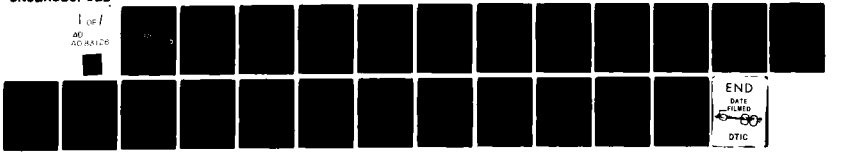
AD-A083 126 HAWAII UNIV HONOLULU DEPT OF INFORMATION AND COMPUT--ETC F/G 9/2
AN INVESTIGATION OF COMPUTER SYSTEMS PROBLEMS.(U)
MAR 80 W W PETERSON, A LEW, S Y ITOGA DAAG29-77-G-0196
ARO-14707.7-EL NL

UNCLASSIFIED

1 of 1
AD
A083126



END
DATE FILMED
DTIC

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: 14 1470.7-EL
2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER

4. TITLE (and Subtitle): 6 AN INVESTIGATION OF COMPUTER SYSTEMS PROBLEMS
5. TYPE OF REPORT & PERIOD COVERED: Final Report 15 Aug 77 - 14 Feb 80
6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s): 10 W. W. Peterson, A. Lew, S. Y. Itoga
8. CONTRACT OR GRANT NUMBER(s): 15 DAAG29-77-G-0196

9. PERFORMING ORGANIZATION NAME AND ADDRESS: University of Hawaii at Manoa, Honolulu, Hawaii 96822
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 12 24

11. CONTROLLING OFFICE NAME AND ADDRESS: U. S. Army Research Office, P. O. Box 12211, Research Triangle Park, NC 27709
12. REPORT DATE: 11 Mar 80
13. NUMBER OF PAGES: 21

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)
15. SECURITY CLASS. (of this report): Unclassified
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

LEVEL II

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DTIC ELECTE APR 15 1980 D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

C

18. SUPPLEMENTARY NOTES

The view, opinions, and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

computers, algorithms, decision tables, inference, grammar, string manipulation, computation, proof checking, microcomputers, text editing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The main problem areas investigated were decision tables, grammatical inference, string merging, computational complexity analysis, and proof checking.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

AD A083126

DDC FILE COPY

AN INVESTIGATION OF COMPUTER SYSTEMS PROBLEMS

W. W. Peterson, A. Lew, and S. Y. Itoga
Co-Principal Investigators

Department of Information & Computer Sciences
University of Hawaii at Manoa
2565 The Mall
Honolulu, Hawaii 96822

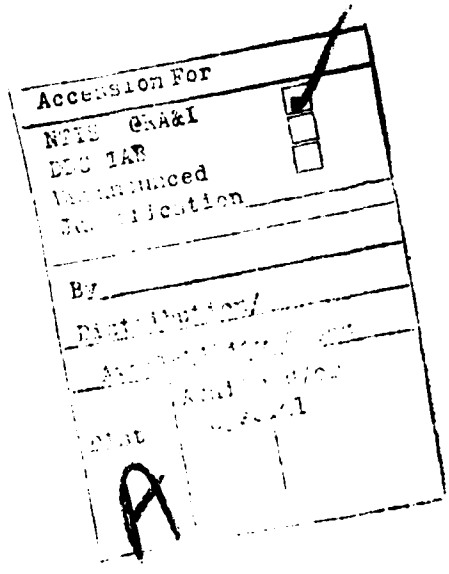
March 1980
Final Report

Prepared for the U.S. Army Research Office
under Grant No. DAAG29-77-G-0196, Project No. P-14707-EL

FOREWARD

This report is a Final Report summarizing research supported by the U.S. Army Research Office under Grant No. DAAG29-77-G-0196, Project No. P-14707-EL. This Final Report covers the entire period of the grant, from August 15, 1977 to February 14, 1980.

The main computer system problem areas investigated were concerned with decision tables, grammatical inference, string merging, computational complexity analysis, and proof-checking.



SUMMARY OF RESEARCH

The main problem areas investigated were decision tables, grammatical inference, string merging, computational complexity analysis, and proof-checking.

