Learning by Failing to Explain[1]

Robert J. Hall
Artificial Intelligence Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139

Abstract

Explanation-based Generalization depends on having an explanation on which to base generalization. Thus, a system with an incomplete or intractable explanatory mechanism will not be able to generalize some examples. It is not necessary, in those cases, to give up and resort to purely empirical generalization methods, because the system may already know almost everything it needs to explain the precedent. *Learning by Failing to Explain* is a method which exploits current knowledge to prune complex precedents and rules, isolating their mysterious parts. This paper describes two techniques for Learning by Failing to Explain: *Precedent Analysis*, partial analysis of a precedent or rule to isolate the mysterious new technique(s) it embodies; and *Rule Re-analysis*, re-analyzing old rules in terms of new rules to obtain a more general set.

1 Introduction

A primary motivation for studying learning from precedents is the intuition that it is easier for a domain expert to present a set of illustrative examples than it would be to come up with a useful set of rules. Explanation-based Generalization ((Mitchell, *et al*, 85)[8], (DeJong and Mooney, 86) [2]) is a powerful method for using knowledge to constrain generalization[2]. It can be defined as generalizing an explanation of why something is an example of a concept, in order to find a weaker precondition for which the explanation of concept membership still holds. (This weaker precondition describes the generalization of the example.)

The motivation for this work is that explanation is hard and often impossible (*e.g.* theorem proving). On the other hand, the efficient student should, as much as possible, *know what he doesn't know*. For example, "I don't understand step 5," is a much more productive query than "Huh?"

There are at least two reasons why an explainer can fail: the theory is incomplete, so that there is no explanation; or the explainer simply can't find the explanation, even though it exists. The latter case is not just mathematical nitpicking: the complexity of VLSI circuits and the rich set of optimizations possible creates large problems for any circuit-understander.

On the other hand, it is seldom the case that a learner knows absolutely *nothing* about an example it fails to explain; frequently, a small mysterious thing comes embedded in a large, mostly well-understood example. For instance, consider a multiplier circuit where the only difference between its design and a known one is in the way one particular XOR gate is implemented. It would be a shame to retain the complexity of the entire multiplier when the only new structural information was in one small subdevice. Rather than just reverting to completely empirical techniques when the explainer fails, it would be better to use current knowledge to throw away the portions of the example which are understood. This is what I call *Learning by Failing to Explain*. It is a complementary notion to explanation-based learning: the former operates precisely when the latter fails, and when the latter succeeds there is no reason to try the former.

Learning by Failing to Explain could be used as a filter in a learning system which combines both explanation- and empirically-based learning methods. That is, when explanation fails, use Learning by Failing to Explain to isolate a much simpler example for the generalizer. This work does not use it this way: the additional generalization issues are beyond its scope. The current system simply formulates rules without further generalization. It is not intended that this method is complete in itself with respect to learning about design.

This work is not intended as a study of human learning. It is motivated by intuitions about human learning, but no claim is made that the techniques described here reflect human learning.

There are two techniques which comprise Learning by Failing to Explain: in the first, the learner analyzes the given precedent as much as possible, then extracts the mysterious part as a new rule (or pair of rules). I call this *Precedent Analysis*. In the second, the learner uses new rules to re-analyze old rules. That is, Precedent Analysis needn't be applied only to precedents; there are cases where it is beneficial to have another look at rules found previously. This is called *Rule Re-analysis*. The system and the latest experiments with it are documented in (Hall, 86)[5]. (Hall, 85) [6] has more detail with regard to the design competences, but documents an earlier version of the system.

2 Domain and Representation

The current system learns rules of the form "structure X implements functional block Y," where by functional block I mean something like "PLUS," which represents a constraint between its inputs and outputs. By structure, I mean an interconnection of functional blocks, where the interconnection represents data flow. As indicated, the illustration domain of the system is digital circuit design. However, the algorithms should apply with minor modifications to other domains, such as program design, which are representable in a generalized data flow format. In fact the system has been applied successfully to learning structural implementation rules in a simplified gear domain, where functional constraints take the form of simple arithmetic relationships among shaft speeds.

