OVERVIEW OF AN EXAMPLE GENERATION SYSTEM

Edwina L. Rissland
Elliot M. Soloway
Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

ABSTRACT

This paper addresses the process of generating examples which meet specified criteria; we call this activity CONSTRAINED EXAMPLE GENERATION (CEG). We present the motivation for and architecture of an existing example generator which solves CEG problems in several domains of mathematics and computer science, e.g., the generation of LISP test data, simple recursive programs, and piecewise linear functions.

I INTRODUCTION

The ability to generate examples is an important process in many domains. Generating examples goes hand-in-hand with trying to prove theorems, distinguish between competing theories, validate programs, and design plans. For instance, in theorem proving one must try out the reasonableness and truth of one's conjectures with special cases in the alternating process of "proof and refutation" ([1], [6], [7], [4]). Examples play a similar role in the writing and testing of computer programs. In the planning domain, scenarios of actions can be thought of as examples and a successful planner must be able to generate and modify them [3]. Thus, example generation is complementary to many processes such as proving, planning and debugging. In subsequent sections, we first present an analysis of this activity, and then describe a computer system built on the basis of this analysis which does in fact generate examples which meet specified constraints.

II THE CEG MODEL

From protocol analyses of experts and novices working CEG problems in mathematics and computer science, we developed the following description of the CEG task [10]. When an example is called for, one can search through one's storehouse of known examples for an example that can be JUDGEd to satisfy the desiderata. If a satisfactory match is found, then the problem has been solved through SEARCH and RETRIEVAL. If a match is not found, one can MODIFY an existing example, judged to be close or having the potential for fulfilling the desiderata. If generation through modification fails, one often switches to another mode in which one CONSTRUCTS an example by instantiating certain models and principals or combining more elementary exemplars.

The CEG model we use thus consists of processes that RETRIEVE, JUDGE, MODIFY and CONSTRUCT examples.

III THE CEG SYSTEM

The CEG system described here is written in LISP and runs on a VAX 11/780. In addition to solving CEG problems concerning data and simple programs in the LISP domain [11], it is being used to solve CEG problems in a number of other domains [12]: the generation of descriptions of scenes for use in conjunction with the VISIONS scene interpretation system [2]; the generation of action sequences in games; and the generation of piecewise linear functions.

The flow of control in the Example Generator is directed by an EXECUTIVE process. In addition, there is: (1) the RETRIEVER which searches and retrieves examples from a data base of examples; (2) the MODIFIER which applies modification techniques to an example; (3) the CONS'ER which instantiates general "model" examples, such as code "templates" ([9], [16]); (4) the JUDGE which determines how well an example satisfies the problem desiderata; (5) the AGENDA-KEEPER which maintains an agenda of examples to be modified, based for instance on the degree to which they meet the desiderata or possess "epistemological" attributes ([8], [9]).

The components use a common knowledge base consisting of two parts: a "permanent" knowledge base which has an Examples-space [9] containing known examples, and a temporary knowledge base which contains information gathered in the solution of a specific CEG problem. In the Examples-space, an example is represented by a frame, which contains information describing various attributes of that example, e.g., epistemological class, worth-rating. Examples are linked together by the relation of "constructional derivation," i.e., Example1 ---> Example2 means that Example1 is used in the construction of Example2. The temporary knowledge

base contains such information as evaluation data generated by the JUDGE and proposed candidate examples created from known examples by the MODIFIER.

IV AN EXAMPLE OF A CEG PROBLEM

In the context of examples of LISP data elements, an example of a simple CEG problem would be the following:

> Give an example of a LISP list of length 3 with the depth of its first atom equal to 2.

(Examples such as this are needed when debugging and teaching.)

Suppose the permanent knowledge base only contains the lists

(A B C), (0 1), (A), ( );

the first two lists are standard "reference" examples, the third, a "start-up" example, and the fourth, a known "counter example," i.e., an example which often is handled incorrectly by programs. Since the knowledge base does not contain an example which completely satisfies the desiderata (i.e., the RETRIEVAL phase fails), the system enters the MODIFICATION phase. Since the list (A B C) satisfies two of the three constraints, and thus has a higher "constraint-satisfaction-count" than the other examples, it is placed as the top-ranking candidate for MODIFICATION by the AGENDA-KEEPER and modifications are tried on it first.

The candidate example, (A B C) is analysed and found to be lacking in one respect, namely, the depth of its first atom, A, must be made deeper by 1. The system accomplishes this by adding parentheses around the atom A to create the new example

((A) B C).

This list meets all the constraints specified; as it is a solution to the problem, it is entered into the Examples-space as a new example constructionally derived from (A B C). Thus, if the following problem is asked of the system,

> Give an example of a list of length 3, the depth of the first atom is 2, and the depth of the last is 3.

the system can use the newly constructed example in its attempt to satisfy the new problem.

V THE HANDLING OF CONSTRAINTS

The handling of constraints is especially important in the MODIFICATION phase where an existing example is modified to create a new example that is no longer deficient with respect to the constraints. For example, if there were more than one unsatisfied constraint, one could attempt to rectify a candidate example along each of the unsatisfied dimensions. Consider the following problem:

> Give an example of a list of 0's and 1's, which is longer than 2, and which has at least one element deeper than depth 1.

