COMPUTERS AND BIOMEDICAL RESEARCH 10, 259–269 (1977)

A Command Processor for the Determination of Specificities from Matrices of Reactions Between Blood Cells and Antisera*

DANA S. NAU† AND MAX A. WOODBURY‡

Department of Computer Science, Duke University, Durham, North Carolina 27706

Received May 10, 1976

An interactive command processing system for the determination of the specificities of lymphocytes and antilymphatic sera is described. This system enables an investigator to quickly formulate and test hypotheses about specificities by presenting a variety of suggestive displays and other data. The system processes any of 22 commands, 2 of which contain 6 subcommands each. The mathematical model of specificities upon which the command processor is based is described, along with the immunological background which led to the formulation of the model.

INTRODUCTION

The importance of the study of the HLA antigens is due among other things to the close link between the relationship of these antigens on donor and recipient and the acceptance or rejection of transplanted organs. Serological data about these antigens are generally obtained by combining antisera with small lymphocytes (white blood cells), and adding rabbit complement to produce a cytolytic reaction and then adding trypan blue to stain the killed cells. If the set of all possible reactions is taken between several different antisera and several samples of cells from different cell donors, the data obtained may be grouped into an incidence matrix tabulating the reactions of each antiserum to each sample of cells. Analysis of such matrices provides clues about the genetics (which we do not discuss here) and serology of the antigen–antibody reactions.

This paper describes an interactive command processing computer program incorporating a number of data manipulation subroutines and other routines, which performs operations on matrices of the kind described above. These programs are written in CPS (Conversational Programming System, an interactive PL/I-like language supported by IBM). The programs provide a powerful tool to aid an experienced investigator in a rapid and interactive analysis of the data.

* This work was supported in part by NSF Grant GJ-434 and in part by an NSF graduate fellowship.

† Currently an NSF Fellow at Duke University.

‡ Professor of Biomathematics, Duke University Medical Center and Professor of Computer Science, Duke University.

ISSN 0010-4809

Recent studies by Nau (*10*) provide strong evidence that the problem of recognizing specificities is algorithmically intractable. This provides ample justification for a heuristic, interactive program such as that described herein. It should be noted that the less complex typing problem has been recommended for heuristic solution by Norton *et al.* (*11*).

Immunological Background

Humoral mediated immunity in man and in other vertebrates depends on the production of antibodies which chemically combine with the antigens against which they are directed. The formation of antibody is in direct response to challenges to the immune system by immunogenic complex biochemical compounds frequently associated with living organisms. For example, the Landsteiner blood groups antigens A1, A2, and B are imitated by bacterial cells so that exposed individuals rapidly form antibodies to any antigen except O in the ABO blood group. As a result of almost universal exposure, each individual is naturally immune to red blood cells of other individuals which differ from their own. Thus blood for transfusions must be cross matched before being used.

It is interesting to note that man is not naturally immune to sheep red blood cells, so that the initial experimental transfusion of sheep blood worked well enough. Repeated transfusions had tragic results since the human immune system—both cell mediated and humoral mediated—responded to the challenge.

Later, Landsteiner and Wiener discovered the Rh blood types (Rh for Rhesus because the original immunization was to Rhesus monkey red blood cells.) Thirty years and several thousand research papers later, the Rh system is a reasonably well understood, clinically useful system. This system is used in genetic counselling of parents, since Rh^+ children of Rh^- mothers are sometimes subject to immune attack with serious results. It is also useful in cross matching blood for transfusion.

In the decade from 1950 to 1960, the histocompatibility immune systems (those controlling reaction to transplanted organs) in the mouse were extensively investigated. The discovery of the H-2 complex of strong histocompatibility antigens on red blood cells and most or all other tissue cells was a major achievement of the study of various strains of the laboratory mouse. Since the conclusions parallel those about the human HLA system discussed below, the structure of the mouse H-2 system will be passed over here.

