EXAMPLES AND LEARNING SYSTEMS*

Edwina L. Rissland

Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003

Technical Report# 83-16

To appear in Adaptive Control of Ill-Defined Systems, Selfridge, Rissland and Arbib (eds.), Plenum, 1983.

EXAMPLES AND LEARNING SYSTEMS*

Edwina L. Rissland

Department of Computer and Information Science
University of Massachusetts
Amherst, MA 01003, U.S.A.

INTRODUCTION

Any system that learns or adapts -- whether well- or ill-defined, man or machine -- must have examples, experiences, and data on which to base its learning or adaptation. Too often, however, the examples that form the basis of learning are taken for granted. This paper will concentrate on the examples as a study in their own right.

BACKGROUND

The importance of examples to learning systems can be seen in many well-known A.I. programs. For instance, Winston's program [1975] learns the concept of "arch" from a sequence of examples: of arches and non-arches. The initial example, and examples that fail to be an arch in just one aspect, "near misses", are what drive this learning system. Samuel's Checker Player is another classic AI program that makes use of examples [1963, 1967]. Samuel gave his program libraries of specific book moves. His work is an example of the interplay between examples, adaptation and learning systems.

Selfridge's recent COUNT and sorting programs [Selfridge 1979, 1980], as well as his classic PANDEMONIUM [Selfridge 1958], also depend heavily on example data and problems. COUNT would not learn

----

*The research reported in this paper was supported in part by the National Science Foundation under grant no. IST-8017343.

its task -- how to count the number of symbols in a string -- if not presented with a sequence of challenging but not too difficult problems to solve. The sorting program would not be able to adaptively tune itself -- learn the "best" choice of sorting method to use-- if it did not have abundant experience with problems of sorting lists.

These same observations can also be made about Soloway's BASEBALL program [Soloway 1978], which learns the concepts and rules of baseball by "watching" baseball games; and about Lenat's AM program [Lenat 1977], which discovers new mathematical concepts partly on the basis of how the concepts and control heuristics fare on examples.

Examples also play a critical role in non-computer systems like law, mathematics and linguistics. For instance, the law -- especially common law, which is based on the doctrine of precedent ("stare decisis") -- is a wonderful example of a system which is hard to define (despite its superficial resemblance to a rule-based system) and which runs entirely in response to the examples -- i.e., cases -- presented to it. Cases must be adjudicated; the court expresses the results as "holdings" and "dicta" (rules and comments of the case); these lead to the further evolution of concepts and rules which are then modified by further cases [Levi 1949, Berman 1968]. In legal scholarship and education, the cases may be hypothetical; in some law classes, "hypos" become as "real" and have as much import as actually litigated cases.

Examples are central to reasoning in mathematics in the cycle of "proofs and refutations" [Lakatos 1976]: a mathematical concept is refined in response to how the system, that is the mathematical theory with all of its definitions, theorems, proofs, examples, etc., fares in the face of examples, ranging from standard examples to unusual "monsters". Of course, a rich set of examples is needed for the guessing and inductive inference needed to get the system started in the first place [Polya 1968].

In linguistics, examples too are central to theory formation and in fact, many theories are built in response to what have now become famous examples, or counter-examples, to other theories. (See for instance, Gazdar's [1981, 1982] response to Chomsky's assertion that English does not have phrase-structure grammar.) In fact, Kuhn has pointed out that this interplay between an evolving system and examples is ubiquitous in the history of science [Kuhn 1970].

Finally, in computer programming itself there is an "inevitable intertwining" [Swartout and Balzer 1982] of the evolving system (e.g., a program), as expressed in code and specifications, and the examples used to test it out. One writes code, tests it out on example data, and then (usually) revises the program; specifications also evolve in this way. The programming and debugging process is thus completely analogous to the cycle of proofs and refutations in mathematics.

