UNDERSTANDING HAND-PRINTED ALGEBRA FOR COMPUTER TUTORING

by

Stephen Clark Purcell

B.S., Stanford University
(1973)

Submitted in Partial Fulfillment
of the Requirements for the
Degree of
MASTER OF SCIENCE
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
January 1977

Signature of Author Signature redacted
Department of Electrical Engineering, January 21, 1977

Certified by Signature redacted
Thesis Supervisor

Accepted by Signature redacted
Chairman, Department Committee

Understanding Hand-Printed Algebra for Computer Tutoring

by

Stephen Clark Purcell

Submitted to the Department of Electrical Engineering
on January 21, 1977 in partial fulfillment of the requirements
for the Degree of Master of Science

ABSTRACT

This thesis demonstrates how the use of a global context can improve the power of a local character recognizer. The global context considered is a computer tutor of high school algebra that observes a student working algebra problems on a graphics tablet. The tutoring system is integrated with a character recognizer to understand the pen strokes of an algebra solution coming from the computer tablet. A tablet based input understander for an algebra tutoring system is designed and implemented.

This thesis joins together two uses of a computer, intelligent tutoring and tablet communication. Natural communication with computers has been pursued through speech understanding, English text understanding, special purpose languages, hand printing and graphics. This work extends the power of hand-printing understanders by using more varied and higher level sources of knowledge than have been used previously.

Thesis Supervisor: Ira P. Goldstein, Assistant Professor of Electrical Engineering

TABLE OF CONTENTS

INTRODUCTION

This thesis demonstrates how the use of a global context greatly improves the power of a local character recognizer. The global context considered is a computer tutor of high school algebra that observes a student working algebra problems on a graphics tablet. The tutoring system is integrated with a character recognizer to understand the pen strokes of an algebra solution coming from the computer tablet.

This thesis joins together two interesting uses of a computer, inteiligent tutoring and tablet communication. Natural communication with computers has been pursued through speech understanding, English text understanding, special purpose languages, hand printing and graphics. This work extends the power of hand-printing understanders by using more varied and higher level sources of knowledge than have been used previously. The tablet is a unique medium for communicating in what could be called Chalkboard Languages.

Chalkboard Languages

Let us examine the Chalkboard Languages that make communicating with a tablet unique. By Chalkboard Languages, I mean the kind of communication that calls for a two-dimensional dynamic medium. Consider what is written on a blackboard at the end of a lecture or discussion and how difficult it can be to reconstruct the conversation. The missing element is the development, in time, of the symbols on the board. To watch the board all through a conversation is quite different than to see it statically at the end. The board is used to outline and elaborate, propose and modify, introduce and refer to ideas. Erasing, pointing and overscoring are uniquely possible. As a two dimensional medium it differs from spoken and written language, and resembles drawing and diagramming. There are extra degrees of freedom for expressing relationships between elements of the conversation. Just as many disciplines augment English with their own vocabulary, there are domain specific conventions for graphical expression. These conventions range from informal outlining and arrow drawing to more formal notations such as architecture or mathematics. Chalkboard Languages have the added property that they are media for communicating with oneself. People talk to themselves and write to themselves but the more natural ways to think about "half-baked ideas" are scribbling, outlining, diagramming, sketching, etc., all two-dimensional, dynamic processes, performed naturally on blackboards.

There are many examples of chalkboard languages. There are block diagrams that use arrows in trees and graphs to express hierarchy, communication paths, similarities, etc. Architectural sketching is a chalkboard language. Electronic circuit schematics have graphical symbols and two dimensional conventions. Formating programmer's code to show structures is two dimensional and idiosyncratic. Geometry is discussed and theorems proved with the help of dynamic diagrams. The outlining of ideas for a paper or talk sometimes becomes a two dimensional process with an interesting development in time. Likewise the proofreader's correction notation is a two dimensional language that depends heavily on pointing, circling and underlining. Maps embody a specialized two dimensional language. The notes that a reader makes in the margin of books and his underlines are a small language. A process is often described with diagrams which are changed to reflect the changing states of the system being described. There have even been programming languages developed around diagrams and flowcharts. Pygmailian [Smith 1975] is an example of such an iconic programming language, though the diagrams are built largely from pointing and menus. And last but hardly least (for me anyway) is the chalkboard language of algebra problem solving. Algebraic manipulations are naturally performed in two dimensions. In problem solving the rigid syntax of algebra is relaxed to allow fragments and overwriting to change subexpressions. In the tutoring situation there can be pointing, focus for emphasis and significant pauses. The algebra domain has a number of good features for a test case.

Overview of AICAI System Model

The computer tutor is a second focus of my thesis. I use a system of modules based on the work of [Brown 1975][Burton 1975][Goldstein 1977]. The block diagram in figure 1 shows the organization of an AICAI System (Artificially Intelligent Computer Aided Instruction), following [Goldstein 1977].

The expert module is central to this system organization, because an intelligent tutor needs to know about its subject matter. Recent work in CAI has incorporated experts into tutors for geography (Scholar [Carbonell 1970]), electronics (Sophie [Brown & Burton 1975]), set theory (Excheck [Smith et al, 1975]), arithmetic (West [Burton & Brown 1976]), planning and debugging [Goldstein & Miller 1976]. In Goldstein's organization, an expert can propose actions in the domain to be compared with the actions chosen by the student. The expert detects non-

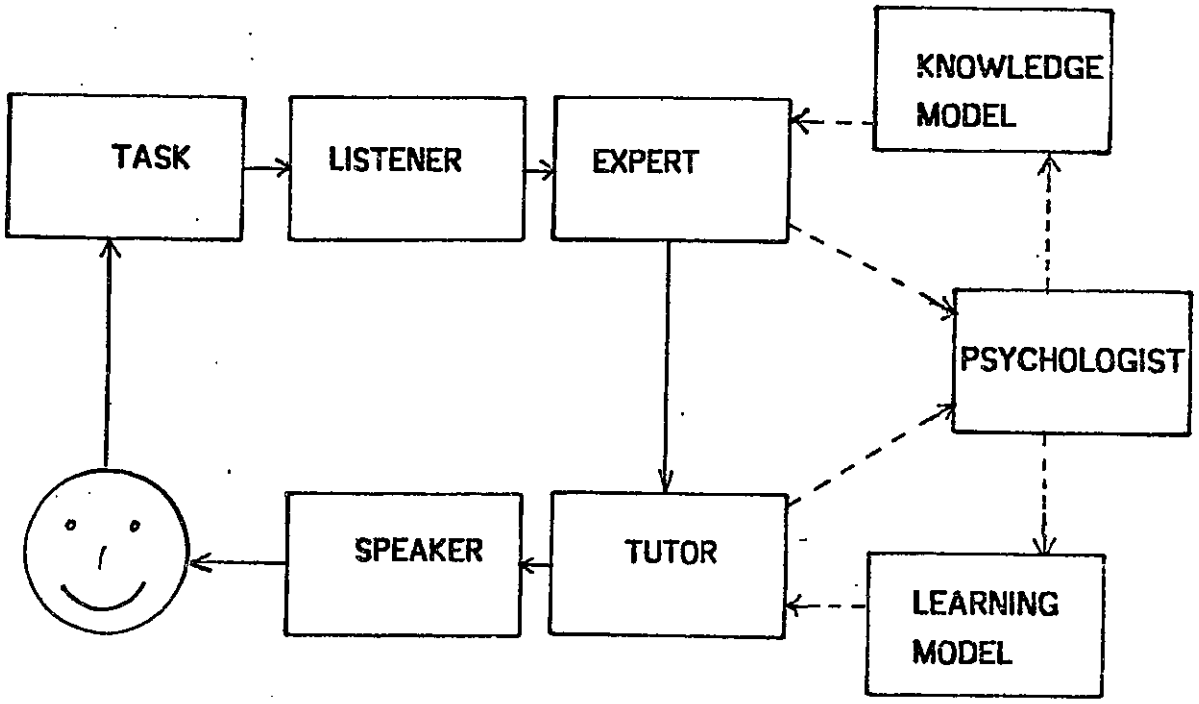


Figure 1. AICAI SYSTEM

optimal actions and reports them to the tutor for comment. Another function of the expert is to aid the input and output modules by answering semantic questions about the domain of discourse.

The Student Knowledge Model represents the student's current skill or what part of the expert's knowledge he has mastered. The knowledge state is represented by overlay modelling [Goldstein&Carr 1977], a technique that models the learner's skills as a subset of the expert's skill. Expert rules that he uses correctly become part of this model, while unused but appropriate rules do not. The tutor consults the model to restrict its remarks to be relevant.

The Psychologist is responsible for watching the student-tutor dialogue and keeping the student knowledge accurate. It associates with each expert rule its confidence that the student does or does not understand and use the rule.

The Tutor decides which problems to present and which issues to discuss or comment on. It decides what, when, and how to advise the student, based on information from the expert, the knowledge model and the learning model.

The Learning Model represents the student's prefered style of receiving advice and comments, which is a subset of the instruction styles available to the tutor.

The Input and Output modules are responsible for communicating with the student, enabling the tutor to instruct and the expert and tutor to follow his work. This module receives the most attention in this thesis.

It is the interaction of these modules that makes the tutor flexible, intelligent and responsive.

SPECIFIC DOMAIN

Algebra Tutoring

The chalkboard language that I concentrate on is elementary algebra problem solving. A student works an algebra problem on a graphics tablet, and the computer watches and reacts. Algebra is a good domain for study; it is bounded but not artificial, and it is rich enough to be interesting. The inclusion of the algebra tutor make this world all the more interesting and makes the communication more focused. This domain has the advantage that hand printing is more natural than cursive hand writing in algebra expressions. This property makes the lower-level character recognition easier, preventing the system from being front-end bound. The algebra conventions are fairly uniform between individual dialects, so a generally useful system is more feasible. A system for architecture sketching, for example, has to know a lot about individual styles [Negroponte 1975][Herot 1974]. Algebra is used both for the individual's problem solving and for communicating that process or result to others. Both of these modes are present in the algebra lesson environment and serve to enrich it. A conversation can be maintained between tutor and student. The possibilities for both long or short interactions allow various uses of discourse phenomena in the understander. In algebra the two dimensional nature of expressions is very important. The procedures being taught are invoked by cues that include 2-D relationships in the expressions. Cancelling rules, for example, are learned with reference to "above and below the fraction bar." The Algebra lesson is interesting for the many sources of knowledge that come into play.

One source of knowledge is the individual character's features. Another is the set of differential diagnostics to distinguish similar characters, i.e. knowledge of which features are most important to choosing one character over another to account for some input. Next the system knows the formal syntax for algebra, which expressions are well formed; for example, parentheses usually match. Also there is an informal syntax for algebra which has to do with such things as the ordering of subexpressions, the spacing and clustering of expressions. Numerical factors precede variables, variables are generally alphabetized, coefficients of one are dropped, etc.. Next, there is semantic reasonableness such as typical values for such things as exponents. There is, in fact, the whole semantics of algebra that underlies the expressions and procedures. The procedures of algebraic manipulation, themselves, are a powerful source of knowledge that determine which transformations are possible in one step (given, perhaps,

the student's level). Substitutions, for example, have a procedure; sometimes parentheses must be introduced in numerical substitutions and sometimes not. A higher-level source of potential knowledge is the discourse structure. The algebra session is goal oriented. There is coherence to the subexpressions throughout a problem session. Finally, the system can have a model of the student user. His strengths and weaknesses, his consistent bugs or conventions can help the system to understand his input. There are so many diverse sources of knowledge to help the system that the major challenge of this project is organizing them to work cooperatively.

Organization

In the AICAI system model, this research concentrates on the input module and part of the expert module. There are three major modules as shown in figure 2.

The first major module is the character recognizer, which communicates with the tablet and the parser. It receives a stream of pen co-ordinates from the tablet, collects them into strokes and collects the strokes into characters for the parser. The characters that it finds are organized and communicated in a chart [Kay 1967]. The chart is a lattice ordered by the arrival time of strokes from the tablet.

The recognizer learns its alphabet in a training session, and can easily be taught to distinguish 50 to 60 different characters. It can be taught an individual's idiosyncratic printing style or it can be trained on a generally universal character set. The recognizer can find multiple interpretations of ambiguous characters and will assign plausibility weights to the alternative interpretations. The characters it finds are next used by the parser.

The parser is the second major module of the system. It receives the chart of characters from the recognizer and builds a chart of phrases and finally one phrase for the expert. The character chart is ordered by the temporal sequence of strokes on the tablet, while the phrase chart is organized spatially by the two dimensions of the tablet.

