Discussion and correspondence
Automatic generation of payroll programs

M. H. Williams and E. V. C. Fielding

Department of Computer Science, Rhodes University, Grahamstown, South Africa

A system for automatically generating programs for a particular class of problem—in this case the class of payroll programs—is discussed. The technique used in the generation process is based on the stepwise refinement approach to problem solving. The method described is applicable to other similar problems and ultimately could be used in a general automatic programming system.
(Received February 1978)

1. Introduction

The term 'automatic programming' refers to a system which will automatically create a program from some specification. Initially (Sammet, 1969) this term was used in the narrow sense to refer to compilers (where the specification is simply a program written in a high level language). More recently (Knuth, 1974) the term has been used to refer to much more ambitious systems which in some way accept the definition of a problem from a user and from this construct a program to solve the problem.

Various workers are already engaged in the task of creating systems to solve general forms of problems (eg. Balzer *et al.*, 1974) while others are approaching the problem from the point of view of accepting an existing program and converting it to a significantly improved form (Cheatham and Wegbreit, 1972).

An alternative approach to automatic programming is to consider specific applications for which the techniques of solution are known but for which many variations in user requirements exist. If one can create a system which will adequately determine the user's needs and generate a unique program to solve each individual user's particular problem, this will provide a small step in the direction of the ultimate goal of automatic programming. Furthermore if the same technique can be applied to other types of problems and eventually the different systems can be merged into a single system, one may indeed obtain a limited form of the all purpose automatic programming system—one which may not be capable of handling 'any' problem but which will certainly be capable of handling the more common problems from a particular area.

One of the most common programs in existence today and certainly one of the most important in most institutions, is the payroll program. The payroll program is also a program for which user requirements differ widely.

The present paper describes a system, the aim of which is the automatic generation of payroll programs. This involves both the determination of user requirements and the generation of a unique program to meet each user's particular needs.

2. Structure of the system

The current approach to designing a system, which will provide for a variety of different user requirements, is to construct a package. In order to use a package, the user must decide which facilities he requires, set up the appropriate parameters and feed these into the package.

From experience the task of deciding on and setting up the parameters can be a lengthy one and one which requires someone with considerable computer expertise to perform. In one instance the setting up of a set of parameters for one computer payroll package which is currently being marketed, took one institution (with under a thousand employees) many man-months to complete. In general parameters may lead to problems unless they are well designed and documented.

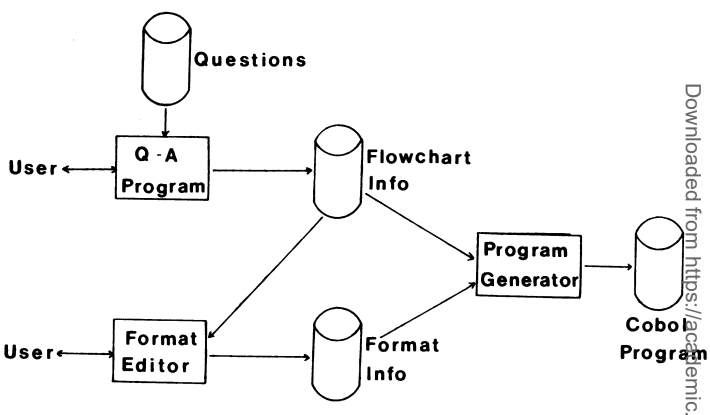

Fig. 1 An overall view of the system

On the other hand a question answering approach is a much more efficient and effective means of determining the user's needs. The question answering system can be designed to ask the user only those questions which are relevant, omitting those which from the user's previous replies are obviously irrelevant. Thus this approach was adopted here.

The question answering system used here is a table driven program for which the data is set up initially on a disc file. The data for each question consists of three components:

(*a*) the text of the question

(*b*) the expected answer

(*c*) the action.

The expected answer field contains a little procedure which must be executed once the text of the question has been displayed. It is responsible for displaying any additional information and reading in and checking the user's reply. It also performs certain actions on the basis of the user's reply.

The action field also contains a small procedure which is executed after the user's reply has been read and the expected answer procedure is completed. The action procedure is concerned essentially with the generation of 'flowchart boxes' on the basis of the user's reply and with deciding which question should be asked next.

The output from the question answering system comprises a set of flowchart boxes which contain most of the information necessary to construct the program. The only details which are missing at this stage are details about input/output formats. A separate program, the Format Editor, is responsible for setting these up. Thus when the user finishes with the question answering system, he is directed to the Format Editor which obtains the necessary information and stores this away on a disc file.