In the area of decision tables, we have (a) derived various results on converting flowcharts to decision tables, including conditions for "semantic" conversion (i.e. without introducing new variables or test and sets) and conditions for reducing their complexity, (b) implemented a decision table processor with facilities for checking consistency and ranges, for generating test data, and for handling subtables, (c) adapted the assertion method for proving decision table programs correct, (d) studied decision table programming methodologies, and as an example developed a decision table processor as a decision table, (e) derived lower upper bounds for the complexity of an optimal decision table conversion algorithm, and (f) examined various properties of the "level-rule" graph representation of a decision table (including loop structures and planarity).

In the area of grammatical inference, we examined a new technique for inferring regular grammars from sets of strings. In many cases, this new technique produces less complex grammars. Of special interest is the technique's use of recursion.

In the area of string merging, which has applicability to text editing, we have extended results of others to permit very general changes to sets of strings. The shortest common supersequence and longest common subsequence problems are solved as special cases.

In the area of computational complexity (or algorithm analysis), we have investigated a particular class of assignment problems - the "stable marriage" problem. We derived upper bounds to a known algorithm for its solution.

Furthermore, we extended this algorithm to the case where the number of participants was allowed to increase with time, and have studied its complexity. Our new procedure has produced stable solutions with considerable computational savings over others. The analysis of a probabilistic version of the problem was also initiated.

In addition, we have continued work on a proof-checking system. A new task we have initiated is the implementation of such a system on a microcomputer system.

A. DECISION TABLES

Emulation of Flowcharts

A decision table can be regarded as a tabular general-purpose programming language. In the past, decision tables have been promoted as good alternatives to flowcharts for the preliminary design of complex logical programs. We have taken the view that decision tables are also useful for implementing algorithms of any sort, not just for their traditional applications. We base this view on the previously reported [1] propositions that a decision table is complete in theory in that any computable (i.e. recursively enumerable) function can be implemented by a decision table, and that a decision table is complete in practice in that any program flowchart can be functionally converted to a decision table. "Functional conversion" means that, for any input, the table computes the same function as the flowchart. In a sense, a decision table incorporates sequencing, alternation, and iteration capabilities within a single program control structure.

Upon examining the decision table as a program control structure, results analogous to those of Peterson/Kasami/Tokura [2] and of Ledgard/Marcotty [3] were obtained. We have shown that any flowchart can be emulated by a decision table having θ rules, where θ is the number of action blocks in the flowchart, and furthermore that any flowchart can be emulated by a decision table having no more than $\pi+1$ "levels," where π is the number of condition-tests in the flowcharts. Emulation is stronger than functional conversion in that the order in which condition-tests and actions are executed must remain identical. However, it may be necessary to introduce new "semantics" (i.e. a new control variable with associated tests and sets). We have also shown that there exist flowcharts that cannot be implemented by a decision table without introducing new tests and actions and/or changing execution sequences, and that a necessary (but not sufficient) condition for a nontrivial flowchart to be implemented by a decision

table without introducing new semantics is that of action-test independence. Action-test independence permits execution sequences to be changed as necessary.

In addition, relationships between the "cyclomatic complexity" measure of McCabe [4] for flowcharts and the level complexity of decision tables were examined. Conditions under which level complexity can be reduced, by merging or coalescing nodes while maintaining semantic equivalence, have been derived. We have shown that a reduction in level complexity may lead to an increase in cyclomatic complexity.

The results described above are reported in [5]. Of course, it is clear that many questions remain. An incidental contribution of this research is in fact the demonstration that decision tables are subject to the same kinds of formal scrutiny as flowchart programs. Indeed many of the questions raised about algorithms and flowcharts should also be raised about decision tables regarded as just another program control structure. In addition to questions about complexity and reducibility, questions of reliability, verifiability, and general programming methodology are well worth investigating. For example, we discuss elsewhere in this report the use of the "assertion method" for proving decision programs correct. Our main conclusion here is that, regarded as a general-purpose programming language with a control structure that enforces certain disciplines, decision tables provide a viable alternative to conventional "algorithmic" languages.

A Decision Table Processor

We have implemented a decision table processor (actually several different versions) - in part to test various ideas on increasing the amount and value of diagnostic information and other aids, and in part to provide a means to actually run decision table programs. The latter enabled us to gain experience in the actual use of decision tables, which was very necessary for our study of programming methodological questions [as reported elsewhere]. In turn, this actual use of a decision table processor gave us added insight into the kinds of diagnostic information that would be of value. A basic objective of ours is to develop the processor to the point where it can be profitably used in introductory programming classes.