The parser includes a grammar of algebra syntax that defines the structures to be discovered in the characters. The phrases that it produces are tree structured algebra expressions. Several structures are built in parallel, and there is a scheduler that allocates resources such as time and space to the alternative, growing interpretations. The scheduler is based on a system of potential plausibility scores for the partial theories and can be tuned to search in a depth first fashion or in a breadth first one or in somewhat both ways. The

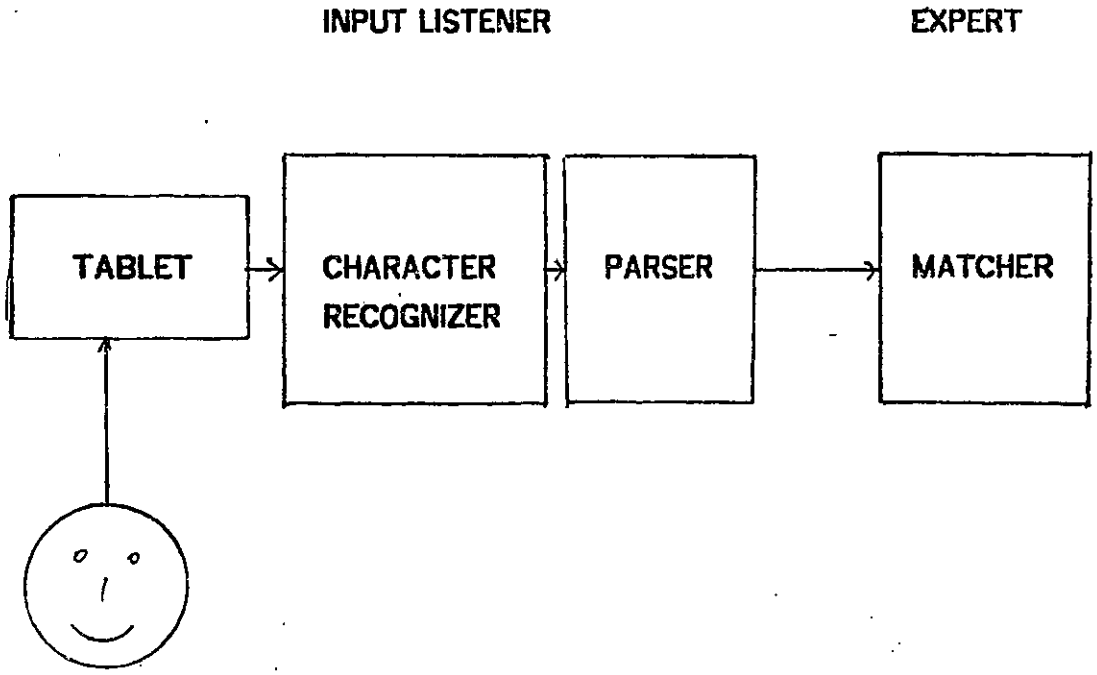


Figure 2. SUBSET OF AICAI SYSTEM

expressions found by the parser are matched by the expert to the expressions in the problem and in previous lines of the solution.

The expert is the third major module of the system and communicates with the parser and nominally with the tutor, which is not included in this work, but is discussed in [Brown et. al. 1975]. The expert takes the phrase chart from the parser and matches expressions to the previous context to discover the series of algebra transformations that the user has made, which is the output to the tutor. A secondary output is confirmation scores for the parser, which are found when subexpressions match the previous context. Also the expert can get ahead of the parser and predict expressions for the parser to verify.

The expert includes an expression matcher, simplifier, canonicallizer and transformer. It can ignore details by abstracting from the surface graphical representation of an expression to a more algebraically canonical form. So, the expert makes sense of the parsers output and intermediate outputs to guide the parsing and, in turn, the character recognition.

Figure 3 shows the hardware that supports the system. Half of the system runs in a dedicated IMLAC computer and half in a time shared PDP-10. The IMLAC is interfaced with a keyboard, a vector display screen, and a COMPUTEK graphics tablet. The processing power of the IMLAC is used to track the pen, maintain the display and extract features from the pen strokes. Most of the system is written in INTERLISP [Teitleman 1975] and runs under the Tenex operating system on the PDP-10.

The above major logical modules of the system will be discussed in further detail in the chapter called SYSTEM ORGANIZATION. Having seen the hardware and software organization, we will now look at an example of the problem solving that the system is designed to understand.

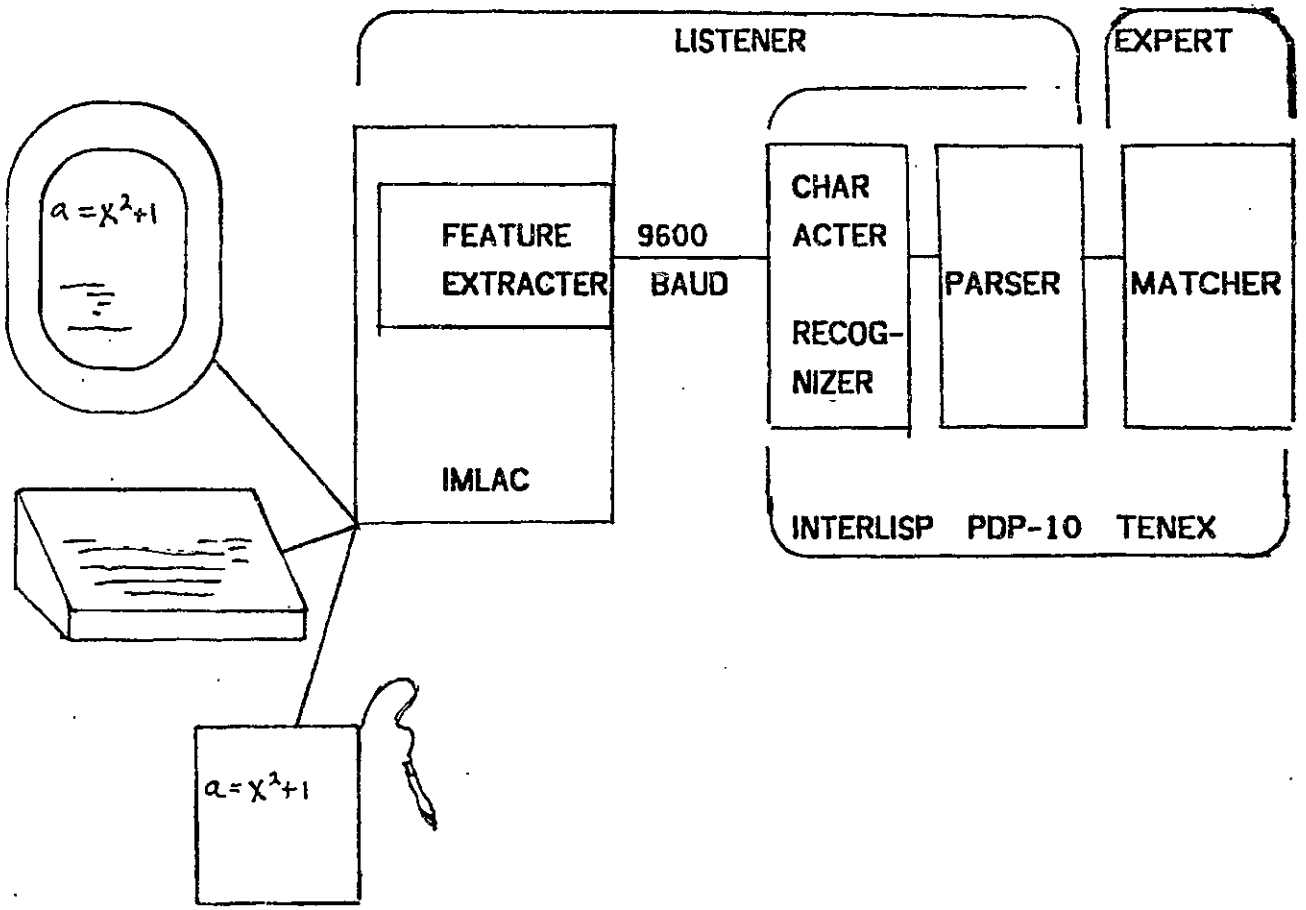


Figure 3. HARDWARE

SCENARIO

What follows is an example of input for the system to understand. The system's output is printed in capitals, and my commentary is written in italics.

PLEASE REDUCE:

{A -:- [(2A-3B)(2A-:-3B)]} + 1/2

YOU CAN WRITE ON THE TABLET NOW.

1.> $A \times \frac{1}{(2A-3B)\left(\frac{2a}{3b}\right)} + \frac{1}{2}$

OK.
DIVISION GOES TO MULTIPLICATION.
DIVISION IS CHANGED TO FRACTION.
UPPER CASE CHANGED TO LOWER.
REMOVAL OF SQUARE BRACKETS.

Note that the system must be aware of common notational variations and ambiguities, such as lower and upper cases for literals. The system must recognize when "x" is used for multiplication and when it is used as a variable. This difference, of course, cannot be detected locally. Here the "(2" closely resembles the letter "R". The local features of these strokes are consistent with either interpretation and can confuse the recognizer. The ambiguity is resolved by more global constraints. For example, the following ")" must be matched; under the "R" interpretation there would be no matching "(". The choice would be reinforced by matching the input to a transformation of the problem statement.

2> $\dfrac{A}{(2A-3B)\frac{2a}{3b}}$

OK.
MULTIPLICATION OF FACTOR AND FRACTION.
REMOVAL OF PARENTHESES.
UPPER CASE CHANGED TO LOWER.
ONLY PART OF EXPRESSION IS WORKED ON.

The student has only transformed a subexpression that his attention is focused on, dropping the "1/2". The difference locally between a "2" and a "Z" is only the sharp upper corner of the "Z". When context otherwise discriminates, this feature need not be drawn carefully; on the other hand this feature might be the only clue to an important difference and then the feature must be reliably detectable. Part of the expression could be interpreted as $\underline{(2A-3B)^2}$. The local parser will build this theory but not extend it.

3> $\dfrac{A}{\left(\frac{4a^2-6ab}{3b}\right)} + \dfrac{1}{2}$

OK.
MULTIPLICATION OF FACTOR AND FRACTION.
DISTRIBUTION OF FACTOR OVER SUM.

Here we pick up the "1/2" that was dropped last line. Notice the stroke segmentation problem that can occur above. If the "1" overlaps the fraction bar "-" it could be mistaken for a "+" or an "L". It is not enough to cluster strokes into characters solely on the basis of overlap.

4> $\dfrac{\dfrac{A}{2a(a-3b)}}{3b} + \dfrac{1}{Z}$

OOPS.
HOW DOES $4a^2-6ab$ GO TO $2a(a-3b)$???
OR $4a^2$ GO TO $(2a)(a)$???
ATTEMPTED FACTORING OF SUM.
PLEASE TRY AGAIN.

The best match between this expression and the last one, breaks down where the student incorrectly factors. In the subexpression "3b", if the vertical alignment of the characters is careless, then the subexpression without context could mean exponentiation. Since the context of the previous expressions rules this out, the system must tolerate a "b" written above to the right of the "3" and make the right interpretation, ignoring local information to satisfy more global considerations. The expert may even want to propose the discrepancy as a student error, but the tutor would probably classify it as careless and not fundamental, unless it were re-occurring.

5> $\dfrac{\dfrac{A}{2a(2a-3b)}}{3b} + \dfrac{1}{2}$

OK.
FACTORING SUM.

A context free parser would have much difficulty interpretting the double fraction bars. Both fraction bars are of equal width, but one must be subordinate to the other (the possible meanings are quite different -- division is not associative). Context will easily help out here.

6> $\frac{3b}{2(2a-3b)} + \frac{1}{2}$

OK.
CANCELLING FACTORS.
REMOVING DOUBLE RECIPROCAL.

There are many handprinted characters that closely resemble each other. The "b" resembles a "6"; the "a" and "u" resemble each other. Likewise, "2" and "Z", "I" and ")" etc.

7> $\frac{3b}{2(2a-3b)} + \frac{2a-3b}{2(2a-3b)}$

OK.
INTRODUCING FACTORS.

8> $\frac{3b+(2a-3b)}{2(2a-3b)}$

OK.
ADD FRACTIONS (LIKE DENOMINATOR).

9> $\frac{\cancel{2}a}{\cancel{2}(2a-3b)}$

OK.
CANCEL TERMS.

Cancellation does not fit well into a tree structured parse. It links separate 'lower' branches of the parse tree. Also it is intimately tied to the two dimensional layout of the expression. In a linear text representation of the expression, cancellation would be much less convenient.

SYSTEM ORGANIZATION

Recognizer

The block diagram in Figure 4 shows the major submodules of the recognizer.

Tablet

A computer graphics tablet is the student's medium to communicate his work to the System. An algebra problem may be worked out line by line on the tablet just as it would be solved on paper. The System follows the motions of the pen and thereby watches the student work. The pen actually writes on paper over the tablet so the student can see his work directly, or he can watch his writing traced on the display screen. Watching the screen assures the student that the system is following the pen correctly. Also if the screen outputs information the student will see it conveniently. Most people find it easy to draw on the tablet and watch the screen. The parallelism is usually learned in about 20 minutes of practice, [Bernstein](adapting to the SRI/Xerox mouse requires similar training). The Computek tablet does not have the most desirable properties for printing. The best tablet and pen combination is one with: (1) low pressure required to depress the pen, (2) even lower pressure to keep it depressed (hysteresis for a click feeling), (3) short travel to depress the pen: about 1/16 inch and (4) high friction between the pen and tablet. The Computek pen fails in pressure and travel, having too much of each. The friction between its pencil lead and paper is adequate. We have begun tests with a Summa-Graphics tablet which seems better in all these categories. Experience with the tablet overcomes these obstacles largely, but when a novice uses the system the poor data is very difficult for the System to cope with.

The tablet is directly connected to the IMLAC display processor, which extracts features from the pen strokes and sends them to LISP on the PDP-10. The IMLAC samples the pen location about 60 times per second and maintains an image of the strokes on its display. A switch in the pen detects when the tip is pressed to the tablet for writing. The signal from this switch defines the start and end of each pen stroke. The feature extraction and character segmentation is organized around the data in each stroke from pen down to pen up. Each stroke is gathered from the tablet; its features are computed and sent along to the character recognizer.