Thus, in summary, examples (by which is meant experiences, data, instances) are critical grist for the mill of learning and adaptation. Once this obvious, but key, point is recognized, one is led immediately to questions like the following:

1. How should examples be grouped into types?
2. Do examples have structure?
3. How are examples related to one another?
4. What properties should an example satisfy?
5. Where do the desired properties come from?
6. How can knowledge of examples be applied to ill-defined systems?

The rest of this paper addresses these questions.

TAXONOMIES OF EXAMPLES

When one considers the different effects and uses examples can have with respect to learning systems, one can distinguish different classes of examples. In this section, we briefly review the "epistemological" classes of examples that we have found useful in disciplines like mathematics and law [Rissland 1978, 1982].

It is important to recognize that not all examples serve the same function in learning. For instance, expert teachers and learners know that certain perspicuous ("start-up") examples provide easy access to a new topic, that some ("reference") examples are quite standard and make good illustrations, and that some examples are anomalous and don't seem to fit into one's understanding. We can develop a taxonomy of items based upon how we use them to learn, understand and teach:

(a) start-up examples: perspicuous, easily understood and easily presented cases;
(b) reference examples: standard, ubiquitous cases;
(c) counter examples: limiting, falsifying cases;
(d) model examples: general, paradigmatic cases;
(e) anomalous examples: exceptions and pathological cases.

Start-up examples are simple, easy to understand and explain cases. They are particularly useful when one is learning or explaining a domain for the first time. Such examples can be generated with minimal reference to other examples; thus one can say they are structurally uncomplicated. A start-up example is often "projective" in the sense that it is indicative of the general case and that what one learns about it can be "lifted" to more complex examples.

Reference examples are examples that one refers to over and over again. They are "textbook cases" which are widely applicable throughout a domain and thus provide a common point of reference through which many concepts, results and other items in the domain are (indirectly) linked together.

Counter-examples are examples that refute or limit. They are typically used to sharpen distinctions between concepts and to refine theorems or conjectures. They are essential to the process of "proofs and refutations" [Lakatos 1976].

Model examples are examples that are paradigmatic and generic. They suggest and summarize expectations and default assumptions about the general case. Thus, they are like "templates" or "frames" [Minsky 1975].

Anomalous examples are examples that do not seem to fit into one's knowledge of the domain, and yet they seem important. They are "funny" cases that nag at one's understanding. Sometimes resolving where they fit leads to a new level of understanding.

An example of applying this classification scheme for an introductory study of continuity from the domain of real function theory might classify: the function f(x)=x as a start-up example; f(x)=x**2, f(x)=e**x as reference examples; f(x)=1/x as a counter-example; "f(x) with no gaps or breaks" as a model example; and f(x)= sin(1/x) as an anomalous example. The first example, f(x)=x, is also a reference example (the "identity" function). Thus, such a classification need not be exclusive. The anomaly sin(1/x) will most likely become a favorite counter-example as one understands that a function can fail to be continuous in at least two ways, that is, by having gaps and breaks and by failing to settle down to a limit. Thus, such a classification is not static. Increased understanding will of course lead to qualifications on the above model of a continuous function, although it will still serve to summarize one's expectations.

In introductory LISP programming one deals with lists of atoms. For the novice, the lists (A) and (A B C) are start-up examples; (A B C) is also a reference example; NIL or ( ), a counter-example; "left-paren atom atom atom ... right-paren", a model example.

The point here is not the particular parsing out of examples into classes, for this depends on many considerations such as one's level of expertise and purposes for taxonomizing; but that there are classes of examples, and that such epistemological knowledge about classifying examples (and other items) is an important part of one's understanding. Such knowledge can be used to help focus one's attention in learning and explaining, for instance by suggesting heuristics like "Check out the conjecture on reference examples before believing too strongly in it" or "Look for counter-examples in the class of known counter-examples".