Among the diagnostic facilities we have implemented are ambiguity checks (for consistency or redundancy) that incorporate some semantic range analysis, and test data generation.

We have also aimed for generality - i.e. an implementation that handles extended-entries, subtables, and recursion (as well as iteration).

The various facilities mentioned above have been worked on separately, in many cases by different individuals. A remaining task is the integration of all these facilities into a single decision table processor.

Correctness Proofs

Decision table programs can be proven correct by an adaptation of the "inductive assertion" method of Floyd. In principle, we may proceed by first representing the decision table to be verified as a "level-rule" graph, which is in essence a flowchart. The main problem then is to discover loop invariants, a problem whose resolution depends upon an in-depth understanding of how the underlying algorithm works.

We have investigated the questions of whether loop invariants are easier to discover for decision table programs, and whether it is possible to use

the table itself (rather than the graph) for the proofs. Our work on the first question is inconclusive in part because "easiness" is so subjective. Regarding the second question, we have indeed proven some decision tables to be correct without appealing to their graph representations. However, these tables were small, and it is not clear that we can always meet with similar success for significantly larger tables.

Decision Table Programming Methodologies

One of our tasks was the development of programming methodologies that are suited to the design of decision table programs. We have proceeded under the assumption that decision tables can and should be designed without thinking in algorithmic language terms (and especially without "flowcharting" an algorithm). This is not to say, however, that methodologies developed for the design of algorithmic language programs cannot or should not be adapted. The concepts of modularity and top-down refinement, for example, are certainly of value for decision table design. We have, in fact, demonstrated this by so designing a non-trivial decision table program - specifically, a decision table preprocessor.

Optimal Conversion

In our earlier work [6], we developed a dynamic programming algorithm for converting decision tables to time or space optimal decision trees. One of our previous results was that the number of minimizations required for the algorithm is <u>at most</u> $\Pi_{i=1}^{N} (1 + M_i)$, where M_i is the number of values that the i-th condition can assume, and where N is the number of conditions. This upper bound is a worst-case bound in that it applies in the case of a fully expanded table. However, a stated advantage of the dynamic programming algorithm is that it can be used on a compressed table, only partially expanding some rules as necessary. The number of required minimizations then is usually less that the worst-case bound by an amount that depends upon the necessary

expansions. How much less is the question we have now addressed.

One reduction in the upper bound given above is based upon the observation that this bound unnecessarily counts evaluations of the end-conditions $f_0(\phi/R_0)$, which evidently don't require minimizations. There are $\Pi_{i=1}^{N} M_i$ of these, hence a better worst-case upper bound is $\Pi_{i=1}^{N}(1 + M_i) - \Pi_{i=1}^{N} M_i$. For the special case where $M_i = M$ for all i, the bound simplifies to $(1 + M)^N - M^N$. [Note: M = 2 is the limited-entry case.]

The case where compressed rules are disjoint was considered next. For each such rule, having (say) j "don't cares", the upper bound may be reduced by $(1 + M)^j - M^j$.

Finally, we considered the case where two compressed rules overlap, where one rule has j_1 "don't cares" and the other rule has j_2. The upper bound may then be reduced by $(1 + M)^{j1} + (1 + M)^{j}2 - K$, where K depends upon the amount of overlap.

Work is continuing along these lines, with the objective of obtaining the lowest possible upper bound while making as few restrictive assumptions as possible.

Graphical Considerations

We have investigated various properties of the level-rule graph associated with any decision table. The level-rule graph is obtained by treating the "levels" of the decision table as nodes, and the rules of the decision table as the interconnecting branches. The loops of such a graph are, of course, of special interest, and so we have designed a loop detection algorithm (as a decision table!). Complexity measures of decision tables based upon the loop structure of their level-rule graphs were also examined, as well as methods for reducing such complexity.

In conclusion, we have examined the question of "well-structuredness" of decision tables. Planarity of their level-rule graphs was found to be

a necessary but not sufficient condition for at least one definition of well-structuredness (analogous to that for algorithmic programs).

References

1. Lew, A. and Tamanaha, D., "Decision Table Programming and Reliability," Proc. 2nd Intl. Conf. on Software Engrg., 1976, pp. 345-349.

2. Peterson, W. W., Kasami, T., and Tokura, N., "On the Capability of While, Repeat, and Exit Statements," Comm. ACM, Vol. 16, No. 8, pp. 503-512.