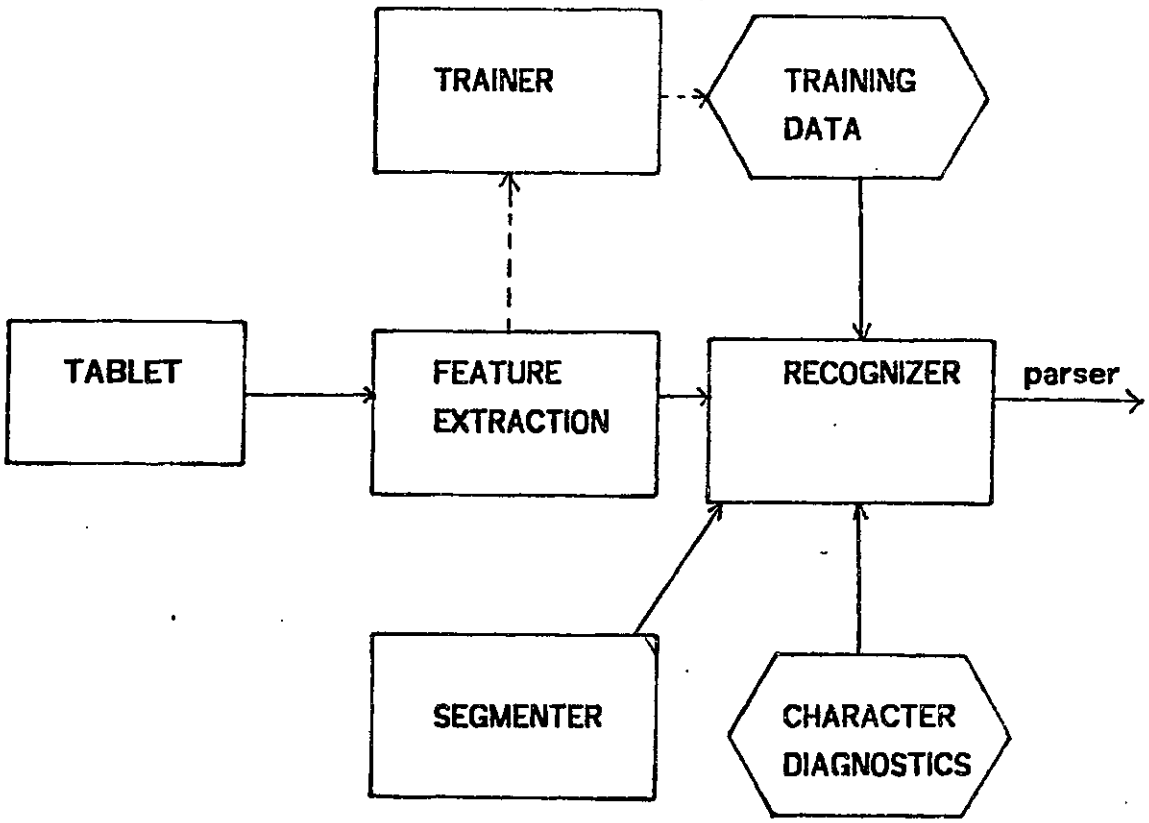


Figure 4. CHARACTER RECOGNITION

Feature Extracter

The character recognizer suggests interpretations for the strokes based on their features and mutual positions. The vector of features computed in the IMLAC for each stroke is: (XMIN XCTR XMAX YMIN YCTR YMAX S1 S2 S3 S4 VISITS CORNERS START END TOTAL) as shown in Figure 5.

The individual features are as follows:

XMIN,XMAX,YMIN,YMAX are the horizontal and vertical boundaries of the stroke, that is its enclosing rectangle.

XCTR,YCTR are the coordinates of the stroke's center.

S1,S2,S3,S4 are boundry crossing counts. The enclosing rectangle is divided into thirds horizontally and also vertically. The four interior boundaries (like a tic tack toe board) generate nine subregions. Each feature is a count of the times the stroke crosses one of these four boundaries. The crossing count is limited to three bits and the side of the boundary that the stroke started on is encoded in a fourth bit. Each Sn, then, ranges from 0 to 15.

VISITS is a nine bit binary integer that records the subregions visited by the stroke. Each bit indicates whether the stroke entered each of the nine regions.

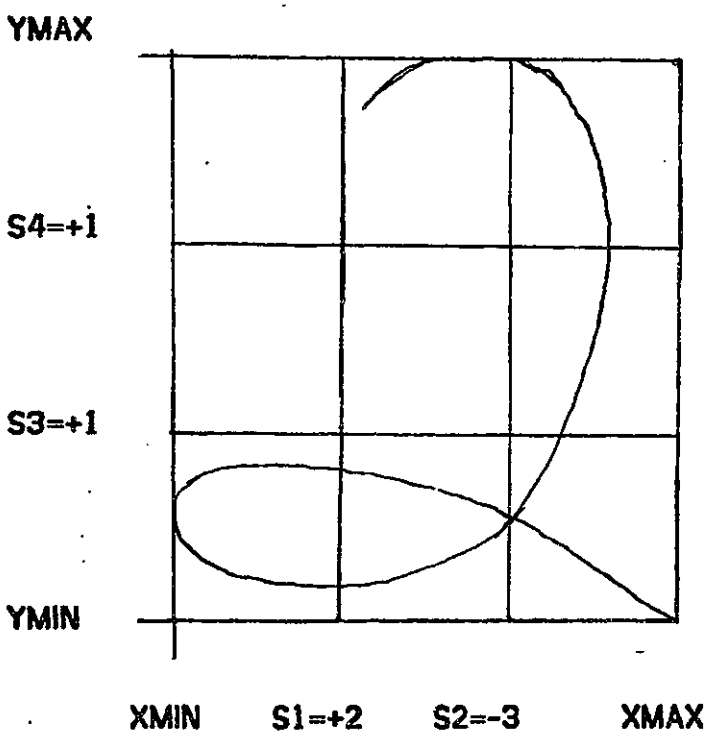
CORNERS is another nine bits to record the inflection points, or corners of the stroke. This information is used to distinguish for example a 2 from a Z. The curvature is calculated at each point in the stroke and compared to a threshold. When the curve is tighter the corner bit for the point's region is set. An improved algorithm could take into account time information, but the one implemented does not.

START,END are the directions in which the stroke travels near its start and end. There are 16 values that these directions can take. These features are important for distinguishing a C from a left parenthesis.

TOTAL is the total winding count of the stroke. This obscure feature was added when ones and twos were confused and the other features were not sufficient to separate them.

Trainer

The character recognizer learns its alphabet from examples that the user prints during a training phase. The trainer presents a menu of characters to be learned and accepts printed examples one character at a time. The feature extracter builds the same vector of



VISITS = 716_8 CORNERS = 0

Figure 5. FEATURE EXTRACTION ON STROKE

features that are used in the purely recognition phase and attempts to recognize the character. When the user corrects the recognizer, the new pairing of features and character is remembered. Since the trainer only accepts single characters it does not have the same segmentation problem that the recognizer has in continuous printing. Instead, a pause between strokes of about 1/2 second signals the end of a character. To relate the multiple strokes of a character, their centers are considered to trace a path in space much like the path of a stroke. This path is summarized like another, final stroke. In multiple stroke characters, then, each of the features named above is really a series of that feature with one instance per stroke that makes up the character.

The four series S1 to S4 are each filed into a tree structured discrimination net for efficient look up later. For the character pairs that are not well distinguished by the S1 to S4 features a table of diagnostics describes which other features are evidence for which characters. This diagnostic information is not subject to training, but there is no reason it couldn't be.

The recognizer can be tailored to one individual's printing, or it can be taught to accept a sort of super set of common printing styles. So far the training sets have been somewhere between a universal character set and the author's own style of printing.

Recognizer

The recognizer must take the features of the strokes in a proposed character and find the most likely interpretations of that collection of strokes. To propose interpretations, it uses the S1 to S4 features of the character strokes as an index into the tree structured dictionary formed during training. Each feature (or sequence of them in multi-stroke characters) suggests a set of characters. The characters that are suggested by most of the features are the best candidates for an interpretation. A character may fail to show exactly the same features as the trainer saw, because a nearest neighbor matching is used. Nevertheless, for a character to be recognized successfully, a similar one must have been seen in training. The similarity needed is a function of how well the features ignore small variations in character style and yet capture the differences between separate characters. The features used are admittedly ad hoc, and are not necessarily a model of the important features in a character as seen by a person. The recognizer produces a set of proposals and a confidence score for each.

Segmenter

The recognizer needs to know the group of strokes that constitutes a character. The system can reliably find the boundry between strokes by watching for the pen to be lifted, but the boundaries between groups of strokes that make up characters is not well marked. A recognizer that accepts continuous characters and strokes is significantly more difficult to construct than one which requires some signal between characters, such as a pause of 1/2 second or so.

The Segmenter's task is to find the group of strokes in each characters in conjunction with the recognizer. These groups are subject to several constraints. First, strokes must be written sequentially in time. Thus the recognizer can not understand i's and t's that are dotted or crossed after intervening characters. The grammar, however, can know about such constructions. Second, the strokes of one character usually touch each other. The exceptions such as '=' are accounted for in a table of inter stroke distances for the spread out characters. Second, the strokes must be grouped in such a way that all the groups can be interpreted. In particular, all the strokes must be accounted for. Within these constraints the segmenter may find several ways to group the strokes.

The multiple groupings form a lattice of characters, the character chart (Figure 6); and the parser must accept these possible groupings as its input. The character recognizer places its proposed characters in this lattice and the spatial relation specialist observes the lattice's implied alternatives. The scoring functions takes into account the mutual exclusion of alternatives in the lattice by normalizing the scores of alternatively competing interpretations to the best score of the group. More will be said about this normalization process in the section on the scheduler. The segmenter's alternatives are based on local information that the recognizer has about characters; the more global constraints of parsing an interpreting the input is expected to provide the constraints to arrive at the definitive interpretation for the segmentation and character recognition.

Character Diagnostics.

After the recognizer finds candidate interpretations based on its training session, this set of proposals is winnowed down by the use of differential diagnostics. The character diagnostic tells, for a pair of characters, which features are most reliable for choosing between

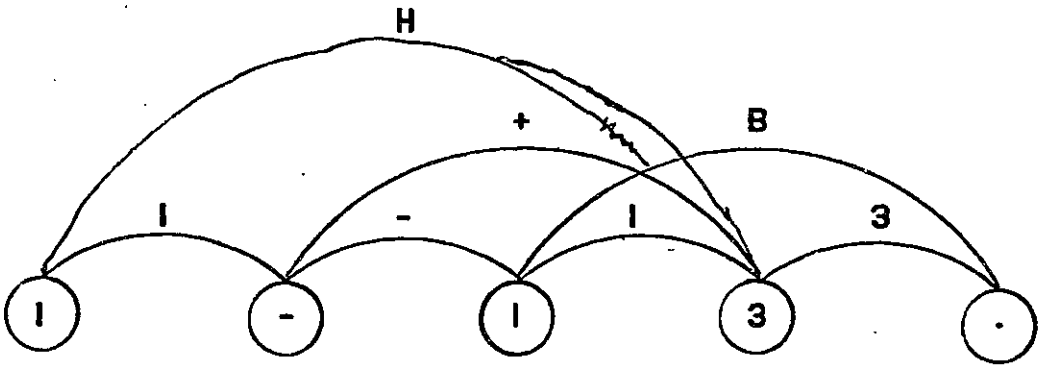


Figure 6. CHARACTER CHART

the pair. For example the knowledge that inflection points in certain areas of characters distinguish the 2 from the Z, or the U from the V. The diagnostics use a wider set of the extracted features than the trainer does. This component of the recognizer is more hand tailored to an a priori alphabet, and as such, complements, nicely the trainable character definitions.

Parser

The block diagram in Figure 7 shows the major submodules of the parser.

Spatial Specialist.

The spatial relationships between characters of an algebra expression express meaning that must be recognized (see Figure 8). Some operations of algebra are not even written with symbols but by spatial arrangement of their operators. Multiplication is denoted by concatenation, exponentiation by superscripts and indexing by subscripts. Even operations denoted by symbols expect their arguments to be found in certain spatial arrangements. Fractions are written vertically, and equations horizontally. In one dimensional languages, the possible concatenation relationships are reduced to two: left and right. A two dimensional language requires more relationships.

This algebra system is organized around nine relations: above, below, left-of, right-of, on, left-above, left-below, right-above and right-below (see Figure 9). There is a specialist that can compare any two characters (or phrases) and find one of these nine relationships. For each character all the other characters fall into nine sets relative to it. Actually only the closest neighbor in each set is likely to combine with it to become a larger phrase. The syntactic rules for algebra refer explicitly to these spatial relationships. In a grammar for a one dimensional language the rules implicitly specify the spatial arrangements. In fact the rules are written in one dimension and rely themselves, on one-dimensional ordering and concatenation just to be expressed. The spatial relations are maintained explicitly in the Spatial Network data structure for the parser and grammar to use.