The search for human histocompatibility antigens did not succeed until it was found that the antigens do not express themselves on red blood cells in humans, but are found on the white blood cells. The difference lies in the fact that human red blood cells are not nucleated and the HLA antigens, if they ever were present, are shed into the plasma and lost. Without DNA for biosynthesis, the red blood cells cannot replace such antigens. The first HLA antiserum discovered for man was a cross reactive serum, Mac, which attacked cells bearing what are now called HLA-A2

and HLA-A28 antigens. Three more antisera were then found. These were called LA1, LA2, and LA3 by their discoverers. At the same time, a number of antigens 4^a, 4^b, 4^c, 4^d, 5^a, 5^b, 6^a, 6^b, 6^c, 7^a, 7^b, 7^c, 7^d, 8^a, and 9^a (reminiscent of the mouse H-2 antigens) were also independently named. Subsequent investigations showed that 8^a was equivalent to Mac mentioned above, and that both Mac and 8^a antisera reacted with LA2 and had extra reactivity as well, which is now known to be due to the antigen HLA-A28. The antisera for 5^a and 5^b and also 9^a were found to be unrelated to the others and the remaining antisera were recognized as responding to several antigens. The sera 4^a and 4^b in particular are very widely reactive sera and the antisera 6^a, 6^b, and 7^b and 7^c are similarly widely reactive. Several of these long specificities have now been subdivided, renamed, and mainly accounted for in two loci, the HLA-A (originally LA or first) locus and the HLA-B (originally FOUR or second) locus, with a large number of alleles at each. Subsequently, a unified HLA nomenclature was adopted which allowed the incorporation of provisional or w (meaning, roughly, working definition) specificities. The antiserum LA3 was found to attack two antigens, now called HLA-A3 and HLA-A11. The antisera for 4^c were split up into HLA-B5, HLA-BW35, HLA-B18. HLA-B5 is complex and exists in a variety of forms worldwide as shown especially when tested on non-Caucasian peoples. The antiserum 7^c attacks cell antigens now called HLA-B7, BW22, and B27, while 7^d is now designated HLA-B8.

This rather confusing picture is still being clarified and now other phenomena genetically associated with them have been uncovered. There is a third recently discovered, serologically detectable C locus and possibly others. For further background material, see the bibliography, especially the references cited in (*1*).

The Specificity Problem

In approaching the problem of using a computer to help an investigator to untangle this set of complex relations it becomes highly desirable to distinguish between the molecular configurations characterizing antigens and those characterizing antibodies in antisera. Antisera are formed when the immune system of the *serum donor* is challenged by the white blood cells or cells in a transplanted organ of the *cell donor*. The fact is that antisera are remarkably selective and attack only antigenic structures virtually identical with those of the cell donor in a relatively small domain of the antigen molecule. The attacked substructure is called a determinant and it is to be supposed that determinants are inherited as codominant Mendelian characters. In order to characterize the property of cells and sera that combine due to chemical bonding of antibody and antigen at the site of the determinant we find it convenient to call this "lock and key" relational property a *specificity*. In the immunological literature the "specificity" is used in a somewhat different sense from that used here, but the meaning is closely related.

It is known that if the cells used in the immunological challenge are placed in the

antiserum in sufficient quantities, all activity is removed. If cells sharing only certain of the antigens in common with the challenging cells are used, the antiserum may be unaffected in activity, reduced in activity, or even have all activity removed. The resultant absorbed serum (the antiserum left after removal of the cells) is assumed to contain none (or very little) of the antibodies that attack the cells used in the absorption. We describe this by saying that the specificities in the antiserum in common with those of the absorbing cells are removed.

We now turn to a mathematical treatment of the properties discussed above. Let U be a set of cells and V be a set of antisera. Define the relation R over $U \times V$ by uRv if and only if the cell u in U is attacked by the antiserum v in V. Let S be the set of specificities. Then there are relations P over $U \times S$ and Q over $V \times S$ such that uRv if and only if there is an s in S such that uPs and vQs. If u is a cell from the same individual whose cells were used in the immunological challenge which created v, then $\{s \text{ in } S|vQs\}$ is a subset of $\{s \text{ in } S|uPs\}$.

