFF command signs off the student, who cannot then resume his course until he signs on again. The student can also sign himself off by typing in '#LOGOUT'.

The COMPLETED command directs the system to sign-off the student, and informs it that he has finished his course. There may be more than one COMPLETED and SIGNOFF in a course.

The CUE, ANSWER, ENDANS, WAITCUE and GIVECUE commands may be used only in a problem environment, a rather loose concept which will be explained later.

The CUE is the command which directs the system to accept and analyse an answer from the student. The answer should be entered within the permitted delay; otherwise the system will execute the next command. The delay can be optionally specified by the T-parameter; if this parameter is absent as in '$CUE.', then an infinite delay is assumed. A repetition factor is also allowed in a CUE, e.g. '$CUE,4(T=10)' is equivalent to four successive '$CUE(T=10)' cards. The actual interpreter action on the CUE will be explained in the section on the problem block.

The ANSWER and ENDANS commands enable the author to specify one or more answers against which the student's responses are to be compared. The M-parameter specifies the answer recognition mode, the default mode being M=3. At present only the modes 0,1,2, ... 14 are implemented. These cover the following kinds of matches: catch-all (in which any answer matches), exact string match without prior editing of student answer, after removal of blanks or some specifiable class of characters, truncated string match, unordered list match, numeric match with specifiable numeric limits and algebraic expression match.

For each problem block, the author supplies a sequence of answers, the last of which must be an ENDANS. Thus if there is only one answer specified, it must be an ENDANS. Only the ENDANS can have M=O, the catch-all mode.

An answer command also carries a C-parameter which can take any of the values 0,1,2, ... 63; the default value is O. The C-parameter will be included in a recording of a response match with the answer. It is useful as a means of categorising answers in the course - for expediting subsequent analysis of recordings of student performance data.

Although the answer recognition capabilities available are adequate in meeting a wide range of needs, there are many answer processing needs that have still to be catered for. The modes so far implemented have been chosen to meet the immediate needs of the current phase of the CAI project. Three examples of the answer command are:-

(1) $ANSWER(M=2,C=3)*PERMUTATION*

(2) $ANSWER(C=4)=WARM=FRIENDLY=GENIAL=

(3) $EN~~S~~DANS.*UBIQUITOUS*

The WAITCUE and GIVECUE commands carry no parameters. The WAITCUE instructs the system to re-execute the last-executed cue in the problem block. The GIVECUE differs from WAITCUE in that it instructs the system to execute the command immediately following the last-executed cue; this is usually some cueing text.

The TIME instruction, e.g. '$TIME,2XX.', instructs the system to record the current time, at which the present course position, specified by the name in the instruction, is reached. This is useful in investigating the time taken by different students to reach various course positions.

The DELAY command instructs the system to pause for a specified period.

Thus '$DELAY(T=100)' is a request to wait for 100 seconds to elapse before resuming course execution, unless there is a student-initiated interrupt.

The WAIT command instructs the system to output in red the character overprint 'W' on the typewriter as a signal to the student to press the carriage return key when he is ready to resume. The WAIT may not be imbedded in a problem block.

The PROBLEM card carries two parameters - a problem block name and a recording or R-parameter, enabling the author to name a problem block, and to specify the recording mode. Student responses and other performance data will be recorded according to the associated PROBLEM card.

There are three kinds of unconditional branching :-

(1) Intra-page e.g. $GOTO,L2.

(2) Inter-page but intra chapter e.g. $GOTO,PAGE(TX2)

(3) Intra-chapter e.g. GOTO,CHAPTER(P2),PAGE(U26).

The intra-page branch enables the author to specify a transfer to anywhere in a page by specifying the label of the destination. The special case '$GOTO,→' is a branch to the next implicit label, i.e. '→', below in the page. The other branch instructions permit transfer to any named page in the same chapter, or the first or any named page of any other named chapter.

Register Arithmetic

The author is provided with fifteen registers (or counters), A,B,C,... M,N,O. Each register is used to hold a 60-bit real number. By means of register statements (see Appendix A), the author can set register values and perform floating-point arithmetic on numeric constants and register values.

All registers are zeroed at the beginning of a course. Registers may be used to keep track of individual student progress. An example of the register statement is: $A=B-4*F/G.

Conditional branching based on register values are allowed. The general form of the conditional branch is:- IF(relational expression)Br where 'Br' stands for any unconditional branch statement. The relational expression, which can be true or false, is a statement about relative register values. The 'Br' is taken only if the relational expression is true. Examples of conditional branching are:-

(1) $IF(B.LE.G)GOTO,K44.

(2) $IF(C.GT.12)GOTO,PAGE(JA3).

We should also mention here that some keywords have equivalent alternatives. Thus all the following keywords in a parenthetical group are equivalent: (CUE,EXPLAIN,EXP); (PROBLEM,BLOCK,PRB,BLK); (ANSWER,ANS); (ENDANS,ENS).

We have now covered briefly all the commands in ALFIE. The important concept of the problem block will now be described and this is quite vital in the understanding of the language.

The Problem Block

The cue, answer, WAITCUE and GIVECUE commands are critical instructions in that they may be used only in the context of a problem environment, and not arbitrarily. Here, 'cue' refers to CUE,EXPLAIN or EXP; 'answer' to any of ANSWER,ANS, ENDANS and ENS.

A cue sequence is a set of one or more cues, possibly interspersed with text and non-critical commands. It has the form:

```
     $CUE---
        :
        :
     $CUE---
        :
        :
     $CUE---
```

```
e.g. $CUE,2(T=10)
        It begins with a 'C'
     $CUE(T=15)
     $TIME,AG2.
        Try again.
     $EXP.
        The answer is 'crisis'.
```

An *answer sequence* is either an ENDANS or a set of answers (not ENDANS) terminated by an ENDANS. The sequence may be interspersed with text and other instructions, but excluding cues and the WAIT.

```
e.g. $ANS(M=2)*COMMON*
        Nearly correct.  Try again.
     $WAITCUE.
     $ENS(M=2)*ORDINARY*
        Correct.
```

A cue sequence followed by an answer sequence constitutes a *problem block*. The cue sequence and the answer sequence have no independent standing; every cue sequence must be accompanied by an answer sequence.

Any answer sequence not immediately preceded by an unconditional branch, will have a 'GOTO,→' automatically inserted there. Thus

```
     :                                  :
   $CUE---                            $CUE---
     :      is equivalent to            :
   $ANS---                            $GOTO,→
     :                                $ANS---
                                        :
```

The problem block is a paradigm for posing a question to the student and for accepting and analysing his responses against the author answers. The first cue of a problem block is normally preceded immediately by the text of a question or problem that is to be answered.

When a cue is encountered, the system outputs on the typewriter, on a new line, a red question mark '?'. This indicates to the student that an answer is expected. The system also sets a pointer to the associated answer sequence; to the first answer in fact. The student's answer must be received before the delay in the cue expires; otherwise the student forfeits his chance to answer, the pointer is cleared, and the next command below the cue is executed. If no T-parameter is set in the cue, the student may take as much time as he likes to enter his answer.

On receiving an answer, the system next executes the command indicated by the pointer; in this case it is the first answer command of the problem block. The answer is checked against the author answer in this command. If the answers match, then execution of the course resumes from immediately after this answer command; any subsequent answer command executed will be treated as matching (see later). If the answers do not match, then the pointer is reset to the next answer command, if any, and answer comparison is again made. The action taken by the system on the outcome of this comparison described above is repeated. Thus each answer in the problem block is executed in turn until either a successful match has been made or all answer commands have been exhausted. In this latter case the student's answer does not match any of the author answers, and the system clears the pointer, types out "Incorrect" and continues the course from immediately after the cue.

What usually follows the cue is some prompting or explanatory text - hence the keyword CUE and its equivalent EXPLAIN.

When an answer command is executed, the student's answer is compared with the author's answer(s), under the specified mode. The outcome of this comparison is either a (successful) match or a mismatch. If there is no student answer to be compared, e.g. when his answer has already been matched with a preceding answer in the same answer sequence, then a match with the answer(s) is assumed. If there is a mismatch, then the answer pointer is reset to the next answer command; this next answer is then executed. If no next answer exists, then the course resumes from the cue which has led to the execution of this answer command. If there is a match, then the course continues from immediately after the answer command just executed.

A problem block may not be split over two or more pages. When an author-defined page exceeds the page limit, the assembler will attempt to spread it over two or more pages without splitting any problem block.

Specimen Course and Interaction:

Figure 3.1a is the source listing of a course which has only one chapter and one page. The course is named 'SPECIMEN', the chapter 'ONE' and the page 'RR4'. The 'non-standard' characters in the texts are mainly codes for indicating upper and lower cases in alphabetics and for terminating text cards. (See Table A.1.) They appear in a more natural form in Figure 3.1b which is an assembly listing of the page. The assembly listing of text simulates as far as the line printer permits, its appearance on the typewriter; thus while spacing, backspace, tab etc. are reflected exactly, this listing

```
     PAGE,RR4.                                              PAGE  01  -  CHAPTER  ONE   RR4 0001
     *                                                                                  RR4 0001.001
     *      THIS IS A DEMONSTRATION PAGE                                                RR4 0001.002
     *                                                                                  RR4 0001.003
)01  ↱                                                                                  RR4 0002
)01  PRB,Q22(R=8)                                                                       RR4 0003
               WHICH ELEMENT HAS THE ATOMIC SYMBOL ≠PT≠¬                                RR4 0004
)13  CUE.                                                                               RR4 0005
               PL-------. TRY AGAIN.                                                    RR4 0006
)21  CUE(T=20)                                                                          RR4 0007
               PT IS A METAL, MAINLY USED AS AN INDUSTRIAL CATALYST.                    RR4 0008
)35  EXP(T=10)                                                                          RR4 0009
               THE ANSWER IS: PLATINUM                                                  RR4 0010
)43  GOTO,PJ.                                                                           RR4 0011
)44  ANS(M=2)*PLATINUM*                                                                 RR4 0012
               GOOD. PLATINUM IS CORRECT.                                               RR4 0013
)54  GOTO,PJ.                                                                           RR4 0014
)55  ANS(M=2)*PLUTONIUM*                                                                RR4 0015
               PLUTONIUM IS ≠PU≠.                                                       RR4 0016
)63  GIVECUE.                                                                           RR4 0017
)64  ENS(M=2)$PHOSPHORUS$                                                               RR4 0018
               PHOSPHORUS IS ≠P≠.                                                       RR4 0019
)73  GIVECUE.                                                                           RR4 0020
)74  ↱PJ:TEXT.                                                                          RR4 0021
               ABCD........                                                             RR4 0022
)79  END                                                                                RR4 0023

       ------          385 UNUSED WORDS FOR ABOVE PAGE
```

3.1b Assembly Listing

```
$COURSE,SPECIMEN.                                                          CARD  0001
$CHAPTER,ONE.                                                              CARD  0002
$PAGE,RR4.                                                                 CARD  0003
↱                                                                          CARD  0004
*                                                                          CARD  0005
*      THIS IS A DEMONSTRATION PAGE                                        CARD  0006
*                                                                          CARD  0007
$PRB,Q22(R=8)                                                              CARD  0008
 ↑W≡HICH ELEMENT HAS THE ATOMIC SYMBOL ≠↑P≡T≠2^^                           CARD  0009
$CUE.                                                                      CARD  0010
 ↑P≡L^7-. ↑T≡RY AGAIN.^^                                                   CARD  0011
$CUE(T=20)                                                                 CARD  0012
 ↑P≡T IS A METAL, MAINLY USED AS AN INDUSTRIAL CATALYST.^^                 CARD  0013
$EXP(T=10)                                                                 CARD  0014
 ↑T≡HE ANSWER IS: ↑PLATINUM^^                                              CARD  0015
$GOTO,PJ.                                                                  CARD  0016
$ANS(M=2)*PLATINUM*                                                        CARD  0017
 ↑G≡OOD. ↑P≡LATINUM IS CORRECT.^^                                          CARD  0018
$GOTO,PJ.                                                                  CARD  0019
$ANS(M=2)*PLUTONIUM*                                                       CARD  0020
 ↑P≡LUTONIUM IS ≠↑P≡U≠.^^                                                  CARD  0021
$GIVECUE.                                                                  CARD  0022
$ENS(M=2)$PHOSPHORUS$                                                      CARD  0023
 ↑P≡HOSPHORUS IS ≠↑P≠.^^                                                   CARD  0024
$GIVECUE.                                                                  CARD  0025
↱PJ:TEXT.                                                                  CARD  0026
 ↑ABCD........^^                                                           CARD  0027
$END                                                                       CARD  0028
```

3.1a Source Listing

Figure 3.1 Sample Listing from ALFIE

cannot show the effects of case changes in alphabetics, and of ribbon changes.

Note the three comment cards with an '*' on column 1. The page location into which each instruction is compiled is also shown in the assembly listing. At the bottom of this listing, the number of unused words in the page is printed.

Page RR4 is essentially a problem block and the following are three student interactions with it. Every student response is terminated with a carriage return, but this is not shown. Student type-ins are given here in italics.

(a) Which element has the atomic symbol 'Pt'?
?PLATINUM
Good. Platinum is correct.
ABCD....

(b) Which element has the atomic symbol 'Pt'?
?POTASSIUM
Incorrect.
Pl___. Try again.
? (Student fails to respond within 20 seconds)
Pt is a metal, often used as an industrial catalyst.
?PLATINUM
Good. Platinum is correct.
ABCD....

(c) Which element has the atomic symbol 'Pt'?
?PHOSPHORUS
Phosphorous is 'P'.
Pl___. Try again.
?PLUTONIUM
Plutonium is 'Pu'.
Pt is a metal, mainly used as an industrial catalyst.
?PLATTINUM
Incorrect
The answer is: PLATINUM.

Cueing Feature of ALFIE

Unlike most CAI languages, ALFIE is a cue-oriented language, especially suited to the presentation of hints or cues. It is an easy matter to program a problem which allows the student say three attempts at it, with a hint to follow each of the first two unsuccessful attempts. It is of course also possible to do the same thing in other CAI languages, but usually with less convenience.

One potentially useful application of the cueing feature of ALFIE is in the investigation of guided discovery learning. Suppose it is desired to teach a concept, which is small in some sense. A common approach in PI is to present a unit U of information, usually a paragraph or two, to describe or explain the concept. An evaluative question Q is then presented to assess the student's understanding. The student may be allowed several attempts, with or without hints. If the student fails to answer correctly, he will be given the correct answer. This conventional approach to instruction may be referred to as the IFTL or inform-first-test-later paradigm.

An interesting variation of the IFTL approach is the TFIL or test-first-inform-later paradigm. In this approach, Q is presented first and the unit of information U presented later only when necessary, i.e. when the student shows he does not already know the concept by failing to answer Q correctly.

The two approaches can be shown diagrammatically in Figure 3.2.

TFIL in effect does not present to the student what he already knows, but gives only that those information he needs. It is therefore an adaptive instructional technique in which good students could save much valuable time by not having to go over familiar material. At the same time students not familiar with a given concept are is not penalised - but has the chance of learning

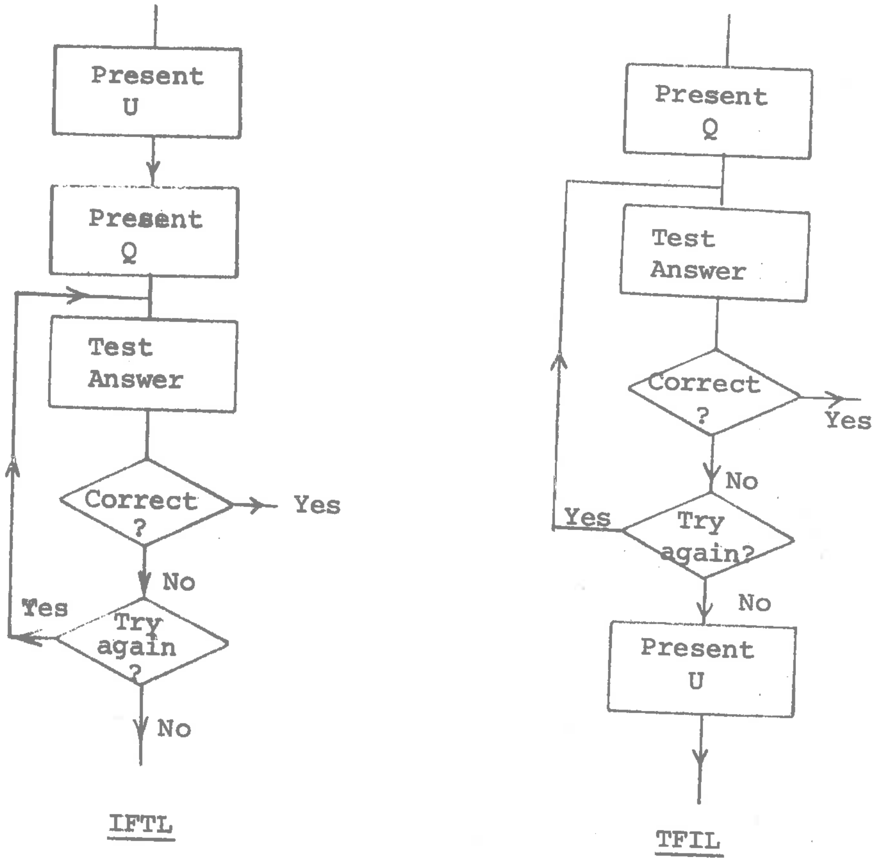


Figure 3.2 Two Instructional Paradigms

it. Since it is both difficult and costly to develop good branching programs, TFIL might prove to be a cheap, simple and effective way of developing adaptive teaching material.

It is not suggested here that TFIL is widely applicable and can be used to achieve all teaching objectives. Rather it might prove to be especially appropriate and advantageous in some areas of application. As an example, TFIL might be very suitable for course revision, since students could be expected to be fairly familiar with the course, but need to review it and be refreshed in some areas. One can also envisage a TFIL course that can be used for teaching as well as for revision, using the same material, since only those materials that have not been properly learnt or remembered will be re-presented.

TFIL also enables the student to learn by discovery, since in being tested on a concept even before it is introduced and explained, the student is required to 'discover' the answer. Materials learnt by discovery are often better retained, although there is no general agreement on this [16].

Between the two extremes of IFTL and TFIL there is a rich spectrum of allied instructional paradigms. In fact it is often possible to break U into subunits of information, U_1, U_2, ... U_k, not necessarily of equal substance. The student can then be taught a concept, by being presented with the evaluative question Q, and then U_1, U_2, ... U_k in that order for each successive incorrect attempt. If U_1 is more substantial than the other subunits U_2, ... U_k, then by presenting firstly U_1, and then Q, and if necessary, successively U_2, U_3, ... U_k we have a cue-oriented approach to instruction. One could think of various alternatives to these - all having the feature of teaching by guided discovery.

There is still considerable confusion over whether discovery learning refers to a method of teaching, a method of learning or something one learns. De Cecco [16] defines it as pertaining to those teaching situations in which the student achieves the instructional objectives with limited or no guidance from the teacher. It is characterised by the amount of guidance the teacher provides.

Bruner [6] claims four advantages for discovery learning. It :-

(a) increases intellectual potency

(b) increases intrinsic motivation

(c) teaches discovery techniques, and

(d) improves retention.

These various aspects of discovery learning can be investigated through the TFIL and other related cue-oriented techniques for teaching that have just been discussed. In particular the effect of varying degrees of guidance on the effectiveness of learning can be studied very conveniently in ALFIE.

Useful changes that could be made to ALFIE include :-

(1) On-line facility: The development of an on-line keyboard-entry course preparation facility would be very useful. No special complicated card code for typewriter characters and functions would be needed. More importantly, in conjunction with immediate testing of course segments as they are built, the course can be quickly debugged and revision effected immediately.

(2) Answer processing: Present facilities are still rather primitive and sufficient to meet only immediate needs. However, these will have to be expanded. This expansion should not be done by simply adding more answer recognition modes. Rather what is required is a concerted examination of the answer processing problem so that simple and consistent methods for specifying acceptable answers, and the corresponding processors for recognising matching answers, can be developed.

(3) Course Structure: It would be far more convenient to the author if the organisation of a course into chapters and pages were left to the compiler. The author can then regard a course as a continuous sequence

of text and instructions.

(4) Imbedding Problems: At present problem blocks may not be nested. The removal of this restriction will enable a richer variety of programming techniques to be implemented.

(5) Edit Feature: The edit card has proved to be very useful, but does not provide enough features. It should therefore be expanded to include two-sided justification (at present all lines start on column 1) and richer options for text editing.

(6) Keywords: Some of the keywords can be simplified, e.g. $GTC(ONE,TWO) instead of the current $GOTO,CHAPTER(ONE), PAGE(TWO), and $GTP(P22) instead of $GOTO,PAGE(P22).

(7) Performance Data: At present student responses can be recorded. But there are no packages for extracting useful information from the recorded data and performing other useful analyses. Such packages should therefore be developed, with special emphasis on ease of use by interested parties. Simple performance summaries should also be available from the keyboard.

CHAPTER IV

EXPRESSION EQUIVALENCE AND CONSISTENCY SETS

Our aim in this and the next three chapters is to investigate some of the main problems in developing a trigonometric proof supervisor (TPS) and to suggest possible solutions. As indicated previously we envisage students inputting proofs from remote CAI terminals. The function of TPS is to check the incoming proofs step by step and to assist and guide the students as required.

For TPS to be of practical value in CAI, students should be serviced without unbearable delay. Therefore we should seek solutions which are practical and efficient enough to meet this real-time requirement.

Problem Scope

We confine our investigation to the class of mono-argument problems. This means that all the trigonometric functions will be of a single argument θ. Our class $\mathcal{R}$ of all allowable trigonometric expressions will be as given by the BNF definition below.

```
<digit>          ::=  0|1|2|...|7|8|9
<unsigned int>   ::=  <digit>|<unsigned int><digit>
<trig fn>        ::=  sinθ|cosθ|tanθ|cscθ|secθ|cotθ
<AOP>            ::=  +|-
<MOP>            ::=  *|/
<primary>        ::=  <trig fn>|<unsigned int>|(<trig exp>)
<factor>         ::=  <primary>|<primary>↑<unsigned int>
<term>           ::=  <factor>|<term><MOP><factor>
<trig exp>       ::=  <term>|<trig exp><AOP><term>|<AOP><term>
```

$\mathcal{R}$ corresponds to the syntactic class <trig exp>. However we exclude from $\mathcal{R}$ expressions like $\sin\theta/(\tan\theta-\sin\theta/\cos\theta)$ which are mathematically ill-defined. The symbol '↑' denotes exponentiation. We have excluded variables from our expressions, but this is merely to simplify the presentation of our theory of consistency sets.

e.g. $(\sin\theta+2*\cos\theta)\uparrow 2$ and $(\csc\theta+1)/(1-\sec\theta\uparrow 2+\cot\theta)$, but not

$\sin(A+B)-\tan C$ and $B*\sin\theta+C/\cos\theta$, are valid expressions in $\mathcal{R}$.

In the subsequent we may use the more usual and convenient notations ab and $a\cdot b$ for $a*b$, a^2 for $a\uparrow 2$ and so on. Also when brevity is desired we may write $\sin\theta$, $\cos\theta$, ... as sin, cos, ... without the θ.

Table 4.1 lists the basic identities I1, I2, ... I8 and the supplementary identities J1, J2, J3, J4 in their standard form. These are the identities which we permit the student to assume in his proofs. Except for the supplementary ones, this choice of reference identities is fairly standard in textbooks. We allow these identities to be used (in substitution) in their various simple variants. Thus I1 may be used in the form $\csc\theta-1/\sin\theta\equiv 0$ and $\csc\theta\sin\theta\equiv 1$ while I6 may be used in such forms as $\sin^2\theta\equiv 1-\cos^2\theta$ and $(\cos^2\theta-1)/\sin^2\theta\equiv -1$. Note that the reference identities are by no means independent since, given I1, I2, I3, I4 and I6, we can derive the remaining identities in the table.

Non-Pythagorean			
$\csc\theta\equiv 1/\sin\theta$	... I1	$\sec^2\theta\equiv\tan^2\theta+1$	... I7
$\sec\theta\equiv 1/\cos\theta$	... I2	$\csc^2\theta\equiv\cot^2\theta+1$	... I8
$\cot\theta\equiv 1/\tan\theta$	... I3	Supplementary	
$\tan\theta\equiv\sin\theta/\cos\theta$	.. I4	$\tan\theta\equiv\sin\theta.\sec\theta$	... J1
$\cot\theta\equiv\cos\theta/\sin\theta$	.. I5	$\tan\theta\equiv\sec\theta/\csc\theta$	... J2
Pythagorean		$\cot\theta\equiv\cos\theta.\csc\theta$	... J3
$\sin^2\theta+\cos^2\theta\equiv 1$	.. I6	$\cot\theta\equiv\csc\theta/\sec\theta$	... J4

Table 4.1 Reference Identities

Difficulty in Proof-Checking

Let $e_\ell \equiv e_r$ $(e_\ell, e_r \in \mathcal{R})$ be a problem identity. The aim of a proof is to show the equivalence of e_ℓ and e_r. There are various waysof doing this, the most direct being to derive e_r from e_ℓ. Other approaches include deriving 0 from $e_\ell - e_r$ and deriving 1 from e_ℓ / e_r. However this is a question of proof strategy which we will ignore for now in this derivation. Without loss of generality we will assume a proof to be a chain of steps which derives a target expression from an initial expression using only algebraic manipulation and trigonometric substitution. Each step derives one expression from another.

The task of proof-checking is then essentially to verify the correctness and acceptability of each step in a proof. The difficulty is that in our algebraic system, the various basic rules and axioms, especially of commutativity, associativity and distributivity, give rise to a proliferation of informal rules. These informal rules have become standard in algebraic reasoning because of their long-standing usage. Thus the step $\sin^2\theta - \cos^2\theta \rightarrow (\sin\theta - \cos\theta) * (\cos\theta + \sin\theta)$ uses an informal rule which may be regarded as a combination of the factorisation rule '$A^2 - B^2 = (A+B)*(A-B)$' and the commutativity of '+' and '*'. The factorisation rule is itself based on the axiom of distributivity, the definition of indices and so on. Because there are so many of these informal rules it would be impractical, if not futile, to check their explicit use in a step.

Some formal systems, like the predicate calculus, do not have so many informal rules because of the nature of their axioms and inference rules. It may therefore be a reasonable approach in such systems, to check a proof by verifying explicitly that only valid inference rules have been used and valid premises assumed.

It is suggested that a direct approach for checking trigonometric proofs would not be very promising. In particular this means that a purely symbolic approach should be avoided. In this thesis we have opted for an approach which is largely numeric.

Preliminary Notions and Definitions

By the rational operations of algebra we shall mean those operations that conform to the laws of equality of expressions which are consequences of the basic laws of equality, the field axioms* and the definitions of subtraction, division and indices. This means that trigonometric substitutions are excluded. We list below some examples of rational algebraic operation. a,b,c and d stand for expressions, terms, or factors as appropriate and m and n are integers.

- $a\times(-b) = -a\times b$
- $(-a)+(-b) = -(a+b)$
- $(a^m)^n = a^{mn}$
- $a^0 = 1$
- $(-a)\times(-b) = a\times b$
- $-(a-b) = -(b-a)$
- $(a\times b)^m = a^m\times b^m$
- $a/b+c/d = (ad+bc)/bd$
- $a\times 0 = 0$
- $-0 = 0$
- $a^m\times a^n = a^{m+n}$

Let us now define a few terms. All expressions are from $\mathbb{Q}$.

Defn: Two expressions f and g are equivalent, denoted $f\equiv g$, if one is derivable from the other solely by rational algebraic operations and substitution of reference identities.

* *The field axioms are the ring axioms (see Appendix B) together with the two additional axioms :*

(9) If $c\neq 0$ and $c\cdot a = c\cdot b$ then $a=b$ (cancellation)
(10) $\forall a\in R$, $a\neq 0$, $\exists\, a^{-1}\in R$ such that $a^{-1}.a=1$ (multiplicative inverse)

R is then a field, if these axioms hold.

e.g. (1) $(1-\sin\theta/\cos\theta)*\sec^2\theta\equiv(1-\tan\theta)*(1+\tan^2\theta)$

(2) $(1-\sin^2\theta)\equiv(1-\sin\theta)*(1+\sin\theta)$

(3) $(1-\sin^2\theta)\not\equiv\tan\theta-\cot\theta$

Defn. f and g are algebraically equivalent or A-equivalent, denoted $f\overset{A}{\equiv}g$, if one is derivable from the other solely by rational algebraic operations.

e.g. (1) $(1-\sin^2\theta)\overset{A}{\equiv}(1-\sin\theta)*(1+\sin\theta)$

(2) $(1-\sin\theta/\cos\theta)*\sec\ \theta\overset{A}{\not\equiv}(1-\tan\theta)*(1+\tan^2\theta)$

Defn: f and g are trigonometrically equivalent or T-equivalent, denoted $f\overset{T}{\equiv}g$, if they are equivalent but not A-equivalent. This means that substitution is required to derive a T-equivalent expression.

e.g. $(1-\sin\theta/\cos\theta)*\sec^2\theta\overset{T}{\equiv}(1-\tan\theta)*(1+\tan^2\theta)$

Defn: A step is an ordered pair of expressions (f,g), and is denoted by $f\to g$. The step is said to be correct if $f\to g$. It is an A-step or a T-step according as $f\overset{A}{\equiv}g$ or $f\overset{T}{\equiv}g$.

Notn: Let $e_0\to e_1\to e_2\to\dots\to e_n$, abbreviated to $e_0\overset{*}{\to}e_n$, represent the sequence of steps $e_0\to e_1$, $e_1\to e_2$, ... $e_{n-1}\to e_n$.

Defn: A proof is a chain of steps $e_0\overset{*}{\to}e_n$ which derives a target expression e_n from an initial expression e_0. (It is assumed that e_0 and e_n are appropriate for the problem identity.)

Under our formulation a proof $e_0\overset{*}{\to}e_n$ is valid if and only if

(1) each step $e_i\to e_{i+1}$ $(i=0,1,\dots n-1)$ is correct and

(2) each step $e_i\to e_{i+1}$ $(i=0,1,\dots n-1)$ is small.

Our primary task in checking a proof is accordingly one of verifying the correctness and smallness of each step in the proof.

Step correctness is a question of expression equivalence. The main difficulty in deciding equivalence is that a given expression can appear in any of an infinite number of equivalent ways. A purely symbolic approach to this problem would suffer not only from inefficiency, but the more serious problem of undecidability as well. Expression equivalence will be discussed in the next section.

A step is said to be small if its correctness is readily verifiable. But what is readily verifiable is rather subjective. The step "$1-\cot\theta\to 1-\cos\theta/\sin\theta$" would probably be regarded as small whereas "$\cot\theta+\tan\theta\to\csc\theta*\sec\theta$" would probably be considered large (i.e. not small). However it is not obvious whether "$1-\tan\theta-\cot\theta+\tan\theta\cot\theta\to 2-1/(\cos\theta\sin\theta)$" should be regarded as large or small. Chapter V is devoted to this step-size problem.

Expression Equivalence

Quite apart from its application in proof-checking, expression equivalence is an important problem in the following.

(1) Answer Recognition: In CAI-based mathematics where a required answer happens to be an algebraic expression, it is clearly impossible for the author to anticipate all possible correct answers. The answer processor should be able to recognise expression equivalence.

(2) Object Code Optimisation:* To produce efficient object codes

* *I am grateful to Dr. J.G. Sanderson for bringing this application to my attention.*

efficiently some compilers perform common subexpression detection, e.g. in array subscripts, so that the generation of duplicative machine codes can be minimized, avoided or speeded up. For this optimisation to be worthwhile, we may need a very cheap method for deciding equivalence, e.g. a numeric one.

(3) Formula Manipulation: Expressions generated by formula manipulation systems (see [10]) are often very long and unwieldy. Different systems generate different, but equivalent, formulae for the same problem. There is often a need to establish the equivalence of two different formulae.

The expression equivalence (decision) problem has been approached in various ways. Simplification techniques in formula manipulation languages [10] provide an indirect algebraic method. However these have enjoyed only partial success because simplification is an ill-defined concept. As an alternative to simplification B.F. Caviness [8] proposed the well-defined concepts of normal and canonical forms for expressions. We shall now examine briefly Caviness's work.

The Work of Caviness

Let $\mathcal{E}$ be a well-defined class of expressions in which the members are formed from a finite set of atomic symbols, a subset of which are variables. The expressions are to be regarded as functions over some domain $\mathfrak{D}$.

Two expressions E_1 and E_2 are identical, written $E_1 \equiv E_2$ (note '$\equiv$' not used for equivalence here), if they are the same string of atomic symbols. They are said to be equivalent if for all assignments of values in $\mathfrak{D}$ to their variables, for which they are defined, they are equal. We write $E_1=E_2$ if E_1 and E_2 are equivalent.

A computable function f is a normal form for $\mathcal{E}$ if it is a mapping from $\mathcal{E}$ to $\mathcal{E}$ satisfying:(1) $f(E)=E \quad E\varepsilon\mathcal{E}$ and (2) $f(E)\equiv 0$ if $E=0$. If further: (3) $f(E_1)\equiv f(E_2)$ whenever $E_1 = E_2$ then it is an f-canonical form for $\mathcal{E}$. An expression E is in f-canonical form if $f(E)\equiv E$. The normal form, unlike the canonical, is too general to be useful for deciding equivalence. Its requirements (1) and (2) can be trivially satisfied by the mapping: $f(E)\equiv E$, but with $f(E)\equiv 0$ whenever $E=0$.

In a class for which a canonical form exists, Caviness has shown that expressions can be reduced to the standard normal from P/Q in which P and Q are canonical. This provides an indirect method for deciding equivalence. But this method is not suitable for our problem because our class $\mathcal{R}$ has no canonical form. Even if $\mathcal{R}$ has a canonical form, the derivation of the standard form is by no means an easy task.

Richardson (cited by Caviness) has shown that for the class $\mathcal{R}_2$, generated by :

(1) the rationals, π and $\log_e 2$

(2) the variable x

(3) the operations +, * and composition

(4) the sine, exponential and absolute functions.

the predicate "E=0" for E in $\mathcal{R}_2$ is recursively undecidable. Caviness has also shown that $\mathcal{R}_2$ has no canonical form. Note that our $\mathcal{R}$ is a richer class than $\mathcal{R}_2$.

A major objection to using a symbolic approach to the equivalence problem is that it would be very inefficient. An expression can appear in far too many distinct forms. A more serious objection however is Richardson's undecidability result which shows that expression equivalence cannot always be resolved symbolically.

While purely algebraic and symbolic techniques are unsuitable, a numeric one can be a very elegant and practical alternative. Numeric methods for deciding equivalence have already been adopted by Oldehoeft [52], Martin [44] and others.

One class of expressions considered by Oldehoeft is the T_1-class. The expressions are constructed from variables, constants and the following operators and functions: +, -, *, /, composition, ↑ (to an integer power), sin, cos, tan, csc, sec, cot, exp, r^x $(r>0)$, sinh, cosh, tanh, csch, sech and coth. Oldehoeft has shown that to test for the equivalence of two T_1-expressions, it is sufficient to compare their values computed at one random point. The expressions are equivalent if and only if their values are equal. The probability of an incorrect decision in this random evaluation method is zero. The numeric approach can be seen as a direct consequence of the alternative definition for equivalence: two expressions are equivalent over some domain if they are equal at every point of the domain where they are defined.

Oldehoeft's method is applicable to our problem since his class T_1 includes our class $\mathcal{Q}$. However we have developed a more general numeric technique which enables us not only to decide equivalence but also to determine a minimum sufficient set of basic identities for proving an identiy. This set is the consistency set of the identity.

Our technique enables us to distinguish between A-equivalence and T-equivalence. Note that Caviness's technique applies only to A-equivalence.

The Theory of Consistency Sets

We have $\mathcal{R}$ our set of expressions. Let $\mathcal{R}^\circ$ be the field of rational functions* in $\underset{\sim}{X}$ with rational number coefficients. $\underset{\sim}{X}$ is the row vector $(x_1,x_2,\ldots x_n)$**. Members of $\mathcal{R}$ will be called <u>θ-expressions</u> and will be said to be in the <u>θ-form</u>. Similarly members of $\mathcal{R}^\circ$ will be called <u>X-expressions</u> and are in the <u>X-form</u>.

Consider the mapping $\mathfrak{J}$ of $\mathcal{R}$ into $\mathcal{R}^\circ$ which transforms θ-expressions into X-expressions by substituting $x_1,x_2,\ldots x_6$ for any occurrence of sinθ, cosθ, ...cotθ respectively. We can denote this mapping by $f(\theta) \overset{\mathfrak{J}}{\rightarrow} f^\circ(\underset{\sim}{X})$. We shall denote the X-form of f under $\mathfrak{J}$ by f° and the θ-form of g° by g, provided g° is known.

e.g. <u>f: θ-form</u>	<u>f°: X-form</u>
5*secθ-2*sinθ/cosθ	$5x_5-2x_1/x_2$
124	124
2-secθ*cscθ↑2	$2-x_5x_4^2$

The X-form of the basic identities are NOT identities but the rational equations R-1, R-2,...R-8 below. Let r_1°, r_2°, ... represent the <u>reference rationals</u> $(x_4x_1-1)/x_1$, $(x_5x_2-1)/x_2$, ...
In keeping with our notation r_1 is (cscθsinθ-1)/sinθ and so on.

* *A rational function is one which is expressible as the ratio of two polynomial functions.*

** *Since our trigonometric expressions do not have variables, $x_1,x_2,\ldots x_6$ would be sufficient actually.*

$r_1^\circ(\underset{\sim}{X}) \equiv (x_4x_1-1)/x_1=0$ .. R-1

$r_2^\circ(\underset{\sim}{X}) \equiv (x_5x_2-1)/x_2=0$ .. R-2

$r_3^\circ(\underset{\sim}{X}) \equiv (x_6x_3-1)/x_3=0$ .. R-3

$r_4^\circ(\underset{\sim}{X}) \equiv (x_3x_2-x_1)/x_2=0$ .. R-4

$r_5^\circ(\underset{\sim}{X}) \equiv (x_6x_1-x_2)/x_1=0$ .. R-5

$r_6^\circ(\underset{\sim}{X}) \equiv x_1^2+x_2^2-1=0$ .. R-6

$r_7^\circ(\underset{\sim}{X}) \equiv x_5^2-x_3^2-1=0$ .. R-7

$r_8^\circ(\underset{\sim}{X}) \equiv x_4^2-x_6^2-1=0$ .. R-8

Consider the corresponding polynomial equations P-1,P-2,..P-8. Here the reference polynomials are represented by p_1°, p_2°, .. p_8°.

$p_1^\circ(\underset{\sim}{X}) \equiv x_4x_1-1=0$ .. P-1

$p_2^\circ(\underset{\sim}{X}) \equiv x_5x_2-1=0$ .. P-2

$p_3^\circ(\underset{\sim}{X}) \equiv x_6x_3-1=0$ .. P-3

$p_4^\circ(\underset{\sim}{X}) \equiv x_3x_2-x_1=0$ .. P-4

$p_5^\circ(\underset{\sim}{X}) \equiv x_1x_2-x_2=0$ .. P-5

$p_6^\circ(\underset{\sim}{X}) \equiv x_1^2+x_2^2-1=0$ .. P-6

$p_7^\circ(\underset{\sim}{X}) \equiv x_5^2-x_3^2-1=0$ .. P-7

$p_8^\circ(\underset{\sim}{X}) \equiv x_4^2-x_6^2-1=0$ .. P-8

The reference rationals and polynomials may be taken as functions which map $\mathbb{C}^n$ into $\mathbb{C}$.* For practical purposes we shall take the mapping to be from $\mathbb{R}^n$ into $\mathbb{R}$.

Define: $M(f^\circ) = \{\underset{\sim}{X}\varepsilon\ \mathbb{C}^n | f^\circ(\underset{\sim}{X})=0\}$

$M(f^\circ)$ denotes the solution set of the equation $f^\circ(\underset{\sim}{X})=0$ in $\mathbb{C}^n$.

Let: $N^* = \{1,2,3,4,5,6,7,8\}$

$Bi = M(r_i^\circ) \quad i\varepsilon N^*$

$Ai = M(p_i^\circ) \quad i\varepsilon N^*$

$\mathcal{B} = \{\bigcap_{i\varepsilon N} Bi | N\subset N^*\}$

$\mathcal{A} = \{\bigcap_{i\varepsilon N} Ai | N\subset N^*\}$

* $\mathbb{R}$ *and* $\mathbb{C}$ *denote the fields of complex and real numbers respectively.*

Members of $\mathcal{A}$, e.g. A1, $A1 \cap A6$, $A2 \cap A3 \cap A6$, will be called sampling sets or S-sets while members of $\mathcal{B}$, e.g. $\mathfrak{C}^n$, $B2 \cap B8$, will be called R-sets. The S-sets are algebraic manifolds.* The largest S-set, $\mathfrak{C}^n$, corresponds to the empty system of equations while $A1 \cap A2 \cap \ldots \cap A8$ is the smallest S-set. This latter set is non-empty since the equations P-1, P-2,...P-8 are consistent.

If S_1 and S_2 are S-sets and $S_1 \subset S_2$ then we say that S_1 is an S-subset of S_2 and that S_2 is an S-superset of S_1. $A1 \cap A2 \cap \ldots \cap A8$ is an S-subset of every S-set while $\mathfrak{C}^n$ is an S-superset of every S-set.

Let $\mathcal{I}^* = \{I1, I2, \ldots I8\}$ and $\mathcal{C} = \{\mathcal{I} \mid \mathcal{I} \subset \mathcal{I}^*\}$ the set of all subsets of $\mathcal{I}^*$. Theoretically there are $2^8 = 256$ subsets. However in $\mathcal{C}$ we shall not distinguish between any two elements $\mathcal{I}_1 = \{Ii \mid i \epsilon N_1\}$ and $\mathcal{I}_2 = \{Ii \mid i \epsilon N_2\}$, $N_1, N_2 \subset N^*$, if $\bigcap_{i \epsilon N_1} Bi = \bigcap_{i \epsilon N_2} Bi$. The elements of $\mathcal{C}$ will be called consistency sets or C-sets. We represent a C-set by the largest set of basic identities if it is not unique.

e.g. The sets {I3,I4}, {I3,I5}, {I4,I5} and {I3,I4,I5} are not distinguished since $B3 \cap B4 = B3 \cap B5 = B4 \cap B5 = B3 \cap B4 \cap B5$, and are represented by the C-set {I3,I4,I5}.

Because our eight basic identities are not independent, there are only 98 distinct C-sets. These are given in Table 4.2 where they are sequenced 0,1,2,...97 representing the C-sets C0,C1,C2,...C97; e.g. C18, the C-set {I1,I2,I4} is shown as [1,2,4] in the table. Note the empty C-set C0 which is a C-subset of every C-set, and C97 the

* *An algebraic manifold is the solution set of a system of polynomial equations. See Appendix B.*

0	[ ] (EMPTY SET)
1	[1]
2	[2]
3	[3]
4	[4]
5	[5]
6	[6]
7	[7]
8	[8]
9	[1,2]
10	[1,3]
11	[1,4]
12	[1,5]
13	[2,3]
14	[2,4]
15	[2,5]
16	[3,4,5]
17	[1,2,3]
18	[1,2,4]
19	[1,2,5]
20	[1,3,4,5]
21	[2,3,4,5]
22	[1,2,3,4,5]
23	[6,7]
24	[6,8]
25	[7,8]
26	[6,7,8]
27	[1,6]
28	[2,6]
29	[3,6]
30	[4,6]
31	[5,6]
32	[1,7]
33	[2,7]
34	[3,7]
35	[4,7]
36	[5,7]
37	[1,8]
38	[2,8]
39	[3,8]
40	[4,8]
41	[5,8]
42	[1,6,7]
43	[3,6,7]
44	[5,6,7]
45	[2,6,8]
46	[3,6,8]
47	[4,6,8]
48	[1,7,8]
49	[2,7,8]
50	[3,7,8]
51	[4,7,8]
52	[5,7,8]
53	[1,2,6]
54	[1,3,6]
55	[1,4,6]
56	[2,3,6]
57	[2,5,6]
58	[1,2,7]
59	[1,3,7]
60	[1,4,7]
61	[1,5,7]
62	[2,3,7]
63	[2,5,7]
64	[1,2,8]
65	[1,3,8]
66	[1,4,8]
67	[2,3,8]
68	[2,4,8]
69	[2,5,8]
70	[3,4,5,6]
71	[1,2,3,6]
72	[3,4,5,7]
73	[1,2,3,7]
74	[1,2,5,7]
75	[3,4,5,8]
76	[1,2,3,8]
77	[1,2,4,8]
78	[1,2,4,6,7]
79	[1,3,6,7]
80	[1,5,6,7,8]
81	[2,3,4,5,6,7]
82	[2,4,6,7]
83	[1,2,5,6,8]
84	[1,3,4,5,6,8]
85	[1,5,6,8]
86	[2,3,6,8]
87	[2,4,6,7,8]
88	[1,2,7,8]
89	[1,3,7,8]
90	[1,4,7,8]
91	[2,3,7,8]
92	[2,5,7,8]
93	[3,6,7,8]
94	[1,3,4,5,7]
95	[2,3,4,5,8]
96	[3,4,5,7,8]
97	[1,2,3,4,5,6,7,8]

C-SET TABLE

Figure 4.2

largest C-set which is a C-superset of every C-set.

We define C(S), the C-set of the S-set S by $C(S)=\{Ii \mid \bigcap_i Ai=S\}$, e.g. $C(A3 \cap A4)=C(A3 \cap A4 \cap A5)=\{I3,I4,I5\}$. If S_1 and S_2 are S-sets and $S_1 \subset S_2$, then $C(S_1) \supset (S_2)$; smaller S-sets correspond to larger C-sets and vice versa. The smallest S-set has the largest C-set C97.

A step $f(\theta) \to g(\theta)$ is said to hold over a set W if $f^\circ(\underset{\sim}{X})=g^\circ(\underset{\sim}{X})$ for every $\underset{\sim}{X} \in W$, $W \subset \mathbb{C}^n$, provided both f° and g° are well defined at $\underset{\sim}{X}$.* If $f(\theta) \to g(\theta)$ holds over W, then $f(\theta) \equiv g(\theta)$ will be said to hold over W. If W is also an S-set, then the step $f \to g$ (as well as $f \equiv g$) is said to hold on the C-set C(W).

Remark: If $f \to g$ holds over $S = \bigcap_{i \in N} Ai$, then $f^\circ(\underset{\sim}{X})=g^\circ(\underset{\sim}{X})$ (pwd) is true where $r_i^\circ(\underset{\sim}{X})=0$ $(i \in N)$ is true. In effect this means that C(S) makes $f^\circ=g^\circ$ (pwd) "consistent" - hence the name consistency set.

On Appendix B

In Appendix B we derive the three theorems below.

Thm 1: Let $w(\underset{\sim}{X}) \equiv u(\underset{\sim}{X})/v(\underset{\sim}{X})$ where u and v are relatively prime polynomials. If a polynomial vanishes on M(w) then it vanishes on M(u).

Thm 2: Let $w_1, w_2, \ldots w_k$ be rational expressions in which $w_i = u_i/v_i$ $(i=1,2,\ldots k)$, u_i and v_j $(1 \leqslant i,j \leqslant k)$ being relatively prime polynomials. Then any polynomial vanishing on $\bigcap_{i=1}^{k} M(w_i)$ also vanishes on $\bigcap_{i=1}^{k} M(u_i)$.

Thm 3: Let $w_1, w_2, \ldots w_k$ be rational expressions in which $w_i=u_i/v_i$ $(i=1,2,\ldots k)$, u_i and v_j $(1 \leqslant i,j \leqslant k)$ being relatively prime polynomials. If r is any rational expression which vanishes (pwd) on $\bigcap_{i=1}^{k} M(w_i)$ then

* *This means whenever $f^\circ(\underset{\sim}{X})$ and $g^\circ(\underset{\sim}{X})$ are well-defined, they are equal. We shall abbreviate this qualifying clause to pwd (provided well-defined).*

$r^m = \sum_{i=1}^{k} s_i \cdot w_i$ where m is a positive integer and each s_i is a rational expression which is not ill-defined all over $M(w_i)$.

Appendix B begins with a brief summary of relevant definitions and basic results in algebraic geometry. The summary includes definitions of rings, ideals and manifolds as well as Hilbert's Nullstellensatz (zero theorem) which states: if a polynomial p vanishes where the polynomials $\ddot{u}_1$, $\ddot{u}_2$, .. $\ddot{u}_k$ jointly vanish, then $p^m = \sum_{i=1}^{k} \ddot{v}_i \cdot \ddot{u}_i$ where m is a positive integer and $\ddot{v}_i$ (i=1,2,..k) are polynomials. All polynomials are in $\underset{\sim}{X}$ and over the complex field $\mathbb{C}$.

To prove the three theorems, Appendix B establishes two lemmas first. Lemma 1 states that if $A, B_1, B_2, \ldots B_k$ are irreducible manifolds (see Defn. B·7 in Appendix B) and C is a manifold which covers $A-(B_1 \cup B_2 \cup \ldots \cup B_k)$ then C covers A. This lemma says in effect that, under the given conditions, there is no manifold smaller than A that covers $A-(B_1 \cup B_2 \cup \ldots \cup B_k)$. Lemma 2 extends this result to the case where A is replaced by a collection of irreducible manifolds which are distinct from $B_1, B_2, \ldots B_k$. The significance of Lemma 2 is that an arbitrary polynomial which vanishes where a rational function u/v vanishes, u and v being relatively prime polynomials, must also vanish where u itself does. This is in fact what Thm 1 is about. Note that $M(u/v)=M(u)-M(v)$ and M(u) and M(v) are expressible as the union of irreducible manifolds (see Res-11, Appendix B), the irreducible manifolds of M(u) and M(v) being distinct. The conditions of Lemma 2 are therefore satisfied.

Thm 2 is an extension of Thm 1. It shows that if a polynomial vanishes on a R-set $\bigcap_{i \in N} M(r_i^\circ)$ (i.e. $\bigcap_{i \in N} Bi$) then it actually vanishes on the corresponding S-set $\bigcap_{i\epsilon N} M(p_i^\circ)$. Consequently there is no need to distinguish between an R-set and its corresponding S-set when a rational expression vanishes on the R-set.

Thm 3 is based on Hilbert's Nullstellensatz and Thm 2. It connects a rational to a set of rationals on whose common zeroes it vanishes by expressing it as a "linear combination" of the rationals in the set. In this relation, viz. $r^m = \sum_{i=1}^{k} s_i . w_i$, we shall say that r is L-expressed in terms of $w_1, w_2, \ldots w_k$. This relationship is non-trivial. A word of explanation is in order.

The rational expressions in $\underset{\sim}{X}$ over a ring form a field and as such every non-zero rational has an inverse. A consequence of this is that given an arbitrary set of rational expressions $R, R_1, R_2, \ldots R_k$, which are non-zero, we can L-express R in terms of $R_1, R_2, \ldots R_k$, viz. as $R = \sum_{i=1}^{k} T_i . R_i$ where $T_i = R/kR_i$. But this L-expression is trivial since each coefficient T_i is ill-defined everywhere on $M(R_i)$. In Thm 3, the coefficient s_i may be ill-defined on $M(w_i)$, but not all over it.

The relevance of Thm 1, Thm 2 and Thm 3 will become clear shortly.

Trigonometric Substitution

Suppose f(θ)→g(θ) is a T-step involving the substitution of say, I1. Let h denote f-g. The substitution can be made in any of the permitted variants* of I1: cscθ→1/sinθ, cscθ-1/sinθ→0, cscθsinθ-1→0 and so on. In general for $\underset{\sim}{X}$ from $\mathbb{C}^n$, $f^\circ(\underset{\sim}{X})$ and $g^\circ(\underset{\sim}{X})$

*See definition of permitted variants in Chapter V.

are unequal, i.e. $h^\circ(X)\neq 0$.* If the substitution is made in the standard form $\csc\theta\to 1/\sin\theta$ as in $\sin^2\theta+\csc\theta\tan\theta\to\sin^2\theta+\tan\theta/\sin\theta$, then $f^\circ(\underset{\sim}{X}) = g^\circ(\underset{\sim}{X})$ provided $x_4=1/x_1$, i.e. $h^\circ(\underset{\sim}{X})=0$ on the R-set B1. By Thm 1, this means $h^\circ(\underset{\sim}{X})=0$ on the S-set A1. If the substitution is made in another variant, then h° vanishes on the set of zeroes of $(x_4x_1-1)/q^\circ(\underset{\sim}{X})$,** i.e. $p_1^\circ(\underset{\sim}{X})/q^\circ(\underset{\sim}{X})$, where q° is some polynomial relatively prime to p_1°. By Thm 1, h° also vanishes on the S-set A1. If the substitution is made in several places, employing k distinct forms, then h° vanishes on the common zeroes of the corresponding set of k rational expressions $p_1^\circ/t_1^\circ, p_1^\circ/t_2^\circ, \ldots p_1^\circ/t^\circ{}_k$, the t_i° $(i=1,2,\ldots k)$ being polynomials relatively prime to p_1°. By Thm 2, h° vanishes on the common zeroes of $p_1^\circ, p_1^\circ, \ldots p_1^\circ$, which is the S-set A1.

In general if $f(\theta)\to g(\theta)$ is a T-step involving the substitution of the identities Ii $(i\varepsilon N, N\subset N^*)$, in whatever permitted variants, then it can be shown by Thm 2, that h° vanishes on the S-set $\bigcap_{i\varepsilon N} Ai$. We remind the reader here that when we say h° vanishes on the S-set, it is its numerator polynomial which does and in fact h° is ill-defined where its denominator polynomial vanishes.

It follows that the h° of every correct step vanishes on some S-set. An A-step vanishes on the S-set $\mathbb{C}^n$. If a step involves the

* *i.e. provided $f^\circ(\underset{\sim}{X})$ and $g^\circ(\underset{\sim}{X})$ are both well-defined. In this section we shall take such qualifications as understood to avoid the frequent need to specify them by 'pwd' etc.*

** *The equation defined by a permitted variant of I1, in its X-form, is of this form. For $\csc\theta-1/\sin\theta\to 0$, g° is X_1 and for $\sin\theta$ $1/\csc\theta$, it is X_4.*

substitution of Ii(iεN) then its h° vanishes on the S-set $\bigcap_{i\varepsilon N}$Ai. However in general the h° of an incorrect step does not vanish on any S-set. It may be regarded as an arbitrary rational expression unrelated to any S-set.

The above result relates a step to the S-set of the identities it uses for substitution. Thm 3 provides a converse result as we shall see.

As before let f→g be a step with h denoting f-g. Suppose h° vanishes where the rationals r_i^o (iεN) jointly vanish, i.e. h° vanishes on $\bigcap_{i\varepsilon N}$Bi (and therefore on $\bigcap_{i\varepsilon N}$Ai, by Thm 1). By Thm 3 $(h^o)^m \equiv \sum_{i\varepsilon N} s_i^o \cdot r_i^o$ where m is a positive integer and s_i^o is a rational expression which is not everywhere ill-defined on Bi. Converting this relation into the Θ-form we obtain $h(\Theta)^m \equiv \sum_{i\varepsilon N} s_i(\Theta) \cdot r_i(\Theta)$. It follows that if $r_i(\Theta) \equiv 0$ (iεN), then $h(\Theta) \equiv 0$, i.e. $f(\Theta) \equiv g(\Theta)$. This shows that if we use the relations $r_i(\Theta) \equiv 0$ (iεN), then the step f→g is correct. But $r_i(\Theta) = 0$ (iεN) are reference identities. What we have in fact shown is therefore that the C-set C($\bigcap_{i\varepsilon N}$Ai) are sufficient for proving the problem identity $f(\Theta) \equiv g(\Theta)$.

To recapitulate we have derived the two results:-

(1) Let $f(\Theta) \to g(\Theta)$ be a correct step involving substitution for the basic identities {Ii|iεN}, N⊂N*, then h° (where h is f-g) vanishes on the S-set $\bigcap_{i\varepsilon N}$Ai.

(2) If f°-g° vanishes on the S-set S then the identities of the C-set C(S) are sufficient for proving the identity $f(\Theta) \equiv g(\Theta)$.

Illustrative Examples

(1) Step: $\sec\theta\csc\theta \rightarrow \frac{1}{\cos\theta\sin\theta}$

Then $h^\circ \equiv x_5x_4-\frac{1}{x_2x_1}$, which vanishes (pwd) on $B1\cap B2$, can be expressed as $\frac{1}{x_1}(x_5-\frac{1}{x_2}) + x_5(x_4-\frac{1}{x_1})$, i.e. $\frac{1}{x_1} r_2^\circ+x_5r_1^\circ$.

Thus $\sec\theta\csc\theta - \frac{1}{\cos\theta\sin\theta} \equiv \frac{1}{\sin\theta}(\sec\theta - \frac{1}{\cos\theta}) + \sec\theta(\csc\theta-\frac{1}{\sin\theta})$.

This shows that the basic identities $\sec\theta \equiv 1/\cos\theta$ and $\csc\theta \equiv 1/\sin\theta$ are sufficient for proving that $\sec\theta\csc\theta \equiv \frac{1}{\cos\theta\sin\theta}$.

(2) Step: $(1-\cot\theta)(1-\tan\theta)\rightarrow(1-\frac{\cos\theta}{\sin\theta})(1-\frac{\sin\theta}{\cos\theta})$.

$h^\circ \equiv (1-x_6)(1-x_3)-(1-x_2/x_1)(1-x_1/x_2)$ vanishes (pwd) on $B4\cap B5$ and is expressible as $(\frac{x_2-x_1}{x_1})r_4^\circ+(x_3-1)r_5^\circ$. h is expressible as $(\frac{\cos\theta-\sin\theta}{\sin\theta})(\tan\theta-\frac{\sin\theta}{\cos\theta})+(\tan\theta-1)(\cot\theta-\frac{\cos\theta}{\sin\theta})$. The problem identity $f\equiv g$ can therefore be proved by using I4 and I5 only.

(3) The identity I7 is derivable from the three identities I2, I4 and I6. In fact r_7° which is $x_5^2-x_3^2-1$ vanishes on $B2\cap B4\cap B6$ and $x_5^2-x_3^2-1\equiv x_2(x_3^2+1)(x_2x_5+1)(x_5-\frac{1}{x_2})+x_2x_5^2(x_3x_2+x_1)(x_3-\frac{x_1}{x_2})+x_5^2(x_1^2+x_2^2-1)$.

i.e. $\underline{r_7^\circ \equiv x_2(x_3^2+1)(x_2x_5+1)r_2^\circ-x_2x_5^2(x_3x_2+1)r_4^\circ+x_5^2r_6^\circ.}$ This shows the relationship between I7 and I2, I4 and I6.

C-set of a Step

If h° vanishes (pwd) on an S-set then it also does on every of its S-subsets. Correspondingly a step which holds on a C-set also holds on every of its C-supersets. Since there are only 98 S-sets, then for the h° of a correct step, there must be a maximal one, S say, on which h° vanishes (pwd) and such that there is no S-superset of S on which it does. C(S) is then the corresponding minimal C-set on which the step f→g holds.

Defn: Let $f \to g$ hold on a C-set C, but not on any C-subset of C. Then C is called a C-set of the step $f \to g$ (or of $f \equiv g$).

A C-set of an identity is a minimum sufficient set of basic identities for proving it. It is minimum in the sense that we cannot prove it by using a proper C-subset of this C-set. An incorrect step does not have a C-set. The C-set of an A-step is the empty C-set Co.

We have used "a" rather than "the" C-set of a step because it is not always unique. Some steps have more than one C-set. An example is $(1+\tan^2\theta)/(1+\cot^2\theta) \to \sin^2\theta/\cos^2\theta$ which has two C-sets, viz. {I3,I4,I5} and {I1,I2,I7,I8}. In practice an overwhelming majority of the correct steps have only one C-set. For this reason, we sometimes, if only loosely, talk of "the C-set of a step".

The Representative Point Technique

Let $p°(\underset{\sim}{X})$ be an arbitrary polynomial, one that occurs "at random". Then since p° is in general not related to any reference polynomial, it is a reasonable assumption that there is a O-probability that M(p°) covers any component (a component is an irreducible manifold) of an S-set. Now if a manifold M_1 does not cover a component M_2, then their intersection is "negligible" compared to M_2. In fact the dimension of the components of $M_1 \cap M_2$ is lower than the dimension of M_2. Given the above assumption, there is an O-probability that $p°(\underset{\sim}{X}_0)=0$ for $\underset{\sim}{X}_0$ sampled randomly from an S-set. This yields the random evaluation criterion: A polynomial vanishes at a random point from an S-set S if and only if it vanishes all over S. There is an O-probability of the conclusion of this test being wrong.

This test can be extended to a rational expression $r°$ since this is expressible as $p°/q°$ where $p°$ and $q°$ are relatively prime polynomials. The criterion now reads: $r°(\underset{\sim}{X}\ )=0$ for random point X_0 from S-set S if and only if $r°$ vanishes on S, whenever it is not ill-defined. There is a zero probability of this test being incorrect and an O-probability that the test breaks down due to $r°(\underset{\sim}{X}_0)$ being ill-defined. This follows if we assume $q°$ to be arbitrary. This random evaluation criterion is similar to that of Oldehoeft's, except that his S-set is only $\mathbb{C}^n$. We do not adopt this random evaluation test because of two objections: (1) it is not practical to randomly sample a point from an S-set and (2) sampling is a time-consuming process.

