EGS: A Transformational Approach to Automatic Example Generation

Myung W. Kim

Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712

ABSTRACT

THIS *paper describes a constraint transformation approach to automatic example generation and its implementation   In this approach examples are generated as the result of successive transformations of thf constraint formulas   Such transformations are carried out based on various forms of knowledge   Systematic global simplification, largely based on declarative knowledge, mitigates tht impact of applying problem-specific and efficient procedural knowledge, with a uniform problem representation scheme   The approach suggests a general framework for example generation in which a language for describing examples can be defined   ft also combines a general formal reasoning capacity and problem-specific procedural knowledge, to achieve both generality and efficiency   The implemented system has proven to be expressively powerful and efficient for a variety of applications*

1. Introduction

Examples are important in Artificial Intelligence   They are critical to machine learning, automated reasoning, knowledge representation, intelligent human interface, and computer-aided instruction |25. 13. 7, 18. 21, 4).   Examples illustrate abstract notions and complicated procedures, provide us with a source of information on which ideas and concepts are developed, and serve a« a tool with which hypotheses can be validated.

Generating examples is, however, a difficult task in general   A problem for example generation can be specified as a constraint - a conjunction of conditions each of which must be satisfied simultaneously. Those conditions usually interact with each other in complicated ways [24| It is often the case that an example generation system is used as a component of a larger system   In such cases, it is important that the example generator be efficient enough that it does not detract from the performance of the larger system

Example generation differs from database retrieval or problem solving   Database retrieval deals with only the previously sorted and stored data items, whereas example generation often requires constructing novel examples where known examples fail   Problem solving usually involves finding a solution or solutions, largely ignoring the issue of finding good solutions.   However, example generation is constantly concerned with choosing *good examples* from among many possibilities.   This problem appears difficult to solve because it involves highly abstract notions like utility, motivation, and objective.

Automatic example generation raises two fundamental issues. *representation* and *evaluation*   The representation problem addresses the questions what is an example and how should it be represented, the evaluation issue deals with methods for evaluating whether and/or how well a proposed example satisfies the initial constraint   In most cases *example* is not clearly defined   Evidence, samples, models, diagrams, and even arbitrarily defined data structures are often referred to as examples   The vagueness in the meaning of *example* causes difficulties in making use of the existing methodologies and tools for powerful reasoning; such reasoning appears critical to example generation   For an effective and efficient evaluation capability it is required that the involved concepts be defined unambiguously or somehow related to their effective computational procedures   It is quite natural that an automatic example generation system should be based on the rigorous formalism of its problem domain

Due to such difficulties, the approaches to automatic example generation in the current literature are problem application specific or limited in expressive power   In the transformational approach examples are generated as the result of successive transformations of the constraint formula   This approach suggests a general framework for example generation and also provides a uniform representation scheme for problems in which problem-specific procedural knowledge can be applied to achieve efficient as well

The Example Generation System (EGS) presented here has adopted the Boyer-Moore theory [2| as the underlying domain formalism   Several advantages are immediate

- The Boyer-Moore theory provides a general and expressively powerful language

- The problems of representation and evaluation can be easily solved

- EGS can share with the Boyer-Moore theorem prover, which implements the Boyer-Moore theory, a substantial portion of its reasoning capability and knowledge base

- The theorem proving environment serves as a rich source of problems for application of examples |7, 18. 23. 10, 11)

EGS, written in INTERLUISP, currently running on a DEC-2060 under TOPS-20 has been implemented on top of the Boyer-Moore theorem prover   In EGS, a constraint is specified as a (well-formed) formula in the Boyer-Moore theory   The variables of tbe constraint, called *constraint variables,* are interpreted as existentially quantified. An *example* is defined as an Assignment of *QiXJTEd constants* to the constraint variables, this assignment causes tbe constraint formula to evaluate to non-F.   A *counter erampU* is

similarly defined except that the assignment causes the constraint formula to evaluate to F. For instance, the assignment '((X . (QUOTE (A B)))) is an example of (LISTP X), whereas '((X . (QUOTE 3))) is a counter-example However, ((X   (CONS U V)) is not an example of (LISTP X) since X is assigned a non-QUOTEd constant