STRUCTURAL ASPECTS OF EXAMPLES

In a complex domain like mathematics or law, there are several types of structure. We can distinguish items, relations, spaces:

(1) items are strongly bound clusters of information: for instance, the statement of a theorem, its name, its proof, a diagram, an evaluation of its importance, and remarks on its limitations and generality.

(2) relations between items: for instance, the logical connections between results, such as predecessor results on which a result depends logically and successor results which depend on it.

(3) spaces are sets of similar types of items related in similar ways: for instance, proved results and their logical dependencies. Such a set of items and relations constitute a space, in the case of results, a Results-space.

In essence the idea is that examples (and other items) are cohesive clusters of information which do not exist in isolation from one another; there are relations between them. What distinguishes a "space" from a "set" is the prominence of the relations. The structure of a complex domain like mathematics contains not just one but many spaces, each of which describes a different aspect of knowledge. Examples are but one type of "item" that comprise the knowledge in such domains; others include concepts, results, strategies and goals [Rissland 1978, 1981b]. Of concern in this paper are examples by which we mean specific situations, data or experiences.

An example has many aspects or pieces of information that comprise it: its name, taggings and annotations as to epistemological class and importance, lists of pointers to other examples from which it is constructed and to whose construction it contributes, the process of how it is constructed, a schematic or diagram, pointers to items like definitions in other spaces, statements of what the example is good for or how it can be misleading, sources of further information about the example.

Examples can be related by constructional derivation of how one example is built from others. Examples plus this relation constitute an Examples-space. For instance, in the LISP programming example, the examples ((A) B C) and (A (B C)) can be thought of as constructionally derived from the reference example list (A B C) by the addition of parentheses.

The construction of a new example from others is a process that is found in many fields, for instance law and mathematics. In teaching the law, one frequently makes use of "hypothetical" examples ("hypos") which are often constructed by modification of a well-known reference example, e.g., a textbook case. In a Socratic discussion between a law professor and his class, the teacher may spin out a sequence of hypos to test out the class's understanding and biases on a doctrinal proposition. Such a sequence might contain increasingly more complex or extreme cases.

The following hypothetical examples are taken from a class discussion in contract law. They were used to point out the difference between the doctrines of "consideration" (which emphasizes what the promisee gives the promisor in return for the promise) and "reliance" (which emphasizes how the promisee acts in reliance on the promise). These are two different ways to approach the problem of determining which contracts are legally enforceable as opposed to which are gratuitous gifts. The base case from which the hypos are constructed through modifications is Dougherty v. Salt, a standard case in a course in contract law [Fuller and Eisenberg 1981].

Hypo1:
Facts: Aunt Tillie says, "Charlie, you are such a nice boy; I promise to give you $10,000."

Hypo2:
Facts: Same as Hypo1 with the addition that Charlie says, "Dear Aunt Tillie, I can't take something for nothing, let me give you my third-grade painting."

Hypo3:
Facts: Same as Hypo2 except that Charlie offers to mow Tillie's lawn.

Hypo4:
Facts: Same as Hypo2 except that Charlie's last name is Picasso.

Hypo5:
Facts: Same as Hypo1 with the addition that Aunt Tillie's assets

are in ruin and that keeping her promise to Nephew Charlie means her own children starve.

Hypo6:
Facts: Same as Hypo1 with the addition that Charlie then makes an unreturnable deposit on a new car.

Thus in summary there are internal and external structural aspects to examples. The internal structure of an example concerns the cluster of strongly bound information that comprises the example, including pointers to other items like examples. The external aspects concern the relations among examples, for instance how one example is constructed from others.

CONSTRAINTS AND EXAMPLES

An important aspect of examples with respect to learning systems is the obvious fact that examples possess certain properties. For instance, the "near misses" in Winston's work are examples that fail to be arches in exactly one of the required properties of archness. What perhaps may not be so obvious is that in selecting examples to give to a learning system, one does not pick them at random: examples are generated for a purpose -- like giving evidence for or against a conjecture -- and thus examples are usually (carefully) chosen to possess certain desired properties, which we call constraints.

