Knowledge Level and Inductive Uses of Chunking (EBL)

Paul S. Rosenbloom
Information Sciences Institute
University of Southern California
4676 Admiralty Way
Marina del Rey, CA 90292
rosenbloom@isi.edu

Jans Aasman
Traffic Research Centre
Rijksuniversiteit Groningen
Rijksstraatweg 76,
9752 AH Haren (Gn), The Netherlands
aas%hgrrug5.bitnet@cunyvm.cuny.edu

Abstract

When explanation-based learning (EBL) is used for knowledge level learning (KLL), training examples are essential, and EBL is not simply reducible to partial evaluation. A key enabling factor in this behavior is the use of domain theories in which not every element is believed a priori. When used with such domain theories EBL provides a basis for rote learning (deductive KLL) and induction from multiple examples (nondeductive KLL). This article lays the groundwork for using EBL in KLL, by describing how EBL can lead to increased belief, and describes new results from using Soar's chunking mechanism — a variation on EBL — as the basis for a task-independent rote learning capability and a version-space-based inductive capability. This latter provides a compelling demonstration of nondeductive KLL in Soar, and provides the basis for an integration of conventional EBL with induction. However, it also reveals how one of Soar's key assumptions — the *non-penetrable memory assumption* — makes this more complicated than it would otherwise be. This complexity may turn out to be appropriate, or it may point to where modifications of Soar are needed.

Introduction[1]

Recent analytical papers on explanation-based learning (EBL) (DeJong & Mooney, 1986; Mitchell, Keller, & Kedar-Cabelli, 1986) comment on how training examples are not logically necessary for EBL (Prieditis, 1988; van Harmelen & Bundy, 1988). Their point is that a training example may serve a useful search control function in EBL — guiding the learner to regions of performance which it would be useful to operationalize — but that the resulting operationalized rule is just a specialization, in the general case, of what a partial evaluation (PE) mechanism could achieve without the training example. This is an important point which reveals a previously submerged connection between learning and program transformation. However, it is flawed by its neglect of the use of EBL in knowledge level learning (KLL) (Flann & Dietterich, 1989; Rosenbloom, Laird, & Newell, 1987); that is, for the acquisition of knowledge not implied by what is already known (Dietterich, 1986). In such situations, the combination of the training example and goal concept — an *instance* — plays an essential role; one that is quite comparable to the role of instances in classic inductive concept learning systems.

The first task of this article is to lay necessary groundwork for the use of EBL in KLL. The key idea is to explain — actually, to *rationalize* — instances via a low-belief domain theory, and then to use EBL to acquire a high-belief rule from the rationalization. Unlike in PE, the instances play a crucial role here in determining what is rationalized, and thus what becomes believed. We then build on this foundation in the context of Soar (Laird, Newell, & Rosenbloom, 1987; Rosenbloom *et al.*, 1990) — whose learning mechanism, chunking, is a variation on EBL (Rosenbloom & Laird, 1986) in which new rules are acquired from a dependency analysis of the traces of rules that fire during subgoal-based problem solving — to take several steps towards the realization of the *integrated-learning hypothesis* (Rosenbloom, 1988). This hypothesis states that "Rote learning, empirical generalization, and explanation-based learning arise as variations in the knowledge-reconstruction process", where "knowledge-reconstruction" should be read as "rationalization". Traditional explanation-based learning differs from rote memorization and induction in that the former uses only high-belief rationalizations while the latter two necessitate aspects that are initially low in belief. Rote memorization differs from induction in that the latter utilizes additional knowledge to affect rationalization.

The subsequent sections introduce a new task-independent, rote-memorization capability in Soar;

[1]This research was sponsored by the National Aeronautics and Space Administration under cooperative agreement number NCC 2-538, and by the Defense Advanced Research Projects Agency (DOD) under contract number N00039-86C-0033 (via subcontract from the Knowledge Systems Laboratory, Stanford University). We would like to thank Yoram Reich, Haym Hirsh, Craig Miller, Peter Pirolli, and John Laird for their helpful comments.

extend this to induction from multiple examples, providing a compelling example of nondeductive knowledge level learning (NKLL) (Dietterich, 1986) in Soar, and introducing a complication caused by Soar's inability to directly examine its own rules; cover the third leg of the integrated-learning hypothesis, explanation-based learning, and its use in induction; and conclude.