The example generation problem in EGS is specified in terms of constraints   The EGS command.

```
FINDX((AND (MEMBER X L)
           (MEMBER Y L)
           (NOT (EQUAL X Y))
           (EQUAL (REVERSE L) L))
       NIL)
```

may be read as-

Find examples of list-palindromes containing at least two difftrent members

FINDX is a top-level command for example generation   Its first argument is a constraint and its second argument specifies the resource limits such as a time limit or the maximum number of examples to be generated   The output might be-*

```
((L   (QUOTE (A B A)))
 (X . (QUOTE A))
 (Y   (QUOTE B)))

((L   (QUOTE (C (A) C)))
 (X   (QUOTE (A))
 (Y   (QUOTE C)))

((L . (QUOTE (1 NIL NIL 1)))
 (X   (QUOTE 1))
 (Y   (QUOTE NIL)))
```

2. The Boyer-Moore Theory and Their Theorem Prover

The Hoyer-Moore theory is obtained from the propositiooal calculus with variables and function symbols   Its three principles   the induction principle, the shell principle, and the definition principle characterize the theory.   Variables in a formula are implicitly universally quantified. Lacking quantifiers and allowing recursion as an alternative gives rise to the constructive nature of the theory. The language of the theory can be viewed as a (functional) programming language since function definitions (admitted) are like programs. In fact, the language of the theory is akin to that of the pure LISP.

When a function is defined, the Boyer-Moore theorem prover generates compiled LISP code corresponding to the definition [3]. This provides a simple and yet efficient interpreter for the language. The interpreter, if given a term together with a proper environment, will compute the terms value with respect to the environment. This feature, in fact, gives a straightforward solution to the evaluation problem

3. The Implementation of EGS

EGS has been implemented as *task-driven* in that examples are generated as the result of iterating the task generation and task performing sequence   A *task descriptor,* which describes a task, is a list of *task srorr, task opt rat or, hard-list, soft-ltst,* and reference-list.   The hard-list is a list of condition formulas assumed conjoined. The soft-list is an association list,, each component of which is of the form (v   t) corresponding to a substitution equality[2] (EQUAL v t). The reference-list contains information needed in performing the task   For example

```
(78 TEST
    ((MEMBER X (CONS U V))
     (NOT (NUMBERP U))
     (LESSP (QUOTE 7) X)
     (PRIME X))
    ((L  (CONS U V)))
    (CLUE (PRIME X) FLG NIL))
```

describes a TEST task   In a task descriptor, the hard-list and the soft-list together can be considered as a *subgoal* and we call the pair of the hard-list and the soft-list a *context*   In this paper *subgoal* and *context* are used interchangeably

Each task operation corresponds to a particular transformation of formulas.   Such transformations include partially instantiating variables with constants, analyzing terms by cases, solving equations, and unfolding definitions   Simplification is a non task-specific global transformation, which is invoked after a task-specific transformation is completed.   Generally a task transforms formulas of the hard-list to create subgoals   Tasks are maintained in an agenda in the order of task scores; the task on the top of the agenda is performed.   The result of performing a task, in turn, is further processed to produce more new tasks   EGS repeats this sequence until any termination condition is met - the agenda becomes empty, the specified number of examples are generated, or the time limit is exceeded.   An illustration of EGS example generation will be given later.   The task-dn\en structure of EGS with agenda can be viewed as best-first search   Its search space can be represented as a tree, each node of w hich corresponds to a task

EGS consists of six major functional components. *Preprocessor, Executive, Task (Unirator, Task Performer, Simplifier,* and E*valuator* and a global data structure *Task Agenda*.   Also the Knowledge *Bast* maintains knowledge of various forms.   Figure 1 depicts the general architecture of EGS.   The function of each component is described in the following.

Preprocessor

This component checks the syntax of the input constraint and also translates the constraint, which may be input in an abbreviated format, into the corresponding well-formed formula.   For example, (AND p q r) is translated into (AND p (AND q r)).

Executive

The Executive manages the flow of control in EGS.   It controls other components of EGS to maintain the proper sequence of example generation   Control loops through the task generation and ta,sk performing sequence until any termination condition is met.

An equality of the form (EQUAL lhs rhs) where lhs(rhs) is a variable and rhs(lhs) is a term not containing the lhs(rhs) variable.