We have called this process of generating examples that meet prescribed constraints "Constrained Example Generation" or "CEG". In past work, we have described, built, and experimented with a model of the CEG process. See for instance, [Rissland 1980, 1981a, Rissland and Soloway 1980a, 1980b]. It is based upon observations of humans working problems in which they are asked to generate examples satisfying certain constraints. Our model of CEG incorporates three major phases: RETRIEVAL, MODIFICATION, and CONSTRUCTION.

When an example is sought, one can search through one's storehouse of examples for one that matches the properties desired. If one is found, the example generation problem has been solved through RETRIEVAL. In retrieval, there are many semantic and contextual factors -- like the last generated example -- and therefore one is not merely plunging one's hand into an unorganized knowledge base. Thus even though retrieval sounds simple, it can be very complex.

However, when a match is not found, how does one proceed? In many cases, one tries to MODIFY an existing example that is judged to be close to the desired example, or to have the potential for being modified to meet the constraints. Often the order of examples selected for modification is based on judgements of closeness between properties of known examples and the desiderata, that is, how "near" the examples are to what is sought.

If attempts at generation through modification fail, experienced example generators, like teachers or researchers, do not give up; rather they switch to another mode of example generation, which we call CONSTRUCTION. Under construction, we include processes such as combining two simple examples to form a more complex one and instantiation of general model examples or templates to create an instance. Construction is usually more difficult than either retrieval or modification.

General Skeleton of the CEG Model

CEG has subprocesses for: Retrieval, Modification, Construction, Judgement, Control

Presented with a task of generating an example that meets specified constraints, one:

1. SEARCHES for and (possibly) RETRIEVES examples JUDGED to satisfy the constraints from an EXAMPLES KNOWLEDGE BASE (EKB); or

2. MODIFIES existing examples JUDGED to be close to, or having the potential for, fulfilling the constraints with domain-specific MODIFICATION OPERATORS; or

3. CONSTRUCTS an example from domain-specific knowledge, such as definitions, general model examples, principles and more elementary examples.

In examining human protocols, one sees two types of generation: (1) retrieval plus modification; and (2) construction. That is, one does not necessarily try first retrieval, then modification, then construction; sometimes construction is attempted straightaway. Clearly, this model needs many other features to describe the CEG process in its entirety; more details can be found in [Rissland 1981a].

To give the reader an idea of the richness and complexity of the CEG process, we present here a synopsis of a CEG problem taken from the domain of elementary function theory. The problem is:

> _Give an example of a continuous, non-negative function, defined on all the real numbers such that it has the value 1000 at the point x=1 and that the area under its curve is less than 1/1000._

Most protocols for this question began with the subject selecting a function (usually, a familiar reference example function) and then modifying it to bring in into agreement with the specifications of the problem. There were several clusters of responses according to the initial function selected and the stream of the modifications pursued. A typical protocol went as follows [Rissland 1980]:

> "Start with the function for a "normal distribution". Move it to the right so that it is centered over x=1. Now make it "skinny" by squeezing in the sides and stretching the top so that it hits the point (1, 1000)."
>
> "I can make the area as small as I please by squeezing in the sides and feathering off the sides. But to demonstrate that the area is indeed less than 1/1000, I'll have to do an integration, which is going to be a bother."
>
> "Hmmm. My candidate function is smoother than it need be: the problem asked only for continuity and not differentiability. So let me relax my example to be a "hat" function because I know how to find the areas of triangles. That is, make my function be a function with apex at (1, 1000) and with steeply sloping sides down to the x-axis a little bit on either side of of x=1, and 0 outside to the right and left. (This is OK, because you only asked for non-negative.) Again by squeezing, I can make the area under the function (i.e., the triangle's area) be as small as I please, and I'm done."