If u' in U is used in an absorption experiment with v, then the absorbed serum v' has the property that

$$\{s \text{ in } S|v'Qs\} = \{s \text{ in } S|vQs\} - \{s \text{ in } S|u'Ps\},$$

where "—" denotes set difference.

It is possible to explain cross reactivity of antisera by putting uPs' whenever (1) uPs, and (2) s' is a specificity supertypic to s.

We are not aware of other useful mathematical statements of the problem of this degree of generality other than those in the bibliography. Considerable care must be taken in abstracting mathematical properties of problems such as this one, since too few assumptions lead to trivial results, and factually incorrect but mathematically consistent assumptions lead to useless results. We have attempted to cite only those articles which meet these criteria. We apologize in advance to authors whose articles we may have inadvertently overlooked.

Data Structures

In CPS, the fundamental unit of program storage is the load/save module. Each load/save module can contain 1 to 4 pages of 1000 bytes each, and the modules can be loaded and saved at will. If a program is too large to fit into a single load/save module, part of it can be declared to be an external subroutine and stored in a separate load/save module, which will be loaded and executed when the subroutine is called.

There are 12 CPS load/save modules in the command processor. These modules may be divided into three types: (1) modules used only for static data storage (cell and serum names and the reaction matrix); (2) modules containing the coding necessary to carry out a specific command or set of commands; (3) the main program, which contains both arrays for data storage and the coding to read in the commands and call any subroutines necessary to execute them.

Due to the limited storage space available to CPS, the modules are overlaid as necessary. This would be the same program philosophy to be applied to any small interactively operated computer or time-sharing system.

Each element of the reaction matrix is a "#," a "—," or a "·," denoting reaction, no reaction, or missing data, respectively. This could have been done more compactly with bit strings, except that bit strings are not implemented in CPS. The use of "#" for "+" is to achieve more rapid printing on the time-sharing system used. It has the additional advantage of better contrast.

The matrix is merely for data reference, and its contents are never changed during the execution of the program. Therefore, it is not necessary to save a new version of matrix each time the program is run, so to save space in the main program, the matrix is stored in a separate load/save module. When the module is called, it returns rows, columns, or single matrix elements, depending on its arguments. The cell and serum names also remain unchanged throughout the program, so they are stored in the same manner as the reaction matrix.

The command processor starts out with the reaction matrix and the cell and serum names, and by using the commands, the user determines and names various specificities, whose names and characteristics are then stored in the program. Considerable experimentation was done in order to find the best data structure in which to represent this information. The data structure which was found to be most efficient for computational purposes and which was still able to meet the space and data type requirements of CPS was a list structure consisting of two linked lists for each specificity: one pointing to those sera containing the specificity, and the other pointing to those cells containing the specificity. Since CPS is not set up to handle lists, we wrote some list processing primitives to operate on a numerical array. This array and the one containing the specificity names must be passed as arguments to most of the external subroutines, and they are modified and must be saved after the command processor stops executing, so they are stored in the main program.

The Commands

In addition to the above arrays, the main program contains the coding to read in the commands. The commands are read in one at a time, and are generally executed by calling external subroutines. Each command consists of a single line of input containing the command name followed by its arguments (if any). As many blanks as desired may precede the command name and each of its arguments, with the restriction that each argument must be preceded by at least one blank. Each argument is a string of not more than four characters, and lower case letters are automatically converted to upper case.

We now briefly describe each of the 22 commands as well as subcommands for

some of the commands. The following conventions are used for each command description:

(1) The first line of each description is the command name and syntax line. It consists of a short descriptive phrase from which the command name is taken, a colon, and a display of the syntax for the command.

(2) The name of each command is an acronym taken from the descriptive phrase. Within the phrase, the letters used to form the command name are capitalized.

(3) The command syntax is in Backus Normal Form. Terms appearing in square brackets are optional, and "*" and "+" are the Kleene star and plus.

(4) The names of the nonterminals are self-explanatory, e.g., ⟨SPEC⟩ denotes a specificity name, ⟨CELL⟩ denotes a cell name, and ⟨DUMMY⟩ denotes an argument that is ignored.

