A GLOBAL VIEW OF AUTOMATIC PROGRAMMING

Robert M- *Ba)zer*

USC Information Sciences Institute
4676 Admiralty Way
Marina del Rey, California 9C291

Abstract

This paper presents a framework for characterizing automatic programming systems In terms of how a task Is communicated to the system, the method and time at which the system acquires the knowledge to perform the task, and the characteristics of the resulting program to perform that task. Jt describes one approach In which both tasks and knowledge about the task domain are stated in natural language In the terms of that domain. All knowledge of computer science necessary to Implement the task Is Internalized Inside the system.

Preface and Acknowledgement

This paper represents the author's personal vlew of a global description of Automatic Programming. This view resulted from the author's dlscussions wlth and suggestions from numerous colleagues on this area for which he Is deeply indebted.

This paper Is a condensation of this view, taken from a larger work [1] which attempts to structure the field on the conceptualization expressed here. The interested reader should consult this work, which describes the issues In greater detail and contains supporting evidence for the views summarized here.

Introduction

The goals of automatic programming are deceptlve1y simple; namely, the effective utilization of computers. This Implies both simplicity of use and efficient application of the computing resources.

Thus, automatic programming is the application *of* a computing system to the problem of effectively utilizing that or another computing system in the performance of a task specified by the user. Although this is certainly what is meant by automatic programming, this definition does little to restrict the set of applicable computer systems included in the automatic programming domain. All compilers, operating systems, debugging systems, text editors, etc., fall into this domain and, In fact, the term "automatic programming itself was applied to early compiler systems during the 1950s.

This research Is supported by the Advanced Research Projects Agency under Contract No. DAHC15 72 C 0308, ARPA Order No. 2223/1, Program Code No. 3D30 and 3P10.

What is needed, therefore Is to either appropriately restrict the definition of automatic programming, or to provide a set of criteria used for measuring the acceptability or performance of an automatic programming system. Such a measure of system merit is extremely hard to arrive at but would contain the following man, machine, *and* system applicability components:

The efficiency of the resulting program;

The amount of computer resources expended In arriving at that program;

The elapsed time used in arriving at the resulting program;

The amount of effort expended by the user in specifying the task;

The reliability and ruggedness of the resulting program;

The ease with which future modifications can be incorporated;

and finally.

The range and complexity of tasks which can be handled by the system.

The Basic Model

One goal of any automatic programming system Is to allow its users to state their problems and any advice about Its solution In terms natural to the problem. Although most problems and their solutions can be most naturally described in the terrnms native to their fields, some can best be stated and/or solved In terms of a different field, such as mathematics. Occasionally, this other field is computer science, and the problem or Its solutlon is expressed in terms of data structures and their manipulations. Such descriptions, in terms other than those of the problem domain, are entirely satisfactory as long as they are *part of the conceptual* repertoire of the user and are not artificially Introduced to enable the system to comprehend the problem or process Its solution.

We therefore treat both the native terms of *a* Held and the terms of other fields which <u>users</u> have found useful to describe and conceptualize problems and solutions in that field *as* the problem domain terms of that field. With this definition, we conjecture that the solution of every computable problem can be represented entirely In problem domain terms as a sequence, which may involve loops and conditionals, of actions in that domain which affect a data base of relationships between the entities of the domain. Included either as part *of the* data base or as a separate part of the model. Is the history of the model (I.e., the sequence of actions applied to the model). This logically completes the model and enables questions or actions involving historical Information to be handled. In a strong sense, such a solution Is a direct simulation of the domain. The system models at each step what would occur In the domain.

The important part of the above conjecture Is that any computable problem can be solved, and hence described, In problem domain terms. This enables us to divide the solution into two parts, an external and an Internal part. The external part Is the problem specification given by the user In completely domaln specific terms. The requirements for such users is no longer a comprehensive knowledge of computers, but rather the ability to completely characterize the relevant relationships between entities of the problem domain and the actions in that domain. In addition, such users should have a rough awareness of the problem solving capability of the system so that they can provide additional help where needed In the form of more appropriate macro-actions, recommendations about the use of certain actlons, and/or Imperative sequences which will solve part or all of the problem In problem related terms.