Notice the important use of such modification operations as "squeezing", "stretching" and "feathering", which are usually not included in the mathematical kit-bag since they lack formality, and descriptors such as "hat" and "apex". All subjects made heavy use of curve sketches and diagrams, and some used their hands to "kinesthetically" describe their functions. Thus the representations and techniques used are very rich.

Another thing observed in _all_ the protocols (of which there were about two dozen for this problem) is that subjects make implicit assumptions -- i.e., impose additional constraints -- about the symmetry of the function (i.e., about the line x=1) and its maximum (i.e., occurring at x=1 and being equal to 1000). There are no specifications about either of these properties in the problem statement. These are the sort of tacit assumptions that Lakatos [1976] talks about; teasing them out is important to studying both mathematics and cognition.

CONSTRAINT GENERATION

Generating examples from constraints presupposes that there are constraints. That is, that properties of examples can be expressed in a language of constraints, and that one actually knows what one wants in the example sought, that is, the constraints. To put it another way, there is a prior problem of <u>constraint generation</u>.

The constraints are often generated from consideration of one's intended use for the example-to-be. For instance, if one were testing a program known to work on simple cases, one would want examples that are more complex or rare, for instance an anomalous or counter-example. If one could not find a satisfying example in one's "Examples Knowledge Base", perhaps organized as an Examples-space, one would need to generate it. To do this one could express the desiderata for the example in terms of constraints and then proceed with the CEG process.

In hypos for a Socratic discussion in law, such constraints would include pedagogical, rhetorical, doctrinal constraints, such as, those in our previous example, arising from the doctrine of consideration. The constraint upon the object given by Nephew Charlie to his Aunt Tille might be loosely described as "being something of value"; this constraint is then varied from something of little value (the typical third-grade painting), to something of some value (mowing the lawn), to something of great value (a "Picasso" third-grade painting).

If one were giving examples of lists to a neophyte LISP programmer, one would make use of domain-specific constraints having to do with "length" (of the list), "order" (of atoms in the list), "depth" (of certain atoms), and whether the list is a "LAT" (i.e., a list of atoms) or a more complex list [Rissland and Soloway 1980b]. For instance, one might want a list such that: (1) it has length 3, and (2) the depth of the first atom is 3. The list (((A)) B C) satisfies both constraints and may be thought of as generated from the reference example (A B C) by a sequence of modifications affecting depth. The lists (((A) B) C) and (((A B C))), which are also generated by modifications affecting depth through the addition of parentheses, satisfy the second (depth) but not the first (length) constraint. Thus, modifications designed to remedy one constraint deficiency might, in the language of Sussman [1973], "clobber a brother goal", that is, another constraint. Such interactions can make the CEG problem quite complex.

APPLICATIONS TO ILL-DEFINED SYSTEMS

In this section we suggest how examples can be used in probing, debugging, specifying, or otherwise dealing with an ill-defined or not well-understood system.

Examples can be useful for probing a system. Consider the following scenario. We've just logged on and are trying to learn how to use or test a system, or a new feature of it. Suppose it were a program to sort letters into alphabetical order.

To probe this program, we might start off by saying, "If this program really works, it ought to do simple little things; it certainly ought to handle A, B, C. If it fails on A, B, C, we know there's a problem from the very beginning." Suppose we find it's OK on A, B, C, a standard start-up or reference example in this mini-domain.

Now we'd try something a bit more complex, like a longer list, say more of the beginning of the alphabet. This is a slight embellishment of the beginning example, arising from a length constraint. Suppose it works on such an example -- which is good since the program didn't have to do "anything" -- let's give the program an opportunity to exercise itself on some other simple cases like C, B, A or the alphabet in reverse order. Then, we might introduce a couple of letter interchanges like A, C, B or M, O, N, P, R, Q. These involve order as well as length constraints and can be generated with an "interchange" modification.

Thus we're probing the system with a sequence of increasingly complex cases derived from standard simple ones. With a minimal level of confidence in the program established, we could go on to test it on more difficult or limiting cases.