(5) The rest of each paragraph is a brief description of the action produced by the command.

The commands may be divided into three main types: processing commands, informative commands, and meta-commands. The processing commands are used to create, delete, and change specificity names and pointers; the informative commands are used to give detailed information about the current state of the system; and the meta-commands provide special services which are not available through the processing and informative commands.

The Processing Commands

Add Cells: AC ⟨SPEC⟩ ⟨CELL⟩*

If ⟨SPEC⟩ is not already in the specificity name array, then it is added. ⟨SPEC⟩ is made to point to each cell in the argument list.

Add Sera: AS ⟨SPEC⟩ ⟨SERUM⟩*

If ⟨SPEC⟩ is not already in the specificity name array, then it is added. ⟨SPEC⟩ is made to point to each serum in the argument list.

Delete specificities: D ⟨SPEC⟩+

Each ⟨SPEC⟩ and its pointer lists are deleted.

Remove Cells: RC ⟨SPEC⟩ ⟨CELL⟩*

Each cell is removed from ⟨SPEC⟩'s serum pointer list.

Remove Sera: RS ⟨SPEC⟩ ⟨SERUM⟩*

Each serum is removed from ⟨SPEC⟩'s serum pointer list.

REName specificity: REN ⟨SPEC1⟩ ⟨SPEC2⟩

If ⟨SPEC1⟩ is a specificity name that is currently in use and ⟨SPEC2⟩ is not, then ⟨SPEC1⟩ is replaced by ⟨SPEC2⟩.

Find Cells: FC ⟨SPEC⟩

All cells are partitioned according to how many of ⟨SPEC⟩'s sera they cover. The various equivalence classes may be displayed and/or saved by means of various subcommands.

Find Sera: FS ⟨SPEC⟩

All sera are partitioned according to how many of ⟨SPEC⟩'s cells they cover. The various equivalence classes may be displayed and/or saved by means of various subcommands.

The above two commands are perhaps the most important ones in the system, for they are used to isolate new specificities on the basis of the reactions between the cells and the sera in the matrix. After partitioning the sera or cells according to their reactions and printing out a certain amount of information about the partition, each of the above two commands invokes a command submode containing a number of subcommands.

The following paragraphs describe the subcommands invoked by FC. The subcommands invoked by FS are identical except that "cells" and "sera" should be interchanged in each description. ⟨SPEC⟩ refers to the specificity name given in the original FC or FS command, and ⟨LOWER⟩ and ⟨UPPER⟩ are integers. ⟨UPPER⟩ defaults to zero, and ⟨LOWER⟩ defaults to ⟨UPPER⟩. All of the subcommands ignore extra arguments. Signalling ATTENTION during the execution of a subcommand returns control to the subcommand processor rather than to the top-level command processor.

Cover NUMbers: CNUM [[⟨LOWER⟩] ⟨UPPER⟩] ⟨DUMMY⟩*

Lines are displayed of the form "*X* cells cover all sera in ⟨SPEC⟩ except *Y*," where *X* and *Y* are integers, and *Y* ranges from ⟨LOWER⟩ to ⟨UPPER⟩.

END: END ⟨DUMMY⟩*

Program execution is terminated.

Help: H ⟨DUMMY⟩*

A short description of all subcommands is produced.

KEEP specificities: KEEP [[⟨LOWER⟩] ⟨UPPER⟩] ⟨DUMMY⟩*

Pointers are made from ⟨SPEC⟩ to exactly those cells covering between ⟨LOWER⟩ and ⟨UPPER⟩ of the sera.

Cover MATrix: CMAT [[⟨LOWER⟩] ⟨UPPER⟩] ⟨DUMMY⟩*

A reaction submatrix is displayed for those cells covering between ⟨LOWER⟩ and ⟨UPPER⟩ of the sera.

Display MATrix: DMAT ⟨DUMMY⟩*

A matrix is displayed showing the reactions of the defining sera.