EBL and KLL

EBL can be applied over many types of domain theories — the only requirement being the provision by the domain theory of a way to generate a (possibly generalized) dependency structure for the instance that relates the training example to the goal concept. In a classical EBL domain theory, all of the elements — e.g., facts and rules — are ascribed a uniform high level of belief. In such a domain theory, EBL performs symbol level learning by explicitly storing knowledge that is already implicit in this domain theory. However, it does not alter levels of belief — they are all already as high as they can get. For knowledge level learning, the domain theory needs to include low-belief elements. With such a theory, EBL can lead to knowledge level learning by increasing the level of belief in selected elements of the theory.

Consider the example of a system with a domain theory that allows it to abductively generate rationalizations — that is, plausible explanations — for what it sees (or hears). One of the key ways in which a rationalization differs from a simple deductive proof — as used in EBG (Mitchell, Keller, & Kedar-Cabelli, 1986), for example — is that the facts and rules utilized in a rationalization need not be completely believed in order to be used. It is in fact essential to the rationalization process that the system be able to derive from its domain theory not just facts that are known to be true, but also ones which are not yet believed. For example, suppose the instance consists of telling the system that "Fred, a dolphin, is warm blooded" — "Fred is a dolphin" is the training example, and "Fred is warm blooded" is the goal concept. To rationalize this knowledge it might use that "dolphins are mammals" and "mammals are warm blooded", even if its a priori belief is that dolphins are fish rather than mammals. This explanation could have been created in the absence of the instance, but it would have little a priori believability. It is the existence of the instance that provides grounds for increasing the believability of the explanation.

When this example is extended to the EBL/PE situation, it becomes clear that arbitrary rules created by partially evaluating this domain theory would have low believability, while comparable rules created by EBL for specific instances could have much higher believability. The instances allow EBL to increase the scope of what is believed, thus enabling knowledge level learning. In the extreme it is possible to start with a theory consisting of a generator able to produce data structures representing all possible pieces of knowledge, all with zero belief. Such a theory has all possible knowledge implicit in it, but none of it initially believed. EBL, in conjunction with appropriate instances, can then be used selectively to learn anything, by increasing the level of belief in the appropriate, possibly implicit, knowledge structures.

One way to view this process is as explicit belief propagation, where there are belief-propagation rules that are used to compute a level of belief for an explanation — and thus for the rules learned via EBL from the explanation — as a function of the believability of the individual domain theory elements and the instance. An alternative view of this process, and the one that guides the research reported here, is that the instance acts as a filter, letting through only those explanations which should be believed. Learning then only occurs for these believed explanations.

To support this type of processing, ***plausible*** domain theories — that is, theories in which only plausible explanations can be generated for conclusions — are required. Such a theory can potentially explain things that aren't true — necessitating the use of instances as filters — but what is true is explained in a plausible manner. As long as a domain theory meets this condition, the a priori believability of the individual elements of the theory can be zero without affecting the believability of what is learned. At the extreme end, the domain theory could contain elements which are not representational in the domain, such as the letter "d" — or which are representational but do not have belief values, such as "dolphin" — but which can be combined syntactically to generate elements which do have belief values.

Given EBL and plausible low-belief domain theories, the one missing component is what makes the newly learned rule believed. A variety of approaches are possible, including ones that augment the EBL mechanism to generate explicit belief values, and ones that partition learned rules — which automatically receive high belief — from domain theory rules. In Soar a variation on this latter approach is used. The domain theory corresponds to the problem space used in a subgoal (Rosenbloom & Laird, 1986) — in our case, this might be a *generation* problem space, where all possible structures are generatable, but none are believed. However, the learned rule is always created for the problem space in the parent goal; perhaps a *fact* problem space, in which all retrievable structures are believed. If restrictions are then placed on which problem spaces are utilized at any point in time, it is possible to know what level of belief — 0 or 1 — to assign to the retrieved knowledge. This point is related to the one recently made in (Flann & Dietterich, 1989). They focus on how EBL can perform KLL if an explanation that is generated for one concept is used to define a second concept that is a specialization of the first one. In our work, EBL performs KLL by transferring (possibly implicit) structures from an unbelieved domain theory to a believed domain theory that contains a subset of the structures in the original theory. Despite the differences, what both have in common — in fact, what every use of EBL for KLL must have in common — is that the learned knowledge is used differently than would be the domain theory from which it is learned. It can be used at a different belief level (as here), for a different goal concept (Flann & Dietterich, 1989), or even as evidence that a particular episode occurred (in which the rule was learned) (Rosenbloom, Newell, & Laird, 1990).