We might now check if the program can handle known troublesome examples like a singleton list, like "A", or the empty list, which are known from experience to be some of the cases that make programs cough and sputter, that is, counter-examples. The singleton list often causes problems because of the false assumption that there's (always) going to be more than one element. A LISP programmer would know of another well-known trouble maker: the empty list, NIL. It is an important case in recursive procedures. In fact, it is a favorite (i.e., reference) example with any LISP hacker. In the sorting domain, there would be some other specific things that are known to cause problems, for instance, repeated elements. Thus we are using knowledge about the examples and about the context and task in evaluating a system.

Note that we are not examining or altering the code, which can be considered a lower level representation of the program; rather we're staying on a representational level more akin to the purposes of the program, that is, what the program is supposed to do, or more realistically, what we believe it's supposed to do. We're dealing with the system at arm's length and just probing and asking if the program appears to work by our experience with it on selected, and well-chosen, examples.

Such probing of a system in this manner is similar to testing theorems. (Again, we have the analogy with mathematics, which has been discussed by De Millo et al. [1980].) One can never show that it works by just trying examples, we can only show, perhaps, that it doesn't work, just as one counter-example can refute a conjecture. However, having a variety of instances that work establishes confidence, even if it doesn't end the verification process. By selecting enough well-chosen cases, one can "span" the possibilities and obtain a sense of the program's correctness. One of the differences between a novice and an expert programmer (or mathematician) seems to be the richness of the "spanning" examples used: novices tend to forget to check the program on complicated or known counter-examples and anomalies, even simple ones like NIL. Thus part of the art of expert programming is epistemological knowledge of the type we have described in previous sections of the paper.

Also note that in this scenario the problem of evaluating the answer required little work on our part: alphabetical order is obvious by inspection. In other cases of generating test data, there's a lot more work involved to being a critic. (See [Dietterich and Buchanan] in this volume.)

These remarks also apply if we are writing or debugging a program. Using specific examples to work out a solution helps one to deal with the complexity or lack of specification of the solution. It might well be the case that the kind of examples used in probing might be different from those in design and implementation.

Since it is impossible to completely specify a program under every condition -- because, for instance, the context of the system is changing or one is not sure of what one wants -- using examples to show what the program should do in certain situations, especially those that matter to the specifier or that are too difficult to describe in symbols or words -- provides another means of describing the program. Together traditional specifications in words, logic or symbols joined with example cases provides a better specification than either alone; each mode compensates for and complements the other.

CONCLUSIONS

In this paper we have discussed examples in their relation to learning systems and in their own right. In particular, we have described the structural aspects of examples, including their internal structure as a strongly bound cluster of information and their external relations to other examples through construction. We have used constraints to approach the problem of generating examples with specific properties and have described the prior problem of constraint generation which involves interactions between the system and the examples or example user. Lastly, we have suggested that examples are central to probing, debugging and even specifying systems: they can probe a system and they can help describe it by showing _how_ it does or should operate.

Thus, in our approach we concentrate on the role that examples (that is, experiences, instances, data) play in systems, well- or ill-defined, and find that they are a rich study in themselves. Not only are they interesting, in fact they are central. As that great polymath Oliver Wendell Holmes put it:

"The life of the law has not been logic; it has been experience"

And it is experience we are capturing with examples.

REFERENCES

Berman, H. J., "Legal Reasoning". In _International Encyclopedia of the Social Sciences_, MacMillan, 1968.

De Millo, R.A., Lipton, R.J., Perlis, A.J., "Social Processes and Proofs of Theorems and Programs", _Communications of the ACM_, Vol. 22, No. 5, May 1979. Reprinted in the _Mathematical Intelligencer_, Vol. 3, No. 1, 1980.

Fuller, L. L., and M. A. Eisenberg, _Basic Contract Law_. West Publishing Co., Minn., 1981.