The Informative Commands

Cover Matrix: CM ⟨DUMMY⟩*

The reaction matrix is displayed, with "#" replaced by "·" and "—" replaced by "n" for every reaction that is covered by some specificity. After each line of the matrix, the specificities in the cell on that line are listed. All arguments are ignored. This command allows the user to see at a glance what the hypothesized specificity cover is and how well it covers the reaction matrix.

Intersect for Cells: IC ⟨SPEC⟩*

The cells in the intersection of the cell lists of each ⟨SPEC⟩ are displayed. IC without arguments displays all valid cell names.

Intersect for Cells with Matrix display: ICM ⟨SPEC⟩*

The reaction submatrix for the intersection of the cell lists of the ⟨SPEC⟩s is displayed. ICM without arguments displays the entire matrix.

Intersect for Sera: IS ⟨SPEC⟩*

The sera in the intersection of the serum lists of each ⟨SPEC⟩ are displayed. IS without arguments displays all valid serum names.

Intersect for Sera with Matrix display: ISM ⟨SPEC⟩*

The transposed reaction submatrix for the intersection of the serum lists of the ⟨SPEC⟩s is displayed. ISM without arguments displays the entire matrix.

List specificities: L ⟨DUMMY⟩*

All existing specificity names are listed. All arguments are ignored.

List Cells: LC ⟨SPEC⟩*

For each ⟨SPEC⟩, the cells to which it points are listed. LC without arguments lists cells for every ⟨SPEC⟩.

List CElls and their SPecificities: LCESP ⟨CELL⟩*

For each ⟨CELL⟩, those specificities pointing to it are listed. If no cells are given, then the specificities are listed for every cell.

List Sera: LS ⟨SPEC⟩*

For each ⟨SPEC⟩, the sera to which it points are listed. LS without arguments lists sera for every ⟨SPEC⟩.

List SEra and their SPecificities: LSESP ⟨SERUM⟩*

For each ⟨SERUM⟩, those specificities pointing to it are listed. If no sera are given, then the specificities are listed for every serum.

The Meta-Commands

END: END ⟨DUMMY⟩*

Program execution is terminated. All arguments are ignored.

Help: H ⟨COMMANDNAME⟩* | H ALL

For each ⟨COMMANDNAME⟩, a description of the command is produced.

Entering H with no arguments produces a list of all available commands. Entering H ALL produces descriptions of all available commands.

SAVE specs: SAVE ⟨DUMMY⟩*

The arrays containing the specificity names and pointers are written onto an external file. All arguments are ignored.

RESTore specs: REST ⟨DUMMY⟩

The arrays containing the specificity names and pointers are read from the file used in the SAVE command. All arguments are ignored.

The SAVE and REST commands, along with the CPS load and save commands, can be used to selectively store, retrieve, and experiment with several different hypothesized specificity structures.

Additional Notes on the Commands

The command structure is symmetric in the sense that for each command which does A to the cells and B to the sera, there is usually a command which does A to the sera and B to the cells. Symmetric pairs of commands are AC and AS, FC and FS, IC and IS, ICM and ISM, LC and LS, LCESP and LSESP, and RC and RS. D, END, H, REN, REST, and SAVE are self-symmetric in that they affect the cells and the sera in the same ways. CM is the only unpaired asymmetric command. Because of the similarities of the commands in each symmetric pair, each such pair is handled by the same subroutine. Processing the commands in this manner eliminates duplication of large amounts of coding.

Program Initialization and Modification

There are several types of program modifications which the user might wish to make:

(1) Once he has thoroughly explored a reaction matrix, he may want to set up the program to use a different one.

(2) If the user has constructed a set of specificity data which he does not wish to keep, he may want to erase it and start over again.

(3) He may want to change the number of specificities (presently 64) which the program can handle. This is not recommended, but it is possible. It necessitates making change (2) as well, and renders unusable any data which was saved in the save/restore file when the old number was in use.

(4) He may want to purge the CPS symbol table for one or more of the load/save modules.

(5) He may want to make changes in the coding (e.g., to implement a new command, or to make changes in the operation of an old one).