When characters (or phrases) are recognized to be constituents of a larger phrase, the new phrase's neighbors must be made explicit. Most of the constituents neighbors are inherited by the new phrase. Of course the constituents themselves were related neighbors but they will not become neighbors of the new phrase. Since a phase's sub-constituents must be disjoint, a

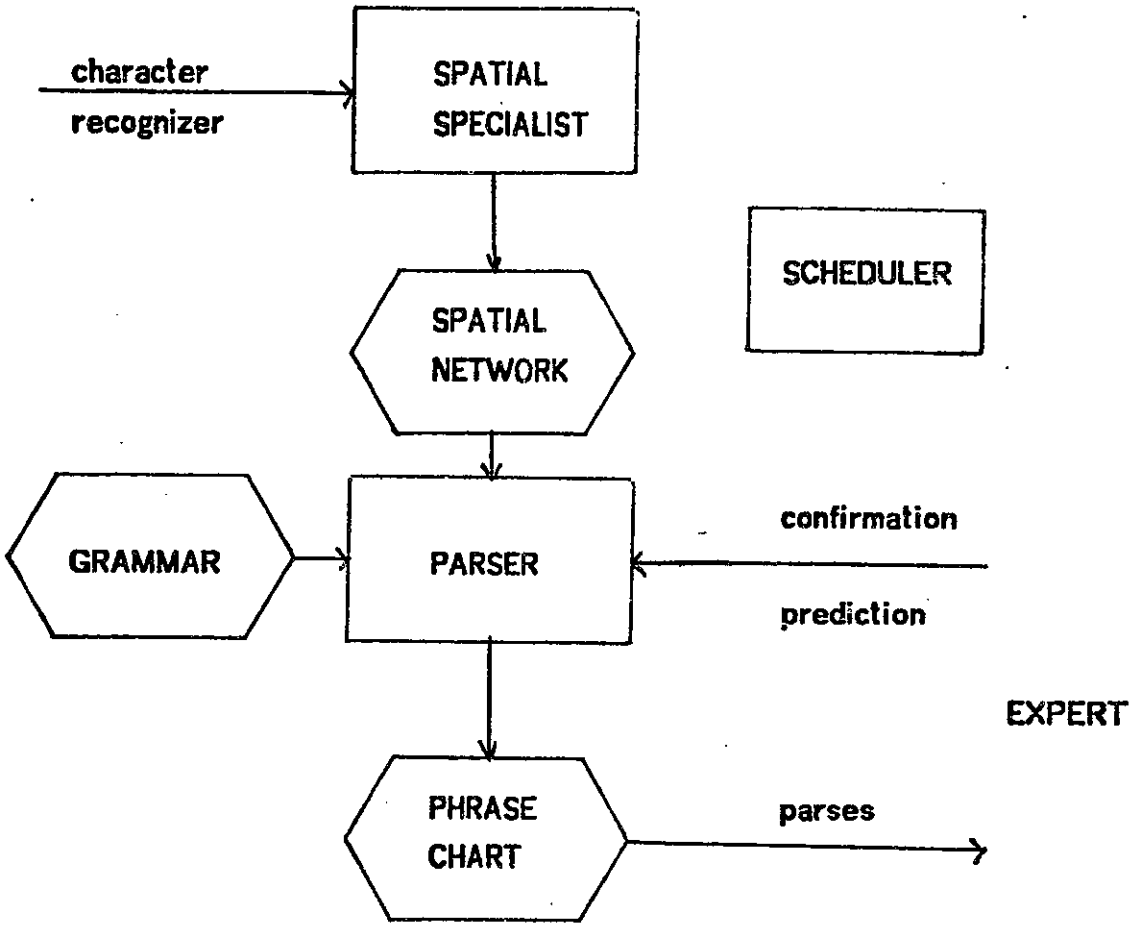


Figure 7. PARSER

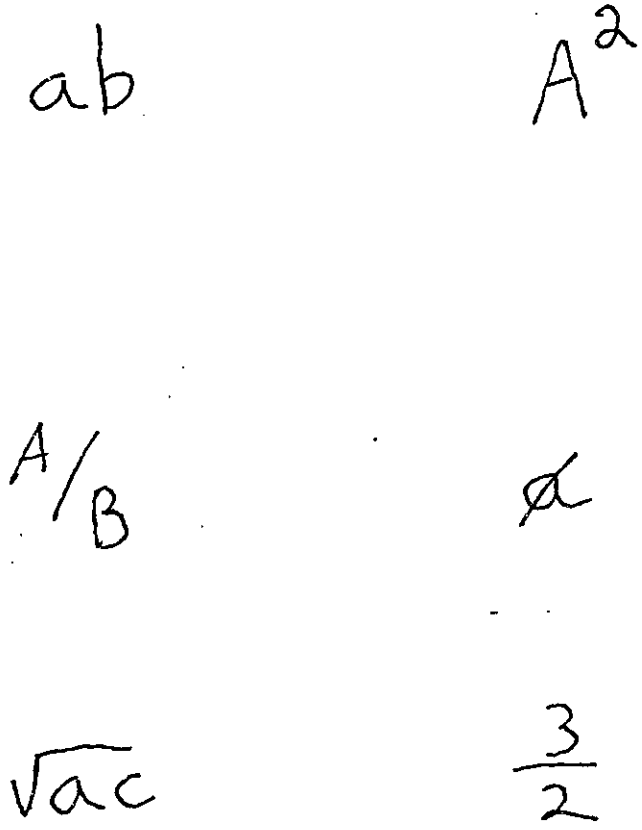


Figure 8. SPATIAL INFORMATION IN ALGEBRA

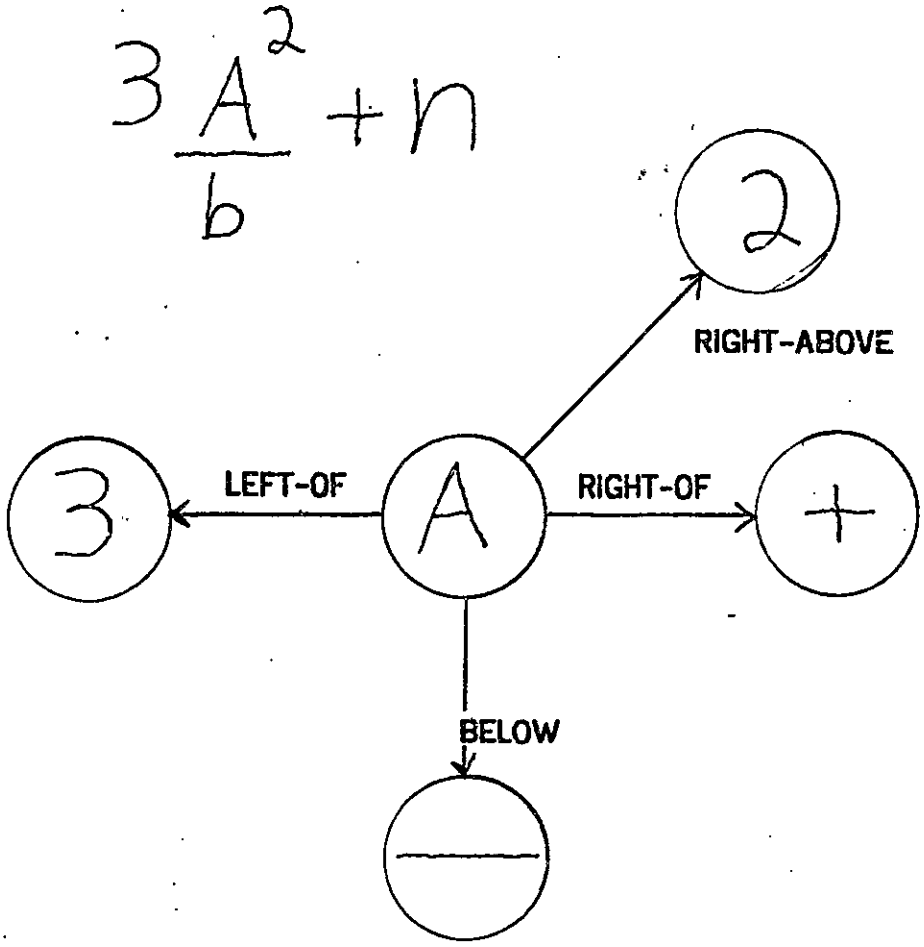


Figure 9. SPATIAL RELATIONS

phrase cannot combine with part of itself to make a new phrase. Since the extent of a phrase is defined to be the union of its constituents' extents, its constituents cannot be its neighbors. The other neighbors of its subphrases, however, can and often will be its new neighbors. If phrases concatenated vertically combine, then the neighbors above the higher one will be neighbors above the new phrase, and likewise below. A phrase that was a right neighbor to both phrases will be right related to the new phrase. A phrase that is a right neighbor to only one phrase is a trickier case; it will be a new right neighbor only if it is almost to the right of both. This inheritance of neighbors is central to parsing algebra and to discovering operator precedence.

Spatial Network

The relationships discovered by the specialist are held in the Spatial Network data structure to guide the parser. A relation connects two characters (or phrases) and the two sets of phrases built on them. New phrases are created across this boundry by matching a grammar rule to the relation type and to the phrase type of two phrases, one from each side of the relation. The phrases on either side of the boundry have plausibility scores, and the relation can estimate the plausibility of the best new phrase that it could find. All the relations in the network can be queued by their potential, and the search for new phrases can be organized around them.

The relations are represented as pairs of complementry relations each directed in an opposite direction. The relations manage the growth of new phrases at their site in such a way as to assure that any phrase is constructed only once. Otherwise the chart of phrases is polluted with costly duplications that will grow in parallel.

Spatial relations can be formed to a new phrase but more often the relations between characters support relations between higher phrases. The relations are sensitive to the lattice-structured chart produced by the character recognizer. The alternatives of the character chart are implicitly mutually exclusive. Since the spatial specialist looks for nearest neighbors it has to really look for nearest consistently existing neighbor. So the spatial network is grown between not just the correctly interpreted characters, but between all the alternative interpretations of the input strokes. Also since the system runs concurrently with user input, new characters can clobber previously closest neighbors. The spatial network is designed to be dynamically modifiable.

Phrase Chart

The recognized phrases are connected to their subphrases and superphrases in the Phrase Chart data structure. The phrase chart is like a well-formed substring table of Wood's ATN system. It is an extensional representation of the grammar, instantiated with phrases recognized from the input. It contains many partial parses that share common substructure. It consists of the linked data structures representing phrases, relations and other objects. The chart holds the state of many progressing parsing produced in a search that tries to extend the most promising interpretations.

The chart is formed from phrases and the objects that they are connected to. A phrase consists of:

TYPE:	e.g: number, expression, term
STROKES:	that part of the input accounted for by the phrase.
SUBPHRASES:	the immediate constituents.
SUBRELATION:	the relation between constituents.
SUPERPHRASES:	phrases built out of this one.
NEIGHBORS:	phrases that might combine with this one.
COORDINATES:	X-Y information: location, extent.
SCORE:	Plausibility or likelihood that this is the correct interpretation.
ALGEBRA:	Algebraic meaning.

Grammar of Algebra

The system's grammar is a generalization of augmented phrase structure grammars for one-dimensional languages, examples of which can be in [Heidorn 1975] and [Pratt 1973]. Each rule specifies a syntactic transformation from one pattern of phrases to another. The rules use the nine spatial relations explicitly to specify the two dimensional structure of the algebra language. There are non-terminal categories of phrases also mentioned in the rules such as: digit, letter, term, expression, etc. The rewrite part of a rule typically looks like:

<phrase type> ==> <phrase type> <spatial relation> <phrase type>

The rule also contains two expressions of scoring information, one to be evaluated before the application of the rule, which can check for preconditions, and one to be evaluated after the rule is applied and a new phrase is built. The rule also gives its algebraic meaning by

specifying how to generate the algebraic interpretation of the phrase in terms of the interpretations of its constituents. For example the rule that describes the construction of multiplication by horizontal concatenation will build the product of the new phrase's subphrases.

The nonterminal phrase categories form a hierarchy by inclusion. A digit can be a number, which can be an expression, etc. The grammar could have rules from each category to the next more general one. Instead these 'IS-A' relations are combined with the grammar to form a more expanded version, where every rule that held for expressions will hold for numbers as well. When the system is initialized, the grammar is compiled into a discrimination net and expanded to include the transitive closure of the 'IS-A' links. This expansion saves the construction of many redundant phrases at runtime.

Parser

The parser matches the tablet input to some phrase-structured tree producible by the grammar. It builds many trees in parallel until one can be extended to account for the entire input, or failing that, it trys to cover the input with only a few trees. The half-built trees are held in the Phrase Chart data structure, while the parser works at extending the phrases with the highest scores. Neighboring phrases in the chart may combine into a new phrase according to the grammar rules. The spatial specialist forms triples of neighboring phrase types and their spatial relationship to use as an index into the grammar. When the triple matches a grammar rule, the parser evaluates the pre-application scoring expression and queues a task with the resulting priority to build a new phrase. When the phrase construction task runs, it adds the new phrase to the chart data structure, finds the new phrase's neighbors and updates the relations that border the phrase. The search for new phrases can be organized by queuing phrases as tasks to look for neighbors to combine with. Or, as currently done, the relations may be queued as tasks which will take pairs of phrases from the phrase sets on either side of themselves and try the combination. From the scores of the phrases it touches, a relation can estimate the potential score of new phrases that it could create; this gives a priority to queue the relation task at. The algebra expert has the chance to see the growing expressions and to modify their plausibility scores as it sees fit. In this way the phrases that shouldn't extended should sink down the task list never to waste resources, and the good phrases should be extended until they become the parsing of the whole input.