As a simple alternative to the above test, we "permanently preselect a random point" $\underset{\sim}{X}(S)$ for each S-set S. $\underset{\sim}{X}(S)$ is called the <u>representative point</u> (RP) of S. $\underset{\sim}{X}(S)$ is selected to satisfy: (1) $\underset{\sim}{X}(S)\epsilon S$ and (2) $\underset{\sim}{X}(S)\not\subset S'$ if S' is a proper S-subset of S. Our <u>representative point technique (RPT)</u> then adopts the following criterion: A rational expression $r°$ vanishes (pwd) on an S-set S if and only if $r°(\underset{\sim}{X}(S))=0$. In practice we choose our RPs to be real numbers. To reduce the chance of a breakdown in the test as well as an erroneous conclusion, we give these numbers several decimal figures. This takes advantage of the fact that in practice our expressions have simple integer coefficients. The RPs we have actually used in our experiments are described in Chapter VII.

The RPT gives us a very simple numeric technique for finding the C-set of a step. In particular the step $f(\Theta)\rightarrow g(\Theta)$ holds over the sampling set S if and only if $f^\circ(\underset{\sim}{X}(S))=g^\circ(\underset{\sim}{X}(S))$.

Simple Illustrations

We show below how we may determine "the" C-set of a step by the RPT. This is however not our adopted approach for C-set determination (see Chapter VII). Our aim here is to show the principles involved. Only the essential numeric tests are shown. The RPs are assigned simple values to simplify manual calculations. The underlined numbers in an RP must satisfy the relevant equations. As an example the RP for A2∩A7, viz. $\{2,\underline{1/\sqrt{2}},\underline{1},5,\underline{\sqrt{2}},3\}$ satisfies the equations $x_5=1/x_2$ and $x_5^2= x_3^2+1$.

C-set	S-set	Representative Point
Ø	$\mathbb{C}^n$	$\underset{\sim}{X}_1$: $\{1,2,3,4,5,6\}$
{I3}	A3	$\underset{\sim}{X}_2$: $\{1,2,\underline{3},4,5,\underline{1/3}\}$
{I2,I4}	A2∩A4	$\underset{\sim}{X}_3$: $\{\underline{1},\underline{2},\underline{1/2},4,\underline{1/2},6\}$
{I2,I6}	A2∩A6	$\underset{\sim}{X}_4$: $\{\underline{3/5},\underline{4/5},1,2,\underline{5/4},6\}$
{I2,I7}	A2∩A7	$\underset{\sim}{X}_5$: $\{2,\underline{1/\sqrt{2}},\underline{1},5,\underline{\sqrt{2}},3\}$
{I4,I6}	A4∩A6	$\underset{\sim}{X}_6$: $\{\underline{3/5},\underline{4/5},\underline{3/4},6,7,8\}$
{I4,I7}	A4∩A7	$\underset{\sim}{X}_7$: $\{\underline{2},\underline{3},\underline{2/3},3,\underline{\sqrt{13}/3},8\}$
{I6,I7}	A6∩A7	$\underset{\sim}{X}_8$: $\{\underline{3/5},\underline{4/5},\underline{1},2,\underline{\sqrt{2}},6\}$
{I2,I4,I6,I7}	A2∩A4∩A6∩A6	$\underset{\sim}{X}_9$: $\{\underline{1/2},\underline{\sqrt{3}/2},1/\sqrt{3},5,2/\sqrt{3},3\}$

(1) Step: $\tan\Theta-\sin\Theta/\cos\Theta\rightarrow(\tan\Theta\cos\Theta-\sin\Theta)/\cos\Theta$

$f^\circ(\underset{\sim}{X}_1)=3-1/4=11/4$, $g^\circ(\underset{\sim}{X}_1)=(3x4-1)/4=11/4$

$\therefore$ $f^\circ(\underset{\sim}{X}_1)=g^\circ(\underset{\sim}{X}_1)$

$\therefore$ step holds over $\mathbb{C}^n$ and therefore on the C-set Ø; this has no C-subset.

$\therefore$ The C-set of the step is $\emptyset$ and we have an A-step.

(2) Step: $(1-\tan\theta)/(1+\tan\theta)\rightarrow(\cot\theta-1)/(\cot\theta+1)$

$f^\circ(\underset{\sim}{X}_2)=(1-3)/(1+3)=-1/2=(1/3-1)/(1/3+1)=g^\circ(\underset{\sim}{X}_2)$

$f^\circ(\underset{\sim}{X}_1)=(1-3)/(1+3)=-1/2\neq 5/7=(6-1)/(6+1)=g^\circ(\underset{\sim}{X}_1)$

The only C-subset of {I3} is $\emptyset$; step holds on {I3} but not on $\emptyset$. Therefore it is not an A-step and its C-set is {I3}.

(3) Step: $(\tan\theta+\sec\theta-1)/(\tan\theta-\sec\theta+1)\rightarrow(1+\sin\theta)/\cos\theta$.

$f^\circ(\underset{\sim}{X}_9)=(1/\sqrt{3}+2/\sqrt{3}-1)/(1/\sqrt{3}-2/\sqrt{3}+1)=\sqrt{3}=(1+1/2)/(\sqrt{3}/2)=g^\circ(\underset{\sim}{X}_9)$

$f^\circ(\underset{\sim}{X}_3)=(1/2+1/2-1)/(1/2-1/3+1)=0\neq 1=(1+1)/2=g^\circ(\underset{\sim}{X}_3)$

$f^\circ(\underset{\sim}{X}_4)=(1+5/4-1)/(1-5/4+1)=5/3\neq 2=(1+3/5)/(4/5)=g^\circ(\underset{\sim}{X}_4)$

$f^\circ(\underset{\sim}{X}_5)=(1+\sqrt{2}-1)/(1-\sqrt{2}+1)=\sqrt{2}+1\neq 3/\sqrt{2}=(1+2)/(1/\sqrt{3})=g^\circ(\underset{\sim}{X}_5)$

$f^\circ(\underset{\sim}{X}_6)=(3/4+7-1)/(3/4-7+1)=-9/7\neq 2=(1+3/5)/(4/5)=g^\circ(\underset{\sim}{X}_6)$

$f^\circ(\underset{\sim}{X}_7)=(2/3+\sqrt{13}/3-1)/(2/3-\sqrt{13}/3+1)=(\sqrt{13}-1)/(5-\sqrt{13})\neq 1=(1+2)/3=g^\circ(\underset{\sim}{X}_7)$

$f^\circ(\underset{\sim}{X}_8)=(1+\sqrt{2}-1)/(1-\sqrt{2}+1)=\sqrt{2}+1\neq 2=(1+3/5)/(4/5)=g^\circ(\underset{\sim}{X}_8)$

Thus the step holds on {I2,I4,I6,I7} but not over any of its immediate C-subsets {I2,I4}, {I2,I6},...{I6,I7}. This means that "the" C-set of the step is {I2,I4,I6,I7} and this can be shown to be <u>the</u> C-set of the step.

Table 4.3 contains a selection of identities and their C-sets, found with the aid of the C-set search table (Table C.2) described in Chapter VII. Note that identities 7 and 11 have non-unique C-sets. It would be interesting to prove some of the identities in the table and verify that the identities in their C-sets are indeed "necessary" and sufficient.

	Identity	C-set
1.	$\cos^4A-\sin^4A+1\equiv 2\cos^2A$	{I6}
2.	$(\sin A+\cos A)(1-\sin A\cos A)\equiv\sin^3A+\cos^3A$	{I6}
3.	$\frac{\sin A}{1+\cos A}+\frac{1+\cos A}{\sin A}\equiv 2\csc A$	{I1,I6}
4.	$\cos^6A+\sin^6A\equiv 1-3\sin^2A\cos^2A$	{I6}
5.	$1-\sin A\equiv(\sec A-\tan A)(1+\sin A)$	{I2,I4,I6,I7}
6.	$\frac{\csc A}{\csc A-1}+\frac{\csc A}{\csc A+1}\equiv 2\sec^2A$	{I3,I7,I8}
7.	$\frac{\csc A}{\cot A+\tan A}\equiv\cos A$	{I2,I3,I7,I8} & {I1,I3,I4,I5,I6}
8.	$(\sec A+\cos A)(\sec A-\cos A)\equiv\tan^2A+\sin^2A$	{I6,I7}
9.	$\frac{1}{\cot A+\tan A}\equiv\sin A\cos A$	{I3,I4,I5,I6}
10.	$\frac{1-\tan A}{1+\tan A}\equiv\frac{\cot A-1}{\cot A+1}$	{I3}
11.	$\frac{1+\tan^2A}{1+\cot^2A}\equiv\frac{\sin^2A}{\cos^2A}$	{I3,I4,I5} & {I1,I2,I7,I8}
12.	$\frac{\sec A-\tan A}{\sec A+\tan A}\equiv 1-2\sec A\tan A+2\tan^2A$	{I7}
13.	$\frac{\tan A}{1-\cot A}+\frac{\cot A}{1-\tan A}\equiv\sec A\csc A+1$	{I3,I7,I8}
14.	$(\sin A+\cos A)(\cot A+\tan A)\equiv\sec A+\csc A$	{I1,I2,I3,I4,I5,I6,I7,I8}
15.	$\sec^4A-\sec^2A\equiv\tan^4A+\tan^2A$	{I7}
16.	$\csc^2A-1\equiv\cos A\csc A$	{I1,I6}
17.	$(1+\cot A-\csc A)(1+\tan A+\sec A)\equiv 2$	{I3,I7,I8}
18.	$\frac{\cot A\cos A}{\cot A+\cos A}\equiv\frac{\cot A-\cos A}{\cot A\cos A}$	{I5,I6}
19.	$\frac{\cos A\csc A-\sin A\sec A}{\cos A+\sin A}\equiv\csc A-\sec A$	{I1,I2}
20.	$\frac{\tan A+\sec A-1}{\tan A-\sec A+1}\equiv\frac{1+\sin A}{\cos A}$	{I2,I4,I6,I7}

Table 4.3 Identities and Their C-sets

Chapter V

THE STEP-SIZE PROBLEM

This chapter is an attempt to answer the question: What is a small step? First we shall discuss some of the considerations that might enter into the human analysis of step-size. We shall then introduce the idea of Φ-equivalence and use it to define three basic expression forms: the additive, the multiplicative and the exponential. These forms are then used to describe a schema for defining empirical models for small steps. It will be shown that we can derive small-step models from this schema which agree well with human analysis. Human data for assessing agreement are obtained by a survey of human analysis of step-size.

Discussion

A teacher checking a trigonometric proof would usually examine it for completeness, correctness of step and elegance. He is seldom conscious of the question of step-size since in practice students usually show their steps in sufficient detail. In fact in the opinion survey, many of the participants could not understand at first what we meant by a large step and a small step. Perhaps step-size is an artificial problem peculiar to the needs of our trigonometric proof checker; used to assess the acceptability of a proof.

Step-size may be considered as referring to the verifiability of a step. A step whose correctness is obvious or readily verified may be considered small. But what is obvious or readily verified is rather subjective, depending also on the teacher's own ability.

The teacher's assessment is influenced by other factors. His opinion of a student may influence his judgement. He may regard a step from one student as small and an identical step from another as large if he is not convinced that the latter can demonstrate it. The preceding steps of the proof may also influence him. Even the nature of the problem identity may be taken into account. A "trivial" one would be expected to be proved in greater detail than usual.

Although it is subjective, we believe there is a consensus of opinion on step-size. Indeed our survey supports this belief. There would be little basis for developing a proof-checker if teachers do not agree well among themselves.

While it may be desirable for a proof-checker to agree closely with human adjudication, closeness of agreement should not be an overriding factor. The possible harmful effect on the student of a "controversial" decision should be considered. One way of reducing this possibility is to ask the student to show his step in greater detail rather than to declare it large (he might not agree!).

Although incorrect steps can be classified for step-size, we shall confine step-size analysis to the correct steps only.

It was thought at one time that step-size might somehow be directly related to the C-set. Large steps could perhaps be associated with certain large C-sets. But such a possibility was soon dismissed when it was found that large steps can have small C-sets and vice versa. A good example is the following :

(1) $$\frac{1+\tan\theta-\sec\theta}{\sec\theta+\tan\theta-1} \to \frac{1+\sec\theta-\tan\theta}{\sec\theta+\tan\theta+1}$$

(2) $$\tan\theta+\csc\theta+\sec\theta-\sin^2\theta+\cot\theta+\sec^2\theta-1+\csc^2\theta+\cot\theta \to$$

$$\frac{\sin\theta}{\cos\theta} + \frac{1}{\sin\theta} + \frac{1}{\cos\theta} - (1-\cos^2\theta) + \frac{\cos\theta}{\sin\theta} + \tan^2\theta+\cot^2\theta+1+\frac{1}{\tan\theta}$$

The C-sets of (1) and (2) are respectively {I7} and {I1,I2,..I8}. Yet clearly (1) is large while (2) is small. It is interesting to note the large proportion of students who could not prove the identity associated with the step (1), although it requires only the substitution of I7.

The two steps above provide a vital insight into the nature of step-size. The first is not obvious as it stands and cannot be broken into substeps. The second, although involving all eight basic identities, can be easily followed because it can be resolved into the eight substeps: $\tan\theta\to\sin\theta/\cos\theta$, $\csc\theta\to 1/\sin\theta$, ... $\cot\theta\to 1/\tan\theta$, each of which is obvious. The following appears to be a reasonable description of the human analysis of step-size.

Each teacher has his own corpus of primitive steps. These primitive steps are those he regards as being obviously small. When a teacher is analysing a non-primitive step, he will attempt to resolve it into primitive substeps. It will be considered large if and only if such a resolution is not possible.

We shall develop a scheme for empirical models of small steps, based on the above conception of the human step-size analyser. As a preliminary we shall define the Φ-operations and use them to define the three basic expression forms mentioned previously.

Φ-Operations

The rational operations enable a very large number of distinct expressions to be derived from a given expression, in fact all the expressions A-equivalent to it. We wish to consider here a more restricted class, Φ, of rational operations, by excluding certain rational operations. These operations, called the Φ-operations, enable us to derive only a small, finite* number of Φ-derivatives from an expression. They enable us to :

(a) rearrange terms and factors in any order,

(b) add and subtract an arbitrary number of zeroes, and to multiply and divide by, an arbitrary number of 1's.

(c) remove or add redundant parentheses and many + or -

(d) collect common terms and factors and

(e) to add, subtract, multiply and divide (provided division is exact) integers.

Φ is essentially the set of rules of algebraic operation which follows from the field axioms, the definitions of subtraction, division and indices (powers) together with certain notations. The distributivity axiom is excluded, but arithmetic operations on integers and the collection of common terms and factors are permitted. It should be noted that addition and multiplication are inherent operations of a field. The absence of the distributivity axiom means that the

* Finite if we discount the trivially distinct expressions derived by the redundant or repeated use of parentheses, identity elements 0 and 1, and the unary + and - as in: $f,(f),((f)),(((f))),....,$ $+(f),+(+(f)),....,f*1,f*1*1,f*1*1+0,....,-(-f+0-0+0),....$.

important operations of expression expansion and factorisation are excluded from Φ. Thus it is not possible to derive by Φ-operations only, $a^2+2ab+b^2$ from $(a+b)^2$ or $ca+ab$ from $a(b+c)$.

To be more precise, we give below some of the basic rules of Φ. We donote expressions by $e,e_1,e_2,\ldots$, terms by $t,t_1,t_2,\ldots$, factors by $f,f_1,f_2,\ldots$ and integers by $n,m,n_1,n_2,\ldots$. The words "expressions", "terms" and "factors" are used here in the sense of the BNF definitions in Chapter IV. The rules are written as "$f \to g$" or "$f \leftrightarrow g$", where "$f \to g$" means that g is derived from f while "$f \leftrightarrow g$" implies the two rules "$f \to g$" and "$g \to f$". The rules written in the unidirectional form are meant to be strictly unidirectional. Thus with the rule "$f^m * f^n \to f^{m+n}$" we can derive f^{2+3} and f^5 from f^{3+2} but not f^{1+4}, f^{7-2}, $f^1 * f^4$ and f^7/f^2.

If g is derivable from f by Φ-operations then we shall denote this by $f \overset{\Phi}{\to} g$. $f \overset{\Phi}{\not\to} g$ indicates that g is not a Φ-derivative of f.

$t_1+t_2 \leftrightarrow t_2+t_1$ $\qquad$ $f_1*f_2 \leftrightarrow f_2*f_1$

$t_1+(t_2+t_3) \leftrightarrow (t_1+t_2)+t_3 \leftrightarrow t_1+t_2+t_3$

$f_1*(f_2*f_3) \leftrightarrow (f_1*f_2)*f_3 \leftrightarrow f_1*f_2*f_3$

$t+0 \leftrightarrow t$ $\qquad$ $f*1 \leftrightarrow f$

$0 \leftrightarrow -0$ $\qquad$ $0-t \leftrightarrow -t \leftrightarrow -(t)$

$(e) \leftrightarrow e \leftrightarrow +(e)$ $\qquad$ $t_1+(-t_2) \leftrightarrow t_1-t_2$

$-(-t) \leftrightarrow t$ $\qquad$ $-(t) \leftrightarrow -t$

$-(t_1+t_2+\ldots+t_n) \leftrightarrow -t_1-t_2-\ldots-t_n$

$f_1^m*f_2^m*\ldots*f_n^m \leftrightarrow (f_1*f_2*\ldots*f_n)^m$

$f' \leftrightarrow f$ $\qquad$ $f^{-n} \leftrightarrow 1/f^n$

$f_1/f_2 \leftrightarrow f_1*f_2^{-1}$	$f*f*...*f \rightarrow f^m$ (m-fold)
$f^m*f^n \rightarrow f^{m+n}$	$f^m/f^n \rightarrow f^{m-n}$
$t-t \rightarrow 0$	$f/f \rightarrow 1$ $(f \neq 0)$
$n_1*f+n_2*f+...+n_k*f \quad \rightarrow$	$(n_1+n_2+...+n_k)*f$

__Examples:__ (a,b,c,d denote factors, terms or expressions as appropriate)

(1) $3+2-4 \overset{\Phi}{\rightarrow} 2+3-4 \overset{\Phi}{\rightarrow} 5-4 \overset{\Phi}{\rightarrow} 1 \overset{\Phi}{\not\rightarrow} 5-4$

(2) $((a)) \overset{\Phi}{\rightarrow} a \overset{\Phi}{\rightarrow} -(-a) \overset{\Phi}{\rightarrow} +(a) \overset{\Phi}{\rightarrow} (a+0+0)^1$

(3) $a-(b-c+d) \overset{\Phi}{\rightarrow} a-d-(b-c)$

(4) $(a*a*b^3)^2 \overset{\Phi}{\rightarrow} (a^2*b^3)^2 \overset{\Phi}{\rightarrow} b^6*a^4$

(5) $(a+b)^3 \overset{\Phi}{\not\rightarrow} a^3+3a^2b+3ab^2+b^3$

(6) $(a+b)/c \overset{\Phi}{\not\rightarrow} a/c+b/c \overset{\Phi}{\not\rightarrow} (a+b)/c$

__Three Basic Forms__

__Defn:__ $\overset{n}{\underset{i=1}{(+)}} t_i$, also written as $t_1(+)t_2(+)...(+)t_n$ and $(+)\{t_i|i=1,2,...n\}$, stands for the expression $t_1+t_2+...+t_n$ or any of its Φ-derivatives. $\overset{n}{\underset{i=1}{(+)}} t_i$ is said to be in the __additive form__ and $t_1,t_2,...t_n$ are its __additive components__ or terms.

e.g. Let $t_1 = a*b/c$, $t_2 = (-2/b)$ and $t_3 = (-c*(b-c))$. Then $\overset{3}{\underset{i=1}{(+)}} t_i$ can be $(a*b/c)+(-2/b)+(-c*(b-c))$ or $-2/b+a*b/c-c*(b-c)$ or $b*a/c-2/b+c*(c-b)$.

__Defn:__ $\overset{n}{\underset{i=1}{(*)}} f_i$, also written as $f_1(*)f_2(*)...(*)f_n$ and $(*)\{f_i|i=1,...n\}$ stands for the expression $f_1*f_2*...*f_n$ or any expression Φ-derivable from it. $\overset{n}{\underset{i=1}{(*)}} f_i$ is said to be in the __multiplicative form__ and $f_1,f_2,...f_n$ are its __multiplicative__ components or factors.

e.g. Let $f_1 = (a+b)$ and $f_2 = (a-b)$. Then $\overset{2}{\underset{i=1}{(*)}} f_i$ can be $(a+b)*(a-b)$ or $(a-b)*(b+a)$ but not a^2-b^2.

Defn: $f(\uparrow)n$ stands for $f\uparrow n$ or any of its Φ-derivatives and is said to be in the exponential form, n being the exponent of f.
e.g. $(2*a^2/b)(\uparrow)3$ can be $2^3*a^{2*3}/b^3$ and $8*a^6/b^3$.

A Schema for Small Steps

We present here a method for formulating empirical models for small steps. A model in this formulation consists of specifications for defining a subset of the set of all correct steps, this subset containing all and only the small steps in the model. Each model contains a set of primitive small steps together with a set of rules for composing small steps from other small steps. These ideas may be formalised as follows :

Let Σ be the set of all correct steps,

Π a set of primitive small steps and

Ω a set of rules for the composition of steps.

Then the set of small steps in the model, $\Sigma^\circ(\Pi,\Omega)$, also written as Σ°, is defined by :

(1) $s\varepsilon\Pi \Rightarrow s\varepsilon\Sigma^\circ$

(2) $s\varepsilon\Sigma^\circ$ if s is derived from members of Σ° by rules of Ω

(3) $s\varepsilon\Sigma^\circ$ if and only if $s\varepsilon\Sigma^\circ$ by virtue of (1) and (2) only.

$\Sigma^\circ(\Pi,\Omega)$ does not, however, define a small-step model until the set Π of primitive steps and the set Ω of step-composition rules are specified. Until then it may be regarded as simply a schema for small-step models.

It can be seen that this formulation is based on our conception of the human analysis of step-size discussed earlier. The set Π corresponds to the corpus of primitive steps of a teacher and Ω is the set of rules he will allow the student to use for composing a small step from other small steps. Together Π and Ω generate the set Σ° of all the small steps. A teacher whose Σ° is a proper subset of the Σ° of another teacher is a stricter judge than the other.

By a careful choice of Π and Ω we can produce a good model for small steps, one which agrees well with human analysis. We have investigated two models, called Model-A and Model-B. Model-B is a modification of Model-A.

Model-A .. A Model for Small Steps

Model-A will be denoted by $\Sigma^\circ_A(\Pi,\Omega)$ or Σ°_A and the corresponding Π and Ω are subscripted with an 'A'. The specifications for Π_A and Ω_A are empirical, but they have been chosen to ensure a good model for small steps.

The Set Π_A

Π_A can be divided into two distinct classes: (1) the primitive A-steps and (2) the primitive T-steps. The primitive A-steps will be defined by their <u>component count</u>. N(e) the component count of an expression e is defined as :

(1) N(e)=0 if e is a constant or a trigonometric function.

(2) $$N(e\uparrow n)=\begin{cases} N(e) & \text{if e is a constant or if e is a trigonometric function and n=0,1 or 2.} \\ N(e)+1 & \text{otherwise} \end{cases}$$

(3) $$N(\overset{n}{\underset{i=1}{(+)}}e_i) = n+\sum_{i=1}^{n} N(e_i)$$

(4) $$N\Big(\overset{n}{\underset{i=1}{(*)}}e_i\Big) = n+\sum_{i=1}^{n} N(e_i)$$

e.g. $N(\sin\theta*(\sec\theta)\uparrow 2*(1+3*\cos\theta)\uparrow 3)$

$= 3 + N(\sin\theta) + N((\sec\theta)\uparrow 2) + N((1+3*\cos\theta)\uparrow 3)$

$= 3 + 0 + N(\sec\theta) + N((1+3*\cos\theta)) + 1$

$= 4 + N(\sec\theta) + N(1+3*\cos\theta)$

$= 4 + 0 + 2 + N(1) + N(3*\cos\theta)$

$= 4 + 0 + 2 + 0 + 2 + N(3) + N(\cos\theta)$

$= 4 + 0 + 2 + 0 + 2 + 0 + 0$

$= 8$

The component count of a step f → g is defined as $N(f)+N(g)$. $N(e)$ gives a rough indication of the size and complexity of the expression e. We define a primitive A-step f → g to be one with $N(f)+N(g)<k$. Whether or not an "optimal" value for k exists is not clear, but the closeness of agreement model-A has with human analysis is not very sensitive to variations in the value of k, provided it is neither too large nor too small. From our investigation, we have found 20 to be a good, although rather arbitrary, value for k. We shall take this to be our value of k.

e.g. (1) $\tan\theta\rightarrow\sin\theta/\cos\theta$ is a T-step and cannot therefore be a primitive A-step.

(2) $\sin\theta(1+\sin\theta)\rightarrow\sin\theta+\sin^2\theta$ is an A-step with a component count of 6, and is therefore a primitive A-step.

(3) $(1+\sec\theta)/(\sec\theta\tan\theta+\tan\theta+2\tan\theta-2\sin\theta)\rightarrow(1+\sec\theta)/(\sec\theta\tan\theta+3\tan\theta-2\sin\theta)$ is not a primitive A-step since it has a component count of 25 ($\geqslant 20$). (It is however a small step in Model-A.)

In Model-A, a primitive T-step must have a singleton C-set,* i.e. one containing only one reference identity. In a primitive T-step the reference identity must be substituted in its standard form (as in Table 4.1) or in one of its permitted variants outlined below. Let $E_\ell \to E_r$ be the reference identity substituted in the standard form. Then :

(1) f → g is a permitted variant if f-g or g-f is Φ'-derivable** from E_ℓ-E_r.

(2) f → g is a permitted variant if f/g is Φ'-derivable from f'/g' (g' ≠ 0), and f'→g' is itself a permitted variant.

(3) f → 0 is a permitted variant if f/g→0 is a permitted variant.

e.g. (1) tanθ+cotθ→1/(sinθ*cosθ) has the C-set {I3,I4,I5,I6} and cannot be a primitive T-step.

(2) 1/sinθ→cscθ, cscθ-1/sinθ→0, sinθ→1/cscθ, (cscθ∘sinθ-1)/cscθ→0 and cscθ∘sinθ-1→0 are primitive T-steps, being permitted variants of the standard form cscθ→1/sinθ.

(3) Permitted variants of $\sin^2\theta+\cos^2\theta\to 1$ include $\sin^2\theta-1\to-\cos^2\theta$ and $(1-\cos^2\theta)/\sin^2\theta\to 1$.

<u>The Set Ω_A</u>

Ω_A consists of the following rules for step-composition :

w_1 : from $e_i \to e'_i$ (i=1,2,..n) derive $\overset{n}{\underset{i=1}{(+)}} e_i \to \overset{n}{\underset{i=1}{(+)}} e'_i$;

w_2 : from $e_i \to e'_i$ (i=1,2,..n) derive $\overset{n}{\underset{i=1}{(*)}} e_i \to \overset{n}{\underset{i=1}{(*)}} e'_i$;

* *Here we include the supplementary single C-sets {J1}, {J2}, {J3} and {J4}.*

** *Φ' is the set of rules Φ with rationalisation included. Rationalisation refers to operations like a+b/c→(a*c+b)/c and a/b+c/d→(a*d+b*c)/(b*d).*

w_3 : from $e \rightarrow e'$ derive $e(\uparrow)m \rightarrow e'(\uparrow)m$ (m an integer);

w_4 : from $f(*)e_i \rightarrow g_i$ $(i=1,2,..n)$ derive $f(*)\ (\overset{n}{\underset{i=1}{(+)}} e_i) \rightarrow \overset{n}{\underset{i=1}{(+)}} g_i$;

w_5 : from $e \rightarrow e'$ derive $e' \rightarrow e$, $-\ e \rightarrow e'$, $e-e' \rightarrow 0$, $e/e' \rightarrow 1(e' \neq 0)$

and reciprocal of $e \rightarrow$ reciprocal of e'.

The rule w_1 in effect says that if each of the steps $e_i \rightarrow e_i'$ $(i=1,2,..n)$ is small then so is any step of the form $\overset{n}{\underset{i=1}{(+)}} e_i \rightarrow (+) e_i'$. To illustrate, if $1-\cos^2\theta \rightarrow \sin^2\theta$ and $\tan\theta \rightarrow \sin\theta\sec\theta$ are small then so are, say, $1-\cos^2\theta+\tan\theta \rightarrow \sin^2\theta+\sin\theta\sec\theta$ and $1+(\tan\theta-\cos^2\theta) \rightarrow \sin\theta\sec\theta+\sin^2\theta$, since they have the form $(1-\cos^2\theta)(+)\ \tan\theta \rightarrow \sin^2\theta(+)\sin\theta\sec\theta$. The interpretation of w_2, w_3 and w_5 should now be obvious. w_4 perhaps requires some elaboration and this is best done by an example.

Suppose $\sin\theta\sec\theta \rightarrow \tan\theta$ and $\sin\theta\sec^2\theta \rightarrow (1+\tan^2\theta)/\csc\theta$ are small. Then $\sin\theta(\sec\theta+\sec^2\theta) \rightarrow \tan\theta+(1+\tan^2\theta)/\csc\theta$ is small since it has the form $\sin\theta(*)(\sec\theta(+)(\sec^2\theta) \rightarrow \tan\theta(+)(1+\tan^2\theta)/\csc\theta$.
In the same way $(\tan\theta+\sec\theta)/\sec\theta \rightarrow \sin\theta+1$ may be regarded as being derived from $\tan\theta/\sec\theta \rightarrow \sin\theta$ and $\sec\theta/\sec\theta \rightarrow 1$ using the composition rule w_4.

By a composite application of the step-composition rules we can obtain small steps of considerable complexity from very simple ones. $(1+\sec\theta)/(\sec\theta\tan\theta-2\sin\theta-\tan\theta) \rightarrow (1+1/\cos\theta)/(1/\cos\theta*\sin\theta/\cos\theta-2\sin\theta-\sin\theta/\cos\theta)$ is a fairly complex step built up from the "irreducible" steps: $1 \rightarrow 1$, $\sec\theta \rightarrow 1/\cos\theta$, $\sec\theta \rightarrow 1/\cos\theta$, $\tan\theta \rightarrow \sin\theta/\cos\theta$, $2 \rightarrow 2$, $\sin\theta \rightarrow \sin\theta$ and $\tan\theta \rightarrow \sin\theta/\cos\theta$.

To test for the use of the step-composition rules w_1, w_2, w_3 and w_4 we need to decompose a step into its component substeps. We call this process substep resolution. Each of the rules has the corresponding resolution indicated below.

(1) Additive Resolution: If a step is expressible as $\overset{n}{\underset{i=1}{(+)}} e_i \rightarrow \overset{n}{\underset{i=1}{(+)}} e'_i$ with $e_i \equiv e'_i$ $(i=1,2,..n)$ then $e_i \rightarrow e'_i$ $(i=1,2,..n)$ are substeps.

(2) Multiplicative Resolution: If a step is expressible as $\overset{n}{\underset{i=1}{(*)}} e_i \rightarrow \overset{n}{\underset{i=1}{(*)}} e'_i$ with $e_i \equiv e'_i$ $(i=1,2,..n)$ then $e_i \rightarrow e'_i$ $(i=1,2,..n)$ are substeps.

(3) Exponential Resolution: If a step is expressible as $e(\uparrow)m \rightarrow e'(\uparrow)m$ with $e \equiv e'$ (or $e \equiv -e'$) then $e \rightarrow e'$ (or $e \rightarrow -e'$) is a substep.

(4) Factored Resolution: If a step is expressible as $f(*)\overset{n}{\underset{i=1}{(+)}} e_i \rightarrow \overset{n}{\underset{i=1}{(+)}} g_i$ with $f(*)e_i \equiv g_i$ $(i=1,2,..n)$ then $f(*)e_i \rightarrow g_i$ $(i=1,2,..n)$ are substeps.

In analysing a step there are two basic approaches that we can adopt. One is to check a step or substep for primitiveness (including the use of w_5) first and performing substep resolution only when this test fails. The rationalé here is to avoid unnecessary resolution on a step when it is already primitive. This is especially important for the primitive A-steps since they are often resolvable.

The second approach is to resolve a step as far as possible until every derived substep is irresolvable. These substeps are then tested for primitiveness. In this approach we economise on the tests for primitiveness. It should be noted that sometimes a step is not uniquely resolvable since at each stage of substep resolution more than one kind (multiplicative, additive, factored) of resolution may be possible.

With either approach we end up with two possibilities. Either (1) we have a resolution into primitive steps only or (2) it is not possible to resolve the step into primitive steps only. We conclude that the step is small if case (1) prevails and that the step is large in case (2).

Whether the first approach is preferable to the second or not depends on the type of expressions normally encountered as well as the relative costs of testing for primitiveness and performing substep resolution. However these two approaches represent extremes between which intermediate approaches are possible.

Model-A has not yet been implemented. However, its precursor, Model-O has already been written as a FORTRAN program called Super-2 and this is briefly described in Chapter VII. Model-O has much in common with Model-A, differing from it mainly in the following :

(1) all A-steps are primitive;

(2) factored resolution is not allowed;

(3) resolution is done at the primary level only (see Chapter VII);

(4) the criteria for primitive T-steps in Model-A are approximated by numeric ones (see description of Super-2).

Super-2 can be modified quite easily to implement Model-A, although the programming effort required would be substantial.

The criteria for primitive T-steps would be rather messy to implement because of the number of permitted variants to be considered.

The unwieldiness of the definition of primitive T-steps in Model-A led us to the alternative definition: *A T-step is primitive if and only if it has a singleton C-set and a component count not exceeding 5*. This definition will be very cheap to implement. It will also be found to be a good empirical criterion. We shall designate as *Model-B* this new model derived from Model-A by this alternative definition for primitive T-steps.

Adequacy of Model-A and Model-B

The adequacy of a given model can be gauged by comparing its analysis with the corresponding human analysis. The necessary comparative data can be generated by choosing (1) a sample of steps and (2) a sample of human judges who are required to express their opinions on the size of each sample step. The opinion data obtained in this way may be treated as a sample of human analysis to be compared with the corresponding analysis in the model.

Table C.3 shows the 207 steps we have chosen as our sample. These steps have been generated from the expressions contained in the sixteen proofs shown in the same table. Each proof $e_1 \to e_2 \to .. \to e_n$ generates the $\binom{n}{2}$ steps $e_i \to e_j$ $(1 \leqslant i < j \leqslant n)$. In the table, step $e_i \to e_j$ is shown as I→J.

Although we could have chosen our sample steps to reflect more realistically the kind of steps TPS would normally expect we have not done this. Such a sample might fail to show up adequately any serious weakness our models may have. We want our sample steps to cover a rich variety of step-sizes so that our models may be subjected to a severe test.

Our sample of human judges consists of fourteen students and teachers familiar with elementary trigonometry. They are designated HO-1, HO-2, .. HO-14. Table C.1 shows their expressed opinions on the sample steps with the entries '0' and '1' indicating 'small' and 'large' respectively. The same table also displays the corresponding data for six machine-related opinions which we designate MO-1, MO-2, .. MO-6. MO-1, MO-2 and MO-3 are from three variants of Model-O, and for them we enter algebraic steps as '2' rather than '0'.

The entries for MO-4 represent the analysis of Model-A performed by hand simulation. MO-5 does not represent any small step model. Its data are assigned by hand to ensure the best possible agreement with the available human data. Each entry is 0 or 1 whichever has the largest number of agreements with the corresponding human entries. The data for MO-6 are assigned at random.

The figures under AGREEMENT COUNT refer to the number of matches the entries of the MO-i (i=1,2,..6) have with the corresponding human entries. As an example MO-2 has an agreement count of 6 for step 34 while MO-5 has a count of 8.

A hand-simulation of Model-B has also been performed on the sample steps. The results have not been separately represented in the table because (for our sample steps) they happen to coincide with the analysis of Model-A. Thus MO-4 also represents Model-B.

Evaluating Agreement

One difficulty in evaluating the agreement between a machine opinion and the human opinions is the apparent lack of a standard method for interpreting the data. In view of this we used the intuitive index given by the percentage of agreement counts between a machine opinion and the human opinions.

MO VS HO AGREEMENT

	COUNT	PERCENT
MO-1	2275	78.50
MO-2	2333	80.50
MO-3	2305	79.54
MO-4	2607	89.96
MO-5	2655	91.61
MO-6	1337	46.14

HO VS HO AGREEMENT

	COUNT	PERCENT
HO- 1	2342	87.03
HO- 2	2332	86.66
HO- 3	2300	85.47
HO- 4	2334	86.73
HO- 5	2366	87.92
HO- 6	2388	88.74
HO- 7	2392	88.89
HO- 8	2410	89.56
HO- 9	2386	88.67
HO-10	2242	83.31
HO-11	2396	89.04
HO-12	2410	89.56
HO-13	2382	88.52
HO-14	2360	87.70

MEAN = 87.70   STD DEV = 1.74

Table 5.1   Agreement Percentages

Table 5.1 gives the total agreement count of each machine opinion (under "MO VS HO AGREEMENT") as well as the corresponding percentage agreement on the base of 2898 (=14x207). The table also gives (under "HO VS HO AGREEMENT") the agreement count and percentage of each HO. Here the count is taken over the matches each HO enjoys with the remaining thirteen HOs and the percentage is computed on the base of 2691 (=13x207). The HO-VS-HO agreement percentage has the average value 87.70 ± 1.74%. The small standard deviation indicates a good agreement among the human judges, a state of affairs we expect.

The percentage agreement 46.14% for MO-6 gives the figure for purely random data, and serves as a useful "lower bound" for the agreement percentage. The figure 91.61% for MO-5 gives the "best possible" figure for the given human data. The value for MO-4 is 89.96% is very close to the highest possible value and is better than the best figure for the HOs, viz. 89.56% for HO-12. It is also higher than the value 87.70 ± 1.74%. The figures for MO-1, MO-2 and MO-3, though lower than for MO-4 and the HO-VS-HO values, are respectably higher than the one for the random data MO-6. The "poor" agreement for MO-1, MO-2 and MO-3 are mainly because they are more "forgiving" especially in treating all A-steps as small.

Because we do not know the underlying statistical behaviour of our opinion data we have not been able to make precise statements on them. We could not say that the agreement between MO-4 and the human judges is significantly better than that which exists between HO-12 and the remaining judges. Nevertheless because of the size of our sample (207 steps and 14 judges) we feel justified in claiming that Model-A and Model-B agree well with the human judges, and in fact better than they do among themselves. Thus Model-A and Model-B are indeed adequate models for small steps.

Chapter VI

Assisting Proof Construction

In this chapter we shall consider how we can assist the student in his proof. We shall regard as a form of assistance any information that can guide him to construct an acceptable proof.

A major source of assistance already exists in the proof-checking process. In informing the student that his step is incorrect TPS enables him to locate his error quickly and also prevents him from propagating the error. The proof-checker can also be modified to provide an identity-checking service. Whenever the student is uncertain about an identity, such as a basic one, he can enter it for verification.

There are more direct forms of assistance which we shall discuss shortly. But first we introduce a few more terms. We have the usual initial and target expressions e_o and e_n and a proof for it has the form $e_o \overset{*}{\rightarrow} e_n$. Let $e_o \overset{*}{\rightarrow} e_i$ be the current stage of a proof. Then we have $e_i \rightarrow e_n$ as the current <u>residual step</u> or <u>residual</u>. We shall denote the j^{th} residual $e_j \rightarrow e_n$ by <u>rsd-j</u>. The aim of a proof can be considered to be one of whittling down the current residual into one which is a small step.

TPS can be designed to render the following forms of assistance :-

(1) To present the student with a set of basic identities

which are relevant for completing his proof. This set can be the C-set of the current residual or an appropriate subset of it. If the C-set is empty then the student may be advised that no further substitutions are required.

(2) To provide the student with a "next-step" $e_i \rightarrow e_{i+1}$ to enable him to continue. To be useful this next-step must bring him closer to the target expression e_n.

(3) To locate the incorrect substeps in a step. This can be useful when the step itself is rather large and complex and the incorrect parts are not obvious from inspection. This can be effected with the help of substep resolution.

(4) To suggest a promising proof strategy. As a case in point this is useful when the student cannot get started. This often happens when the direct derivation of one side from the other does not appear to be feasible. On the other hand the student may also be discouraged from pursuing any further a strategy that is not promising.

(5) To give the student timely warning that his proof is "going astray". This happens when the last expressions derived .. e_{i-2}, e_{i-1}, e_i are getting further and further away from the target expression.

(6) To complete the student's partial proof from where he gives up. This may be preferable to giving him an entirely fresh proof in that he has the benefit of seeing how he could have completed the proof himself.

In trying to provide the above forms of assistance TPS faces one major problem. This is the lack of a sensitive "measure" of the "distance" between two expressions. This makes it rather difficult to automatically construct a next step, to determine whether the proof is straying - in fact makes it difficult to generate a proof automatically. Much of this difficulty can be overcome with the aid of a companion proof $E_o \overset{*}{\rightarrow} E_m$ for the problem identity $e_o \equiv e_n$. We assume that E_o and e_o are identical as are E_m and e_n. This means that we are assuming that it is of the same strategy as the student's proof. We also assume that it is a good proof in that $E_o, E_1, E_2, \ldots$ are progressively closer to E_m.

By comparing e_i against $E_o, E_1, E_2, \ldots$ in turn (we can start with an expression later than E_o if we already know the stage the student's proof last "reached") we can find one, E_j say, which is not the same as e_i but which is closest to it on the target expression side of e_i. Then useful basic identities for the student may be taken as the C-set of $e_i \rightarrow E_j$, or of $e_i \rightarrow E_{j+1}, \ldots$ or of $e_i \rightarrow E_m$ whichever is most appropriate. If the current residual $e_i \rightarrow E_m$ has an empty C-set then it is algebraic. If $e_i \rightarrow E_j$ is a small step, then it can be presented to the student as a next step. If it is not small then it is still nearer from e_i than the target expression and the student may be prompted to derive E_j first, thus bringing him a stage closer to E_m. If the student abandons his proof then TPS may complete it as $e_i \rightarrow E_j \rightarrow E_{j+1} \rightarrow \ldots \rightarrow E_m$, filling in an intermediate step or two should

$e_i \to E_j$ be large.

If the student's proof is diverging from the target expression, then e_i will be further away from it than e_{i-1} and/or e_{i-2} and/or e_{i-2} and so on. However because we lack a sensitive measure for distance, we may fail to discover the divergence solely by comparing $e_i, e_{i-1}, e_{i-2}, \ldots$ with E_m since when two expressions are sufficiently remote from a third, it might be difficult to find out which of them is nearer to the third. Therefore the comparison may have to be made against some other intermediate expressions of the companion proof.

One way of making a companion proof available is to store it with the problem identity. An obvious disadvantage here is that TPS must confine its assistance to those problem identities for which comparison proofs have been prepared. Another disadvantage is that the student may use a strategy different from that of a companion proof; it is not easy to modify a companion proof to suit the student's strategy. Because of these objections we propose instead to incorporate an automatic proof construction capability into TPS. This would be more flexible, and powerful and obviates any need to restrict the student's proofs.

Problem-Solving Systems

Automatic problem-solving and theorem-proving are already important areas of investigation in artificial intelligence. In theorem-proving most of the effort has been theoretical in nature. The works in problem-solving have been more practical. Examples can be found in Newell et al. [51], Slagle [66] and Quinlan and Hunt [57].

Surprisingly little work has been done in the area of trigonometry, possibly because the problem appears to be trivial. Johnson and Holden [35] and Newell et al. [51] have dealt briefly with the subject. In both cases trigonometry has been only an area of application for a more general system. We shall discuss only the work of Johnson and Holden because they have treated the subject more fully.

In Johnson and Holden's system an identity is converted into a standard form before it is proved. This makes it very much easier to prove. In the standard form, trigonometric functions are ordered, parentheses removed, negative terms transposed and denominators eliminated by rationalisation and cross-multiplication. As an example the standard form for $\frac{\tan\theta-\sin\theta}{1-\cos\theta} \equiv 1+\sec\theta$ is $\sin\theta\tan\theta+\cos\theta+\cos\theta\sec\theta \equiv \sec\theta+1$.

This approach is however unsuitable for TPS because the proofs produced are stereotype. Johnson and Holden were mainly concerned with establishing the correctness of an identity, and not the elegance of its proof. But the correctness of an identity can be determined very trivially in TPS using our numeric test. In fact Johnson and Holden could have simplified their task even further by expressing all trigonometric functions in terms of sinθ and cosθ. In this way the only substitution that may be required would be for $\sin^2\theta+\cos^2\theta \equiv 1$.

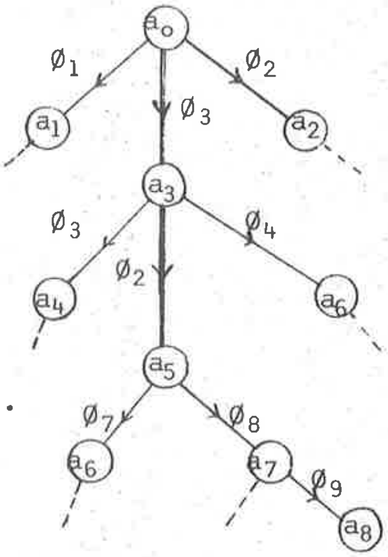
Most problem-solving systems adopt a heuristic search approach. A heuristic is a problem-solving strategy with a good chance of success but success is not guaranteed. It involves trial-and-error search of profitable paths.

Ernst and Newell [20] regard a heuristic search as consisting of operators and objects. An operator may or may not be applicable to a given object. Applied to an object it produces another object. A heuristic search problem has the form:

Given: an initial situation represented as an object,
a desired situation represented as an object,
a set of operators.

To Find: a sequence of operators that will transform the initial situation into the desired situation.

To solve a problem is to search the solution tree defined by the initial situation and the operators for a path from the initial situation to the goal situation. For instance if the desired situation is a_8 and the initial situation a_o and $\emptyset_1, \emptyset_2, \emptyset_3, ..$ are operators (see figure on the right) then the sequence of operators



$\emptyset_3, \emptyset_2, \emptyset_8, \emptyset_9$ is a solution since $\emptyset_9(\emptyset_8(\emptyset_2(\emptyset_3(a_o)))) = a_8$. Applied to our problem the objects are trigonometric expressions while the operators are the various rational algebraic operations and trigonometric substitutions.

An important requirement in heuristic problem-solving is a criterion for solution progress so that the search can be guided efficiently and effectively. GPS [20] employs means-ends analysis to guide the tree search. This is done by taking the difference between what is given and what is desired, the difference indicating some discrepancy to be removed. The problem may also be broken down into simpler subproblems. Differences are removed progressively with the aid of a table-of-connections which associates with each difference type a list of operators capable of reducing the difference.

Heuristic Generation of Trigonometric Proofs

To develop a heuristic generator of trigonometric proofs, we must first of all decide on our choice of indicators of expression differences. This choice determines our criteria for proof progress as well as our choice of operators for removing the differences indicated. Preferably the types of differences chosen should be :

(1) sensitive to changes in the difference; and

(2) simple and cheap to compute.

We propose the following indicators of difference between two expressions f and g :

(1) the resolvability of $f \to g$ into substeps,

(2) the C-set of $f \to g$,

(3) the primitiveness of $f \to g$,

(4) the presence of a latent structure (q.v.) in $f \to g$.

We shall now examine our criteria for progress. Let $e_i \to e_n$ be the current residual and $e_{i+1} \to e_n$ the next residual. The new step $e_i \to e_{i+1}$ will be regarded as having made progress if any of the following happens :

(1) $e_i \to e_n$ is not resolvable but $e_{i+1} \to e_n$ is,

(2) the C-set of $e_{i+1} \to e_n$ is a proper subset of the C-set of $e_i \to e_n$,

(3) $e_i \to e_n$ is not primitive but $e_{i+1} \to e_n$ is,

(4) $e_i \to e_n$ has no latent structure but $e_{i+1} \to e_n$ has.

Unlike proof-checking, in which we need to be concessive, we will have a stricter requirement for primitiveness. It will be similar to that in Model-B except that for a T-step to be primitive, its component count must be less than 5, while primitive A-steps should not have a count exceeding 10.

The above criteria reflects our conception of the proof process as one of removing the differences that exist between two equivalent expressions. There are two kinds of differences, algebraic or A-differences and trigonometric or T-differences. The T-differences are represented by the C-set of the expressions. The A-differences refer to structural dissimilarity. When a step is resolvable it indicates some structural similarity that can be used to break it into smaller substeps. A latent structure can be used to guide substitution.

To effect the above forms of proof progress we have the following classes of operations :

(1) substep resolution,

(2) C-set reduction,

(3) algebraic manipulation, and

(4) latent structure probing.

Substep Resolution

If the current residual $e_i \rightarrow e_n$ is resolvable, then the problem of proving $e_i \equiv e_n$ can be converted into the problem of proving the "subidentities" corresponding to the substeps. This latter problem is normally very much simpler than the original one. For this reason we shall treat resolution as one of the most important heuristics for proof generation.

An irresolvable step may become resolvable after suitable algebraic manipulation and/or trigonometric substitution. The four types of resolution may be tried in turn as appropriate and as required. The practical problems of resolution are considered in the next chapter.

C-set Reduction

This is done by substituting for basic identities belonging to the C-set of the current residual. The substitution process is not an easy one. Various algebraic manipulations may have to be attempted first before substitution is possible. This is because the functions, corresponding to an identity to be substituted for, may occur in a form not favourable for substitution. As an example we can substitute for I7 in $(\sec^2\theta-1)/\sin\theta \equiv \tan^2\theta/\sin\theta$ but not in $\frac{1+\tan\theta-\sec\theta}{\sec\theta+\tan\theta-1} \equiv \frac{1+\sec\theta-\tan\theta}{\sec\theta+\tan\theta+1}$ as it stands. Even when a favourable situation exists, there may be several alternatives for substitution and many trial-and-error efforts may have to be made before the basic identity being substituted for can be removed from the C-set. Even when a substitution is done correctly, C-set reduction may not occur. This happens when further substitutions for the same identity are required, often in conjunction with some algebraic manipulation.

To expedite the C-set reduction process we suggest the following guidelines :-

(1) Attempt the elimination of one basic identity at a time.

(2) Attempt removing the identities in the order I1,I2,I3,.. I7,I8. This means trying for the non-Pythagorean ones first - these are simpler to eliminate than the Pythagoreans. It should be understood that we try only for members of the C-set.

(3) To remove I1, replace $\csc\theta$ by $1/\sin\theta$ rather than $\sin\theta$ by $1/\csc\theta$. Similar remarks for I2,I3,I4 and I5. For last two, replace $\tan\theta$ by $\sin\theta/\cos\theta$ and $\cot\theta$ by $\cos\theta/\sin\theta$ consistently.

(4) When C-set contains I3,I4,I5, and they have not been removed individually, then remove them together by substituting for I4 and I5 only.

(5) If a step is sparse* with respect to the C-set express all functions in terms of sinθ and cosθ.

Algebraic Manipulation

As we have mentioned earlier, algebraic manipulation (AM) may make a step resolvable or favourable for substitution. For our work, we restrict our AM to a small repertoire of algebraic operations such as the simplification, rationalisation, simple factorisation and expansion of expressions. We have chosen only those useful operations which are relatively cheap to apply. Thus although a much more general factorisation capability than we have would be very useful for substep resolution, we have not included it because it is relatively costly to implement.

Our repertoire of algebraic operations will be the following :

AM-1: Simplification

Reducing the size of an expression by collecting common terms and factors, and removing zero terms and unit factors and powers.

e.g. $a-bc+2a \rightarrow 3a-bc$

$a+b-a \rightarrow b$

$ax/(ay) \rightarrow x/y$

$\frac{a.b^2.2a}{b} \rightarrow 2a^2b$

* *A step is sparse with respect to its C-set if there is a function in the C-set absent in the step, e.g. cscθ/(cotθ+tanθ)→cosθ does not contain the function secθ, which occurs in its C-set {I2,I3,I7,I8}.*

AM-2: Rationalisation

Reducing a sum of terms into a ratio of two polynomial expressions.

e.g. $\frac{a}{2b} - \frac{c}{d} + e \rightarrow \frac{ad-2bc+2bde}{2bd}$

AM-3: Unrationalisation

Converse of rationalisation.

e.g. $\frac{cd-bc+(2-e)}{d} \rightarrow \frac{cd}{d} - \frac{bc}{d} + \frac{(2-e)}{d}$

AM-4: Levelling

Converting a multi-level expression into a single or two-level one.

e.g. $\frac{(a+b/c)}{\frac{c+d}{pq}} \rightarrow \frac{(ac+b)pq}{(c+d)c}$

AM-5: Simple Factorisation

Factorisation based on any of the following formulae :

$a^{2n}-b^{2n} \rightarrow (a^n-b^n)(a^n+b^n)$ (n is an integer, omitted if 1)

$a^{3n}-b^{3n} \rightarrow (a^n-b^n)(a^{2n}+a^nb^n+b^{2n})$

$a^{3n}+b^{3n} \rightarrow (a^n+b^n)(a^{2n}-a^nb^n+b^{2n})$

$\overset{k}{\underset{i=1}{(+)}}[a(*)b_i] \rightarrow a(*)[\overset{k}{\underset{i=1}{(+)}}b_i]$

AM-6: Guided Factorisation

Here we try to derive a factor in one expression of a step, which is already present in the other expression. Whether or not a given expression is a factor can be determined by the remainder theorem (again using a numeric test).

e.g. In the step $\sin^3\theta-\cos^3\theta \rightarrow (\sin\theta-\cos\theta)(1+\sin\theta\cos\theta)$,

$(\sin\theta-\cos\theta)$ is a factor to test for in $\sin^3\theta-\cos^3\theta$.

AM-7: Expansion

A single-level expansion of expressions of the form $((\overset{k}{\underset{i=1}{(+)}})a_i)*((\overset{\ell}{\underset{j=1}{(+)}})b_j)$ and $((\overset{k}{\underset{i=1}{(+)}})t_i)^n$ and distribution of powers in $((\overset{k}{\underset{i=1}{(*)}})f_i)^n$ $(\rightarrow$ $\overset{k}{\underset{i=1}{(*)}}f_i^n)$.

e.g. $(a+b)(c-d) \rightarrow ac-ad+bc-bd$

$(ab^2/c)^3 \rightarrow a^3b^6/c^3$

AM-8: Addition of Identity

This can be roughly represented as :

(1) $e \rightarrow e+a-a$

(2) $e \rightarrow e*a/a \quad (a \neq 0)$

The 'a' above is not arbitrary, but may be a term or a factor in the target expression.

e.g.	e	a	result
	csc-cot	csc+cot	$\frac{(csc-cot)*(csc+cot)}{(csc+cot)} \equiv \frac{1}{csc+cot}$
	$\frac{1}{sec-tan}$	sec+tan	$\frac{(sec+tan)}{(sec-tan)(sec+tan)} \equiv (sec+tan)$

Latent Structure Probing

C-set determination enables a proof constructor to find the basic identities required to prove an identity. This gives it a very good measure of look-ahead. But even so, the task of proving the identity is in general not trivial. It is difficult, merely by inspecting the identity, to predict a sequence of algebraic transformations and trigonometric substitutions that will constitute a proof. However, for an identity whose C-set is singleton, it is sometimes possible to predict a useful form into which it can be converted.

As a case in point, $\cot^4\theta-\csc^4\theta \equiv 1-2\csc^2\theta$, which has the singleton C-set {I8}, is expressible in the form: $f-(\csc^2\theta-1)^2 \equiv f-(\cot^2\theta)^2$. The f can be shown to be $\cot^4\theta-2\csc^2\theta+1$ in this case. This shows that the identity can be proved by deriving $f-(\csc^2\theta-1)^2$ from $\cot^4\theta-\csc^4\theta$ via a sequence of A-steps, adding the known T-step $f-(\csc^2\theta-1)^2 \rightarrow f-(\cot^2\theta)^2$, and then deriving $1-2\csc^2\theta$ from $f-(\cot^2\theta)^2$ via another sequence of A-steps. Note that $\csc^2\theta-1 \equiv \cot^2\theta$ implied in the above form is a permitted variant of I8. The above form is a special case of the <u>structure</u> $f\pm g_\ell^n \equiv f\pm g_r^n$ where n is some positive integer and $g_\ell \equiv g_r$ is a permitted variant of a reference identity. We have found the following eight structures to be useful; f is an arbitrary expression and $g_\ell \equiv g_r$ is a permitted variant.

(1) $\pm g_\ell : \pm g_r$

(2) $\pm ng_\ell : \pm ng_r$

(3) $\pm g_\ell^n : \pm g_r^n$

(4) $f*g_\ell : f*g_r$

(5) $f*g_\ell^n : f*g_r^n$

(6) $f\pm g_\ell : f\pm g_r$

(7) $f\pm g_\ell^n : f\pm g_r^n$

(8) $f\pm ng_\ell : f\pm ng_r$

The structure (1) is a special case of (2) and of (3), just as (6) is a special case of (7) and (8). In the structures (1),(2),(3) and (7), g_ℓ and g_r are determined uniquely. But in (4) and (5) they are determined up to g'_ℓ and g'_r where $g'_\ell/g'_r \equiv g_\ell/g_r$ $(g_r \neq 0)$* and in (6) and (8), they are

* *Proof: Suppose* $g'_\ell \equiv kg_\ell$ *and* $g'_r \equiv kg_r$ *thus satisfying:* $g'_\ell/g'_r \equiv g_\ell/g_r$. *Then* $f*g'^n_\ell:f*g'^n_r$ *is still of the form* $\check{f}*g^n:\check{f}*g^n_r$ *where* $\check{f} \equiv k^n*f$. *(4) is a special case of (5) in which n=1.*

determined up to g'_ℓ and g'_r where $g'_\ell - g'_r \equiv g_\ell - g_r$.*

An identity which can be transformed into one of these structures is said to possess a <u>latent structure</u>. Thus $\cot^4\theta - \csc^4\theta \equiv 1 - 2\csc^2\theta$ has a latent structure. Whether an identity, with a singleton C-set, has a latent structure or not can be found by simple numeric tests.

The rationalé behind latent structures can be briefly explained as follows. Suppose identity $f_\ell \equiv f_r$ has a singleton C-set corresponding to the reference identity $r \equiv 0$. In the X-form, Thm 3 tells us that $f_\ell - f_r$ is related to r. However this relation is too general. Suppose $g_\ell \equiv g_r$ is a permitted variant of $r \equiv 0$, and let f be an arbitrary expression and n a positive integer. Then it may be possible to express $f_\ell : f_r$ <u>in terms of</u> g_ℓ and g_r in one of the structures above.

Let u_1, u_2, v_1 and v_2 be the values of $f^\circ_\ell, f^\circ_r, g^\circ_\ell$ and g°_r computed at an arbitrary point. Then $f_\ell : f_r$ has one of the structures shown in Table 6.1, if its values satisfy the corresponding numeric test indicated.

To probe an identity $f_\ell \equiv f_r$ for a latent structure, it must first be verified to have a singleton C-set. If the series of tests in Table 6.1 fails for a given permitted

* *Proof: Suppose* $g'_\ell - g'_r \equiv g_\ell - g_r$. *Then* $f \pm ng'_\ell : f \pm ng'_r$ *is also of the form* $\tilde{f} \pm ng_\ell : \tilde{f} \pm ng_r$ *where* $\tilde{f} \equiv f \pm n(g'_\ell - g_\ell) \equiv f \pm n(g'_r - g_r)$. *n=1 gives the case (6).*

variant, then it may be repeated for another. In each series of tests, the n may be varied to take any desired value, in practice n=2,3,4,..9 would be quite adequate. If all possible tests for $f_\ell:f_r$ fail, then they may be repeated for $f_r:f_\ell$ and for $1/f_\ell:1/f_r$. However for certain tests, a repetition for $1/f_\ell:1/f_r$ would be redundant. Similarly for some structures, there is no need to test for every permitted variant. As an example, for the structure $f*g_\ell:f*g_r$, it would be useless to repeat a test for permitted variants which are multiples of $g_\ell:g_r$.

Structure	Numeric Test
[1]$\pm g_\ell:\pm g_r$	$u_1=\pm v_1$ & $u_2=\pm v_2$ (i.e. $u_1=v_1$ & $u_2=v_2$ for the structure $g_\ell:g_r$ and $u_1=-v_1$ & $u_2=-v_2$ for $-g_\ell:-g_r$)
[2]$\pm ng_\ell:\pm ng_r$	$u_1=\pm nv_1$ & $u_2=\pm nv_2$ (for given n)
[3]$\pm g_\ell^n:\pm g_r^n$	$u_1=\pm v_1^n$ & $u_2=\pm v_2^n$
[4]$f*g_\ell:f*g_r$	$u_1/u_2=v_1/v_2$(provided $u_2\neq 0$) and [1] not satisfied
[5]$f*g_\ell^n:f*g_r^n$	$u_1/u_2=(v_1/v_2)^n$ (provided $u_2\neq 0$) and [3] not satisfied
[6]$f\pm g_\ell:f\pm g_r$	$u_1-u_2=\pm(v_1-v_2)$ and [1] not satisfied
[7]$f\pm g_\ell^n:f\pm g_r^n$	$u_1-u_2=\pm(v_1^n-v_2^n)$ and [3] not satisfied
[8]$f\pm ng_\ell:f\pm ng_r$	$u_1-u_2=\pm(nv_1-nv_2)$ and [2] not satisfied

Table 6.1 Test for Latent Structures

It is clear that latent structures give a proof constructor an additional measure of look-ahead, enabling it to perform goal-oriented algebraic manipulations.

Table 6.2 contains several examples of latent structures. Table C . 4 in Appendix C shows the results of a latent structure analysis on the sample steps. Here the probes are performed on the derived substeps, or on the steps if they are not resolved. It can be seen that most of the structures are of type (1). This is because the substeps are often the permitted variants of the reference identities.