The final step in the process is the Program Generator. This

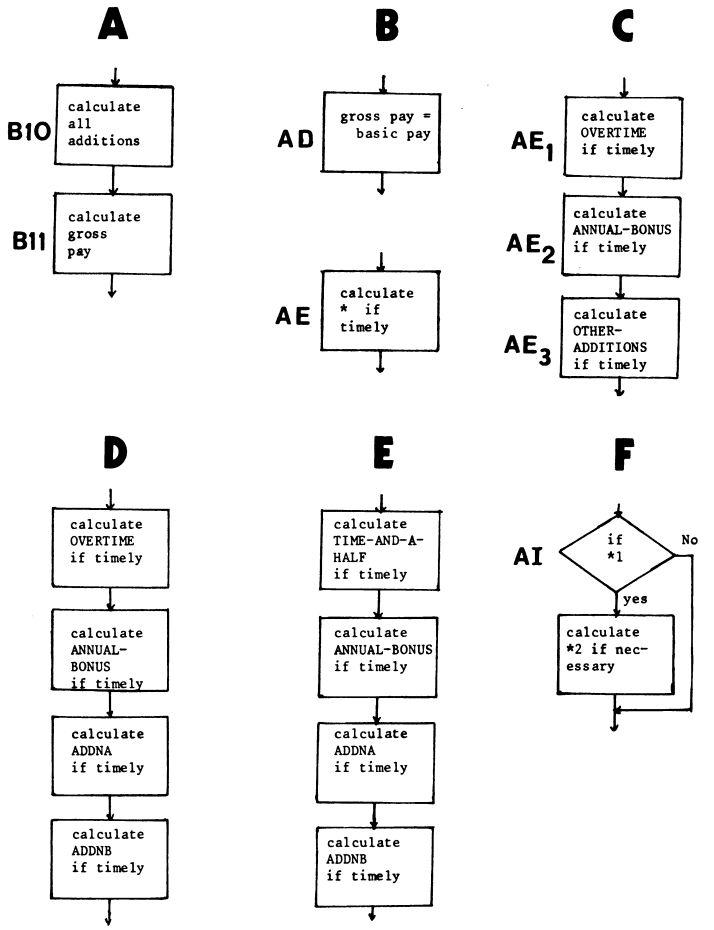

Fig. 2 (*a*) **A portion of the initial flowchart;**
(*b*) **possible replacement boxes;**
(*c*) **part of the flowchart after expansion by question 9;**
(*d*) **the same part after expansion by question 10;**
(*e*) **same as** (*d*) **after expansion for overtime;**
(*f*) **the pair of boxes which replace each box of type AE**

accepts as input the information set up by the question answering system and that by the Format Editor and uses these to generate a COBOL program which is tailored to the user's requirements (see **Fig. 1**).

3. The generation process

The strategy followed in the process of generating flowchart boxes is based on the principle of stepwise refinement. The first few questions determine the nature of the program to be generated (e.g. weekly payroll, single shot program, setup program, etc.); from these the initial flow diagram is determined. Each subsequent question, which is asked, is concerned with the refinement of boxes in this flowchart and, depending on the user's reply, one or more boxes in the flowchart may be replaced by other boxes with more specific information.

To demonstrate this, consider the portion of flowchart in **Fig. 2**(*a*). This represents part of the whole flowchart at an early stage in the question answering process. One will then encounter the question

Q.9. WHAT ADDITIONS?
WHAT ADDITIONS ARE THERE TO BASIC PAY/ STANDARD SALARY?
A) OVERTIME
B) ANNUAL BONUS
C) HOLIDAY BONUS
⋮
N) NONE
O) OTHER?

The expected answer procedure will read the user's reply—a string of characters—checking that each character lies in the range A to O. The characters are stored on a stack. The action procedure does the following:

1. If the answer is 'N', remove box B10 completely and replace box B11 by box AD (see Fig. 2(*b*)); goto question 13.
2. Otherwise replace box B10 by a series of boxes of type AE (see Fig. 2(*b*)) replacing the parameter * by the appropriate string. If 'A' occurs in the stack, set the overtime pointers; if 'O' occurs in the stack, set the 'other' pointers and go to' Q. 10. If the overtime pointers are set but not the 'other pointers, go to Q. 11 otherwise goto Q. 13.

Assuming that the user has replied giving the letters A, B and O, then box B10 would be replaced by the flowchart in Fig. 2(*c*). Now since the user has specified other additions (O), the next question to be asked will be:

Q.10 OTHER ADDITIONS
WHAT OTHER ADDITIONS ARE THERE?

The expected answer procedure reads one or more strings and stores them on a stack. The action procedure replaces the OTHER ADDITION box (in this case box AE_3) by one or more boxes of type AE with the appropriate addition name substituted in place of the parameter. Thus if there are two other additions, say ADDNA and ADDNB, the updated flowchart will look like that in Fig. 2(*d*).