Scheduler

The Scheduler controls the effort allocated to extending the competing theories in the parser's phrase chart. The system uses a numerical scoring system that is tenuously based on probabilities. Zero is the perfect score, and larger numeric scores indicate less plausible constructions. Negative scores are not used. When combining scores of a phrase's constituents, the size of each constituent is used to weight each score. In this way, scores behave as densities, and since they are combined additively they should correspond to log probabilities. When a set of mutually exhaustive alternatives are scored the scores are normallized to make the best alternative have score zero. In this way, the parser works on phrases that are locally implausible, but without better alternative; while good phrases that have as alternatives even better phrases are pursued less actively. This normalizing system follows one used at SRI [Paxton&Robinson 1975] and Woods' Shortfall scoring [Woods 1976]. Parsing can be viewed as search, and it is up to the scheduler to control that search.

Expert

The block diagram in Figure 10 shows the major submodules of the expert.

Abstracter

The expert uses a hierarchy of abstractions to represent algebra expressions. Each level of abstraction is like an equivalence class of expressions over stronger and stronger algebraic equivalences. Each class is represented by its canonical member. For example, the associativity of addition generates equivalence classes of sums, each of which can be represented by the left associating versions. These abstractions allow the expert to find corresponding expressions in a problem solution that differ only in the application of algebraic transformations. Sums that are equivalent by associativity have the same abstractions (Lisp EQ). The more powerful abstractions help the matcher bridge larger transformations of expressions. On the other hand, simple transformations may not have any effect on the abstractions. So if the system is to notice a simple change like removing parentheses, it must have a very literal representation of the user's expressions as well. When the student transforms an expression, the level of abstraction that best reflects the transformation is a measure of his sophistication. Beginners operate on the surface representation of algebra with

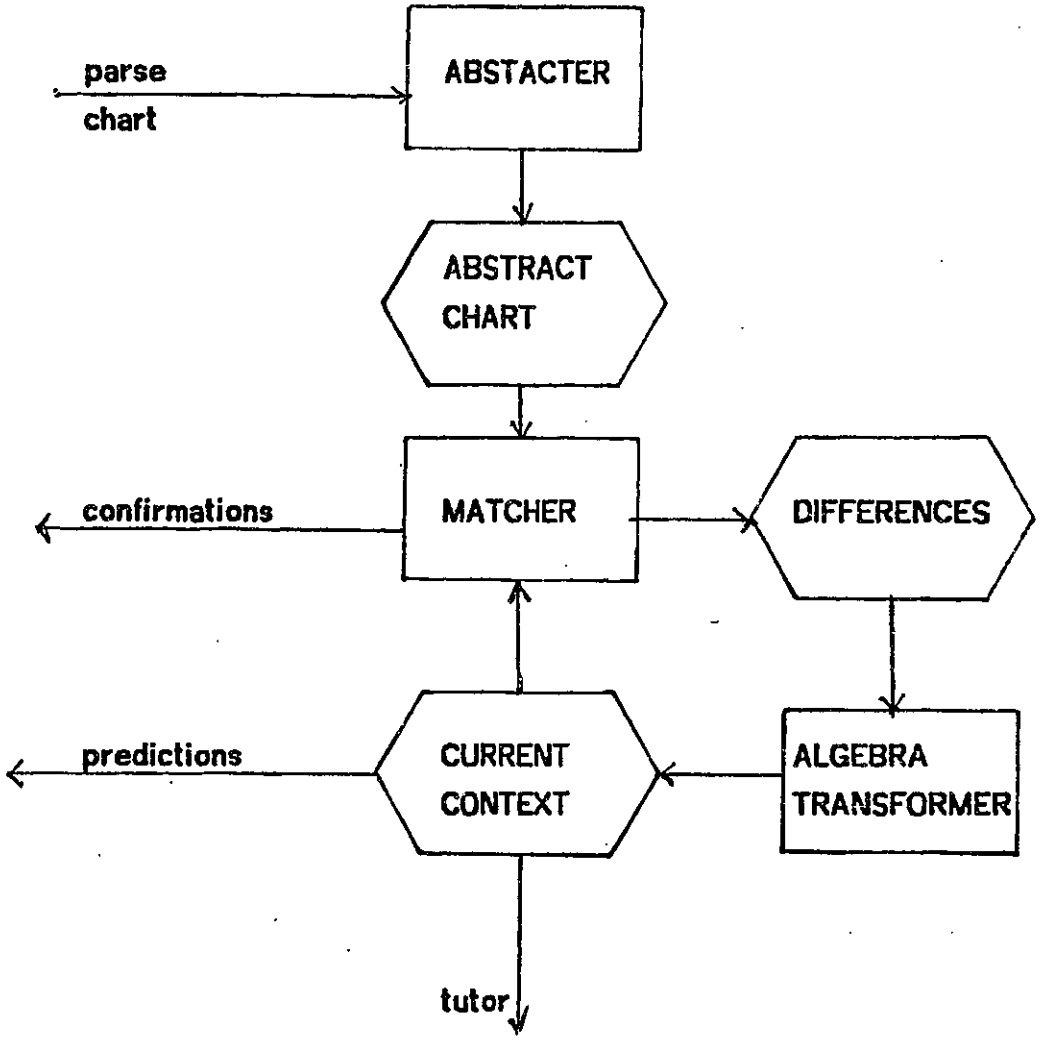


Figure 10. EXPERT

rules for moving signs and parentheses, while experts operate on more underlying trees of operations. For example, the expert combines like terms that may be alike not at the surface representation, but only in the expert's head after reordering factors. So, an algebra expert needs to see many layers of detail in an expression.

The matcher part of the expert tries to see the current line of algebra as some transformation of a previous line. To do so it matches expressions and subexpressions at various levels of abstraction. It acts a little like a theorem prover trying to achieve a goal (the current line) from the premises (the previous lines). It could find a series of steps at the surface representation level, but the sequence could be long and hard to find. Looking for a chain at more abstract levels is like using lemmas or finding islands in the proof to aim for. As an alternative to forward and backwards chaining, this method is really an intelligent middle out strategy. Abstractions are powerful aids to matching and searching.

What are the levels of abstraction? First, the lowest level is just the characters and spatial relations as the parser finds them. These objects and links are a very undigested representation of algebra. But they may be just the one used by beginning students. When a beginner misapplies transformations, the strongest invariants may be at this surface level, while the tree structure seen by an expert will undergo radical change when the illegal transformation is performed. The second level of representation is the phrase structured parse tree, just what's on the paper, but grouped correctly. This tree still represents artifacts of the external forms of algebra such as parentheses, small and capital letters that might be interchangable representations of one variable. Graphical variations for the same algebraic operations are still represented as this level. Multiplication can be expressed by 'x', by a dot or by concatenation. Fractions can be written horizontally or vertically. Next level is just the underlying algebraic variables and operations without regard to the graphical idiom. This level ignores parentheses because operators and arguments are represented unambiguously. But the order of arguments to commutative operators is still preserved. At the next level that arbitrary order is removed; as is the nesting of successive associative operators. The next deeper level simplifies the expressions as much as it conveniently can.

Complete simplification is too difficult and open ended in general [Moses 67]. Instead, the abstracter only applies conservative strategies to make expressions simpler. It applies basic arithmetic identities such as elimination of factors of one and terms of zero. It combines like terms and like factors; it expands certain products, and it factors certain sums.

There is no guarantee that all algebraically equivalent expressions will have the same canonical form, but many will. Algebraic equivalence, in its most general form is undecidable.

A more abstract level yet, is to evaluate the expression in some model. That is, the abstracter makes some random assignment of value to each variable, where instances of a variable all get the same assignment. Then the expression is evaluated for those values. There are technical details associated with the choice of domain for the variables and operations. With real numbers, roundoff and overflow are problems for any finite representation. A finite field such as the integers modulo a prime can be used, but division by zero and advanced functions like square root are troublesome. In any case this hash evaluation method is even more powerful than the simplifier for matching legally transformed expressions.

So far the abstractions have captured legal transformations at many levels, but what about illegal ones? The first answer is that they do, in fact, help catch mistakes by identifying the fragments of an expression that can be accounted for and matched. The remaining transformation, if it can't be accounted for legally, must be an illegal one. Once it is isolated it can be looked for in a table of common mistakes. Still, it would be nice to have an abstract level that bridges even illegal transformations. There would probably have to be a series of standard forms that would suggest two expressions were related. These forms would characterize features of expressions that probably would not change during most operations. The occurance of each (free) variable in the expression is such a feature, that will not disappear across most transformations. Certainly, new variables are not introduced often. A less reliable feature is the presence of a given operator. The power of abstractions for matching across mistakes begins to break down because equality at some level must be replaced by some more difficult nearest neighbor match. All in all the abstract levels greatly aid the matcher.

Abstract Chart

Corresponding to the phrases in the phrase chart are phrases in the Abstract Chart. An abstract version is associated with each syntactic phrase in the parser's chart. These abstractions are forced to be unique. When two expressions are equal in Lisp they must also be eq in Lisp. Equality is detected with a data base of expressions, with links from each expression to a list of all the larger expressions containing it. The equalizer uses this hash array of backpointers like Conniver does, to match lists by intersections. Each element of a

new expression is uniquized and then a previous occurance of that list is looked for in the intersection of the element's back pointers. The uniqueness of expressions helps to order terms of a sum uniquely. The abstracter and equalizer work hand in hand to give the expert many useful views of an expression.

Previous Context

Previous lines of an algebra solution provide a context for interpreting the latest line of input. All the lines of a problem solution should be connected by the application of algebra rules. Further, there is continuity between lines that can aid the recognizer and parser. The same variables should appear in the lines. Even larger subexpressions are often copied from line to line as other parts of expressions are modified. The system, then, can be guided by the previous context. For a simplification problem the context is the series of expressions leading to the solution. Likewise in solving equations the context is a series of transformed equations. The matcher will compare new input expressions with the previous context to judge their plausibility and to make predictions.

Matcher

The matcher tries to read between the lines of a solution. It compares the previous context to the current input, as algebraic expressions and as their associated abstractions. The matcher looks for syntactic congruence in the tree structures, and builds a list of corresponding fragments over some portion of the tree. Rather than match from the fringe of the expression tree up, all fragments of the tree-structured expressions are examined for matches. In particular all instances of the variable X in the first expression are tenatively matched to all other instances in the second expression. Likewise all additions are matched to all other additions. Then these seeds of a match are extended wherever connected structure continues to match. In this way the seed matches are extended over as much of the trees as possible.

Similarities

The areas that can be matched are the *similarities* between expressions. When there are alternative matchings, all but one should be eliminated. Matching at one layer of abstraction can guide the matching at other layers. The *similarity* is a tree structure like the matched expressions, but incomplete. It has loose ends where the match failed to continue.

These loose ends are the boundaries of the expression differences.

Differences

The differences are those unmatched portions of the trees, which may be loosely related by their connection to matched portions. The differences are evidence that an algebraic transformation has been applied. A difference can be characterized by the appearance or disappearance of operators or variables. When multiplication is distributed over addition, a multiplication operator appears as does a new copy of one factor. The features of the differences are used as an index to the rules of algebra.

Algebra Rules

The algebra rules allow the matcher to continue matching expressions that differ by algebraic operations. The rules must be organized according to the differences that the matcher will find in the *before* and *after* expression. When the rule is applied to the before expression, the result should match the after expression. A transformer makes the rule applications.

Transformer

The transformer applies the algebra rules to finish the work of the matcher. A successful match using a rule is evidence that the rule was used between the previous context line and the current context line. The expectation model of the current line can be transformed by the discovered rule and thereby reflect more accurately the best current expectation.

Current Context

The Current Context is a dynamically changing expression of what the system expects the user to be writing. As each line begins the current context is initialized to the previous context, giving rise to the expectation that the previous line will be repeated. As evidence accumulates that the current line is not merely a copy of the previous line but a transformation of that line, the current expectation is modified to reflect discovered transformations. At that point the current context may be a more accurate model of the user's input than the parser was able to discover by itself, and so it acts as top down prediction to

guide the parser.

It is a simplification to assume that any line will simply repeat the last line. A natural extension is to predict the transformations that the user will apply and let the current context reflect the expected transformation before any evidence is discovered. Of course the table of transformation rules acts like a more general expectation that some of those rules will be applied, but it is not as specific a prediction.

When the current context matches the parser's output then all transformations have been discovered and the understanding is complete. Dynamically, the current context mechanism acts like a servo-system with negative feedback. The matcher differences the input with the current context and provides feedback through the algebra rules that modifies that current context until it matches the input. The series of modifications defines how the input is related to the previous context.