[1]This paper is based upon work supported under a National Science Foundation Graduate Fellowship.

[2](Mahadevan, 85)[7] and (Ellman, 85)[3] have applied this to logic design. (Smith, *et al*, 85)[10], have applied explanation-based techniques to knowledge base refinement. (Mooney and DeJong, 85)[9] have applied it to learning schemata for natural language processing. (Winston, *et al*, 83)[12] abstracts analogy based explanations to form rules.

It should be noted that, while the implemented algorithms are dependent on the *functional semantics* of the domains, the basic idea behind Learning by Failing to Explain should be applicable to many other domains. This is one area for future work.

The representation for design knowledge is a *Design Grammar*. This is a collection of rules of the form LHS $\Longrightarrow$ RHS, where LHS denotes a single functional block and RHS denotes a description of an implementation for LHS. The basic representational unit is a graph, encoding the functional interconnection of the blocks. Design Grammars are not, in general, context free grammars, as the system is allowed to run the rules from right to left as well as left to right. This allows us to understand optimization of designs as a reverse rule use, followed by a forward rule use: find a subgraph of the current design stage which is isomorphic to the RHS of a rule and replace it with the LHS symbol. Then expand that new functional block instance with a different implementation.

A Design Grammar is an interesting representation of structural design knowledge both because it is learnable from examples via the method described here, and because it enables four interesting design competences:

- *Top-Down Design*: the ability to take a relatively high level specification of the function of a device and refine it successively by choosing implementations of subfunctions, then refining the refinement, and so on. In terms of a Design Grammar, this is viewed as using rules in the forward direction.
- *Optimization*: the ability to take one device and replace a piece of it with some other piece so that the resulting device is functionally the same. In Design Grammar terms, an optimization step is viewed as a reverse rule use, followed by a forward rule use.
- *Analysis*: the problem of establishing a justification for why some device performs some given function. This is the parsing problem for Design Grammars. (Winston, *et al*, 83)[12] takes a similar approach, but extracts rules "on the fly" from analogous precedents.
- *Analogical Design*: the ability to solve a new problem in a way similar to some already solved problem, or by combining elements of the solutions to many old problems. In a Design Grammar setting, this is "running" known design derivations on new design problems. This is accomplished by finding a partial match between the problem specification and the initial specification of the known derivation, applying those transformations which have an analog in the problem, and leaving out steps which do not. This technique for controlling search has been explored by (Mitchell and Steinberg, 84)[11], and even as early as the MACROPS in STRIPS(Fikes, Hart, and Nilsson, 72)[4].

A Design Grammar certainly does not represent all there is to know about design. It is intended that it serve as the backbone of structural knowledge in a larger system of knowledge which includes such things as search control heuristics and analytic knowledge.

3 Learning Rules Using Precedent Analysis

Precedent Analysis is where the learner uses its current knowledge to partially explain an example, so that the truly mysterious aspects of the example are brought to light.

The algorithm has two steps: first construct a maximal partial parse of the precedent, then throw away the matched nodes (the parsed sections), leaving two functionally equivalent subgraphs which can be turned into two rules with the same LHS.

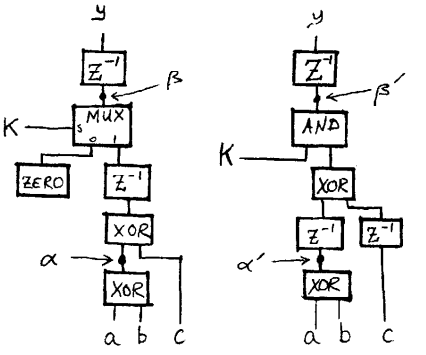

Figure 1: Learning Example Precedent.

3.1 An Example

Suppose the learner is given the precedent shown in Figure 1. (The Z^{-1} boxes represent time delay of one clock cycle.) The left-hand graph is considered to be the high level graph. Suppose further that the learner knows

- that a one-bit multiplexer (MUX) is implemented by y = (OR (AND a (NOT s)) (AND b s)),
- that (AND a (ZERO)) is an implementation of ZERO,
- that (OR x (ZERO)) is an implementation of BUFFER, and
- that a BUFFER may be implemented simply by a connection point.