The Internal part Is first concerned with *finding* a solution In problem related terms. if this has not already been provided by the user. Second, this part Is concerned with finding efficient solutions given the available computing resources. Such optimizations occur at two levels beyond what is normally considered optimizatlon. First, at the problem level, recognition that certain entities *and/or* relation ships *are i*rrelevant enables their removal from the model. Second, since only part of the state of the modelled domain Is required, and only *at certain* points in the solutlon process, rather than simulating the model completely at each step, the system can employ alternative representations which require less maintenance and which either directly mirror the required part of the domain state or allow such parts to be computationally inferred. Such representations may also enable *more direct* solution of the problem. It Is these optimizations which form the main distinction between the code generation part of an Automatic Programming system and current state of the art compilers.

Thus, our definition of an Automatic Programming system Is *one* which accepts a problem in terms of a model of the domain, which obtains a solution for the problem in terms of this model, and produces an efficient computer implementation of this solution in the form of a program.

<u>System Requirements</u>

There are seven facilities to be provided by, or criteria to be satisfied by future Automatic Programming Systems. The first is an interactive *system Interaction* between the system and the user Is required so that the specification can be given incrementally and any errors or discrepancies that arise in or from such specification can be handled as they occur.

Along with this Interactiveness the system should be very forgiving. It should allow great flexibility in the way and time at which information is specified. It should also be forgiving by allowing the user to change or retract previous communications with the system.

The second criteria is the amount of non-proceduralness used In the specification of the task to be performed. As far as possible the system should be told what to do rather than how to do it. There is a continuum between the statement of a problem as the transformation from an Initial state to a goal state, and the specification of how to perform such a transformation. Most of computer languge development can be viewed as a movement from specifying HOW to do something towards a statement of WHAT is desired. The level of non-proceduralness achi evable wlthin an Automatic Programming Sytem is directly related to the system's capability of turning goals Into actions and this Is dependent upon fts knowledge of how to acleve certain results in the problem domain. The ability of the system to achieve results in the problem domain Is used as the main distinction between non-procedural and procedural languages. Thus, problems must be stated in a language appropriate for that domain, I.e., one that can express the structure and interrelationships of the entitles within that domain, and one that users are familiar with for discussing and describing tasks and problems within that domain. Only with such a language can the system know how to achieve the desired results rather than being directly told how to produce the desired result. Some actions can, however, best be described In terms *of* bow to accomplish them rather than by the resulting state. Such procedural descriptions are quite acceptable as long as they are specified entirely in problem *domain* terms rather than implementatlon computer terms.

In addition to problem oriented specifications, the amount of Information that must be specified for the system to correctly process the problem must be reduced. This can best be done by removing details from the specification and allowing the system to fill them in. As with non-proceduralness there Is a continuum here, but the level that is desi red is one which omits from the specification all references to entities not contained within the problem domain (and which are not necessary for transferring knowledge between users of the field). Specifically, artificial references to the data structures of computer science (e.g.,lists, rings, arrays, symbol tables, and the like), that have been Invented as a means of specifying "how" rather than "what" to do must be avoided.

Next, a mechanism Is required for the modification of specifications that have been previously entered either because they don't work, or because the environment In which they are operating has changed, or because df fferent behavior Is desired. Such specifications should be given In terms of changes in deslred results, and, entirely within the language of the problem domain Itself.

Once a problem has been specified, a mechanism Is needed for Insuring that the system produced Is the one desired. This Is especially critical since the system will be greatiy augmenting the specifications. Discrepancies between the desired system and the one produced can arise from an Inadequate knowledge *of* the system about the domain, from a mis-statement by the user, or from a misinterpretation of something that was specified. Whatever the cause. It Is Important that the user can see the produced system In operation In his own terms. I.e., In the language of the problem domain Itself, so that he can check the expected behavior against the behavior produced. In addition, the system should be able to generate test Input data *so* that a wide range *of* behavior of the task specified can be observed by the user. If a discrepancy Is found, the user additionally requires capabilities to locate and Isolate the source of the discrepancy, and then, to modify it to obtain the desired result.