SCENARIO REVISITED

Now that we have seen the system organization, let us return to our scenario for a closer look (refer to previous figures). In particular let's look at the student's second expression and see how the system processes it. The student wrote:

$$2> \quad \frac{A}{(2A-3B)\ \frac{2a}{3b}}$$

First the tablet will divide the input into 15 strokes and extract the features from each. The features that the Imlac will send to Lisp look like:

```
(((373 781 399 799 3 3 1 9 379 257 339))
          ((380 791 394 792 9 9 0 0 56 0 752))
          ((177 751 642 766 9 9 0 0 56 0 512))
          ((688 751 692 771 0 0 1 1 393 0 716))
          ((686 759 704 760 9 9 0 0 56 0 512))
          ((723 769 727 791 0 0 1 1 457 0 714))
          ((714 759 751 760 9 9 0 0 56 0 512))
          ((718 736 743 751 11 9 1 1 470 272 512))
          ((184 691 200 727 2 1 1 1 423 0 671))
          ((205 697 223 719 11 11 1 1 503 257 527))
          ((239 696 257 723 2 9 3 3 379 257 939))
          ((247 707 261 708 9 9 0 0 56 0 512))
          ((288 706 289 708 9 9 0 0 7 0 736))
          ((310 698 328 726 12 12 1 1 511 97 4))
          ((594 685 622 736 10 10 1 1 399 0 728))))
```

Each list holds the features for one stroke, as described above in feature extraction section.

From training sessions the recognizer has a character-set description, structured as four trees. These four trees are applied to the features S1 to S4 of groups of strokes, yielding character interpretations for the strokes, which are passed in a chart to the parser:

```
(PH037 b)
(PH035 3)
(PH033 -)
(PH031 A)
(PH029 a)
(PH027 2)
(PH025 %))
(PH023 b)
(PH021 3)
(PH019 -)
(PH017 a)
(PH015 2)
(PH014 Z)
(PH012 R)
(PH010 %()
(PH008 -)
(PH006 -)
(PH004 A)
(PH002 a)
```

Next the parser uses the algebra grammar to build phrases from the characters. Every subphrase that will eventually be part of the interpretation must be discovered, and along the way many side pathes will be explored. Many "obviously wrong" theories are discovered by this parser because it tries to parse every subset of the characters without regard to the characters outside that subset. Phrases are proposed in a context free way and then evaluated with respect to context. Some of the phrases built by the parser:

```
(PH084 (DIVIDE A (TIMES (PAREN (- (TIMES 2 a)
```

```
                                        (TIMES 3 b)))
                                (DIVIDE (TIMES 2 a)
                                        (TIMES 3 b)
[PH083 (TIMES (PAREN (- (TIMES 2 a)
                                (TIMES 3 b)))
                        (DIVIDE (TIMES 2 a)
                                (TIMES 3 b)
[PH082 (DIVIDE A (DIVIDE (TIMES 2 a)
                                 (TIMES 3 b)
(PH081 (DIVIDE (TIMES 2 a)
                        (TIMES 3 b)))
(PH080 (DIVIDE (TIMES 2 a)))
(PH079 (DIVIDE A (DIVIDE 2 b)))
(PH078 (DIVIDE A (TIMES (PAREN (- (TIMES 2 a)
                                        (TIMES 3 b)))
                                (DIVIDE 2 b)
(PH077 (TIMES (PAREN (- (TIMES 2 a)
                                (TIMES 3 b)))
                        (DIVIDE 2 b)))
(PH076 (DIVIDE A (TIMES (PAREN (- (TIMES 2 a)
                                        (TIMES 3 b)))
                                (DIVIDE 2 (TIMES 3 b)
(PH075 (TIMES (PAREN (- (TIMES 2 a)
                                (TIMES 3 b)))
                        (DIVIDE 2 (TIMES 3 b)
[PH074 (DIVIDE A (DIVIDE 2 (TIMES 3 b)
(PH073 (DIVIDE 2 b))
(PH072 (DIVIDE 2 (TIMES 3 b)))
```

Phrase 84 (PH084) is the complete parse which includes phrases PH083, PH081, and PH080, but not the others. Recall the five abstract levels for expressions; I use <u>*n.m*></u> for line *n* of the input at abstract level *m*. The Abstracter receives the phrase:

```
2.2>          (DIV-BAR A (TIMES (PAREN (- (TIMES 2 a)
                                          (TIMES 3 b)))
                                (DIV-BAR (TIMES 2 a)
                                         (TIMES 3 b]
```

It builds the other levels:

```
2.3>          (DIVIDE A (TIMES (- (TIMES 2 A)
                                  (TIMES 3 B)
                                (DIVIDE (TIMES 2 A)
                                        (TIMES 3 B]

2.4>          (DIVIDE A (TIMES (ADD (TIMES 2 A)
                                    (MINUS (TIMES 3 B)))
                                (DIVIDE (TIMES 2 A)
                                        (TIMES 3 B]

2.5>          (MUL (RAT 3 2)
                   B
                   (EXP (ADD (MUL 2 A)
                             (MUL -3 B))
                        -1)))
```

The student's next line was:

```
          A              1
3> ---------------   +  -
    / 4a^2 - 6ab \       2
   ( ---------- )
    \     3b     /
```

A major operation is performed between line 2 and line 3, a factor (2a) is distributed over a sum (2a-3b). Nevertheless the level 5 abstraction does not change. The expressions are not directly equal, due to the student's changing focus, but the matcher can easily see that line 2 is a subexpression in line 3. When the matcher tries to make this matching on level 4, it finds itself matching the product (2a-3b)(2a/3b) with the quotient $(4a^2-6ab)/3b$. The difference can be accounted for by the transformation "multiplication of factor and fraction." The resulting numerator would be (2a-3b)2a instead of $4a^2-6ab$. These are seen to be equal at level 5 but not at level 4. The transformation to account for the difference is "Distribution of factor over sum."

At abstract level 5 (canonically simplified) there are only two different expressions in the whole scenario (except line 2 focuses on a subexpression). Lines 1 to 6 reduce to:

```
        3b          1       (ADD (MUL (RAT 3 2)
>  ------------  +  -                  B
     2(2a-3b)       2                  (EXP (ADD (MUL 2 A)
                                                 (MUL -3 B))
                                            -1))
                                  (RAT 1 2))
```

While lines 7 to 9 reduce to:

```
       a          (MUL A
>>  -------              (EXP (ADD (MUL 2 A)
    2a-3b                        (MUL -3 B))
                           -1))
```

This example shows two algebraically equivalent expressions that have different semi-canonical representations. The step that the student took but the simplifier did not is the introduction of factors while adding fractions.

PREVIOUS AND RELATED WORK

This thesis builds on previous research in several areas, extending some and borrowing from others.

Recognizers for Tablets

My character recognizing module is similar to many recognizers previously built for tablets [Diamond 1957], [Bernstein 1969], [Teitleman 1963], (Ledeen in [Newman&Sproull 1973]). Researchers have found dynamic character recognition on tablets to be significantly easier than the more general problem of static character recognition through cameras or image scanners, especially in the hand printing domain. The time information available from the tablet makes stroke segmentation much easier. A number of schemes were developed to classify strokes and characters by significant features such as some description of shape. Diamond and Bernstein classified shape by local geometry, namely the sequence of directions that the stroke traveled in. At each interval of time or stroke length, the direction was quantized and added to a growing stroke description. This description would be matched against a dictionary of descriptions, which defined a character set. This relative stroke description failed to capture important features such as closed or intersecting. The difference between a small a and a small u is whether or not the character is closed at the top. The relative directions at each interval of the strokes are nearly the same (figure II).

To overcome this difficulty, some features of global geometry must be used. Teitleman, Groner and Ledeen computed global features of strokes by imposing grids of lines over characters which divided them into regions. The stroke could be viewed as the sequence of regions visited or as boundry-crossing features. Now features like closure could be described as starting in, and returning to the same region. The recognizer module that I use takes this global approach. The general shape of strokes is not enough to recognize characters. The u and v have similar shapes but are distinguished by the v's sharp point. The presence of points or corners is an important feature that early recognizers ignored. In the Ledeen recognizer, for example, the letter Z was crossed to distinguish it from the digit 2. My recognizer, like some of Bernstein's, looks for the inflection points as features. It uses only spatial information to find them, but velocity information can greatly aid the detection of corners [Negroponte 1975]. There is a tradeoff between using local descriptions and using global ones. Just as we saw the local descriptions fail at closure, the global descriptions fail to capture efficiently features that

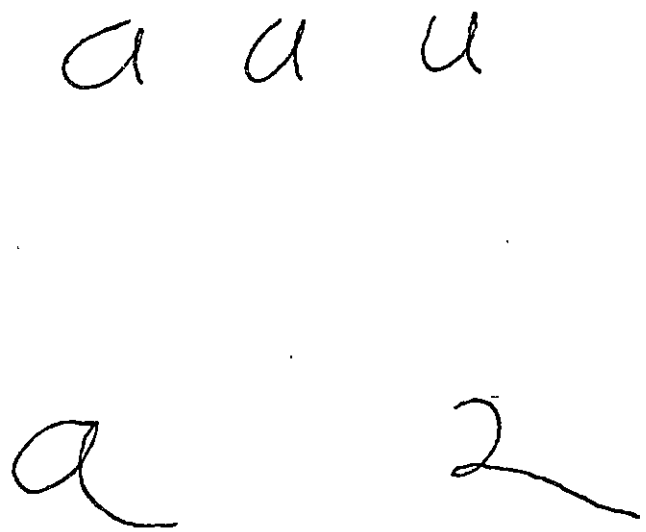


Figure 11. CHARACTER EXAMPLES

really are local. For example many characters can be printed with long tails at the end of a stroke (figure II). The letter a or the digit 2 may have long tails that will distract a global system from properly normalizing the size and position of the character before imposing the grid. When the grid is misaligned, the stroke will not be properly described, and recognition may fail. The tail can affect most all of the global features, yet in a local scheme only a very few of the features describe the tail. (Here again time information can be used to advantage. If the intervals are chosen by time, because it corresponds closely to importance, then the long tail which is most likely quickly executed, will not appear in as many 'syllables' of the description and not carry as much significance. The local method can thus filter out certain variations in style and execution which only confuse the global method. Probably the best recognizer would combine these methods, but mine does not, and I know of none that does.

There is a basic difference of emphasis between my system and previous recognizers. A common theme was to keep the systems as context free as possible, designing them to identify characters soley on local evidence, not to make second guesses and not to make mistakes. Systems that incorporated recognizers expected perfect characters from them, so the user was given immediate feedback to check constantly. My system frees the user from checking the local results of the recognizer by using all the global evidence and context that it can.

Algebra Systems

A number of two-dimensional parsers for algebra have been designed or built [Henderson 1968], [Anderson 1968], [Guertin 1971], [Martin 1971], [Bernstein 1971]. Henderson and Anderson view algebra as an instance of a two dimensional language which is a generalization of the one dimensional simple phrase structure language. The syntax of the language is a set of replacement rules that can generate any legal sentence from the start symbol by successively replacing phrase categories with phrases according to the replacement rules. Anderson parses an expression by performing this generation non-deterministically until he generates an expression to match the input, then the tree structure which is apparent in the generated expression is taken as the structure of the input expression, which was not apparent. This top down approach to parsing depends on strong predictions from the language's grammar, or gross inefficiencies arise. In fact for efficiency, Anderson designed another parser better tailored to algebra. The more efficient parser was not as committed to algebra's intrinsic two dimensions, but sought to reduce the input first to one dimension before parsing. A

preprocessor linearized the expression by inserting special characters to mark the two dimensional information. This linearization was done with as little information as possible and as locally as possible. Guertin, in his Matter system, and Martin also used linearizers before parsing. Unfortunately the linearizing choices are made with minimum evidence and are difficult to undo later, which tends to make these systems fragile and exacting. An example of the assumptions made by the linearizer is that neither the numerator nor denominator of a fraction will extend farther to the left than the fraction bar. This may usually hold, and users may easily adapt to the requirement, but such a system will not degrade gracefully. A two-dimensional feature that is hard to decide without context is the vertical alignment of sequential characters. If exponentiation is plausible, then the raised character is significant, if not then the user can be sloppier and the relative vertical positions is accidental.

Parsers

Much of my system is patterned after various speech understanding systems: HWIM at BBN [Woods 1976], HEARSAY at CMU [Lesser 1975], and one at SRI [Walker 1975]. All these systems try to assign meaning to user input in the face of uncertainty based on fallible knowledge sources. Algebra understanding is like a mini speech understanding project. It is more difficult than text, because the input is much less reliable and less constrained. Text parsers rely heavily on the small function words that cannot be reliably found in speech, to guide parsing. Algebra has fewer function symbols, less redundancy and has the uncertainty similar to speech. Speech is, of course, much harder because the information is so locally sparse. There are, at least in algebra, simpler methods that begin to give results. The previous algebra systems have gotten as far as they did with much less power than must be used to do speech. In borrowing from these natural language domains, I had to generalize them to two dimensions; but they did serve as guides.

The first similarity between these projects and mine is the use of multiple knowledge sources. All the speech systems use roughly the same sources: 1) Acoustics, 2) Phonetics, 3) Lexicon, 4) Verification, 5) Prosodics, 6) Syntax, 7) Semantics, 8) Pragmatics, and 9) Control. All of the systems organize the interaction of these knowledge sources around theories or partial interpretations of input. Each KS (Knowledge Source) can inspect a theory and extend it or criticize it. When a theory is extended enough to explain the whole input adequately then the system has done its job and has understood the input. The idea of a parser is usually

extended to include structure building in the domain of each KS. My system is like these ones when its parser builds parallel structures in the syntactic and semantic domains of algebra, guided by the high level context and the low level features. Character recognition is like phoneme and word identification.