3. Ledgard, H. F. and Marcotty, M., "A Geneology of Control Structures," Comm. ACM, Vol. 18, No. 11, pp. 629-639.

4. McCabe, T. J., "A Complexity Measure," IEEE Trans. Software Engrg., Vol. SE-2, No. 4, pp. 308-320.

5. Lew, A., "On the Emulation of Flowcharts by Decision Tables," submitted to Comm. ACM.

6. Lew, A., "Optimal Conversion of Extended-Entry Decision Tables with General Cost Criteria," Comm. ACM, Vol. 21, No. 4, pp. 269-279.

B. COMPUTATION THEORY

Grammatical Inference

Part of our study resulted in an alternative inference scheme to one proposed by Feldman, et.al. in a Stanford Artificial Intelligence Project [1]. This scheme was described in a report entitled "A New Heuristic For Inferring Regular Grammars" which has been accepted for publication.

The most significant aspect of the new scheme was a simplified set of conditions for reducing the number of production rules in the inferred grammar. Comparison tests using five cases reported by Feldman, et.al, showed improvements in complexity under the measures proposed by Wharton [2] and by Feldman [1].

Another interesting aspect of the new scheme was its use of recursion in reducing the complexity of its output. This enabled the scheme to infer infinite languages without resorting to the explicit "residue" rules found in Feldman's scheme. In summary, this study melded ideas and notions from the areas of grammatical inference, computational complexity, and recursion theory to generate new results.

String Merging (Text Editing)

In this part of our study we extended results by Wagner and Fischer [3] to the case where very general changes to sets of strings could be studied. These results were summarized in a report entitled "The String Merging Problem" which has been submitted for publication.

Wagner and Fischer [3] posed the fundamental string-to-string correction problem (STS-problem) that has received much interest [4, 5, 6, 7]. Recently there has been work done to improve the time and space complexity requirements of their original algorithm [8, 9, 10]. In this research we generalized their original problem to address the case where more than two strings are considered and where none of the strings are given any special designation. This approach

incorporates some of the ideas found in [6] and [11].

In order to describe this approach we need the following notion. Let Σ denote an arbitrary finite set of symbols, Σ^* the monoid freely generated from Σ under concatenation, and $P(\Sigma^*)$ the set of all subsets of Σ^*. If X is a string (finite sequence) of symbols then $\bar{X}\langle i\rangle$ is the ith symbol of $\bar{X}$, $\bar{X}\langle i{:}j\rangle$ is the substring $\bar{X}\langle i\rangle\ \bar{X}\langle i+1\rangle\ \ldots\ \bar{X}\langle j\rangle$, and $|\bar{X}|$ is the length (number of symbols) of $\bar{X}$, i.e., $\bar{X} = \bar{X}\langle 1{:}|\bar{X}|\rangle$.

For a fixed Σ, an edit operation s is any mapping from Σ^* to $P(\Sigma^*)$. We will use S to denote an arbitrary finite set of edit operations. Then for A, B $\in \Sigma^*$ and $s \in S$, B is a result of editing A under s (written A $\Rightarrow$ B via s) if B $\in s$(A). An edit sequence s is any finite sequence of edit operations. The action of $s = s_1, s_2, \ldots, s_m$ on string X is defined to be $s(\bar{X}) = s_m(s_{m-1}(\ldots s_1(X))\ldots)$. We let ψ represent the null sequence and define $\psi(\bar{X}) = \bar{X}$. For a sequence of strings $\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_m$, a merge sequence of operations is any sequence of edit sequences $\mu = s_1, s_2, \ldots, s_m$ such that $\bigcap_{i=1}^{m} s_i(\bar{X}_i) \neq \Phi$ (is not empty). I.e., a merge sequence of operations can edit a sequence of strings into a common string. For a fixed Σ, a set S of operations is said to be complete if for any finite sequence of strings $\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_m$ from Σ^* there always exists a merge sequence from S.