Task Generator

This component generates tasks from contexts resulting from performing, a previous task   A task is generated by assigning an applicable task operator and attaching the collected task-relevant information to the context   This task is scored with its plausibility. The scoring algorithm is

Figure 1: EGS Architecture

$$SCORE = (C1 * OP\text{-}SCORE)$$
$$+ (C2 * S\text{-}SCORE)$$
$$+ (C3 * CREDIT)$$

OP-SCORE   the score associated with the task operator,
S-SCORE   the score computed based on the syntactic
          complexity of the formulas In the hard-list,
CREDIT   the score for extra credit,
and C1, C2 and C3 are the weight coefficients.

This scoring mechanism provides EGS with substantial controllability

Task Performer

Lach task operator is associated with a unique process which generates a list of subgoals (contexts)   Performing a task involves retrieving the relevant information and applying the process corresponding to the task operator to the context of the task   The following are the functional descriptions of the task operators

- START marks the initial task for a single session of example generation

- TEST retrieves *known examples* corresponding to a clue and tests each of the examples with the hard-list until the fixed number of successful tests are obtained   A suhgoal for each of the successful tests is generated.

- SOLVE invokes the built-in *Linear Solver* to solve the simultaneous linear equations/inequalities or applies the *solver* knowledge to directly solve a condition or a set of conditions

- ANALYZE applies the *rase* knowledge to do case analysis.

- DOPAND opens up the definition of a selected function in the hard-list. For an DOPAND task, each of the resulting subgoals is temporarily inactivated by making a RECALL task.

- ASSEMBLE reconstructs an example based on information from the soft-list

- RECALL activates an inactive suhgoal for being processed by other task operations
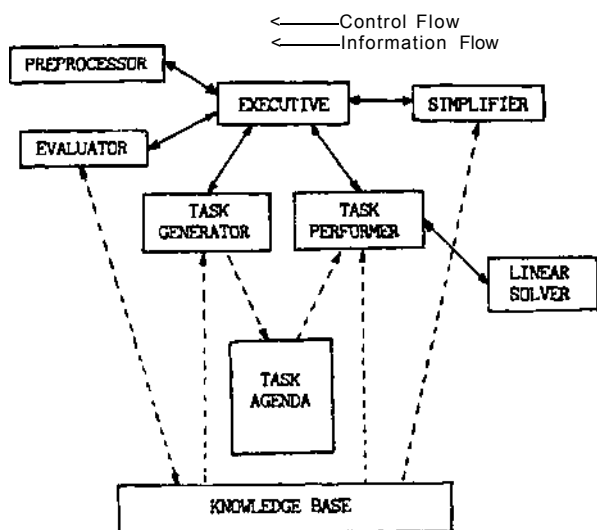
Simplifier

The major functions of the simplifier are

1 to transfer sequentially each substitution equality in the hard-list onto the soft-list, after carrying out the corresponding substitution on the formulas of the hard-list,

2 to rewrite condition formulas on the hard-list to make them simpler and easier to handle

The simplifier is generally invoked on each of those subgoals resulting from performing a task   Simplification possibly eliminates a subgoal or splits it into more subgoals   The resulting subgoals are then subject to furl tier processing by the task generator

Some of the important advantages of simplification are as follows

- By earning out a global simplification, the simplifier relieves the task generator of the need to care about mundane details

- By normalizing formulas, the simplifier allows the task generator to be able to relatively easily predict the patterns of formulas to be encountered

t By eliminating apparently unsatisfiable subgoals, the simplifier enables EG*S* to focus more on the promising ones

- The fact that the simplifier makes use of rewrite lemmas provides EGS with some degree of extensibility.

Evaluator

This component evaluates how well the proposed examples satisfy the initial constraint   This evaluation function would be important for the system to be able to learn or self-organize from experience Currently it only judges whether a proposed example satisfies the initial constraint

4 . The Knowledge Base of EGS

EGS's ability to generate examples critically depends on its knowledge base *EGS* employs various forms of knowledge, heuristic information, definitions, theorems, known examples, and equation solvers in the form of production rules, etc   Under each concept (function) name, knowledge is frame-structured   Knowledge is divided into the knowledge shared with the Boyer-Moore theorem prover and EGS-specific knowledge (boxed under X-FRAME in Figure 2).