A second similarity is that like my system, each of the others must manage partial theories and usually maintains several of them, working on them concurrently. Each system has a data structure and bookkeeping system to hold the theories in. SRI uses the Parse Net, a consumer and producer structure based on Kaplans GSP [Kaplan 1973]. CMU uses a Blackboard as a communication channel where KSs make proposals and criticisms. BBN uses a Well Formed Substring Table to hold completed phrases and a word-phoneme lattice for the Phonetics, the Segmentation, the Lexicon and the Verification. A common theme in all these data structures is to eliminate duplicated effort. A phrase should not be constructed twice for different purposes, if a phrase is discovered twice it should only be represented once. The phrases that are discovered by producers should be routed to the appropriate consumers. Phrases are filed in the data structure according to some characteristics such as position, phrase type, etc. This data structure serves as the market place for consumers and producers. My phrase chart organizes partial interpretations (the phrases) according to their position and neighboring phrases. The scheduler actively matches the mutually consuming phrases.

My system can be compared to PAZATN, an automatic protocol analyzer for elementary programming [Miller & Goldstein 1976b]. Both systems have synthetic grammars which can generate interpretations to be matched to an input. Like my phrase chart, their DATACHART holds the state of partially completed interpretations. Their PLANCHART serves a role similar to my expert's current context, providing expectations for the parser. PAZATN's preprocessor serves a function similar to my character recognizer; it classifies and locally processes input items. Both systems rely on a scheduler to conduct a "best first" coroutine search. There are parallels in the structure building of each: the :protocol (fringe) register corresponds to the strokes register in my phrases. :Title corresponds to name, :inputs to inputs, :plan to algebra expression. The systems differ in their choice of linguistic formalisms. As its name implies, PAZATN is based on the ATN formalism rather than on augmented context free rules. I also tried to use the ATN formalism but encountered difficulties that will be discussed later. PAZATN maintains Conniver contexts as it builds interpretations, while I have nothing similar. Miller and Goldstein have capitalized on the idea that linguistic parsing

methods can be generalized to many forms of analysis. In a sense I have made a similar generalization of parsing to analyzing algebra problem solving. To show its generality they have also applied the PAZATN system to a mathematical domain, symbolic integration. The algebra tutor could benefit greatly from their formalisms for planning and debugging [Miller & Goldstein 1976a].

Multiple Representations

The Expert module's use of abstractions takes advantage of multiple representations as many other AI programs do. MYCROFT [Goldstein 1974,1975] derives much of its power from the correspondences between turtle programs and an analytic model of geometry. Each domain uses its own representations and procedures, but the system has additional knowledge of mapping between these domains. Another example of mapping in and out of a model is the SOPHIE electronics tutor [Brown&Burton 1975]. A quantitative model (simulation) of a circuit is used to answer both quantitative and qualitative questions about the circuit and its possible faults. Many systems model their domain but few have knowledge about the model that is separate and that use the model in ways that extend it so. My system uses multiple representations of its algebra knowledge. Its syntactic grammar defines one representation of algebra expressions, namely the two dimensional printed one. In the expert each level of abstraction is another representation of the form and meaning of an expression. The abstracter maps expressions from one representation to another. The parser uses two different charts for its task; the character chart is ordered temporally, while the phrase chart is ordered and connected spatially. The spatial specialist controls the mapping between these representations. The system benefits from multiple specialized but connected representations, rather than trying to use one universal formalism to express its knowledge.

Computer Instruction

Computers Aided Instruction (CAI) has earned a very bad name in the education world, and rightly so. Most CAI has used computers as page-turners and bookkeepers. Flexibility and adaptability are repeatedly recommended for this field. Yet the typical answer is a preprogrammed lesson with branches between possible "paths". In essence, every possible lesson must be anticipated by the author. This kind of CAI could be a medium for a very gifted teacher, but the computer is used for little more than distribution and glitter. The

horizons of CAI were broaded by the introduction of simulations that students could perform on the machine. The system could only guess what use the student made of the simulation, however, because that simulation was a black box to it. For example, in physics there are simulations of arbitrary gravitational fields [Bork 1975] and elementary circuits; in genetics there is a fruitfly simulation of inherited genes and characteristics written for the PLATO terminal system. These systems vary in their generality. Some only allow the student to vary parameters, others allow more "structural" modifications. Still, most CAI system lack anything resembling intelligence.

An early attempt at adding intelligence to CAI was the SCHOLAR system [Carbonell 1970]. The SCHOLAR system teaches geography by a two way dialogue of question posing, question answering, and reasoning from a semantic "net" data base [Collins 1975]. Another approach to improving CAI was the use of a theorem prover to teach formal logic [Goldberg 1973]. That system could check students proofs, give hints, and complete partially solved proofs.

A system that added some intelligence (about the domain) to simulation is the SOPHIE electronics tutor [Brown&Burton 1975]. That system uses an electronics simulator to answer questions and "reason" about a power supply circuit. Faults are introduced and measurements are simulated. The student troubleshoots the circuit and the tutor comments on the students strategies and reasoning.

Many people have advocated "student models" for CAI but few have been able to give substance to the phrase. A system called WEST [Burton&Brown 1975] does make effective use of modeling student strategies and performance in comparison to an "expert" strategy and performance. WEST is a game involving arithmetic that the student plays "against" the computer, who makes hints and comments. In WEST the student's choices are noted and hypothetically explained by various strategies or methods. Reoccurrences of methods are commented on; suggestions for improved play are made on the basis of these patterns. WEST partially inspired the AICAI system model presented above, and its greatest contribution is in student knowledge modelling by means of overlays. It includes all the other modules (except the learning model), in at least rudimentary form.

The AICAI paradigm has been applied to learning decision theory and probability in the game of Wumpus [Stansfield,Carr&Goldstein 1976]. WUSOR, the Wumpus Advisor program offers advice to players on choosing moves in a game of uncertain knowledge. The

advisor is designed to illustrate the AICAI organization. The authors try to implement each module as a rule based system. Future directions for tutors of computer games are outlined in a proposal for Computer Coaches [Goldstein 1976] that exploits the similarities of good tutoring for an enjoyable game to good coaching in athletics. The research would address goals of AI, psychology, pedagogy, and computer science.

Systems like the above are changing the image of CAI. Computers can make a positive impact on education [Brown et.al. 1975]. The economics of education for too long have been that only one teacher can be provided per 20 to 40 students. Computers will have the advantage of availability. AICAI systems may eliminate the disadvantage of stupidity.

SIGNIFICANCE

The significance of this research can be discussed in terms of the good ideas that went into the system, the ideas that led to dead ends, the surprises encountered, and the A.I. issues and techniques applied.

The Good Ideas

I think this research combines many good ideas, some original and some drawn from other work. The focus idea was to integrate a graphical communication channel with an AICAI tutor of algebra. To accomplish this, many more ideas were employed.

The trainable character recognizer is a very general and powerful model of recognition. A set of features is chosen to be extracted from the input, and a record is kept of the training set which guides future recognition. This general model was successfully applied to tablet character recognition, guided by existing recognizers and extending them where necessary.

Augmented context free parsing is another good idea that was adapted to Algebra Understanding and was generalized from one dimensional languages to two dimensional ones. Algebra fits conveniently into this kind of syntactic language description. Parsing is facilitated by many techniques. The character chart for the recognizer-parser interface and the phrase chart are important organizing devices. The spatial specialist formalizes the geometric relations underlying graphical algebra expressions. Parsing can be viewed with insight both as search and as a pattern match between the grammar and the input. Parsing would be next to impossible without the addition of a Scheduler and Scoring system. In this area the good ideas include the use of multi-tasking with a queue of tasks and priorities. The scoring strategy of shortfall density is also interesting.

The expert's abstracter is a general scheme for representing algebra expressions at many levels. The particular levels chosen are based on ideas for simplification and hash-evaluation. The expression matcher generalizes simple subexpression matching and can take advantage of the expert's levels of abstraction. The abstracter and matcher combine in a novel way to use the "discourse" context in the understanding process and to potentially guide the parser directly.

Intelligent systems are the combination of many good ideas, not merely a few 'universal' principals as early researcher hoped. It has taken many ideas to design and build

this algebra system, and not all the ideas could be integrated in or made to work.

Dead Ends

Some ideas looked promising but for various reasons were eventually rejected. For example I thought the parsing paradigm of Augmented Transition Nets (ATNs) [Woods 1970] could be generalized to two dimensions. I thought the arcs could include spatial direction restrictions and 'pushes' for subexpressions. But on closer examination, the generalization fell apart. When recursion was included, the direction restrictions and the pushes for subexpressions did not occur together in pairs as envisioned. Instead, if an ATN had three pushes it would have two spatial restrictions to knit them together. This modification might have been accommodated but the ATN view suffered from worse problems. The spatial predictions broke down across the recursion; the subexpression's location could be predicted, but not the location of its 'first' constituent. In fact there isn't a very natural notion of 'first' constituent of an expression. A left-to-right and up-to-down ordering can be imposed, but it is not as natural as the time ordering in natural language and other one dimensional languages. For example, after a parenthesis is found, an expression can be expected to its right. But if the expression is for example a fraction, then the first constituent, the numerator, is up to the right from the parenthesis. It doesn't help to call the fraction bar the first constituent either, because the next constituents, the numerator and denominator would not be directly related, and since the fraction might occur as a denominator, the natural first constituent would again be the numerator. More juggling has failed to produce a suitable generalization of ATNs and I have given up that attractive, but unworkable idea.

Another dead end was the use of destructive modifications of partial theories to build larger theories. Standardly, subexpressions are incorporated into higher expressions without modification. The tree structured phrase structures can share subtrees; in this way duplication of effort is avoided. In parsing infix operators with precedence, it looked attractive to modify phrases as new information was discovered. For example the parser might find $\underline{a + b}$ and build the appropriate phrase. Next it might look further and see that the input was $\underline{2\ a + b}$ where the multiplication had higher precedent and therefore preceded the addition. Either the addition phrase could be discarded or the first argument $\underline{a}$ could be changed to $\underline{2\ a}$. The modification seemed like an elegant way to handle precedence of operators, but it turned out that it conflicted fundamentally with the chart idea. There is always uncertainty in the theories

that the parser builds, and so it cannot afford in general to be modifying a phrase that might actually have been correct without modification. If the system could be sure that it always did the right thing, then it could use destructive modification and parse deterministically as Marcus is able to do for English [Marcus 1975]. To get around the objectionable modification, I experimented with putting a level of indirection between operators and arguments. I tried using formal names for function arguments and maintaining contexts of name value bindings to preserve the notion of alternative theories that shared structure. In the above example the plus operator would have two symbolic arguments, one of which would be bound to a. When the multiplication was discovered, that argument name would be rebound to the product of 2 and a. The resulting system would have intensional names which could be manipulated without knowing their referents. Equivalence classes would be necessary, because many names could have the same referent, and in different interpretations, one name could have different referents. The design became unwieldy and I could not get a clear picture of what I wanted, nor see my way through the details. This idea became another dead end explored and abandoned.

Surprises

Research is never without surprises. I am ever surprised how easy it is to design or describe a system of processes, techniques and features, but how slow and difficult it is to put those ideas into programs. The dead ends mentioned above came as surprises.

Characters in context are surprisingly sloppier than those printed alone. It was probably a mistake to build the recognition trainer to handle characters one at a time. One cannot help printing a single character more carefully than one character in many. The same effect occurs in speech; a word in isolation is pronounced distinctly, while in the context of a sentence it may undergo radical transformation and degradation. I expected this effect that speech has, but I was still surprised by it.

Also I was surprised to find duplicate theories could develop in my chart and clog the parser. Rather then check for duplication, the parser tries to enumerate possible phrases in such a way that each potential phrase is proposed only once. Unfortunately, this desired behavior is a global property of the algorithm and is fragile with respect to many local perturbations. When consecutive phrases are duplicated the extra phrases increase

multiplicatively. If I were redesigning the system I would have more consistency checks to avoid wasted replication.

Finally, I was pleasantly surprised how small and simple the expression simplifier became as I understood it better and could combine similar rules into more general ones. The first steps of symbolic math processing were instructive and satisfying to rediscover. It is surprises like these that make a project like this one interesting.

AI Issues

This system is more an exercise in AI-engineering than basic research on one topic. The two goals, computerized tutoring and tablet communication, can each be pursued with and without the use of AI techniques, but I think the use of knowledge based programming is necessary for achieving either goal. I think the techniques drawn together in my system begin to acheive these goals of machine intelligence.

A central AI issue in this research is the use and control of multiple knowledge sources. One source of knowledge is the individual character's features. Another is the set of differential diagnostics to distinguish similar characters, i.e. knowledge of which features are most important to choosing one character over another to account for some input. Next the system knows the formal syntax for algebra, that is, which expressions are well formed; for example, parentheses usually match. Also there is an informal syntax for algebra which has to do with such things as the ordering of subexpressions, the spacing and clustering of expressions. Numerical factors precede variables, variables are generally alphabetized, coefficients of one are dropped, etc. Next, there is semantic reasonableness such as typical values for such things as exponents. There is, in fact, the whole semantics of algebra that underlies the expressions and procedures. These semantics are modeled by the levels of abstraction the expert uses and the transformation rules of algebra. The next higher source of knowledge is the discourse structure. The algebra session is goal oriented. There is coherence to the subexpressions throughout a problem session. Finally, the system can have a model of the student user. His strengths and weaknesses, his consistent bugs or conventions can help the system to understand his input.