Rote Memorization

In previous work this general approach to knowledge level learning has been employed to perform several simple rote memorization tasks — recognition, recall, cued recall, and paired-associate recall — for hierarchical letter strings and for objects that are described by attributes with values (Rosenbloom, Laird, & Newell, 1987; Rosenbloom, Laird, & Newell, 1988; Rosenbloom, Newell, & Laird, 1990). This work also provided solutions for two additional problems that arise when rules are to be learned that recall new structure: (1) the *data chunking problem* — how to enable the retrieval of new information without its already being present — and (2) the *selective retrieval problem* — how to avoid retrieving everything ever learned. The data chunking problem is solved by reconstructing new knowledge from what is already known — that is, the domain theory — rather than directly rationalizing the input structures representing the new knowledge. Since this reconstruction process is not dependent on the input structures, tests of the input do not appear in the conditions of the learned rules. The selective retrieval problem is solved by selectively acquiring retrieval cues as conditions of learned rules.

Recently, this earlier work has been extended with the development of a general, task-independent rote-memorization operator. This is a normal Soar operator that uses problem solving in a subgoal to implement solutions to the data chunking and selective retrieval problems. The operator takes two arbitrary graphs of Soar working memory elements as inputs — the first is the training example, and the second is the goal concept. The result of memorizing the pair is the acquisition of a rule that tests for the existence of the first graph (the *cue graph*) and if the test succeeds, retrieves the second graph (the *recalled graph*) into working memory. To do this, the memorization operator reconstructs the recalled graph by assembling primitive domain-theory elements into a copy of the graph (solving the data chunking problem), and then makes this copy dependent on the structure of the cue graph (solving the selective retrieval problem). Figure 1 shows such a chunk, schematized as a pair of graph structures. The cue and recalled graphs are attached to operator <O1> in the rule's conditions and actions, respectively.

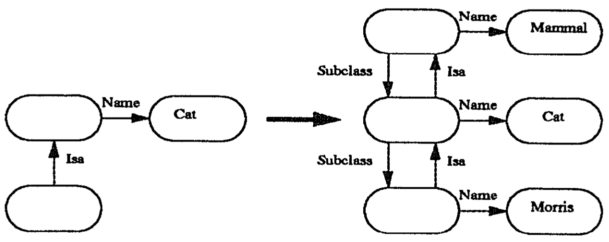

Figure 1: Chunk learned from memorizing a pair of graph structures.

Multi-Example Induction

When the contents of the initial domain theory are restricted to only elements that are completely believed, it is not possible for the domain theory itself to perform inductive leaps. If a system containing such a domain theory is to perform inductive generalization, it must be done outside the domain theory — in the EBL mechanism, for example. The most common approach is to augment EBL's standard explanation processing with some form of inductive postprocessing of either the entire explanation, or just its operational components (Flann & Dietterich, 1989; Hirsh, 1989; Sarrett & Pazzani, 1989).

If, on the other hand, the contents of the initial domain theory can include unbelieved elements, the option opens up of doing induction directly in the domain theory, and leaving explanation processing unaltered. This was proposed in (Rosenbloom, 1988) as part of the integrated-learning hypothesis, and is the approach taken here.[2] The domain theory includes knowledge about how to reconstruct presented objects (a stripped-down version of the task-independent memorization operator), generalization hierarchies (the basis of the concept language), rules which perform inductive leaps, and rules learned from prior instances. When a new instance is perceived, this domain theory is used to determine what is to be rationalized — it may be a generalization of the instance, rather than the instance itself — as well as how it should be rationalized.