After a correct program has been produced, a mechanism Is needed for transforming It Into an efficient one. Such efficiency will rest on two kinds of informatlon. First, knowledge about the problem domain which enables alternative ways of performing the same task to be evaluated. Secondly, Information about efficient ways to utilize computers so that a total cost can be assigned to each of the different ways of solving the problem In domain terms. implied, but not stated In the above, Is that Automatic Programming systems don"t just automate programming. They also provide facilities which help the user move from an understanding of a task to be accomplished to a finished running system which performs that task. An Automatic Programming system Is a system which aids the user In all the steps from problem definition and design to final completed running programs.

To meet all the above criteria automatic programming systems require detailed Knowledge about the problem domain. The requirement for this knowledge li mits the , system's applicability to other areas, and hence, one measure of such systems Is the range of problem domains which they can adequately handle and their method of obtaining this knowledge.

The Four Phases: An Overvlew

Automatic Programming begins with the application of problem solving to problem statements rather than problem solutions; I.e., with the attempt by *a* computer system to obtain an understanding of the task being specified. Once the task has been understood. If it is not in process form. It must be transformed into one. This Is the traditional area of Artificial Intelligence and human program design. It must be verified that the resulting process model Is the one desired by the user and that It Is adequate for the user's problem. If not, It must be modified and transformed by the above steps and reverified. It must then be made Into an efficiently running program. This involves the automation of the ad hoc knowledge of computer science.

A complete Automatic Programming system thus consists *of* four major phases- Problem Acqulsition, Process Transformation, Model Verification, and Automatic Coding. Problem Acquisition Is the process by which the system obtains (1) a description of the problem to be solved or task to be performed In a form processable by the system, and (2) the knowledge needed to solve the problem. The result of this phase Is a well formed problem and knowledge base which can be manipulated by the system and transformed Into a high level process for solving the problem during the second phase. The third phase Is used to verify that this process is the one desired and that it is adequate for the problem solution. The fourth phase, Automatic Coding, fills In the necessary details, optimizes the process, and produces the actual code to solve the problem.

The Automatic Programming Model

One of the most striking and deep rooted features *of* the Automatic Programming model presented here Is the Interface It creates between a high level external specification of a problem which omits data structures which are not part of the domain and the internal implementation of that specificaton In an efftclent representation.

This choice of a basic Interface has predicated large parts of the entire model. Through this *choice* as the basic Interface wlthln the Automatic Programming model four Important gains are expected.

First, the complete mode] conjecture states that such a division Is feasible for stating and solving domain dependent problems.

Second, since the choice of a data representatton and the mat ntenance of is consistency occupy such a large portion *of* current programs, the size and complexity of spect fications without such representations should be drastically reduced over those which retain these representations.

Third, since so much detail has been removed from the specification It Is easier for. the system to understand what the task Is rather than getting lost In the details of what is going on.

Finally, since the problem has not been overspecified with a particular choice of representation so that the problem was expressible, the system Is now free to choose a representation that will efficiently solve the problem at hand. The system has been given Increased flexibility in Its choice and may well outperform humans In correctly making representation choices; not because the system Is more Intelligent than the user, but because it can cycle through more possibilities and bring to bear a greater level of effort In such optimizations than any user Is willing or able to Invest In such issues.

Problem Acquisitlon

The Problem Acquisition phase Is concerned with obtaining an understanding of the users problem and the domain In which It exists so that the Process Transformation phase can attempt to find a sequence of transformations or operations in that domain which will obtain the solution required by the user. Thus, the Problem Acquisition phase is concerned with building a model of the users domain which represents the interactions between the entities of that domain and the effect on those entities by the allowed transformations or operations applicable within the domain. It is our primary contention that only through the development of such a model of the user's domain can the Automatic Programming system have any degree of generality In the domains for which It Is applicable.

Currently, a] 1 such models of user domains have been coded into a system. It Is proposed here that such models can be specified to the system by Its users and that through these models the system can acquire the knowledge necessary to solve problems within these domains and to understand what Is required for such a solution. The two main Issues, then, are what constitutes an adequate and appropriate model and how Is such a model specified or communicated to the system.