My approach to organizing these diverse knowledge sources is extending the non-determinism of the parsing strategies to encompass them. Observing the principle of least commitment, each module offers multiple explanations of what it sees. A scheduling and

scoring system combines the evidence of the various modules and filters out the less likely interpretations. The representations and interactions among the modules is not as uniform as in CMU's Hearsay, for example, because the system's layers are arranged to communicate directly with their neighbors. The recognizer communicates with the parser but not with the expert, for example. The simple stylized communication helps to keep the system simple.

Like most AI systems, this one can be viewed as rule based. I think rules are the more regular component of knowledge but cannot be separated from their interpreter. My description of the two-dimensional constraints in algebra is composed of simple rules built out of a handful of spatial relations. Making the rule behave correctly, however, required spatial knowledge to be embedded in the rule interpreter, the parser. I think generally that there are domains of knowledge that are simple to describe when the right primitives are chosen. Programming is greatly simplified by a good language; linguistics has been searching for the right interpreter for rules of syntax. Transformational grammar is one interpreter, Marcus's WASP parser is another interpreter that makes grammar rules very simple. Beginners in a field or to a skill often find they must absorb a methodology before they can acquire any content. For my system, I know I have the right primitives and interpreter when the rules can express simply the knowledge in a module. The character recognition trainer is almost like a rule editor and debugger. It helps you input rules for character identification and shows conflicting rules where further debugging is needed.

My system benefits from a large collection of ideas and view points that have developed in the name of Artificial Intelligence.

CONCLUSION

This research does not have a clean conclusion. I chose a very open-ended project with many goals, the first of which were met, many more were explored to differing degrees, and some goals were completely out of reach. The possibilities for continued research are plentiful. The Algebra Tutoring system as sketched could be completed. Only the start of the expert module was designed. The tutoring and modeling modules were ignored. Many topics in the input module remain to be explored. There is the possibility of specializing the character set to a student as he works. A more thorough attempt could be made to understand a student that consistently mis-parses expressions; this system actually leans heavily on nearly correct algebra. A more deterministic approach to the parsing could probably succeed and be more efficient. Another direction of development which ought to be pursued is self explanation. The student is learning to do just what the tutor must do, and if the machine's algebra procedures resemble the student's, then it could explain in more understandable ways how to, for example, clump symbols of an expression into phrases observing precedence and other conventions, or it could explain why it chose some algebraic transformation to perform over another. The use of computers in education has this potential to be an example; to be not a black box, but "a white box" [Goldstein&Papert 76].

In conclusion, I think tablet understanders and computer tutors complement each other nicely. This research has demonstrated some of the capabilities of the combination and some of the techniques for engineering such systems. Much remains to be done in this area to build fluent computer tutors.

BIBLIOGRAPHY

Anderson, Robert H. (1968) *Syntax-Directed Recognition of Hand-Printed Two-Dimensional Mathematics*, PhD Dissertation, Harvard University, Cambridge, MA.

Bernstein, Morton I. (1969) *Hand-Printed Input for On-Line Systems*, Technical Report, System Development Corporation, Santa Monica, CA.

Bernsein, Morton I. (1971) "Computer Input Output of Two-Dimensional Notations," in the proceedings of Second Symposium on Symbolic and Algegraic Manipulation, March 23-25, 1971, Los Angeles, CA.

Bobrow, Daniel and Collins, Allan. (1975) *Representation and Understanding, Studies in Cognitive Science*, Academic Press, New York, NY.

Bork, Alfred M. (1975) "Effective Computer Use in Physics Education," *American Journal of Physics.*

Brown, John Seely and Burton, Richard R. (1975) "Multiple Representations of Knowledge for Tutorial Reasoning," in Bobrow and Collins, Representation and Understanding, Studies in Cognitive Science, Academic Press, New York, NY.

Brown, John Seely, Burton, Richard R, Miller, Mark L, deKleer, Johan, Purcell, Stephen, Hausemann, Catheline and Bobrow, Robert. (1975) *Steps Toward a Theoretical Foundation for Complex Knowledge-Based CAI*, Bolt Beranek and Newman, Cambridge, MA.

Burton, Richard R and Brown, John Seely. (1975) "A Tutoring and Student Modelling Paradigm for Gaming Environments," *SIGCSE Bulletin*, February.

Carbonell, Jaime R. (1970) *Mixed-Initiative Man-Computer Instruction*, Technical Report 1971, Bolt, Beranek and Newman, Cambridge, MA.

Collins, Allan, Warnock, Eleanor, Aiello, Nelleke and Miller, Mark L. (1975) "Reasoning from Incomplete Knowledge," in Bobrow and Collins, Representation and Understanding, Studies in Cognitive Science, Academic Press, New York, NY.

Dimond, T L. (1957) "Devices for Reading Hand Written Characters," proceedings of Eastern Joint Computer Conference, December 1957.

Goldberg, Adele. (1973) *Computer-Assisted Instruction, The Application of Theorm-Proving to Adaptive Response Analysis*, Technical Report 203, Psychology and Education Series, Institute for Mathematical Studies in the Social Sciences, Stanford University, Palo Alto, CA.

Goldstein, Ira P. (1974) *Understanding Simple Picture Programs*, Technical Report 294, A.I. Lab, M.I.T., Cambridge, MA.

Goldstein, Ira P. (1975) "Summary of MYCROFT: A System for Understanding Simple Picture Programs," *Artificial Intelligence* 6, 3, 249-288.

Goldstein, Ira P and Miller, Mark M. (1976) *AI Based Personal Learning Environments: Directions for Long Term Research*, AI Memo 384, AI Lab, M.I.T., Cambridge MA.

Goldstein, Ira P and Papert, Seymour. (1976) *A.I., Language and Study of Knowledge*, A.I. Memo 337, A.I. Lab, M.I.T., Cambridge, MA.

Goldstein, Ira P. (1977) *A Preliminary Proposal for Research on The Computer as Coach: An Athletic Paradigm for Intellectual Education*, Memo 388, AI Lab., M.I.T., Cambridge MA.

Goldstein, Ira and Carr, Brian. (1977) *Overlays: a Theory of Modelling for Computer Aided*

Instruction, Forthcoming Memo, AI Lab., M.I.T., Cambridge MA.

Guertin, F E. (1971) *MATTER, Its Functional Description*, Dynamic Modeling Computer Graphics System Document SR.15.17, Project MAC, M.I.T., Cambridge, MA.

Heidorn, George. (1975) "Augmented Phrase Structured Grammar," in Theoretical Issues in Natural Language Processing, June 10-13, 1975, Cambridge, MA.

Henderson, Austin. (1968) *Pattern Grammars*,Computational Structures Internal Memo, M.I.T.

Herot, Christopher F. (1974) *Using Context in Sketch Recognition*, (Master's) Dissertation, M.I.T., Cambridge, MA.

Hitachi, I. (1975) *Direct Input System for Handprinted Programs and Data*, Central Research Laboratory of Hitachi, Ltd., Tokyo, Japan.

Kaplan, Ron. (1973) "A general Syntactic Processor," in Rustin, Natural Language Processing, Algorithmics Press, New York.

Kay, Martin. (1967) *Experiments with a Powerful Parser*, RM-5452-PR RAND Corporation, Santa Monica, CA.

Lesser, Victor, Fennel, Richard, Erman, Lee and Reddy, Raj. (1975) "Organization of the Hearsay II Speech Understanding System," *IEEE Transaction on Acoustics, Speech and Signal Processing* Assp-23, 1, 11-24.

Marcus, Mitch. (1975) "Diagnosis as a Notion of Grammar," in Theoretical Issues in Natural Language Processing, June 10-13, 1975, Cambridge, MA.

Martin, William A. (1971) "Computer Input Output of Mathematical Expressions," in the proceedings of Second Symposium on Symbolic and Algegraic Manipulation, March 23-25, 1971, Los Angeles, CA.

Miller, Mark L and Goldstein, Ira P. (1976a) *Overview of a Linguistic Theory of Design*, Memo 383, AI Lab., M.I.T., Cambridge, MA.

Miller, Mark L and Goldstein, Ira P. (1976b) *PAZATN: A Linguistic Approach to Automatic Analysis of Elementary Programming Protocols*, Memo 388, AI Lab., M.I.T., Cambridge, MA.

Moses, Joel. (1967) *Symbolic Integration*, Proect MAC, TR-47, M.I.T., Cambridge, MA.

Negroponte, Nicholas. (1975) *Sketching A Computational Paradigm for Personalized Searching*, Working Paper Architecture Machine Group, Department of Architecture, M.I.T., Cambridge, MA.

Newman, William and Sproull, Robert F. (1973) *Principles of Interactive Computer Graphics*, McGraw Hill, New York, NY.

Paxton, William and Robinson, Ann. (1975) "System Integration and Control in a Speech Understanding System," *American Journal of Computational Linguistics* 5.

Pratt, Vaughan. (1973) "A Lingistics Oriented Programming Language," Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, CA.

Smith, David Canfield. (1975) *PYGMALION: A Creative Programming Environment*, A.I.M.-260, A.I. Lab, Stanford University, Palo Alto, CA.

Stansfield, J, Carr, B and Goldstein, I. (1976) *Wumpus Advisor I: A First Implementation of a Program that Tutors Logical and Probabilistic Reasoning Skills*, Memo 381, AI Lab., M.I.T., Cambridge.

Teitleman, Warren. (1963) *New Methods for Real-Time Recognition of Hand-Drawn Characters*, Report 1015, Bolt Beranek and Newman, Cambridge, MA.

Teitelman, Warren. (1975) *INTERLISP Reference Manual*, Xerox P.A.R.C., Palo Alto.

Walker, D.E, W.H. Paxton, J.J. Robinson, G.G. Hendrix, B.G. Deutsch, A.E. Robinson. (1975) *Speech Understanding Research*, Stanford Research Institute, Menlo Park, CA.

Woods, William. (1970) "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM*, Vol. 13, No. 10.

Woods, William, et. al. (1976) *Speech Understanding Systems, Final Report*, BBN Report No. 3438, Bolt Beranek and Newman, Cambridge, MA

APPENDIX

Character Set

The character recognizer has been trained to recognize the digits, the upper and lower case alphabet (where they differ) and various math symbols as follows:

0 1 2 3 4 5 6 7 8 9

Aa Bb C Dd Ee Ff Gg Hh Ii Jj K Ll Mm

Nn O P Qq Rr S Tt U V W X Yy Z

( ) { } [ ]

+ - = < > ? ! / \

square root

long division

less than or equal

greater than or equal

division symbol (dot bar dot)

integral sign

sigma

pi

Algebra Grammar

Category Inclusions:

```
EXP-----|TERM
        |PROD---FACTOR-|LETTER--------{A,B,...,Z,a,b,...z
                       |NUMB----DIGIT--{0,1,...,9
ADDOP-{+,-,PM
```

Rewrite Rules:

(ADDOP	LEFT-OF	PROD)	TERM
(EXP	LEFT-OF	TERM)	EXP
(PROD	LEFT-OF	FACTOR)	PROD
(:	LEFT-OF	PROD)	RECIP
(/	LEFT-OF	PROD)	RECIP
(PROD	LEFT-OF	RECIP)	EXP
(EXP	ABOVE	--)	NMRTR
(NMRTR	ABOVE	EXP)	FACTOR
(ROOT	ON	EXP)	EXP
(NUMB	LEFT-OF	DIGIT)	NUMB
(+	ABOVE	-)	PM
(/	ON	EXP)	EXP
(FACTOR	LEFT-BELOW	EXP)	FACTOR

Simplifier

At level 5, the abstracter simplifies expressions find a semi-canonical form for them. The simplifier applies algebraic identities, evaluates the variable-free expressions, regroups the arguments of associative operators, reorders the arguments of commutative operators, combines like terms and factors (cancellation), and selectively applies the distribution properties. In order to order the arguments of commutative operators, all expressions are given a unique identification number. The following type conventions are used to express the simplifier's rules:

a,b,c	algebra expressions
r,q	rational numbers
n,m	signed integers
a0,bl	algebra expressions ordered by unique id-number (a0<bl)
+,*,^	addition, multiplication, exponentiation

SIMPLIFIER'S RULES:

Identity

0 + a

0 * a

1 * a

0 ^ a

1 ^ a

a ^ 0

a ^ 1

Evaluation

+ q + r

* q * r

q ^ n

Associativity

a + (b + c) ==> (a + b) + c

a * (b * c) ==> (a * b) * c
a ^ (b * r) ==> (a ^ b) ^ r
a ^ (b * c) <== (a ^ b) ^ c

Commutivity

+ b! + a0 ==> + a0 + b!
* b! * a0 ==> * a0 * b!

Cancellation

+ a*q + a*r ==> a*(q+r)
* a^q * a^r ==> a^(q+r)

Distribution

+ r*(a+b) ==> + r*a + r*b
* (a*b + a*c) ==> * a * (b+c)
(a*b + a*c) ^ r ==> a^r * (b+c)^r

The internal representation for expressions at this level is in Lisp S-expressions using the prefix operators:

(VAR a) variable
(RAT m n) rational number
(ADD a b) addition
(MUL a b) multiplication
(exp a b) exponentiation

All algebraic operations must be expressed in terms of these basic ones. A/B is expressed as (MUL A (EXP B -1)).