In order to make such changes as easy as possible, the following program design has been used.

With the exception of the main program, no module is used for both data storage and command processing. If a module is designed to execute a command, then it gets the cell and serum names and the relevant reaction matrix information by subroutine calls. All other data are passed to it from the main program in its argument list. This is done even to the extent of passing array dimensions, and redeclaring or reallocating any character strings or arrays whose dimensions depend on these numbers. Thus, such a module remains entirely unaffected by changes (1), (2), (3), and (4).

The main program, as well as the data storage modules, do contain data which must be retained in between each time the program is executed, and this information is affected by these changes. To aid in making such changes in these modules, each one contains an initialization subroutine. The use of various combinations of these subroutines aids in making each of the above changes.

Before making a change such as (5), the user should be thoroughly familiar with the design of the command processor, and all such changes should be in keeping with the design philosophy.

Final Remarks

The program was written in CPS because it was the only language available at the time which had the necessary combination of ATTENTION handling capabilities, storage capacity and data types, and file input/output capabilities. The limitations of CPS forced a number of compromises in program design. Dennis Rockwell has suggested that the operation of CPS's ON ATTENTION feature can be duplicated using TSO's STAX macro; this opens the way for implementation of a compiled version of the program running under timeshared PL/I. This would allow for complete restructuring of the subroutines, faster access of the specificity data using bit matrices instead of lists, and the inherently greater efficiency of an object code program over an interpreted one.

The present CPS version runs interpretively, and thus is somewhat expensive to use. However, this factor is mitigated by the rapidity with which the analysis can be carried out. It is suggested that repeated short sessions are more advantageous than fewer long ones. The computer programs used can produce information in a short time that it may take hours, days, or even weeks to completely assimilate. In the words of John Kemeny, "typing is not a proper substitute for thinking" *and conversely.*

References

1. Amos, D. B., and Ward, F. E. Immunogenetics of the HL-A System. *Physiol. Rev.* **55**, 206 (1975).
2. Balner, H., Cleton, F. J., and Eernisse, J. G. (Eds.). "Histocompatibility Testing 1965." Munksgaard, Copenhagen, 1965.

3. CEPPELLINI, R. Old and new facts and speculations about transplantation antigens of man. *In* "Progress in Immunology" (Bernard Amos, Ed.), pp. 973–1025. Academic Press, New York, 1971.
4. CIFTAN, M. Boolean analysis of histocompatibility data and genetic mapping. *Math. Biosci* **6**, 487 (1970).
5. Computer Workshop, Academisch Ziekenhaus Leiden. *Tissue Antigens* **2**, 147 (1972).
6. "Conversational Programming System (CPS) Terminal User's Manual." IBM Corporation Publ. GH20-0758-1, White Plains, New York, 1972.
7. CURTONI, E. S., MATTIUZ, P. L., AND TOSI, R. M. (EDS.). "Histocompatibility Testing 1967." Munksgaard, Copenhagen, 1967.
8. DAUSSET, J., AND COLUMBANI, J. (EDS.). "Histocompatibility Testing 1972." Munksgaard, Copenhagen, 1973.
9. KISSMEYER-NIELSEN, F. (ED.). "Histocompatibility Testing 1975." Munksgaard, Copenhagen, 1975.
10. NAU, D. S. "Specificity Covering: Immunological and Other Applications, Computational Complexity and Other Mathematical Properties, and a Computer Program." A.M. Thesis, Duke University, Durham, North Carolina, 1976.
11. NORTON, L. M., DIXON, J. K., RAY, J. G., AND KAYHOE, D. E. A heuristic program for evaluating histocompatibility typing data. *Comput. Biomed. Res.* **7**, 554 (1974).
12. RUSSELL, P. S., AND WINN, H. J. (EDS.). "Histocompatibility Testing." Natl. Acad. Sci. Publ. 1229. Washington, D.C., 1965.
13. WOODBURY, M. A., AND GALLIE, T. M. The mathematics of antigenicity and immunogenicity. Unpublished notes. Duke University, Durham, North Carolina, December 1972.