As in [11] we let γ be an arbitrary cost function such that $\gamma(s)$ is a nonnegative real number for all $s \in S$. Then γ is extended to edit sequences $s = s_1, s_2, \ldots, s_m$ via $\gamma(s) = \sum_{i=1}^{m} \gamma(s_i)$. For consistency we set $\gamma(\psi) = 0$. In addition γ is extended to merge sequences $\mu = s_1, s_2, \ldots, s_m$ via $\gamma(\mu) = \sum_{i=1}^{m} \gamma(s_i)$. We now define the merge distance for a finite sequence of strings $\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_m$ to be the minimal cost of editing all the strings into one common string. Formally, $D(\bar{X}_1, \bar{X}_2, \ldots, \bar{X}_m) = \min\{\gamma(\mu) \mid \mu$ is a merge sequence for $\bar{X}_1, \bar{X}_1, \ldots, \bar{X}_m\}$. Since the merge distance for a sequence of strings is independent of the particular arrangement of the strings, we can then use the terms merge set of operations and merge distance for a set of strings without any ambiguity.

This notation enables us to state the string merging problem (SM-problem). Given a finite set of strings $\{\bar{X}_1,\bar{X}_2,\ldots,\bar{X}_m\}$, a complete set of edit operations S, and an associated cost function γ, the problem is to determine the merge distance $D(\bar{X}_1,\bar{X}_2,\ldots,\bar{X}_m)$ and any particular member of $\bigcap_{i=1}^{m} s_i(\bar{X}_i)$ where $D(X_1,X_2,\ldots,X_m) = \sum_{i=1}^{m} \gamma(s_i)$. Necessary and sufficient conditions for this problem to match the STS-problem were formulated, The special case where deletion is the only allowed editing operation was shown to solve the longest common subsequence problem (LCS-problem) while the special case where insertion is the only allowed editing operation was shown to solve the shortest common supersequence problem (SCS-problem). Algorithms to solve each special case were found that operated in polynomia time complexity with respect to the number of strings under consideration, but exponential time complexity with respect to the maximum length of the strings.

The biological and data compression applications mentioned in [6] are still valid under the present formulation. Since we are not actually constrained to looking for solutions to the LCS-problem and SCS-problem, we actually have broadened the scope of possible applications. For example, in the case of data compression for several files, depending on the cost function γ, it may be desirable to store something other than the solution to the LCS or SCS problems. Also, in the field of molecular biology when studying the amino acid sequences of several proteins, perhaps the solution to the SM-problem for particular γ and S would be of considerable interest.

In summary, this problem formulation seems to be a very general approach to describing related text editing problems. An application to data compression techniques is briefly discussed at the end of the above mentioned report.

Aspects of the Stable Marriage Problem

Here we studied the computational complexity, the sequential versus batch processing, and the probabilistic aspects of the classical version of the assignment problem formulated by Gale and Shapley. In the first case, we published a paper entitled "The Upper Bound for the Stable Marriage Problem." In the second case we have submitted a report entitled "A Generalization of the Stable Marriage Problem" for publication. In the last case we are currently in the process of writing a report on our investigation of a probabilistic interpretation of the stability condition posed by Gale and Shapley.

The problem of pairing n boys with n girls in accordance with their preferences was first formulated and solved by Gale and Shapley [12]. The first paper derived the upper bounds to the Gale and Shapley algorithm in a manner that illustrated several characteristics of worst case situations. The net effect of these results was to indicate the unusualness of a situation in which the upper bound actually occurred.

The second report extended the solution algorithm to the case where the number of participants was allowed to increase with time.

To state this problem in simpler terms consider the following situation. Suppose we have just computed a stable solution for 50 boys and 50 girls when lo and behold another boy and girl enter the scene. Would it be necessary to start all over and apply the algorithm by McVitie and Wilson [13] to the problem of 51 boys and 51 girls, or is there an efficient way of augmenting our current solution to incorporate the new persons? An affirmative answer to this question was the main result of this part of the research. Indeed, instead of the expected 200 proposals (Wilson [14]) to process the group from scratch, it was shown that less than 90 proposals would normally suffice.

The essential notions for our main result are the concepts of a stable community and a ripple operation. Informally, a stable community consists

of paired couples and possibly some surplus persons of one sex such that the group meets the stability requirements as mentioned in Gale and Shapley [12]. Any extra person(s) must be less appealling to all members of the opposite sex. E.g., any solutions to the case of possibly uneven sets of boys and girls as in McVitie and Wilson [13] would constitute a stable community. The second notion is that of a ripple operation which can be viewed to consist of the following steps. The inputs are a stable community and a separate set of persons of one sex (possibly different from the sex of any surplus persons in the community). The ripple operation starts by letting all members of the sex of this separate set propose to their best choices. All members of the opposite sex that have been paired regard their mates as being placed on hold. Any person receiving one or more proposals must decide amongst these choices and any current person on hold. As in the original algorithm by Gale and Shapley [12], all rejected persons must proceed to propose to their next choice. The process is repeated until all members of the proposing sex are on hold or until those that aren't have been rejected by all members of the opposite sex. A key result is that the resulting situation again forms a stable community.