There are basically two types of models,analogical and fregean. Analogical models bear a strong resemblence to the structure of the object being described, such as the floor plan for a room, or the diagram model used, by Gerlernter In his geometry proving program.

Fregean systems, on the other hand, are linguistic or relation based, l,n which expressions are built up on the relation .between functions and arguments to those functions.

Since one of the basic goals of the Automatic Programming system is the generality of problem domains that It Is willing and capable of handling, the fregean model approach has been chosen.

We can now define the adequacy and appropriateness of the models for an Automatic Programming system. A fregean model is adequate If it contains a complete enough description of the relations between the entitles in the problem domain that a sequence of operations or transformations on this model can be built to solve the problem posed by the user. This Is what was referred to as the complete model In the Basic Model Section. Thus, the adequacy of a model Is dependent upon the use of that model to solve the problem. Operationall, this requires that the Automatic Programming system is capable of finding the complete set of applicable transformations on the model and can calculate the consequences of each of these actions. The appropriateness of the model is a measure of how well suited the available transformations are to solving the problem at hand, I.e., an adequate model can be made more approprlate by addi ng to It non-primi tlve transformations made up of a sequence of primitive ones, which are suitable building blocks for the problem being posed. The model rr,dy also be made more appropriate by including recommendatlons about the suitablliy of alternative strategies for sequences of model transformations.

Users can significantly reduce the well-known problem of building a powerful general purpose problem solver by tailorlng the specified model to make It *more* appropriate for the problem at hand.

The state of the art In natural language understanding appears adequate for the description of problems and of models and beyond our ability to utilize the information thus obtained, and hence, should not be a bottleneck in an Automatic Programming system. Evidence for this viewpoint comes from the work of Woods in the Moonrocks Program, from Winograd in the blocks Descri ption Program, and from Martin Kay in the Mind System. Each of these systems represents an alternative linguistic technology and each Is capable of handling a wide range of linguistic forms within the domain of its competence.

The basic viewpoint, then, is to process the user's natural language communication with the understanding that It is meant to convey to the Automatic Programming system a model of his problem domain. Towards this end the system can extract entitles and the relationships between them from the communication. It can further query the user as to the relationships between entitles which have not, as yet, been explicitly specified but which have been inferred by the previous communication. Such inferences by the system about the Incompleteness of the model require a sophisticated understanding not only of the communication but of the types of models used for problem domain specification. unfortunately, *our* sophistication In both these areas Is quite limited. In communication we need to be able to understand how Information Is ordered for presentation, how context Is established and utilized, how the capabilities of the recipient effects the communication, and how these capabilities are perceived by the speaker. In modelling we need to have a space of possible models, an understanding of how the parts of a model Interact, a means for recognIzIng incompleteness and inconsistencies In models, a means for obtaining all the allowed operations on the model, and the means for transforming the models with these operations.

Process Transformation

Our contention Is that the main activity In programming Is not finding a solution but In finding *a* solution which drops out the Irrelevancles and which abstracts the necessary processing so that it can be efficiently implemented. It is recognized that this Is a strong contention, but In most programming problems It Is felt that a solution is known and the main concern is In finding a more efficient one. This Is not optimization In the normal sense of the term. The concern, rather. is with finding irrelevancies in the complete model and representational abstractions based on the required processing of that model. Once these logical representations have been found they must be efficiently Implemented.

The above contention. If true, greatly shifts the emphasls within the Process Transformation Phase from that of a general problem solver solving problems In a domain Independent way to modifying a solution so that it does not maintain any irrelevant portions of the complete model and which abstracts the relevant portions into a more effident representation for the processing required. Together with Problem Acquisition, the ability to find representational abstractions and transform complete model solutions into ones which utilize these representations represent the matn technological deficiencies with obtaining an Automatic Programming system.

Model Verification

Although the Automatic Coding phase will produce only correct code. Program Testing cannot disappear. This Is because the Problem Acquisition Phase and the Process Transformation Phase will undoubtedly employ a number of heuristies and may very well Incorrectly Interpret either the problem statement or the allowed transformations that can occur In the user's model. Because of this, the user must verify that the system created is the one that he desired.