Definitions and LISP Code

When a function is being defined, the Boyer-Moore system generates LISP code, which implements the defined function in the running LISP environment [3| This code is invoked when a term is evaluated   Evaluating terms is extensively required when ft TEST task is performed or a proposed example is verified by the evaluator

The symbolic definitions of functions are also important. Performing an EXPAND task retrieves the definition of a selected function and expands the corresponding function call in the hard-list with the definition By appealing to the symbolic definitions, EGS is able to generate examples for constraints which are very restrictive or involve novel concepts
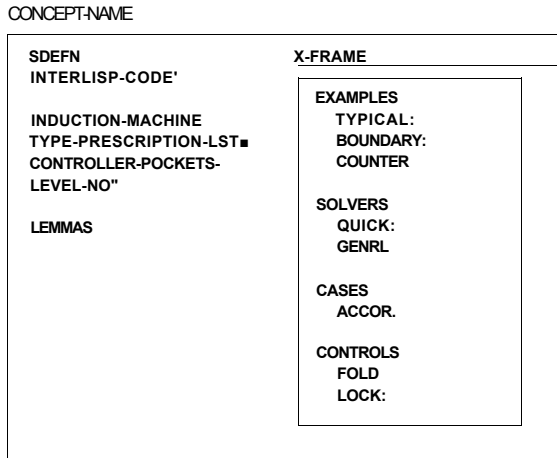
CONCEPT-NAME

```
┌─────────────────────────────────────────────────┐
│  SDEFN              X-FRAME                    │ │
│  INTERLISP-CODE'                                 │
│                     ┌──────────────────────────┐ │
│  INDUCTION-MACHINE  │ EXAMPLES                 │ │
│  TYPE-PRESCRIPTION-LST■│  TYPICAL:             │ │
│  CONTROLLER-POCKETS-│   BOUNDARY:              │ │
│  LEVEL-NO"          │   COUNTER                │ │
│                     │                          │ │
│  LEMMAS             │ SOLVERS                  │ │
│                     │   QUICK:                 │ │
│                     │   GENRL                  │ │
│                     │                          │ │
│                     │ CASES                    │ │
│                     │   ACCOR.                 │ │
│                     │                          │ │
│                     │ CONTROLS                 │ │
│                     │   FOLD                   │ │
│                     │   LOCK:                  │ │
│                     └──────────────────────────┘ │
│                                                 │
└─────────────────────────────────────────────────┘
```

Figure 2: Frame-like EGS Knowledge Structure

Lemmas

In the Boyer-Moore theorem prover lemmas are proved and stored under their ke\ function names with the user-assigned types The EGS simplifier makes use of *rewrite* type lemmas when it rewrites terms

Known Examples

Known examples are important knowledge to EGS because they play the role of building blocks for generating examples. Generated examples are often simply known examples or, at least, constituted of known examples.   Rich and well-sorted known examples substantially increase the efficiency of EGS and make the system behave more naturally. Known examples are used when a TEST task is performed.

In EGS, for a given function a known example is represented as a list of QUOTEd constants - actually stored unQUOTEd - each of which corresponds to a formal argument of the function.   Known examples for a function are classified into three types: *Typical, Boundary,* and *Counter.*   Known examples for MEMBER can be represented as.

```
(EXAMPLES
  (TYPICAL
   (A (A B C)) (4 (7 3 4 . 5)) ((C) (A B (C) (D)))
   (B ((0) A 9 B G)) ((E F) (M N (E F) 6)))
  (BOUNDARY                                        (*)
   (A (A)) (NIL (NIL NIL NIL)) (C (C . C)))
  (COUNTER
   (E E) (NIL 8) (6 (2 4 7 5))
   (B (A . B)) (D (A B C)))).
```

Currently known examples must be sorted **and given** by the **user.** This, however, has **advantages** in that **generated examples are more user-tuned.**

Solvers

A solver is user-supplied, procedurally attached, production rule-like knowledge whose action part transforms a formula, possibly a set of formulas, into a form which is simpler and easier to handle   For example, a formula: (EQUAL (REVERSE L) (QUOTE (A B C))) can be directly transformed into: (EQUAL L (REVERSE (QUOTE (A B C)))). The built-in linear solver is a special solver for simultaneous linear equations/inequalities   Solvers are specific to the patterns of formulas and often heuristic; however, they are efficient. The solver knowledge substantially increases the efficiency of EGS

5. A Simple Illustration of EGS Example Generation

Suppose the following command is typed in:

```
    FINDX((AND (MEMBER X L)
               (LESSP X Y)
               (LESSP Y (CAR L)))
          NIL) .
```

1. The preprocessor translates the input constraint into:

```
    (AND (MEMBER X L)
         (AND (LESSP X Y)
              (LESSP Y (CAR L)))) .
```