Step	Detected Structure		Actual Structure
$(\mathrm{sin-cos})^2 \rightarrow 1-2\mathrm{sin.cos}$	(6)	$f+(\sin^2+\cos^2) \rightarrow f+1$	$-2.\mathrm{sin.cos}+(\sin^2+\cos^2)$ $\rightarrow -2\mathrm{sin.cos}+1$
$\cos^2 * \sec^2 \rightarrow 1$	(3)	$(\cos * \sec)^2 \rightarrow 1^2$	$(\cos * \sec)^2 \rightarrow 1^2$
$\sec^4-\sec^2 \rightarrow \tan^4-\tan^2$	-	unclassified	-
$(\tan^2+1)^2-\sec^2 \rightarrow \tan^4+\tan^2$	(6)	$f-(\sec^2-\tan^2-1) \rightarrow f-0$	$(\tan^4+\tan^2)-(\sec^2-\tan^2-1)$ $\rightarrow \tan^4+\tan^2$
$\frac{1}{\mathrm{sec-tan}} \rightarrow \mathrm{sec+tan}$	(4)	$f*1 \rightarrow f*(\sec^2-\tan^2)$ (equiv. to $\frac{g}{\sec^2-\tan^2} \rightarrow \frac{g}{1}$)	$\frac{1}{\mathrm{sec-tan}} \rightarrow \frac{\sec^2-\tan^2}{\mathrm{sec-tan}}$
$\mathrm{sec.tan}-2\sin-\tan \rightarrow$ $(1+\sec)(\tan-2\sin)$	(8)	$f-2\tan \rightarrow f-2\mathrm{sin.sec}$	$(\mathrm{sec.tan}+\tan-2\sin)-2\tan$ $\rightarrow (\mathrm{sec.tan}+\tan-2\sin)-$ $2\mathrm{sin.sec}$
$\cot^4-\csc^4 \rightarrow 1-2\csc^2$	(7)	$f-(\csc^2-1)^2 \rightarrow f-(\cot^2)^2$	$(\cot^4-2\csc^2+1)-(\csc^2-1)^2$ $\rightarrow (\cot^4-2\csc^2+1)-(\cot^2)^2$

Table 6.2 Examples of Latent Structure

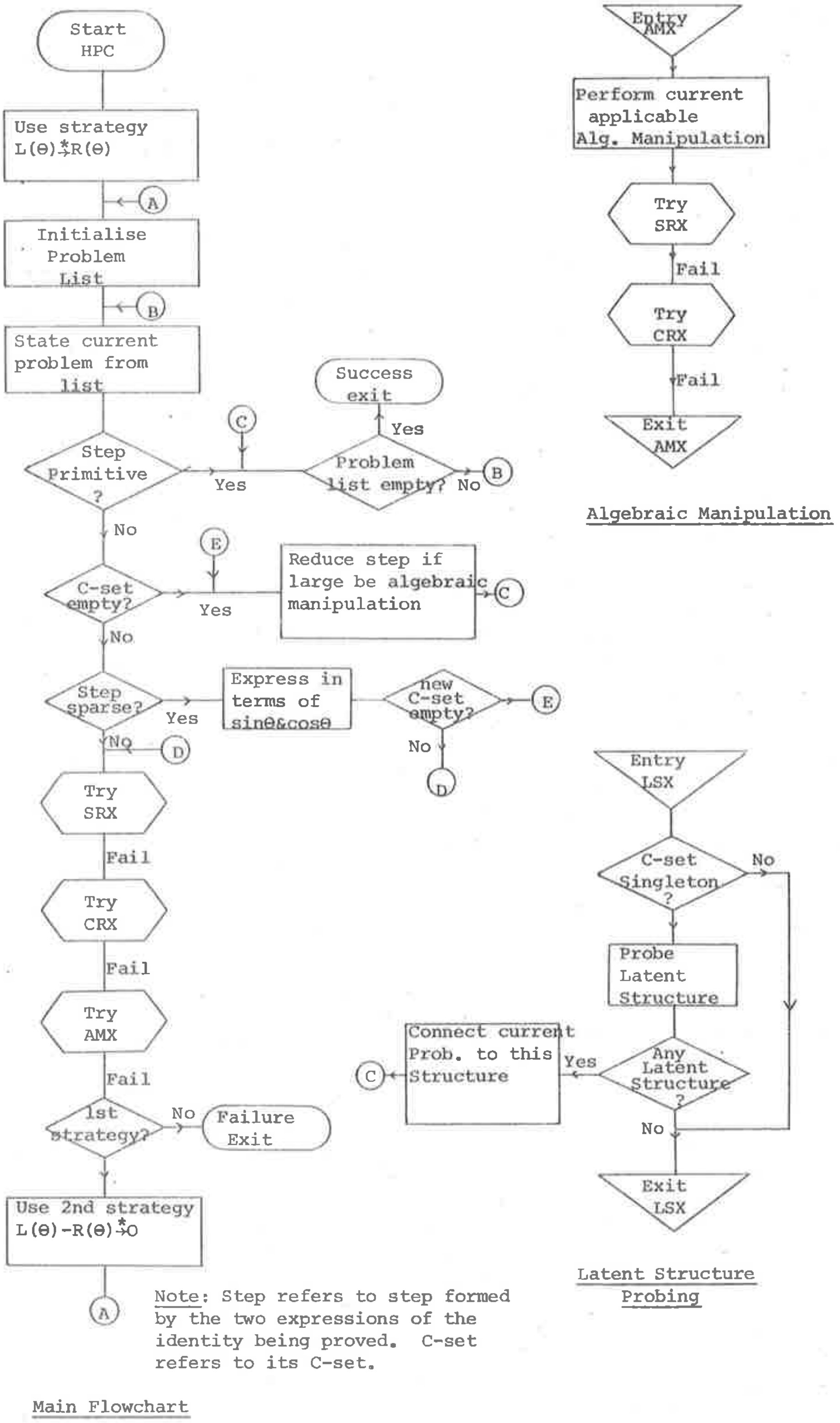
A Heuristic Proof Constructor

The flowchart in Figure 6.1 shows the general set-up of a heuristic proof constructor (HPC). The formulation is an ad hoc one and needless to say there are other ways of organising our various heuristics into a proof generator. But our immediate aim is to demonstrate the feasibility of constructing a proof generator using our heuristics.

HPC maintains an identity list which starts off with the identity to be proved. Let $L(\theta) \equiv R(\theta)$ be this problem identity. The proof strategy chosen is $L(\theta) \overset{*}{\rightarrow} R(\theta)$, or equivalently $R(\theta) \overset{*}{\rightarrow} L(\theta)$ since the proof will be essentially bidirectional in the construction.

If $L(\theta) \rightarrow R(\theta)$ is already primitive then there is nothing to prove. If it is a large A-step then the proof will consist of algebraic manipulations to reduce the step size. If it is a sparse step then the trigonometric functions will all be converted into their $\sin\theta, \cos\theta$ equivalents.

HPC continually tries to decompose the identity into subidentities. In fact substep resolution is given top priority in HPC. If an identity is resolvable into subidentities, then it is deactivated and the subidentities appended to the list to be proved in turn. Although subidentities are proved individually the proof displayed to the student, or any part of it, will have the usual form. This is done by combining the various substeps of a step.



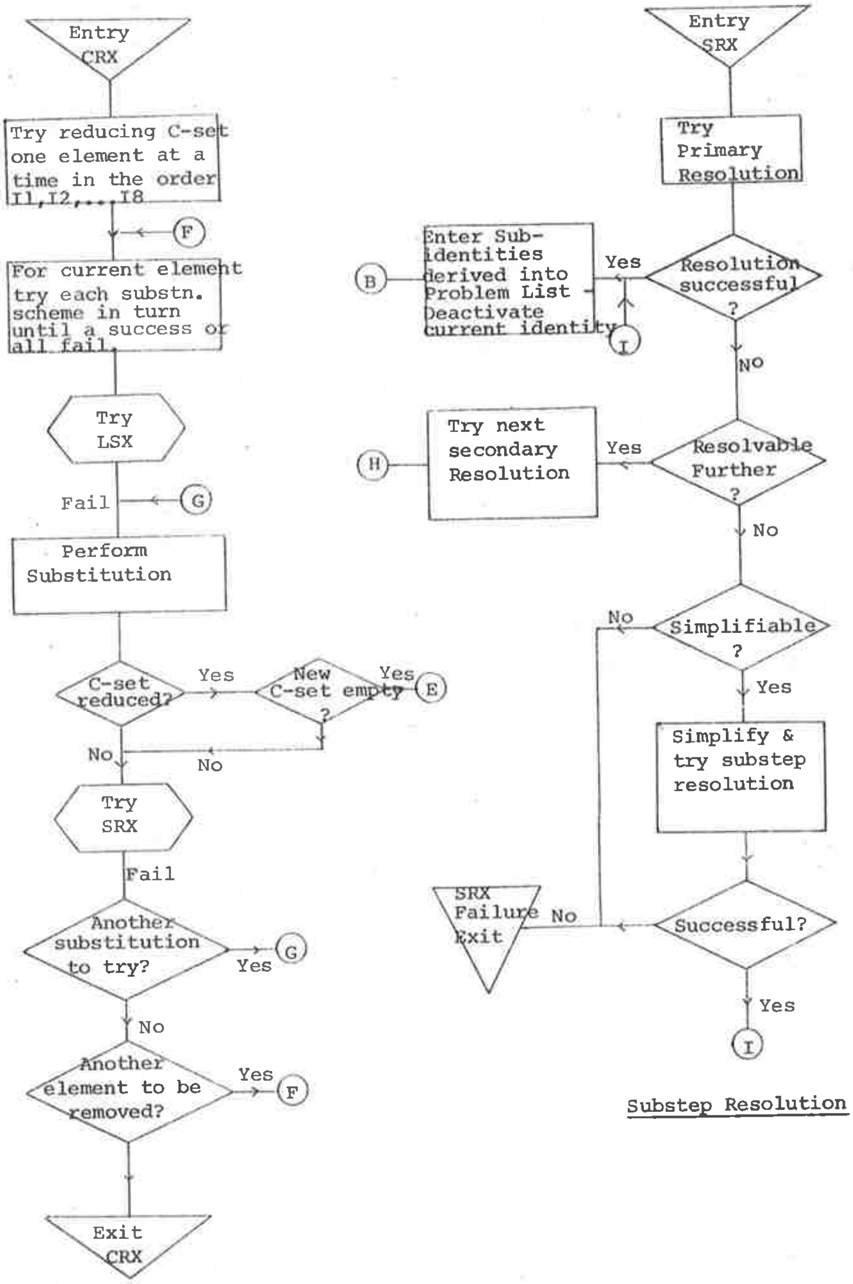
Note: Step refers to step formed by the two expressions of the identity being proved. C-set refers to its C-set.

Main Flowchart

Algebraic Manipulation

Latent Structure Probing

Figure 6.1 Heuristic Proof Constructor



C-set Reduction

Substep Resolution

Figure 6.1 continued

Overlapping operations are prohibited to avoid large steps.

The resolution is carried out in stages if necessary, starting with a primary one (see Chapter VII). When a given stage of resolution is unsuccessful another stage is attempted. The reason for resolving in stages is to take advantage of the way in which subexpressions occur. Often a primary resolution has as much chance of succeeding as the complete resolution. A complete resolution is on the average much costlier to perform than a primary one.

C-set reduction and latent structure probing can be carried out as explained earlier. For algebraic manipulation each applicable AM may have to be attempted in turn, each such AM, e.g. factorisation or simplification, being immediately followed by an attempt at resolution and C-set reduction.

The original identity is proved when the identity list is empty, i.e. has no active entries. If the list is not empty then each entry must be proved in turn. If the list cannot be emptied in this way, then the original identity will be proved all over again, this time using the strategy $L(\theta)-R(\theta) \not\equiv 0$. If this again fails no new strategy will be attempted, and no proof is produced.

We have hand-simulated HPC on some sixty problem identities and have succeeded in producing a proof for each. The following illustrates our hand-simulation on three of the identities. In order to be brief we have omitted much of the details of the simulation, especially in the second and third problems. We use the following abbreviations: X=unsuccessful attempt, ✓=successful, Ø empty set, SR= substep resolution, CR=C-set reduction, AM=algebraic manipulation, LS=left side.

Hand Simulation of HPC

(1) Prove: $\cos^4\theta-\sin^4\theta+1 \equiv 2\cos^2\theta$

Select strategy LS $\xrightarrow{*}$ RS
C-set of step LS → RS is {I6}
Step not primitive
C-set not empty
Step not sparse
Try SR - X
Try CR -

- Probe latent structure - X
- Substitute in LS to give $(1-\sin^2\theta)^2-\sin^2\theta+1$ (new tentative residual is $(1-\sin^2\theta)^2-\sin^2\theta+1 \rightarrow 2\cos^2\theta$)
  - C-set not reduced (to Ø)
  - Try SR - X
  - Try latent structures - ✓
    - Detected structure is $2*(1-\sin^2\theta)\rightarrow 2*\cos^2\theta$
    - Convert tentative residual to this form.
- Deactivate current problem identity
- Problem List empty.

Understandably many of the non-essential steps in the simulation have been omitted in this description. The derived proof (suitable for display to the student) is constructed from this internal solution. Overlapping of "internal steps" are avoided in constructing this "external" proof.

Derived Proof

$\cos^4\theta-\sin^4\theta+1$

$= (1-\sin^2\theta)^2-\sin^4\theta+1$ (substitution)

$= 1-2\sin^2\theta+\sin^4\theta-\sin^4\theta+1$ (goal-directed transformation guided by detected latent structure)

$= 2-2\sin^2\theta$

$= 2(1-\sin^2\theta)$

$= 2\cos^2\theta$ QED

(2) Prove: $\frac{1-\sin\theta}{1+\sin\theta} \equiv (\sec\theta-\tan\theta)^2$

C-set is {I2,I4,I6,I7} (not empty)

Step not primitive

Step sparse - convert identity to new identity $\frac{1-\sin\theta}{1+\sin\theta} \equiv \left(\frac{1}{\cos\theta} - \frac{\sin\theta}{\cos\theta}\right)^2$

New C-set is {I6}

Try SR - X

Try CR - . Probe latent structure - ✓

. Detected structure is $f*1 \to f*\left(\frac{1-\sin^2\theta}{\cos^2\theta}\right)$ with $f = \frac{(1-\sin\theta)}{(1+\sin\theta)}$

. Guided by detected structure, transform new identity to the above form.

Derived Proof

$(\sec\theta-\tan\theta)^2$

$= \left(\frac{1}{\cos\theta} - \frac{\sin\theta}{\cos\theta}\right)^2$

$= \left(\frac{1-\sin\theta}{\cos\theta}\right)^2$

$= \frac{(1-\sin\theta)^2}{\cos^2\theta}$

$= \frac{1-\sin\theta}{1+\sin\theta} \cdot \frac{(1-\sin\theta)(1+\sin\theta)}{\cos^2\theta}$

$$= \frac{1-\sin\theta}{1+\sin\theta} \cdot \frac{1-\sin^2\theta}{\cos^2\theta}$$

$$= \frac{1-\sin\theta}{1+\sin\theta} \qquad \text{QED}$$

(3) Prove: $\cos^6\theta+\sin^6\theta \equiv 1-3\sin^2\theta\cos^2\theta$

C-set is {I6} (Singleton, non-empty)

Identity not sparse

Try SR - X

Try CR - X (no latent structure)

Try AM - . applying AM-5 (factorisation) to LS we derive $(\cos^2\theta+\sin^2\theta)(\cos^4\theta-\sin^2 0\cos^2\theta+\sin^4\theta)$

- Try SR - ✓ obtaining the substeps $\cos^2\theta+\sin^2\theta \rightarrow 1$ (this is primitive) and $\cos^4\theta-\sin^2\theta.\cos^2\theta+\sin^4\theta \rightarrow 1-3\sin^2\theta\cos^2\theta$
- Append new subidentities to problem list and deactivate current identity (1st one need no further action)

New Subidentity

Prove: $\cos^4\theta-\sin^2\theta\cos^2\theta+\sin^4\theta \equiv 1-3\sin^2\theta\cos^2\theta$

C-set is {I6}

Try SR - X

Try CR - . Probe latent structure - ✓

- Detect $f + (\sin^2\theta+\cos^2\theta)^2 \rightarrow f+1^2$
- Transform subidentity to this structure

Derived proof:

$$\cos^6\theta+\sin^6\theta$$
$$= (\cos^2\theta+\sin^2\theta)(\cos^4\theta+\sin^4\theta-\cos^2\theta\sin^2\theta)$$
$$= 1*(\cos^4\theta+\sin^4\theta-\cos^2\theta\sin^2\theta)$$
$$= (\cos^2\theta+\sin^2\theta)^2-3\sin^2\theta\cos^2\theta$$
$$= 1-3\sin^2\theta\cos^2\theta \qquad \text{QED}$$

In our hand-simulation we have found some sections of the program and several minor heuristics to be of little use. Perhaps some of these may be removed without detriment to the effectiveness of HPC. But until we have carried out adequate field study we are not certain how we can best reorganise HPC.

In this chapter we have proposed a method for generating proofs. This method we have demonstrated to be effective. Such a proof-generating capacity can be used to aid in the dispensing of assistance for proof as explained earlier in the chapter.

Chapter VII

PRACTICAL CONSIDERATIONS

Although there was time for implementing only the proof-checking program Super-2, we have considered many of the practical problems of implementing TPS. In particular we have looked into the question of C-set determination, subset resolution and data structures. Some of the solutions we propose have already been adopted in Super-2. In the final section of this chapter, Super-2 will be briefly described.

C-Set Determination

The purpose of C-set determination is to find a minimal C-set on which a step holds. This involves testing the step, using the representative point technique, over various S-sets to find a maximal one over which it holds.

In Super-2 the representative points are based on a small set of values. These values, thirty two of them, are shown in Table 7.1, listed to 9 decimal places only. The first six are arbitrary and are designated as sin°,cos°,...cot° (SIN.,COS.,...COT. in the table). The remaining values are derived as shown; X, Y and Z represent three arbitrary arguments. As an illustration, entry 11 has the value cos°/sin° while entry 30 has the value cosX/cotY.

Each representative point is made up of six values from the table. In the C-set search table (Table C.2, see later) each RP is represented by its substitution set, which is simply an ordered sextuple of pointers to the substitution table.

```
NO        FUNCTION                          VALUE

 1      SIN=SIN.                        3.017429760
 2      COS=COS.                        2.711166337
 3      TAN=TAN.                        4.927116030
 4      CSC=CSC.                        7.001775210
 5      SEC=SEC.                       15.331701000
 6      COT=COT.                       11.030379100
 7      CSC=1/SIN.                       .331407880
 8      SEC=1/COS.                       .368844946
 9      COT=1/TAN.                       .202958484
10      TAN=SIN./COS.                   1.112963716
11      COT=COS./SIN.                    .898501888
12      SIN=SIN(X)                       .937020590
13      COS=COS(X)                       .349274123
14      TAN=TAN(X)                      2.682765566
15      CSC=CSC(X)                      1.067212407
16      SEC=SEC(X)                      2.863080698
17      COT=COT(X)                       .372749678
18      SIN=SIN(Y)                       .721523394
19      COS=COS(Y)                       .692390058
20      TAN=TAN(Y)                      1.042076479
21      CSC=CSC(Y)                      1.385956448
22      SEC=SEC(Y)                      1.444272616
23      COT=COT(Y)                       .959622465
24      TAN=TAN(Z)                       .437011085
25      SEC=SEC(Z)                      1.091319700
26      COT=COS(X)/SIN.                  .115752197
27      SIN=COS./COT(X)                 7.273423693
28      SIN=COS.*TAN(X)                 7.273423693
29      CSC=1/(COS.*TAN(X))              .137486835
30      SIN=COS(X)/COT(Y)                .363970348
31      TAN=SIN(X)/COS.                  .345615309
32      COS=SIN(X)/TAN(Y)                .899186009

          (WHERE X=1.214, Y=0.806, Z=0.412)

               SUBSTITUTION  TABLE
```

Table 7.1

Examples

C-set	Substitution Set	Representative Point
C0=$\emptyset$	(1,2,3,4,5,6)	(sin°,cos°,tan°,csc°,sec°,cot°)
C22={I1,I2,I3,I4,I5}	(1,2,10,7,8,11)	(sin°,cos°,sin°/cos°,1/sin°, 1/cos°,cos°/sin°)
C42={I1,I6,I7}	(12,13,24,15,25,6)	(sinX,cosX,tanZ,cscX,secZ,cot°)
C94={I1,I3,I4,I5,I7}	(28,2,14,29,16,17)	(cos°tanX,cos°,tanX,$\frac{1}{(\text{cos°tanX})}$, secX,cotX)

The RPs above can be easily shown to satisfy their respective requirements. As an example, the RP for C42 satisfies I1,I6 and I7 (cscX=1/sinX,$\sin^2$X+$\cos^2$X=1 and $\sec^2$Z=$\tan^2$Z+1) but not I2,I3,I4,I5 and I8 (secZ≠1/cosX,cot°≠1/tanZ,tanZ≠sinX/cosX,cot°≠cosX/sinX, $\csc^2$X≠cot°^2+1). From the above examples the reader should have an insight into the way in which we have assigned values to the various representative points in the search table.

Zero Approximation

In testing for a step f → g over a sampling set S, RPT uses the criterion $f°(\underset{\sim}{X}(S))=g°(\underset{\sim}{X}(S))$ or equivalently $f°(\underset{\sim}{X}(S))-g°(\underset{\sim}{X}(S))=0$, $\underset{\sim}{X}(S)$ being the RP of S. However when the test is performed on a computer, we should allow for errors due to limited precision arithmetic by replacing the criterion by $|f°(\underset{\sim}{X}(S))-g°(\underset{\sim}{X}(S))|<\varepsilon$, where ε is a tolerance factor. ε should neither be too large nor too small.

Our experiment has shown that for a fairly broad range of values for ε no incorrect conclusions are incurred. This involves a test on some 10,000 steps for various values of ε and using the RPs we have just described. The computation was performed in single-precision arithmetic on a CDC 6400. In this experiment, no error occurred for $10^{-11}<\varepsilon<10^{-4}$. In Super-2 we adopt $\varepsilon=10^{-8}$.

Finding the C-set of a Step

The set $\mathcal{C}$ can be arranged into a network whose nodes are C-sets and whose directed edges join C-sets to their immediate C-subsets. A portion of this network is shown in Figure 7.1. C97 the largest C-set is the "earliest" node while the empty C-set C0 is terminal, having no C-subsets.

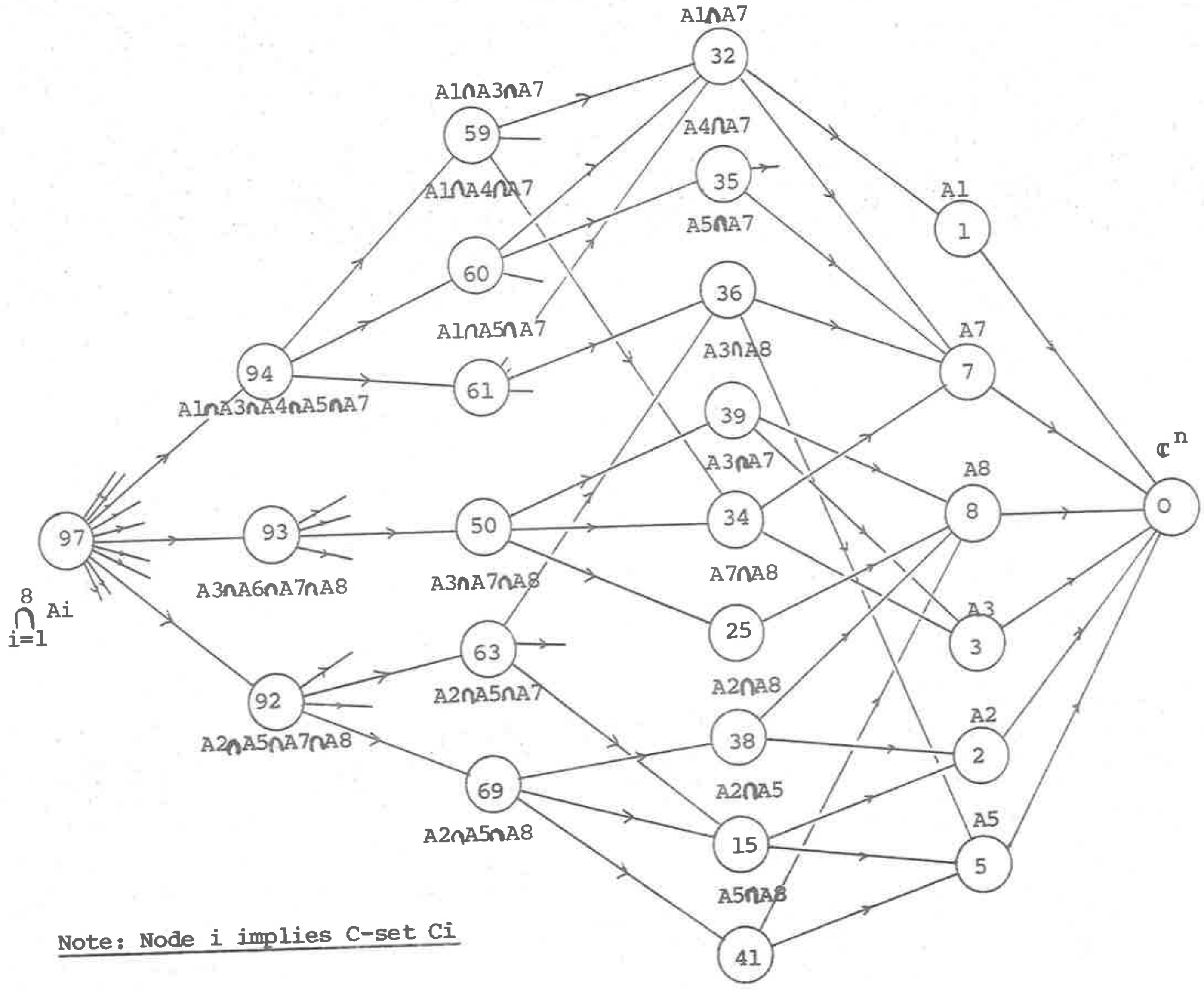
Using this network we have the following algorithmn for finding "the" C-set of a step:

[1] Test step at initial node C97. If test succeeds set i←97 and go to [2]; otherwise step is incorrect. Exit.

[2] Test step at each immediate successor node of Ci in turn until either [a] it succeeds at Cj or [b] each test fails and there is no other successor node to test at. If [a], set i←j and go to [2]; else it is [b] and the desired C-set is Ci. Exit.

It may be noted that this procedure produces only one C-set of a step.

This search algorithm is not the best possible since it requires more tests than are normally necessary. It initiates the search at C97 and this has more than twenty successor nodes. This means that a C-set search may require more than thirty tests since the earlier nodes have many successors. However most of the steps encountered in practice have a rather small (few elements) C-set and therefore a very significant saving is made in the required number of tests if we



Note: Node i implies C-set Ci

Figure 7.1 C-set Search Network

can somehow initiate the search at a later node than C97. Such an improved procedure can indeed be devised requiring on the average less than six tests to effect a search. It is based on the configuration (see below) of a step, i.e. the types of functions occurring in it.

Configuration-Directed Search

The binary sequence $b_1b_2...b_6$ defines the configuration of a step; b_i is 1 or 0 depending on whether or not the i^{th} trigonometric function occurs in the step. Thus $\sin\theta,\cos\theta,...\cot\theta$ are flagged by $b_1,b_2,...b_6$ respectively. As an example the configuration of $(1-\tan\theta)(1-\cot\theta)\rightarrow 2-\sec\theta\csc\theta$ is 001111.

The largest C-set a (correct) step may have depends on its configuration, but it usually much smaller than C97. A step in which only $\sin\theta$ and $\cos\theta$ occur cannot have a C-set which is a superset of {I6}. This is because if its C-set is larger than (is a superset of) I6 then the correctness of the step must depend on some governing relation (i.e. basic identity) involving some function other than $\sin\theta$ and $\cos\theta$. But this is impossible since this step contains only these two functions and so cannot be dependent on any other function. Thus a step with the configuration 110000 must either be an A-step or have the C-set {I6}, provided it is correct.

Given a set of basic identities we may, by substitution among them, eliminate certain functions to derive new relations. As an example, we can derive $1/\csc^2\theta+1/\sec^2\theta\equiv 1$, involving only $\csc\theta$ and $\sec\theta$, from {I1,I2,I6}. A step with the configuration 000110 could therefore be dependent on $1/\csc^2\theta+1/\sec^2\theta\equiv 1$ for its correctness. Expressed in terms of our basic identities, this means its correctness could depend on {I1,I2,I6}.

We shall refer to our basic identities and all those identities derivable from among them by substitution as operating identities. These include J1,...J4. The basic identities from which an operating identity is derived will be called its spanning identities. Thus the spanning identities of $1/\csc^2\theta+1/\sec^2\theta\equiv 1$ are I1,I2 and I6.

For a given configuration, the largest C-set possible for a step may be obtained in the following manner. Collect all the operating identities ("implied" ones may be omitted) whose functions are indicated in the configuration. Then the spanning identities of this collection is the required largest C-set. We shall call this largest C-set the entry C-set of the configuration.

e.g. configuration : 010110

Operating identities: $\sec\theta\equiv 1/\cos\theta, \cos^2\theta+1/\csc^2\theta\equiv 1$

Spanning identities : $\csc\theta\equiv 1/\sin\theta, \sec\theta\equiv 1/\cos\theta, \sin^2\theta+\cos^2\theta\equiv 1$

Entry C-set : {I1,I2,I6}.

In a configuration-directed search, we avoid a great number of tests by initiating the C-set search at the entry C-set. The method we have actually experimented with is a configuration-directed one, but it is based on a C-set search table instead of the network.

The C-set Search Table

The C-set search table, given in Table C.2 is a modification of the network organised as a table. It has 153 entries, beginning with entry-0 and ending in entry-152, and is organised into three sections :

(1) (entries 0→46) associated with non-Pythagorean C-sets only;

(2) (entries 47→53) associated with Pythagorean C-sets only;

(3) (entries 54→152) associated with hybrid C-sets only; i.e. C-sets with both Pythagorean as well as non-Pythagorean elements.

Each table entry has the following fields (from left to right):

(1) spanning C-set

(2) C-set sequence number

(3) substitution set of C-set

(4) configuration

(5) successor nodes (entry numbers)

(6) Operating identities.

<u>Example</u>: Entry-61

Spanning C-set	: {I1,I2,I6}
C-set seq. no.	: 53
Substitution set	: (12,13,3,15,16,6)
Configuration	: 110110
Successor nodes	: 71,72
Operating Identities	: I1,I2,I6 (same as spanning identities)

The successor nodes of an entry represent immediate C-subsets of its C-set, but these sets must be all Pythagorean, or all non-Pythagorean or all hybrid. Thus although {I1,I2,I6} has three immediate C-subsets, {I1,I2} is not one of its successors in entry-61, since it is non-hybrid. Entry-74 has no successors for, although {I1,I3,I8} has immediate C-subsets of the same kind (hybrid), they exceed* the configuration.

* *The hybrid C-subsets of {I1,I3,I8} are {I1,I8} and {I3,I8}, but they imply functions not indicated by the configuration 101100; e.g. {I1,I8} would imply the presence of cotθ.*

The spanning C-sets of those entries in the table, for which their configurations are indicated, are the entry C-sets of the configurations. It will be seen on inspection that in each section of the C-set search table the entries are ordered by configuration from the densest* to the sparsest (not including entries without configuration shown). Within configurations of a given density, the ordering is numeric, from larger down to smaller.

The following procedure for finding the C-set of a step, using the C-set search table has been used. In testing at a given entry, the computation of expression values is made at the representative point indicated by the substitution set.

[1] (Step Correct?) Test at entry-152. If successful, go to [2], otherwise step is incorrect and there is no C-set. Exit.

[2] (A-step?) Test at entry-0. If test fails go to [3], otherwise step is algebraic and C-set is Ø, the empty set. Exit.

[3] (non-Pythagorean?) Test at entry-1. If test fails, go to [4], otherwise step is non-Pythagorean. Enter this section of table at entry with a matching configuration with the step. Let this be entry-i. Go to [6].

[4] (Pythagorean?) Test at entry-47. If test fails go to [5], otherwise it is Pythagorean. Enter this section of table at entry-i, where the configurations of the step and this entry agree most closely. (If these configurations are respectively F1 and F2, then they must satisfy F2=F1•A•F2 where•A• is the

* *One configuration is said to be denser than another if it has more '1' bits.*

FORTRAN •AND• masking function). Go to [6].

[5] (Step is hybrid). Attempt entering this section of table at entry-i, whose configuration matches that of the step. If succeed in locating such an entry, go to [6]. Else test at entries-97,98,...116 and entries-149,150,151 (configuration shown as XXXXXX) in turn until either [a] it succeeds at some entry, say entry-i or [b] all the tests fail. If [a] is the case go to [6]; if [b] then required C-set is the largest, C97. Exit.

[6] (Track down C-set). Two cases are possible: [a] entry-i has no successor (immediate C-subset) or [b] entry-i has one or more successors. In the case [a], the C-set of entry-i is the required one. In case [b], test for each of the successors in turn. Then either [i] it succeeds at say entry-j or [ii] all successors fail. In case [i], set i←j and go to beginning of [6]. In case [ii] the step fails to hold on any of the C-subsets of the C-set of entry-i and so the latter is the required C-set. Exit.

Remarks on the Search Table

(1) Every C-set is represented in the search table.

(2) The following entries have the special functions indicated.

Entry	Corresponding C-set	For Testing Whether or not Step is:
0	[ ]	algebraic
152	[1,2,3,4,5,6,7,8]	correct
1	[1,2,3,4,5]	non-Pythagorean
47	[6,7,8]	Pythagorean

(3) The successors of entry-152 are those with configuration shown as XXXXXX. Because there are so many of them, they are not indicated in the field for successor nodes for entry-152.

(4) The entries 149,150 and 151 should have been placed with those other entries with configuration shown as XXXXXX. However their omission was discovered only after the table has been constructed. This explains the break in the sequence of successors of C97.

(5) Some configurations, e.g. 111101 and 001111, are absent in the hybrid section. However they are already covered by some other entries.

(6) Some configurations, e.g. entries-88 and 93, have multiple entry C-sets. In fact implicitly, 111111 in the hybrid section has several entry C-sets too.

(7) A full C-set search table is not required for proof-checking. Since in this case, the system is interested only in whether or not a step is correct, algebraic or has a singleton C-set, only entries for C0,C1,C2,...C8, and C97 (and {J1},...{J4}) are necessary.

Substep Resolution

There are four types of resolution to implement. Exponential resolution presents little problem. Factored resolution can be converted, either explicitly or implicitly, into the additive case by expanding $f(*)(\overset{k}{\underset{i=1}{(+)}}e_i)$ into the form $\overset{k}{\underset{i=1}{(+)}}f(*)e_i$. We are left with the multiplicative and additive cases to consider, but since these are very similar, we shall treat only the additive case.

The additive resolution problem may be stated as follows: given two equivalent expressions e and e', resolve e and e' into k terms each, giving $e_1, e_2, \ldots e_k$ and $e'_1, e'_2, \ldots e'_k$ respectively, such that $e_i \equiv e'_i$ $(i=1,2,\ldots k)$. The difficulty is that k is not known before hand. The components derived from the expressions of a step have to be regrouped into matching terms.

Expression Decomposition

Given an expression remove all its outermost parentheses, if any, and separate the resultant expression into its terms. This process is called a primary decomposition. The decomposition of ab-(c-d) into ab and -(c-d) is primary, but not the decomposition of say, ab-(c-d) into ab,-c and d or of a+(b+c) into a,b and c. If an expression undergoes two or more successive primary decompositions, then the resultant decomposition will be called secondary provided it is not the same as the primary decomposition. Thus a+b-c has no secondary decomposition.

Components will be described as primary or secondary depending on whether they are derived by a primary or a secondary decomposition.

Component Matching

Let e → e' be a correct step and let the sets $\varepsilon = \{e_1, e_2, \ldots e_n\}$ and $\varepsilon' = \{e'_1, e'_2, \ldots e'_{n'}\}$ be the derived components of e and e' respectively. Thus $(+)\varepsilon = (+)\varepsilon'$ ($(+)\varepsilon$ means $\overset{n}{\underset{i=1}{(+)}} e_i$). To resolve the step into k substeps we impose a k-partition on ε and ε'. Let this partition yield $\{E_1, E_2, \ldots E_k\}$ and $\{E'_1, E'_2, \ldots E'_k\}$ where $\varepsilon = \bigcup_{i=1}^{k} E_i$, $\varepsilon' = \bigcup_{i=1}^{k} E'_i$, $E_i \cap E_j = \emptyset$ and $E'_i \cap E'_j = \emptyset$ $(i \neq j)$. Furthermore we require the partition to satisfy: $(+)E_i = (+)E'_i$ $(i=1,2,\ldots k)$. Thus we are converting the step into the form $\overset{k}{\underset{i=1}{(+)}}\ ((+)E_i) \rightarrow \overset{k}{\underset{i=1}{(+)}}\ ((+)E_i')$.

There are S(n,k) ways of distributing n distinct objects into k non-distinct cells such that none is empty and such that the order of the objects in a cell is irrelevant. Here S(n,k) is Stirling's number of the second kind and is given by the formula $\frac{1}{n!}\sum_{i=1}^{n}(-1)^i\binom{n}{i}(n-i)^r$ (see Liu [39], p. 38). It follows that the number of ways of partitioning a set of n distinct elements into k classes is also S(n,k).

Assuming distinct components, we have S(n,k) and S(n',k) ways of k-partitioning ε and ε'. For each such partitioning of ε and of ε' there are k! possible 1-1 mappings between the two sets of classes. Considering all possible k-partitionings there are a total of $T1 = S(n,k) \times S(n',k) \times k!$ such mappings. From among these mappings we seek one, g say, satisfying: $g(E_i) = E_i'$ and $(+)E_i \equiv (+)E_i'$ $(i=1,2,\ldots k)\ldots(*)$. T1 represents the number of 1-1 mappings we may have to try before finding one satisfying the requirements (*), or discovering that no such match exists. Considering all possible values of k, we may have to try $T2 = \sum_{k=0}^{n_o} S(n,k) \times S(n',k) \times k!$ such mappings, where n_o = minimum (n,n') before we are assured that no resolution of the step e → e' is possible. Table 7.2 shows some Stirling numbers of the second kind, to indicate how steeply T2 grows with n and n'.

The above formulation assumes a partitioning in which no class is empty. But this fails to account for the case of implicit substeps as is found in say $\sin^2\theta + (\tan\theta - \sin\theta/\cos\theta) \rightarrow (1-\cos^2\theta)$. Here

n \ k	1	2	3	4	5	6	7	8	9	10	k
1	1										
2	1	1									
3	1	3	1								
4	1	7	6	1							
5	1	15	25	10	1						
6	1	31	90	65	15	1					
7	1	63	301	350	140	21	1				
8	1	127	966	1701	1050	266	28	1			
9	1	255	3025	7770	6951	2646	462	36	1		
10	1	511	9330	34105	42525	22827	5880	750	45	1	
n											

Table 7.2 Some Values of S(n,k)

$(\tan\theta-\sin\theta/\cos\theta)\to 0$ is a substep although the 0 is present only implicitly in the right expression. (In multiplicative resolution we have an implicit match with ± 1). The number of k-partitions of n distinct objects, permitting empty classes, is $\sum_{i=1}^{k} S(n,i)$, this being also the number of ways of distributing n distinct objects into k non-distinct cells, with empty cells allowed (see Liu [39]). The effect of this revision is to greatly increase the size of T2 derived above.

The aim of our brief combinatorial exercise has not been so much to derive specific formulae as to show that to carry out a substep resolution, a very large number of trial-and-error operations might be required, this number growing dramatically (exponentially) with the number of components obtained from the step expressions. There is therefore a need for a very efficient method for deciding expression equivalence, and this is one application in which our numeric approach has a big edge over a purely symbolic one. The choice of data structures used in the implementation also becomes very important; we shall discuss this topic later.

Primary and Secondary Resolution

Substep resolution based on primary components of the step expressions will be called a primary resolution; similarly a secondary resolution is based on secondary components only. Our ad hoc approach is to perform a primary resolution first. If this fails, then a secondary resolution is attempted. If a secondary resolution fails, then another (based on a further secondary decomposition of the expressions) is attempted, unless this is not possible.

The rationalé for this approach is that a primary resolution involves fewer components than a secondary one, and therefore is much faster to carry out, since the effort required to perform a resolution increases exponentially with the number of components. Besides, by the way subexpressions occur, a primary resolution has a relatively high chance of succeeding. It would be inefficient therefore to attempt a secondary one straight away.

Non-uniqueness

The resolution of a step is not always unique, but where all possible resolutions lead to the same conclusion regarding step-size, then we have no problem. However a small step may give rise to a resolution which causes it to be regarded as large. One such case is the following: $a+b \to c+d+d'$ being resolved into $a \to c+d'$ and $b \to d$ with both substeps small, and into $a \to c+d$ and $b \to d'$ in which $b \to d'$ is large.

In practice the proportion of cases incorrectly analysed in consequence of non-uniqueness is so small that we can ignore it without serious consequence. There are ways in which this problem can be handled, but we shall not consider any of them here. We merely wish to point out the existence of the problem.

Sequential Resolution Procedure

So far our formulation has more or less assumed simultaneous resolution. But there is no need for deriving all the substeps of a resolution at once. Instead we can obtain the same results sequentially, and this approach is simpler to implement. In sequential resolution, we remove a substep from the step as it is detected, continuing in like manner with the remaining step. This is in fact the approach used in Super-2.

Sequential resolution may be described briefly as follows. Let the components of the step expressions be, as usual, $\varepsilon=\{e_1,e_2,\ldots e_n\}$ and $\varepsilon'=\{e'_1,e'_2,\ldots e'_n\}$. We attempt to detect substeps by a sequence of schedules 1-1,1-2,2-1,2-2,1-3,.... . A schedule r-s is a program for seeking out a substep involving r components of ε and s components of ε'. The $\binom{n}{r}$ classes of r components of ε are matched against the $\binom{n'}{s}$ classes of s components of ε' in turn until there is a successful match or until all attempts fail. A successful match indicates a substep and this is removed. The matching comparison is done numerically, using maximum consistency (q.v.) values. When a substep is removed, n-r components of ε and n'-s components of ε' remain. The search for another substep under the same schedule continues with these remaining components provided there are enough of them, i.e.

$n-r \geq r$ and $n'-s>s$. If this schedule no longer applies, the next one is tried until no further schedules are applicable, by reason of insufficient number of components. When a given class of r components is initiated, a test for an implicit test is also made by testing it against 0.

In Super-2 only the 9 schedules 1-1,1-2,2-1,2-2,1-3,3-1,2-3,3-2 and 3-3 are used, in that order. In analysing the 207 sample steps, it was found that the removal of the last five schedules did not affect the result of the analysis. This is because the number of components involved in the expressions are small, (Super-2 does not perform secondary resolution) and the substeps themselves involve relatively few components. If the first four schedules prove adequate, then a significant improvement to the resolution algorithmn is achieved, since the last five schedules often involve ~~much~~ many more tests than the first four.

Data Structures

How efficiently and elegantly we can implement a program will depend to a large extent on our choice of the underlying data structures. In TPS this consideration is vital if we are to expect the full advantages of our largely numeric approach. There are several important factors to bear in mind.

Our approach involves a large number of computations for deriving expression values at various selected points. A standard approach for evaluating an expression is to convert it first into a Polish form and then to carry out the computation interpretively. There are several forms of the Polish notation (see Hamblin [30]) that can be used. In Super-2 we choose the early prefix Polish.

Expressions under investigation should be carried in the Polish form to avoid repeating the costly process of converting an expression into this form every time it is to be evaluated. Polish expressions should be maintained internally as a simple doubly-linked list to expedite the insertion and deletion of subexpressions. The latter operations occur during sequential substep resolution.

A doubly-linked list consists of a sequence of cells, each linked to its left and right neighbours; the first and the last cells have only one link each. The diagram below shows sinθ+cosθ*(1-tan^{2}θ)-tanθ maintained in its Polish form as a doubly-linked list ('↔' represents a double link).

[-]↔[+]↔[sinθ]↔[*]↔[cosθ]↔[-]↔[1]↔[↑]↔[tanθ]↔[2]↔[tanθ]

TPS requires the values of expressions and subexpressions at various representative points. Of these the most important are the maximum consistency (MXC) values computed at the RP of C97 and the minimum consistency (MNC) values computed at the RP of C0. Since the MXC and MNC values are used very frequently they are carried permanently with the Polish list in Super-2. The diagram below shows how the fictitious MNC values of the last expression (assuming sinθ=1, cosθ=2 and tanθ=3) may be stored with the list. Note that each cell carries

-18	-15	1	-16	2	-8	1	9	3	2	3
[-]↔	[+]↔	[sinθ]↔	[*]↔	[cosθ]↔	[-]↔	[1]↔	[↑]↔	[tanθ]↔	[2]↔	[tanθ]

the value of the subexpression it initiates. In particular the values of all the components are now known and this is a great advantage since these values are required again and again during substep resolution. The dramatic improvement derived by this saving of the MXC and MNC values will be shown in our discussion on Super-2.

For the same reason as for the MXC and MNC values, we should also derive the component counts of the various elements and subexpressions only once and save them in the list, ready for use when required.

Two basic formula manipulation operations in resolution are the locating of components and the detaching of matched components. In both we want to find the initial and terminal cells of a desired component so that we can read off its MXC or MNC value, or compute its value as some RP or to remove it from the parent list. To ensure efficiency, we should perform our formula manipulations in the Polish form only so that we do not need to operate at the usual infix form as well. Unfortunately, a Polish expression, maintained as a simple doubly-linked list is highly unstructured. It does not reveal easily the location of desired components.

Furthermore, during component detachment, and attachment, certain of the values and component counts in the resultant list become incorrect and must be recomputed. To avoid unnecessary corrections, we should recompute only those affected quantities. There is then the problem of locating the affected cells.

In the following section, we develop a simple numbering scheme which imposes a useful structure on the list by assigning an index to each cell. The indices enable us to develop very simple algorithms for the various formula manipulation operations. The indices are carried with the list, in the same way, and for much the same reasons, as for the other appended quantities.

Index Theory

We shall define an index scheme for prefix Polish and study some of its properties. We shall then show the relevance of these properties in our formula manipulation operations.

Basic Definitions

D.1 A symbol is a member of a set $\{s_1,s_2,s_3,...\}$ of primitive operands. e.g. A,ZR,12,5

D.2 An operator is a member of a set $\{\emptyset,\emptyset_1,\emptyset_2,...\}$ of operators. e.g. +,-,*,↑,~,sin,sec. (~ is the unary minus)

D.3 An element is a symbol or an operator.

D.4 A string is a sequence of elements.

D.5 The degree D(e) of an element e is given by: (1) D(e)=0 if e is a symbol and (2) D(e)=m if e is an m-ary operator. e.g. D(sec)=D(~)=1, D(A)=D(5)=0 and D(*)=D(+)=2.

D.6 The rank R(e) of an element e is D(e)-1.

D.7 The rank of a string S is the sum of the ranks of its elements.

D.8 Well-Formed Formula (wff)

(a) a single symbol is a wff;

(b) if $\emptyset$ is an m-ary operator and $F_1,F_2,...F_m$ are wffs then $\emptyset F_1F_2...F_m$ is a wff;

(c) a string is a wff if and only if it is so by virtue of (a) and (b).

Note: D.8 defines a well-formed expression in the prefix Polish form. Note that the rightmost element of a wff is always a symbol.

D.9 A substring of a wff which is also a wff is called its well-formed subformula (wfsf).

e.g. 'a' and '-bc' are wfsfs of the wff '+a/-bcd'

Notation: We will usually denote a wff F by $e_1e_2e_3...e_n$ where $e_1,e_2,...e_n$ are elements.

D.10 W(e_i) or W(i) denotes the wfsf of $e_1e_2...e_n$ beginning with e_i.

e.g. If $e_1e_2...e_7$ is '+a/-bcd' then W(2) is 'a', W(4) is '-bc' and W(9) is undefined.

The problem of locating W(i) is to find its rightmost element. One way of locating this element is to use the Rank Theorem (see Nelson [50], p.45) which states: The rank of a well-formed formula is -1. Since W(i) is unique all we have to do then is to add the ranks of the string elements beginning with e_i and proceeding to the right until the sum is -1. However we shall use an alternative approach based on indices (q.v.).

D.11 Definition of Index

The index of the element e_i in the string $e_1e_2...e_n$, denoted by I(e_i) or I(i) is given by:

(a) $I(i)=I(i+1)-R(e_i)$

(b) $I(n+1)=N$

N, the index base, is an arbitrary integer value to be set. The standard index has the base N=0. Unless otherwise specified, all indices will be assumed to be standard, as in Super-2.

e.g. The Polish string and its associated indices for the expression

A+(B-C)/(2*A-P/(Q+R)) is as follows. The indices are computed from the right.

I(i):	1	2	1	2	3	2	1	2	3	2	1	2	1	2	1
e_i:	+	A	/	-	B	C	-	*	2	A	/	P	+	Q	R

D.12 The rightmost and the leftmost elements e_n and e_1 of the string S: $e_1 e_2 \ldots e_n$ are denoted by Rm(S) and Lm(S) respectively. If S is an element then Rm(S) and Lm(S) are the same element.

We shall now derive a number of simple index-theoretic results which are useful for our numeric heuristics. These results will be called observations or Obsns for short. They pertain to expressions and subexpressions which are well-formed. It would be helpful to the reader to consult the last example when reading the observations.

The Indices of Well-formed Expressions

Obsn 1: Let $F_i F_{i+1}$, where F_i, F_{i+1} are wffs, be a substring of a wff. Then $I(Lm(F_{i+1})) = I(Rm(F_i))-1$.

Proof: As noted earlier the rightmost element of a wff is always a symbol and therefore has the rank -1. It follows from the definition of index, that $I(Rm(F_i)) = I(Lm(F_{i+1}))-R(Rm(F_i)) = I(Lm(F_{i+1}))+1$. Hence the observation. QED.

Obsn 2: If F is a wff then $I(Rm(F)) = I(Lm(F))$.

Proof: We prove this observation by induction on the length of F. If F is of length 1, then result is obviously true since Lm(F) and Rm(F) are the same element (true for n=1).

Suppose observation holds for all F's of length $\leq$ n. Let F be of length n+1. Since F has at least two elements it has the form

$\emptyset F_1F_2...F_m$ where $\emptyset$ is an m-ary operator and $F_1,F_2,...F_m$ are all well-formed. Since each F_i is of length $\leqslant n$, $I(Rm(F_i))=I(Lm(F_i))$ $(i=1,2,...m)$, by the induction hypothesis. Let $I(Lm(F_m))=k$. Then by Obsn 1, $I(Lm(F_{m-1}))=k+1$, $I(Lm(F_{m-2}))=k+2$, ..., $I(Lm(F_1))=k+(m-1)$. By definition, $I(Lm(F))=I(\emptyset)=I(Lm(F_1))+R(\emptyset)=k+(m-1)-(m-1)=k=I(Lm(F_m))=I(Rm(F_m))=I(Rm(F))$. QED.

Obsn 2 states that the extremal elements of a wff have equal indices. This motivates the next definition.

D.13 The index of a wff is the common index of its extremal elements.

Thus if F is a wff, then $I(F)=I(Rm(F))=I(Lm(F))$.

Obsn 3: Let $\emptyset$ be an m-ary operator and $F_1,F_2,...F_m$ wffs. Then the indices of $\emptyset,F_1,F_2,...F_m$ in the wff $\emptyset F_1F_2...F_m$ are respectively $1,m,m-1,m-2,...3,2,1$.

Proof: See proof for Obsn 2.

Note that all standard indices are positive and the index of a wff, not treated as a wfsf of another wff, is 1.

Obsn 4: Let $F \equiv e_1e_2...e_n$ be a wff. Then $I(i)\geqslant 1$ $(i=1,2,...n)$

Proof: Follows from Obsn 3 and induction.

Obsn 4 states that no standard index in a wff can be less than 1. In fact in any wfsf of a wff, the indices are no less than those of the extremal elements of the wfsf.

Obsn 5: Let $F \equiv e_1e_2...e_n$ be a wff. Then $Rm(W(i))$ is the first element e_j to the right of e_i satisfying $I(j+1)<I(i)$ or if this e_j does not exist, it is e_n.

Proof: Either (1) W(i) has a right-hand neighbour in F or (2) it does not. In the case (2), W(i) is e_n.

In the case (1), since W(i) has a right neighbour it must have a wfsf as a right neighbour. By Obsn 1, the index of the right neighbour must be k-1 if the index of W(i) is k. By Obsn 4, every element in W(i) has an index $\geqslant$ k. Therefore the right neighbour of W(i) is the first element on its right with a smaller index than its own. QED.

This Observation enables us to locate the right extremity of a component in a Polish expression very easily. There is no need to apply the Rank Theorem.

D.14 Let $ØF_1F_2...F_m$ be a wff in which operator Ø is m-ary and $F_1,F_2,...F_m$ are well-formed. Then Ø is called the <u>governing operator</u> of $F_1,F_2,...F_m$, which in turn are the <u>operands of Ø</u>. $ØF_1F_2...F_m$ will be called the governing wff of $Ø,F_1,F_2,...F_m$.

<u>D.15</u> The operator Op maps a wff on its governing operator. Thus (cf D.14) $Op(F_1)=Op(F_2)=...=Op(F_m)=Ø$. $Op^{i+1}(F)$ denotes $Op(Op^i(F))$. A wff of length 1 has no governing operator.

<u>Obsn 6</u>: If $ØF_1F_2...F_m$ is a wff, then $F_1,F_2,...F_m$ are respectively $W(i_1),W(i_2),...W(i_m)$ where e_{i_j} (j=1,2,...m) is the first element to the right of Ø whose index is m-j+1.

<u>Proof</u>: Very similar to proof for Obsn 7 below.

This observation gives us a simple index-directed method for locating the operands of an operator in a Polish string.

<u>Obsn 7</u>: The governing operator of a wfsf F_i of a wff is the first element on its left with an index not greater than its own.

<u>Proof</u>: F_i must be in the context $...ØF_1F_2...F_m...$ as one of $F_1,F_2,..,F_m$.

Let I(Ø)=k. Then $F_1, F_2, \ldots F_m$ have the indices k+(m-1),k+(m-2),... ...k+2,k+1,k(cf Obsn 3). By Obsn 4, all the elements in $F_1, F_2, \ldots F_{m-1}$ have indices greater than k. Hence the observation. QED.

Obsn 8: If we replace any wfsf in a wff with a wff, then the resultant string is a wff.

Proof: Definition of well-formed formula.

Using Indices in our Data Structures

In TPS the operators encountered are the arithmetic ones +,-,*, /,↑,~ and V (reciprocal). Since we are treating sinθ,cosθ,...cotθ as the special variables SIN,COS,...COT (i.e. $x_1, x_2, \ldots x_6$) we do not concern ourselves with trigonometric operators. Our operators are all of degree 1 or 2 and although our index theory applies to general m-ary operators, m≤2 is adequate for TPS.

We maintain an expression internally as a prefix Polish string $e_1e_2\ldots e_n$. To each element e_i we append its MXC and MNC values, its index I(i) and its component count N(i). To simplify description we represent the MXC and MNC values by a single one V(i) which can be regarded as being derived for some given substitution.

We shall refer to a string $e_1e_2\ldots e_n$ in which e_i is associated with I(i),V(i) and N(i) as a structure. If the string is a wff and I(i),V(i) and N(i) (i=1,...n) are all correct and consistent then we shall call the structure well-formed.

Let $e_1e_2\ldots e_n$ be a wff. Then V(i) the value of element e_i under some given substitution is given by the following.

If (1) e_i is a numeric constant, V(i) is that number;

(2) e_i is a variable, V(i) is the substituted value of e_i;

(3) e_i is an operator, V(i) is the value obtained by applying the operator to the values of its operands. (If e_i is +, then addition is implied, etc.)

The component count N(i) of e_i is defined as follows :

If (1) e_i is a variable or constant then N(i)=0;

(2) e_i is ↑ and its left operand is SIN,COS,...or COT, and the right operand is 0,1 or 2, then N(i)=0.

(3) e_i is ↑ and the operands are not of the type in (2), then N(i) is the component count of its left operand plus 1, i.e. N(i+1)+1.

(4) e_i is +, -, * or / then if e_{i+1} is of the same hierarchy* then N(i) is the sum of the component count of its 2 operands plus 1;

(5) e_i is +, -, * or / but does not satisfy condition (4), then N(i) is the sum of the component count of its two operands plus 2.

Figure 7.2 below shows the structure of § ≡ A-B+(C-D)+P/(C-P*Q), assuming the substitution A=2,B=4,C=6,D=5,P=5 and Q=4.

<u>Remark</u>: The indices carried in a string are relative to the base N. The following diagram shows three valid assignments for the string of A*(B/C+D).

1	2	1	2	3	2	1
*	A	+	/	B	C	D

(N=0)

6	7	6	7	8	7	6
*	A	+	/	B	C	D

(N=5)

-3	-2	-3	-2	-1	-2	-3
*	A	+	/	B	C	D

(N= -4)

** + and - have equal hierarchy as have * and /, but the latter pair are of a higher hierarchy than the former.*

i :	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I(i):	1	2	3	4	3	2	3	2	1	2	1	2	1	2	1
e_i :	+	+	-	A	B	-	C	D	/	P	-	C	*	P	Q
V(i):	$\frac{-19}{14}$	-1	-2	2	4	1	6	5	$\frac{-5}{14}$	5	-14	6	20	5	4
N(i):	12	5	2	0	0	2	0	0	6	0	4	0	2	0	0

Figure 7.2 A Structured List

The subexpression B/C in the first assignment has the indices $\overset{2}{/}\ \overset{3}{B}\ \overset{2}{C}$ and this can be regarded as being relative to a base of 1.

Applying Index Theory to Formula Manipulation in Prefix Polish

We shall now consider several important applications of our index-theoretic results to subexpression manipulation.

Expression Decomposition

In Super-2 an expression is decomposed only implicitly. The components are located and their positions in the string noted in a component table. Obsn 5 and Obsn 6 enable us to locate the operands of an operator.

If the operator is unary then its only operand begins immediately on its right, the right extremity of this operand being easily found by using Obsn 5. If the operator is binary, of index k say, then it has a left and a right operand. The left operand is located in the same way as the operand of a unary operator. The right operand begins with the element which is the first to the right of the operator to have the index k+1. Its right extremity can again be found by using Obsn 5.

e.g. In Figure 7.2 the operands of e_2 are W(3): -AB and W(6): -CD, and the left and right operands of e_1 are W(2) and W(9).

When an expression is in the additive or multiplicative form, several components are often involved. The operands of any one operator do not yield directly the desired components. To illustrate, the primary components of § are A,-B,(C-D) and P/(C-P*Q) but the operands of say e_1 (cf Polish string of §) are W(2) and W(9) but not the components. We shall now present an algorithm for deriving the primary components using indices. We shall treat only the additive case as the multiplicative one is similar.

To locate the primary additive components, scan the string $e_1e_2...e_n$ from the left for the leading chain $e_1e_2...e_r$ of additive operators. If there is no such chain, then the expression is its only component. Otherwise, the primary components are $W(i_0)$, $W(i_1)$,...$W(i_r)$ where i_0 is r+1 and e_{i_j} (j=1,2,...r) is the first element to the right of e_r for which $I(i_j)=I(j)$. This follows from a repeated application of Obsn 6.

An expression such as -(A+B-P/Q) is not in the additive form strictly speaking. However by taking care of the signs separately, we can derive the components by bypassing the unary minus. At the Polish level this means applying our algorithm to $e_2e_3...e_n$.

Applied to § the primary components are, ignoring signs, W(4),W(9),W(6) and W(5), i.e. A, /P-C*PQ, -CD and B. Note that V(4),V(9),V(6) and V(5) are the values of these components. When taking into account the signs, a component should be negated only if it is the right operand of a '-'. In our example only B should be negated.

Our method does not strictly yield the primary components in all cases. It gives the same components as for § when applied to (A-B)+(C-D)+P/(C-P*Q), which has only three primary components. The anomaly is due to the first primary component being itself an additive form. We shall however ignore this unimportant anomaly.

If we apply our algorithm a second time to §, we derive the secondary components A,-B,C,-D and P/(C-P*Q). Note that only (C-D) of the primary components is decomposed in this secondary resolution.

Subexpression Detachment

When a substep is detected during sequential resolution, the components forming the substep are detached from the parent string. A substep expression may involve several components. We shall now examine the process of detaching a component from a wff.

Let the parent string be of the form $F \equiv \ldots \overset{k}{\emptyset}\, \overset{k+1}{F_1}\overset{k}{F_2} \ldots$ where Ø is a binary operator, and F_1 and F_2 are wfsfs. To detach F_2 from F we must replace $\emptyset F_1 F_2$ (a wfsf) by F_1 (another wfsf) in the string of F. This involves the delinking of F_2 and Ø. If F_1 is to be detached, then we must replace $\emptyset F_1 F_2$ by F_2 if Ø is + or *, by $\sim F_2$ if Ø is - and by ∇F_2 if Ø is /. These actions are summarised in Table 7.3.

When we have taken the above actions to detach a component, the resultant string reflects correctly the resultant expression. However the resultant structure is no longer well-formed, because some of the appended quantities (the V(i),N(i) and N(i)) are no longer correct. To rectify these affected values, we do not have to recompute completely the appended quantities since normally not many of them are incorrect.

If it is F_1 which is detached, then no index correction is required since $\emptyset F_1F_2$ is replaced by a wfsf of the same index. If F_2 is the component detached, then only the indices of the elements of F_1 which replaces $\emptyset F_1F_2$ need be corrected. This is done by reducing each incorrect index by 1, to make the new index of F_1 the same as that of the wfsf it replaces.