These would all be represented as Design Grammar rules in a straightforward manner.

By applying these rules to the left hand graph, in the order given, the system concludes that the MUX tied to ZERO on the left actually corresponds to the AND on the right. Moreover, because of their positions relative to inputs and outputs, the Z^{-1} attached to y and the XOR attached to a and b can be seen to correspond to the ones on the left. With this motivation, the system transforms the high-level graph into the one on the left of Figure 2.

It is possible to construct a partial matching between the left- and right-hand graphs of that Figure which matches all nodes but the ones circled in dashes. It is the equivalence of those two portions which is truly mysterious to the system. The rest is derivable. Therefore, the system supposes that the equivalence is true and creates new rules. Since both subgraphs are more than single nodes, neither can be the LHS of a rule. Thus, the system creates a new functional block type and makes it the LHS of two rules: one whose RHS is the subgraph on the right, the other whose RHS is the subgraph on the left. The variable correspondences are determined by the partial match.

3.2 The Method

The current system implements a hill climbing approach to finding maximal partial matches. The algorithm is greedy. It searches for a partial derivation, which, when applied to the high-level graph, results in a graph with as large as possible a

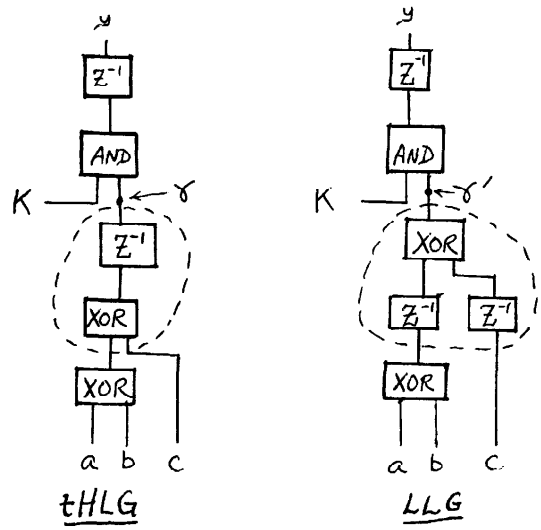

Figure 2: Learning Example Precedent After Partial Understanding (Transformation).

partial match with the low-level graph. The partial match cannot, however, be just any partial match. It must be one which associates functionally equivalent nodes in the two graphs. The heuristic criteria for extending the partial match are given below. The partial match is initialized to the input and output variable correspondences given with the precedent.

Starting from the current high level graph, the typical step looks ahead through all possible combinations of allowable transformations to a fixed search depth. (This depth is a parameter to the system. It's value affects both success and speed of the algorithm.) It tries to find some combination which results in a valid extension to the partial match. If it ever finds one, all lookahead is terminated, the transformation is made, and the lookahead starts over anew. The process terminates when no progress is made on any lookahead path.

Because the subgraph isomorphism problem is NP-complete, it is desirable to have grammar rules with graphs as small as possible. Therefore, when no progress is found, the smallest graph encountered in the last lookahead phase is returned.

The overall criterion for extending the partial match is that the matched subgraphs must always be functionally equivalent with respect to the overall function. This is implemented heuristically as follows. The partial match creeps inward from the input variables and the output variables. There are two principle ways the match can be extended. One is from the input "edge" of the match, the other is from the output "edge" of the match. Two connection points (one from each graph) which are driven by the same function of corresponding matched nodes may be matched (this moves in from the inputs). On the other hand, consider the case of two nodes (one in each graph) which are inputs to the same type of functional block. If the blocks drive matched nodes, then they can be matched, as long as there is no ambiguity in matching their inputs. (There is ambiguity if the function has more than one input of the same type as the two inputs, and at least two of those inputs remain unmatched.) This second way of extending the partial match moves in from the output edge.

There are some subtleties: consider what happens to the first criterion when *one graph* has two nodes which are (syntactically) the same function of the same inputs. This is an ambiguity which is not addressed by the criterion. However it is possible, as a pre-pass, to transform the graph to the equivalent one which has all of those functionally identical subgraphs merged into a single one. Since the graphs are assumed to have functional semantics, this move maintains functional equivalence. This pre-pass is computationally simple.