The next question is responsible for handling different types of overtime and expanding box AE_1 accordingly, e.g. as in Fig. 2(*e*). One might then be asked about the frequency of each addition as follows:

Q. 15 FREQUENCY OF ADDITIONS
AN ADDITION MAY BE CALCULATED IF
(A) IT IS A WEEKEND

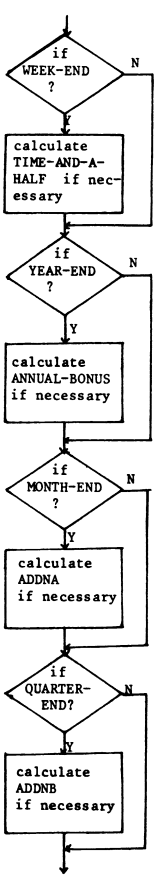

Fig. 3 The portion of the flowchart after question 15

(B) IT IS A FORTNIGHTEND
(C) IT IS A MONTHEND
⋮
(I) A SPORADIC INDICATOR HAS BEEN SET.
THE ADDITIONS ARE LISTED BELOW. USE THE ABOVE LIST TO INDICATE NEXT TO EACH WHICH FREQUENCIES ARE APPLICABLE.

In this case the action field replaces each box of type AE by the pair AI with the appropriate substitution of parameters. The result is shown in **Fig. 3.**

4. Conclusion

The question answering interface with the user is an effective way of establishing the user's requirements. Even someone with little or no computer experience can generate a payroll program within a matter of hours and a complex system can be set up in a short space of time.

The question answering system produces as output a flowchart structure. This has several advantages, the most important of which is that the use of a flowchart structure makes this part of the system to a certain extent language independent. It is possible to write a Program Generator for other languages, e.g. FORTRAN, in which case one could produce one's final generated program in a variety of languages.

The reason for producing the final generated program in a high level language instead of producing a machine code program directly is that the user may then inspect the result to check that it does suit his requirements. Thus if he should want a facility not catered for by the system it is possible to add the appropriate COBOL statements to the generated program. Since COBOL is a reasonably efficient language for this type of problem the loss of efficiency in generating COBOL statements is not particularly serious.

The use of the stepwise refinement approach meant that the questions had to be organised in such a way that information is obtained in order of increasing detail. This is the most logical way of presenting the questions and is the most useful from the point of view of manipulating the flowchart structure produced.

The system has been implemented using PASCAL on an ICL 1902T.

Acknowledgements

One of the authors, Miss E. V. C. Fielding, was in receipt of a CSIR bursary when this work was carried out.

References

BALZER, R., GREENFELD, N., KAY, M., MANN, W., RYDER, W., WILCZYNSKI, D. and ZOBRIST, A. (1974). Domain-independent Automatic Programming, in *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, pp. 326-330.

CHEATHAM, T. E. and WEGBREIT, B. (1972). A Laboratory for the Study of Automatic Programming, *AFIPS Conf. Proc.*, 40, pp. 11-21.

KNUTH, D. E. (1974). Computer Programming as an Art, *CACM*, Vol. 17 No. 12, pp. 667-673.

Predicting student success in an introductory programming course

L. J. Mazlack

Computing and Information Science, University of Guelph, Guelph, Ontario, Canada

Over several years, more than a thousand students have taken the same introductory programming course. The students were drawn from all academic disciplines and academic experience levels. No correlations were found between success in the course or its components when posed against academic programme, gender or semester in school. Additionally, when the IBM Programmer's Aptitude Test was administered to a smaller group of mathematically orientated students taking the same course, the predicative value of the test was found to be low.

There are several questions that immediately come to mind when constructing an introductory course. Some of them are:

(*a*) who should it serve?

(*b*) should the disciplines be separated?

(*c*) are the students from varying disciplines equally competitive?

(*d*) what is the potential performance difference due to the number of semesters of academic experience?

(*e*) does one gender perform better than the other?

(*f*) is it possible to predict performance by an aptitude test?

The importance of these questions varies from person to person. Answers which may appear to be self-evident to one person may be considered to be an emotional response to another. An empirical study was developed to shed some light on these questions. The course material was held constant over a period of several semesters. The student's performance was then correlated against a standard aptitude test and against various personal descriptions (semester in school, sex, etc.).

The three PAT (IBM Programmer's Aptitude Test) test components and PAT total score were correlated against the various course components. The highest correlations existed between student performance in the PAT scores and the midterm examination results. The midterm examination contained the least programming of any component in the course. The correlation with the pure programming and problem analysis component of the course (assignments) was very low. It logically follows that this type of test should not be administered to college level people as the results are meaningless and might have adverse effects on the test takers.

No significant correlation was found between academic performance and academic discipline. This is directly opposed to the often made assumption that a student's academic discipline is a good predictor of potential competency in programming. This implies that if a course is properly constructed and presented, there is no need to segregate students from differing academic disciplines due to concerns based on learning ability or interdiscipline competitiveness.

No significant difference was found in academic performance between the genders. The only observable difference is that women appear to be more consistent than men.

The correlations found between semester in school and academic performance were very low. This clearly indicates that in a properly constructed course, there is no reason to worry more about those beginning their university experience