The only "V(i)" and "N(i)" that are incorrect are those of the governing operators of the affected wfsf in the resultant string. Thus if F_2 has been detached, then the affected operators are $Op(F_1)$, $Op^2(F_1)$, $Op^3(F_1)$... (till there are no more). These operators can be easily located using Obsn 7. The correction for the "V(i)" and "N(i)" should be made in the order of the governing operators shown.

Figure 7.3 illustrates the detachment of (C-D) from the expression 5, in three stages. The first shows the well-formed structure prior to the detachment. The second shows the effect of delinking '-CD' and its governing '+'. The affected appended quantities are shown in italics and they are rectified in stage three.

Table 7.3 summarises the actions necessary to effect the detachment of a subexpression under various conditions. Note that the 'k' and 'k+1' on the expressions are indices.

So far we have considered only the detachment of a single component. When several components are involved it would not be a good approach to handle the detachment by detaching each component independently.

i:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
I(i):	1	2	3	4	3	2	3	2	1	2	1	2	1	2	1
e_i:	+	+	-	A	B	-	C	D	/	P	-	C	*	P	Q
V(i):	$\frac{-19}{14}$	-1	-2	2	4	1	6	5	$\frac{-5}{14}$	5	-14	6	20	5	4
N(i):	12	5	2	0	0	2	0	0	6	0	4	0	2	0	0

(I)

i:	1	3	4	5	9	10	11	12	13	14	15
I(i):	1	3	4	3	1	2	1	2	1	2	1
e_i:	+	-	A	B	/	P	-	C	*	P	Q
V(i):	$\frac{-19}{14}$	-2	2	4	$\frac{-5}{14}$	5	-14	6	20	5	4
N(i):	12	2	0	0	6	0	4	0	2	0	0

(II)

i:	1	3	4	5	9	10	11	12	13	14	15
I(i):	1	2	3	2	1	2	1	2	1	2	1
e_i:	+	-	A	B	/	P	-	C	*	P	Q
V(i):	$\frac{-33}{14}$	-2	2	4	$\frac{-5}{14}$	5	-14	6	20	5	4
N(i):	9	2	0	0	6	0	4	0	2	0	0

(III)

Figure 7.3 Component Detachment

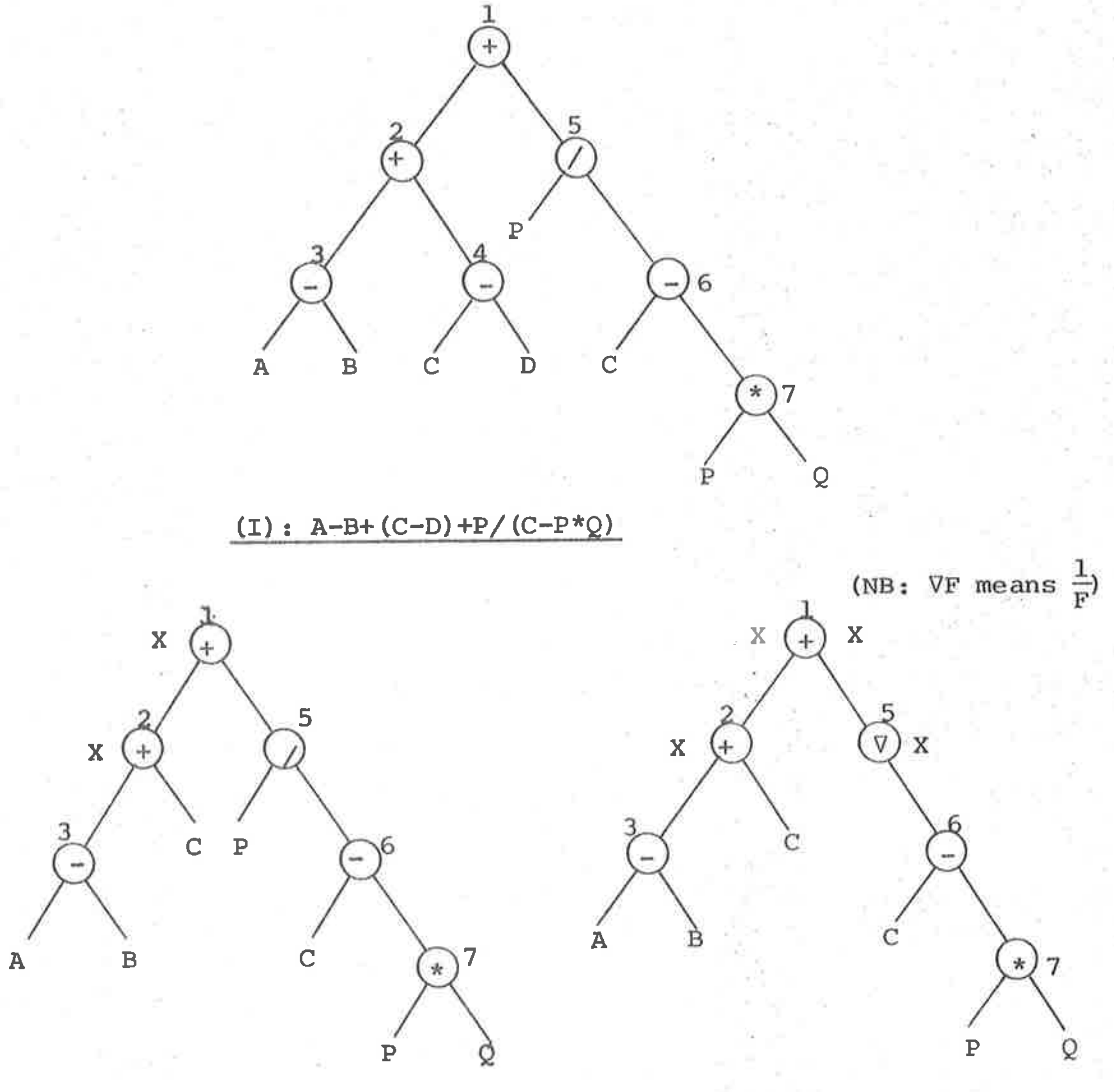
This is because the "V(i)" and "N(i)" of affected operators may be corrected more than once. Affected indices may, as before, be corrected after each component is detached. However the correction of affected "V(i)" and "N(i)" should be deferred until all detachable components have been removed.

As an illustration consider the detachment* of D and of P (the first one) from §. We detach D first say, and then P flagging affected operators in each case. Affected "V(i)" and "N(i)" are then corrected. Figure 7.4 shows what happens in three stages.

* *In TPS, we detach only components at the outermost level; so this example is more general than we need in TPS.*

Expression	To Detach	Delinking Action	Resultant Expression	Correction	
				Indices	V(i) and N(i)
$..\overset{k}{+}\overset{k+1}{F_1}\overset{k}{F_2}..$	F_1	Delink + & F_1	$..\overset{k}{F_2}..$	none	$i=Op(F_2),Op^2(F_2),..$
$..\overset{k}{+}\overset{k+1}{F_1}\overset{k}{F_2}..$	F_2	Delink + & F_2	$..\overset{k+1}{F_1}..$	Reduce all indices in F_1 by 1	$i=Op(F_1),Op^2(F_1),..$
$..\overset{k}{-}\overset{k+1}{F_1}\overset{k}{F_2}..$	F_1	Delink F_1 Change - to $\sim$	$..\overset{k}{\sim}\overset{k}{F_2}..$	none	$i=Op(\sim F_2),Op^2(\sim F_2),..$
$..\overset{k}{-}\overset{k+1}{F_1}\overset{k}{F_2}..$	F_2	Delink - & F_2	$..\overset{k+1}{F_1}..$	Reduce all indices in F_1 by 1	$i=Op(F_1),Op^2(F_1),..$
$..\overset{k}{*}\overset{k+1}{F_1}\overset{k}{F_2}..$	F_1	Delink * & F_1	$..\overset{k}{F_2}..$	none	$i=Op(F_2),Op^2(F_2),..$
$..\overset{k}{*}\overset{k+1}{F_1}\overset{k}{F_2}..$	F_2	Delink * & F_2	$..\overset{k+1}{F_1}..$	Reduce all indices in F_1 by 1	$i=Op(F_1),Op^2(F_1),..$
$../\overset{k}{}\overset{k+1}{F_1}\overset{k}{F_2}..$	F_1	Delink F_1 Change / to ∇	$..\overset{k}{\nabla}\overset{k}{F_2}..$	none	$i=Op(\nabla F_2),Op^2(\nabla F_2),..$
$../\overset{k}{}\overset{k+1}{F_1}\overset{k}{F_2}..$	F_2	Delink / & F_2	$..\overset{k+1}{F_1}..$	Reduce all indices in F_1 by 1	$i=Op(F_1),Op^2(F_1),..$

Table 7.3 Subexpression Detachment



(I): A-B+(C-D)+P/(C-P*Q)

(NB: ∇F means $\frac{1}{F}$)

(II): A-B+C+P/(C-P*Q)

(III): A-B+C+(∇(C-P*Q))

Figure 7.4 Multiple Component Detachment

The first shows § as a tree prior to the removal of D. The second shows the effect of D's removal, the affected operators being flagged with an 'X'. The situation after P is removed is given in the third stage, affected operators are again flagged. Note that the uppermost '+' (node 1) is flagged twice showing that by deferring correction, we have avoided one correction of its value and component count.

The affected values and component counts can now be recomputed by traversing down the final tree. Thus V'(1)=V'(2)+V'(5), V'(2)=V(3)+V(C) and V'(5)=1/V(6), where V'(i) is the corrected value of V(i). Note that V'(1) cannot be evaluated until V'(2) and V'(5) have been computed. The component counts can be recomputed in a similar way.

Super-2

Super-2 is a FORTRAN program written to implement the step-size model Model-0, a precursor to Model-A. It runs on a CDC 6400 in about 35K (octal) of central memory although this requirement can be reduced to about 25K by removing the many diagnostic and debugging codes and revising the program. A listing of Super-2, together with its data of sixteen proofs, are given in Appendix D.

The main aims of Super-2 have been the following.

(1) To check out an actual step-size model, in fact one which has many features in common with Model-A and Model-B.

(2) To verify the effectiveness of the various ideas we have proposed. These include C-sets, our representative point technique and substep resolution.

(3) To give us some idea of the programming efforts required to implement TPS fully and an appreciation of the practical problems that may be expected.

(4) To provide us with some data on the likely performance of TPS, at least in its proof-checking aspects. We are anxious to confirm that it will be efficient enough to meet our real-time objectives.

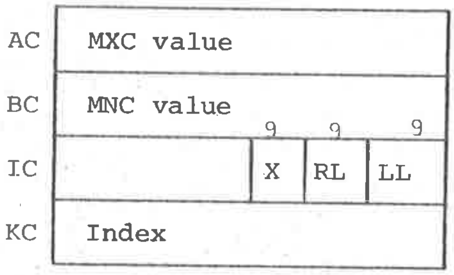
<u>The Program</u>

Super-2 reads in a proof '$e_1=e_2=\ldots-e_n$' and generates the $\binom{n}{2}$ steps $e_i \to e_j$ $(1 \leq i<j \leq n)$. The proof expressions are checked for correctness of syntax. Each step is analysed for step-size in turn.

When a step is analysed its expressions are first converted into early prefix Polish and maintained as a doubly-linked list. The conversion is done by scanning an expression backwards, using an operator stack and a table of operator hierarchies. The Polish string is thus generated in reverse. As each element of the string is produced, its index and its MXC and MNC values are computed and appended.

Figure 7.5 below shows one cell of a list made up of four words AC,BC,IC and KC. AC and BC hold the MXC and MNC values while KC holds the index.* IC contains the left and right links as well as identifying information for this element of the Polish string. Super-2 does not use any component count although this can be easily packed into IC if required. For the computation of the index, MXC and MNC values three stacks are used.

* *An index, being normally a small integer, should be packable into the spare bits of IC. However in Super-2, the indices are negative because we used an earlier definition of index in which '$I(i) = I(i+1)+R(e_i)$' (see D.11).*



RL - right link (9 bits)
LL - left link (9 bits)
X - identifier for number constant, operator and trig. function.

Figure 7.5 A List Cell

When the expressions of a step are converted into their Polish lists, their MNC values are compared. If they are equal (subject to a tolerance factor of 10^{-8}) then the step is algebraic and it is not analysed any further. If the MNC values are unequal, then the MXC values are compared; non-equality implies that the step is incorrect.

Super-2 performs only the additive, multiplicative and exponential resolutions, and these are done at the primary level only. The sequential resolution method is used. Super-2 employs the schedules 1-1,1-2,2-1,...2-3,3-2 and 3-3. In the analysis of the 207 sample steps, the last five schedules were found to be redundant.

Super-2 decomposes a step as far as possible into its substeps before testing for primitiveness. A-steps, being regarded as small, are not tested for primitiveness. To be primitive, the C-set of a step must be singleton. Furthermore it must satisfy one of the discrepancy tests explained below.

Let $F \to G$ be a permitted variant of the reference identity Ii (or J_j) and let $\underset{\sim}{U}$ be an arbitrary point from $\mathbb{R}^6$ that is not from an S-set (apart from $\mathbb{C}^n$). Define the <u>difference discrepancy DD</u> as $|F^\circ(\underset{\sim}{U})-G^\circ(\underset{\sim}{U})|$ and the <u>ratio discrepancy RR</u> as $|F^\circ(\underset{\sim}{U})/G^\circ(\underset{\sim}{U})|$(provided $G^\circ(U)\neq 0$). DD would in general be nonzero and RR not equal to 1. Let f→g be a step whose C-set is {Ii}. Then we have the following criteria of primitiveness.

(1) f→g is primitive if and only if $|f^\circ(\underset{\sim}{U})-g^\circ(\underset{\sim}{U})|=DD$ (difference criterion).

(2) f→g is primitive if and only if $|f^\circ(\underset{\sim}{U})/g^\circ(\underset{\sim}{U})|=RR$ or $|g^\circ(\underset{\sim}{U})/f^\circ(\underset{\sim}{U})|=RR$ (ratio criterion).

(3) f→g is primitive if it is so by (1) or (2) (mixed criterion).

These criteria roughly cover the specifications in Model-A for primitive T-steps together with the step composition rule W_5, without rigorously implementing the model.

Since standard FORTRAN has no list-processing facility, Super-2 has provided its own. It maintains an available cells list (ACL) which is initialised to 511 cells of four words each. Idle cells are returned to the bottom of ACL while cells requested are released from the top. Various subroutines and functions perform the list-processing functions such as the linking and delinking of cells. In our experiment on the 207 steps, it was found that not more than 160 cells were in use at any one time. This shows that we can easily reduce the size of our ACL if desired.

Experiments With Super-2

Super-2 was used to analyse the steps generated from the sixteen proofs shown in Table C.3. The analysis was carried out under the three different criteria for primitive T-steps listed above. The results under each criterion are shown in the table under the headings RUN-1, RUN-2 and RUN-3. These correspond to the mixed, the difference and the ratio criteria respectively.

RUN-1 was performed under the two different conditions:

(1) the MXC and MNC values are recomputed whenever required;

(2) the MXC and MNC are computed only once and saved.

The aim of this experiment is to assess the gain in speed derived from saving the MXC and MNC values. The times taken to check each step are shown in the table, TIME-A for condition (1) and TIME-B for condition (2). N-COMP in the table gives approximately the number of numeric tests involved in the analysis of each step under RUN-1.

Ignoring the algebraic steps, it can be seen that TIME-A is about 2½ to 3 times as large as TIME-B and that this ratio tends to be higher as the value of N-COMP gets larger. This experiment has demonstrated a significant gain in speed in saving the MXC and MNC values. It also shows that most of the steps can be checked in less than 0.1 second and often much less.

Super-2 has also been modified to perform the latent structure analysis shown in Table C.4.

Chapter VIII

CONCLUDING REMARKS AND RECOMMENDATIONS

We have described our work in computer-assisted instruction and our investigation into the problem of supervising trigonometric proofs in CAI.

Chapter II described UACAIS, our experimental CAI system and offered some suggestions for its modification and extension. In Chapter III the author language ALFIE was described. Chapter IV introduced the problem of supervising trigonometric proofs and discussed its context and scope. It also examined the problem of expression equivalence and developed the theory of consistency sets. Chapter V was devoted to the step-size problem. It described a schema for defining small step models. Chapter VI considered the problem of assisting the student in his proof and described an approach for developing an automatic proof constructor. Chapter VII dealt with some of the important practical problems in the implementation of our proposed proof supervisor.

Although we have spent far more time developing our CAI system than investigating proof supervision, it is the latter that is more important and original. The following points summarize the main significance of our work.

(1) UACAIS is the first major effort in CAI in Australia.

(2) ALFIE is a cue-oriented author language. This special feature makes it especially suitable for programming

the test-first-inform-later instructional paradigm. TFIL can be used as a vehicle for guided discovery teaching and for course revision. It is also a very cheap way of preparing adaptive, branching programs. The full potential of TFIL has yet to be explored.

(3) TPS is one of the earliest specific proposals for a mathematical supervisory subsystem for CAI.

(4) One of the most important contributions of this work is the development of a very simple numeric test for determining a set of basic identities that is "necessary and sufficient" for proving a given problem identity. This test is based on our theory of consistency sets and the representative point technique. The test also enables us to determine very cheaply whether a step is correct or not and to distinguish between A-steps and T-steps. Clearly the C-set of an identity gives the identity prover valuable look-ahead. However our numeric technique is appropriate only for a computer and not the human solver since a considerable amount of numeric computation is involved.

(5) We have been able to relate our theory of consistency sets and the associated numeric test to algebraic geometry. In particular the application of Hilbert's Nullstellensatz is interesting.

(6) We have developed a schema for defining small steps and we were able to derive good empirical models for

step-size from it. In particular we have derived Model-A and Model-B, the latter being much easier to implement than the former since it has a simpler definition for primitive T-steps.

(7) We have proposed an approach for generating trigonometric proofs which we believe to be effective and feasible. It utilises several numeric-oriented heuristics including C-set determination and latent structure probing.

(8) Latent structure is itself an interesting concept and it is very useful because it can be detected by simple numeric tests. It provides the identity prover with an additional measure of look-ahead.

(9) Our indexing scheme is a novel technique for imposing relevant structures on the Polish string of an expression, thereby expediting the various subexpression manipulation tasks encountered in TPS. Such an indexing scheme could well find new applications in the area of formula manipulation.

(10) The most significant contribution of our work in proof-supervision is perhaps in its indirect, long term implications. We have adopted a largely numeric approach for solving problems that are essentially symbolic. The numeric approach should be used for detecting subexpression equivalence in program compilation where the potential benefits have to be traded-off against the cost of deriving them. More importantly however

it is hoped that our work here will stimulate others to look into the possibility of using numeric techniques for solving symbolic problems.

Suggestions for Further Work

In this research we have not been able to carry out fully the various experiments and investigations. Also our efforts so far have opened up new areas to explore. We would like to suggest follow-up studies in the following :

(1) In the area of CAI, to investigate the test-first-inform-later instructional technique using ALFIE. Experiments should be conducted to assess the comparative effectiveness of TFIL. Other areas of potential application of TFIL should also be examined.

(2) TPS should be implemented. Initially this may be done off-line to check out the proof-checking and proof-generating components first. Eventually however it should be integrated into UACAIS so that its performance in a CAI environment may be studied. In particular we wish to ensure its real-time capability. Many of the essential ideas of TPS have already been successfully implemented in Super-2. Several others have also been tested separately. We therefore foresee no feasibility problem in the implementation of TPS. For step-size analysis, we would urge the adoption of Model-B.

(3) We should investigate the effectiveness of TPS as an

aid in the teaching of trigonometry. How does TPS-assisted teaching compare with the traditional method of teaching the subject? Is the effectiveness of TPS adversely affected by its communication interface? Is the keyboard a serious impediment to the usefulness of TPS?

(4) Extend our numeric approach for proof supervision to the multi-argument trigonometric problems, in which the addition, product and multiple argument formulae are permitted. Here we are no longer working with only twelve reference identities. An indefinite number of reference identities may have to be considered since the number of arguments may vary. We have already looked into the problem partially and have found the idea of independence and interdependence of classes of arguments relevant. This is to detect classes of relevant reference identities.

(5) Attempt extending our numeric approach to other areas of mathematics. The most immediate area is that of ~~are~~ the hyperbolic functions since these in many ways resemble the trigonometric functions. However it would be interesting if we could devise cheap numeric techniques to aid in symbolic (i.e. formal) differentiation and integration, in contrast to the approach of Slagle [66].

(6) Attempt to extend our numeric approach philosophy to traditionally symbolic problems with the aim of

finding better and cheaper methods of solution. One target area is the field of formula manipulation [10].

APPENDIX A

A
MANUAL AND AUTHOR GUIDE
FOR
ALFIE

ADELAIDE LANGUAGE FOR INSTRUCTION AND EDUCATION

(VERSION 1.0)

FEBRUARY 1969

2ND REVISION - OCTOBER 1970

LEE KIM CHENG

DEPARTMENT OF COMPUTING SCIENCE

UNIVERSITY OF ADELAIDE

INTRODUCTION

ALFIE, the Adelaide Language for Instruction and Education is the language for writing teaching programs for the University of Adelaide Computer-Assisted Instruction System (UACAIS). The version being described is designated ALFIE 1.0 and represents the latest development in the language as of October 1970.

This manual is intended to be both an author guide and a description of ALFIE. Course preparation will be described in the context of the card medium only; paper tape and on-line keyboard entry facilities are not yet available.

An author language is an integral part of a CAI system. It is designed to give the author convenient access to the various CAI facilities in a system. These facilities usually include those for presenting stimulus material (e.g. typewriter text, CRT display, etc.), analysing student responses, sequencing course material, branching, and recording student responses and performance data.

In designing CAI languages, a popular objective has been to make them easy to learn and convenient to use. Thus users are not required to be familiar with computers. As far as possible, subject to the limitations imposed by available hardware, ALFIE has been designed with this in mind.

OVERVIEW OF UACAIS

UACAIS is a dedicated CAI system based on the computing centre's CDC 6400 machine. Thus when under CAI operation, it may not process non-CAI jobs. The system has been designed to drive up to 512 remote student consoles - of which four are implemented at present. Each console consists of an IBM Selectric typewriter, but other pieces of terminal equipment may be added in future.

Courses are prepared off-line on cards and are then assembled by the ALFIE course compiler onto magnetic tape for subsequent use under UACAIS. During CAI operation, required courses are loaded from magnetic tape onto the disc, ready for use.

Students sitting at the remote consoles receive their lessons on the typewriter and communicate with the system via the keyboard. Different students may be receiving different segments of a course simultaneously and independently. More than one course may be conducted by the system at any one time.

TYPEWRITER CHARACTERS & CONTROL FUNCTIONS

The characters available on the typewriters are :-

(1) Alphabetic A B C ... X Y Z a b c ... x y z

(2) Numeric 1 2 3 ... 7 8 9 0

(3) Special + - * / ( ) $ = blank , . [ ] # : ' ° &
_ " % @ ¢ ? ! ;

Available control functions are: carriage return, index, tab, backspace and black and red ribbon select.

Textual material must be composed from this set of characters and functions only. Since not all these are available on a standard card punch, a card code for representing them is required. Table A.1 gives the list of all available typewriter characters and functions and their card codes, based on the IBM 029 punch, as well as other codes, as explained below :-

Table Column	Description
1	Hollerith Punch
2	Key or key combination on IBM 029 which gives this Hollerith punch
3	corresponding CDC 6400 display code
4	corresponding line-printer character
5	corresponding 7-bit typewriter code
6	corresponding typewriter character or function

HOLLERITH	029 KEY	DC	DC CHAR	TW CODE	TW CHAR	REMARK
12-1	A	01	A	x16	A/a	x=0 for upper and
12-2	B	02	B	x01	B/b	x=1 for lower cases
12-3	C	03	C	x15	C/c	
12-4	D	04	D	x55	D/d	
12-5	E	05	E	x51	E/e	

HOLLERITH	029 KEY	DC	DC CHAR	TW CODE	TW CHAR	REMARK
12-6	F	06	F	x34	F/f	
12-7	G	07	G	x74	G/g	
12-8	H	10	H	x41	H/h	
12-9	I	11	I	x12	I/i	
11-1	J	12	J	x70	J/j	
11-2	K	13	K	x11	K/k	
11-3	L	14	L	x45	L/l	
11-4	M	15	M	x76	M/m	
11-5	N	16	N	x31	N/n	
11-6	O	17	O	x46	O/o	
11-7	P	20	P	x50	P/p	
11-8	Q	21	Q	x10	Q/q	
11-9	R	22	R	x56	R/r	
0-2	S	23	S	x42	S/s	
0-3	T	24	T	x71	T/t	
0-4	U	25	U	x35	U/u	
0-5	V	26	V	x36	V/v	
0-6	W	27	W	x02	W/w	
0-7	X	30	X	x75	X/x	
0-8	Y	31	Y	x40	Y/y	
0-9	Z	32	Z	x73	Z/z	
0	0	33	0	143	0	
1	1	34	1	177	1	
2	2	35	2	133	2	
3	3	36	3	137	3	
4	4	37	4	147	4	
5	5	40	5	153	5	
6	6	41	6	113	6	
7	7	42	7	157	7	
8	8	43	8	117	8	
9	9	44	9	103	9	
12	+	45	+	030	+	
11	-	46	-	100	-	
11-8-4	*	47	*	017	*	
0-1	/	50	/	144	/	
0-8-4	(	51	(	003	(	
12-8-4	)	52	)	043	)	
11-8-3	$	53	$	047	$	
8-3	=	54	=	130	=	
no punch	blank	55	blank	020	blank	
0-8-3	,	56	,	014	,	
12-8-3	.	57	.	032	.	
0-8-6	S̄	60				lower case flag
8-7	C̄	61		077	[	
0-8-2	T̄	62	≡	037	#	
8-2	D̄	63	:	054	:	
8-4	₸	64	≠	152	'	single quote
0-8-5	W̄	65	→	072	°or¼	

HOLLERITH	029 KEY	DC	DC CHAR	TW CODE	TW CHAR	REMARK
11-0	B̄	66		057	&	
11-8-5	Ē	70	↑			upper case flag
11-8-6	F̄	71	↓	052	"	double quote
12-0	R̄	72				null character
11-8-7	Ḡ	73	>	053	%	
8-5	Ā	74	≤	013	¢	
12-8-5	N̄	75	≥	044	?	
12-8-6	Ō	76	¬	172	¡or ½	
12-8-7	Ȳ	77	;	154	;	
	X̄A			033	@	
	X̄C			024		carriage return
	X̄I			023		index
	X̄R			026		red ribbon
	X̄B			027		black ribbon
	X̄T			025		tab
	X̄-			000	—	underscore
	X̄X̄					end-of-card
	X̄			021		a backspace

The specification X̄ny or X̄nny where n is any digit 0,1,..,9 and y is any of C,A,I,R,B,T, - and blank means n or nn X̄y's where the meaning of X̄y for various y is as given above. Any X̄v not corresponding to any of the above combinations will be treated as X̄X̄ - the end-of-card code.

TABLE A.1 CARD CODES

The 'x' used in column 5 takes the value 0 or 1, depending on whether the letter is in the upper or the lower case. In column 2, D̄ (d-bar) refers to the upper case of the D key on the IBM 029; similarly for S̄, Ē, R̄, etc. T̄ refers to the lower case of the '=' key.

There are two case flags: the upper, Ē, and the lower S̄. The presence of an Ē in a text stream (see below) indicates that all subsequent letters are in the upper case until an S̄ is encountered. Similarly when an S̄ occurs the lower case prevails until an Ē appears.

The R̄ punch is for the null character - and is used to suppress a space. It is useful for card correction. For instance, if we have 'SPEED' on a card, but discover that it should have been 'SPED' instead, we can effect the necessary correction by duplicating the card, but replacing one of the E's with an R̄ as in 'SPER̄D'.

X̄R selects the red ribbon and X̄B the black. X̄T sends a tab function. It is assumed that the tab spacing has been set at columns 11, 21, 31, 41, ... etc. X̄C, the carriage return, positions the typewriter carriage to the left margin of the next line. X̄X̄ is the end-of-card code, and indicates that subsequent punches on the card are to be ignored.

Whether we get a ¼ or ° for the code 072 depends on the typeball; some carry ° while others carry the ¼. Similar remarks apply to the T/W code 172.

The remaining codes in the table should be now self-explanatory.

THE TEXT STREAM

A text stream is a string of typewriter (T/W) codes to be output on the typewriter. It is punched as a set of successive textbody cards (TBCs), bounded by two keyword cards (KWCs). A line or a paragraph of text may be coded on one or more TBCs. The following example illustrates the coding of text in ALFIE. We assume all punches begin on column 2 of the cards here.

e.g. 1 Intended T/W output:
"The vowels of the alphabet are a, e, i, o and u." This may be coded as:

```
ĒTS̄HE VOWELS OF THE ALPHABET ARE A, E, I, O AND U.X̄X̄
```

The same output may be more wastefully coded on two cards as

```
HABET ARE A, E, I, O AND U.X̄X̄          card 2
ĒTS̄HE VOWELS OF THE ALPX̄X̄              card 1
```

The 1st punch Ē calls for capital letters, but this affects only the 1st letter 'T' since an S̄ is immediately encountered, setting all subsequent letters to the lower case. Note the use of X̄X̄.

e.g. 2 To output the following tabular text:

Country	Capital
Australia	Canberra
China	Peking
Canada	Ottawa
U.A.R.	Cairo

with the two columns set at 11 and 31. This table may be coded as :-

```
  ĒPS̄EKINGX̄CX̄TĒCS̄ANADAX̄TX̄TĒOS̄TTAWAX̄CX̄TĒU.A.R.X̄2TCS̄AIROX̄X̄  2nd card
 ĒXT̄COUNTRYX̄7 X̄7-X2TCAPITALX̄7 X̄7-X̄CX̄TAS̄USTRALIAX̄2TĒCSANBERRAX̄CX̄T ...
```

(Note the use of X̄ ,X̄T, X̄C and X̄- for backspace, tab, c/r and underscore.)

e.g. 3 To output 'sin(∅+θ)=0.1624" we need punch only:

```
S̄SIN(ĒOX̄ /+OX̄ -)=0.1624X̄X̄
```

A basic problem in the off-line preparation of text concerns the line limit and justification. It is indeed a tedious job for the author to keep track of his text, making sure that each line does not exceed the limit set; line-justification is even more involved. With an on-line keyboard entry facility, a larger part of this problem would simply be non-existent. As an aid for the ALFIE author, we have included a simple text-editing facility through the EDIT card.

=EDIT(n,m)

n, which takes a value between 40 and 80 inclusive, specifies the width over which a line of text is to be truncated and justified. If justified, the text will be dispersed randomly over columns 1 through n subject to mode m, where m can be one of the following:

m=0 - break string and resume on the next line when current word extends beyond column n. Do not distribute the words.

m=1 - truncate current line and continue on the next line when current word exceeds column n. Distribute words over the field (columns 1 to n) overriding any c/r which occurs after column n-7.

m=2 - as for m=1, but do not override any c/r.

Each EDIT specification holds at a new text stream, and applies to all subsequent text until a new EDIT is encountered. The default EDIT specification is '=EDIT(80,1)'.

The following listings (Figures A.1 and A.2) illustrate the use of the EDIT card. They also show the two listings produced by ALFIE .. one, Figure A.1, by the course preprocessor, and this is the source deck listing; the other, Figure A.2, is produced by the compiler proper. Note that in the source deck listing X̄ is shown as ∧ , Ē as ↑, and so on, as shown in Table A.1.

COMPILATION OF ALFIE PROGRAMS

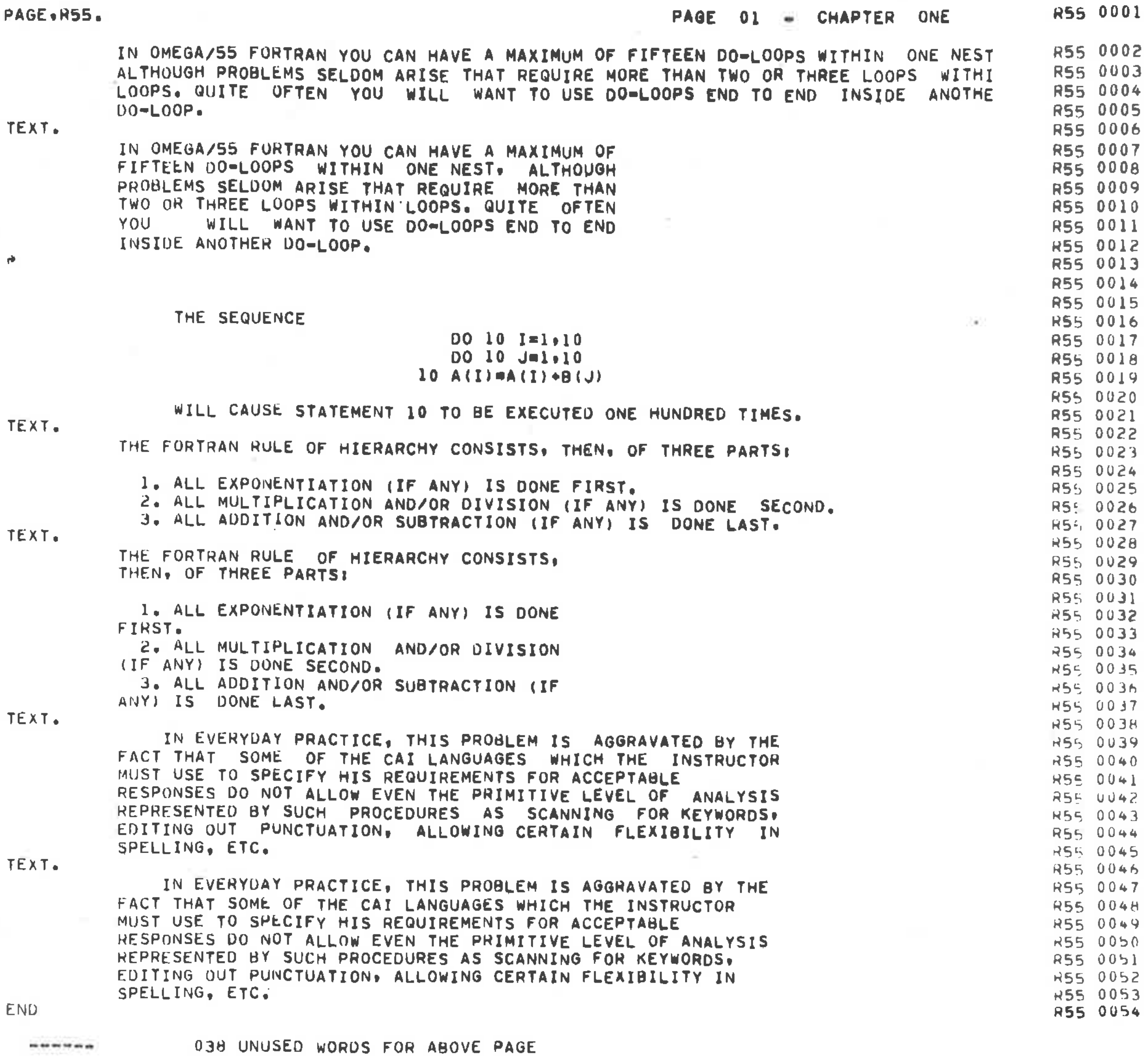
In preparing a course, we must first of all code it in detail. This course is then punched on cards, and the course deck is set up as a job deck (computer program) to be read by the card reader. Before it can be used, it must be checked and if satisfactory, converted into its object code form and output on magnetic tape. The course in its object code form, may then be loaded during a CAI run, to be executed interpretively by a resident central program.

The process of converting a course on deck into its object code form is called an assembly or a compilation, and the computer program which actually carries this out is called an assembler or a compiler.

A course is assembled in two stages. The first stage is performed by LKC, a preprocessor program for translating the course in its card format into a common intermediate code called CONCODE. The actual assembler ALF then carries out the second stage by accepting a course in CONCODE and producing the object codes. It can be seen that we have made provisions for a course to be prepared in other media; all that would be required for a new medium is a corresponding preprocessor for converting a course in that medium into the CONCODE. Figure A.3 illustrates the approach just discussed.

```
$COURSE,DEMON                                                              CARD  0001
$CHAPTER,ONE.                                                              CARD  0002
$PAGE,R55.                                                                 CARD  0003
 ↑I≡N ↑O≡MEGA/55 ↑FORTRAN ≡YOU CAN HAVE A MAXIMUM OF FIFTEEN ↑DO-≡LOOPS WITHIN O   CARD  0004
 NE NEST, ALTHOUGH PROBLEMS SELDOM ARISE THAT REQUIRE MORE THAN TWO OR THREE LOO   CARD  0005
PS WIT<<HIN LOOPS. ↑Q≡UITE OFTEN YOU WILL WANT TO USE ↑DO-≡LOOPS END TO END INSI   CARD  0006   TBC
 DE ANOTHER ↑DO-≡LOOP.^^                                                   CARD  0007
=EDIT(45,1)                                                                CARD  0008
$TEXT.                                                                     CARD  0009
 ↑I≡N ↑O≡MEGA/55 ↑FORTRAN ≡YOU CAN HAVE A MAXIMUM OF FIFTEEN ↑DO-≡LOOPS WITHIN O   CARD  0010
 NE NEST, ALTHOUGH PROBLEMS SELDOM ARISE THAT REQUIRE MORE THAN TWO OR THREE LOO   CARD  0011
PS WIT<<HIN LOOPS. ↑Q≡UITE OFTEN YOU WILL WANT TO USE ↑DO-≡LOOPS END TO END INSI   CARD  0012   TBC
 DE ANOTHER ↑DO-≡LOOP.^^                                                   CARD  0013
=EDIT(70,2)                                                                CARD  0014
↱                                                                          CARD  0015
 ^2C      ↑T≡HE SEQUENCE^C^3T↑DO 10 I=1,10^C^3TDO 10 J=1,10^C^3T^3 10 A(I)=A(I)+B   CARD  0016
 (J)^2C      ≡WILL CAUSE STATEMENT 10 TO BE EXECUTED ONE HUNDRED TIMES.^^       CARD  0017
=EDIT(65,1)                                                                CARD  0018
$TEXT.                                                                     CARD  0019
 ↑T≡HE ↑FORTRAN ≡RULE OF HIERARCHY CONSISTS, THEN, OF THREE PARTS:^2C  1. ↑A≡LL    CARD  0020
 EXPONENTIATION (IF ANY) IS DONE FIRST.^C  2. ↑A≡LL MULTIPLICATION AND/OR DIVISI   CARD  0021
 ON (IF ANY) IS DONE SECOND.^C  3. ↑A≡LL ADDITION AND/OR ^^                CARD  0022
 SUBTRACTION (IF ANY) IS <<<< DONE LAST.^^                                 CARD  0023
=EDIT(40,1)                                                                CARD  0024
$TEXT.                                                                     CARD  0025
 ↑T≡HE ↑FORTRAN ≡RULE OF HIERARCHY CONSISTS, THEN, OF THREE PARTS:^2C  1. ↑A≡LL    CARD  0026
 EXPONENTIATION (IF ANY) IS DONE FIRST.^C  2. ↑A≡LL MULTIPLICATION AND/OR DIVISI   CARD  0027
 ON (IF ANY) IS DONE SECOND.^C  3. ↑A≡LL ADDITION AND/OR ^^                CARD  0028
 SUBTRACTION (IF ANY) IS <<<< DONE LAST.^^                                 CARD  0029
=EDIT(60,1)                                                                CARD  0030
$TEXT.                                                                     CARD  0031
     ↑I≡N EVERYDAY PRACTICE, THIS PROBLEM IS AGGRAVATED BY THE^CFACT THAT SOME O   CARD  0032
 F THE ↑CAI ≡LANGUAGES WHICH THE INSTRUCTOR^CMUST USE TO SPECIFY HIS REQUIREMENT   CARD  0033
 S FOR ACCEPTABLE^CRESPONSES DO NOT ALLOW EVEN THE PRIMITIVE LEVEL OF ANALYSIS^C   CARD  0034
 REPRESENTED BY SUCH PROCEDURES AS SCANNING FOR KEYWORDS,^CEDITING OUT PUNCTUATI   CARD  0035
 ON, ALLOWING CERTAIN FLEXIBILITY IN ^CSPELLING, ETC.^^                    CARD  0036
=EDIT(60,2)                                                                CARD  0037
$TEXT.                                                                     CARD  0038
     ↑I≡N EVERYDAY PRACTICE, THIS PROBLEM IS AGGRAVATED BY THE^CFACT THAT SOME O   CARD  0039
 F THE ↑CAI ≡LANGUAGES WHICH THE INSTRUCTOR^CMUST USE TO SPECIFY HIS REQUIREMENT   CARD  0040
 S FOR ACCEPTABLE^CRESPONSES DO NOT ALLOW EVEN THE PRIMITIVE LEVEL OF ANALYSIS^C   CARD  0041
 REPRESENTED BY SUCH PROCEDURES AS SCANNING FOR KEYWORDS.^CEDITING OUT PUNCTUATI   CARD  0042
 ON, ALLOWING CERTAIN FLEXIBILITY IN ^CSPELLING, ETC.^^                    CARD  0043
$END                                                                       CARD  0044
```

Figure A.1 Source Card Listing of a Course



```
PAGE.R55.                                                       PAGE  01  -  CHAPTER  ONE
          IN OMEGA/55 FORTRAN YOU CAN HAVE A MAXIMUM OF FIFTEEN DO-LOOPS WITHIN  ONE NEST
          ALTHOUGH PROBLEMS SELDOM ARISE THAT REQUIRE MORE THAN TWO OR THREE LOOPS  WITHI
          LOOPS. QUITE  OFTEN  YOU  WILL  WANT TO USE DO-LOOPS END TO END  INSIDE  ANOTHE
          DO-LOOP.
TEXT.
          IN OMEGA/55 FORTRAN YOU CAN HAVE A MAXIMUM OF
          FIFTEEN DO-LOOPS  WITHIN  ONE NEST.  ALTHOUGH
          PROBLEMS SELDOM ARISE THAT REQUIRE  MORE THAN
          TWO OR THREE LOOPS WITHIN LOOPS. QUITE  OFTEN
          YOU     WILL  WANT TO USE DO-LOOPS END TO END
          INSIDE ANOTHER DO-LOOP.
*



               THE SEQUENCE
                                         DO 10 I=1,10
                                         DO 10 J=1,10
                                      10 A(I)=A(I)+B(J)

               WILL CAUSE STATEMENT 10 TO BE EXECUTED ONE HUNDRED TIMES.
TEXT.
          THE FORTRAN RULE OF HIERARCHY CONSISTS, THEN, OF THREE PARTS:

            1. ALL EXPONENTIATION (IF ANY) IS DONE FIRST.
            2. ALL MULTIPLICATION AND/OR DIVISION (IF ANY) IS DONE  SECOND.
            3. ALL ADDITION AND/OR SUBTRACTION (IF ANY) IS  DONE LAST.
TEXT.
          THE FORTRAN RULE  OF HIERARCHY CONSISTS,
          THEN, OF THREE PARTS:

            1. ALL EXPONENTIATION (IF ANY) IS DONE
          FIRST.
            2. ALL MULTIPLICATION  AND/OR DIVISION
          (IF ANY) IS DONE SECOND.
            3. ALL ADDITION AND/OR SUBTRACTION (IF
          ANY) IS  DONE LAST.
TEXT.
              IN EVERYDAY PRACTICE, THIS PROBLEM IS  AGGRAVATED BY THE
          FACT THAT  SOME  OF THE CAI LANGUAGES  WHICH THE  INSTRUCTOR
          MUST USE TO SPECIFY HIS REQUIREMENTS FOR ACCEPTABLE
          RESPONSES DO NOT ALLOW EVEN THE PRIMITIVE LEVEL OF  ANALYSIS
          REPRESENTED BY SUCH  PROCEDURES  AS  SCANNING  FOR KEYWORDS,
          EDITING OUT  PUNCTUATION,  ALLOWING CERTAIN  FLEXIBILITY  IN
          SPELLING, ETC.
TEXT.
              IN EVERYDAY PRACTICE, THIS PROBLEM IS AGGRAVATED BY THE
          FACT THAT SOME OF THE CAI LANGUAGES WHICH THE INSTRUCTOR
          MUST USE TO SPECIFY HIS REQUIREMENTS FOR ACCEPTABLE
          RESPONSES DO NOT ALLOW EVEN THE PRIMITIVE LEVEL OF ANALYSIS
          REPRESENTED BY SUCH PROCEDURES AS SCANNING FOR KEYWORDS,
          EDITING OUT PUNCTUATION, ALLOWING CERTAIN FLEXIBILITY IN
          SPELLING, ETC.
END
  ------          038 UNUSED WORDS FOR ABOVE PAGE
```

Figure A.2 Assembly Listing of a Course

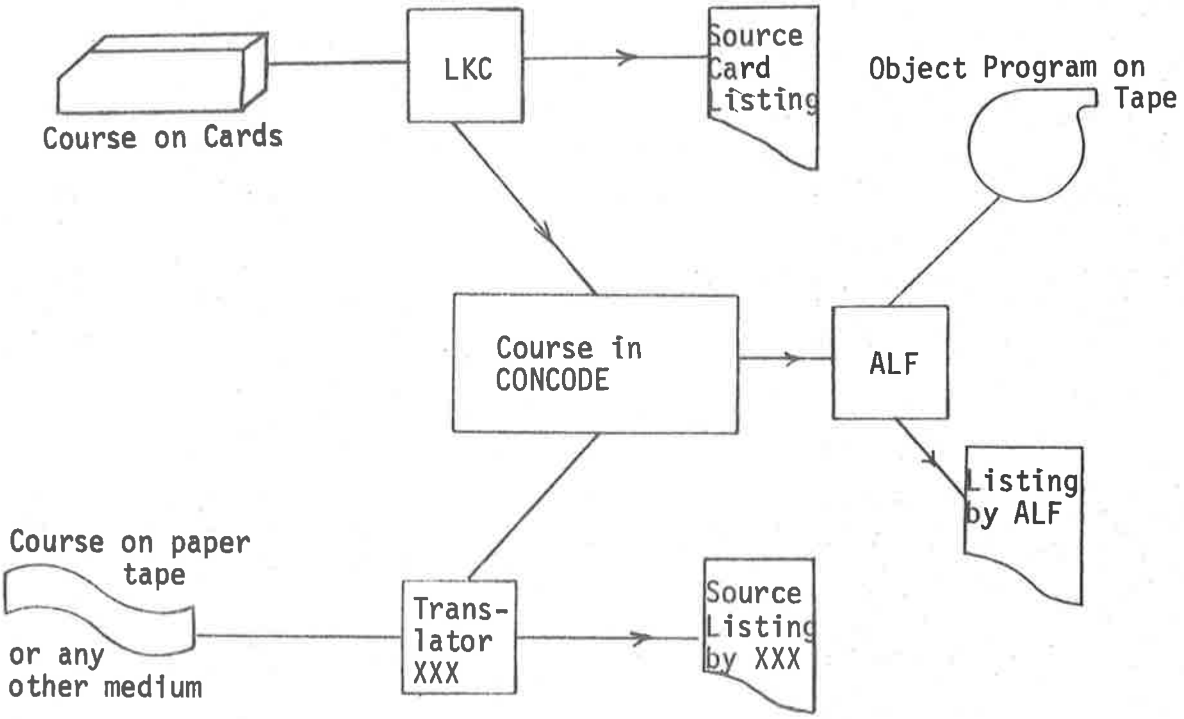


Figure A.3 Compiling an ALFIE Course

Two kinds of listings are produced when a course is compiled. One is the course card listing produced by LKC as illustrated in Figure A.1. The source card listing can be produced or suppressed by the listing control card (see below). When no LCC is present, the listing status is assumed to be 'on'. The other listing is the assembly listing, produced by ALF as shown in Figure A.2. It will be seen that this listing of the text simulates as closely as the line-printer will permit, its appearance on the typewriter. Thus while spacing, backspace and tab etc. are reflected exactly, this listing is unable to show alphabetic case changes, ribbon changes and certain special typewriter characters like '&' and '?'. The assembly listing can be suppressed and reinstated at will by the LIST and NOLIST cards. The initial setting is for listing to be produced.

The assembler listing is an important debugging aid for ALFIE programs and greatly compensates for the very ungainly card codes for text.

LKC distinguishes five kinds of course cards, by examining the first column of the cards. These five kinds are :-

(1) Listing Control Card (LCC): This has a 2-8 (D̄ on IBM 029) punch on column 1. The LCC acts as a source listing switch, turning the listing status to 'on' if it is 'off' and vice versa. It does not produce any object code.

(2) Comment Card (CC): A comment card is characterised by a * on column 1 and is used for passing comments on the source and assembly listings. The whole 80 columns of a CC is listed, if the listing status is 'on'. No object code is produced for the CC.

(3) Edit Card (EC): Has '=' on column 1 and 'EDIT' on the next four columns. Its use has been explained above.

(4) Keyword Card (KWC): A card with a '$' or '→' (W̄ on 029) on column 1, and a 'C' if it is a continuation KWC. KWCs are used for specifying course directives. In a KWC, blanks are ignored except in an answer string.

(5) Textbody Card (TBC): A TBC is a card which is not an LCC, CC, EC or KWC and usually with column 1 blank. If column 1 is not blank, then a warning 'TBC' is tagged alongside its source listing. If column 1 of a TBC is blank, then its contents are assumed to begin on column 2 and end on the first end-of-card code X̄X̄, or on column 80 in the absence of the X̄X̄.

ALFIE THE LANGUAGE

A course in ALFIE is a sequence of directives and texts. Directives are KWCs while texts are made up of TBCs, as explained earlier. A course is organised into an arbitrary number of named chapters, each of which comprise of from one to sixty-four named pages.

A KWC begins with a '$' or a '→' (W̄ on the IBM 029) on column 1, and any continuation KWC must bear a 'C' on column 1. Except in answer specifications, all blanks in a KWC are ignored.

Each directive is associated with a keyword and belongs to one of the three categories :-

(1) listing control
(2) course organisational
(3) command.

We shall now describe the various KWCs in ALFIE and their syntax.

Some BNF Definitions:

```
<alphabetic>   ::= A|B|C|D|....|X|Y|Z
<digit>        ::= 0|1|2|.... 7|8|9
<alphanumeric> ::= <alphabetic>|<digit>
<special>      ::= +|-|*|/|(|)|$|=| |,|.|≡|[|]|:|≠|→|∨|∧|↑|↓|<|>|≤|≥|¬|;
<display code> ::= <alphabetic>|<digit>|<special>
<identifier>   ::= <alphanumeric>|<identifier><alphanumeric>
```

'<', '>', '::=' and '|' above are called metalinguistic symbols and <alphabetic> and <identifier> etc., are metalinguistic variables. <alphabetic> ::= A|B|C|....|X|Y|Z simply means that the variable '<alphabetic>' may take the letter 'A' or 'B' or 'C' or .... or 'Z'. Similarly the variable identifier represents any non-null alphanumeric string. Note that the latter variable is defined recursively - i.e. in terms of itself.
e.g. of '<identifier>' : 'A', '232', 'PX24' and 'FREDERICI'.

We shall also define <name> to be any <identifier> not exceeding 6 characters long. A <name> is used for a label, chapter name or page name. Thus we take <pagename> and <chapter name> to be <name>.

In ALFIE command KWC's may be labelled. A label is any <name> followed by a colon, preceding a keyword.
e.g. `$LX2:PROBLEM,Q02.` A label may also appear by itself as in `$TGX:` and serves to define a page location. Apart from the implicit label '→' which appears on column 1 of a KWC, all labels in a given page must be unique. The use of the implicit label will become clear later.

We shall now describe each keyword and the form of its associated command. Before doing so, we explain the 'metalanguage' used. The metalinguistic variable <--> has been explained earlier. The brace pair {--} means that one of its contained options must occur. [x] means that the x may, but need not, occur. Other characters shown must occur literally. Blanks not in answer specifications, will be ignored in a KWC. The column 1 characters of the KWC (viz. '$', '→' or 'C') will not be shown, but is taken as understood.

e.g.: GOTO,CHAPTER({*|<chaptername>})[,PAGE({*|<pagename>})]
means that the string 'GOTO,CHAPTER(' must occur, to be followed by either a '*' or a <chaptername>, and then ')', to be further followed optionally by: ',PAGE(', an '*' or a <pagename>, and then a ')'.

Examples of this command are:-

```
GOTO,CHAPTER(S7),PAGE(X02)
GOTO,CHAPTER(PT2)
GOTO,CHAPTER(*),PAGE(12)
```

But the following are invalid:-

```
GOTOCHAPTER(PT2)    (no comma)
GOTO,CHAPTER(VERY LONG)  (chapter name too long)
```

Any command not terminated by a ')' may be terminated, but only if desired, by any special character, usually the period. Thus 'LIST', 'LIST.' and 'LIST*' are all equivalent.

The listing control keywords are LIST and NOLIST. These control only the assembler listing and should not be confused with the LCC which affects only the source card listing. We shall refer to non-listing control KWCs as proper.

LIST

e.g. $LIST
This is a request to the assembler to produce the assembler listing of all subsequent sections of the course, until a NOLIST is encountered.

NOLIST

e.g. $NOLIST
This directive is a request to suppress assembler listing of all subsequent course material until a LIST is encountered.

The course organisational keywords are COURSE, CHAPTER, PAGE and END.

COURSE,<coursename>

e.g. $COURSE,ALGEBRA
$COURSE,BOOLEAN

This card begins and names a course. The <coursename> is any <identifier> not exceeding 10 characters. The COURSE must be the first KWC of a course. Following a course KWC, the next KWC, other than listing control, must be the CHAPTER. In a given CAI run, all course names must be distinct.

CHAPTER, <chaptername>

e.g. $CHAPTER,PD003.

This KWC begins and names a new chapter and ends the previous chapter (if any). In a given course, chapter names must be unique. If a chapter name is not acceptable, for reason of syntax or multiplicity, the system generates a name of the form SYSCHn where n = 1 for the first system-given chapter name in a course, n = 2 for the second and so on. The proper KWC which immediately succeeds a CHAPTER must be a PAGE. A course is allowed any number of chapters.

PAGE, <pagename>

e.g. $PAGE(RR22)

This KWC begins and names a new page of a course, and terminates the previous page (if any). Each chapter may contain as many as 64 pages, but at least one. Page names within a chapter must be unique. If the author-supplied pagename is unacceptable for some reason, then a system-defined name of the form SYSxxx is substituted. The first such page in a chapter is SYS001, the next SYS002 and so on.

A page is a unit of a course which can be assembled into a block of central memory words less than or equal to the limit PGSIZE (currently PGSIZE = 464). A page constitutes a segment of a course that may reside in core at any one time during CAI operation. It is also the unit of course which is swapped between disc and core. The first word of a page contains its name, i.e. <pagename>, in display code, left justified zero filled.

If the author attempts to assemble more than PGSIZE words of code into a page, the assembler will try to split in into two (or more) pages, if possible. It does this by accommodating as much into the current page as possible, and attempts to place the remaining codes in a new system-defined page. In splitting a page, a problem block (see later) must not be split between pages. A page which cannot be split without violating this requirement will be abandoned.

The assembler automatically assembles at the end of each page an exit to the next page, unless this is the last page, or its last KWC is COMPLETE or END. The next page above includes the first page of the next chapter, if the current page is the last in a chapter. In the assembler listing of a page, the location at which each command in the page is assembled is also shown. The listing ends with the number of unused central memory words in the page (see Figure A.2).

END

This keyword ends the current course - this means also that the current page and chapter are also terminated.

As a further illustration of the organisational KWCs, the course listed in Figure A.1 consists of only one chapter, and one page.

The remaining keywords are mainly command keywords. Each is assembled into one or more central words, the first of which must bear a 6-bit opcode, to the far left. Succeeding cards in a page are assembled into succeeding locations, in their order of appearance. All KWCs assemble into one central memory word unless stated otherwise.

SIGNOFF opcode: 00

e.g. $L2:SIGNOFF.

This commands the CAI system to sign-off the student. When a student is signed off he cannot resume his lesson until he signs on again.

TEXT[T=<delay>] opcode: 01

e.g. $TEXT(T=24)

→TEXT(T=*)

This modifies the immediately following text stream. If the next card is not a TBC, i.e. no such text stream follows, then the KWC is ignored. However a text stream need not be preceded by a TEXT card, in which case '$TEXT.' is assumed to occur. <delay> is a number taking an integer value from 0 through 999, and represents the time the system must wait on presenting the text stream, before resuming the execution (interpretation) of the course. If no T-parameter is specified, T=1 is assumed; i.e. '$TEXT' is equivalent to '$TEXT(T=1)'. The special form T=* implies an infinite delay. The system will resume execution of the course prior to the expiry of the delay, if the student types in '#GO'.

A 'TEXT' without an accompanying text stream causes no object code to be produced. With its text stream, it is assembled as follows. The typewriter codes of the text stream are assembled into as many text groups as required. Each text group contains two header words followed by up to 16 central memory words in which the T/W codes are packed 5 per word. Thus eac~~t~~h text group can accommodate up to 80 T/W codes and a text stream with n T/W codes occupies N central memory words where:

$$N = 18 \times [n/80] + [\tfrac{1}{5}(n-1-80 \times [n/80])] + 1.$$

$[x]$ stands for the largest integer not exceeding x.

e.g. 97 T/W codes will take up $18 + [\tfrac{1}{5} \times 16] + 1 = 22$ words.

The CUE, ANSWER, ENDANS, WAITCUE and GIVECUE commands must be used only in a problem environment - a loose concept to be explained later.

{CUE|EXPLAIN|EXP} [,<n>] [(T=<delay>)]   opcode: 02

```
e.g. $LX:CUE(T=15)
     $CUE,4(T=10)
     $EXP.
```

This instruction will be referred to as a cue. The keywords CUE, EXPLAIN and EXP are treated as equivalent. <n> takes any integer from 1 to 99. <delay> is as for TEXT; if the T-parameter is not specified, T=* is assumed. The <n> is a repetition factor and indicates the number of 'single' cues intended. Thus '$CUE,4(T=12)' is equivalent to four successive '$CUE,(T=12)' cards saving three cards. Each single cue assembles into 1 word; thus '$CUE,3.' takes three words.

A cue must be part of a problem block (q.v.). When it is executed, a red question mark is typed out on the left margin of a new line, indicating to the student to respond with an answer. The student must then respond within the delay time of the cue or else he forfeits his chance to answer; in this latter case the course execution will resume from immediately after the cue. When an answer is received, the system next executes the first answer command in the associated answer sequence (q.v.). This then in effect triggers off (see execution of answer command) a comparison of the student's answer againstthe author answers, one after another in the answer sequence, until either (a) a match is made or (b) there is no more author answer to compare. In the case (a) the course will continue from immediately after the matching answer command. In the case (b), no match has been achieved and the system types out the message "Incorrect" and then resumes execution from the cue just executed. What follows the cue would usually be a piece of explanatory or prompting text - whence the keywords CUE and EXPLAIN.

The following is a simple example of the use of cues in the context of a problem block. This is followed by two hypothetical student interactions. The italicised texts within square brackets are comments for the reader.

COURSE SEGMENT

```
    $TEXT.
     Kuala Lumpur is the capital of ______?
    $CUE.
     M_______. Please try again.
    $CUE(T=10)
     MAL_____. Try again.
    $EXPLAIN(T=10)
     The answer is MALAYSIA.
    $GOTO,L20.
    $ANSWER(M=3)*MALAYA*
```

```
 Not quite.  Malaya is now part of another country.
$GIVECUE.
$ENS(M=3)$MALAYSIA$
 Very good.
$L20:TEXT.
 In what year ......
```

Interaction 1

```
SYSTEM:    Kuala Lumpur is the capital of ________?
STUDENT:   ?BURMA  [The '?' from the system is in red; student ends
                    answer with a carriage return.]
SYSTEM:    Incorrect.
           M_______.  Please try again.
           ?  [Student fails to respond within 10 seconds.]
SYSTEM:    MAL_____.  Try again.
STUDENT:   ?MALAYA
SYSTEM:    Not quite.  Malaya is now part of another country.
           The answer is MALAYSIA.
           In what year .....
```

Interaction 2

```
SYSTEM:    Kuala Lumpur is the capital of ________?
STUDENT:   ?SINGAPORE
SYSTEM:    Incorrect.
           M_______.  Please try again.
STUDENT:   ?MALAYA
SYSTEM:    Not quite.  Malaya is now part of another country.
           MAL_____.  Try again.
STUDENT:   ?MALAYSIA
SYSTEM:    Very good.
           In what year .....
```

```
{ANSWER|ANS|ENDANS|ENS}{[<t>]|([M=<mode>][,C=<category>])}<answer specn>
e.g. $ANS(M=2,C=3)*PERMUTATION*                          opcode: 03
     $ANSWER(C=4)*WARM*FRIENDLY*GENIAL*
     $ENS.=BIG=LARGE=
     $ANSWER(M=6)$123565$2434$
```

The keywords ANSWER and ANS are equivalent; so are ENDANS and ENS. The instruction forms are only approximately represented by the above specification; so further explanation is in order. <t> is either ',' or '.'. <mode> is an integer, at present taking one of the values 0,1,2, ...9,10,11,12,13. Other modes have not been yet implemented. <category> may take any of the integer values 0,1,2,...63. When not specified, the default values for M is 3 and C is 0. The C-parameter and M-parameter may appear in any order.