To illustrate these criteria, consider the pair of graphs in Figure 1. The first criterion would say that since connection points α and α' are each XOR of corresponding previously matched nodes, they may be matched. The second criterion would say that since β and β' are each the unambiguous inputs of the Z^{-1} boxes which drive y they may be matched. After the transformations resulting in Figure 2, the match may be further extended using the second criterion to include the connection points γ and γ'.

There is another criterion for extending the partial match when a subgraph transforms to a single connection point. (That would happen when that subgraph was functionally equivalent to the identity function.) In that case, the previous two criteria don't apply, as there are no new nodes to which to extend the match. In that case, the system looks for a situation in which the size of the inverse image of a connection point under the partial match decreases. For example, the system judges progress after transformation of "y =BUFFER(a)" to "a."

Once the partial match is extended as much as possible, it is straight-forward to construct the two rules by throwing away the matched subgraphs and creating a new functional block.

4 Rule Re-analysis

The method described in Section 3 may produce rules which are not very general, because there might be more than one unknown rule used in constructing the precedent. Thus, the learned rules will have RHSs which are a combination of more than one unknown, more general rule. It is much less likely to see again a given complex group of rule instances than it is to see instances of the rules singly. It is possible, however, later to learn new rules which would allow Precedent Analysis to find the more general rules of which the first one was constructed. This leads to the idea of re-analyzing old learned rules in terms of newer rules.

Suppose the rules are always presented to the learning system in the best possible order. Might it not be that Rule Re-analysis is a waste of time? The answer to this is no. This is demonstrated, by counterexample, as follows. Suppose that, unknown as yet to the system, there are four general design rules involved in constructing three precedents. The four design rules are as follows.

- $f_1(x) \Longrightarrow g_1(x)$
- $f_2(x) \Longrightarrow g_2(x)$
- $f_3(x, y) \Longrightarrow g_3(x, y)$
- $f_4(x) \Longrightarrow g_4(x)$

The three precedents are the following:

1. $g_3(f_1(x), f_2(y)) = f_3(g_1(x), g_2(y))$
2. $f_2(f_4(t)) = g_2(g_4(t))$

3. $[g_3(f_1(z), f_2(z)), z = f_4(w)] \equiv$
$[f_3(g_1(z'), g_2(z')), z' = g_4(w)]$

Suppose that the Learning by Failing to Explain system is presented with these precedents in the order 1, 2, 3. On seeing 1, the system is not able to analyze it at all. Likewise, on seeing 2, the system can not analyze it at all. Thus far, the system has 4 rules: two rules implementing blocks representing each of the overall functions of the precedents (one rule for each graph of each precedent).

On seeing precedent 3, the system may analyze it using rules derived from precedent 1. This results in one new rule: $f_4(x) \Longrightarrow g_4(x)$. Rule Re-analysis applies this new rule to precedent 2. This results in the rule, $f_2(x) \longrightarrow g_2(x)$. The system may then re-analyze the precedent-1 rules and arrive at two simpler rules. One has RHS $g_3(f_1(x))$, the other has RHS $f_3(g_1(x))$. Hence, the system is left with the following rules.

- $f_4(x) \Longrightarrow g_4(x)$,
- $f_2(x) \Longrightarrow g_2(x)$,
- $h(z, w) \Longrightarrow g_3(f_1(z), w)$,
- $h(z, w) \Longrightarrow f_3(g_1(z), w)$.

On the other hand, if one picks any of the six possible orders of presentation and applies Precedent Analysis without Rule Re-analysis, the set of rules conjectured is less general than the four rules. For example, suppose they are given in the order 1, 2, 3. Without Rule Re-analysis, Precedent Analysis conjectures the following set of rules as an addition to those made from each entire precedent. (h is a block created by the system.)

- $f_4(x) \Longrightarrow g_4(x)$
- $h(z, w) \Longrightarrow g_3(f_1(z), w)$
- $h(z, w) \Longrightarrow f_3(g_1(z), w)$

It thus failed to find the f_2 rule.