2 A START task is initiated The task agenda would be

```
    ((O START ((AND (MEMBER X L)
                    (AND (LESSP X Y)
                         (LESSP Y (CAR L)))))
              NIL NIL)) .
```

3. The START task is performed and the resulting subgoal is simplified to produce a new subgoal

```
    (((MEMBER X L) (LESSP X Y) (LESSP Y (CAR L)))
     NIL) .
```

4 The task generator is invoked with the subgoal and generates new tasks; the task agenda would be then:

```
    ((76
      TEST
      ((MEMBER X L) (LESSP X Y) (LESSP Y (CAR L)))
      NIL
      (CLUE (MEMBER X L) FLG NIL))
     (74
      TEST
      ((MEMBER X L) (LESSP X Y) (LESSP Y (CAR L)))
      NIL
      (CLUE (LESSP X Y) FLG NIL))
              .........
              .. . ...
     (33
      ANALYZE
      ((MEMBER X L) (LESSP X Y) (LESSP Y (CAR L)))
      NIL
      (CLUE (L CAR)))
     (-9000
      EXPAND
      ((AND (MEMBER X L)
            (AND (LESSP X Y)  (LESSP Y (CAR L)))))
      NIL
      NIL)) .
```

5 The top task is processed. The task performer retrieves MEMBER'S typical and boundary stored examples - let's assume MEMBER'S stored examples are given as (*) - and tests them with the hard-list i.e. evaluates each of condition formulas in the hard-list under the environment established by associating the CLUE arguments with the corresponding values in an example. Only the example (4 (7 3 4 . 5)) survives the test; in

```
       Constraint                                  Examples³

(SUBBAGP X (DELETE U Y))              ((U . 1) (X . B) (Y   NIL))
                                      ((U . C) (X . (A B)) (Y . (C C B A)))
                                      ((U . 3) (X . (2)) (Y . (2 3 5)))

(NOT (EQUAL                           ((P . 1FALSE) (Q   1TRUE) (R   1FALSE))
     (IMPLIES P (IMPLIES Q R))        ((P . 1FALSE) (Q   NIL) (R . 1FALSE))
     (IMPLIES (IMPLIES P Q) R)))      ((P . 1FALSE) (Q . 2) (R . 1FALSE))

(AND (MEMBER X Y)                     ((X   A) (Y   (A)) (Z . (A (A))))
     (MEMBER Y Z))                    ((X . A) (Y . (B A   B)) (Z . (A (B A   B) B C)))
                                      ((X . 2) (Y   (3 2)) (Z . ((3 2) A B)))

(AND (ORDERED L)                         ((L   (A B C)))
     (NOT (EQUAL (SORT L) L)))           ((L   (3 5 7 . 2)))

(AND (MEMBER 'P L)                       ((L   (P Q P)))
     (MEMBER 'Q L)                       ((L . (Q P Q)))
     (EQUAL (REVERSE L) L))              ((L   (P Q Q P)))

(AND (NUMBERP Z)                      ((B   15) (K   21) (X   5) (Z   7))
     (PRIME X)                        ((B   18) (K . 30) (X . 3) (Z   5))
     (NOT (DIVIDES X Z))              ((B   5) (K   3) (X . 5) (Z   3))
     (EQUAL (TIMES X K) (TIMES B Z)))

(AND (PRIME P) (PRIME Q)              ((A   43) (B   3) (P   5) (Q   2))
     (NOT (EQUAL P Q))                ((A . 37) (B   1) (P   2) (Q   3))
     (EQUAL (REMAINDER A P)           ((A   9) (B   3) (P . 2) (Q   3))
            (REMAINDER B P))
     (EQUAL (REMAINDER A Q)
            (REMAINDER B Q))
     (NUMBERP B)
     (LESSP B (TIMES P Q)))
```

Figure 3:   Some EGS-generated Examples

the case that none of the examples succeed, EGS would appeal to other types of tasks, for example, definition expansion The subgoal produced after simplification is:

```
(((LESSP (QUOTE 4) Y) (LESSP Y (QUOTE 7)))
 ((L   (QUOTE (7 3 4 . 5))) (X   (QUOTE 4))))
```