One could modify the list (0 1) to meet the unsatisfied second and third constraints by adding at least one more element, say another 1 to give the list (0 1 1), and then modify this new list by adding parens around one of the elements. Alternatively, one could add parens and then fix the length, or even do both 'at once' by appending on an element such as (1) to the original list (0 1).

In this example, there are many ways to modify the original list to meet the constraints, and the order of modification does not much matter. One can "divide-and-conquer" the work to be done in a GPS-like manner of difference-assessment followed by difference-reduction since the constraints are independent [5].

However, in other cases the order of difference reduction matters greatly. For instance, consider the problem:

> Give an example of a list of length 5 with an embedded sublist of length 2.

Suppose one is working with the list (A B C). If one first rectifies the length by adding two more elements, such as 1 and 2, to create the list (A B C 1 2), and then modifies this list to have the needed embedded list by "making-a-group" of length 2 around any of the first four elements, say to arrive at the list (A B C (1 2)), one has also modified the length as a side-effect of the grouping modification, i.e., one has messed up the length constraint. In other circumstances, it possible to set up totally contradictory constraints where satisfaction of one precludes satisfaction of the other. Thus a purely GPS approach is not sufficient to handle the complexity of constraint interaction. We are currently investigating the constraint interaction problem as well as issues concerning maintenance of agendas. For example, we are looking to employ planning-type control mechanisms for dealing with the constraint interaction problem. ([13], [3])

VI THE LARGER PICTURE

An application of our CEG system is in an intelligent computer-assisted instruction tutoring environment. We are currently building a tutor to

teach students about programming languages such as LISP and PASCAL ([14], [15]). In this context, CEG will serve the tutor in two ways: (1) it will generate examples to the specifications set by the tutor; and (2) it will evaluate student generated examples for the tutor. The same JUDGE used by CEG can be used to evaluate a student's example and help track down his misconceptions and bugs through analyses of differences between what the tutor requested and what the student acutually generated.

In the future, we also plan to incorporate adaptation into the system. For example, the system can keep track of the performance of the various example ordering functions and choose the one that has the best performance. Also, we plan to apply hill-climbing techniques to the modifying processes themselves. That is, since there are alternative ways to massage and modify an example, those routines which lead to the most succeses should eventually be selected first. Adaptation on the modification techniques will be particularly important if the system is to be able to improve its performance, and thus "learn" from its own experience.

The current implementation is only a "first-pass" and does not capture the richness of the CEG model. Nonetheless, we feel that it has demonstrated the utility of this model and we feel that subsequent implementations incorporating the characteristics of additional task domains should provide us with a rich environment to continue our investigation into the important process of example generation.

REFERENCES

[1] Bledsoe, W. (1977) A Maximal Method for Set Variables in Automatic Theorem Proving, Univ. of Texas at Austin, Math Dept. Memo ATP-33.

[2] Hanson, A. and Riseman E. (1978) VISIONS: A Computer System for Interpreting Scenes, in Computer Vision Systems, Hanson and Riseman, Eds., Academic Press, New York.

[3] Hayes-Roth, B., and F. Hayes-Roth (1978) Cognitive Process in Planning, Rand Report R-2366-ONR, The Rand Corporation, CA.

[4] Lakatos, I. (1963) Proofs and Refutations, British Journal for the Philosophy of Science, Vol. 19, May 1963. Also published by Cambridge University Press, London, 1976.

[5] Newell, A., Shaw, J., and Simon, H. (1959) Report on a General Problem-Solving Program. Proc. of the International Conference on Information Processing. UNESCO House, Paris.

[6] Polya, G. (1973) How To Solve It, Second Edition, Princeton Univ. Press, N.J.

[7] Polya, G. (1968) Mathematics and Plausible Reasoning, Volumes I and II, Second Edition, Princeton Univ. Press, N.J.

[8] Rissland (Michener), E. (1978a) Understanding Understanding Mathematics, Cognitive Science, Vol. 2, No. 4.

[9] Rissland (Michener), E. (1978b) The Structure of Mathematical Knowledge, Technical Report No. 472, M.I.T Artificial Intelligence Lab, Cambridge.

[10] Rissland, E. (1979) Protocols of Example Generation, internal report, M.I.T., Cambridge.

[11] Rissland, E. and E. Soloway (1980) Generating Examples in LISP: Data and Programs, COINS Technical Report 80-07, Univ. of Mass, (submitted for publication).

[12] Rissland, E., Soloway, E., O'Connor, S., Waisbrot, S., Wall, R., Wesley, L., and T. Weymouth (1980) Examples of Example Generation using the CEG Architecture. COINS Technical Report, in preparation.

[13] Sacerdoti, E. (1975) The Nonlinear Nature of Plans, Proc. 4th. Int. Joint Conf. Artificial Intelligence, Tbilisi, USSR.

[14] Soloway, E. (1980) The Development and Evaluation of Instructional Strategies for an Intelligent Computer-Assisted Instruction System, COINS Technical Report 80-04, Univ. of Mass., Amherst.

[15] Soloway, E., and E. Rissland (1980) The Representation and Organization of a Knowledge Base About LISP Programming for an ICAI System, COINS Technical Report 80-08, Univ of Mass., in preparation.

[16] Soloway, E., and Woolf, B. (1980) Problems, Plans and Programs, Proc. of the ACM Eleventh SIGCSE Technical Symposium, Kansas City.