We decide which rule set is more general by asking which is capable of generating the other. Clearly, the set produced using Rule Re-analysis suffices to generate all the rules in the other set. However, there is no derivation of the f_2 rule in terms of the rules produced without Rule Re-analysis. Thus, Rule Re-analysis resulted in more general rules. The reader may verify that all six orders of presentation result in less general rules if Rule Re-analysis is not used.

The fact is that without re-analysis, the system requires *more* precedents to reach a given level of generality. Since precedents are in general much harder to come by than the time needed for Rule Re-Analysis, it is clear that re-analysis is worthwhile.

5 Role vs Behavior

Since Learning by Failing to Explain is intended to be used as a component in a larger learning/design system, it is reasonable to assume that other knowledge sources might exist which enable other forms of reasoning about rules; say, knowledge about the semantics of the functional blocks. Is there a way the system could attempt to judge which contexts the conjectured rules are true in? It turns out that there is a case where a conjectured rule can be judged to be true in all contexts, using only properties of previously known functional blocks.

To state this case most concisely, it is convenient to introduce terminology. A *role* is a mapping from inputs to sets of allowable outputs. That is, each input vector determines a set of allowable output vectors. A role is also called a *behavior* when it *uniquely* determines each output for any given set of inputs. Thus, the squaring function on integers is a behavior, but the square root function on integers is only a role, because it maps negative integers to the empty set. Another example of a role which is not a behavior is when a component has "don't care" entries in its truth table.

To any subgraph, S, of a device, G, there corresponds a role which I shall refer to as the *induced role of S in G*. It is defined as follows. G represents a role (usually in this system, a behavior). Consider replacing S with any S' that maintains the overall behavior of G. Define a new role, f, which maps an input vector, v, to the union over S' of S'(v). It is this (unique) least restrictive role which I call the induced role of S in G. Note that no matter which S' fills the hole, the induced role depends on the hole, not on S'.

The criterion arises as follows. Looking back at the partial parse which generated the conjectured rule, one can ask about the induced roles of the unmatched subgraphs in their respective graphs. They will, of course, be the same, as the induced role depends only on the hole and not on what fills it. The matched portions of the two graphs are identical and they determine the induced roles of their complements.

Suppose this induced role is a behavior. Then there is exactly one behavioral specification which could possibly fill the hole. Thus, the two subgraphs, even though they are structurally different *must* have the same behavior. On the other hand, if the induced role is not a behavior, the two subgraphs may or may not be behaviorally equivalent.

Summing this up, the conjectured rules will be true in all contexts if the induced role in the precedent is a behavior. What is interesting about this criterion is that it can tell us a fact about a previously completely mysterious object (the unmatched subgraph) solely in terms of the properties of known objects (the constituents of the matched portion of the precedent).

Note also that this is merely a sufficient condition for behavioral equivalence, not a necessary one.

6 Summary and Discussion

First, a summary of the main ideas:

- Four interesting design competences can be understood by having a *Design Grammar* as the backbone of a design system: top-down design, optimization, explanation, and Analogical Design.
- *Precedent Analysis*, wherein the learner uses current knowledge to partially understand the precedent before conjecturing a new rule, is a method for learning from precedents which does not require the ability to prove a rule before learning it, as in Explanation-Based Learning, yet still produces more plausible conjectures than empirical generalization methods. It uses current knowledge to guide the system to general rule conjectures.
- *Rule Re-analysis*, the technique of using new rules to try to analyze old ones, is inherently more powerful than simple acceptance of new rules, even if one supposes that the precedents are ordered optimally.
- The distinction between *behavior* and *role* sheds light on the conditions under which the conjectured rule can fail to be true in all contexts.

It would seem that there is an interesting relationship between this work and that of (Berwick, 85)[1]. Berwick's model of learning can be construed as a Learning by Failing to Explain method. His domain was natural language learning, where the

grammars are, of course, string grammars. His mechanism attempted to parse an input sentence according to its current rules as much as possible, then *if the result satisfied certain criteria* the system proposed a new rule. His system did not attempt Rule Re-analysis. He argues that natural languages satisfy certain constraints which enable them to be learned in this manner. Thus, his system could be described as Precedent Analysis, together with some additional criteria regarding when to actually form a new rule.