6. The above subgoal gives rise to several new tasks and the top task would be.

```
(82 SOLVE
    ((LESSP (QUOTE 4) Y) (LESSP Y (QUOTE 7)))
    ((L . (QUOTE (7 3 4 . 5))) (X . (QUOTE 4)))
    (MODE LINEAR
    POLY LST ((-5 ((Y . -1)) LEQ)
              (6 ((Y   1)) LEQ)))).
```

7 To perform the SOLVE task, the built-in linear solver is invoked As the result Y gets '(QUOTE 6) and further simplification yields a new subgoal:

```
(NIL
 ((Y   (QUOTE 6))
  (L . (QUOTE (7 3 4 . 5)))
  (X . (QUOTE 4))))
```

8. Only an ASSEMBLE task is generated with the subgoal and it would be the highest scored task:

```
(10000 ASSEMBLE NIL
       ((Y . (QUOTE 6))
        (L . (QUOTE (7 3 4 . 5)))
        (X . (QUOTE 4)))
       (TEMPLET (L X Y))).
```

9 The ASSEMBLE task constructs an example.

```
((L . (QUOTE (7 3 4 . 5)))
 (X   (QUOTE 4))
 (Y . (QUOTE 6)))
```

l() This example is verified by the evaluator   More examples would possibl) he generated by continuing the similar procedure of performing and generating tasks

6. Related Work and Future Research

Examples have long been used for many A I. tasks, especially for machine learning |13. 14. 25] However, very few learning system can generate examples by themselves and often examples are given by the user interactively or are stored initially in the system. Lenat's AM |13| lacks reasoning for example generation, therefore, its example generation capability is limited   AM generates examples for a concept simply by executing the algorithmic specification (a LISP-like program) of the concept with the known examples Gelernter |7 8] experimented with the idea that humans almost always draw diagrams when trying to prove geometry theorems   In his geometry machine, Gelernter used diagrams to prune the backward chaining during the proof search   The use of diagrams in this way reduced the proof search many orders of magnitude. CEG (20] is a 3-phasc structured example generation system over the domain of LISP data and programs.   In CEG, a constraint is specified in the form of a set of desired property-value pairs. CEG's expressive power and ability in problem handling are limited. Moreover, its means-ends analysis method for example modification may not be suitable for cases where moderate expressive power is required.   Ballantyne   and   Bledsoe's GRAPHER [1] generates

---

³ *Here examples* are printed without "QUOTE" For most cases the thown example have been generated well within 10 seconds of CPU time.

counter-examples for non-trivial topological conjectures. Given a conjecture, GRAPHER constructs a set-theoretic relationship graph among the set variables occurring in the conjecture The graph serves as a global constraint specification when the set variables are assigned values   REF-ARF |6| employed the constraint-satisfaction technique REF-ARF was able to solve admittedly difficult problems such as cryptoarithmetic problems and the 8-queens problem. The *regression* method for plan modification [24], proposed by Waldinger can also be useful for example generation Green's QA3 [9] and the PROLOG interpreter |15| are both driven by resolution-based theorem proving, however, they can hardly be classified as example generators in their own right

Experiments have been carried out applying EGS to the problems of controlling backward chaining in the Boyer-Moore theorem prover and semantically checking conjectures by way of finding counter-examples These experiments have shown EGS to be powerful and efficient   Howe\er, several improvements are still needed, the following only describes those projected for the near future.

- incorporating a simple learning capability - updating the stored examples in favor of better examples through evaluating and classifying *experienced* examples;

- devising a high-level control mechanism for dynamically adjusting the task score computation - for example, the task tree which represents the derivational relationship of tasks. can be used to evenly distribute examples,

- extending and elaborating the knowledge base;

- enriching the user interface facility.

Integration of EGS into the Boyer-Moore theorem prover is also being considered   This would result in a user-friendly environment for theory development where examples can play an invaluable role

7. Conclusions

In this paper, a new approach to automatic example generation and its implementation have been presented.   The transformational approach has several advantages

- It suggests a general paradigm for example generation. In this approach, example generation is viewed as transformation from a constraint specification to a representation which conforms to an example description scheme - the substitution equality for EGS. This view provides flexibility in selecting representation schemes for constraints and examples.

- The problems of representation and evaluation can relatively easily be solved.

- It enables general formal reasoning and problem-specific procedural knowledge to cooperate with each other to achieve both generality and efficiency.

- It provides modifiability and controllability of the system.

Considering the important role examples play in human intelligence and in A I as a study of computer simulation of human intelligence, the author believes that better understanding of the true nature of *examples* will lead us to greatly enrich contemporary A.I. research.