An <answer specn> consists of one or more answer strings bounded and separated by a common delimitter character. An answer string is made up of any CDC 6400 display code characters. Imbedded blanks are significant. A delimitter can be any display code character not used in the answer strings. An answer specification has the form: $<answer string 1>$<answer string 2>$.....$<answer string n>$ where $ is the delimitter character. The second and subsequent answer strings, if present, may not begin with a blank - if one does, then it, and all subsequent strings in the specification will be ignored. Thus $ANS(M=2)*AA* AB*AC* will be treated as $ANS(M=2)*AA* only. An answer instruction with n answer strings is equivalent to n separate answer instructions, containing only one of the n answer strings each, and with the same M and C parameters. Thus $ANS(M=2,C=5)*A01*A02*A03* is equivalent to the three successive instructions: $ANS(M=2,C=5)*A01*, $ANS(M=2,C=5)*A02*, and $ANS(M=2,C=5)*A03*. But note that $ENS(--)*JACK* JILL* is equivalent to $ANS(--)*JACK* and $ENS(--)*JILL* in that order.

The C-parameter is included in any recording of a response match against an answer. It is useful for classifying answers in a course. For instance, the author may designate the categories 0,1,2 and 3 for incorrect, partially correct, acceptable and best-match answers respectively.

When an answer command (ANSWER or ENDANS) is executed, the student's input string will be compared with the author's answer string (or strings), under the prescribed mode. The outcome is either a match or a mismatch.

A match is always obtained when an answer command is executed in the absence of a student answer. If a match is made, the next command to be executed is the one immediately following the answer command. If there is a mismatch then the next action depends on whether or not this is the last answer command in the problem block. If it is, then the message "Incorrect" is output and the course resumes immediately after the last cue executed. We have not discussed answer recording here to avoid complicating the above explanations.

To understand the execution of an answer command in the absence of a student answer, consider the answer sequence :

```
$ANS(M=3)*A*
 Message-1
$ANS(M=3)*B*
 Message-2
$ENS(M=3)*C*
 Message-3.
```

If the student's answer is 'A', and the first answer command is executed, a match is obtained. The next command to be executed is the one immediately following this first answer command, and this is the text "Message-1"; this text is typed out. The next command is then executed - this is the second answer, and since there is now no student answer (his answer 'A' has been successfully matched with the first answer command and is no longer active), a match is assumed, and therefore "Message-2" is next typed out, followed thereafter by "Message-3", since a match is again assumed with the last answer command (ENS).

Currently we have only the answer processing modes M=0,1,2... 12,13,14 to meet the immediate needs of an experimental system. These modes have the functions described below :-

Mode — Function

0 — Permitted only in an ENDANS. In Mode 0, there is no answer string associated with the instruction. Any response will match a mode 0 answer - i.e. it is a catch-all. e.g. $ENS(M=0).

Mode	Function
1	Perform an exact string match of the student's response, without prior edit~~t~~ing, against the author answers, e.g. $ANS(M=1)*NEIN* will match 'NEIN' but not 'NE IN' nor 'MEIT'.
2	As for M=1, but remove blanks from student response prior to string matching, e.g. $ANS(M=2)*ALPHA* will match any of 'ALPHA', 'AL PH A' and 'ALP HA' but not 'BETA' or 'GAMMA'.
3	This is the <u>default mode</u> - i.e. if no mode is specified by the author, M=3 is assumed. As for M=2, but in addition, any , ; : ? " ' and . are removed from the student string before it is matched. e.g. $ANS(M=3)ZALPHAZ will match 'AL,.PHA' and 'ALPHA.'.
4	Remove from student's answer any non-alphanumeric before comparison. Clearly then $ANS(M=4)*F44.6* will never match any student response as any '.' present in it will be removed.
5	As for M=4, alphanumerics only. But student responses may be truncated on the right before matching. Thus the response 'PROGRAMME' will match '$ANS(M=5)*PROGRAM*.
6	All non-numeric characters to be removed prior to matching. Thus an author answer string containing a non-numeric will fail to match all student answers.
7	Remove any blank, ',', ';' and ':' before matching. Thus 'TWO;THREE' but not 'TWO.THREE' will match $ENS(M=7)↑TWOTHREE↑.
8	Remove blanks and if necessary truncate trailing zeroes, before matching. Thus '46.2300' but not '46.23243' will match $ANS(M=8)*46.23*.
9	This mode is for the recognition of a list of answer items, which may be separated by blanks, ',', ';' or ':', and in which the items are allowed to occur in any order. In the author list, the items are separated by T̄ (T-bar on IBM-029).

Mode Function

e.g. (1) $ANS(M=9)BLUET̄REDT̄GREEN* matches 'BLUE,GREEN,RED' and 'RED; BLUE:,GREEN'.

(2) $ANS(M=9)*ALPT̄BETT̄GAM*PPT̄QQT̄RRT̄SS* is equivalent to $ANS(M=9)*ALPT̄GAMT̄BET* and $ANS(M=9)*PPT̄QQT̄RRT̄SS* in succession.

10 This is for keyword match; author strings and keywords, whose occurrence are searched for in the student string; if search is successful, a match is said to be made.
e.g. $ANS(M=10)*HAPPY* matches 'I AM HAPPY'.

11 This is for matching a number; the student's number must be less than the author answer. e.g. '14.64' and '7' match $ANS(M=11)*15.0* but not '15.01' nor '20'.

12 Same as for M=11, except that the student answer must be greater than the author's number.

13 In this mode, each answer string is two numbers separated by a semi-colon. Student answers match if they are between these two numbers in value. The author does not have to place the number pair in numerical order. Thus $ANS(M=13)*12;13* and $ANS(M=13)*13;12* are equivalent and will match '12.23' but not '13.4'.

14 This is for algebraic expression match. The author string is an algebraic expression. A student answer matches if it is algebraically equivalent to it.
e.g. $ANS(M=14)*(A+B)↑2* matches '(B+A)↑2', 'A↑2+B↑2+2*A*B' and '(2*B+(A-B))*(B+A)' but not '2*(A+B)'.

In an answer sequence (q.v.), the last answer instruction must be an ENDANS. Thus an answer sequence with only one answer instruction consists of only an ENDANS.

The number of words taken to assemble an answer instruction depends on the number of answer strings and the lengths of these strings. Each string is assembled as a separate answer block, made up of one header word, and one or more words to accommodate the actual answer string, packed 10 per word, with a zero byte (6-bits) to terminate it. Thus *ANSWER(M=3)*SHORT* takes 2 words, $ANS(M=2)*BELLYLONGSTLING* requires 3 full words and $ENS(M=0) 1 word.

Unconditional Branch:

There are three unconditional branch instructions, facilitating

(a) intra-page,

(b) inter-page, but intra-chapter, and

(c) inter-chapter transfers.

GOTO{<name>|,|→|→} opcode: 04

e.g. (1) $GOTO,L4 (2) $GOTO→

This instruction effects a branch to another instruction within a page. <name> refers to a label in the page. When executed, the system continues the course at the instruction bearing the label. → is an implicit label. '$GOTO→' or '$GOTO,→' causes the execution of the course to be transferred to the first instruction below in the same page which bears a → on column 1. If no → exists, then the branch will be to the last instruction assembled in the course - this is normally a system-supplied transfer to the next page.

GOTO,PAGE({<pagename>|*}) opcode: 05

e.g. (1) $GOTO,PAGE(T004) (2) →GOTO,PAGE(*)

Here <pagename> refers to another page in the same chapter; the execution of this instruction causes a transfer to be made to the beginning of that new page. $GOTO,PAGE(*) is a transfer to the next page in the chapter.

GOTO,CHAPTER({<chaptername>|*})[,PAGE({<pagename>|*})] opcode: 06 or 15

e.g. (1) $GOTO,CHAPTER(FIVE),PAGE(14)

(2) $GOTO,CHAPTER(T664)

(3) $GOTO,CHAPTER(*),PAGE(SIX)

This instruction causes a transfer to the named page of the named chapter. '*' refers to the next chapter or page as the case may be. When no page is mentioned (as in the 2nd example above) the first page of the chapter is intended. Assembles in one or two words depending on the instruction.

WAITCUE opcode: 07

This is a command to re-execute the last cue executed in the problem block - in effect giving the student another chance to answer. This is not the same as an explicit branch to a cue (see example on right) as in :- since this involves a particular cue, whereas WAITCUE "transfers" to the last 'active' cue.

```
$LI:CUE
$CUE.
$GOTO,L1.
```

GIVECUE opcode: 07 (as for WAITCUE)

This commands the system to continue from immediately after the last cue executed. What follows is usually a line or two of cueing text - hence the keyword GIVECUE.

TIME,<name> opcode: 13

e.g. (1) →TIME,XX2   (2) $TIME,LOC4.

This instructs the system to record on the student performance tape, the current time, using the specified name to identify the location in the course. This facilitates an investigation by researchers into various routes taken by the different students, and their 'arrival' time at the various sections.

{DELAY|DEL}(T=<delay>) opcode: 04

e.g. $DELAY(T=200)

This instructs the system not to continue executing the course until the specified delay (in seconds) has elapsed, unless the student interrupts during this time, with an input requiring processing, e.g. #GO.

WAIT opcode: 14

This instruction is equivalent to the three successive instructions:-

```
$TEXT.
  X̄RWX̄ .X̄X̄
$CUE.
$ENS(M=0)
```

This command causes a 'W̥' in red to be output on a new line, which acts as a signal to the student to press the carriage return key (or any input followed by the c/r) to indicate his readiness to continue. This instruction may be used to wait on the student until he is ready to continue. The WAIT may not be used in a problem environment.

{COMPLETED|FINISHED}   opcode: 17

This command indicates that the student's course terminates here. There may be more than one 'COMPLETED' in a course, since a course may terminate at several different points. The system also automatically assembles a COMPLETED instruction on encountering $END. as a precaution. The system action is to sign-off the student and to flag the end-of-course status.

{PROBLEM|PRB|BLOCK|BLK}[<name>[(R=<recording mode>)]]   opcode: 12

e.g. (1) $PROBLEM,Q002(R=8)   (2) $PRB.

This command allows the author to name a problem block and to specify the recording mode for the students' responses to it. In the CAI system, there is a location in the student area, called the blockname location - for saving a blockname and the recording mode. The effect of executing a PRB is to reset this location with a new name and R-value. When a student answer is received, the system examines this location. If it is zero in value, no recording of his response is made. Otherwise a recording of the response with this blockname and under the recording mode indicated is made.

When a new page is assembled, a flag F=PRB is set to zero. When a PRB command is encountered, this flag is set, and when a problem block is encountered, this flag is cleared; but before being cleared, the system ensures that it is set. If not, the system assembles a '$PRB.' - so that no recording for this block is made.

The PRB command must be used with care - being executable. Thus if the example (see opposite) shown, is executed, responses to the problem block could be recorded under the name Q1 and mode 2, instead of the presumably-intended name Q7 and mode 8.

```
$PRB,Q1(R=2)
$GOTO,L2.
$PRB,Q7(R=8)
$L2:TEXT.
 What is ...?
$CUE.
```

The command '$PRB.' is treated as having no name and 0 mode.

The recording mode, R, may assume any of the values 0,1,2..30,31 (decimal). R is in fact a 5-bit status flag; thus R=13 is '01110'. Each of the five bits has the significance indicated below:-

Bit Set — Recording Action

00001 Record the number of the answer in the answer sequence matched by the student response. Do not record anything if there is no match. Thus for the answer sequence: $ANS(M=2)*AA*PP*
$ENS(M=2)*BB*QQ*
the number 3 will be recorded for the student answer 'BB'.

00010 If student response matches, record the match number. If non-matching, record his whole response.

00100 Record whole answer if catch-all (i.e. $ENS(M=0)).

01000 Record student response unconditionally.

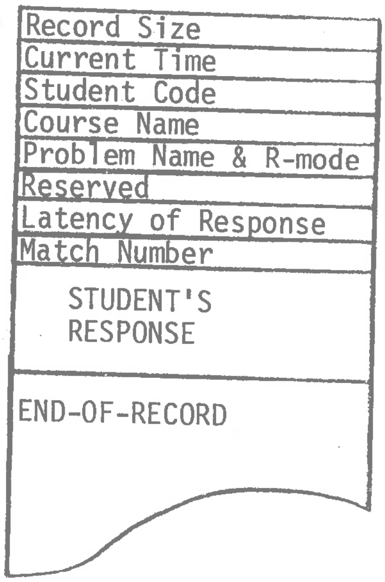
10000 Make a record of only the first response to this question.

e.g. (1) R=3 ($3=00011_2$) means same as R=2, i.e. record match number if match; else record the non-matching response.

(2) R=8 ($8=01000_2$) Record all responses to this problem.

Recording Details

At present, no standard programs exist, for processing the recorded data of student responses. Users will have to develop their own programs to extract the information they want. To assist in this, the recording format is described in some detail here. (See figure opposite)



Record Size
Current Time
Student Code
Course Name
Problem Name & R-mode
Reserved
Latency of Response
Match Number
STUDENT'S RESPONSE
END-OF-RECORD

The figure shows a record of recorded data.
Record size = number of CM words in the record.
Current time recorded in the form: '^*HH.MM.SS.ss'
where HH = hour, MM = minute, SS.ss = time in seconds to 2 decimal places.
Latency of Response - given in seconds.

The student's response is recorded in the T/W code (7-bit T/W code), packed to the left, 8 codes per word; first 4 bits of each word here = code count. Unused bits are zeroed.

Register Arithmetic

The author is provided with 15 registers A,B,C,...M,N,O, for holding working values. Each register is a 60-bit word and is used to hold a (floating point) number. At the beginning of a course, these registers are initialised to zero. Registers can be used to keep track of student performance. For instance, register A can be used to hold the total number of tries at a given set of questions and B to hold the corresponding number of successful ones.

Simple arithmetic on register values may be performed using register statements. The forms that a register statement may take are given by the BNF grammar below. Arithmetic is performed in floating point. Each register can hold a number n, where |n| lies between 10^{-294} and 10^{322}. Division by zero gives zero as the answer, e.g. 3/0 = 0.

The kinds of permitted expressions are those involving constants, registers and +, *, / and - only. These last four symbols, +, *, / and - are respectively the operators for addition, multiplication, division and subtraction. Rather complex arithmetic expressions are possible, but it is envisaged that the more usual ones would be sufficient to meet most author needs.

Examples:-

```
(1)  $A=B=3.64          (set registers A and B to the value 3.64)
(2)  $D=-3              (set register D to -3)
(3)  $A=12*B/(C-D)      (set A to the value of the expression 12*B/(C-D))
(4)  $A=B=G*(M=13.2/(4*K))-(E=F/G)
```

(4) is equivalent to the three separate statements:

```
$E=F/G
$M=13.2/(4*K)
$A=B=G*M - E
```

BNF Grammar for Register Statement

```
<digit>                ::=  0|1|2|...7|8|9
<register>             ::=  A|B|C|...M|N|O
<aop>                  ::=  +|-
<mop>                  ::=  *|/
<unsigned int>         ::=  <digit>|<unsigned int><digit>
<fraction>             ::=  <unsigned int>
<unsigned no>          ::=  <unsigned int>|<fraction>|<unsigned int>.|
                            <unsigned int><fraction>
<factor>               ::=  <register>|(<expression>)|(<register statement>)
<term>                 ::=  <factor>|<term><mop><factor>
<expression>           ::=  <term>|<expression><aop><term>|<aop><term>
<register statement>   ::=  <register>=<expression>|<register>=<register>
                             statement>
<rel op>               ::=  EQ|NE|GT|LT|LE|GE
<sign op>              ::=  PS|NG|NZ|ZR|NP|NN
<constant>             ::=  <unsigned no>|+<unsigned no>|-<unsigned no>
```

Conditional Branching

The conditional branch statement has the general form :-

IF(<relational expression>)Br

Here 'Br' stands for an unconditional branch statement. The relational expression has either TRUE or FALSE as value; if TRUE, the branch 'Br' is taken; otherwise the next instruction below is executed. The relational expression has one of the three forms below :-

(1) <register>.<rel op>.<register> e.g. A.LE.F
(2) <register>.<rel op>.<constant> e.g. B.NE.12.6
(3) <register>,<sign op> e.g. M,NG

The relational operators have the meanings:- EQ (equal), NE (not equal), GT (greater than), GE (greater than or equal to), LT (less than), LE (less than or equal to); PS (is positive), NG (negative), NZ (non-zero), ZR (zero), NP (non-positive) and NN (non-negative).

Examples of Conditional Branch

```
(1)  $IF(A.LE.F)GOTO,L6.
(2)  $IF(B.GT.-6.4)GOTO,PAGE(TWO)
(3)  $IF(D,NZ)GOTO,CHAPTER(XA7)
```

In example (1), if the value of A is 7 and if F is 13, then 'GOTO,L6' will be executed.

The Problem Block

The rather loose concept of a problem block is now discussed. A "cue" refers to a CUE, EXP or EXPLAIN instruction; an "answer" refers to ANSWER, ANS, ENDANS or ENS instruction while "endanswer" refers to ENS or ENDANS. Because the cue, answer, WAITCUE, GIVECUE (and even WAIT) instructions cannot be used as freely as the other instructions, they are said to be critical.

A cue sequence is a set of one or more cues, possibly interspersed with non-critical instructions. It has the form

```
     $CUE....)
      :      )
     $CUE....) a cue sequence
      :      )
     $CUE....)
```

```
e.g. $CUE,2(T=10)
       It begins with a 'C'
     $CUE(T=15)
     $TIME,ABB
       Try again.
     $EXP.
       The answer is 'crisis'.
```

The following is not a cue sequence :-

```
     $CUE.
     $ANS(M=4)*PQR*  (because it imbeds an answer)
     $CUE,2.
       Try again.
     $EXP.
```

An answer sequence is either an end answer or a set of answers (which are not end answers) terminated by an end answer. An answer sequence may be interspersed by other instructions, excluding cues and WAIT.

```
e.g. (1)  $ENS(M=2).ALPHA.BETA.

     (2)  $ANS(M=2)*COMMON*
           Not quite - try again.
          $WAITCUE.
          $ENS(M=2)*ORDINARY*
           Very good - correct.

     (3)  $ANS(M=2)*PORT*
          $ENS(M=5)*GORT*  (is not an answer sequence)
          $ENS(M=2)*WORT*
```

A cue sequence followed by an answer sequence constitutes a problem block. The problem block is a means for posing a question to the student and for accepting and analysing his responses against author answers. The cue sequence and the answer sequence have no independent standing. Every cue sequence must be followed by an answer sequence. This answer sequence is said to be associated with the cue sequence.

A problem block may not be split over two or more pages; it must be fully accommodated in one page. When an author-defined page is too long, the compiler will try to split into two or more pages without splitting a problem block.

It is normal for an answer sequence to be immediately preceded by an unconditional branch, to lead the student out of the problem block. If such a branch statement is absent, then a '$GOTO,→' is automatically inserted just above the first answer.

```
e.g. $CUE.....                      $CUE.....
       XXXXXX                         XXXXXX
     $ANS..... is equivalent to     $GOTO,→
                                    $ANS.....
```

Programming Examples

(1) Specification: Present student with a question 'Q...' and give him one chance to answer. There is only one current answer 'XXX'; if the answer given is correct, reinforce student, else if incorrect inform him so. No time limit for answering. 2 possible codes are :-

```
(a)  Q....
    $CUE
    $GOTO,L4
    $ENS(M=2)*XXX*
     Good. Correct.
    $L4:TEXT.
```

```
(b)  Q....
    CUE.
    $ENS(M=2)*XXX*
     Good. Correct.
    →
```

(2) Specification: Give student question; three attempts, with a time limit of 10 seconds each; no prompting material. Two possible codes are :-

```
(a) ...question...
    $CUE(T=10)
    $CUE(T=10)
    $CUE(T=10)
     The answer is ...
    $ANS(   ) ....
    $ENS(   ) ....
    →
```

```
(b) ...question...
    $CUE,3(T=10)
     The answer is ...
    $ANS(   ) ....
    $ENS(   ) ....
    →
```

(3) Specification: Let student try a question again and again; if incorrect answer, tell student to 'try again'. No cueing text. No time limit for attempts. Two ways to do this are :-

```
(a) ...question...
    $CUE.
     Try again.
    $WAITCUE.
    $ANS(   )....
    :
    $ENS(   )....
    →
```

```
(b) ...question...
    $CUE.
    $ANS(   ) ....
    :
    $ENS(M=0).
     No. Try again.
    $WAITCUE.
    →
```

(4) Specification: Let student try a question the number of times specified in register G - but at least once. No ~~limit~~ time limit for each attempt.

```
...question...
$A=0
$LOOP:CUE.
$A=A+1
$IF(A.LT.G)GOTO,LOOP
$ANS(   )....
 :
 :
$ENS(   )....
→
```

(5) Specification: 6 attempts, without time limit. If correct in first attempt, type out 'Very Good'; if correct on 2nd or 3rd attempt output 'Good', and if correct only after three or more incorrect attempts, type out 'Correct'. No cueing material. As usual, output 'Incorrect' for each wrong answer received.

```
...question...
$I=0
$CUE.
$I=I+1
$IF(I.GT.5)GOTO,4.
$ANS(   )....
 :
 :
$ENS(   )....
$IF(I.GT.1)GOTO,2.
 Very Good.
$GOTO,→
$2:IF(I.GT.3)GOTO,3.
 Good.
$GOTO,4.
$3:TEXT
 Correct.
→4.
```

Structure of a Course

An ALFIE course has the deck structure shown below. The card indentation is intended to emphasise the division of a course into chapters and pages. The vertical dots indicate intervening cards which are assumed not to be '$COURSE' or '$END'.

```
$COURSE,____
    $CHAPTER,____
        $PAGE,____
          :
        $PAGE,____
          :
        $PAGE,____
          :
    $CHAPTER,____
      :
    $CHAPTER,____
      :
    $CHAPTER,____
        $PAGE,____
          :
        $PAGE,____
          :
$END
```

```
SCOPE 3 job card.           (beginning card of job deck)
RFL,100.
REQUEST,XAOLR.  K55 READ ONLY
RFL,30000.
REWIND(XAOLR)
COPYBF(XAOLR,LKC)           (copy off program LKC)
COPYBF(XAOLR,ALF)           (copy off program ALF)
LKC.                        (pre-process course)
REQUEST,XAOBJ.  K40 WITH RING
ALF.                        (assemble course onto K40)
End-of-Record Card
  { your course deck }
End-of-File Card.
```

It is possible to compile more than one course in one run. This is done simply by placing the several courses to be compiled, one after another, and placing them in the job-deck (see above) where the single course deck is. The object codes of each course will be output on the course object tape, and treated as a file (i.e. terminated with an EOF mark) - with the last course having a double EOF terminator.

NOTE: *The version of ALF on K55 does not permit answer processing modes 10, 11, 12, 13, 14.*

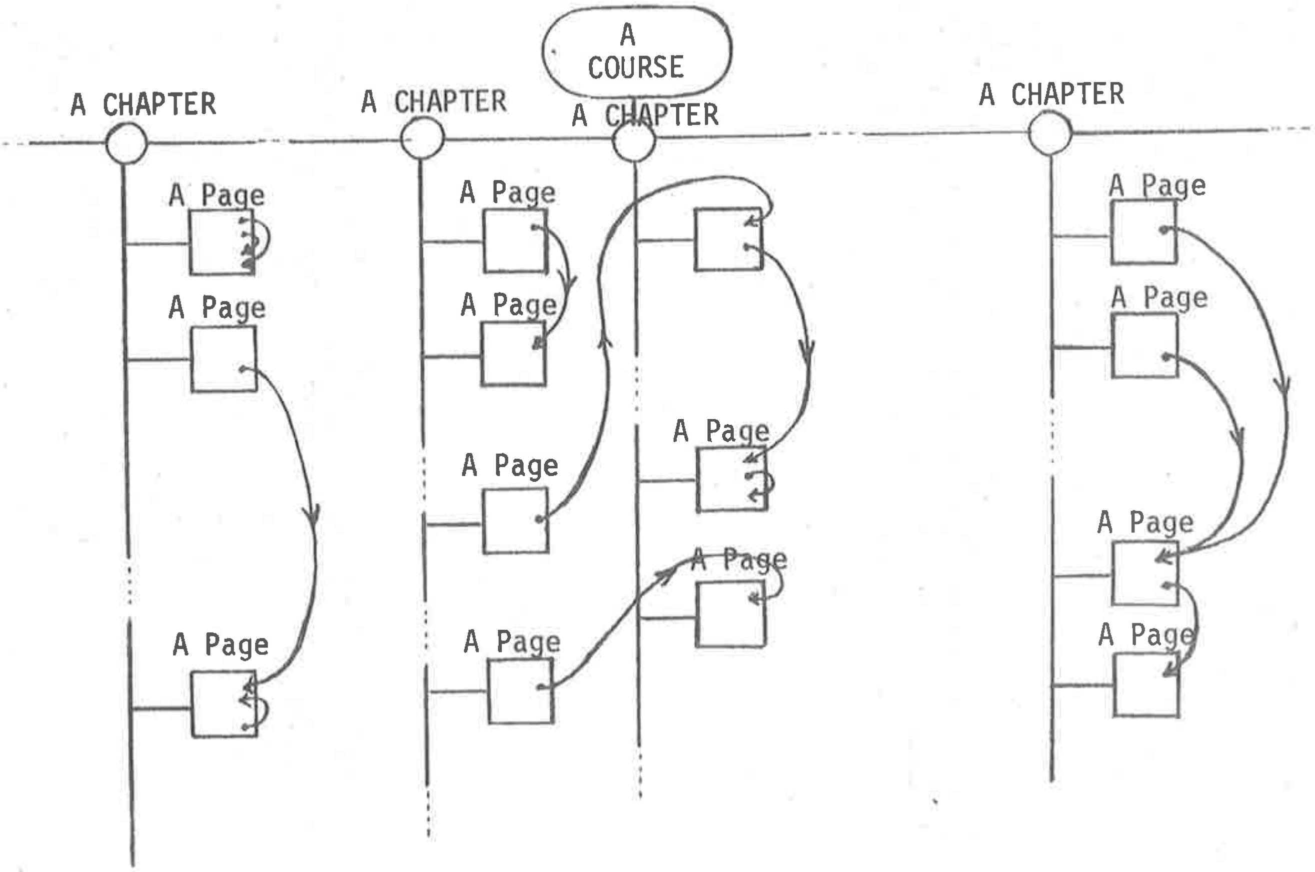


Figure A.4 Schematic of a Course

Job Deck Structure For Course Compilation

To compile a course onto a magnetic tape, the course deck must be organised with other cards to form a job - for running under the SCOPE operating system of the CDC 6400. The object code is output on the file XAOBJ, which is a tape file if a tape is assigned to it.

A typical job deck for compiling an ALFIE course has the structure shown below. The tape K55, contains the preprocessor LKC, and assembler program ALF in their binary form, as the first and second tape files. Tape K40 is assigned to XAOBJ.

APPENDIX B

THEORETICAL DEVELOPMENT

Résumé of Algebraic Geometry

This section reviews some of the main ideas and results from the theory of rings and algebraic geometry which are especially relevant to the derivation of our Thm 1, Thm 2 and Thm 3. The main sources of reference are [3,26,37,46].

Commutative Ring

Let R be a non-empty set of elements a,b,c,... in which the sum a+b and the product a·b of any two elements a and b of R are defined. Then R is a commutative ring with unity if the following axioms hold.

(1) $\forall\, a,b\in R \quad a+b\in R$ and $a\cdot b\in R$ [closure]

(2) $\forall\, a,a',b,b'\in R$ where $a=a'$ and $b=b'$, $a+b=a'+b'$ and $a.b=a'.b'$ [uniqueness]

(3) $\forall\, a,b\in R \quad a+b=b+a$ and $a\cdot b=b\cdot a$ [commutativity]

(4) $\forall\, a,b,c\in R \quad a+(b+c)=(a+b)+c$ and $a\cdot(b\cdot c)=(a\circ b)\cdot c$ [associativity]

(5) $\forall\, a,b,c\in R \quad a\cdot(b+c)=a.b+a\circ c$ [distributivity]

(6) $\exists \quad o\in R$ such that $\forall\, a\in R \quad a+o=a$ [zero]

(7) $\exists\, 1\in R,\ 1\neq o$ such that $\forall\, a\in R \quad a\circ 1=a$ [unit]

(8) $\forall\, a\in R\ \exists\, x\in R$ such that $a+x=o$ [additive inverse].

The following are examples of commutative rings with unity :-

(1) the set $\mathbb{Z}$ of all integers

(2) the field $\mathbb{R}$ of real numbers

(3) the complex field $\mathbb{C}$ and

(4) the ring $\mathbb{R}[\underset{\sim}{X}]$ of all polynomials in $\underset{\sim}{X}$ with real coefficients.

Henceforth all our rings will be assumed to be commutative rings with unity.

Defn. B.1 A non-empty subset I of a ring R is an ideal if and only if for every $a,b\in I$ and $r\in R$, $a-b\in I$ and $a\cdot r\in I$.

e.g. The set of all even integers is an ideal of the ring $\mathbb{Z}$.

Every ideal contains 0 (since 0 = a-a) and the additive inverse of every of its elements (since -a = 0-a). Every ring R contains at least two ideals, viz. the trivial ideals {0} and R. {0} which contains only the zero element is also called the zero ideal. An ideal that is non-trivial is said to be a proper ideal.

Defn. B·2 Let $S = \{s_1, s_2, \ldots s_k\}$ be a set of elements of R. Then the set $\{\sum_{i=1}^{k} a_i \cdot s_i \mid \text{all } a_i \in R,\ s_i \in S\}$ of all "linear combinations" of the elements of S is an ideal, called the ideal generated by S. S is called a basis of the ideal. The ideal is denoted by $(s_1, s_2, \ldots s_k)$. An ideal like this, with a finite basis is said to be finitely generated.
e.g. The ideal (2) of all even integers is generated by {2}.

Defn. B·3 A Noetherian ring is one in which every ideal is finitely generated. While not all rings are Noetherian, all our rings of interest are.

Notation: Let R be a ring. Then $R[\underset{\sim}{X}] \equiv R[x_1, x_2, \ldots x_n]$ denotes the ring of all polynomials in $x_1, x_2, \ldots x_n$ over the ring R - i.e. the set of all polynomials in $x_1, x_2, \ldots x_n$ with coefficients from R. Commonly used rings are the integer, the real and the complex rings $\mathbb{Z}$, $\mathbb{R}$ and $\mathbb{C}$.

Defn. B·4 'a' is a root of $F \in R[x]$ if $F \equiv (x-a) \cdot G$ for some $G \in R[x]$. A field K is algebraically closed if any non-constant $F \in K[x]$ has a root. Thus $\mathbb{C}$ is closed but not $\mathbb{R}$. A point $\underset{\sim}{X}_0 \in \mathbb{R}^n$ is a zero of the polynomial $F \in R[\underset{\sim}{X}]$ if $F(\underset{\sim}{X}_0) = 0$. If F is not a constant then the set of all zeroes of F, denoted H_F is called the hypersurface of F.

More generally a set $S = \{F_1, F_2, \ldots F_k\}$ of polynomials from $R[\underset{\sim}{X}]$ defines the set $M(S) \overset{\text{defn}}{=} \{\underset{\sim}{X} \in R^n \mid F_i(\underset{\sim}{X}) = 0 \text{ for } i=1,2,\ldots k\}$ of all the common zeroes of the polynomials $F_1, F_2, \ldots F_k$. We note that $M(S) = \bigcap_{i=1}^{k} M(F_i)$. We also write $M(S) \equiv M(\{F_1, F_2, \ldots F_k\})$ as $M(F_1, F_2, \ldots F_k)$.

Defn. B·5 A subset A of R^n is called an algebraic manifold (or simply a manifold) if $A = M(S)$ for some set S of polynomials in $R[\underset{\sim}{X}]$.

The set of all common zeroes of a set of polynomial is a manifold. A hypersurface in particular is a manifold.

Notation: If I is an ideal then $M(I) \overset{\text{defn}}{=\!=} \{\underset{\sim}{X}\varepsilon R^n | F(\underset{\sim}{X})=0 \; \forall F\varepsilon I\}$.

We now present some well-known results of algebraic geometry.

Res-1: If I is the ideal generated by S then $M(I)=M(S)$. Thus every manifold is equal to $M(I)$ for some ideal I.

Res-2: If $\{I_1,I_2,\ldots I_k\}$ is any set of ideals of $R[\underset{\sim}{X}]$ then $M(\bigcup_{i=1}^{k} I_i) = \bigcap_{i=1}^{k} M(I_i)$. So the intersection of any collection of manifolds is a manifold.

Res-3: If $I \subset J$ then $M(I) \supset M(J)$.

Res-4: $M(F\cdot G)=M(F) \cup M(G)$ for any polynomials F and G. $M(I)$ $M(J)$ $= M(\{F\cdot G | F\varepsilon I, G\varepsilon J\})$. Thus any finite union of manifolds is a manifold.

Res-5: $M(0)=R^n$ and $M(1)=\emptyset$. $M(x_1-a_1,x_2-a_2,\ldots x_n-a_n)=\{(a_1,a_2,\ldots a_n)\}$ for $a_i \varepsilon R$. So any finite subset of R^n is an algebraic manifold.

Defn. B·6: Let $A \subset R^n$. The set $I(A)$ of all polynomials in $R[\underset{\sim}{X}]$ which vanish on A form an ideal called the ideal of A.

Res-6: If $A \subset B$, then $I(A) \supset I(B)$.

Res-7: $I(\emptyset)=R[\underset{\sim}{X}]$ and $I(R^n)=(0)$ if R is an infinite field. $I(\{(a_1,a_2,\ldots a_n)\})=(x_1-a_1,x_2-a_2,\ldots x_n-a_n)$ for $a_i \varepsilon R$.

Res-8: $I(M(S))$ S for every set S of polynomials. $M(I(A))$ A for every set $A \subset R^n$.

Res-9: $M(I(M(S)))=M(S)$ for every set S of polynomials and $I(M(I(A))) = I(A)$ for every subset A of R^n. So if M' is a manifold then $M(I(M'))=M'$ and if I' is the ideal of a manifold, then $I(M(I'))=I'$.

Defn. B·7: A manifold M is irreducible if it is not possible to express it as the union of two proper submanifolds - i.e. $M=M_1 \cup M_2$ where $M_1, M_2 \neq M$, $M_1, M_2 \subset M$ and M_1 and M_2 are manifolds.

Defn. B.8: An ideal I is prime if for every $a \cdot b \varepsilon I$, either $a \varepsilon I$ or $b \varepsilon I$.

Res-10: A manifold M is reducible if and only if there exists two polynomials f and g such that neither one vanishes all over M, but such that f·g does.

Res-11: Every manifold is uniquely expressible as the union of irreducible manifolds $M_1, M_2, \ldots M_k$; i.e. $M=M_1 \cup M_2 \cup \ldots \cup M_k$ and $M_i \not\subset M_j$ for every $i \neq j$.

The M_i's are called the irreducible components of M and $M=\bigcup_{i=1}^{k} M_i$ is said to be the decomposition of M into its components. If M is itself irreducible then it is its only component.

Res-12: If $V \subset W$ and V,W are two manifolds of R^n then each irreducible component of V is contained in some irreducible component of W.

Defn. B·9: Let I be an ideal of R. The radical of I, rad(I) is defined as the set $\{a \varepsilon R | a^r \varepsilon I$ for some integer $r>0\}$.

It can be shown easily that rad(I) is itself an ideal. An ideal which coincides with its radical is called a radical ideal.

Res-13: Let V be an irreducible manifold of dimension r in $\mathbb{C}^n$. Let $F \varepsilon \mathbb{C}[\underset{\sim}{X}]$ be such that $V \cap H_F \neq \emptyset$ and $V \not\subset H_F$. Then all the components of $V \cap H_F$ have dimension r-1.

Res-14: Hilbert's Nullstellensatz: Let I' be an ideal of $K[\underset{\sim}{X}]$ where K is algebraically closed. Then Rad(I') = I(M(I')).

This is a central result in algebraic geometry and is also vital to our work. This zero-theorem says that if $G \varepsilon K[\underset{\sim}{X}]$ vanishes where $F_1, F_2, \ldots F_k \; \varepsilon \; K[\underset{\sim}{X}]$ vanish jointly, then $G^m = \sum_{i=1}^{k} f_i . F_i$ (Hilbert's Formula) for some positive interger m and $f_i \varepsilon \; K[\underset{\sim}{X}]$. Note that K must be a closed field.

This theorem gives the form of the set of all polynomials which vanish on the set of common zeroes of a set of polynomials.

Corollaries

(a) If I' is a radical ideal in $K[\underset{\sim}{X}]$ then I(M(I'))=I'. So there is a 1-1 correspondence between radical ideals and manifolds. Also if $(F_1, F_2, \ldots F_k)$ is a prime ideal, then m=1 in Hilbert's formula.

(b) If I' is prime then M(I') is irreducible. Thus there is a 1-1 correspondence between prime ideals and irreducible manifolds.

(c) Let $F \in R[\underset{\sim}{X}]$ where $F = F_1^{n_1}.F_2^{n_2}. \ldots .F_k^{n_k}$ is the decomposition of F into its irreducible factors $F_1, F_2, \ldots F_k$. Then M(F) = $\bigcup_{i=1}^{k} M(F_i)$ is <u>the</u> decomposition of M(F) and $I(M(F))=F_1 \cdot F_2 \cdot \ldots \cdot F_k)$. There exists thus a 1-1 correspondence between irreducible polynomials in $R[\underset{\sim}{X}]$ (up to a nonzero constant) and irreducible hypersurfaces in R^n.

Results Derived

In this section we derive Thm 1, Thm 2 and Thm 3, the last of which concerns the relationship between an arbitrary rational function and a set of reference rationals on whose common zeroes it vanishes. But first we establish two lemmas.

Lemma 1: Let $A, B_1, B_2, \ldots B_k$ be distinct irreducible manifolds and let C be any manifold such that $C \supset A-(\bigcup_{i=1}^{k} B_i)$. Then $C \supset A$.

Proof: If $A-(\bigcup_{i=1}^{k} B_i) = A$ then the lemma is trivially true.
Therefore assume otherwise, i.e. $A \cap (\,_{i=1}^{k} B_i) \neq \emptyset$.
Let $D = A \cup (\bigcup_{i=1}^{k} B_i)$. Then by Res-11, the unique decomposition theorem, $A \cup B_1 \cup B_2 \cup \ldots \cup B_k$ is the unique decomposition of D.
Suppose the lemma is false, i.e. $C \not\supset A$.
But $C \supset A-\bigcup_{i=1}^{k} B_i$ (hypothesis)
Therefore $A \cap C \supset A \cap [A-\bigcup_{i=1}^{k} B_i] = A-\bigcup_{i=1}^{k} B_i$.
Therefore $(A \cap C) \cup [\bigcup_{i=1}^{k} B_i] \supset (A-\bigcup_{i=1}^{k} B_i) \cup [\bigcup_{i=1}^{k} B_i]$
ie. $(A \cap C) \cup [\bigcup_{i=1}^{k} B_i] \supset A \cup [\bigcup_{i=1}^{k} B_i]$

But $(A\cap C)\cup[\overset{k}{\underset{i=1}{\cup}}B_i]\subset A\cup[\overset{k}{\underset{i=1}{\cup}}B_i]$ since $A\cap C\subset A$.

Therefore $(A\cap C)\cup[\overset{k}{\underset{i=1}{\cup}}B_i] = A\cup[\overset{k}{\underset{i=1}{\cup}}B_i] = D$ since $A\subset B$ and $A\supset B \rightarrow A=B$.

Now by assumption $A\cap C \neq A$ [cf. $C\not\supset A$] and whether $A\cap C$ is irreducible or not, we have displayed two <u>distinct</u> decompositions of D - a contradiction of Res-11. Therefore $C\supset A$. QED

<u>Lemma 2</u>: Let A and B be any two manifolds with no irreducible components in common. Then for any manifold for which $C\supset A-B$, $C\supset A$.

<u>Proof</u>: Let $A=\overset{k}{\underset{i=1}{\cup}}A_i$ and $B=\overset{l}{\underset{j=1}{\cup}}B_j$ be the unique decompositions of A and B. Then by the hypothesis $A_i\neq B_j$ for all $1<i<k$ and $1<j<l$.

$C\supset \underset{i}{\cup}A_i - \underset{j}{\cup}B_j$ (hypothesis)

$\therefore$ $C\supset A_i - \underset{j}{\cup}B_j$ (for $i=1,2,\ldots k$)

$\therefore$ $C\supset A_i$ $(i=1,2,\ldots k)$ by Lemma 1.

$\therefore$ $C\supset \overset{k}{\underset{i=1}{\cup}}A_i$ QED.

<u>Thm 1</u>: Let $r(\underset{\sim}{X})\equiv p(\underset{\sim}{X})/q(\underset{\sim}{X})$ where p and q are relatively prime polynomials. Then if any polynomial p' vanishes on M(r)* it also vanishes on M(p).

<u>Proof</u>: Let the unique factorisation of p and q be respectively $\overset{n_1}{\underset{i=1}{\Pi}}p_i^{\sigma_i}$ and $\overset{n_2}{\underset{j=1}{\Pi}}q_j^{\delta_j}$. Then $M(p) = \overset{n_1}{\underset{i=1}{\cup}}M(p_i)$ and $M(q) = \overset{n_2}{\underset{j=1}{\cup}}M(q_j)$ and for all i,j. $M(p_i)\neq M(q_j)$. Furthermore $M(p')\supset M(r)=M(p)-M(q)$. Therefore by lemma 2 $M(p')\supset M(p)$; i.e. p' vanishes on M(p). QED.

<u>Thm 2</u>: Let $r_1,r_2,\ldots r_k$ be rational expressions in which $r_i = p_i/q_i$ $(i=1,2,\ldots k)$, p_i and q_i being relatively prime polynomials. The p_i and q_j $(1<i,j<k)$ have no common factors. Then any polynomial p which vanishes on $\overset{k}{\underset{i=1}{\cap}}M(r_i)$ also vanishes on $\overset{k}{\underset{i=1}{\cap}}M(p_i)$.

* *M(r) is the set of zeroes of r in* $\mathbb{C}^n$, *i.e.* $\{\underset{\sim}{X}\in\mathbb{C}^n|r(X)=0\}$.

<u>Proof</u>: $\bigcap_i M(r_i) = \bigcap_i [M(p_i)-M(q_i)]$

$= \bigcap_i [M(p_i) \cap M'(q_i)]$

$= \bigcap_i M(p_i) \cap [\bigcap_i M'(q_i)]$

$= \bigcap_i M(p_i) \cap [\bigcap_i M(q_i)]'$

$= \bigcap_{i=1}^{k} M(p_i) - \bigcup_{i=1}^{k} M(q_i)$.......(*)

Let the distinct prime factors of $p_1,p_2,\ldots p_k$ and of $q_1,q_2,\ldots q_k$ be $p'_1,p'_2,\ldots p'_m$ and $q'_1,q'_2\ldots q'_n$ respectively. By hypothesis, these two groups of factors are distinct and therefore $M(p'_i) \neq M(q'_j)$ for all i and j. Therefore $M(q'_1),M(q'_2),\ldots M(q'_n)$ and the components of $\bigcap_{i=1}^{m} M(p'_i)$ are distinct. This is because if m=1, then it is already true while if m>1, the components of $\bigcap_{i=1}^{m} M(p'_i)$ are of lower dimension (cf Res-13) than $M(q'_j)$ (j=1,2,...n) and they are therefore distinct. But $\bigcap_{i=1}^{k} M(P_i) = \bigcap_{i=1}^{m} M(p'_i)$ and $\bigcup_{j=1}^{k} M(q_j) = \bigcup_{j=1}^{n} M(q'_j)$. Noting (*) and applying lemma 2, $\bigcap_{i=1}^{k} M(r_i) \supset \bigcap_{i=1}^{m} M(p'_i) = \bigcap_{i=1}^{k} M(p_i)$. But by hypothesis $M(p) \supset \bigcap_{i=1}^{k} M(r_i)$. Therefore $M(p) \supset \bigcap_{i=1}^{k} M(p_i)$, i.e. p vanishes on $\bigcap_{i=1}^{k} M(p_i)$. QED.

<u>Thm 3</u>: Let $r_1,r_2,\ldots r_k$ be rational expressions where $r_i = p_i/q_i$ (i=1,2,...k), the p_i and q_i being relatively prime polynomials. The p_i and q_j ($1 \leq i,j \leq k$) have no common factors. If r is any rational expression which vanishes on $\bigcap_{i=1}^{k} M(r_i)$ (pwd), then $r^m = \sum_{i=1}^{k} s_i \cdot r_i$ where m is a positive integer and each s_i is a rational expression which is not ill-defined <u>all over</u> $M(r_i)$. (All polynomials are assumed to be from $C[\underset{\sim}{X}]$).

<u>Proof</u>: Let r = p/q. If r vanishes on $M = \bigcap_{i=1}^{k} M(r_i)$ then it is p which vanishes on M. By Thm 2, p vanishes on $\bigcap_{i=1}^{k} M(p_i)$. By Hilbert's nullstellensatz, $p^m = \sum_{i=1}^{k} h_i \cdot p_i$ where m is a positive integer (m=1 if the ideal $(p_1,p_2,\ldots p_k)$ is prime) and h_i (i=1,2,...k) are polynomials. Therefore $p^m/q^m = \sum_{i=1}^{k} (h_i \cdot p_i)/q^m = \sum_{i=1}^{k} ((h_i \cdot q_i)/q^m).(p_i/q_i)$. This is of form $r^m = \sum_{i=1}^{k} s_i \cdot r_i$ where $s_i = (h_i . q_i)/q^m$ and is therefore not ill-defined except where q vanishes. But $M(q) \not\supset M(r_i)$ for else it is not possible for r to vanish on M. Therefore as required s_i is not ill-defined all over $M(r_i)$. QED.

APPENDIX C

STEP	MACHINE	OPINION					HUMAN	OPINION													AGREEMENT	COUNT				
	1	2	3	4	5	6	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1	2	3	4	5	6
1	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
2	0	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	2	2	12	12	12	12
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	12	12	12	12	12	12
5	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
7	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
8	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	1	2	12	2	12	12	12
9	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
10	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
11	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
12	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	0	12	12	12	12	12	2
13	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	13	13	13	13
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
15	0	0	1	0	0	0	1	1	0	0	0	1	1	0	0	0	0	1	0	1	8	8	6	8	8	8
16	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
17	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
18	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
19	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	13	13	13	13	13	1
20	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
21	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	1	0	0	12	12	12	12	12	2
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
23	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
24	1	1	1	1	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0	1	7	7	7	7	7	7
25	0	0	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	14	14	0
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
27	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
28	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
29	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
30	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
31	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
32	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
33	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
34	1	1	1	1	0	0	0	0	0	0	1	1	0	1	1	0	1	0	0	1	6	6	6	6	8	8
35	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	0	1	1	1	1	12	12	12	12	12	2
36	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
37	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
38	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
39	2	2	2	0	0	1	1	0	0	0	0	1	0	0	0	0	0	1	0	0	11	11	11	11	11	3
40	0	0	0	1	1	1	1	0	0	0	1	1	1	1	1	0	1	1	0	1	5	5	5	9	9	9
41	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
43	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
44	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
45	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	13	13	13	13	13	1
46	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	0	1	1	1	0	10	10	10	10	10	10
47	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	13	13	13	13	13	1
48	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	13	13	13	13	13	1
49	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
50	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
51	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
52	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	13	13	13	13	13	1
53	1	1	1	0	1	0	0	1	0	1	1	1	1	1	1	0	1	1	1	1	11	11	11	3	11	3
54	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	13	13	13	13	13	13
55	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
56	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
57	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
58	1	1	1	0	1	0	1	0	0	1	1	1	1	1	0	0	1	1	1	0	9	9	9	5	9	5
59	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	0	12	12	12	12	12	12
60	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	13	13	13	13	13	13
61	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
62	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
63	0	1	0	1	1	1	1	0	0	1	1	1	0	1	1	0	1	1	0	1	5	9	5	9	9	9
64	0	1	0	1	1	0	1	1	0	1	1	1	1	1	1	0	0	1	1	1	3	11	3	11	11	3
65	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
66	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
67	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
68	0	0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	1	0	0	1	8	8	8	8	6	8

Table C.1 Opinion Data

STEP	MACHINE OPINION						HUMAN OPINION														AGREEMENT COUNT					
	1	2	3	4	5	6	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1	2	3	4	5	6
69▸	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
70▸	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
71▸	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
72▸	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
73▸	2	2	2	0	1	0	0	0	0	1	1	1	0	1	0	0	1	1	1	0	7	7	7	7	7	7
74▸	2	2	2	0	1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	12	12
75▸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
76▸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
77▸	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
78▸	2	2	2	0	0	1	0	0	0	0	0	1	0	0	1	0	1	1	1	1	8	8	8	8	8	6
79▸	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	14	14	14	0
80▸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
81▸	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
82▸	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	14	14	14	0
83▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
84▸	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	0	14	14	0
85▸	1	1	1	1	1	0	1	1	0	1	1	1	1	1	0	0	1	1	1	0	10	10	10	10	10	4
86▸	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
87▸	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
88▸	0	0	1	1	1	0	1	0	0	0	1	1	1	1	1	1	1	1	1	0	4	4	10	10	10	4
89▸	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	14	14	14	0
90▸	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	0	14	14	14
91▸	0	0	1	1	1	0	0	0	1	0	0	1	1	1	1	0	1	1	1	1	5	5	9	9	9	5
92▸	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
93▸	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
94▸	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	0	1	1	1	0	8	8	8	8	8	8
95▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
96▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
97▸	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	13	13	13	13	13	1
98▸	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	11	11	11	11	11	3
99▸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
100▸	2	2	2	0	0	1	0	0	0	1	1	0	1	0	0	0	1	0	0	0	10	10	10	10	10	4
101▸	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	13	13	13
102▸	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
103▸	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
104▸	0	0	1	1	1	1	0	0	0	1	1	1	0	1	1	0	1	1	1	1	5	5	9	9	9	9
105▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
106▸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
107▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
108▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
109▸	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
110▸	2	2	2	1	1	1	0	1	0	0	1	1	1	1	1	0	0	1	1	0	6	6	6	8	8	8
111▸	2	2	2	1	1	0	1	1	1	0	1	1	1	1	1	0	1	1	1	1	2	2	2	12	12	2
112▸	2	2	2	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	14	14	0
113▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
114▸	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
115▸	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
116▸	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	14	14	14
117▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
118▸	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
119▸	2	2	2	1	1	0	0	1	0	1	0	1	1	0	1	1	1	1	1	0	5	5	5	9	9	5
120▸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
121▸	2	2	2	0	0	1	0	1	0	0	0	0	0	0	1	0	0	0	1	0	11	11	11	11	11	3
122▸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
123▸	1	1	1	0	0	1	0	0	0	1	1	0	1	1	0	0	0	0	0	0	4	4	4	10	10	4
124▸	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
125▸	1	1	1	1	1	1	1	0	0	0	1	1	0	1	1	0	1	1	1	0	8	8	8	8	8	8
126▸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
127▸	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
128▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
129▸	2	2	2	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	13	13	13	13	13	13
130▸	2	2	2	1	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1	0	3	3	3	11	11	11
131▸	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	14	0	14	14	0
132▸	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
133▸	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
134▸	0	1	0	1	1	1	0	1	1	1	1	1	1	1	1	0	1	1	1	1	2	12	2	12	12	12
135▸	1	1	1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	12	12	12	12	12	12
136▸	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
137▸	1	1	1	1	0	0	0	0	0	0	1	1	0	1	0	1	1	0	0	0	5	5	5	5	9	9

Table C.1 Continued

TEP	MACHINE OPINION						HUMAN OPINION														AGREEMENT COUNT					
	1	2	3	4	5	6	1	2	3	4	5	6	7	8	9	10	11	12	13	14	1	2	3	4	5	6
38	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
39	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
40	1	1	1	0	0	1	0	0	0	1	0	1	0	1	1	0	1	1	0	0	6	6	6	8	8	6
41	0	0	0	0	1	0	0	1	0	1	1	1	1	1	1	0	1	1	1	0	4	4	4	4	10	4
42	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
43	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
44	2	2	2	0	0	1	0	0	0	0	1	1	0	0	1	0	0	0	0	0	11	11	11	11	11	3
45	0	1	0	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	14	0	14	14	0
46	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
47	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	13	1	13	13	13	13
48	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
49	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
50	2	2	2	0	1	1	1	1	0	0	0	1	1	1	1	0	1	1	0	0	6	6	6	6	8	8
51	0	1	0	1	1	1	1	1	0	0	1	1	1	1	1	1	1	1	1	1	2	12	2	12	12	12
52	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
53	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
54	0	0	0	1	0	0	0	1	0	0	1	1	0	1	0	0	1	0	0	0	9	9	9	5	9	9
55	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	13	13	13	13
56	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
57	0	0	1	0	0	0	1	1	0	0	1	0	1	0	0	0	0	0	0	0	10	10	4	10	10	10
58	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	0	14	14	14
59	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
60	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	0	1	1	1	1	11	11	11	11	11	11
61	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
62	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
63	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
64	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
65	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
66	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
67	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	13	13	13
68	2	2	2	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	14	14	0
69	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	14	14	14
70	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
71	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
72	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
73	2	2	2	1	1	0	0	1	0	1	1	1	1	1	1	0	1	1	1	0	4	4	4	10	10	4
74	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	13	13	13
75	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
76	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
77	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
178	2	2	2	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	14	14	14
179	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
180	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
181	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
182	1	1	1	0	0	1	1	1	0	0	1	1	0	1	0	0	0	0	0	0	5	5	5	9	9	5
183	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	10	10	10	10	10	10
184	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
185	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	1	10	10	10	10	10	4
186	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
187	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
188	1	1	1	0	0	1	1	0	0	0	1	1	0	0	1	0	1	1	1	0	7	7	7	7	7	7
189	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
190	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
191	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
192	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
193	2	2	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
194	2	2	2	1	1	1	1	1	0	1	1	1	1	1	1	0	1	1	1	1	2	2	2	12	12	12
195	2	2	2	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	14	14	0
196	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
197	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	14
198	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
199	2	2	2	0	0	1	0	1	0	1	0	1	0	0	1	0	0	1	1	0	8	8	8	8	8	6
200	1	1	1	1	1	0	1	1	1	1	0	1	0	1	1	1	1	1	1	1	12	12	12	12	12	2
201	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	14	14	14	14	14	0
202	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0
203	0	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	12	12	2	2	12	12
204	1	1	1	1	1	1	1	1	1	0	1	1	0	1	1	1	1	0	1	1	11	11	11	11	11	11
205	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	14
206	1	1	1	0	0	1	1	1	0	0	1	0	0	0	0	0	0	0	0	0	3	3	3	11	11	3
207	2	2	2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	14	14	14	14	14	0

Table C.1 Continued

SPANNING C-SET	NX	SUBSTITUTION SET	CONFIG	IMMEDIATE C-SUBSETS	OPERATING IDENTITIES
-	0	1 2 3 4 5 6	------	-	
5 4 3 2 1	22	1 2 10 7 8 11	111 111	46 5 4 3 2	CSC=1/SIN, SEC=1/COS, TAN=SIN/COS, COT=COS/SIN
4 2 1	18	1 2 10 7 8 6	111 110	11 9 8	CSC=1/SIN, SEC=1/COS, TAN=SIN/COS
5 4 3 1	20	1 2 10 7 5 11	111 101	15 11 10 8	CSC=1/SIN, COT=1/TAN, TAN=SIN/COS, COT=COS/SIN
5 4 3 2	21	1 2 10 4 8 11	111 011	20 13 10 9	SEC=1/COS, COT=1/TAN, TAN=SIN/COS, COT=COS/SIN
5 2 1	19	1 2 3 7 8 11	110 111	13 12 11	SEC=1/COS, CSC=1/SIN, COT=COS/SIN
5 4 3 2 1	22	1 2 10 7 8 11	101 111	15 5 4 2	CSC=1/SIN, COT=1/TAN, TAN=SIN*SEC, COT=1/(SIN*SEC)
5 4 3 2 1	22	1 2 10 7 8 11	011 111	20 3 2	SEC=1/COS, COT=1/TAN, COT=COS*CSC, TAN=1/(COS*CSC)
4 1	11	1 2 10 7 5 6	111 100	24 23	CSC=1/SIN, TAN=SIN/COS
4 2	14	1 2 10 4 8 6	111 010	25 23	SEC=1/COS, TAN=SIN/COS
5 4 3	16	1 2 10 4 5 11	111 001	29 26 23	COT=1/TAN, TAN=SIN/COS, COT=COS/SIN
2 1	9	1 2 3 7 8 6	110 110	25 24	CSC=1/SIN, SEC=1/COS
5 1	12	1 2 3 7 5 11	110 101	26 24	CSC=1/SIN, COT=COS/SIN
5 2	15	1 2 3 4 8 11	110 011	26 25	SEC=1/COS, COT=COS/SIN
4 2 1	18	1 2 10 7 8 6	101 110	24 9	CSC=1/SIN, TAN=SIN*SEC, TAN=SEC/CSC
3 1	10	1 2 3 7 5 9	101 101	29 24	CSC=1/SIN, COT=1/TAN
5 4 3 2	21	1 2 10 4 8 11	101 011	31 29 9	COT=1/TAN, TAN=SIN*SEC
5 2 1	19	1 2 3 7 8 11	100 111	13 24	CSC=1/SIN, COT=CSC/SEC
4 2 1	18	1 2 10 7 8 6	011 110	25 8	SEC=1/COS, TAN=SEC/CSC, TAN=1/(COS*CSC)
5 4 3 1	20	1 2 10 7 5 11	011 101	29 12 8	COT=1/TAN, TAN=1/(CSC*COS)
3 2	13	1 2 3 4 8 9	011 011	29 25	SEC=1/COS, COT=1/TAN
5 4 3 2 1	22	1 2 10 7 8 11	010 111	25 12	SEC=1/COS, COT=CSC/SEC
5 4 3 2 1	22	1 2 10 7 8 11	001 111	29 5 2	COT=1/TAN, TAN=SEC/CSC
4	4	1 2 10 4 5 6	111 000	-	TAN=SIN/COS
1	1	1 2 3 7 5 6	110 100	-	CSC=1/SIN
2	2	1 2 3 4 8 6	110 010	-	SEC=1/COS
5	5	1 2 3 4 5 11	110 001	-	COT=COS/SIN
1	1	1 2 3 7 5 6	101 100	-	CSC=1/SIN
4 2	14	1 2 10 4 8 6	101 010	-	TAN=SIN*SEC
3	3	1 2 3 4 5 9	101 001	-	COT=1/TAN
1	1	1 2 3 7 5 6	100 110	-	CSC=1/SIN
1	1	1 2 3 7 5 6	100 101	-	CSC=1/SIN
5 2	15	1 2 3 4 8 11	100 011	-	COT=1/(SIN*SEC)
4 1	11	1 2 10 7 5 6	011 100	-	TAN=1/(COS*CSC)
2	2	1 2 3 4 8 6	011 010	-	SEC=1/COS
3	3	1 2 3 4 5 9	011 001	-	COT=1/TAN
2	2	1 2 3 4 8 6	010 110	-	SEC=1/COS
5 1	12	1 2 3 7 5 11	010 101	-	COT=COS*CSC
2	2	1 2 3 4 8 6	010 011	-	SEC=1/COS
4 2 1	18	1 2 10 7 8 6	001 110	-	TAN=SEC/CSC
3	3	1 2 3 4 5 9	001 101	-	COT=1/TAN
3	3	1 2 3 4 5 9	001 011	-	COT=1/TAN
5 2 1	19	1 2 3 7 8 11	000 111	-	COT=CSC/SEC
1	1	1 2 3 7 5 6	100 100	-	CSC=1/SIN
2	2	1 2 3 4 8 6	010 010	-	SEC=1/COS
3	3	1 2 3 4 5 9	001 001	-	COT=1/TAN
3 2 1	17	1 2 3 7 8 9	------	20 15 11	CSC=1/SIN, SEC=1/COS, COT=1/TAN
8 7 6	26	12 13 24 21 25 23	111 111	50 49 48	SIN↑2+COS↑2=1, SEC↑2=TAN↑2+1, CSC↑2=COT↑2+1
7 6	23	12 13 24 4 25 6	111 010	52 51	SIN↑2+COS↑2=1, SEC↑2=TAN↑2+1
8 6	24	12 13 3 21 5 23	110 101	53 51	SIN↑2+COS↑2=1, CSC↑2=COT↑2+1
8 7	25	1 2 24 21 25 23	001 111	53 52	SEC↑2=TAN↑2+1, CSC↑2=COT↑2+1
6	6	12 13 3 4 5 6	110 000	-	SIN↑2+COS↑2=1
7	7	1 2 24 4 25 6	001 010	-	SEC↑2=TAN↑2+1
8	8	1 2 3 21 5 23	000 101	-	CSC↑2=COT↑2+1
7 6 4 2 1	78	12 13 14 15 16 6	111 110	141 128 126 61 59 58	CSC=1/SIN, SEC=1/COS, TAN=SIN/COS, SIN↑2+COS↑2=1
8 6 5 2 1	83	12 13 3 15 16 17	110 111	138 123 119 63 62 61	CSC=1/SIN, SEC=1/COS, COT=COS/SIN, SIN↑2+COS↑2=1
8 7 3 1	89	12 2 14 15 16 17	101 111	135 133 127 65	CSC=1/SIN, COT=1/TAN, SEC↑2=TAN↑2+1, CSC↑2=COT↑2+1
8 7 3 2	91	1 13 14 15 16 17	011 111	134 133 121 68	SEC=1/COS, COT=1/TAN, SEC↑2=TAN↑2+1, CSC↑2=COT↑2+1
6 4 1	55	12 13 14 15 5 6	111 100	71 70	CSC=1/SIN, TAN=SIN/COS, SIN↑2+COS↑2=1
7 6 4 2	82	12 13 14 4 16 6	111 010	146 79 72 70	SEC=1/COS, TAN=SIN/COS, SIN↑2+COS↑2=1
6 5 4 3	70	12 13 14 4 5 17	111 001	148 73 70	TAN=SIN/COS, COT=COS/SIN, SIN↑2+COS↑2=1
6 2 1	53	12 13 3 15 16 6	110 110	72 71	CSC=1/SIN, SEC=1/COS, SIN↑2+COS↑2=1
8 6 5 1	85	12 13 3 15 5 17	110 101	142 78 73 72	CSC=1/SIN, COT=COS/SIN, SIN↑2+COS↑2=1
6 5 2	57	12 13 3 4 16 17	110 011	73 72	SEC=1/COS, COT=COS/SIN, SIN↑2+COS↑2=1
7 6 4 2 1	78	12 13 14 15 16 6	101 110	141 128 126 61 59 58	CSC=1/SIN, TAN=SIN*SEC, SEC↑2=TAN↑2+1
8 3 1	65	12 2 14 15 5 17	101 101	85 78	CSC=1/SIN, COT=1/TAN, CSC↑2=COT↑2+1
8 6 5 2 1	83	12 13 3 15 16 17	100 111	138 119 123 63 61 62	COT=CSC/SEC, CSC=1/SIN, CSC↑2=COT↑2+1
7 6 4 2 1	78	12 13 14 15 16 6	011 110	141 126 128 58 61 59	SEC=1/COS, TAN=SEC/CSC, SEC↑2=TAN↑2+1
7 3 2	62	1 13 14 4 16 17	011 011	86 79	SEC=1/COS, COT=1/TAN, SEC↑2=TAN↑2+1
8 6 5 2 1	83	12 13 3 15 16 17	010 111	138 123 119 63 62 61	SEC=1/COS, COT=CSC/SEC, CSC↑2=COT↑2+1
6 4	30	12 13 14 4 5 6	111 000	-	TAN=SIN/COS, SIN↑2+COS↑2=1
6 1	27	12 13 3 15 5 6	110 100	-	CSC=1/SIN, SIN↑2+COS↑2=1
6 2	28	12 13 3 4 16 6	110 010	-	SEC=1/COS, SIN↑2+COS↑2=1
6 5	31	12 13 3 4 5 17	110 001	-	COT=COS/SIN, SIN↑2+COS↑2=1
8 3 1	65	12 2 14 15 5 1	101 100	-	CSC=1/SIN, CSC↑2=1/TAN↑2+1
7 6 4 2	82	12 13 14 4 16 6	101 010	-	TAN=SIN*SEC, SEC↑2=TAN↑2+1
8 3 1	65	12 2 14 15 5 17	101 001	-	COT=1/TAN, 1/SIN↑2=COT↑2+1
6 2 1	53	12 13 3 15 16 6	100 110	-	CSC=1/SIN, SIN↑2+1/SEC↑2=1
8 1	37	12 2 3 15 5 17	100 101	-	CSC=1/SIN, CSC↑2=COT↑2+1
7 2	33	1 13 14 4 16 6	011 010	-	SEC=1/COS, SEC↑2=TAN↑2+1
7 3 2	62	1 13 14 4 16 17	011 001	-	COT=1/TAN, 1/COS↑2=TAN↑2+1
6 2 1	53	12 13 3 15 16 6	010 110	-	SEC=1/COS, COS↑2+1/CSC↑2=1
8 6 5 1	85	12 13 3 15 5 17	010 101	73 71	COT=COS*CSC, COS↑2+1/CSC↑2=1, CSC↑2=COT↑2+1
7 3 2	62	1 13 14 4 16 17	010 011	-	SEC=1/COS, SEC↑2=1/COT↑2+1
7 6 4 2 1	78	12 13 14 15 16 6	001 110	-	TAN=SEC/CSC, SEC↑2=TAN↑2+1
8 3	39	1 2 14 15 5 17	001 101	-	COT=1/TAN, CSC↑2=COT↑2+1
7 3	34	1 2 14 4 16 17	001 011	-	COT=1/TAN, SEC↑2=TAN↑2+1
8 6 5 2 1	83	12 13 3 15 16 17	000 111	-	COT=CSC/SEC, CSC↑2=COT↑2+1
8 3 1	65	12 2 14 15 5 17	101 000	-	1/SIN↑2=1/TAN↑2+1