The technology for this is at hand. It consists of today's methods wherein a test case is given to the system and its performance Is used to validate the model that it constructed. Additionally, the system can aid the process by generating test cases of its own which probe areas of which It Is uncertain and which could have led to either misunderstanding or incompleteness in the original model. One might also expect that program debugging would disappear, but for very similar reasons It too will remain under Automatic Programming. if there is a disparity between the user's model and the system's model, then the reason for this disparity must be obtained.

Automatic Coding

Automatic Coding Is concerned with finding an effletent computer implementation of the process description obtained from the proceeding phase. This description does not yet Include a choice of data representations, but does specify the major processing elements and sequences. It Is intended that this phase will not need any domain specific knowledge except for input frequency and distribution information. The major logical representation and processing decisions have already been made by the Process Transformation phase.

Of all the phases in the Automatic Programming System, the Automatic Coding one is the one essential component of any Automatic Programming System. Without it the system cannot produce programs, and hence, though It may be useful It Is not an Automatic Programming System.

Most people are not truly creative when they reorganize sections of their program to increase efficiency. Rather than inventIng totally new representations, they appear to select one out of an Ill-defined set of such possible representations and to adapt and modify It to function In the current situation. This is probably the main challenge to the Automatic Coding phase, the ability not only to cycle through a set of alternative representations but to adapt and modify them to the existing situation. Such an ability would vastly increase the appl[cableness of a small set of alternative representations.

From such Automatic Coding studies, one would expect to see both a set of heuristics and a calculus, eventually, for data representation choices.

Summary and Conclusions

The definition of automatic programming started with a goal, namely, reducing the effort required to get a task running on a computer. From this a framework was adopted in which the external characteristics of Automatic Programming Systems could be described in terms of:

1. The terms in which the problem is stated;

2. The method and time at which the system acquires the knowledge of the problem domain;

3. The characteristi c of the resulting program.

The choices we made are;

1. problem statement In natural language in terms of the problem domain.

2. Knowledge about the domain acquired interactively in natural language in terms of the complete model of the problem domaln.

3. Resulting programs which are optimized with respect to data representations, control structure, and code.

This approach requires significant advances In Artificial Intelligence techniques, in such areas as knowledge representation, Inference systems, learning, and problem solving, and In the codification of programming knowledge in the areas of data representations, algorithm selection, and optimization techniques.

Other choices could certainly be made, and the resulting systems might look very different. One particular set of choices, suggested by. Al Perils, represents a system based on a completely different paradigm. His system is predi cated on incremental growth from an accepted base, namely, FORTRAN. This Idea Is to Increase the declarative parts of the language at the expense of the procedural. The declarative parts are In turn replaced by a series of questions from the system which specify how a defined concept Is to be used. For instance, the concept "array* might generate queries to see whether the size was dynamic at run time, whether insertions or deletions are being done, whether the elements are homogeneous, and whether they are accessed sequentially. From such questions and the programs which use these concepts, an optimal representation can be chosen.

In this system, higher levels occur when enough example systems have been generated and understood by humans that someone can codify this knowledge and Introduce a new level of semantics and questions.

There Is no doubt that such a system would be useful, and It has the appealing attribute that the facilities of the system can be incrementally expanded from a widely distributed and available base. Also, such a system could Instantly be used by existing programmers who could gradually learn to use the new facilities on top of their ability to use FORTRAN.

On the other hand. it Is not clear how far such a technique can be pushed, and whether this Is the best way to achieve the goal except In the short-term.

Automatic programming systems, however realized, would substantially reduce the effort and training required by Its users and would enable the subsystem produced to more closely reflect the intents of their designers.

So much time, money, and effort is currently being expended, and even greater amounts forecasted In the future, for the creation of software products, that the potential benefits from automatic programming systems are enormous. Therefore, since the required technologies seem feasible, such systems, utilizing either the approach outlined In this paper or various others, should be extensively Investigated.

References

1. R. M. Balzer, Automatic Programming. Institute Technical Memorandum. USC/Information Sciences Institute, September 1972.