Only requiring that relative choices remain constant, this new procedure produces stable solutions with considerable computational savings over other implementations of Gale and Shapley's algorithm. It is our contention that this result is significant enough to make feasible the consideration of this algorithm for many practical situations.

The last aspect of this phase of the study is still in progress. We have made computer simulated analyses of a probabilistic interpretation of the Stable Marriage Problem. The results have shown that for certain parameter settings, considerable computational savings may be gained at the expense of relatively little "stability" loss. We are investigating the use of this notion in conjunction with our results from the second phase of this study to produce an efficient multi-pass algorithm for the original problem.

References

1. J. A. Feldman, J. Gips, J. J. Horning and S. Reder, "Grammatical Complexity and Inference," Stanford Artificial Intelligence Project, Technical Report No. CS125, June 1969.

2. R. M. Wharton. "Grammatical Enumeration and Inference," Information and Control, vol. 22, pp. 253-272, 1977.

3. R. A. Wagner and M. J. Fischer,"The String to String Correction Problem," J. ACM 21, 1 (Jan. 1974), 168-173.

4. A. V. Aho, D. S. Hirschberg, and J. D. Ullman, The longest common subsequence problem, J. ACM 23,1 (Jan. 1976), 1-12.

5. R. Lowrance and R. A. Wagner,"An extension of the string-to-string correction problem,"J. ACM 22, 2 (April 1975), pp. 177-183.

6. D. Maier,"The complexity of some problems on subsequences and supersequences," J. ACM 25,1 (April 1978), pp. 322-336.

7. C. K. Wong and A. K. Chandra,"Bounds for the string editing problem,"J. ACM 23, 1 (Jan. 1976), pp. 13-16.

8. D. S. Hirschberg,"A linear space algorithm for computing maximal common subsequences,"Comm. ACM 18, 6 (June 1975), pp.341-343.

9. J. W. Hunt and T. G. Szymanski,"A fast algorithm for computing longest common subsequences,"Comm. ACM 20, 5 (May 1977), pp. 350-353.

10. W. J. Masek and M. S. Paterson,"A faster algorithm computing string edit distances," TM-105, MIT Laboratory for Computer Science, (May 1978), pp. 1-26.

11. P. H. Sellers,"An algorithm for the distance between two finite sequences," J. Combin.Theory (A), 16 (1974), pp. 253-258.

12. D. Gale and L. S. Shapley,"College admissions and the stability of marriage," Am. Math. Monthly, 69, (1962), pp. 9-15.

13. D. G. McVitie and L. B. Wilson,"Stable marriage assignment for unequal sets," BIT, 10, (1970), pp. 295-109.

14. L. B. Wilson,"An analysis of the stable marriage assignment problem,"BII, 12 (1972), pp. 569-575.

C. COMPUTER AIDED INSTRUCTION - PROOF CHECKING SYSTEM

We have developed a system on the IBM370 that examines a mathematical proof written in a language that resembles that used by mathematicians and tries to verify that each step follows logically from the reasons given, and thus that the proof is valid. Each step is considered a "theorem" and the reasons "axioms", and the computer uses the resolution method to try to construct a proof for that step. In addition, the system does bookkeeping and provides editing facilities.

Our primary purpose was to provide a tool for teaching mathematics students to write valid proofs – the student is asked not only to write a proof, but also to check it on the computer, just as we assign students the task of writing a program and checking it on the computer. It affords the student prompt, objective feedback and the opportunity to retry until he succeeds. If this system proves successful, it should be possible to modify it to (a) check program correctness proofs, and (b) check decision tables for correctness.

The IBM370 system is working, but needs some further checking before it is used with a class, and we are continuing to work on it. We know that the system can be used at least with selected subject matter in the intended way.

We believe the system can also be implemented on a microcomputer system. We expect such a system to be more economical, and if so, it could be made available in many places such as high schools and small colleges that cannot afford an IBM370. This is true especially if the system is portable.