Table C.2 C-set Search Table

```
  NO SPANNING C-SET  NX    SUBSTITUTION SET  CONFIG          IMMEDIATE C-SUBSETS       OPERATING IDENTITIES

           7 6 4 2   82   12 13 14  4 16  6                                    72
  89           6 2   28   12 13  3  4 16  6  100 010                            -   1/SEC↑2+SIN↑2=1
  90           8 1   37   12  2  3 15  5 17  100 001                            -   1/SIN↑2=COT↑2+1
  91           7 2   33    1 13 14  4 16  6  011 000                            -   1/COS↑2=TAN↑2+1
  92           6 1   27   12 13  3 15  5  6  010 100                            -   COS↑2+1/CSC↑2=1
  93         7 3 2   62    1 13 14  4 16 17  010 001                            -   1/COS↑2=1/COT↑2+1
           8 6 5 1   85   12 13  3 15  5 17                                    73
  94           8 3   39    1  2 14 15  5 17  001 100                            -   CSC↑2=1/TAN↑2+1
  95         6 2 1   53   12 13  3 15 16  6  000 110                            -   1/CSC↑2+1/SEC↑2=1
             8 7 3   50    1  2 14 15 16 17                                     -
  96         6 5 2   57   12 13  3  4 16 17  000 011                            -   SEC↑2=1/COT↑2+1
               7 3   34    1  2 14  4 16 17                                     -
  97     8 7 5 4 3   96   28  2 14 15 16 17  XXXXXX           117 118 131 132 133
  98     8 5 4 3 2   95   28  2 14 15  8 17  XXXXXX               121 120 119 117
  99     7 5 4 3 1   94   28  2 14 29 16 17  XXXXXX               127 126 125 118
 100       8 7 6 3   93   18 19 14 15 16 17  XXXXXX                   140 137 133
 101       8 7 5 2   92   30 13 14 21 16 23  XXXXXX               134 131 124 119
 102       8 7 3 2   91    1 13 14 15 16 17  XXXXXX               134 133 121  80
 103       8 7 2 1   88   18 13 14 21 16 23  XXXXXX               135 134 128 123
 104     8 7 6 4 2   87   12 13 14 21 16 23  XXXXXX           139 136 134 132  58
 105       8 6 3 2   86   12 13 20 21 16 23  XXXXXX               138 137 129 121
 106     8 7 6 5 1   80   12 13 24 15 25 17  XXXXXX               135 131 139 141
 107       7 6 3 1   79   12 13 20 15 22 23  XXXXXX               141 140 130 127
 108       8 4 2 1   77   12  2 31 15  8 17  XXXXXX                   123 122 120
 109       8 3 2 1   76   12  2 14 15  8 17  XXXXXX                   123 121  65
 110       7 5 2 1   74    1 13 14  7 16 26  XXXXXX                   128 125 124
 111       7 3 2 1   73    1 13 14  7 16 17  XXXXXX                   128 127  68
 112       6 3 2 1   71   12 13  3 15 16  9  XXXXXX                   130 129  61
 113     7 6 4 2 1   78   12 13 14 15 16  6  XXXXXX       141 128 126  58  61  59
 114     8 6 5 2 1   83   12 13  3 15 16 17  XXXXXX       138 123 119  63  61  62
 115       8 7 3 1   89   12  2 14 15 16 17  XXXXXX               135 133 127  65
 116       8 7 3 2   91    1 13 14 15 16 17  XXXXXX               134 133 121  68
 117       8 5 4 3   75   28  2 14 15  5 17  ------                   143 142  85
 118       7 5 4 3   72   28  2 14  4 16 17  ------                   146 145  86
 119         8 5 2   69   27  2  3 15  8 17  ------                       144 142
 120         8 4 2   68    1  2 10 15  8 17  ------                       144 143
 121         8 3 2   67    1  2 14 15  8 17  ------                       144  85
 122         8 4 1   66   12  2 31 15  5 17  ------                       143  78
 123         8 2 1   64   12  2  3 15  8 17  ------                       144  78
 124         7 5 2   63    1 13 14  4 16 26  ------                       145  79
 125         7 5 1   61    1  2 14  7 16 11  ------                       147 145
 126         7 4 1   60   28  2 14 29 16  6  ------                       147 146
 127         7 3 1   59    1  2 14  7 16 17  ------                       147  86
 128         7 2 1   58    1 13 14  7 16  6  ------                       147  79
 129         6 3 2   56   12 13  3  4 16  9  ------                       148  72
 130         6 3 1   54   12 13  3 15  5  9  ------                       148  71
 131         8 7 5   52   27  2 24 15 25 17  ------                       145 142
 132         8 7 4   51   28  2 14 21 16 23  ------                       146 143
 133         8 7 3   50    1  2 14 15 16 17  ------                        86  85
 134         8 7 2   49    1 13 14 21 16 23  ------                       144  79
 135         8 7 1   48   12  2 24 15 25 17  ------                       147  76
 136         8 6 4   47   12 13 14 21  5 23  ------                       143  70
 137         8 6 3   46   12 13 20 21  5 23  ------                       148  85
 138         8 6 2   45   12 13  3 21 16 23  ------                       144  72
 139         7 6 5   44   12 13 24  4 25 17  ------                       145  73
 140         7 6 3   43   12 13 20  4 22 23  ------                       148 145
 141         7 6 1   42   12 13 24 15 25  6  ------                       147  71
 142           8 5   41   27  2  3 15  5 17  ------                             -
 143           8 4   40    1  2 10 15  5 17  ------                             -
 144           8 2   38    1  2  3 15  8 17  ------                             -
 145           7 5   36    1  2 14  4 16 11  ------                             -
 146           7 4   35   28  2 14  4 16  6  ------                             -
 147           7 1   32    1  2 14  7 16  6  ------                             -
 148           6 3   29   12 13  3  4  5  9  ------                             -
 149   7 6 5 4 3 2   81   12 13 14  4 16 17  XXXXX  140 139 130 118  59  68  60
 150   8 6 5 4 3 1   84   12 13 14 15  5 17  XXXXX  137 136 130 117  65  62  60
 151       8 7 4 1   90   12 32 20 15 22 17  XXXXX                135 132 126 122
 152   8 7 ... 2 1   97   12 13 14 15 16 17  ------                 .......

                              C-SET SEARCH TABLE
```

Table C.2 Continued

```
1.  EXP-(1): (SIN-COS)↑2
    EXP-(2): SIN↑2-2*SIN*COS+COS↑2
    EXP-(3): 1-2*SIN*COS

    COMPILATION TIME:       .021 SEC.

    STEP      N-COMP     TIME-A         TIME-B         RUN-1      RUN-2      RUN-3

1   1 ↱ 2       1      .007 SEC.      .007 SEC.      ALGEB      ALGEB      ALGEB
2   1 ↱ 3       7      .024 SEC.      .014 SEC.      SMALL      SMALL      LARGE
3   2 ↱ 3      26      .060 SEC.      .021 SEC.      SMALL      SMALL      SMALL


2.  EXP-(1): (1-SIN↑2)*(1+TAN↑2)
    EXP-(2): COS↑2*SEC↑2
    EXP-(3): 1

    COMPILATION TIME:       .019 SEC.

    STEP      N-COMP     TIME-A         TIME-B         RUN-1      RUN-2      RUN-3

4   1 ↱ 2      24      .071 SEC.      .030 SEC.      SMALL      SMALL      SMALL
5   1 ↱ 3      10      .024 SEC.      .007 SEC.      LARGE      LARGE      LARGE
6   2 ↱ 3      10      .027 SEC.      .014 SEC.      SMALL      SMALL      SMALL


3.  EXP-(1): COT↑4-CSC↑4
    EXP-(2): (COT↑2-CSC↑2)*(COT↑2+CSC↑2)
    EXP-(3): -1*(COT↑2+CSC↑2)
    EXP-(4): -1*(CSC↑2-1+CSC↑2)
    EXP-(5): 1-2*CSC↑2

    COMPILATION TIME:       .048 SEC.

    STEP      N-COMP     TIME-A         TIME-B         RUN-1      RUN-2      RUN-3

 7  1 ↱ 2       1      .006 SEC.      .006 SEC.      ALGEB      ALGEB      ALGEB
 8  1 ↱ 3      29      .099 SEC.      .031 SEC.      SMALL      LARGE      SMALL
 9  1 ↱ 4      49      .150 SEC.      .039 SEC.      LARGE      LARGE      LARGE
10  1 ↱ 5      16      .052 SEC.      .019 SEC.      LARGE      LARGE      LARGE
11  2 ↱ 3      14      .048 SEC.      .023 SEC.      SMALL      SMALL      SMALL
12  2 ↱ 4      34      .104 SEC.      .039 SEC.      SMALL      SMALL      SMALL
13  2 ↱ 5      34      .111 SEC.      .038 SEC.      SMALL      SMALL      LARGE
14  3 ↱ 4      23      .063 SEC.      .024 SEC.      SMALL      SMALL      SMALL
15  3 ↱ 5      27      .087 SEC.      .029 SEC.      SMALL      SMALL      LARGE
16  4 ↱ 5       1      .006 SEC.      .006 SEC.      ALGEB      ALGEB      ALGEB


4.  EXP-(1): (1-SIN↑2)*CSC↑2
    EXP-(2): COS↑2*CSC↑2
    EXP-(3): COS↑2/SIN↑2
    EXP-(4): COT↑2
```

Table C.3 Step-Size Analysis

```
       COMPILATION TIME:        .025 SEC.

       STEP      N-COMP      TIME-A          TIME-B         RUN-1      RUN-2      RUN-3

 17    1 ↑ 2       14      .045 SEC.      .019 SEC.       SMALL      SMALL      SMALL
 18    1 ↑ 3       16      .056 SEC.      .026 SEC.       SMALL      SMALL      SMALL
 19    1 ↑ 4       10      .033 SEC.      .015 SEC.       LARGE      LARGE      LARGE
 20    2 ↑ 3        5      .026 SEC.      .013 SEC.       SMALL      SMALL      SMALL
 21    2 ↑ 4       10      .036 SEC.      .020 SEC.       SMALL      SMALL      SMALL
 22    3 ↑ 4       10      .033 SEC.      .016 SEC.       SMALL      SMALL      SMALL


  5.   EXP-(1): (TAN↑3-COT↑3)/(TAN-COT)
       EXP-(2): ((TAN-COT)*(TAN↑2+COT↑2+TAN*COT))/(TAN-COT)
       EXP-(3): (TAN-COT)*(TAN↑2+COT↑2+1)/(TAN-COT)
       EXP-(4): TAN↑2+COT↑2+1

       COMPILATION TIME:        .052 SEC.

       STEP      N-COMP      TIME-A          TIME-B         RUN-1      RUN-2      RUN-3

 23    1 ↑ 2        1      .006 SEC.      .006 SEC.       ALGEB      ALGEB      ALGEB
 24    1 ↑ 3       30      .122 SEC.      .035 SEC.       LARGE      LARGE      LARGE
 25    1 ↑ 4       30      .097 SEC.      .030 SEC.       SMALL      SMALL      LARGE
 26    2 ↑ 3       21      .079 SEC.      .037 SEC.       SMALL      SMALL      SMALL
 27    2 ↑ 4       48      .141 SEC.      .034 SEC.       SMALL      SMALL      SMALL
 28    3 ↑ 4        1      .006 SEC.      .006 SEC.       ALGEB      ALGEB      ALGEB


  6.   EXP-(1): (TAN+SEC)/(TAN+SEC-COS)
       EXP-(2): (SIN/COS+1/COS)/(SIN/COS+1/COS-COS)
       EXP-(3): (SIN+1)/(SIN+SIN↑2)
       EXP-(4): (SIN+1)/(SIN*(SIN+1))
       EXP-(5): 1/SIN
       EXP-(6): CSC

       COMPILATION TIME:        .047 SEC.

       STEP      N-COMP      TIME-A          TIME-B         RUN-1      RUN-2      RUN-3

 29    1 ↑ 2       43      .114 SEC.      .052 SEC.       SMALL      SMALL      SMALL
 30    1 ↑ 3       16      .078 SEC.      .036 SEC.       LARGE      LARGE      LARGE
 31    1 ↑ 4       16      .074 SEC.      .035 SEC.       LARGE      LARGE      LARGE
 32    1 ↑ 5       25      .068 SEC.      .030 SEC.       LARGE      LARGE      LARGE
 33    1 ↑ 6       10      .031 SEC.      .014 SEC.       LARGE      LARGE      LARGE
 34    2 ↑ 3       16      .086 SEC.      .032 SEC.       LARGE      LARGE      LARGE
 35    2 ↑ 4       16      .082 SEC.      .032 SEC.       LARGE      LARGE      LARGE
 36    2 ↑ 5       25      .092 SEC.      .030 SEC.       LARGE      LARGE      LARGE
 37    2 ↑ 6       10      .041 SEC.      .018 SEC.       LARGE      LARGE      LARGE
 38    3 ↑ 4        1      .006 SEC.      .006 SEC.       ALGEB      ALGEB      ALGEB
 39    3 ↑ 5        1      .005 SEC.      .005 SEC.       ALGEB      ALGEB      ALGEB
 40    3 ↑ 6       10      .030 SEC.      .013 SEC.       SMALL      SMALL      SMALL
 41    4 ↑ 5        1      .005 SEC.      .005 SEC.       ALGEB      ALGEB      ALGEB
 42    4 ↑ 6       10      .028 SEC.      .013 SEC.       SMALL      SMALL      SMALL
 43    5 ↑ 6        8      .017 SEC.      .009 SEC.       SMALL      SMALL      SMALL
```

Table C.3 Continued

```
 7.  EXP-(1): (1-TAN)*(1-COT)
     EXP-(2): 1-TAN-COT+TAN*COT
     EXP-(3): 1-COT-TAN+1
     EXP-(4): 2-COS/SIN-SIN/COS
     EXP-(5): 2-(COS↑2+SIN↑2)/(SIN*COS)
     EXP-(6): 2-1/(SIN*COS)
     EXP-(7): 2-1/COS*1/SIN
     EXP-(8): 2-SEC*CSC

     COMPILATION TIME:      .056 SEC.

     STEP      N-COMP     TIME-A         TIME-B        RUN-1     RUN-2     RUN-3

44   1 ↱ 2        1      .007 SEC.      .007 SEC.      ALGEB     ALGEB     ALGEB
45   1 ↱ 3       69      .103 SEC.      .034 SEC.      SMALL     SMALL     LARGE
46   1 ↱ 4       30      .080 SEC.      .031 SEC.      LARGE     LARGE     LARGE
47   1 ↱ 5       15      .070 SEC.      .034 SEC.      LARGE     LARGE     LARGE
48   1 ↱ 6       15      .047 SEC.      .024 SEC.      LARGE     LARGE     LARGE
49   1 ↱ 7       15      .051 SEC.      .026 SEC.      LARGE     LARGE     LARGE
50   1 ↱ 8       15      .045 SEC.      .025 SEC.      LARGE     LARGE     LARGE
51   2 ↱ 3       19      .034 SEC.      .020 SEC.      SMALL     SMALL     SMALL
52   2 ↱ 4       51      .086 SEC.      .037 SEC.      SMALL     SMALL     LARGE
53   2 ↱ 5       66      .224 SEC.      .059 SEC.      LARGE     LARGE     LARGE
54   2 ↱ 6       59      .132 SEC.      .045 SEC.      LARGE     LARGE     LARGE
55   2 ↱ 7       69      .154 SEC.      .047 SEC.      LARGE     LARGE     LARGE
56   2 ↱ 8       46      .099 SEC.      .037 SEC.      LARGE     LARGE     LARGE
57   3 ↱ 4       33      .059 SEC.      .027 SEC.      SMALL     SMALL     SMALL
58   3 ↱ 5       63      .201 SEC.      .053 SEC.      LARGE     LARGE     LARGE
59   3 ↱ 6       56      .114 SEC.      .038 SEC.      LARGE     LARGE     LARGE
60   3 ↱ 7       66      .134 SEC.      .041 SEC.      LARGE     LARGE     LARGE
61   3 ↱ 8       43      .081 SEC.      .031 SEC.      LARGE     LARGE     LARGE
62   4 ↱ 5        1      .006 SEC.      .006 SEC.      ALGEB     ALGEB     ALGEB
63   4 ↱ 6       39      .101 SEC.      .033 SEC.      SMALL     LARGE     SMALL
64   4 ↱ 7       49      .118 SEC.      .034 SEC.      SMALL     LARGE     SMALL
65   4 ↱ 8       26      .067 SEC.      .025 SEC.      LARGE     LARGE     LARGE
66   5 ↱ 6       17      .058 SEC.      .024 SEC.      SMALL     SMALL     SMALL
67   5 ↱ 7       17      .060 SEC.      .025 SEC.      SMALL     SMALL     SMALL
68   5 ↱ 8       31      .082 SEC.      .036 SEC.      SMALL     SMALL     SMALL
69   6 ↱ 7        1      .005 SEC.      .005 SEC.      ALGEB     ALGEB     ALGEB
70   6 ↱ 8       26      .056 SEC.      .024 SEC.      SMALL     SMALL     SMALL
71   7 ↱ 8       24      .049 SEC.      .023 SEC.      SMALL     SMALL     SMALL
```

```
 8.  EXP-(1): ((1+TAN-SEC)/(SEC+TAN-1))/((1+SEC-TAN)/(SEC+TAN+1))
     EXP-(2): ((1+TAN-SEC)*(SEC+TAN+1))/((SEC+TAN-1)*(1+SEC-TAN))
     EXP-(3): ((1+TAN)↑2-SEC↑2)/(SEC↑2-(TAN-1)↑2)
     EXP-(4): (1+2*TAN+TAN↑2-SEC↑2)/(SEC↑2-TAN↑2+2*TAN-1)
     EXP-(5): 2*TAN/(2*TAN)
     EXP-(6): 1

     COMPILATION TIME:      .086 SEC.

     STEP      N-COMP     TIME-A         TIME-B        RUN-1     RUN-2     RUN-3

72   1 ↱ 2        1      .007 SEC.      .007 SEC.      ALGEB     ALGEB     ALGEB
73   1 ↱ 3        1      .005 SEC.      .005 SEC.      ALGEB     ALGEB     ALGEB
74   1 ↱ 4        1      .005 SEC.      .005 SEC.      ALGEB     ALGEB     ALGEB
75   1 ↱ 5       82      .275 SEC.      .053 SEC.      LARGE     LARGE     LARGE
76   1 ↱ 6       26      .116 SEC.      .032 SEC.      LARGE     LARGE     LARGE
77   2 ↱ 3        1      .006 SEC.      .006 SEC.      ALGEB     ALGEB     ALGEB
```

Table C. 3 Continued

```
 78   2 ↱ 4      1      .005 SEC.     .005 SEC.     ALGEB    ALGEB    ALGEB
 79   2 ↱ 5     61      .160 SEC.     .056 SEC.     SMALL    SMALL    LARGE
 80   2 ↱ 6     26      .115 SEC.     .033 SEC.     LARGE    LARGE    LARGE
 81   3 ↱ 4      1      .006 SEC.     .006 SEC.     ALGEB    ALGEB    ALGEB
 82   3 ↱ 5     46      .124 SEC.     .045 SEC.     SMALL    SMALL    LARGE
 83   3 ↱ 6     10      .049 SEC.     .025 SEC.     LARGE    LARGE    LARGE
 84   4 ↱ 5    128      .296 SEC.     .077 SEC.     SMALL    SMALL    LARGE
 85   4 ↱ 6     10      .067 SEC.     .032 SEC.     LARGE    LARGE    LARGE
 86   5 ↱ 6      1      .007 SEC.     .007 SEC.     ALGEB    ALGEB    ALGEB


  9.  EXP-(1): (1+SIN+COS)↑2
      EXP-(2): 1+SIN↑2+COS↑2+2*SIN+2*COS+2*SIN*COS
      EXP-(3): 2+2*SIN+2*COS+2*SIN*COS
      EXP-(4): 2*(1+SIN)*(1+COS)

      COMPILATION TIME:      .043 SEC.

      STEP     N-COMP     TIME-A        TIME-B        RUN-1    RUN-2    RUN-3

 87   1 ↱ 2      1      .007 SEC.     .007 SEC.     ALGEB    ALGEB    ALGEB
 88   1 ↱ 3     61      .179 SEC.     .039 SEC.     SMALL    SMALL    LARGE
 89   1 ↱ 4     22      .054 SEC.     .023 SEC.     SMALL    SMALL    LARGE
 90   2 ↱ 3     99      .196 SEC.     .051 SEC.     SMALL    SMALL    LARGE
 91   2 ↱ 4    189      .808 SEC.     .094 SEC.     SMALL    SMALL    LARGE
 92   3 ↱ 4      1      .008 SEC.     .008 SEC.     ALGEB    ALGEB    ALGEB


 10.  EXP-(1): (1+SEC)/(SEC*TAN-2*SIN-TAN)
      EXP-(2): (1+SEC)/(SEC*TAN+TAN-2*TAN-2*SIN)
      EXP-(3): (1+SEC)/(SEC*TAN+TAN-2*SEC*SIN-2*SIN)
      EXP-(4): (1+SEC)/((1+SEC)*(TAN-2*SIN))
      EXP-(5): 1/(TAN-2*SIN)

      COMPILATION TIME:      .055 SEC.

      STEP     N-COMP     TIME-A        TIME-B        RUN-1    RUN-2    RUN-3

 93   1 ↱ 2      1      .007 SEC.     .007 SEC.     ALGEB    ALGEB    ALGEB
 94   1 ↱ 3     24      .076 SEC.     .032 SEC.     LARGE    LARGE    LARGE
 95   1 ↱ 4     34      .119 SEC.     .036 SEC.     LARGE    LARGE    LARGE
 96   1 ↱ 5     25      .098 SEC.     .035 SEC.     LARGE    LARGE    LARGE
 97   2 ↱ 3     26      .082 SEC.     .039 SEC.     SMALL    SMALL    SMALL
 98   2 ↱ 4     69      .262 SEC.     .054 SEC.     LARGE    LARGE    LARGE
 99   2 ↱ 5     25      .114 SEC.     .039 SEC.     LARGE    LARGE    LARGE
100   3 ↱ 4      1      .007 SEC.     .007 SEC.     ALGEB    ALGEB    ALGEB
101   3 ↱ 5      1      .005 SEC.     .005 SEC.     ALGEB    ALGEB    ALGEB
102   4 ↱ 5      1      .006 SEC.     .006 SEC.     ALGEB    ALGEB    ALGEB


 11.  EXP-(1): (1+SEC)/(SEC*TAN-2*SIN-TAN)
      EXP-(2): (1+1/COS)/(1/COS*SIN/COS-2*SIN-SIN/COS)
      EXP-(3): ((COS+1)/COS)/((SIN-2*SIN*COS↑2-SIN*COS)/COS↑2)
      EXP-(4): (COS+1)*COS↑2/(COS*(SIN-2*SIN*COS↑2-SIN*COS))
      EXP-(5): (1+COS)*COS/(SIN*(1-2*COS↑2-COS))
      EXP-(6): (1+COS)/(SIN/COS*(1-2*COS)*(1+COS))
      EXP-(7): 1/(TAN-2*SIN)
```

Table C.3 Continued

```
      COMPILATION TIME:       .105 SEC.

      STEP       N-COMP      TIME-A          TIME-B          RUN-1      RUN-2      RUN-3

103   1 → 2        55       .146 SEC.       .060 SEC.       SMALL      SMALL      SMALL
104   1 → 3        62       .253 SEC.       .083 SEC.       SMALL      SMALL      LARGE
105   1 → 4        36       .254 SEC.       .071 SEC.       LARGE      LARGE      LARGE
106   1 → 5        36       .195 SEC.       .060 SEC.       LARGE      LARGE      LARGE
107   1 → 6        16       .116 SEC.       .051 SEC.       LARGE      LARGE      LARGE
108   1 → 7        25       .098 SEC.       .035 SEC.       LARGE      LARGE      LARGE
109   2 → 3         1       .006 SEC.       .006 SEC.       ALGEB      ALGEB      ALGEB
110   2 → 4         1       .005 SEC.       .005 SEC.       ALGEB      ALGEB      ALGEB
111   2 → 5         1       .005 SEC.       .005 SEC.       ALGEB      ALGEB      ALGEB
112   2 → 6         1       .005 SEC.       .005 SEC.       ALGEB      ALGEB      ALGEB
113   2 → 7        25       .123 SEC.       .037 SEC.       LARGE      LARGE      LARGE
114   3 → 4         1       .007 SEC.       .007 SEC.       ALGEB      ALGEB      ALGEB
115   3 → 5         1       .005 SEC.       .005 SEC.       ALGEB      ALGEB      ALGEB
116   3 → 6         1       .005 SEC.       .005 SEC.       ALGEB      ALGEB      ALGEB
117   3 → 7        61       .308 SEC.       .053 SEC.       LARGE      LARGE      LARGE
118   4 → 5         1       .006 SEC.       .006 SEC.       ALGEB      ALGEB      ALGEB
119   4 → 6         1       .006 SEC.       .006 SEC.       ALGEB      ALGEB      ALGEB
120   4 → 7        61       .308 SEC.       .054 SEC.       LARGE      LARGE      LARGE
121   5 → 6         1       .006 SEC.       .006 SEC.       ALGEB      ALGEB      ALGEB
122   5 → 7        61       .238 SEC.       .047 SEC.       LARGE      LARGE      LARGE
123   6 → 7        25       .107 SEC.       .033 SEC.       LARGE      LARGE      LARGE
```

```
  12.  EXP-(1):  CSC/(CSC-1)+CSC/(CSC+1)
       EXP-(2):  (1/SIN)/(1/SIN-1)+(1/SIN)/(1/SIN+1)
       EXP-(3):  1/(1-SIN)+1/(1+SIN)
       EXP-(4):  (1+SIN+1-SIN)/(1-SIN↑2)
       EXP-(5):  2/COS↑2
       EXP-(6):  2*SEC↑2

       COMPILATION TIME:       .052 SEC.

       STEP       N-COMP      TIME-A          TIME-B          RUN-1      RUN-2      RUN-3

 124   1 → 2        50       .138 SEC.       .063 SEC.       SMALL      SMALL      SMALL
 125   1 → 3        54       .124 SEC.       .047 SEC.       LARGE      LARGE      LARGE
 126   1 → 4        15       .068 SEC.       .027 SEC.       LARGE      LARGE      LARGE
 127   1 → 5        15       .040 SEC.       .011 SEC.       LARGE      LARGE      LARGE
 128   1 → 6        15       .039 SEC.       .010 SEC.       LARGE      LARGE      LARGE
 129   2 → 3         1       .006 SEC.       .006 SEC.       ALGEB      ALGEB      ALGEB
 130   2 → 4         1       .005 SEC.       .005 SEC.       ALGEB      ALGEB      ALGEB
 131   2 → 5        15       .066 SEC.       .026 SEC.       SMALL      LARGE      SMALL
 132   2 → 6        15       .052 SEC.       .012 SEC.       LARGE      LARGE      LARGE
 133   3 → 4         1       .008 SEC.       .008 SEC.       ALGEB      ALGEB      ALGEB
 134   3 → 5        15       .047 SEC.       .020 SEC.       SMALL      LARGE      SMALL
 135   3 → 6        15       .037 SEC.       .010 SEC.       LARGE      LARGE      LARGE
 136   4 → 5        13       .043 SEC.       .018 SEC.       SMALL      SMALL      SMALL
 137   4 → 6         5       .024 SEC.       .010 SEC.       LARGE      LARGE      LARGE
 138   5 → 6         5       .022 SEC.       .011 SEC.       SMALL      SMALL      SMALL
```

```
  13.  EXP-(1):  CSC/(COT+TAN)
       EXP-(2):  (1/SIN)/(COS/SIN+SIN/COS)
       EXP-(3):  (1/SIN)/((COS↑2+SIN↑2)/(SIN*COS))
       EXP-(4):  (1/SIN)*(SIN*COS)/(SIN↑2+COS↑2)
```

Table C.3 Continued

```
        EXP-(5): COS

        COMPILATION TIME:      .048 SEC.

        STEP      N-COMP     TIME-A         TIME-B        RUN-1     RUN-2     RUN-3

  139   1 ↱ 2       30      .072 SEC.      .032 SEC.      SMALL     SMALL     SMALL
  140   1 ↱ 3       47      .134 SEC.      .051 SEC.      LARGE     LARGE     LARGE
  141   1 ↱ 4       21      .069 SEC.      .037 SEC.      SMALL     SMALL     SMALL
  142   1 ↱ 5       10      .021 SEC.      .012 SEC.      LARGE     LARGE     LARGE
  143   2 ↱ 3        1      .006 SEC.      .006 SEC.      ALGEB     ALGEB     ALGEB
  144   2 ↱ 4        1      .006 SEC.      .006 SEC.      ALGEB     ALGEB     ALGEB
  145   2 ↱ 5       28      .070 SEC.      .027 SEC.      SMALL     LARGE     SMALL
  146   3 ↱ 4        1      .006 SEC.      .006 SEC.      ALGEB     ALGEB     ALGEB
  147   3 ↱ 5       28      .095 SEC.      .033 SEC.      SMALL     LARGE     SMALL
  148   4 ↱ 5       29      .059 SEC.      .026 SEC.      SMALL     SMALL     SMALL


   14.  EXP-(1): (SEC-TAN)/(SEC+TAN)
        EXP-(2): (SEC-TAN)*(SEC-TAN)/((SEC+TAN)*(SEC-TAN))
        EXP-(3): (SEC↑2-2*SEC*TAN+TAN↑2)/(SEC↑2-TAN↑2)
        EXP-(4): SEC↑2-2*SEC*TAN+TAN↑2
        EXP-(5): 1-2*SEC*TAN+2*TAN↑2

        COMPILATION TIME:      .060 SEC.

        STEP      N-COMP     TIME-A         TIME-B        RUN-1     RUN-2     RUN-3

  149   1 ↱ 2        1      .007 SEC.      .007 SEC.      ALGEB     ALGEB     ALGEB
  150   1 ↱ 3        1      .005 SEC.      .005 SEC.      ALGEB     ALGEB     ALGEB
  151   1 ↱ 4       30      .103 SEC.      .028 SEC.      SMALL     LARGE     SMALL
  152   1 ↱ 5       30      .097 SEC.      .027 SEC.      LARGE     LARGE     LARGE
  153   2 ↱ 3        1      .006 SEC.      .006 SEC.      ALGEB     ALGEB     ALGEB
  154   2 ↱ 4       46      .172 SEC.      .033 SEC.      SMALL     SMALL     SMALL
  155   2 ↱ 5       74      .252 SEC.      .056 SEC.      SMALL     SMALL     LARGE
  156   3 ↱ 4       25      .117 SEC.      .024 SEC.      SMALL     SMALL     SMALL
  157   3 ↱ 5       62      .228 SEC.      .054 SEC.      SMALL     SMALL     LARGE
  158   4 ↱ 5       39      .115 SEC.      .031 SEC.      SMALL     SMALL     LARGE


   15.  EXP-(1): TAN/(1-COT)+COT/(1-TAN)
        EXP-(2): (SIN/COS)/(1-COS/SIN)+(COS/SIN)/(1-SIN/COS)
        EXP-(3): SIN↑2/(COS*(SIN-COS))+COS↑2/(SIN*(COS-SIN))
        EXP-(4): (SIN↑3-COS↑3)/(COS*SIN*(SIN-COS))
        EXP-(5): (SIN-COS)*(SIN↑2+COS↑2+SIN*COS)/((SIN-COS)*SIN*COS)
        EXP-(6): (SIN↑2+COS↑2)/(SIN*COS)+(SIN*COS)/(SIN*COS)
        EXP-(7): 1/(COS*SIN)+1
        EXP-(8): SEC*CSC+1

        COMPILATION TIME:      .109 SEC.

        STEP      N-COMP     TIME-A         TIME-B        RUN-1     RUN-2     RUN-3

  159   1 ↱ 2       70      .152 SEC.      .064 SEC.      SMALL     SMALL     SMALL
  160   1 ↱ 3       36      .157 SEC.      .062 SEC.      LARGE     LARGE     LARGE
  161   1 ↱ 4       15      .090 SEC.      .041 SEC.      LARGE     LARGE     LARGE
  162   1 ↱ 5       30      .188 SEC.      .056 SEC.      LARGE     LARGE     LARGE
  163   1 ↱ 6       16      .124 SEC.      .047 SEC.      LARGE     LARGE     LARGE
```

Table C.3 Continued

```
164   1 → 7     16      .065 SEC.     .028 SEC.     LARGE     LARGE     LARGE
165   1 → 8     16      .062 SEC.     .031 SEC.     LARGE     LARGE     LARGE
166   2 → 3      1      .006 SEC.     .006 SEC.     ALGEB     ALGEB     ALGEB
167   2 → 4      1      .005 SEC.     .005 SEC.     ALGEB     ALGEB     ALGEB
168   2 → 5      1      .006 SEC.     .006 SEC.     ALGEB     ALGEB     ALGEB
169   2 → 6      1      .005 SEC.     .005 SEC.     ALGEB     ALGEB     ALGEB
170   2 → 7     16      .075 SEC.     .028 SEC.     LARGE     LARGE     LARGE
171   2 → 8     16      .076 SEC.     .032 SEC.     LARGE     LARGE     LARGE
172   3 → 4      1      .008 SEC.     .008 SEC.     ALGEB     ALGEB     ALGEB
173   3 → 5      1      .005 SEC.     .005 SEC.     ALGEB     ALGEB     ALGEB
174   3 → 6      1      .005 SEC.     .005 SEC.     ALGEB     ALGEB     ALGEB
175   3 → 7     16      .081 SEC.     .028 SEC.     LARGE     LARGE     LARGE
176   3 → 8     16      .081 SEC.     .033 SEC.     LARGE     LARGE     LARGE
177   4 → 5      1      .006 SEC.     .006 SEC.     ALGEB     ALGEB     ALGEB
178   4 → 6      1      .005 SEC.     .005 SEC.     ALGEB     ALGEB     ALGEB
179   4 → 7     15      .069 SEC.     .026 SEC.     LARGE     LARGE     LARGE
180   4 → 8     15      .067 SEC.     .029 SEC.     LARGE     LARGE     LARGE
181   5 → 6      1      .006 SEC.     .006 SEC.     ALGEB     ALGEB     ALGEB
182   5 → 7     31      .171 SEC.     .040 SEC.     LARGE     LARGE     LARGE
183   5 → 8     31      .168 SEC.     .043 SEC.     LARGE     LARGE     LARGE
184   6 → 7     18      .072 SEC.     .031 SEC.     SMALL     SMALL     SMALL
185   6 → 8     32      .094 SEC.     .042 SEC.     SMALL     SMALL     SMALL
186   7 → 8     25      .060 SEC.     .026 SEC.     SMALL     SMALL     SMALL


 16.  EXP-(1): (1+COT-CSC)*(1+TAN+SEC)
      EXP-(2): (1+COS/SIN-1/SIN)*(1+SIN/COS+1/COS)
      EXP-(3): ((SIN+COS-1)/SIN)*((COS+SIN+1)/COS)
      EXP-(4): ((SIN+COS)↑2-1)/(SIN*COS)
      EXP-(5): (SIN↑2+COS↑2+2*SIN*COS-1)/(SIN*COS)
      EXP-(6): (1+2*SIN*COS-1)/(SIN*COS)
      EXP-(7): 2

      COMPILATION TIME:      .078 SEC.

      STEP     N-COMP     TIME-A        TIME-B        RUN-1     RUN-2     RUN-3

187   1 → 2     44      .116 SEC.     .055 SEC.     SMALL     SMALL     SMALL
188   1 → 3     78      .196 SEC.     .068 SEC.     LARGE     LARGE     LARGE
189   1 → 4     16      .132 SEC.     .081 SEC.     LARGE     LARGE     LARGE
190   1 → 5     16      .183 SEC.     .110 SEC.     LARGE     LARGE     LARGE
191   1 → 6     16      .135 SEC.     .084 SEC.     LARGE     LARGE     LARGE
192   1 → 7     10      .038 SEC.     .022 SEC.     LARGE     LARGE     LARGE
193   2 → 3      1      .006 SEC.     .006 SEC.     ALGEB     ALGEB     ALGEB
194   2 → 4      1      .005 SEC.     .005 SEC.     ALGEB     ALGEB     ALGEB
195   2 → 5      1      .006 SEC.     .006 SEC.     ALGEB     ALGEB     ALGEB
196   2 → 6     16      .098 SEC.     .037 SEC.     LARGE     LARGE     LARGE
197   2 → 7     10      .046 SEC.     .023 SEC.     LARGE     LARGE     LARGE
198   3 → 4      1      .007 SEC.     .007 SEC.     ALGEB     ALGEB     ALGEB
199   3 → 5      1      .005 SEC.     .005 SEC.     ALGEB     ALGEB     ALGEB
200   3 → 6     39      .195 SEC.     .042 SEC.     LARGE     LARGE     LARGE
201   3 → 7     26      .080 SEC.     .025 SEC.     LARGE     LARGE     LARGE
202   4 → 5      1      .007 SEC.     .007 SEC.     ALGEB     ALGEB     ALGEB
203   4 → 6     26      .081 SEC.     .031 SEC.     SMALL     SMALL     LARGE
204   4 → 7     10      .033 SEC.     .016 SEC.     LARGE     LARGE     LARGE
205   5 → 6     45      .115 SEC.     .043 SEC.     SMALL     SMALL     SMALL
206   5 → 7     10      .051 SEC.     .024 SEC.     LARGE     LARGE     LARGE
207   6 → 7      1      .007 SEC.     .007 SEC.     ALGEB     ALGEB     ALGEB
```

Table C.3 Continued

```
INIT TIME=    R.   R      NOW=    R.   R      FLAPSED  TIME=*9309.7500  SE(

     1.  EXP-(1): (SIN-COS)↑2
         EXP-(2): SIN↑2-2*SIN*COS+COS↑2
         EXP-(3): 1-2*SIN*COS


     1   1 → 2         ALGER
     2   1 → 3         F(X)+G(L) → F(X)+G(R):  SIN2+COS2→1
     3   2 → 3         G(L) → G(R):  SIN2+COS2→1


     2.  EXP-(1): (1-SIN↑2)*(1+TAN↑2)
         EXP-(2): COS↑2*SEC↑2
         EXP-(3): 1


     4   1 → 2         G(L) → G(R):  1-SIN2→COS2
                       G(L) → G(R):  TAN2+1→SEC2
     5   1 → 3         NON-SINGLETON CSET
     6   2 → 3         G(L)↑2 → G(R)↑2:  SEC*COS→1


     3.  EXP-(1): COT↑4-CSC↑4
         EXP-(2): (COT↑2-CSC↑2)*(COT↑2+CSC↑2)
         EXP-(3): -1*(COT↑2+CSC↑2)
         EXP-(4): -1*(CSC↑2-1+CSC↑2)
         EXP-(5): 1-2*CSC↑2


     7   1 → 2         ALGER
     8   1 → 3         F(X)*G(L) → F(X)*G(R):  CSC2-COT2→1
     9   1 → 4         F(X)+G(L)↑2 → F(X)+G(R)↑2:  CSC2-1→COT2
    10   1 → 5         F(X)-G(L)↑2 → F(X)-G(R)↑2:  CSC2-1→COT2
    11   2 → 3         G(L) → G(R):  CSC2-COT2→1
    12   2 → 4         G(L) → G(R):  CSC2-COT2→1
                       G(L) → G(R):  COT2→CSC2-1
    13   2 → 5         G(L) → G(R):  CSC2-COT2→1
                       F(X)+G(L) → F(X)+G(R):  CSC2→COT2+1
    14   3 → 4         G(L) → G(R):  COT2→CSC2-1
    15   3 → 5         F(X)+G(L) → F(X)+G(R):  CSC2→COT2+1
    16   4 → 5         ALGER


     4.  EXP-(1): (1-SIN↑2)*CSC↑2
         EXP-(2): COS↑2*CSC↑2
         EXP-(3): COS↑2/SIN↑2
         EXP-(4): COT↑2


    17   1 → 2         G(L) → G(R):  1-SIN2→COS2
    18   1 → 3         G(L) → G(R):  1-SIN2→COS2
                       G(L) → G(R):  1/CSC→SIN
    19   1 → 4         NON-SINGLETON CSET
    20   2 → 3         G(L) → G(R):  1/CSC→SIN
    21   2 → 4         G(L)↑2 → G(R)↑2:  COS*CSC→COT
    22   3 → 4         G(L)↑2 → G(R)↑2:  COS/SIN→COT


     5.  EXP-(1): (TAN↑3-COT↑3)/(TAN-COT)
         EXP-(2): ((TAN-COT)*(TAN↑2+COT↑2+TAN*COT))/(TAN-COT)
         EXP-(3): (TAN-COT)*(TAN↑2+COT↑2+1)/(TAN-COT)
         EXP-(4): TAN↑2+COT↑2+1
```

Table C.4 Latent Structure Analysis

```
23    1 → 2         ALGEB
24    1 → 3         UNCLASSIFIED
25    1 → 4         F(X)+G(L) → F(X)+G(R):  COT*TAN→1
26    2 → 3         G(L) → G(R):  COT*TAN→1
27    2 → 4         G(L) → G(R):  COT*TAN→1
28    3 → 4         ALGEB

 6.   EXP-(1): (TAN+SEC)/(TAN+SEC-COS)
      EXP-(2): (SIN/COS+1/COS)/(SIN/COS+1/COS-COS)
      EXP-(3): (SIN+1)/(SIN+SIN↑2)
      EXP-(4): (SIN+1)/(SIN*(SIN+1))
      EXP-(5): 1/SIN
      EXP-(6): CSC

29    1 → 2         G(L) → G(R):  TAN→SIN/COS
                    G(L) → G(R):  TAN→SIN/COS
                    G(L) → G(R):  SEC→1/COS
                    G(L) → G(R):  SEC→1/COS
30    1 → 3         NON-SINGLETON CSET
31    1 → 4         NON-SINGLETON CSET
32    1 → 5         NON-SINGLETON CSET
33    1 → 6         NON-SINGLETON CSET
34    2 → 3         UNCLASSIFIED
35    2 → 4         UNCLASSIFIED
36    2 → 5         UNCLASSIFIED
37    2 → 6         NON-SINGLETON CSET
38    3 → 4         ALGEB
39    3 → 5         ALGEB
40    3 → 6         G(L) → G(R):  1/SIN→CSC
41    4 → 5         ALGEB
42    4 → 6         G(L) → G(R):  1/SIN→CSC
43    5 → 6         G(L) → G(R):  1/SIN→CSC

 7.   EXP-(1): (1-TAN)*(1-COT)
      EXP-(2): 1-TAN-COT+TAN*COT
      EXP-(3): 1-COT-TAN+1
      EXP-(4): 2-COS/SIN-SIN/COS
      EXP-(5): 2-(COS↑2+SIN↑2)/(SIN*COS)
      EXP-(6): 2-1/(SIN*COS)
      EXP-(7): 2-1/COS*1/SIN
      EXP-(8): 2-SEC*CSC

44    1 → 2         ALGEB
45    1 → 3         F(X)+G(L) → F(X)+G(R):  COT*TAN→1
46    1 → 4         NON-SINGLETON CSET
47    1 → 5         NON-SINGLETON CSET
48    1 → 6         NON-SINGLETON CSET
49    1 → 7         NON-SINGLETON CSET
50    1 → 8         NON-SINGLETON CSET
51    2 → 3         G(L) → G(R):  COT*TAN→1
52    2 → 4         G(L) → G(R):  TAN→SIN/COS
                    G(L) → G(R):  COT→COS/SIN
                    F(X)+G(L) → F(X)+G(R):  COT*TAN→1
53    2 → 5         F(X)+G(L) → F(X)+G(R):  COT*TAN→1
                    G(L) → G(R):  SIN2+COS2→1
                    NON-SINGLETON CSET
```

Table C.4 Continued

```
54    2 → 6        F(X)+G(L) → F(X)+G(R):  COT*TAN→1
                   NON-SINGLETON CSET
55    2 → 7        F(X)+G(L) → F(X)+G(R):  COT*TAN→1
                   NON-SINGLETON CSET
56    2 → 8        F(X)+G(L) → F(X)+G(R):  COT*TAN→1
                   NON-SINGLETON CSET
57    3 → 4        G(L) → G(R):  COT→COS/SIN
                   G(L) → G(R):  TAN→SIN/COS
58    3 → 5        G(L) → G(R):  SIN2+COS2→1
                   NON-SINGLETON CSET
59    3 → 6        NON-SINGLETON CSET
60    3 → 7        NON-SINGLETON CSET
61    3 → 8        NON-SINGLETON CSET
62    4 → 5        ALGEB
63    4 → 6        F(X)*G(L) → F(X)*G(R):  SIN2+COS2→1
64    4 → 7        F(X)*G(L) → F(X)*G(R):  SIN2+COS2→1
65    4 → 8        NON-SINGLETON CSET
66    5 → 6        G(L) → G(R):  SIN2+COS2→1
67    5 → 7        G(L) → G(R):  SIN2+COS2→1
68    5 → 8        G(L) → G(R):  SIN2+COS2→1
                   G(L) → G(R):  1/SIN→CSC
                   G(L) → G(R):  1/COS→SEC
69    6 → 7        ALGER
70    6 → 8        G(L) → G(R):  1/SIN→CSC
                   G(L) → G(R):  1/COS→SEC
71    7 → 8        G(L) → G(R):  1/COS→SEC
                   G(L) → G(R):  1/SIN→CSC


 8.  EXP-(1): ((1+TAN-SEC)/(SEC+TAN-1))/((1+SEC-TAN)/(SEC+TAN+1))
     EXP-(2): ((1+TAN-SEC)*(SEC+TAN+1))/((SEC+TAN-1)*(1+SEC-TAN))
     EXP-(3): ((1+TAN)↑2-SEC↑2)/(SEC↑2-(TAN-1)↑2)
     EXP-(4): (1+2*TAN+TAN↑2-SEC↑2)/(SEC↑2-TAN↑2+2*TAN-1)
     EXP-(5): 2*TAN/(2*TAN)
     EXP-(6): 1


72    1 → 2        ALGER
73    1 → 3        ALGER
74    1 → 4        ALGER
75    1 → 5        UNCLASSIFIED
76    1 → 6        UNCLASSIFIED
77    2 → 3        ALGER
78    2 → 4        ALGER
79    2 → 5        F(X)+G(L) → F(X)+G(R):  SEC2→TAN2+1
                   F(X)+G(L) → F(X)+G(R):  TAN2+1→SEC2
80    2 → 6        UNCLASSIFIED
81    3 → 4        ALGER
82    3 → 5        F(X)+G(L) → F(X)+G(R):  SEC2→TAN2+1
                   F(X)+G(L) → F(X)+G(R):  TAN2+1→SEC2
83    3 → 6        UNCLASSIFIED
84    4 → 5        F(X)*(G(L) → 0):  SEC2-TAN2-1→0
                   F(X)*(G(L) → 0):  SEC2-TAN2-1→0
85    4 → 6        UNCLASSIFIED
86    5 → 6        ALGER

 9.  EXP-(1): (1+SIN+COS)↑2
     EXP-(2): 1+SIN↑2+COS↑2+2*SIN+2*COS+2*SIN*COS
     EXP-(3): 2+2*SIN+2*COS+2*SIN*COS
     EXP-(4): 2*(1+SIN)*(1+COS)
```

Table C.4 Continued

```
 87    1 → 2          ALGEB
 88    1 → 3          F(X)+G(L) → F(X)+G(R):  SIN2+COS2→1
 89    1 → 4          F(X)+G(L) → F(X)+G(R):  SIN2+COS2→1
 90    2 → 3          F(X)+G(L) → F(X)+G(R):  SIN2+COS2→1
 91    2 → 4          F(X)+G(L) → F(X)+G(R):  SIN2+COS2→1
 92    3 → 4          ALGEB


 10.  EXP-(1): (1+SEC)/(SFC*TAN-2*SIN-TAN)
      EXP-(2): (1+SEC)/(SFC*TAN+TAN-2*TAN-2*SIN)
      EXP-(3): (1+SEC)/(SFC*TAN+TAN-2*SEC*SIN-2*SIN)
      EXP-(4): (1+SEC)/((1+SEC)*(TAN-2*SIN))
      EXP-(5): 1/(TAN-2*STN)


 93    1 → 2          ALGEB
 94    1 → 3          F(X)+2*G(L) → F(X)+2*G(R):  SIN*SEC→TAN
 95    1 → 4          F(X)+2*G(L) → F(X)+2*G(R):  SIN*SEC→TAN
 96    1 → 5          UNCLASSIFIED
 97    2 → 3          G(L) → G(R):  TAN→SIN*SEC
 98    2 → 4          F(X)+2*G(L) → F(X)+2*G(R):  SIN*SEC→TAN
 99    2 → 5          UNCLASSIFIED
100    3 → 4          ALGEB
101    3 → 5          ALGEB
102    4 → 5          ALGEB


 11.  EXP-(1): (1+SEC)/(SFC*TAN-2*SIN-TAN)
      EXP-(2): (1+1/COS)/(1/COS*SIN/COS-2*SIN-SIN/COS)
      EXP-(3): ((COS+1)/COS)/((SIN-2*SIN*COS↑2-SIN*COS)/COS↑2)
      EXP-(4): (COS+1)*COS↑2/(COS*(SIN-2*SIN*COS↑2-SIN*COS))
      EXP-(5): (1+COS)*COS/(SIN*(1-2*COS↑2-COS))
      EXP-(6): (1+COS)/(SIN/COS*(1-2*COS)*(1+COS))
      EXP-(7): 1/(TAN-2*STN)


103    1 → 2          G(L) → G(R):  TAN→SIN/COS
                      G(L) → G(R):  SEC→1/COS
                      G(L) → G(R):  SEC→1/COS
                      G(L) → G(R):  TAN→SIN/COS
104    1 → 3          NON-SINGLETON CSET
                      F(X)+G(L) → F(X)+G(R):  SEC→1/COS
105    1 → 4          NON-SINGLETON CSET
106    1 → 5          NON-SINGLETON CSET
107    1 → 6          NON-SINGLETON CSET
108    1 → 7          UNCLASSIFIED
109    2 → 3          ALGEB
110    2 → 4          ALGEB
111    2 → 5          ALGEB
112    2 → 6          ALGEB
113    2 → 7     RCP  F(X)+G(L) → F(X)+G(R):  SIN/COS→TAN
114    3 → 4          ALGEB
115    3 → 5          ALGEB
116    3 → 6          ALGEB
117    3 → 7     RCP  F(X)+G(L) → F(X)+G(R):  SIN/COS→TAN
118    4 → 5          ALGEB
119    4 → 6          ALGEB
120    4 → 7     RCP  F(X)+G(L) → F(X)+G(R):  SIN/COS→TAN
121    5 → 6          ALGEB
122    5 → 7     RCP  F(X)+G(L) → F(X)+G(R):  SIN/COS→TAN
123    6 → 7     RCP  F(X)+G(L) → F(X)+G(R):  SIN/COS→TAN
```

Table C.4 Continued

```
 12.  EXP-(1): CSC/(CSC-1)+CSC/(CSC+1)
      EXP-(2): (1/SIN)/(1/SIN- )+(1/SIN)/(1/SIN+1)
      EXP-(3): 1/(1-SIN)+1/(1+SIN)
      EXP-(4): (1+SIN+1-SIN)/(1-SIN↑2)
      EXP-(5): 2/COS↑2
      EXP-(6): 2*SEC↑2


124   1 → 2         G(L) → G(R):  CSC→1/SIN
                    G(L) → G(R):  CSC→1/SIN
                    G(L) → G(R):  CSC→1/SIN
                    G(L) → G(R):  CSC→1/SIN
125   1 → 3    RCP  F(X)+G(L) → F(X)+G(R):  SIN→1/CSC
               RCP  F(X)+G(L) → F(X)+G(R):  1/CSC→SIN
126   1 → 4         UNCLASSIFIED
127   1 → 5         NON-SINGLETON CSET
128   1 → 6         NON-SINGLETON CSET
129   2 → 3         ALGEB
130   2 → 4         ALGEB
131   2 → 5         F(X)*G(L) → F(X)*G(R):  COS2→1-SIN2
132   2 → 6         NON-SINGLETON CSET
133   3 → 4         ALGEB
134   3 → 5         F(X)*G(L) → F(X)*G(R):  COS2→1-SIN2
135   3 → 6         NON-SINGLETON CSET
136   4 → 5         G(L) → G(R):  1-SIN2→COS2
137   4 → 6         NON-SINGLETON CSET
138   5 → 6         G(L) → G(R):  1/COS→SEC


 13.  EXP-(1): CSC/(COT+TAN)
      EXP-(2): (1/SIN)/(COS/SIN+SIN/COS)
      EXP-(3): (1/SIN)/((COS↑2+SIN↑2)/(SIN*COS))
      EXP-(4): (1/SIN)*(SIN*COS)/(SIN↑2+COS↑2)
      EXP-(5): COS


139   1 → 2         G(L) → G(R):  CSC→1/SIN
                    G(L) → G(R):  COT→COS/SIN
                    G(L) → G(R):  TAN→SIN/COS
140   1 → 3         G(L) → G(R):  CSC→1/SIN
                    G(L) → G(R):  SIN2+COS2→1
                    NON-SINGLETON CSET
141   1 → 4         G(L) → G(R):  CSC→1/SIN
                    NON-SINGLETON CSET
                    F(X)*G(L) → F(X)*G(R):  SIN2+COS2→1
142   1 → 5         NON-SINGLETON CSET
143   2 → 3         ALGEB
144   2 → 4         ALGEB
145   2 → 5         F(X)*G(L) → F(X)*G(R):  1→SIN2+COS2
146   3 → 4         ALGEB
147   3 → 5         F(X)*G(L) → F(X)*G(R):  1→SIN2+COS2
148   4 → 5         F(X)*G(L) → F(X)*G(R):  1→SIN2+COS2


 14.  EXP-(1): (SEC-TAN)/(SEC+TAN)
      EXP-(2): (SEC-TAN)*(SEC-TAN)/((SEC+TAN)*(SEC-TAN))
      EXP-(3): (SEC↑2-2*SEC*TAN+TAN↑2)/(SEC↑2-TAN↑2)
      EXP-(4): SEC↑2-2*SEC*TAN+TAN↑2
      EXP-(5): 1-2*SEC*TAN+2*TAN↑2


149   1 → 2         ALGEB
150   1 → 3         ALGEB
```

<u>Table C.4 Continued</u>

```
151   1 → 4        F(X)*G(L) → F(X)*G(R):  1→SEC2-TAN2
152   1 → 5        UNCLASSIFIED
153   2 → 3        ALGEB
154   2 → 4        F(X)*G(L) → F(X)*G(R):  1→SEC2-TAN2
155   2 → 5        F(X)*G(L) → F(X)*G(R):  1→SEC2-TAN2
                   F(X)+G(L) → F(X)+G(R):  SEC2→TAN2+1
156   3 → 4        F(X)*G(L) → F(X)*G(R):  1→SEC2-TAN2
157   3 → 5        F(X)*G(L) → F(X)*G(R):  1→SEC2-TAN2
                   F(X)+G(L) → F(X)+G(R):  SEC2→TAN2+1
158   4 → 5        F(X)+G(L) → F(X)+G(R):  SEC2→TAN2+1


 15.  EXP-(1): TAN/(1-COT)+COT/(1-TAN)
      EXP-(2): (SIN/COS)/(1-COS/SIN)+(COS/SIN)/(1-SIN/COS)
      EXP-(3): SIN↑2/(COS*(SIN-COS))+COS↑2/(SIN*(COS-SIN))
      EXP-(4): (SIN↑3-COS↑3)/(COS*SIN*(SIN-COS))
      EXP-(5): (SIN-COS)*(SIN↑2+COS↑2+SIN*COS)/((SIN-COS)*SIN*COS)
      EXP-(6): (SIN↑2+COS↑2)/(SIN*COS)+(SIN*COS)/(SIN*COS)
      EXP-(7): 1/(COS*SIN)+1
      EXP-(8): SEC*CSC+1


159   1 → 2        G(L) → G(R):  TAN→SIN/COS
                   G(L) → G(R):  COT→COS/SIN
                   G(L) → G(R):  COT→COS/SIN
                   G(L) → G(R):  TAN→SIN/COS
160   1 → 3        NON-SINGLETON CSET
                   NON-SINGLETON CSET
161   1 → 4        NON-SINGLETON CSET
162   1 → 5        NON-SINGLETON CSET
163   1 → 6        NON-SINGLETON CSET
164   1 → 7        NON-SINGLETON CSET
165   1 → 8        NON-SINGLETON CSET
166   2 → 3        ALGEB
167   2 → 4        ALGEB
168   2 → 5        ALGEB
169   2 → 6        ALGEB
170   2 → 7        UNCLASSIFIED
171   2 → 8        NON-SINGLETON CSET
172   3 → 4        ALGEB
173   3 → 5        ALGEB
174   3 → 6        ALGEB
175   3 → 7        UNCLASSIFIED
176   3 → 8        NON-SINGLETON CSET
177   4 → 5        ALGEB
178   4 → 6        ALGEB
179   4 → 7        UNCLASSIFIED
180   4 → 8        NON-SINGLETON CSET
181   5 → 6        ALGEB
182   5 → 7        UNCLASSIFIED
183   5 → 8        NON-SINGLETON CSET
184   6 → 7        G(L) → G(R):  SIN2+COS2→1
185   6 → 8        G(L) → G(R):  SIN2+COS2→1
                   G(L) → G(R):  1/SIN→CSC
                   G(L) → G(R):  1/COS→SEC
186   7 → 8        G(L) → G(R):  1/COS→SEC
                   G(L) → G(R):  1/SIN→CSC


 16.  EXP-(1): (1+COT-CSC)*(1+TAN+SEC)
      EXP-(2): (1+COS/SIN-1/SIN)*(1+SIN/COS+1/COS)
      EXP-(3): ((SIN+COS-1)/SIN)*((COS+SIN+1)/COS)
      EXP-(4): ((SIN+COS)↑2-1)/(SIN*COS)
      EXP-(5): (SIN↑2+COS↑2+2*SIN*COS-1)/(SIN*COS)
      EXP-(6): (1+2*SIN*COS-1)/(SIN*COS)
      EXP-(7): 2
```