References

[1]   Ballantyne, A M and Bledsoe, W W.
      *On Gtnerating and Using Examples in Pr*oof *Discovery*
      Technical Report ATP-64, Departments of Mathematics and
         Computer Sciences, University of Texas at Austin, 1980

[2]   Boyer, R. S and Moore, .J S.
      A Computational Logic.
      Academic Press, New York, 1979

[3]   Boyer, R S and Moore. J S.
      Metafunctions: Proving Them Correct and Using Them
         Efficiently as New Proof Procedures.
      In Boyer, R. S and Moore, J S (editors), *The Correctness
         Problem in Computer Scienct,*   Academic Press, London,
         1981

[1]   Brown, .J. S. and Burton, R R
      Diagnostic Models for Procedural Bugs in Basic Mathematical
         Skill.
      Cognitive Scienct 2.155-192, 1978

|5| Bundy, Alan
      *Artificial Mathematicians. The Computer Modelling of
         Mathematical   Reasoning.*
      Academic Press, London, 1983

[fi]   Fikes,   R E
      REF-ARF A System for Solving Problems Stated as
         Procedures,
      Artificial   Intelligence 1.27-120, 1975

|7| Gelernter,   H
      Realization of a Geometry-Theorem Proving Machine
      In Feigenbaum. E. A. and Feldman, J (editors), *Computers
         and Thoughts, .* McGraw-Hill Book Company, 1963.

[8]   Gelernter, H. and Hansen, J. R. and Loveland, D W
      Empirical Explorations of the Geometry-Theorem Proving
         Machine.
      In Feigenbaum, E A and Feldman, J. (editors), *Computers
         and Thoughts, .* McGraw-Hill Book Company, 1963.

[9j   Green, Cordell
      Application of Theorem Proving to Problem Solving.
      P*roc of IJCAI-69 .*219-239, 1969.

[10] Jouannaud, Jean-Pierre et al.
      SISP/1 An Interactive System Able to Synthesize Functions
         from Examples.
      *Proc of IJCAI-*77:412-418, 1977.

[II| Kim, Myung W.
      Measure Guessing. An Experiment in Automatically
          Generating and Using Examples
      1984.
      Institute for Computing Science and Computer Applications,
          University of Texas at Austin, Working Paper.

[12] Lakatos, I.
      Proofs and Refutations.
      Cambridge University Press, 1976.

[13] Lenat. D H
      *A M An Artificial Intelligence Approach to Discovery in
          Mathematics   as Heuristic Starch*
      PhD thesis, Stanford University, 1976

[11] Mitchell, T. M.
      version Spaces An Approach to Concept learning
      PhD thesis, Stanford University, 1978

[15] Pereira, I, M. and Pereira, F C. and Warren, D II.
      User's Guide to DECsystem-10 PROLOG.
      Sept , 1978

[16] Polya, G
      How To Solve ft
      Pneeton University Press, 1954
      2nd ed

[17] Polya, G
      Mathematical Discovery
      Wiley, New York, 1962
      Vol. 1 and 2

[18] Reiter, R
      A Semantically Guided Deductive System for Automatic
          Theorem Proving
      IEEE Trans on Computers (C-25(4)) 328-334, 1976

[19] Rissland, Edwina.
      Understanding Understanding Mathematics
      Cognitive Science 2, 1978

[20] Rissland, Edwina and Soloway, Elliot
      Generating Examples in LISP. Data and Programs
      Technical Report COINS TR-80-07, Department of Computer
          and Information Science, University of Massachusetts at
          Amherst, 1980.

[21] Rissland, Edwina L. and Valcarce, Eduardo M. and Ashley,
      Kevin D
      Explaining and Arguing with Examples
      Proc. of AAAI-84 288-294, 1984.

[22] Stefik, Mark.
      Planning with Constraints
      Artificial Intelligence 16 .111-140, 1981.

[23] Summers, P. D.
      *A Methodology for LISP Program Construction from
          Examples.*
      Technical Report, IBM T.J. Watson Research Center, 1976

[24] Waldinger R. J.
      Achieving Several Goals Simutaneously.
      Machine Intelligence 8:94-136, 1977.

[25] Winston, P. H.
      Learning Structural Descriptions from Examples.
      In Winston, P. II. (editor), *The Psychology of Computer
          Vision, .* McGraw-Hill Book Company, 1975.