Gazdar, G., "Phrase Structure Grammar". In Jacobson, P., and G.K. Pullun (Eds.) _The Nature of Syntactic Representation_, Reidel, 1982, pp. 131-187.

Gazdar, G., "Unbounded Dependencies annd Coordinate Structure". _Linguistic Inquiry 12_ (1981), 155-184.

Kuhn, T. S., The Structure of Scientific Revolutions. Second Edition. University of Chicago Press, 1970.

Lakatos, I., Proofs and Refutations. Cambridge University Press, London, 1976. Also in British Journal for the Philosophy of Science, Vol. 19, No. 3, May 1963.

Lenat, D. B., "Automatic Theory Formation in Mathematics", Proceedings of Fifth International Joint Conference on Artificial Inteligence. Cambridge, MA., 1977.

Levi, E. H., An Introduction to Legal Reasoning. University of Chicago Press, 1949.

Minsky, M. L., "A Framework for Representing Knowledge". In The Pyschology of Computer Vision, Winston (ed), McGraw-Hill, 1975.

Polya, G., Mathematics and Plausible Reasoning, Volumes I and II. Princeton University Press, 1968.

Restatement, Second, Contracts. American Legal Institute, Philadelphia, 1981.

Rissland, E. L.,"Examples in the Legal Domain: Hypotheticals in Contract Law". In Proceedings Fourth Annual Conference of the Cognitive Science Society. Ann Arbor, August 1982.

_____, Constrained Example Generation. COINS Technical Report 81-24, University of Massachusetts, 1981a.

_____, The Structure of Knowledge in Complex Domains. COINS Technical Report 80-07, University of Massachusetts, 1981b. Also to appear in Proceedings NIE-LRDC Conference on Thinking and Learning Skills, Lawrence Erlbaum Associates.

_____, "Example Generation". In Proceedings Third National Conference of the Canadian Society for Computational Studies of Intelligence, Victoria, B. C., May 1980.

_____, The Structure of Mathematical Knowledge. A.I. Tech Report 472, A. I. Laboratory, Massachusetts Institute of Technology, 1978.

_____, "Understanding Understanding Mathematics". Cognitive Science, Vol. 2, No. 4, 1978.

Rissland, E. L., and E. M. Soloway, "Overview of an Example Generation System". In Proc. First National Conference on Artificial Intelligence. Stanford, August 1980a.

_____, "Generating Examples in LISP". _Proceedings International Workshop on Program Construction_. Bonas, France, 1980b.

Samuel, A. L., "Some Studies in Machine Learning Using the Game of Checkers". In _Computers and Thought_, Feigenbaum and Feldman (Eds.), McGraw-Hill, 1963.

_____, "Some Studies in Machine Learning Using the Game of Checkers. II--Recent Progress". In _IBM J. Research and Development_, 11:601-617, 1967.

Selfridge, O. G., "Pandemonium -- A Paradigm for Learning". In _Symposium on the Mechanization of Thought Processes_. Teddington, England, 1958.

_____, "An Adaptive Sorting Algorithm". In _Proceedings Third National Conference of the Canadian Society for Computational Studies of Intelligence_, Victoria, B.C., May 1980.

_____, "COUNT -- How a Computer Might Do It". Internal Report, Bolt Beranek and Newman Inc, 1979.

Soloway, E. M., "Learning = Interpretation + Generalization: A Case Study in Knowledge-Directed Learning". COINS Technical Report 78-13, University of Massachusetts, 1978.

Sussman, G. K., _A Computational Model of Skill Acquisition_. A.I. Tech Report 297, A. I. Laboratory, Massachusetts Institute of Technology, 1973. (Doctoral dissertation.)

Swartout, W. and R. Balzer, "On the Inevitable Intertwining of Specification and Programming". CACM Vol. 25, No. 7, July 1982.

Winston, P. H., "Learning Structural Descriptions from Examples" in _The Psychology of Computer Vision_, Winston (ed), McGraw-Hill, 1975.