Table C.4 Continued

```
187    1 → 2        G(L) → G(R):  COT→COS/SIN
                    G(L) → G(R):  TAN→SIN/COS
                    G(L) → G(R):  CSC→1/SIN
                    G(L) → G(R):  SEC→1/COS
188    1 → 3        NON-SINGLETON CSET
                    NON-SINGLETON CSET
189    1 → 4        NON-SINGLETON CSET
190    1 → 5        NON-SINGLETON CSET
191    1 → 6        NON-SINGLETON CSET
192    1 → 7        NON-SINGLETON CSET
193    2 → 3        ALGEB
194    2 → 4        ALGER
195    2 → 5        ALGEB
196    2 → 6        UNCLASSIFIED
197    2 → 7        UNCLASSIFIED
198    3 → 4        ALGER
199    3 → 5        ALGER
200    3 → 6        UNCLASSIFIED
201    3 → 7        UNCLASSIFIED
202    4 → 5        ALGEB
203    4 → 6        F(X)+G(L) → F(X)+G(R):  SIN2+COS2→1
204    4 → 7        UNCLASSIFIED
205    5 → 6        G(L) → G(R):  SIN2+COS2→1
206    5 → 7        UNCLASSIFIED
207    6 → 7        ALGEB
```

Note: RCP stands for 'reciprocal of'

Table C.4 Continued

APPENDIX D

LISTING OF SUPER-2

```
 NOLIST
 PROGRAM SUPER2(INPUT,OUTPUT,TAPE1=INPUT)
 COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
XBPI,BPJ,INDEX
 COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
 COMMON/TAB/H(14),TRGTAB(6),VTAB(26),TABA(6,10),TABB(6,10)
 COMMON/PRNT/LIST(10),KAD(80),IPAK,L1,L2
 COMMON/BREG/IE(20),JE(20)
 COMMON/AUG20/ARG(20),IRG                                  ....(#1)....
 COMMON/TT/T1,T2,T3,T4,T5,T6,V1,V2
 COMMON/XV/T(6),V(32),LQ(12),JFG(12),D(36),R(36)
 COMMON/NOV/KS(25),JA(2,10),NSCH(2,3),M(2),JP(2),KP1,KP2
 COMMON NX,JPAK,MONO,ISCH,ERR
 DIMENSION U(6,10)
 EQUIVALENCE (U,TABB)
 COMMON/F/IH,JH,KH,LH,MH,NH
     COMMON/FEB71/NT
 DIMENSION TX(4),NY(4)
 INTEGER TPI,TPJ,BPI,BPJ
 INTEGER H,TRGTAB
 LOGICAL JPAK,MONO
 DATA IN/103B,406B,711B,1214B,1517B,2026B,2735B,3644B,4547B,
X 5355B,5052B,5660B /
 DATA H,TRGTAB,VTAB   / 6*7,1,1,2,3,3,4,5,5,3RSIN,3RCOS,3RTAN,
A3RCSC,3RSEC,3RCOT,   12.323,  2.3128,  7.934,  54.977,  .3142,
B 3.882,  14.8542,  3.6603,  71.9726,  3.1212,  2.3042,  8.9234,
C 12.8364,  -2.423,  1.1923,  37.546,  -49732,  4.336,  9.57238,
D 24.327,  -6.1132,  0.7632,  -11.637,  7.116,  21.124,  33.505  /
 DATA TABB        /
E 3.01742,  2.7116,  4.92711,  7.00177,  15.33170,  11.03037,
F 14.32341,  -1.63124,  2.31367, -0.03725,  .74176,  4.74231,
G  1.23456,  2.34567,  3.45678,  4.56789,  5.67890,  6.78901,
H 12.21341,  4.36724,  -1.37113,  2.16167,  9.46032,  0.42374,
I 0.62345,  1.37632,  -39.04728, 12.14176,  23.19426,  2.37416,
J 11.21304,  2.17003,  -2.14567,  -8.44026,  -6.03372,  4.20366,
K 23.12345,  19.80341,  -2.14156,  3.11224,  8.30064,  -2.92974,
L 1.3602,  37.1232,  2.9674,  1.3374,  14.1046,  2.1628,
M 2.3032,  3.4042,  -4.5052,  5.6062,  6.7072,  7.8082,
N 8.3366,  7.3366,  6.3366,  5.3366,  4.3366,  3.3366  /
 DATA JFG     / 0102010444B, 0304051022B, 0506111411B, 0710152070B,
X 1112212461B, 1320253060B, 2126313412B, 2734354005B,
Y 3536414152B,  3740424225B, 4142434316B, 4344444407B  /

 LQ(1)=0102030705068  $  LQ(2)=010203041006B  $ LQ(3)=010203040511B
 LQ(4)=10212040506B  $  LQ(5)=102030405138  $  LQ(6)=141503040506B
 LQ(7)= 10230043106B    $    LQ(8)=010203250527B
  LQ(9)= 140216042006B    $    LQ(10)= 011503170521B
 LQ(11)=010216172006B   $   LQ(12)=010203172021B
 X1=1.24   $   X2=0.806   $   X3=0.412
 V(1)=3.01742976   $   V(2)=2.711166337   $   V(3)=4.92711603
 V(4)=7.00177521   $   V(5)=15.331701   $   V(6)=11.0303791
 V(7)=1/V(1)  $  V(8)=1/V(2)   $   V(9)=1/V(3)  $   V(10)=V(1)/V(2)
 V(11)=V(2)/V(1)   $   V(12)=SIN(X1)   $   V(13)=COS(X1)
 V(14)=TAN(X1)   $   V(15)=1/V(12)  $  V(16)=1/V(13)
 V(17)=1/V(14)   $   V(18)=SIN(X2)  $  V(19)=COS(X2)
 V(20)=TAN(X2)  $  V(21)=1/V(18)   $   V(22)=1/V(19)
 V(23)=1/V(20)    $     V(24)=TAN(X3)   $   V(25)=1/COS(X3)
 V(26)=V(13)/V(1)   $   V(27)=V(2)/V(17)   $   V(28)=V(2)*V(14)
 V(29)=1/V(28)  $  V(30)=V(13)/V(23)  $  V(31)=V(12)/V(2)
 V(32)=V(12)/V(20)
 ERR=10E-8    $    ISCH=333223311322211211B

*************   INITIALISE  TABLE  OF DICREPANCY  VALUES
```

```
C     CREATE DICREPANCY TABLES (D( ) FOR DIFFERENCES) AND R FOR RATIOS
      T1=12.3  $  T2=3.4  $  T3=.457  $  T4=5.6  $  T5=31.5  $  T6=23.4
C     REVERSE NEXT 2 CARDS TO REVERSE TRACING STATUS
      CALL TIME(1,Y)
      IH=2
      IH=0
      JH=3
      KH=5                                              ....(#2)....
       IH=JH=KH=LH=MH=NH=0
C     CSC=1/SIN - D(1)...D(4),R(1),R(2)
      D(2)=ABS(T4*T1-1.0)    $    D(1)=D(2)/T1   $   D(3)=D(2)/T4
      D(4)=D(1)/T4   $   R(1)=T4*T1   $   R(2)=1/R(1)
C     SEC=1/COS   -   D(5) ... D(8),R(3),R(4)
      D(6)=ABS(T5*T2-1.)   $   D(5)=D(6)/T2   $   D(7)=D(6)/T5
      D(8)=D(5)/T5   $   R(3)=T5*T2   $   R(4)=1/R(3)
C     COT=1/TAN  -   D(9) .. D(12),R(5),R(6)
      D(10)=ABS(T6*T3-1)   $   D(9)=D(10)/T3   $   D(11)=D(10)/T6
      D(12)=D(9)/T6   $   R(5)=T6*T3   $   R(6)=1/R(5)
C     TAN=SIN/COS  -  D(13)...D(16) , R(7),R(8)
      D(14)=ABS(T3*T2-T1)   $   D(13)=D(14)/T2   $   D(15)=D(13)/(T1*T3)
      D(16)=D(15)/T2   $   R(7)=T3*T2/T1   $   R(8)=1/R(7)
C     COT=COS/SIN   -   D(17),...D(20),R(9),R(10)
      D(18)=ABS(T6*T1-T2)   $   D(17)=D(18)/T1   $   D(19)=D(18)/T6/T2
      D(20)=D(19)/T1   $   R(9)=T6*T1/T2   $   R(10)=1/R(9)
C     SIN*2+COS*2=1   D(21) .. D(24),R(11),R(12)...R(16)
      D(21)=T1**2+T2**2-1    $     R(11)=T1**2+T2**2   $   R(12)=1/R(11)
      D(22)=ABS(1/(T1*T1+T2*T2)-1)   $  D(23)=ABS(1/(T1*T1)-1/(1-T2*T2))
      D(24)=ABS(1/T2**2-1/(1-T1**2))
      R(13)=ABS(T1**2/(1-T2**2))   $   R(14)=1/R(13)
      R(15)=ABS(T2**2/(1-T1**2))   $   R(16)=1/R(15)
C     SEC*2=TAN*2+1   -   D(25), .. D(28),R(17)..R(22)
      D(25)=ABS(T5**2-T3**2-1)   $   R(17)=ABS(T5**2-T3**2)
      D(26)=ABS(1/T5**2-1/(1+T3**2))  $  D(27)=ABS(1/T3**2-1/(T5**2-1))
      D(28)=ABS(1/(T5**2-T3**2)-1)
      R(18)=1/R(17)   $   R(19)=ABS(T5**2/(T3**2+1))  $  R(20)=1/R(19)
      R(21)=ABS((T5**2-1)/T3**2)   $   R(22)=1/R(21)
C     CSC*2=COT*2+1  -  D(29), .. D(32),R(23),... R(28)
      D(29)=ABS(T4**2-T6**2-1)    $   R(23)=ABS(T4**2-T6**2) -
      D(30)=ABS(1/T4**2-1/(1+T6**2))  $  D(31)=ABS(1/T6**2-1/(T4**2-1))
      D(32)=ABS(1-1/(T4**2-T6**2))
      R(24)=1/R(23)   $   R(25)=ABS(T4**2/(1+T6**2))   $   R(26)=1/R(25)
      R(27)=ABS((T4**2-1)/T6**2)   $   R(28)=1/R(27)
C     TAN=SIN*SEC    D(33),R(29),R(30)
      D(33)=ABS(T3-T1*T5)   $   R(29)=ABS(T3/(T1*T5))  $  R(30)=1/R(29)
C     COT=COS*CSC    D(34),R(31),R(32)
      D(34)=ABS(T6-T2*T4)   $   R(31)=ABS(T2*T4/T6)   $   R(32)=1/R(31)
C     TAN=SEC/CSC        D(35),R(33),R(34)
      D(35)=ABS(T3-T5/T4)   $   R(33)=ABS(T3*T4/T5)  $  R(34)=1/R(33)
C     COT=CSC/SEC      D(36),R(35),R(36)
      D(36)=ABS(T6-T4/T5)   $   R(35)=ABS(T6*T5/T4)  $  R(36)=1/R(35)
*
      TA1=TB1=TA2=TB2=0
      NPRB=0

C     FILL TABLE TABA( , )
      DO 10 I=1,10
      AI=I
      TABA(1,I)=SIN(AI)   $   TABA(2,I)=COS(AI)   $   TABA(3,I)=TAN(AI)
      TABA(4,I)=1/TABA(1,I)
      TABA(5,I)=1/TABA(2,I)    $    TABA(6,I)=1/TABA(3,I)
   10 CONTINUE
C     INITIALISE LIST AND POINTERS
      TPI=TPJ=1
      HPI=511
      DO 12 I=1,510
   12 ICELL(I)=I+1
      PRINT 15
      N=0
   15 FORMAT(1H1)
      CALL TIME(0,Y)
C
C     GET A NEW PROOF - INITIALISE TO BEGINNING
C
   20 CALL GNP(II)                                   ....(#3)....
      IF(II.EQ.0)GOTO 21
      PRINT 17,TA1,TB1,TA2,TB2
   17 FORMAT(
     X////*1*,//10X,*TOTAL  TIMES:*,2F10.3,/,10X,*NON-AL TIMES:*,2F10.3)
      STOP 1
   21 NPRB=NPRB+1
      TX(1)=TX(2)=TX(3)=TX(4)=0
      CALL TIME(0,Y)
      PRINT 147,NPRB
  147 FORMAT(// /,1H+,I3,*.*)
      IEP=1
      IRG=0
      LIST(1)=10H EXP-(0):
   22 LIST(1)=LIST(1)+1000000B
           L2=10    $     L1=2     $     IPAK=0
      CALL GETEXP(II)
      CALL PRNT(0)
      IF(II.NE.0)GO TO 25
      CALL POLISH
      IE(IEP)=LAST     $     IEP=IEP+1
      ICELL(LAST)=ICELL(LAST).A.77777777777777777000B
      IF(NX.EQ.0)GO TO 26
      IF((NX.EQ.1R=).O.(NX.EQ.1R:).O.(NX.EQ.1R#))GO TO 22
      IF(NX.NE.1R>)GO TO 22
C     ABOVE STATEMENTS ARE TEMPORARY

C     PRINT OUT STUFF
      JJ=IEP-1    $    DO 40 I=1,JJ    $    KK=IE(I)
   40 CONTINUE
      CALL TIME(0,Y)
      PRINT 88,Y
   88 FORMAT(/7X,*COMPILATION TIME: *,F9.3,* SEC.*)
      PRINT 90
   90 FORMAT(//7X,*STEP*,5X,*N-COMP*,5X,*TIME-A*, 8X,*TIME-B*,2X,5X,
     X*RUN-1*,5X,*RUN-2*,5X,*RUN-3*/)
C  90 FORMAT(//,7X,*STEP*,15X,*RUN-1*,15X,*RUN-2*,15X,*RUN-3*,15X,*RUN-4
C    X*/)
      NST=IEP-1    $    NSTT=NST-1
      IF(NST.LT.2)STOP 777
      DO 1001 IMI=1,NSTT
      NSS=IMI+1
      DO 1000 JMJ=NSS,NST
      LGE=0
              NT=0
      N=N+1
      IH=JH=KH=LH=MH=0
      TA1=TA1+TX(1)        $        TB1=TB1+TX(2)
      IF(NY(1).NE.10HALGEBRAIC  )TA2=TA2+TX(1)
      IF(NY(1).NE.10HALGEBRAIC  )TB2=TB2+TX(2)
C     EVALUATE STEP IMI → JMJ
        NH=0
      II=IE(IMI)    $    JJ=IE(JMJ)
      III=II     $     JJJ=JJ
   42 IF(NH.GT.3)GOTO 400
      NH=NH+1     $     II=III     $     JJ=JJJ
            NT=NT+1
      IF(ABS(ACELL(II)-ACELL(JJ)).GT.ERR)GO TO 65
            NT=NT+1
      IF(ABS(BCELL(II)-BCELL(JJ)).GT.ERR)GOTO 70
C     STEP ALGEBRAIC
      CALL TIME(0,Y)                                   ....(#4)....
            NAT=1
      TX(1)=TX(2)=TX(3)=TX(4)=Y    $    NY(1)=NY(2)=NY(3)=NY(4)=10HALGEBRA
     XIC     $     GO TO 400
      PRINT 60,N,IMI,JMJ,Y        $     GO TO 1000
   60 FORMAT(2I4,* →*,I2,4X,*ALGEBRAIC*  F9.3,    * SEC.*)
C     STEP INCORRECT
   65 CALL TIME(0,Y)    $    TX(1)=TX(2)=TX(3)=TX(4)=Y
      NY(1)=NY(2)=NY(3)=NY(4)=10HINCORRECT
  400 PRINT 67,N,IMI,JMJ,NAT,TX(1),TX(2),NY(1),NY(3),NY(4)
   67 FORMAT(2I4,* →*,I2,I8,4X,F6.3,* SEC.*,3X,F6.3,* SEC.*,3(5X,A5))
   66 FORMAT(6X,I2,* →*,I2,4X,*INCORRECT* ,  F9.3 ,* SEC.*)
      GOTO 1000
C     PREPARE TO CALL S/R SMALL
   70 KP1=1    $    KP2=2    $    CALL COPY(II)    $    CALL COPY(JJ)
      KS(1)=II+LSHFT(JJ,9)
   75 CALL SMALL(KP1,JT)
            IF(NH.EQ.1)NAT=NT
      IF(JT.EQ.0)GO TO 80
C     STEP LARGE
      LGE=LGE+1
   82 KP1=KP1+1
         IF(KP1.EQ.KP2)GO TO 83
      JS=KS(KP1)      $      IS=JS.A.777B      $      JS=LSHFT(JS,-9).A.777B
      IF(ABS(BCELL(IS)-BCELL(JS)).GT.ERR)GOTO 75
      PRINT 822      $      GO TO 82
  822 FORMAT(10X,*ALGEBRAIC*)
   83 IF(LGE.NE.0)74,84
   73 KP1=KP1-1       $       IF(KP1.EQ.KP2)GOTO 74
      JJ=KS(KP1)    $    II=JJ.A.777B    $    JJ=LSHFT(JJ,-9).A.777B
      CALL RELEASE(II)    $    CALL RELEASE(JJ)    $    GO TO 73
   74 CALL TIME(0,Y)  $  NY(NH)=10HLARGE        $    TX(NH)=Y    $  GOTO42
   76 FORMAT(6X,I2,* →*,I2,4X,*LARGE* , F13.3,* SEC.*)
      GO TO 1000
  303 FORMAT(2I4,* →*,I2,8X,A5)
C
C
   80 KP1=KP1+1    $    IF(KP1.LT.KP2)GO TO 75
      CALL TIME(0,Y)
   84 CONTINUE
       TX(NH)=Y  $  NY(NH)=10HSMALL        $    GOTO 42
      PRINT 85,IMI,JMJ,Y    $                GO TO 1000
   85 FORMAT(6X,I2,* →*,I2,4X,*SMALL*,4X,F9.3, * SEC.*)
 1000 CONTINUE
      CALL RELEASE(IE(IMI))
 1001 CONTINUE
      CALL RELEASE(IE(JMJ))
      GO TO 20
   25 CONTINUE
   26 PRINT 27
   27 FORMAT( 5X,*SYNTAX LOUSY*)
      GO TO 20
      END

      SUBROUTINE SIZE(I,J)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/NOV/KS(25),JA(2,10),NSCH(2,3),K(2),JP(2),KP1,KP2
      COMMON/XV/T(6),V(32),LQ(12),JFG(12),D(36),R(36)
      COMMON/TT/T1,T2,T3,T4,T5,T6,V1,V2
      COMMON NX,JPAK,MONO,ISCH,ERR
      COMMON/F/IH,JH,KH,LH,MH,NH
C     NOTE ARRAY K(2) INSTEAD OF M(2) IN VIEW OF OTHER USE OF M
C     EVALUATE SIZE OF STEP-I IN KS-TABLE RETURNING J=0 IF SIZE SMALL
      I1=KS(I).A.777B    $     I2=LSHFT(KS(I),-9).A.777B
      III=I1      $       JJJ=I2
C     GET STEP CONFIG
C     AT SOME LATER STAGE REMOVE CONFIG TEST FOR COMPARISON
      J=KFIG(I1).O.KFIG(I2)
      DO 10 M=1,12
C     TEST IF MIN. CONFIG FOR IDENTITY-M SATISFIED ....(#5)....
      IF((J.A.JFG(M)).NE.(JFG(M).A.77B))GO TO 10
      CALL SETT(M)
      V1=XVAL(I1)     $      IF(I2.NE.0)V2=XVAL(I2)
      IF(ABS(V1-V2).LT.ERR)GO TO 15
   10 CONTINUE
   11 CALL RELEASE(III)              $       CALL RELEASE(JJJ)
      RETURN
C     ASSUME J NONZERO
C     IDENTITY-M SATISFIES STEP
   15 T(1)=T1  $  T(2)=T2  $  T(3)=T3  $  T(4)=T4  $  T(5)=T5 $  T(6)=T6
      V1=XVAL(I1)     $      IF(I2.NE.0)V2=XVAL(I2) $   ADF=ABS(V1-V2)
      N=NPWR(III,JJJ)        $       IF(N.GT.50)GOTO 18
      X2=1.0/N            $      X1=ABS(V1)**X2      $       X2=ABS(V2)**X2
      IF(V1.LT.0)X1=-X1          $             IF(V2.LT.0)X2=-X2
   18 CONTINUE
C     TEST DIFFERENCE DISCREPANCIES
      I1=LSHFT(JFG(M),-12).A.77B
      I2=LSHFT(JFG(M),-6).A.77B
   20 IF(NH.EQ.4)GOTO 26
      IF(ABS(ADF-D(I1)).LT.10E-8)GOTO 24
      IF(N.GT.50)GOTO 22
      IF(ABS(ABS(X1-X2)-D(I1)).LT.ERR)GOTO 24
   22 CONTINUE
      IF(I1.EQ.I2)GOTO 26  $  I1=I1+1
      GO TO 20
C     STEP  SMALL
   24 J=0                    $      GO TO 11
C     TRY RATIO DISCREPANCIES
C     REMEMBER TO REMOVE IT SOME TIME FOR COMPARISON
   26 IF(NH.EQ.3)GOTO 11
      IF(ABS(V2).LT.10E-15)GOTO11
      ADF=ABS(V1/V2)
      I1=LSHFT(JFG(M),-24).A.77B    $    I2=LSHFT(JFG(M),-18).A.77B
   28 IF(ABS(ADF-R(I1)).LT.ERR)GOTO 24
      IF(N.GT.50)GOTO 29
      IF(ABS(ABS(X1/X2)-R(I1)).LT.ERR)GOTO 24
   29 CONTINUE
      IF(I1.EQ.I2)GO TO 11    $    I1=I1+1
      GO TO 28
      END
        FUNCTION BC(I)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
      COMMON/XV/T(6),V(32),LQ(12),JFG(12),D(36),R(36)
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER RL,DA,FB,DF,FR
```

```
      IF(NH.NE.1)RETURN
      I2=1     $     CALL WFF(I,J,K)
    5 I1=FB(K)    $    IF(I1.EQ.2)GOTO 20    $    IF(I1.EQ.0)GOTO 12
      PRINT 10        $ BC=1000        $        RETURN
   10 FORMAT(5X,*FN  BC  - VARIABLES NOT ALLOWED*)
   12 STK(I2)=ACELL(K)
   13 I2=I2+1
   14 IF(K.EQ.I)GOTO 15    $    K=LL(K)     $     IF(K.NE.0)GOTO 5
   15 CONTINUE
        BC=STK(1)     $     RETURN                          ....(#6)....
   20 I1=DA(K)
      GOTO(21,21,21,21,21,21,23,24,25,26,27,28,29,29),I1
   21 STK(I2)=T(I1)    $    GOTO 13
   23 STK(I2-2)=STK(I2-2)+STK(I2-1)    $    GO TO 31
   24 STK(I2-2)=STK(I2-1)-STK(I2-2)     $     GO TO 31
   25 STK(I2-1)=-STK(I2-1)    $    GO TO 14
   26 STK(I2-2)=STK(I2-2)*STK(I2-1)    $     GO TO 31
   27 STK(I2-2)=STK(I2-1)/STK(I2-2)     $     GO TO 3 1
C     RECIPROCAL
   28 STK(I2-1)=1/STK(I2-1)      $     GO TO 14
   29 XVAL=STK(I2-1)    $    IF(XVAL.LT.0)GOTO 32  $  XVAL=XVAL**STK(I2-2)
   30 IF(I1.EQ.14)XVAL=1/XVAL     $     STK(I2-2)=XVAL
   31 I2=I2-1     $     GO TO 14
   32 J=STK(I2-2)      $     XVAL=(-XVAL)**J
      IF(J.NE.(2*(J/2)))XVAL=-XVAL
      GO TO 30     $     END
       FUNCTION AC(I)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
      COMMON/XV/T(6),V(32),LQ(12),JFG(12),D(36),R(36)
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER RL,DA,FB,DF,FR
      IF(NH.NE.1)RETURN
      I2=1     $     CALL WFF(I,J,K)
    5 I1=FB(K)    $    IF(I1.EQ.2)GOTO 20    $    IF(I1.EQ.0)GOTO 12
      PRINT 10        $ AC=1000        $        RETURN
   10 FORMAT(5X,*FN  AC  - VARIABLES NOT ALLOWED*)
   12 STK(I2)=ACELL(K)
   13 I2=I2+1
   14 IF(K.EQ.I)GOTO 15    $    K=LL(K)     $     IF(K.NE.0)GOTO 5
   15 CONTINUE
        AC=STK(1)     $     RETURN
   20 I1=DA(K)
      GOTO(21,21,21,22,22,22,23,24,25,26,27,28,29,29),I1
   21  STK(I2)=SIN(.856324)    $    GO TO 13
   22 STK(I2)=TAN(1.8356479)     $     GO TO 13
   23 STK(I2-2)=STK(I2-2)+STK(I2-1)    $    GO TO 31
   24 STK(I2-2)=STK(I2-1)-STK(I2-2)     $     GO TO 31
   25 STK(I2-1)=-STK(I2-1)    $    GO TO 14
   26 STK(I2-2)=STK(I2-2)*STK(I2-1)    $     GO TO 31
   27 STK(I2-2)=STK(I2-1)/STK(I2-2)     $     GO TO 31
C     RECIPROCAL
   28 STK(I2-1)=1/STK(I2-1)      $     GO TO 14
   29 XVAL=STK(I2-1)    $    IF(XVAL.LT.0)GOTO 32  $  XVAL=XVAL**STK(I2-2)
   30 IF(I1.EQ.14)XVAL=1/XVAL     $     STK(I2-2)=XVAL
   31 I2=I2-1     $     GO TO 14
   32 J=STK(I2-2)      $     XVAL=(-XVAL)**J
      IF(J.NE.(2*(J/2)))XVAL=-XVAL
      GO TO 30     $     END
      SUBROUTINE GNP(II)
      COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
      COMMON/PRNT/LIST(10),KAD(80),IPAK,L1,L2
      COMMON NX,JPAK,MONO
      LOGICAL JPAK,MONO
      INTEGER PP
C *** POSITION AT BEGINNING OF PROOF AND ALSO SEE IF MONO-ARGUMENT PROB
C *** GNP SKIPS TILL A CARD BEGINNING WITH < AS 1ST NON-BLANK
      JPAK=.F.
      II=2
    1 READ 2,KAD
    2 FORMAT(80R1)
      IF(EOF,1)30,3                                   ....(#7)....
    3 IK=1                        $    NX=PP(X)
      IF(NX.EQ.1R<)10,4
    4 PRINT 5,KAD
    5 FORMAT(*  IGNORED*2X,80R1)
      GO TO 1
   10 II=0
      NX=PP(X)       $      IF(NX.EQ.1R$)11,14
   11 MONO=.F.        $       RETURN
   14 IK=IK-1        $      MONO=.T.
   30 RETURN
      END

      SUBROUTINE GETEXP(II)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
      COMMON/TAB/H(14),TRGTAB(6),VTAB(26),TABA(6,10),TABB(6,10)
      COMMON NX,JPAK,MONO
      INTEGER PP
      INTEGER H,TRGTAB
      INTEGER TPI,TPJ,BPI,BPJ
      LOGICAL JPAK,MONO
C     PACK CURRENT CHAR FOR LISTING IF JPAK IS .T.
      JPAK=.F.       $      NPAR=MPAR=II=0
      IJ=1      $      NX=PP(X)       $      JPAK=.T.      $      GO TO 3
C     EXPECT VERY 1ST TO BE A + OR - OR ( OR OPERAND
    2 NX=PP(X)
CCCCCCCCCC  FOLLOWING 3 CARDS DEC 70 MODS
    3 IF(NX.EQ.1R+)GO TO 4
      IF(NX.NE.1R-)GO TO 6
      IMP(IJ)=NX    $    IJ=IJ+1
    4 NX=PP(X)
C     HERE WE EXPECT ONLY ( OR A OPERAND
    6 IF(NX.NE.1R()GO TO 8
      IMP(IJ)=NX          $         IJ=IJ+1
      NPAR=NPAR+1       $       GO TO 2
    8 IF(NX.LT.45B)GO TO 15
C     ERROR
   10 II=1       $       RETURN
   15 JSAV=NX       $       KOUNT=2       $       IF(NX.GT.32B)GO TO 35
C     ALPHABETIC
   17 NX=PP(X)
      IF(NX.GT.32B)GO TO 20
      JSAV=LSHFT(JSAV,6).OR.NX        $        KOUNT=KOUNT-1
      IF(KOUNT.LT.0)10,17
C     CHECK IF A,B, ... X,Y,Z  OR  SIN,COS, ...
   20 IF(KOUNT.EQ.2)GO TO 25
      IF(KOUNT.NE.0)GO TO 10
C     CHECK LEGAL TRIG FUNCTION
      DO 22 I=1,6
      IF(JSAV.EQ.TRGTAB(I))GO TO 30
   22 CONTINUE
      GO TO 10
```

```
   25 IMP(IJ)=JSAV      $      GO TO 44
   30 IMP(IJ)=JSAV    $    IJ=IJ+1    $    IF(MONO)45,60
C     NUMERIC STRING
   35 JSAV=NX-27
   36 NX=PP(X)
      IF((NX.LT.45B).A.(NX.GT.32B))37,40
   37 JSAV=10*JSAV+NX-27        $      GO TO 36
   40 IF(JSAV.GT.511)GO TO 10
      IMP(IJ)=JSAV+100 000 000
C     100 MILLION AS ADDEND                                ....(#8)....
   44 IJ=IJ+1
   45 IMP(IJ)=NX       $       IJ=IJ+1
C     EXPECTING + - * / ↑ = EOM ...
   46 IF((NX.GT.44B).A.(NX.LT.51B))GO TO 4
      IF(NX.NE.1R))GO TO 50
      NPAR=NPAR-1    $    IF(NPAR.LT.0)GO TO 10
      NX=PP(X)      $     GO TO 45
   50 IF(NX.EQ.1R↑)GO TO 55
C     TEST FOR EOM,:,$ ETC
      JPAK=.F.       $       IJ=IJ-1
      IF((NX.NE.0).A.(NX.NE.1R=).A.(NX.NE.1R$).A.(NX.NE.1R:).A.(NX.NE.1R
     X>))GO TO 10
      IF(NPAR.EQ.0)RETURN
      GO TO 10
C     ...  ↑  ...
   55 NX=PP(X)
      IF(NX.NE.1R+)GO TO 57
      NX=PP(X)       $      GO TO 59
   57 IF(NX.NE.1R-)GO TO 59
      IMP(IJ-1)=1R+      $      NX=PP(X)
   59 IF((NX.LT.33B).O.(NX.GT.44B))GO TO 10
      IMP(IJ)=NX-27+100 000 000       $       IJ=IJ+1
      NX=PP(X)       $      GO TO 45
C     ANALYSING THE ARGUMENT SECTION OF A TRIG. FUNCTION
   60 IF(NX.NE.1R()GO TO 10
      MPAR=1       $       IMP(IJ)=NX       $       IJ=IJ+1
   62 NX=IMP(IJ)=PP(X)       $       IJ=IJ+1
      IF((NX.NE.1R-).A.(NX.NE.1R+))GO TO 65
   64 IMP(IJ)=NX=PP(X)       $       IJ=IJ+1
   65 IF(NX.NE.1R()GO TO 70
      MPAR=MPAR+1       $       GO TO 62
   70 IF(NX.GT.32B)GO TO 75
C     EXPECTING  + - * / OR )
   71 NX=IMP(IJ)=PP(X)       $       IJ=IJ+1
   72 IF(NX.NE.1R))GO TO 73
      MPAR=MPAR-1       $       IMP(IJ)=NX=PP(X)       $       IJ=IJ+1
      IF(MPAR.EQ.0)46,72
   73 IF((NX.LT.45B).OR.(NX.GT.50B))10,64
C     NUMERIC STRING
   75 IF(NX.GT.44B)GO TO 10
      JSAV=NX-27
   76 NX=PP(X)
      IF((NX.GT.32B).A.(NX.LT.45B))78,80
   78 JSAV=JSAV*10+NX-27        $        GO TO 76
   80 IF(JSAV.GT.511)GO TO 10
C *** IT MAY PROVE TO BE A STOOPID AND UNNECESSARY LIMIT -- 511 I MEAN
      IMP(IJ-1)=JSAV+ 100 000 000    $    IMP(IJ)=NX    $    IJ=IJ+1
      GO TO 72
      END

      SUBROUTINE POLISH
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
```

```
      COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
      COMMON/TAB/H(14),TRGTAB(6),VTAB(26),TABA(6,10),TABB(6,10)
      COMMON/AUG20/ARG(20),IRG
      INTEGER TPI,TPJ,BPI,BPJ
      COMMON NX,JPAK,MONO
      INTEGER H,TRGTAB
      LOGICAL JPAK,MONO
C     THIS S/R CONVERTS EXPRESSION (MODIFIED) IN IMP(1) ... IMP(IJ-1)
C     TO PREFIX POLISH IN THE FORM OF A LIST OF CELL-PAIRS .. ICELL()
C     AND ACELL() WITH ACELL() CONTAINING THE MAX. CONSISTENCY VALUE
C ***  COMMON ETC
      LAST=0      $      I1=I2=1      $      JJ=IJ-1
      INDEX=0
      DO 100 I=1,JJ
      IT=IMP(IJ-I)                                     ....(#9)....
C     TEST IF NUMBER
      IF(IT.LT.1000000)GO TO 10
      RTK(I2)=STK(I2)=IT-100 000 000
      CALL ADLST(0)      $      GO TO 100
C     CHECK FOR VARIABLE A,B,... X,Y,Z
   10 IF(IT.GT.26)GO TO 20
      RTK(I2)=STK(I2)=VTAB(IT)
      CALL ADLST(IT+100B)      $      GO TO 100
C     TEST FOR )  OR  (
   20 IF(IT.GT.72B)GO TO 70
      IF(IT.NE.1R))GO TO 25
      ISTK(I1)=IT      $      I1=I1+1      $      GO TO 100
   25 IF(IT.NE.1R()GO TO 30
   26 I1=I1-1      $      IT=ISTK(I1)
      IF(IT.EQ.1R))GO TO 100
      CALL OP(IT)      $      GO TO 26
C     TEST FOR +,-,*,/,↑,↓
   30 IF(IT.NE.1R+)GO TO 40
      IF(I.EQ.JJ)GO TO 100      $      IR=IMP(IJ-I-1)
C     IGNORE UNARY +
      IF((IR.EQ.1R().OR.(IR.EQ.1R↑))GO TO 100
      IR=7
C     CODE FOR + IS 7
   35 IF(I1.EQ.1)GO TO 37
      JK=ISTK(I1-1)
      IF(JK.EQ.1R))GO TO 37
C     COMPARE HIERARCHIES
      IF(H(IR).GE.H(JK))GO TO 37
      I1=I1-1      $      CALL OP(JK)      $      GO TO 35
   37 ISTK(I1)=IR      $      I1=I1+1      $      GO TO 100
C     ....
   40 IF(IT.NE.1R-)GO TO 50
      IF(I.NE.JJ)GO TO 42
   41 IR=9      $      GO TO 35
   42 IR=IMP(IJ-I-1)
      IF(IR.EQ.1R()GO TO 41
      IF(IR.NE.1R↑)GO TO 44
      IMP(IJ-I-1)=1R↓
      GO TO 100
   44 IR=8   $   GO TO 35
C     ...  *  ...
   50 IF(IT.NE.1R*)GO TO 55
      IR=10      $      GO TO 35
C     ...  /  ...
   55 IF(IT.NE.1R/)GO TO 60
      IR=11      $      GO TO 35
C     ...  ↑  ...
   60 IF(IT.NE.1R↑)GO TO 65
```

```
      IR=13      $      GO TO 35
C     ...  ↓  ...
   65 IF(IT.NE.1R↓)GO TO 67
      IR=14      $      GO TO 35
C     ***  SHOULD NOT BE NECESSARY IN A WORKING VERSION  ***
   67 PRINT 68
   68 FORMAT(/*  ... IMPOSSIBLE AT POLISH 68 .. */)
      RETURN
C     SIN,COS,TAN, ...
   70 CONTINUE                                             ....(#10)....
      IF(IT.NE.3RSIN)GO TO 75
      IF(MONO)GO TO 73          $    I2=I2-1
      STK(I2)=SIN(STK(I2))
   72 CALL ADLST(201B)
      RTK(I2)=ATRG(1,STK(I2))          $    GO TO 100
   73 STK(I2)=TABA(1,1)
      RTK(I2)=TABB(1,1)            $   GO TO 72
   75 IF(IT.NE.3RCOS)GO TO 78
      IF(MONO)GO TO 77            $    I2=I2-1
      STK(I2)=COS(STK(I2))
      RTK(I2)=ATRG(2,STK(I2))
   76 CALL ADLST(202B)      $      GO TO 100
   77 STK(I2)=TABA(2,1)            $      RTK(I2)=TABB(2,1)
      GO TO 76
   78 IF(IT.NE.3RTAN)GO TO 84
      IF(MONO)GO TO 80              $      I2=I2-1
      STK(I2)=TAN(STK(I2))
      RTK(I2)=ATRG(3,STK(I2))
   79 CALL ADLST(203B)      $      GO TO 100
   80 STK(I2)=TABA(3,1)
      RTK(I2)=TABB(3,1)          $      GO TO 79
   84 IF(IT.NE.3RCSC)GO TO 88
      IF(MONO)GO TO 86            $      I2=I2-1
      STK(I2)=1/SIN(STK(I2))
      RTK(I2)=ATRG(4,STK(I2))
   85 CALL ADLST(204B)      $      GO TO 100
   86 STK(I2)=TABA(4,1)
      RTK(I2)=TABB(4,1)            $     GO TO 85
   88 IF(IT.NE.3RSEC)GO TO 92
      IF(MONO)GO TO 90            $    I2=I2-1
      STK(I2)=1/COS(STK(I2))
      RTK(I2)=ATRG(5,STK(I2))
   89 CALL ADLST(205B)      $      GO TO 100
   90 STK(I2)=TABA(5,1)
      RTK(I2)=TABB(5,1)      $      GO TO 89
   92 IF(IT.NE.3RCOT)GO TO 67
      IF(MONO)GOTO 94              $     I2=I2-1
      STK(I2)=1/TAN(STK(I2))
      RTK(I2)=ATRG(6,STK(I2))
   93 CALL ADLST(206B)      $      GO TO 100
   94 STK(I2)=TABA(6,1)
      RTK(I2)=TABB(6,1)                $      GO TO 93
  100 CONTINUE
C     NOW UNLOAD ANY REMAINING OPERATORS STILL IN ISTK( )
  102 IF(I1.EQ.1)GO TO 110
      I1=I1-1      $      IT=ISTK(I1)      $      CALL OP(IT)      $ GOTO 102
  110 RETURN
      END

      SUBROUTINE PAK
      COMMON/PRNT/LIST(10),KAD(80),IPAK,L1,L2
      COMMON NX,JPAK,MONO
C *** L1 IS LIST( ) POINTER, L2 10-CTR
```

```
      IPAK=LSHFT(IPAK,6).OR.NX
      L2=L2-1
      IF(L2.GT.0)RETURN
      LIST(L1)=IPAK            $    IPAK=0
      L2=10                    $    L1=L1+1
      IF(L1.LT.9)RETURN
      PRINT 1,(LIST(I),I=1,6)                            ....(#11)....
    1 FORMAT(6X,13A10)
      LIST(1)=10H              $    IPAK=0
      L1=2                     $    L2=10              $    RETURN
      END

      SUBROUTINE PRNT(I)
      COMMON/PRNT/LIST(10),KAD(80),IPAK,L1,L2

C *** LIST CURRENT EXPRESSION - DO NOT ADVANCE LINE IF I IS NONZERO
      IF(IPAK.EQ.0)GO TO 10
      LIST(L1)=LSHFT(IPAK,6*L2)
      IPAK=0                   $    L2=10
C *****   ABOVE CARD UNCERTAIN
      GO TO 11
   10 L1=L1-1
   11 IF(I.NE.0)15,12
   12 PRINT 13,(LIST(J),J=1,L1)
   13 FORMAT(6X,10A10)
      RETURN
   15 PRINT 16,(LIST(J),J=1,L1)
   16 FORMAT(1H+,5X,10A10)
      RETURN       $       END

      SUBROUTINE OP(M)
      COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
C *** INSERT COMMON , INTEGER ETC CARDS
      MM=M-6       $       I2=I2-2
      GOTO(10,20,30,40,50,55,60,60),MM
C     ...  +  ...
   10 STK(I2)=STK(I2+1)+STK(I2)
      RTK(I2)=RTK(I2+1)+RTK(I2)
   15 CALL ADLST(M+200B)
      RETURN
C     ...  -  ...
   20 STK(I2)=STK(I2+1)-STK(I2)
      RTK(I2)=RTK(I2+1)-RTK(I2)
      GO TO 15
C     ...  ¬  ...
   30 I2=I2+1      $     STK(I2)=-STK(I2)
      RTK(I2)=-RTK(I2)     $     GO TO 15
C     ...  *  ...
   40 STK(I2)=STK(I2+1)*STK(I2)
      RTK(I2)=RTK(I2+1)*RTK(I2)          $      GO TO 15
C     ...  /  ...
   50 STK(I2)=STK(I2+1)/STK(I2)
      RTK(I2)=RTK(I2+1)/RTK(I2)      $     GO TO 15
C      .... < .... RECIPROCAL OPERATOR
   55 I2=I2+1    $    STK(I2)=1/STK(I2)    $    RTK(I2)=1/RTK(I2)
      GO TO 15
C     ...  ↑  AND  ↓  ...
   60 IF(STK(I2+1).LT.0)GO TO 65
      STK(I2)=STK(I2+1)**STK(I2)
      IF(M.EQ.14)STK(I2)=1/STK(I2)       $      GO TO 70
   65 B=-STK(I2+1)       $       J=STK(I2)
      B=B**STK(I2)
      IF((J-2*(J/2)).NE.0)B=-B       $       STK(I2)=B
```

```
      IF(M.EQ.14)STK(I2)=1/B
   70 IF(RTK(I2+1).LT.0)GO TO 75
      RTK(I2)=RTK(I2+1)**RTK(I2)    $    IF(M.EQ.14)RTK(I2)=1/RTK(I2)
      GO TO 15
   75 B=-RTK(I2+1)    $    J=RTK(I2)    $    B=B**RTK(I2)
      IF((J-2*(J/2)).NE.0)B=-B    $    RTK(I2)=B
      IF(M.EQ.14)STK(I2)=1/B     $    GO TO 15
      END                                                  ....(#12)....

      SUBROUTINE ADLST(M)
      COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      INTEGER GIVE  ,RANK
C     CALLED BY S/R POLISH TO ADD ON A CELL-QUAD AS THE POLISH LIST
C     IS CREATED - BACKWARDS
      LNK=GIVE(I)      $      ACELL(LNK)=STK(I2)
      BCELL(LNK)=RTK(I2)    $    I2=I2+1
C     COMPUTE INDEX OF CURRENT CELL
      ICELL(LNK)=LSHFT(M,18) + LSHFT(LAST,9) + ICELL(LNK)
      INDEX=KCELL(LNK)=INDEX+RANK(LNK)
      LAST=LNK      $      RETURN      $      END


      INTEGER FUNCTION GIVE(X)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
       COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER TPI,TPJ,BPI,BPJ
C     GIVE A CELL QUADRUPLE
      GIVE=TPI      $      TPI=ICELL(TPI)      $      RETURN
      END
      SUBROUTINE SAVE(N)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
       COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER TPI,TPJ,BPI,BPJ
C     SAVE CELL-QUAD N
      BPI=ICELL(BPI)=N      $      RETURN
      END

      FUNCTION PP(X)
      COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
      COMMON/PRNT/LIST(10),KAD(80),IPAK,L1,L2
      COMMON NX,JPAK,MONO
      INTEGER PP
      LOGICAL JPAK,MONO
C *** PP RETURNS NEXT NON-BLANK CHAR. UNLESS EOF ENCOUNTERED
      IF(JPAK)CALL PAK
C      COMMON JPAK  ...   LOGICAL JPAK
      IF(IK.LT.81)GO TO 4
    1 READ 2,KAD
    2 FORMAT(80R1)
      IF(EOF,1)10,3
    3 IK=1
    4 IF(KAD(IK).EQ.1R )5,6
    5 IK=IK+1
      IF(IK.GT.80)1,4
    6 PP=KAD(IK)                $    IK=IK+1
      RETURN
   10 PP=0                      $    RETURN
      END
```

```
          IDENT    LSHFT
          ENTRY    LSHFT
LSHFT     DATA     0
          SA1      B1                  WORD
          SA2      B2                  SHIFT COUNT
          SB2      X2
          LX6      X1,B2
          EQ       LSHFT
          END                                      ....(#13)....

      INTEGER FUNCTION RANK(L)
      INTEGER RL,DA,FB,DF,FR
      COMMON NX,JPAK,MONO.ISCH,ERR
      LOGICAL MONO
C   23 NOV 1970
C     RETURN THE RANK OF ICELL(L)
      I=FB(L)     $     IF(I.LT.2)1,5
    1 RANK=-1        $       RETURN
    5 I=DA(L)
      IF(I.LT.7)GOTO9
      IF((I.EQ.9).OR.(I.EQ.12))7,8
    7 RANK=0     $     RETURN
    8 RANK=1     $     RETURN
    9 IF(MONO)10,7
   10 RANK=-1
      RETURN
      END
      SUBROUTINE COPY(L)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER RL,DA,FB,DF,FR
      INTEGER GIVE
C     COPY WFF-L AND RETURN ADDRESS OF DUPLICATED LIST IN L
      I=L      $      J=L=GIVE(X)
C     GET 1ST CELL OF DUPLICATE LIST
      LAST=0      $      IND=KCELL(I)
    5 ICELL(J)=LSHFT(ICELL(J),9).O.(ICELL(I).A.777 000 000B).OR.LAST
      ACELL(J)=ACELL(I)    $    BCELL(J)=BCELL(I)    $    KCELL(J)=KCELL(I)
      LLL=I
      LAST=J    $    I=RL(I)
      IF(I.EQ.0)GO TO 6
      IF(KCELL(I).GT.IND)PRINT 455,I,KCELL(I),IND
  455 FORMAT(5X,*KCELL(*,I3,*)=*,I5,*GT IND=*,I5)
      IF(KCELL(I).GT.IND)GOTO 6
      J=GIVE(X)
      GO TO 5
    6 ICELL(LAST)=ICELL(LAST).A.777 000 777B     $     RETURN
      ENTRY RELEASE
C     RELEASE LIST-L
      IF(IH.EQ.0)GOTO8080
      PRINT 88,L
   88 FORMAT(8X,*.. RELEASE..*,I5 )
 8080 CONTINUE
      I=L
    1 IF(I.EQ.0)RETURN
      CALL SAVE(I)    $    I=RL(I)     $     GO TO 1
      END
      FUNCTION LL(L)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
C      LL RETURNS LEFT-LINK
C      TEMPORARY DEBUGGING MSG TO BE REMOVED
```

```
      IF(L.EQ.0)STOP 55
      LL=ICELL(L).A.777B      $      RETURN
C     RL RETURNS THE RIGHT LINK OF LIST-L - RETURNS 0 IF NONE
      END
      INTEGER FUNCTION RL(L)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
C      TEMPORARY DEBUGGING MSG TO BE REMOVED
      IF(L.EQ.0)A=2/(A-A)                               ....(#14)....
      RL=LSHFT(ICELL(L),-9).A.777B
      RETURN
      END
      INTEGER FUNCTION DF(L)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
C     DF RETURNS THE DATA AND FLAG FIELDS
      DF=LSHFT(ICELL(L),-18).A.777B
      RETURN
      END
      INTEGER FUNCTION DA(L)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
C      RETURNS DATA FIELD OF ICELL-L
      DA=LSHFT(ICELL(L),-18).A.77B
      RETURN
      END
      INTEGER FUNCTION FB(L)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
C     FB=FLAG BITS
      FB=LSHFT(ICELL(L),-24).A.3
      RETURN
      END
      INTEGER FUNCTION FR(L)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      INTEGER RL
C     REMOVE FRONT CELL OF LIST-L AND RETURN NEW HEADER ADDRESS
      FR=RL(L)     $     CALL SAVE(L)
      ICELL(FR)=ICELL(FR).A.777 777 000B
      RETURN
      END
      FUNCTION NPWR(I,J)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER RL,DA,FB,DF,FR
C     RETURN THE LCM (NOT 0 OR 1) OF THE POWERS OF EXP-I AND EXP-J
C     - IF NONE RELEVANT RETURN 1000
      NPWR=1000     $     L=I
    2 K=L
    3 KK=DF(K)
      IF((KK.NE.215B).A.(KK.NE.216B))GOTO 20
C     ↑  OR  ↓
      CALL ROP(K,KK)
      IF(DF(KK).NE.0)STOP 474
      KJ=ACELL(KK)     $     IF((KJ.EQ.0).O.(KJ.EQ.1))GO TO 10
      IF(KJ.LT.NPWR)NPWR=KJ
   10 IF(L.NE.J)GO TO 11
      RETURN
   11 L=J    $    GO TO 2
   20 IF(KK.EQ.0)GO TO 10
      IF((KK.NE.211B).A.(KK.NE.214B))GOTO 30
      K=RL(K)     $     GO TO 3
   30 IF((KK.NE.212B).A.(KK.NE.213B))GO TO 10
C     SCAN THRU LEADING (*)-CHAIN
   31 K=RL(K)       $       KK=DF(K)
      IF((KK.EQ.212B).O.(KK.EQ.213B))GO TO 31
   32 IF((KK.NE.215B).A.(KK.NE.216B))GO TO 35
      CALL ROP(K,KJ)
      KK=DF(KJ)    $    IF(KK.NE.0)STOP 425
      KK=ACELL(KJ)
      IF((KK.EQ.0).O.(KK.EQ.1))GO TO 35                ....(#15)....
      IF(NPWR.GT.KK)NPWR=KK
   35 CALL WFF(K,KJ,KK)
      IF(KJ.EQ.0)GO TO 10
C     KJ IS ADDRESS OF NEXT WFF
      K=KJ     $     KK=DF(K)     $     GO TO 32
      END

      FUNCTION XVAL(I)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/POLE/IMP(100),ISTK(20),STK(20),RTK(20),IJ,IK,I1,I2,LAST,LNK
      COMMON/XV/T(6),V(32),LQ(12),JFG(12),D(36),R(36)
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER RL,DA,FB,DF,FR
C     RET. VAL. OF EXP-I ASSUMING VALS OF SIN,COS,... IN T(1),T(2),...
C     WE ASSUME THAT WE ARE DEALING WITH A SINGLE-ARG PROBLEM
      I2=1     $     CALL WFF(I,J,K)
C     WFF RETURNS RIGHT END OF WFF-I IN K
    5 CONTINUE
      I1=FB(K)    $    IF(I1.EQ.2)GOTO 20    $    IF(I1.EQ.0)GOTO 12
C     ELSE ERROR
      PRINT 10   $    XVAL=1000    $    RETURN
   10 FORMAT(5X,*FN XVAL - VARIABLES NOT ALLOWED*)
   12 STK(I2)=ACELL(K)
   13 I2=I2+1
   14 IF(K.EQ.I)GOTO 16    $    K=LL(K)     $     IF(K.NE.0)GOTO 5
C     DEBUGGER
      IF(I2.EQ.2)GO TO 16
      PRINT 15,I2     $     RETURN
   15 FORMAT(5X,*FN XVAL - I2=*,I3)
   16 XVAL=STK(1)      $     RETURN
   20 I1=DA(K)
      GOTO(21,21,21,21,21,21,23,24,25,26,27,28,29,29),I1
   21 STK(I2)=T(I1)     $    GOTO 13
C     +
   23 STK(I2-2)=STK(I2-2)+STK(I2-1)    $    GO TO 31
C     -
   24 STK(I2-2)=STK(I2-1)-STK(I2-2)     $     GO TO 31
C     -
   25 STK(I2-1)=-STK(I2-1)    $    GO TO 14
C     *
   26 STK(I2-2)=STK(I2-2)*STK(I2-1)    $      GO TO 31
C     /
   27 STK(I2-2)=STK(I2-1)/STK(I2-2)     $     GO TO 31
C     RECIPROCAL
   28 STK(I2-1)=1/STK(I2-1)      $     GO TO 14
C     ↑ OR +
   29 XVAL=STK(I2-1)    $    IF(XVAL.LT.0)GOTO 32  $  XVAL=XVAL**STK(I2-2)
   30 IF(I1.EQ.14)XVAL=1/XVAL     $     STK(I2-2)=XVAL
   31 I2=I2-1     $     GO TO 14
   32 J=STK(I2-2)     $     XVAL=(-XVAL)**J
      IF(J.NE.(2*(J/2)))XVAL=-XVAL
      GO TO 30
```

```
      END

      SUBROUTINE SETT(I)
      COMMON/XV/T(6),V(32),LQ(12),JFG(12),D(36),R(36)
C     SET T(1),...T(6) ACCORDING TO ENTRY-I OF LQ-TABLE
      COMMON/F/IH
      IF(IH.EQ.0)GOTO8080
         PRINT 77,I                                          ....(#16)....
   77 FORMAT(10X,*SETT CALLED, ARG=*,I4)
 8080 CONTINUE
      IX=LQ(I)
      K=LSHFT(IX,-30).A.77B  $  T(1)=V(K)
      K=LSHFT(IX,-24).A.77B   $   T(2)=V(K)
      K=LSHFT(IX,-18).A.77B   $   T(3)=V(K)
      K=LSHFT(IX,-12).A.77B   $   T(4)=V(K)
      K=LSHFT(IX,-6).A.77B     $     T(5)=V(K)
      K=IX.A.77B              $      T(6)=V(K)
      RETURN     $     END

      FUNCTION KFIG(I)
      INTEGER RL,DA,FB,DF,FR
C     GIVE CONFIGURATION OF WFF-I
      KFIG=0    $   J=I
      IF(I.EQ.0)RETURN
    2 JG=FB(J)          $      IF(JG.EQ.2)GO TO 5
    3 J=RL(J)    $    IF(J.EQ.0)RETURN    $    GO TO 2
    5 JG=DA(J)    $    IF(JG.GT.6)GO TO 3
      KFIG=KFIG.O.(LSHFT(1,6-JG))    $    GO TO 3
      END

      SUBROUTINE WFF(I,J,K)
C     NOTE .. S/R WFF NOW TAKES 3 ARGUMENTS .. THERE ARE WRONG CALLS TO
      INTEGER RL,DA,FB,DF,FR
      INTEGER RANK
C     23NOV MOD.  K IS RIGHT END OF WFF
C     GIVEN WFF (WELL-FORMED FORMULA) BEGINNING AT ICELL(I), RETURN THE
C     BEGINNING OF THE NEXT WFF IN J - ZERO IF NONE
      K=I     $     ISUM=0
    1 ISUM=RANK(K)+ISUM
      IF(ISUM.EQ.-1)GOTO 2
      K=RL(K)      $     GO TO 1
    2 J=RL(K)      $     RETURN      $     END

      FUNCTION ATRG(L,Z)
C     RET. MAX INCONSIS VAL FOR TRG-FN-L WITH ARGUMENT Z
CCCCCCCC  THIS FUNCTION EXISTS - BUT IS NOT EXPECTED TO BE USED   CCCCCC
      COMMON/AUG20/ARG(20),IRG
*********    THEREFORE A DUMMY     *********
      RETURN
      END
      SUBROUTINE RTP(M)
C     REMOVE 1ST LEVEL TRIVIAL POWER IF ANY IN EXP-M ASSUMING NO
C     EXP OF FORM ((...)↑N)↑N)↑M  ETC
C     24NOV70  ...  NOT EXPECTED TO BE RELEVANT FOR PRELIM USE
C     THEREFORE MADE A DUMMY
      RETURN     $     END

      SUBROUTINE DISTRIB(L,N)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      INTEGER RL,DA,FB,DF,FR
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER GIVE
      GOTO(1,9,20,30),N
C     DISTRIBUTE ¬ OVER ++++.....
    1 CALL DELETE(L)
      I=L=RL(L)                                          ....(#17)....
    2 IX=DF(I)   $   IF(IX.NE.207B)GOTO 4
      ICELL(I)=ICELL(I) + 1000000B
    3 ACELL(I)=-ACELL(I)  $  BCELL(I)=-BCELL(I)  $  I=RL(I)  $  GOTO 2
    4 IF(IX.NE.210B)GOTO6  $  ICELL(I)=ICELL(I)-1000000B  $  GOTO 3
C     REMOVE ¬ IF PRECEDING 1ST COMPONENT - ELSE APPEND ONE
    6 IF(IX.NE.211B)GOTO7  $  CALL DELETE(I)  $  RETURN
    7 IX=GIVE(X)   $   ICELL(IX)=211000000B   $   ACELL(IX)=-ACELL(I)
      BCELL(IX)=-BCELL(I)   $   KCELL(IX)=KCELL(I)  $  CALL INSERT(IX,I)
      RETURN
C     DISTRIBUTE < OVER ***... IN CONTEXT OF ¬<***...
    9 I=RL(L)   $   CALL DELETE(I)
   10 I=RL(I)   $   IN=IM=KCELL(I)
   11 IX=DF(I)   $   IF((IX.EQ.212B).O.(IX.EQ.213B))12,14
   12 IN=KCELL(I)    $    ACELL(I)=1/ACELL(I)    $   BCELL(I)=1/BCELL(I)
      I=RL(I)    $    GOTO 11
   14 IF(IX.NE.214B)GOTO 15  $  CALL DELETE(I)  $  GO TO 18
   15 IF(IX.NE.211B)GOTO17  $  IX=RL(I)  $  IX=DF(IX)
      IF(IX.NE.214B)GOTO 17    $   ACELL(I)=1/ACELL(I)
      BCELL(I)=1/BCELL(I)  $  I=RL(I)  $  CALL DELETE(I)  $  GOTO 18
   17 IN=IN-1   $   GOTO 19
   18 I=RL(I)    $    IF(KCELL(I).NE.IN)GO TO 18
   19 IX=GIVE(X)   $   ICELL(IX)=214000000B   $   ACELL(IX)=1/ACELL(I)
      BCELL(IX)=1/BCELL(I)  $  KCELL(IX)=KCELL(I)  $  CALL INSERT(IX,I)
      IF(IN.EQ.IM)RETURN   $   IN=IN+1   $   GO TO 18
C     DISTRIBUTE < OVER ***... IN CONTEXT: <¬***...
   20 CALL DELETE(L)   $   I=L=RL(L)
      ACELL(L)=1/ACELL(L)   $   BCELL(L)=1/BCELL(L)   $   GOTO 10
C     DISTRIBUTE < OVER *** IN CONTEXT: <***...
   30 CALL DELETE(L)   $   I=L=RL(L)
      IM=IN=KCELL(I)   $   GOTO 11
      END

      SUBROUTINE VAL(I,J,K,X,Y)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/NOV/KS(25),JA(2,10),NSCH(2,3),M(2),JP(2),KP1,KP2
      COMMON/F/IH,JH,KH,LH,MH,NH
C     RET TOTAL VAL OF THE COMPS OF EXP-I (I=1 OR 2) UNDER SUBMATCH-J
C     (J=0 FOR(+)-SUBMATCH), TOTAL NO OF COMPS IS K (K=1,2,3)
C     IN X(MAX CONSISTENCY) AND Y(ALG-CONSISTENCY)
      IF(J.NE.0)GOTO10
C     (+)-SUBMATCH
      X=Y=0
      DO 5 L=1,K
      N=NSCH(I,L)   $   N1=JA(I,N).A.777B   $   N2=JA(I,N).A.70000B
      IF(N2.NE.0)GOTO 4
      Z=AC(N1)       $       Z=BC(N1)
      X=X+ACELL(N1)   $   Y=Y+BCELL(N1)   $   GO TO 5
C     NEGATION
    4 X=X-ACELL(N1)   $   Y=Y-BCELL(N1)
      Z=AC(N1)       $       Z=BC(N1)
    5 CONTINUE
      IF(KH.EQ.0)GO TO 221
      PRINT 44,I,J,K,X,Y  ,NSCH(I,1),NSCH(I,2),NSCH(I,3)
  221 CONTINUE
      RETURN
C     (*)-SUBMATCH
   10 X=Y=1.
      DO 15 L=1,K
      N=NSCH(I,L)   $   N1=JA(I,N).A.777B   $   N2=JA(I,N).A.70000B
      IF(N2.NE.0)GOTO 14
      Z=AC(N1)       $       Z=BC(N1)
      X=X*ACELL(N1)     $     Y=Y*BCELL(N1)    $    GO TO 15
   14 X=X/ACELL(N1)    $    Y=Y/BCELL(N1)
      Z=AC(N1)       $       Z=BC(N1)
   15 CONTINUE
      IF(KH.EQ.0)GOTO 222
      PRINT 44,I,J,K,X,Y      ,NSCH(I,1),NSCH(I,2),NSCH(I,3)
   44 FORMAT(10X,*VAL - I,J,K =*,3I7,*    X,Y=*, 2F20.9,3I7)
  222 CONTINUE
      RETURN    $    END
                                                         ....(#18)....
      SUBROUTINE SMALL(II,MM)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/NOV/KS(25),JA(2,10),NSCH(2,3),M(2),JP(2),KP1,KP2
      COMMON NX,JPAK,MONO,ISCH,ERR
          COMMON/FEB71/NT
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER RL,DA,FB,DF,FR
C     ANALYSE STEP-II IN KS-TABLE - RESOLVING INTO SUBSTEPS IF POSSIBLE
C     ELSE DETERMINE STEP SIZE IF UNRESOVABLE
C     RETURN MM NONZERO IF UNRESOLVED AND LARGE, ZERO OTHERWISE
C
C     INITIALISE
      MM=0
      I=KS(II)
      M(1)=I.A.777B      $      M(2)=LSHFT(I,-9).A.777B
       I11=M(1)      $      I12=M(2)
      IF(ABS(ACELL(I11)+ACELL(I12)).LT.ERR)CALL TABON(M(1),0)
      N11=LSHFT(I,-18).A.77B      $      N12=LSHFT(I,-24).A.77B
      N21=LSHFT(I,-30).A.77B      $      N22=LSHFT(I,-36).A.77B
C     REMOVE TRIVIAL POWERS
    5 CONTINUE
C     REMOVE ANY MATCHING LEADING < OR ¬ IN THE 2 EXPRESSIONS .. IF A
C     LEADING (BUT NON-MATCHING) <¬ EXISTS - FLIP < AND ¬ TO GIVE ¬< AS
C     STANDARD FORM
    8 I11=DF(M(1))      $      I21=DF(M(2))
      IF(I11.NE.211B)GOTO 24    $    IF(I21.NE.211B)GOTO 10
C     211B IS ¬  AND  214B IS <
C     REMOVE LEADING OPERATOR PAIR
    9 M(1)=FR(M(1))      $      M(2)=FR(M(2))      $      GO TO 8
   10 IF(I21.NE.214B)GO TO 30
      I21=RL(M(2))     $     I22=DF(I21)
      I11=RL(M(1))      $      I12=DF(I11)
      IF(I22.NE.211B)GOTO 22      $      IF(I12.EQ.214B)GO TO 20
C     ¬... VS  <¬.....
C     INTERCHANGE (M(2))-1  WITH (M(2)) - NO WORRY RE NEW (M(2))-1
   14 I3=RL(I21)      $      IF(I22.EQ.211B)15,18
   15 ICELL(I21)=ICELL(I21)+ 3 000 000B
      ACELL(I21)=1/ACELL(I3)     $     BCELL(I21)=1/BCELL(I3)
C     REMOVE LEADING OPERATOR PAIR
   16 M(1)=FR(M(1))      $      M(2)=FR(M(2))      $      GO TO 30
   18 ICELL(I21)=ICELL(I21)-3000000B
      ACELL(I21)=-ACELL(I3)     $     BCELL(I21)=-BCELL(I3)    $    GOTO 16
C     REMOVE <¬ PAIR
   20 M(1)=FR(M(1))    $    M(2)=FR(M(2))    $    M(1)=FR(M(1))
      M(2)=FR(M(2))     $     GO TO 30
   22 IF(I12.NE.214B)GOTO 30
      I21=I11     $     I22=I12     $     GO TO 14
   24 IF(I11.NE.214B)GO TO 26
      IF(I21.EQ.214B)GOTO 9
      IF(I21.NE.211B)GO TO 28

      I21=RL(M(2))       $      I22=DF(I21)

      I11=RL(M(1))      $      I12=DF(I11)
      IF(I22.EQ.214B)GO TO 25
      IF(I12.NE.211B)GO TO 30
C     INTERCHANGE (M(1))-1  AND (M(1))-2
      I21=I11   $   I22=I12   $   GOTO 14              ....(#19)....
   25 IF(I12.EQ.211B)20,14
C     INTERCHANGE < AND ¬ IN THE CONTEXT <¬ ... IN EXP-2
   26 IF(I21.NE.214B)GO TO 30
      I21=RL(M(2))     $     I22=DF(I21)
      IF(I22.NE.211B)GOTO 30     $     I22=M(2)
   27 ICELL(I22)=ICELL(I22)-3000000B   $   I11=RL(I22)
      ICELL(I11)=ICELL(I11)+3000000B
      ACELL(I11)=-ACELL(I22)    $   BCELL(I11)=-BCELL(I22)   $   GOTO 30
   28 I11=RL(M(1))     $     I12=DF(I11)
      IF(I12.NE.211B)GOTO 30    $   I22=M(1)     $    GOTO 27
   30 IFLAG=0     $     CALL TYPE(M(1),I11)       $    CALL TYPE(M(2),I22)
      GOTO(40,41,42,43,44,45,46,47,48,49,50,51,52,53,54,55,56),I22
C     ERROR
   35 STOP 5
C     ↑BM
   40 GOTO(60,70,75,77,75,77,60,70,80,80,88,88,84,84,84,84,88),I11
C     ¬↑BM
   41 GOTO(70,35,77,35,77,35,70,35,80,35,88,35,84,35,84,35,88),I11
C     <↑BM
   42 GOTO(75,77,35,35,60,70,35,35,80,80,35,35,84,84,35,35,88),I11
C     ¬<↑BM
   43 GOTO(77,35,35,35,70,35,35,35,80,35,35,35,84,35,35,35,88),I11
C     ↓BM
   44 GOTO(75,77,60,70,60,70,75,77,80,80,88,88,84,84,84,84,88),I11
C     ¬↓BM
   45 GOTO(77,35,70,35,70,35,77,35,80,35,88,35,84,35,84,35,88),I11
C     <↓BM
   46 GOTO(60,70,35,35,75,77,35,35,80,80,35,35,84,84,35,35,88),I11
C     ¬<↓BM
   47 GOTO(70,35,35,35,77,35,35,35,80,35,35,35,84,35,35,35,88),I11
C     +.....+PQ....
   48 GOTO(80,80,80,80,80,80,80,80,80,80,80,80,90,90,90,90,80),I11
C     ¬+...+PQ....
   49 GOTO(80,35,80,35,80,35,80,35,80,35,80,35,90,35,90,35,80),I11
C     <+.....+PQ....
   50 GOTO(88,88,35,35,88,88,35,35,80,80,35,35,84,84,35,35,88),I11
C     ¬<+.....+PQ....
   51 GOTO(88,35,35,35,88,35,35,35,80,35,35,35,84,35,35,35,88),I11
C     *...*PQ...
   52 GOTO(84,84,84,84,84,84,84,84,90,90,84,84,84,84,84,84,84),I11
C     ¬*...*PQ...
   53 GOTO(84,35,84,35,84,35,84,35,90,35,84,35,84,35,84,35,84),I11
C     <*...*PQ....
   54 GOTO(84,84,35,35,84,84,35,35,90,90,35,35,84,84,35,35,84),I11
C     OTHERS
   55 GOTO(84,35,35,35,84,35,35,35,90,35,35,35,84,35,35,35,84),I11
   56 GOTO(88,88,88,88,88,88,88,88,80,80,88,88,84,84,84,84,88),I11
C     A=B   OR   A=-B
C
   60 ASSIGN 62 TO JBACK
   61 CALL PWR(M(1),I11)      $      CALL PWR(M(2),I22)
      IF(I11.NE.I22)GO TO 88
      CALL MANTISA(M(1))     $    CALL MANTISA(M(2))
      I11=M(1)     $     I22=M(2)      $     GOTO JBACK
   62       NT=NT+1
```

```
      IF(ABS(ABS(ACELL(I11))-ABS(ACELL(I22))).LT.10E-8)GOTO 66
C     ERROR
   63 PRINT 64,ACELL(I11),ACELL(I22)
   64 FORMAT((/5X,*S/R SMALL AT 64 - VALS=*2F20.9/)
      STOP 3
   66       NT=NT+1
      IF(ABS(ACELL(I11)-ACELL(I22)).LT.10E-8)GOTO 69
   67 CALL TAGON(M(1),0)
   69 N11=N12=N21=N22=0                                  ....(#20)....
      GOTO 5
C     A=-B
   70 ASSIGN 71 TO JBACK                 $          GO TO 61
   71       NT=NT+1
      IF(ABS(ACELL(I11)+ACELL(I22)).GT.10E-8)63,67
C     A=1/B  OR  A=-1/B
   75 ASSIGN 76 TO JBACK     $     GOTO 61
   76 IF(ABS(ABS(ACELL(I11))-1/ABS(ACELL(I22))).GT.10E-8) GOTO 63
      CALL TAGON(M(1),1)    $    I11=M(1)
      IF(ABS(ACELL(I11)-ACELL(I22)).LT.ERR)69,67
C     A=-1/B
   77 ASSIGN 78 TO JBACK            $      GO TO 61
   78       NT=NT+1
      IF(ABS(ACELL(I11)+1/ACELL(I22)).GT.ERR)GOTO 63
      CALL TAGON(M(1),1)    $    I11=M(1)    $    GOTO 67
C     TRY (+)-SUBMATCH IF APPLICABLE
C     SEE IF (+)-SUBMATCH TRIED 84 - IF SO WAS IT OVER FEWER COMPS.
   80 IF(N11.EQ.0)GOTO 82
      CALL NCOMP(M(1),207B,I11)
      IF(N11.NE.I11)GOTO 82
      CALL NCOMP(M(2),207B,I11)
      IF(N12.NE.I11)82,83
   82 IZ=0      $        CALL SBMATCH(IZ,JZ)
      IF(JZ.NE.0)RETURN
      N11=JP(1)     $     N12=JP(2)
   83 IF(IFLAG.EQ.0)GO TO 88
C     TRY (*)-SUBMATCH
   84 IF(N12.EQ.0)GOTO 85
      CALL NCOMP(M(1),212B,I11)
      IF(I11.NE.N21)GO TO 85
      CALL NCOMP(M(2),212B,I11)
      IF(I11.NE.N22)GOTO 88
   85 IZ=1      $        CALL SBMATCH(IZ,JZ)
      IF(IH.EQ.0)GOTO 909
       PRINT 99
   99 FORMAT(15X,*...... RETURNED FROM S/R SBMATCH ...*)
  909 CONTINUE
      IF(JZ.NE.0)RETURN
CCCCCCCC    INSERT  2-NDARY SUBMATCH HERE
C
C     TEST SIZE OF CURRENT STEP IF IT WAS NOT RESOLVABLE
      N21=JP(1)     $     N22=JP(2)
   88 KS(II)=(KS(II).A.777 777 777 000 000B)+M(1)+M(2)*1000B
C  ***   ABOVE CARD REDUNDANT AND SILLY
      IF(M(2).EQ.0)GOTO 890
   89 CALL SIZE(II,MM)     $     RETURN
  890 IF(IZ.NE.0)GOTO 891
      V2=0.     $     GO TO 89
  891 V2=1.     $     GO TO 89
C     TRY (+)-SUBMATCH ... IF FAIL TRY *-SUBMATCH
   90 IFLAG=1     $     GOTO 80
C     UPDATE KS-TABLE
C     ***$  WHERE IS THIS SUPPOSED TO GO
C     KS(II)=(KS(II).A.700 000 000 000 000B)+M(1) + M(2)*1000B
```

```
C     X  +  LSHFT(N11,18) + LSHFT(N12,24) + LSHFT(N21,30)+LSHFT(N22,36)
      END

      SUBROUTINE SBMATCH(IZ,JZ)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/NOV/KS(25),JA(2,10),NSCH(2,3),M(2),JP(2),KP1,KP2
      COMMON NX,JPAK,MONO,ISCH,ERR
          COMMON/FEB71/NT
      COMMON/F/IH,JH,KH,LH,MH,NH                        ....(#21)....
      INTEGER RL,DA,FB,DF,FR
C     INITIALISE TO NO-SUBMATCH
      JZ=0
C     RESOLUTION INTO COMPONENTS SECTION
      K=1    $    IF(IZ.NE.0)GOTO 10
C     ADDITIVE COMPONENTS
    2 IF(M(K).NE.0)GO TO 302
      JP(K)=JA(K,1)=0       $       GO TO 8
  302 I11=DF(M(K))
      IF((I11.EQ.207B).O.(I11.EQ.210B))GOTO 7
      IF(I11.EQ.211B)GOTO 4
C     ONLY 1 COMPONENT
    3 JP(K)=1     $     JA(K,1)=M(K)     $     GO TO 8
    4 I11=RL(M(K))     $     I11=DF(I11)
      IF((I11.NE.207B).A.(I11.NE.210B))GOTO 3
      CALL DISTRIB (M(K),1)
C     DISTRIBUTE ¬ OVER ++++ ....
C     RESOLVE INTO (+)-COMPONENTS
    7 CALL RESOLVE(M(K),207B,K)
    8 IF(K.EQ.2)GOTO 25
      K=2    $    GOTO 2
C     MULTIPLICATIVE COMPONENTS
   10 IF(M(K).NE.0)GOTO 303
      JP(K)=JA(K,1)=0       $       GO TO 18
  303 I11=DF(M(K))
      IF((I11.EQ.212B).O.(I11.EQ.213B))GO TO 20
      IF(I11.EQ.211B)GOTO 22
      IF(I11.NE.214B)GOTO 14
      I11=RL(M(K))      $      I12=DF(I11)
      IF((I12.EQ.212B).O.(I12.FQ.213B))12,13
C     DISTRIBUTE < OVER *** .....
   12 CALL DISTRIH(M(K),4)                  $      GO TO 20
   13 IF(I12.EQ.211B)GOTO 15
   14 JP(K)=1     $     JA(K,1)=M(K)     $     GO TO 18
   15 I12=RL(I11)     $     I12=DF(I12)
      IF((I12.NE.212B).A.(I12.NF.213B))GOTO 14
C     DISTRIBUTE < IN CONTEXT  <¬***...
      CALL DISTRIH(M(K),3)
C     23 AUG 70 - IN VIEW OF <¬***  BEING REARRANGED TO ¬<***.. PRIOR TO
C     CALL TO S/R SUBMATCH ABOVE SITUATION IMPOSSIBLE
   17 CALL RESOLVE(I11,212B,K)
   18 IF(K.EQ.2)GOTO 25
      K=2     $     GOTO 10
   20 I11=M(K)      $     GOTO 17
   22 I11=RL(M(K))      $      I12=DF(I11)
      IF((I12.EQ.212B).O.(I12.EQ.213B))GOTO 17
      IF(I12.NE.214B)GOTO 14
      I12=RL(I11)     $     I12=DF(I12)
      IF((I12.NE.212B).A.(I12.NE.213B))GOTO 14
C     DISTRIBUTE < OVER ***... IN CONTEXT ¬<*****...
      CALL DISTRIB(M(K),2)
      GO TO 17
C     EXIT FROM RESOLUTIOM SECTION
```

```
   25 IF(( JP(1).EQ.1).A.(JP(2).EQ.1)) RETURN
C     SINCE THERE IS ONLY 1 CONP EACH - NO SUBMATCH POSSIBLE
*
      NO1=NO2=0
      NF1=NF2=0
**************
      N1=JP(1)    $   N2=JP(2)    $    ISCX=ISCH     ....(#22)....
   27 I2=ISCX.A.7B     $     IF(I2.EQ.0)GO TO 32
      ISCX=LSHFT(ISCX,-3)     $     I1=ISCX.A.7B    $    ISCX=LSHFT(ISCX,-3)
*
*     BECAUSE OF CHANGE IN HANDLING OF IMPLICIT STEPS - SOME CLEANING
*      UP OF OLD CODES IN ORDER
*
   28 IF(N1.NE.0)GO TO 29
      CALL ONE(IZ,M(1))   $  N1=1   $  NF1=0   $  JA(1,1)=M(1)
   29 IF(N2.NE.0)GO TO 30
      CALL ONE(IZ,M(2))   $  N2=1   $  NF2=0   $  JA(2,1)=M(2)
   30 IF((I1.GE.N1).A.(I2.GE.N2))GO TO 32
      IF(NF1.NE.0)CALL RJA(NF1,I1,N1)
      IF(NF2.NE.0)CALL RJA(NF2,I2,N2)
      NF1=NF2=0
      NSCH(1,1)=0
      IF(I1.GT.N1)GOTO 430
      LM1=NCM(N1,I1)      $     GO TO 431
  430 LM1=1    $    X1=Y1=5000
  431 IF(I2.GT.N2)GO TO 432
      LM2=NCM(N2,I2)      $     GO TO 435
   32 IF(JZ.EQ.0)RETURN
C     ***  CHECK FOR ALG-REMNANT STEP NOT DETECTED BEFORE  ******
      I11=M(1)    $    I12=M(2)
       Z=AC(I11)       $       Z=AC(I12)     $     Z=BC(I11)    $   Z=BC(I12)
            NT=NT+1
      IF(ABS(ABS(BCELL(I11))-ABS(BCELL(I12))).LT.ERR)GOTO 80
      IF(IH.EQ.0)GOTO 808
      PRINT 199,M(1),M(2),KP2
  199 FORMAT(16X,*+++ SBMATCH ABOUT TO RETURN ++ M(1),M(2),KP2=*,3I6)
  808 CONTINUE
      KS(KP2)=M(1)+LSHFT(M(2),9)    $    KP2=KP2+1    $    RETURN
  432 LM2=1     $     X2=Y2=3000
C     IFX IS A FLAG SET FOR ALG SUBSTEP
  435 IFX=0
   35 DO 250 I=1,LM1
      NSCH(2,1)=0
      IF(N1.LT.I1)GOTO436
      CALL G(N1,1,I1)
      CALL VAL(1,IZ,I1,X1,Y1)
  436 CONTINUE
      DO 200 J=1,LM2
      IF(N2.LT.I2)GOTO 437
      CALL G(N2,2,I2)
      CALL VAL(2,IZ,I2,X2,Y2)
  437 CONTINUE
C     NOW SEE IF THERE IS A MATCH
*
*
C     (+) OR (*) SUBMATCH
      IF(IZ.NE.0)GO TO 40
C     (+)-S/MATCH ATTEMPT
            NT=NT+1
      IF(ABS(X1-X2).GT.ERR)GO TO  100
      NF1=1    $    NF2=2
C     IS IT ALGEBRAIC
            NT=NT+1
      IF(ABS(Y1-Y2).GT.ERR)GO TO 60
   38 IFX=5     $     GO TO 60
C     (*)-S/MATCH ATTEMPT
   40         NT=NT+1
      IF(ABS(ABS(X1)-ABS(X2)).GT.ERR)GO TO 150
      NF1=1    $    NF2=2
              NT=NT+1
      IF(ABS(ABS(Y1)-ABS(Y2)).GT.ERR)60,38
********* 1 DEC 70 ******
C     TRY IMPLICIT (+)-MATCH
  100 IF(NO2.GE.I2)GOTO 200
              NT=NT+1
      IF(ABS(X2).GT.ERR)GO TO 200
C     GUARD AGAINST A SELF-MATCH .. I.E. 1 VS 1  OR  0 VS 0
      CALL TRV(2,I2,I11)
      IF(I11.NE.0)GO TO 200
      NF2=2
      JZ=5    $    CALL ONE(IZ,NX)
              NT=NT+1
      IF(ABS(Y2).GT.ERR)GO TO 120
C     IMPLICIT ALGEBRAIC
  101 CALL DELETE(NX)
      IF(I2.LT.N2)GOTO 105
      CALL RELEASE(M(2))    $    N2=0    $    GO TO 28
  105 I11=1
  108 IP=NSCH(2,I11)     $     IP=JA(2,IP).A.777B
      CALL DETACH(IP,M(2))     $     CALL RELEASE(IP)
      IF(I11.EQ.I2)GO TO 110
      I11=I11+1     $     GO TO 108
  110 N2=N2-I2     $     GO TO 28
C     IMPLICIT T-MATCH
  120 IF(I2.NE.N2)GOTO 130
      KS(KP2)=M(2)+LSHFT(NX,9)
  122 KP2=KP2+1     $    N2=0    $    GO TO 28
  130 CALL WELD(2,IZ,I2,IP)
        Z=AC(IP)
      IF(ABS(ACELL(IP)+1).LT.ERR)CALL TAGON(IP,0)
      KS(KP2)=IP+LSHFT(NX,9)
      KP2=KP2+1     $     GO TO 110
C     IMPLICIT (*)-MATCH
  150 IF(NO2.GE.I2)GO TO 200
              NT=NT+1
      IF(ABS(ABS(X2)-1).GT.ERR)GO TO 200
      CALL TRV(2,I2,I11)
      IF(I11.NE.0)GO TO 200
      NF2=2
      JZ=5    $    CALL ONE(IZ,NX)
              NT=NT+1
      IF(ABS(ABS(Y2)-1).GT.ERR)120,101
  200 CONTINUE
      IF(NO2.LT.I2)NO2=I2
      IF(NO1.GE.I1)GOTO 250
      IF(IZ.NE.0)GOTO 240
              NT=NT+1
      IF(ABS(X1).GT.ERR)GOTO 250
      CALL TRV(1,I1,I11)
      IF(I11.NE.0)GO TO 250
      NF1=1
      JZ=10    $    CALL ONE(IZ,NX)
              NT=NT+1
      IF(ABS(Y1).GT.ERR)GO TO 220
C     IMPLICIT ALGEBRAIC
  201 IF(I1.LT.N1)GOTO 205
```