We have started the task of converting the system to an 8080. The 370 system is written in PL/I, but uses only a small subset of PL/I. We have written a compiler for the 8080 for a suitable subset of PL/I, and in that subset. This compiler includes 16-bit integers, strings, and logical vari-

ables, internal and external procedures, and the common execution sequence control. It produces relocatable code, and we have an overlay loader. Much of the present proof-checking system will convert with minor changes, but certain parts will have to be rewritten to adapt efficiently to the new environment.

This system will also serve as a good example to compare a microcomputer to a large computer system for a non-trivial application, because essentially the same system will be available on both machines, and with roughly 8000 cards of PL/I source program, it is not a small system.

LIST OF PUBLICATIONS

1. Itoga, S. Y., "The Upper Bound for the Stable Marriage Problem," J. Opl. Res. Soc., Vol. 29, No. 8, pp. 811-814.

2. Itoga, S. Y., "A New Heuristic for Inferring Regular Grammars," accepted by IEEE Trans. Patt. Anal. and Mach. Int. [A revision of a manuscript entitled "Grammatical Inference for Very Large Sample Sets "]

3. Itoga, S. Y., "The String Merging Problem," submitted to J. ACM.

4. Itoga, S. Y., "A Generalization of the Stable Marriage Problem," submitted to Comm. ACM.

5. Lew, A., "On the Emulation of Flowcharts by Decision Tables," submitted to Comm. ACM.

Abstracts of the above are appended.

In addition, much of our research on decision tables will appear in:

6. Tamanaha, D., Ph.D. dissertation (in preparation).

"A New Heuristic For Inferring Regular Grammars"

Abstract

Modifications to a grammatical inference scheme by Feldman, et al. are presented. A comparison of the relative performance of the original and modified schemes is made using the complexity measures of Feldman and Wharton. The case where a complex model is used to generate the sample set is then analyzed. A set of 104 samples was found that trained the program to infer the grammar that corresponded to the original model. The results of a study of the performance of this algorithm when there is a large number of samples is then presented. The major conclusion of this study is that the modified scheme has a superior performance on small sample sets but is highly unsuitable for large ones.

(Accepted for publication in IEEETPAMI)

"The String Merging Problem"

Abstract

The string merging problem is to determine a "merged" string from a given set of strings. The distinguishing property of a solution is that the total cost of editing all of the given strings into this solution is minimal. Necessary and sufficient conditions are presented for the case where this solution matches the solution to the string-to-string correction problem. A special case where deletion is the only allowed editing operation is shown to have the longest common subsequence of the strings as its solution. Similarly, the case where insertion is the only allowed operation is shown to have the shortest common supersequence as its solution. Algorithms for these cases are presented that have time complexities that are polynomial with respect to the length of the strings, but exponential with respect to the number of strings.

(Submitted for publication)

"The Upper Bound To The Stable Marriage Problem"

Abstract

The stable marriage problem was originally posed by Gale and Shapley. The worst case performance of their solution is derived in a manner that illustrates the complexity characteristics of the problem. Several conclusions about the nature of the worst case situation are presented.

(J. Opl. Res. Soc., vol 29, 8, pp 811 to 814)

"A Generalization Of The Stable Marriage Problem"

Abstract

The stable marriage problem posed by Gale and Shapley is generalized to the case where the number of participants may increase with time. This new problem is solved in a manner that permits the introduction of arbitrary numbers of participants of either sex. The complexity of the new algorithm is analyzed in terms of the required number of proposals. Worst case and average case results are presented. In addition, the average satisfaction of the participants under this new approach is studied.

(Submitted for publication)

"On The Emulation Of Flowcharts By Decision Tables"

Abstract

Any flowchart can be emulated by a decision table, whose "complexity" depends on that of the flowchart. However, it may be necessary to introduce a new control variable with associated tests and sets, or to permit changes in execution sequences provided action-test independence holds. Two measures of decision table complexity are discussed and interrelated. Finally, conditions and procedures for reducing complexity are presented.

(Submitted for publication)

LIST OF PARTICIPATING PERSONNEL

1. W. W. Peterson, Co-Principal Investigator
2. A. Lew, Co-Principal Investigator
3. S. Y. Itoga, Co-Principal Investigator
4. D. Tamanaha, Graduate Assistant (Ph.D. candidate)
5. C. Peck, Graduate Assistant (M.S. awarded May 1979)
6. L. Blake, Graduate Assistant (M.S. candidate)
7. S. W. Lee, Graduate Assistant (M.S. expected May 1980)
8. P. Tang, Research Assistant (B.S. expected May 1980)