Inasmuch as there is no reason to believe that the world of design obeys such a learnability constraint, it is not to be expected that Berwick's mechanism would work in learning Design Grammars from any kind of realistic examples. (Of course, any system could learn if it were handed the most general rules as precedents.) It is possible, however, that the use of Rule Reanalysis can substitute, at least in part, for the missing learnability constraint.

Some Limitations.

Experimentation suggests the following limitations. Some of these are limitations of Learning by Failing to Explain in general, and some are limitations of the particular algorithms employed in the current system.

- Sometimes the *maximal* partial parse is not the most desirable partial parse to use. In some cases a much more useful rule can be obtained from a non-maximal parse.
- In some cases, it is desirable to find more than one partial parse. This algorithm currently finds only one.
- The algorithm can be too greedy at times; this causes it to miss a better partial parse by, for example, expanding some node instead of applying a better rule.
- The system needs a better approach to search control in the analysis algorithm. In particular, some method of focusing attention on small sections of large graphs would reduce the size of the search tree generated. Currently, the system keeps track of all paths from the initial graph.

Future Work.

- No account of learning design knowledge is complete without discussion of both acquisition of analytic knowledge and search control knowledge. It would be interesting to investigate Learning by Failing to Explain as applied to these very different types of knowledge.
- The intuition behind the method seems to be applicable to domains other than design domains. How would the knowledge be represented, and what additional issues arise in applying Learning by Failing to Explain to other types of domains?
- How can the search done by the analysis algorithm be reduced?
- A Design Grammar is a restrictive representation. In particular, it needs some method for representing generalized roles and parameterized structures. How will a more powerful representation affect the parsing performance?
- What would it take to be able to reason about induced roles, so that the system could find the contexts in which a given inferred rule is true?

Acknowledgements

Thanks to Patrick Winston, for providing guidance for the original thesis; and thanks to Rick Lathrop and the Reviewers, whose comments greatly improved the final version of this paper.

References

[1] Robert C. Berwick. *The Acquisition of Syntactic Knowledge.* MIT Press, Cambridge Mass., 1985.

[2] Gerald DeJong and Raymond Mooney. *Explanation-Based Learning: An Alternative View.* Technical Report UILU-ENG-86-2208, Coordinated Science Lab, University of Illinois, March 1986.

[3] Thomas Ellman. Generalizing logic circuit designs by analyzing proofs of correctness. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, IJCAI-85, 1985.

[4] Richard E. Fikes, Peter E. Hart, and Nils J. Nilsson. Learning and executing generalized robot plans. *Artificial Intelligence*, 3, 1972.

[5] Robert Joseph Hall. *Learning by Failing to Explain.* Technical Report, M.I.T. Artificial Intelligence Laboratory, 1986. forthcoming.

[6] Robert Joseph Hall. *On Using Analogy to Learn Design Grammar Rules.* Master's thesis, Massachusetts Institute of Technology, 1985.

[7] Sridhar Mahadevan. Verification-based learning: a generalization strategy for inferring problem-reduction methods. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, IJCAI-85, 1985.

[8] Tom M. Mitchell, Richard M. Keller, and Smadar T. Kedar-Cabelli. *Explanation-Based Generalization: A Unifying View.* Technical Report ML-TR-2, SUNJ Rutgers, 1985.

[9] Raymond Mooney and Gerald DeJong. Learning schemata for natural language processing. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985.

[10] Reid G. Smith, Howard Winston, Tom M. Mitchell, and Bruce G. Buchanan. Representation and use of explicit justifications for knowledge base refinement. In *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1985.

[11] Louis I. Steinberg and Tom M. Mitchell. A knowledge based approach to vlsi cad: the redesign system. In *Proceedings of the 21st Design Automation Conference*, IEEE, 1984.

[12] Patrick H. Winston, Thomas O. Binford, Boris Katz, and Michael Lowry. *Learning Physical Descriptions from Functional Definitions, Examples, and Precedents.* Technical Report AIM-679, Massachusetts Institute of Technology, 1983.