....(#23)....

```
      CALL RELEASE(M(1))     $     N1=0     $     GO TO 28
  205 I11=1
  208 IP=NSCH(1,I11)     $     IP=JA(1,IP).A.777B
      CALL DETACH(IP,M(1))    $    CALL RELEASE(IP)
      IF(I11.EQ.I1)GO TO 210
      I11=I11+1     $     GO TO 208
  210 N1=N1-I1    $    GO TO 28
C     T-IMPLICIT
  220 IF(I1.NE.N1)GOTO 230
      KS(KP2)=M(1)+LSHFT(NX,9)
  222 KP2=KP2+1    $    N1=0     $     GO TO 28
  230 CALL WELD(1,IZ,I1,IP)
        Z=AC(IP)
      IF(ABS(ACELL(IP)+1).LT.ERR)CALL TAGON(IP,0)
      KS(KP2)=IP+LSHFT(NX,9)
      KP2=KP2+1     $     GO TO 210
  240         NT=NT+1
      IF(ABS(ABS(X1)-1).GT.ERR)GO TO 250
      CALL TRV(1,I1,I11)
      IF(I11.NE.0)GO TO 250
      NF1=1
      JZ=10    $    CALL ONE(IZ,NX)
              NT=NT+1
      IF(ABS(ABS(Y1)-1).GT.ERR)220,201
  250 CONTINUE
      IF(NO1.LT.I1)NO1=I1
      GO TO 27
   55 PRINT 56,ACELL(I11),ACELL(I12)     $     STOP
   56 FORMAT(5X,*SUBMATCH - STEP BAD - VALS=*,2F20.9)
   60 JZ=24567
      IF(IFX.NE.0)GO TO 70
C     CHECK CASE OF WHOLE-EXP MATCHING SUB-EXP
      IF(I1.NE.N1)GOTO 64
      N1=0     $     I11=M(1)     $     M(1)=0
      CALL WELD(2,IZ,I2,I12)
      N2=N2-I2
   61 IF(ABS(ACELL(I11)+ACELL(I12)).LT.ERR)CALL TAGON(I11,0)
        Z=AC(I11)      $      Z=AC(I12)
      KS(KP2)=I11+LSHFT(I12,9)    $    KP2=KP2+1    $    GO TO 28
   64 IF(I2.NE.N2)GOTO 65
      N2=0    $    I12=M(2)    $    M(2)=0
      CALL WELD(1,IZ,I1,I11)
      N1=N1-I1
      GO TO 61
   65 CONTINUE
      CALL WELD(1,IZ,I1,I11)    $    CALL WELD(2,IZ,I2,I12)
        Z=AC(I11)      $      Z=AC(I12)
              NT=NT+1
      IF(ABS(ABS(ACELL(I11))-ABS(ACELL(I12))).GT.ERR)GO TO 55
C     COMPENSATE SIGN
      IF(ABS(ACELL(I11)+ACELL(I12)).LT.ERR)CALL TAGON(I11,0)
      KS(KP2)=I11+LSHFT(I12,9)+LSHFT(I1,18)+LSHFT(I2,24)
      KP2=KP2+1
C     CHECK REMNANT STEP
      I11=M(1)    $    I12=M(2)
        Z=BC(I11)        $        Z=BC(I12)
              NT=NT+1
      IF(ABS(ABS(BCELL(I11))-ABS(BCELL(I12))).LT.ERR)GO TO 80
        Z=AC(I11)      $      Z=AC(I12)
              NT=NT+1
      IF(ABS(ABS(ACELL(I11))-ABS(ACELL(I12))).GT.ERR)GO TO 55
C     COMPENSATE SIGN
      IF(ABS(ACELL(I11)+ACELL(I12)).LT.ERR)CALL TAGON(M(1),0)
```

....(#24)....

```
   62 N1=N1-I1    $    N2=N2-I2    $    GO TO 28
C     RELEASE MATCHED ALGEBRAIC SUBSTEP
   70 I11=1    $    NX=I1    $    I12=1    $    NY=N1
   71 IF(NX.NE.NY)GOTO 72                              ....(#25)....
C     CASE OF WHOLE-EXP MATCHING SUB-EXP
      CALL RELEASE(M(I11))    $    M(I11)=0    $    GO TO 75
*
*******    MAKE SURE ALL REFS TO POINTER IN JA(-,-) ARE MASKED
*
   72 IP=NSCH(I11,I12)    $    IP=JA(I11,IP).A.777B
      CALL DETACH(IP,M(I11))    $    CALL RELEASE(IP)
      IF(I12.EQ.NX)GO TO 75    $    I12=I12+1    $    GO TO 72
   75 IF(I11.EQ.2)GOTO 62   $   I11=2   $   NX=I2   $   NY=N2
      I12=1     $     GOTO 71
   80 CALL RELEASE(M(1))     $     CALL RELEASE(M(2))
      RETURN
*
C     THERE IS STILL CASE OF ..F1*F2*F3... ≠ 0  TO CONSIDER
*
      END

      SUBROUTINE RESOLVE(I,J,K)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/NOV/KS(25),JA(2,10),NSCH(2,3),M(2),JP(2),KP1,KP2
      COMMON/F/IH
      INTEGER RL,DA,FB,DF,FR
C     RESOLVE EXPRESSION-K (K=1 OR 2) BEGINNING AT I INTO (+)-COMPS
C     IF J SET TO 207B ELSE TO (*)-COMPS IF  J=212B
C     SCAN LEADING OPERATORS UNTIL AN OPERATOR  OF A DIFF. CLASS
      IF(IH.EQ.0)GOTO 606
        PRINT 77,I,J,K
   77 FORMAT(12X,*S/R RESOLVE, ARGS=*,3I9 )
  606 CONTINUE
      IN=IM=KCELL(I)     $     I1=I
    5 IL=I1     $     I1=RL(I1)     $     ID=DF(I1)
      IF((ID.NE.J).A.(ID.NE.J+1))GOTO 10
      IN=KCELL(I1)     $     GO TO 5
   10 IP=1    $    JA(K,IP)=I1
   12 I1=RL(I1)
C      TO BE REMOVED
      IF(I1.EQ.0)STOP 424
      IF( KCELL(I1).NE.IN)GOTO 12
      IP=IP+1     $     JA(K,IP)=I1     $     ID=DF(IL)
      IF(ID.EQ.J)GOTO 15
C     IF  ID.NE.J  NEGATE FLAG ENTRY
      JA(K,IP)=JA(K,IP)+10000B
   15 IF(IN.EQ.IM)GOTO 20
      IN=IN+1     $     IL=LL(IL)     $     GO TO 12
C     CLEAN UP
   20 JP(K)=IP
      RETURN
C     CALLING SEQUENCE:    NCOMP(I,J,K)
      ENTRY NCOMP
C     RETURN K = NO OF (+) (J=207B) OR (*) COMPS IN EXPR-I
      K=1     $     IN=I
   25 I1=DF(IN)
      IF((I1.EQ.J).O.(I1.EQ.J+1))GOTO 30
      IF(J.EQ.212B)GOTO 27
      IF(I1.EQ.211B)GOTO 28
   26 RETURN
   27 IF((I1.EQ.211B).O.(I1.EQ.214B))28,26
   28 IN=RL(IN)     $    GOTO 25
```

```
   30 K=K+1     $     IN=RL(IN)     $     I1=DF(IN)
      IF((I1.EQ.J).O.(I1.EQ.J+1))30,26
      ENTRY RJA
C     UPDATE JA-TABLE FOLLOWING SUBSTEP DETACHMENT
C     I=1 OR 2, J=I1 OR I2 AND K=N1 OR N2
C           RESET  K TO FORMER N1 OR N2
      KK=K+J
      DO 60 IN=1,J
      IM=NSCH(I,IN)    $    JA(I,IM)=0                    ....(#26)....
   60 CONTINUE
      IM=0
      DO 70 IN=1,KK
      IF(JA(I,IN).EQ.0)GO TO 70
      IM=IM+1     $     JA(I,IM)=JA(I,IN)
   70 CONTINUE
      RETURN
      ENTRY TRV
C     RETURN K NONZERO IF IMPLICIT MATCH IS WITH SELF - I.E. 0 VS 0
C     OR 1 VS 1.  I IS 1 OR 2 AND J IS I1 OR I2
      K=0    $    IF(J.GT.1)RETURN
      KK=NSCH(I,1)    $    KK=JA(I,KK).A.777B
C     A CONSTANT HAS A ZERO DF( )
      IF(DF(KK).NE.0)RETURN    $    K=4    $    RETURN
      END
      SUBROUTINE TIME(X,TM)
      DIMENSION LT(4),LS(2)
      CALL LTIME(LT)
C     WE NEED LT(4) EVEN THO WE USE ONLY 1ST TWO BECAUSE LTIME EXPECTS IT
      TM=(LT(1)-LS(1))+0.001*(LT(2)-LS(2))
      IF(X.EQ.0)GOTO 2
      PRINT 50,LS(1),LS(2),LT(1),LT(2),TM
   50 FORMAT(5X,*INIT TIME=*,I5,*.*,I4,5X,*NOW=*,I5,*.*,I4,5X,*ELAPSED
     XTIME=*,F10.4,*  SEC*)
    2 LS(1)=LT(1)    $    LS(2)=LT(2)    $    RETURN    $    END

      SUBROUTINE DETACH(I,J)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/F/IH
      INTEGER RL,DA,FB,DF,FR
C     DETACH WFF-I FROM FORMULA-J
C     FREE WFF-I
      CALL RWFF(I,JL)
      IND=KCELL(I)    $    L=RL(JL)
C     IS WFF-I F1 OR F2 IN  ...+F1F2....
    5 IF(KCELL(JL)-IND)10,30,40
C     WFF-I IS F2
   10 KCELL(JL)=KCELL(JL)+1
      L=JL     $     JL=LL(JL)     $     IF(KCELL(JL).LT.IND)GOTO 10
C     REMOVE THE OPERATOR ASSOC. WITH WFF
   11 CALL DELETE(JL)
   12 JL=L     $     L=LL(L)     $     IF(L.NE.0)GOTO 15
      J=JL     $     RETURN
   15 IF(KCELL(L)-IND)12,18,16
   16 CALL ROP(L,IG)
   18 L1=DF(L)-206B
      GOTO(20,21,22,23,24,25),L1
   20 ACELL(L)=ACELL(JL)+ACELL(IG)   $  BCELL(L)=BCELL(JL)+BCELL(IG)
       GOTO 12
   21 ACELL(L)=ACELL(JL)-ACELL(IG)   $  BCELL(L)=BCELL(JL)-BCELL(IG)
       GOTO 12
   22 ACELL(L)=-ACELL(JL)              $  BCELL(L)=-BCELL(JL)
       GOTO 12
   23 ACELL(L)=ACELL(JL)*ACELL(IG)   $  BCELL(L)=BCELL(JL)*BCELL(IG)
       GOTO 12
   24 ACELL(L)=ACELL(JL)/ACELL(IG)   $  BCELL(L)=BCELL(JL)/BCELL(IG)
       GOTO 12
   25 ACELL(L)=1/ACELL(JL)              $  BCELL(L)=1/BCELL(JL)
       GOTO 12
C     NEITHER
   30 L1=JL    $    JL=LL(JL)    $    CALL DELETE(L1)    $    GOTO 5
C     WFF-I IS F1
   40 IG=DF(JL)
      IF((IG.EQ.207B).O.(IG.EQ.212B))GOTO 11
      L=RL(JL)    $    L1=DF(L)
      IF(IG.NE.210B)GOTO 48                                 ....(#27)....
      IF(L1.NE.211B)GOTO 42
   41 CALL DELETE(L)
      L=RL(JL)     $     GOTO 11
C     CONVERT - TO ¬
   42 ICELL(JL)=ICELL(JL)+1000000B
      ACELL(JL)=-ACELL(L)    $    BCELL(JL)=-BCELL(L)
      GO TO 12
   48 IF(IG.NE.213B)STOP 426
      IF(L1.EQ.214B)GO TO 41
C     CONVERT / TP <
      ICELL(JL)=ICELL(JL)+1000000B
      ACELL(JL)=1/ACELL(L)    $    BCELL(JL)=1/BCELL(L)
      GO TO 12
      END
      FUNCTION NCM(N,M)
      COMMON/F/IH
C     RETURN N-COMBINATION-M
      NR=N+1    $    NUM=1    $    NDEN=1
      DO 10 I=1,M
      NR=NR-1    $    NUM=NUM*NR    $    NDEN=NDEN*I
   10 CONTINUE
C     REMOVE DEBUGGING PRINT
      NCM=NUM/NDEN
      IF(IH.EQ.0)GOTO 505
      PRINT 12,N,M,NCM
  505 CONTINUE
   12 FORMAT(5X,I2,* -COMB-*,I2,* IS *,I3)
      RETURN     $     END

      SUBROUTINE G(N,I,J)
      COMMON/NOV/KS(25),JA(2,10),NSCH(2,3),M(2),JP(2),KP1,KP2
      COMMON/F/IH
C     SET THE NEXT COMBINATION OF J-WRT-N
C     IF THIS IS A 1ST CALL - INITIALISE TO 1ST VALUE
      IF(NSCH(I,1).NE.0)GO TO 10
C     INITIALISE TO 1,2,3,... J
      DO 5 K=1,J
      NSCH(I,K)=K
    5 CONTINUE
    6 RETURN
   10 IF(NSCH(I,J).LT.N)11,12
   11 NSCH(I,J)=NSCH(I,J)+1
      RETURN
   12 K=1
      IF(J.GT.1)GOTO 13
      NSCH(I,1)=1    $    RETURN
   13 IF(NSCH(I,J-K).LT.(N-K))14,18
   14 NSCH(I,J-K)=NSCH(I,J-K)+1
   15 K=K-1    $    IF(K.LT.0)GO TO 6
      NSCH(I,J-K)=NSCH(I,J-K-1)+1
```

```
      GO TO 15
   18 K=K+1    $    GO TO 13                              ....(#28)....
      END

      SUBROUTINE TAGON(I,K)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER GIVE
C     TAG ON A ¬ OR A < TO THE FRONT OF EXP-I DEPENDING ON WHETHER K=0
C     OR K .NE. 0 . SET I TO NEW CELL ADDRESS
      IF(K.NE.0)GO TO 10    $    J=GIVE(X)
      ICELL(J)=211000000B + LSHFT(I,9)
      ICELL(I)=ICELL(I)+J    $    ACELL(J)=-ACELL(I)
      BCELL(J)=-BCELL(I)     $    KCELL(J)=KCELL(I)    $    I=J
      RETURN
   10 J=GIVE(X)    $    ICELL(J)=214000000B+LSHFT(I,9)
      ICELL(I)=ICELL(I)+J    $    ACELL(J)=1/ACELL(I)
      BCELL(J)=1/BCELL(I)     $    KCELL(J)=KCELL(I)    $    I=J
      RETURN
      ENTRY RWFF
C     REMOVE WFF-I AND RETURN ITS LL IN K
      CALL WFF(I,J,K)
      ICELL(K)=ICELL(K).A.777000777B
      K=LL(I)
      ICELL(I)=ICELL(I).A.777 777 000B
C     PATCH-UP GAP
      IF(K.EQ.0)GO TO 11
      ICELL(K)=(ICELL(K).A.777 000 777B).O.LSHFT(J,9)
   11 IF(J.EQ.0)GOTO 20
      ICELL(J)=(ICELL(J).A.777 777 000B).O.K
   20 CONTINUE
      RETURN
      ENTRY ONE
C     RETURN A 0-LIST OR 1-LIST DEPENDING ON I (0 OR 1) AND RETURN
C     ITS ADDRESS IN K
      K=GIVE(X)   $  ACELL(K)=BCELL(K)=I   $  KCELL(K)=-1   $  ICELL(K)=0
      RETURN
      END

      SUBROUTINE DELETE(L)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      INTEGER RL,DA,FB,DF,FR
C     DELETE CELL-QUAD L FROM LIST
      IL=LL(L)    $    IR=RL(L)    $    CALL SAVE(L)
      IF(IL.EQ.0)GO TO 5
      ICELL(IL)=(ICELL(IL).A.777000777B).O.LSHFT(IR,9)
    5 IF(IR.EQ.0)RETURN
      ICELL(IR)=(ICELL(IR).A.777777000B).O.IL
      RETURN     $     END
      SUBROUTINE INSERT(L,M)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
C     INSERT QUAD-L BETWEEN QUAD -(M-1) AND QUAD-M
      IL=LL(M)
      ICELL(L)=(ICELL(L).A.777000000B).O.IL.O.LSHFT(M,9)
      ICELL(IL)=(ICELL(IL).A.777000777B).O.LSHFT(L,9)
      ICELL(M)=(ICELL(M).A.777777000B).O.L
      RETURN
      ENTRY  LINK
C     JOIN LIST-M TO THE RIGHT OF LIST-L
      CALL WFF(L,IL,IX)
      ICELL(IX)=(ICELL(IX).A.7770007778).O.LSHFT(M,9)
      ICELL(M)=(ICELL(M).A.777777000B).OR.IX
      RETURN
      ENTRY ADON
C     ADD-ON TO HEAD OF LIST-L CELL-QUAD-M AND SET L=M
      ICELL(M)=ICELL(M)+LSHFT(L,9)   $   ICELL(L)=ICELL(L)+M
      L=M    $     RETURN     $     END

      SUBROUTINE MANTISA(L)                              ....(#29)....
      INTEGER RL,DA,FB,DF,FR
C     EXTRACT MANTISSA A FROM EXP-L WHICH HAS ONE OF THE FORMS ↑AN,
C     ¬↑AN,<↑AN,¬<↑AN ETC AND SET L=ADDR. O  NEW LIST
    1 I=DF(L)
      IF((I.EQ.215B).O.(I.EQ.216B))GO TO 10
      IF((I.EQ.211B).O.(I.EQ.214B))GO TO 5
      PRINT 3,I
    3 FORMAT(/5X,*S/R MANTISSA - CALL BAD - I=*,I3/)
      STOP 4
    5 CALL DELETE(L)
      L=RL(L)    $     GO TO 1
C     LOCATE EXPONENT , DELETE IT AND ↑ OR ↓
   10 CALL ROP(L,I)    $    CALL DELETE(I)    $    CALL DELETE(L)
      L=RL(L)    $    RETURN    $    END
      SUBROUTINE ROP(I,J)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER RL,DA,FB,DF,FR
C     LOCATE RIGHT OPERAND OF OPERATOR ICELL-I AND RETURN IT IN J
C     GET REF INDEX
      J=I    $    I1=KCELL(J)
    1 J=RL(J)
      IF(KCELL(J).EQ.I1)RETURN
      GO TO 1
      END

      SUBROUTINE WELD(II,IZ,IQ,JQ)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      COMMON/NOV/KS(25),JA(2,10),NSCH(2,3),M(2),JP(2),KP1,KP2
      COMMON/F/IH,JH,KH,LH,MH,NH
      INTEGER GIVE,RANK
C     CREATE EXP FROM COMPONENTS AND DO HOUSEKEEPING
C     II=1 OR 2. IQ IS I1 OR I2. JQ RETURNED WITH ADDRESS OF NEW EXPR.
      KK=1    $    I=NSCH(II,KK)   $   I=JA(II,I)   $   JQ=I.A.777B
      CALL DETACH(JQ,M(II))
      IF(IZ.EQ.0)2,20
C     (+) - COMPS
    2 IF((I.A.10000B).EQ.0)GO TO 5
C     TAG ON A ¬
      CALL TAGON(JQ,0)
    5 IF(KK.EQ.IQ)GOTO 40
      KK=KK+1   $   I=NSCH(II,KK)   $   I=JA(II,I)   $   KQ=I.A.777B
      CALL DETACH(KQ,M(II))     $     CALL LINK(JQ,KQ)
      J=GIVE(X)    $    IF((I.A.10000B).NE.0)GO TO 10
      ACELL(J)=ACELL(JQ)+ACELL(KQ)
      BCELL(J)=BCELL(JQ)+BCELL(KQ)
      ICELL(J)=207000000B     $     GO TO 12
   10 ACELL(J)=ACELL(JQ)-ACELL(KQ)
      BCELL(J)=BCELL(JQ)-BCELL(KQ)
      ICELL(J)=210000000B
   12 CALL ADON(JQ,J)      $      GO TO 5
C     (*)-COMPS
   20 IF((I.A.10000B).EQ.0)GO TO 25
C     TAG ON A <
      CALL TAGON(JQ,1)
   25 IF(KK.EQ.IQ)GOTO 40
      KK=KK+1    $    I=NSCH(II,KK)   $   I=JA(II,I)    $    KQ=I.A.777B
      CALL DETACH(KQ,M(II))     $     CALL LINK(JQ,KQ)
      J=GIVE(X)
      IF((I.A.10000B).NE.0)GOTO 30
      ACELL(J)=ACELL(JQ)*ACELL(KQ)                       ....(#30)....
      BCELL(J)=BCELL(JQ)*BCELL(KQ)
      ICELL(J)=212000000B    $    GO TO 32
   30 ACELL(J)=ACELL(JQ)/ACELL(KQ)
      BCELL(J)=BCELL(JQ)/BCELL(KQ)
      ICELL(J)=213000000B
   32 CALL ADON(JQ,J)     $     GO TO 25
C     RESET INDICES
   40 CALL WFF(JQ,J,KK)
      KCELL(KK)=-1
   41 J=KK     $    KK=LL(KK)
      IF(KK.EQ.0)GO TO 44
      KCELL(KK)=KCELL(J) + RANK(KK)
      GOTO 41
   44 CONTINUE
      RETURN
      END


      SUBROUTINE TYPE(I,J)
      COMMON/LST/ICELL(511),ACELL(511),BCELL(511),KCELL(511),TPI,TPJ,
     XBPI,BPJ,INDEX
      INTEGER RL,DA,FB,DF,FR
C     RETURN THE TYPE NO OF EXP-I IN J
C      FOR NON-EXISTANT EXPRESSION
      COMMON/F/IH,JH,KH,LH,MH,NH
      I1=DF(I)
C     LOOK AT 1ST LEADING OPERATOR (IF ANY)
      IF(I1.GT.206B)GO TO 5
    2 J=17   $   RETURN
    5 IF((I1.NE.207B).A.(I1.NE.210B))GO TO 6
      J=9     $     RETURN
    6 IF((I1.NE.212B).A.(I1.NE.213B))GO TO 7
      J=13   $   RETURN
    7 IF(I1.NE.215B)GO TO 8
      J=1     $     RETURN
    8 IF(I1.NE.216B)GO TO 9
      J=5     $     RETURN
    9 IF(I1.EQ.211B)GO TO 20
      IF(I1.NE.214B)GO TO 2
      J=RL(I)    $    I1=DF(J)
C     LOOK AT 2ND OPERATOR OR OPERAND
      IF((I1.NE.207B).A.(I1.NE.210B))GO TO 10
      J=11         $    RETURN
   10 IF((I1.NE.212B).A.(I1.NE.213B))GO TO 11
      J=15    $    RETURN
   11 IF(I1.NE.215B)GO TO 12
      J=3    $    RETURN
   12 IF(I1.NE.216B)GO TO 2
      J=7     $     RETURN
*******
   20 J=RL(I)      $      I1=DF(J)
      IF((I1.NE.207B).A.(I1.NE.210B))GO TO 21
      J=10       $       RETURN
   21 IF((I1.NE.212B).A.(I1.NE.213B))GO TO 22
```

```
      J=14    $    RETURN
   22 IF(I1.NE.215B)GO TO 23
      J=2      $      RETURN
   23 IF(I1.NE.216B)GO TO 24
      J=6     $     RETURN
   24 IF(I1.NE.214B)GO TO 2
      J=RL(J)    $    I1=DF(J)
      IF((I1.NE.207B).A.(I1.NE.210B))GO TO 25            ....(#31)....
      J=12               $        RETURN
   25 IF((I1.NE.212B).A.(I1.NE.213B))GO TO 26
      J=16     $     RETURN
   26 IF(I1.NE.215B)GO TO 27
      J=4     $     RETURN
   27 IF(I1.NE.216B)GO TO 2
****   NOTE:  ABOVE 50 OR SO CARD VERY ERROR-PRONE CAUSE IM SLEEPY
      J=8     $     RETURN
C
C     IT IS ASSUMED THAT ¬ PRECEDES < WHERE THEY OCCUR TOGETHER
      ENTRY PWR
C     RETURN IN J THE EXPONENT OF THE LEADING ↑ OR ↓ OF EXP-I
      I1=I
   30 J=DF(I1)
      IF((J.EQ.215B).O.(J.EQ.216B))GO TO 35
      I1=RL(I1)    $    GO TO 30
   35 CALL ROP(I1,J)    $    J=ACELL(J)    $    RETURN
      END
```

DATA

```
 (SIN-COS)↑2=SIN↑2-2*SIN*COS+COS↑2=1-2*SIN*COS>
<(1-SIN↑2)*(1+TAN↑2)=COS↑2*SEC↑2=1>                      K. C. LEE
< COT↑4-CSC↑4=(COT↑2-CSC↑?)*(COT↑2+CSC↑2)=-1*(COT↑2+CSC↑2)=-1*(CSC↑2-1+CSC↑2)=
 1-2*CSC↑2   >
 <(1-SIN↑2)*CSC↑2=COS↑2*CSC↑2=COS↑2/SIN↑2=COT↑2 >
 <(TAN↑3-COT↑3)/(TAN-COT)=((TAN-COT)*(TAN↑2+COT↑2+TAN*COT))/(TAN-COT)=(TAN-COT)*
 (TAN↑2+COT↑2+1)/(TAN-COT)=TAN↑2+COT↑2+1 >
 <(TAN+SEC)/(TAN+SEC-COS)=(SIN/COS+1/COS)/(SIN/COS+1/COS-COS)=(SIN+1)/(SIN+SIN↑2
 )=(SIN+1)/(SIN*(SIN+1))=1/SIN=CSC >
 <(1-TAN)*(1-COT)=1-TAN-COT+TAN*COT=1-COT-TAN+1=2-COS/SIN-SIN/COS=2-(COS↑2+SIN↑2
 )/(SIN*COS)=2-1/(SIN*COS)=2-1/COS*1/SIN=2-SEC*CSC >
 <((1+TAN-SEC)/(SEC+TAN-1))/((1+SEC-TAN)/(SEC+TAN+1))=((1+TAN-SEC)*(SEC+TAN+1))/
 ((SEC+TAN-1)*(1+SEC-TAN))= ((1+TAN)↑2-SEC↑2)/(SEC↑2-(TAN-1)↑2) = (1+2*TAN+TAN↑2
 -SEC↑2)/(SEC↑2-TAN↑2+2*TAN-1) = 2*TAN/(2*TAN) = 1 >
<(1+SIN+COS)↑2 = 1+SIN↑2+COS↑2+?*SIN+2*COS+2*SIN*COS = 2+2*SIN+2*COS+2*SIN*COS=
 2*(1+SIN)*(1+COS) >
 <(1+SEC)/(SEC*TAN-2*SIN-TAN) = (1+SEC)/( SEC*TAN+TAN-2*TAN-2*SIN) = (1+SEC)/(SE
 C*TAN + TAN-2*SEC*SIN-2*SIN) = (1+SEC)/((1+SEC)*(TAN-2*SIN)) = 1/(TAN-2*SIN) >
 <(1+SEC)/(SEC*TAN-2*SIN-TAN)=(1+1/COS)/(1/COS*SIN/COS-2*SIN-SIN/COS) = ((COS+1)
 /COS)  /((SIN-2*SIN*COS↑?-SIN*COS)/COS↑2) = (COS+1)*COS↑2 /(COS*(SIN-2*SIN*COS↑
 2-SIN*COS))=(1+COS)*COS/(SIN*(1-2*COS↑2-COS))= (1+COS)/(SIN/COS*(1-2*COS)*(1+CO
 S)) =1/(TAN-2*SIN) >
 <CSC/(CSC-1) + CSC/(CSC+1)=(1/SIN)/(1/SIN-1)+(1/SIN)/(1/SIN+1) = 1/(1-SIN) + 1/
 (1+SIN)=(1+SIN+1-SIN)/(1-SIN↑2)=2/COS↑2=2*SEC↑2  >
< CSC/(COT+TAN)=(1/SIN)/(COS/SIN+SIN/COS)=                          (1/SIN)/((C
 OS↑2+SIN↑2)/(SIN*COS))=(1/SIN)*(SIN*COS)/(SIN↑2+COS↑2)=COS >
 <(SEC-TAN)/(SEC+TAN) = (SEC-TAN)*(SEC-TAN)/((SEC+TAN)*(SEC-TAN))=(SEC↑2-2*SEC*T
 AN +TAN↑2)/ (SEC↑2-TAN↑2)=SEC↑2-2*SEC*TAN+TAN↑2=1-?*SEC*TAN+2*TAN↑2>
 < TAN/(1-COT) + COT/(1-TAN)=(SIN/COS)/(1-COS/SIN)+(COS/SIN)/(1-SIN/COS)=SIN↑2/(
 COS*(SIN-COS))+COS↑2/(SIN*(COS-SIN))=(SIN↑3-COS↑3)/(COS*SIN*(SIN-COS))=(SIN-COS
 )*(SIN↑2+COS↑2+SIN*COS)/((SIN-COS)*SIN*COS)=(SIN↑2+COS↑2)/(SIN*COS)+(SIN*COS)/(
 SIN*COS)    =1/(COS*SIN)+1=SEC*CSC+1   >
 <(1+COT-CSC)*(1+TAN+SEC) = (1+COS/SIN-1/SIN)*(1+SIN/COS+1/COS)=((SIN+COS-1)/SIN
 )*((COS+SIN+1)/COS)=((SIN+COS)↑2-1)/(SIN*COS)=( SIN↑2+COS↑2+2*SIN*COS-1)/(SIN*C
 OS)= (1+2*SIN*COS-1)/(SIN*COS) = 2 >
```

BIBLIOGRAPHY

1. Anderson, R.H., "Syntax Directed Recognition of Hand-Printed 2-dimensional mathematics," Ph.D. Thesis, Harvard U., 1968.

2. Bell, F.H., "A Study of the Effectiveness of a Computer-Oriented Approach to Calculus," Ph.D. Thesis, Dissertation Abstracts International,Vol. 31, No. 3, 1096-A, Sept. 1970.

3. Birkhoff, G. & MacLane, S., "A Survey of Modern Algebra," 3rd Edn., MacMillan, N.Y., 1965.

4. Bobrow, D.G., "Natural Language Input for a Computer Problem Solving System," Ph.D. Thesis, MIT, 1964 (in Minsky [47]).

5. Borman, K.G., Report No. R-26 of the Computer Assisted Instruction Laboratory, The Pennsylvania State University, University Park, Pa., pp. 29-35.

6. Bruner, J.S., "The Act of Discovery," Harvard Educational Review, Vol. 3, pp. 21-32, 1961.

7. Butler, C.F., "CAI in New York City - Report on the First Year's Operation," Educational Technology, Oct. 1969, pp. 84-87.

8. Caviness, B.F., "On Cano~~m~~nical Forms and Simplification," Jnl. ACM, Vol. 17, No. 2, April 1970, pp. 385-396.

9. Chapin, J.R., "Problem Areas in Estimating Costs of Computer-Assisted Instruction," Proc. ACM Nat. Conf., 1968, pp. 111-116.

10. Communications of the ACM, Special Issue on Symbol Manipulation, Vol. 9, No. 8, Aug. 1966.

11. Control Data Corp., "6400/6600 Computer Systems Reference Manual," Pub. No. 60100000, 1966.

12. Control Data Corp., "6400/6500/6600 COMPASS Reference Manual," Pub. No. 60190900, 1967.

13. Coulson, J.E. (Ed), "Programmed Learning and Computer-Based Instruction," Wiley, N.Y., 1962.

14. Crowder, N.A., "Automatic Tutoring by Intrinsic Programming," reprinted in [41], pp. 286-298.

15. Culler, C.J. & Fried, B.D., "The TRW Two-Station On-line Scientific Computer : General Description," in "Computer Augmentation of Human Reasoning," Spartan Press, Washington, 1965.

16. De Cecco, J.P., "The Psychology of Learning and Instruction," Prentice-Hall, Englewood Cliffs, N.J., 1968.

17. Dunn, W.R. & Mulroyd, C., Aspects of Educational Technology, Vol. II, Methuen, April 1968.

18. Educational Technology, Vol. 9, June 1969, p. 2.

19. Engel, G.L., "Computer Assisted Instruction : A Selected Bibliography and KWIC Index," NWL Tech. Rep. TR-2283, Dahlgren, Va., 1969.

20. Ernst, G.W. & Newell, A., "GPS : A Case Study in Generality and Problem Solving," Academic Press, N.Y., 1969.

21. Feingold, S.L., "PLANIT - A Language for CAI," DATAMATION, Sept. 1968, pp. 41-47.

22. Feldhusen, J.F. & Szabo, M., "A Review of Developments in Computer-Assisted Instruction," Educational Technology, Vol. 9, No. 4, 1969, pp. 32-39.

23. Feurzeig, W., "New Instructional Potential of Information Technology," IEEE Transactions on Human Factors in Electrons, Vol. HFE-8, No. 2, June 1967, pp. 84-88.

24. Feurzeig, W. & Papert, S., "CAI Problems and Prospects," Proc. SJCC, 1969, pp. 613-616.

25. Frye, C.H., "CAI Languages : Capabilities and Applications," Datamation, Sept. 1968, pp. 34-37.

26. Fulton, W., "Algebraic Curves - An Introduction to Algebraic Geometry," W.A. Benjamin, New York, 1969.

27. Glaser, R., "Psychology and Educational Technology," Educational Technology, Vol. 6, No. 6, pp. 1-14.

28. Grubb, R.E., "Learner-Controlled Statistics," Programmed Learning, Jan. 1968, pp. 38-42.

29. Grubb, R.E. & Selfridge, L.D., "Computer Tutoring in Statistics," Computers and Automation, March 1964, pp. 20-26.

30. Hamblin, C.L., "Translation to and from Polish Notation," Computer Journal, Vol. 5, 1962 pp. 210-215.

31. Hansen, D.N. & Dick, W., "Semiannual Progress Report : January 1, 1967 through June 30, 1967," CAI Centre, Institute of Human Learning, Florida State University, Tallahassee, Florida.

32. Hesselbart, J.C., "FOIL - a File-Oriented Interpretive Language," Proc. ACM National Conference, 1968, pp. 93-98.

33. Hickey, A.E. (Ed), "Computer-Assisted Instruction Guide", Entelek Corp., Newburyport, Mass., 1968.

34. Hickey, A.E., "CAI in the States : A Status Report," in Aspects of Educational Technology, Vol. III [43], pp. 489-500.

35. Johnson, D.L. & Holden, A.D.C., "Computer Learning in Theorem Proving," IEEE Convention Records, Part 6, pp. 51-60. date

36. Kopstein, F.F. & Seidel, R.J., "Computer-Assisted Instruction vs Traditionally-Administered Instruction : Economics," Professional Paper 31-67, Human Resources Office, George Washington University, June 1967. see also AVCR 1968 ?

37. Lang, Serge, "Introduction to Algebraic Geometry," Interscience Tracts in Pure and Applied Mathematics, No. 5, Interscience, N.Y., 1958.

38. Leonard, J.M. & Wing, R.L., "Advantages of Using a Computer in Some Kinds of Educational Games," IEEE Transactions on Human Factors in Electronics, Vol. HFE-8, No. 2, June 1967, pp. 75-81.

39. Liu, C.L., "Introduction to Combinatorial Mathematics," McGraw-Hill, N.Y., 1968.

40. Love, W.P., "Individual vs Paired Learning of an Abstract Algebra Presented by CAI," Ph.D. Thesis, Dissertation Abstracts International, Vol. 31, No. 1, 246-A, July 1970.

41. Lumsdaine, A.A. & Glaser, R. (Ed), "Teaching Machines and Programmed Learning - A Source Book," Dept. of Audio-Visual Instruction, N.E.A. U.S.A., 1960.

42. MacDonald-Ross, M., "Programmed Learning - A Decade of Development," Int. Jnl. of Man-Machine Studies, Vol. I, No. 1, 1969, pp. 73-100.

43. Mann, A.P. & Brunstrom, C.K. (Eds), Aspects of Educational Technology, Vol. III, Pitman, London, 1969.

44. Martin, W.A., "Symbolic/Mathematical Laboratory," Ph.D. Thesis, MAC-TR-36, MIT, Jan. 1967.

45. McCarthy, J., "Computer Programs for Checking Mathematical Proofs," Proc. Symp. in Pure Mathematics, On Recursive Function Theory, Vol. 5 (American Maths. Soc.), pp. 219-227. date

46. McCoy, N.H., "Rings and Ideals," The Carus Mathematical Monographs No. 8, The Mathematical Association of America, 1948.

47. Minsky, M.L. (Ed), "Semantic Information Processing," MIT Press, Cambridge, Mass., 1968.

48. Minsky, M.L., "Steps towards Artificial Intelligence," Proc. I.R.E., Vol. 49, pp. 8-30, 1961.

49. Moses, J., "Symbolic Integration," unpublished Ph.D. Thesis, MIT, Cambridge, Mass., 1967.

50. Nelson, R.J., "Introduction to Automata," Wiley, N.Y. 1968.

51. Newell, A. et al., "Report on a General Problem Solving Program," Proc. Int. Conf. on Information Processing, UNESCO, PARIS, Butterworth, London, 1959, pp. 256-264.

52. Oldehoeft, A.E., "Computer Assisted Instruction in Teaching Numerical Methods," Ph.D. Thesis, Purdue University, Lafayette, Indiana, 1970.

See also CACM 1971

53. Oliver, P. & Brooks, F.P. Jr., "Evaluation of an Interactive Display Systems for Teaching Numerical Analysis," Proc. FJCC, 1969, pp. 525-533.

54. Ovenstone, J.A., "Computer-Assisted Instruction in Undergraduate and Postgraduate Medicine," The Medical Journal of Australia, Vol. 2, No. 10, Sept. 1966, pp. 487-492.

55. Perry, P.G. & Lee, K.C., "A Computer-Assisted Instruction System," Proc. of the Fourth Australian Computer Conference, Adelaide, 1969, pp. 401-406.

56. Potter, R.J., "Computer-Assisted Instruction," unpublished internal document describing interface and terminal equipment for the University of Adelaide CAI Project.

57. Quinlan, J.R. & Hunt, E.B., "A Formal Deductive Problem-Solving System," Journal of the ACM, Vol. 15, No. 4, pp. 625-646, 1968.

58. Raphael, B., "SIR : A Computer Program for Semantic Information Retrieval," Ph.D. Thesis, MIT, 1964 (also in Minsky [47]).

59. Rath, G.J., "The Development of Computer-Assisted Instruction," IEEE Transactions on Human Factors in Electronics, Vol. HFE-8, No. 2, June, 1967, pp. 60-63.

60. Rath, G.J. et al., "The IBM Research Centre Teaching Machine Project," in E.H. Galanter (Ed), "Automatic Teaching : The State of the Art," Wiley, N.Y., 1959.

61. Robinson, J.A., "A Review of Automatic Theorem Proving," Proc. Symp. in Appl. Math., Amer. Math. Soc., Providence, R.I., 1967.

62. Schramm, W., p. 99 in J.V. Edling et al., "Programmed Instruction," New York, Fund~~s~~ for the Advancement of Education, June 1964.

63. Shulman, L.S. & Keislar, E.R. (Ed), "Learning by Discovery," Rand McNally, Chicago.

64. Silvern, G.M. & Silvern, L.C., "Computer-Assisted Instruction : Specifications of Attributes for CAI Programs and Programmers," Proc. ACM National Meeting, 1966, pp. 57-62.5.

65. Skinner, B.F., "The Technology of Teaching," Appleton-Century-Crofts, New York, 1968.

66. Slagle, J.R., "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus," Jnl. ACM, Vol. 10, No. 4, pp. 507-520.

67. Suppes, P. et al., Progress Report in Computer Assisted Instruction for the period October 1, 1967 to December 31, 1967, Stanford University, 1968.

68. Suppes, P. & Jerman, M., "Computer Assisted Instruction at Stanford," Educational Technology, Vol. 9, No. 1, pp. 22-24.

69. Taylor, E.F., "The ELIZA Program : Conversational Tutorial," IEEE Convention Records, Part 10, 1967.

70. Uhr, L., "Teaching Machine Programs that Generate Problems as a Function of Interaction with Students," Proc. ACM National Conference, 1969, pp. 125-134.

71. Unwin, D. & Leedham, J. (Ed), Aspects of Educational Technology, Vol. I, Methuen, London, 1966.

72. Weizenbaum, J., "ELIZA - A Computer Program for the Study of Natural Language Communication between Man and Machine," Communications of the ACM, Vol. 9, No. 1, Jan. 1966.

73. Zinn, K.L., "Comparative Study of Languages for Programming Interactive Use of Computers in Instruction," EDUCOM Research Memorandum, RM-1469, Feb. 1969.

74. Zinn, K.L., "Instructional Uses of Interactive Computer Systems," DATAMATION, Sept. 1968, pp. 22-27.