[2]Widmer recently proposed a similar "Explain and Compile" approach to using EBL over an expanded range of domain theories (Widmer, 1989). Anderson has also used knowledge compilation, a form of EBL, over analogical and discriminative theories (Anderson, 1986).

One decision that must be made at this point is the orientation of the rule to be learned — whether the concept's name should be in the actions and its definition in the conditions, or vice versa. In EBL (and chunking), the concept definition appears in the conditions because that is where EBL has its primary generalization effect, and also where the definition can act as a recognizer of instances of the concept. However, to go beyond simple concept recognition — to retrieval and modification of the concept definition as further instances are processed — requires a fuller declarative access to the definition. Such access is not available in Soar if the definition is stored in the conditions because Soar's rule memory is *non-penetrable* — rules can be executed (forward), but can not be directly examined. Non-penetrability arises because the rules are compiled procedures that can be executed but not examined. In psychological terms, they represent *automatized* behavior (for example, (Shiffrin & Schneider, 1977)). A consequence of non-penetrability is that, to access the concept definition explicitly, it must be stored in the rule's actions, where it can be retrieved by rule execution. Once retrieved, the concept definition can be used to influence what is learned for new instances, and to interpretively recognize instances of the concept (concept recognition rules, in which the definition is in the conditions, can be learned by chunking this interpretation process). As discussed later, the downside of this decision is that the generality of the concept definition is affected only by the decision of what to rationalize, and not by the rationalization process or the chunking/EBL mechanism.

The domain theory that has been implemented within Soar utilizes a variant of the Focussing algorithm (Young, Plotkin, & Linz, 1977; Bundy, Silver, & Plummer, 1985). This is a version space algorithm that works for spaces describable by a conjunction of attributes, where each attribute is defined by a tree-structured generalization hierarchy. The key ideas underlying the implemented algorithm are that: (1) the version space is kept factored — bounds are maintained independently for each attribute (Subramanian & Feigenbaum, 1986); (2) only near-miss negatives (Winston, 1975) are processed (the *zero option*)[3] to guarantee that the boundary sets will not fragment; and (3) once the first positive example is processed, the entire factored version space is explicitly represented, rather than just the boundary sets (at worst this requires space proportional to the number of attributes times the maximum depth of the generalization hierarchies).

Suppose that the task is to learn that the definition of (GOOD=TRUE) is (MOBILITY=MOBILE, SHAPE=ANY, SIZE=ANY). The algorithm starts with the goal concept, (GOOD=TRUE), and a positive example, such as (MOBILITY=WHEELED, SHAPE=SQUARE, SIZE=LARGE).[4] It then uses the generalization hierarchies in the domain theory to elaborate the example with all of the superclasses of its attributes' values, yielding for this case (MOBILITY={ANY, MOBILE, WHEELED}, SHAPE={ANY, POLYGON, REGULAR, SQUARE}, SIZE={ANY, LARGE}). This elaborated example is then memorized as the initial version space, with the retrieval cue being the goal concept:

```
(GOOD=TRUE) -->
          (MOBILITY={ANY, MOBILE, WHEELED},
           SHAPE={ANY, POLYGON, REGULAR, SQUARE},
           SIZE={ANY, LARGE})
```

The memorization operator succeeds here only because the initial domain theory implicitly contains within itself every possible version space that could be generated for (GOOD=TRUE), all initially unbelieved. The example then determines which of these version spaces is explicitly stored, and thus believed.

When a succeeding positive example of this same goal concept is encountered, processing again begins by elaborating it with its values' superclasses. The current version space for the concept is then retrieved and compared to the elaborated example. All value classes in the concept version space that are not also in the elaborated example are then rejected from the version space (a form of incremental version space merging (Hirsh, 1989)). Chunking over this rejection process yields rules which, in the future, reject the inappropriate segments of the old version space. This rejection process is itself reconstructive so that the updated version space can later be retrieved without the presence of the positive example.

As an example, suppose that the next example for this goal concept is (MOBILITY=TRACKED, SHAPE=ELLIPSE, SIZE=LARGE), which when elaborated becomes (MOBILITY={ANY, MOBILE, TRACKED}, SHAPE={ANY, CONIC, ELLIPSE}, SIZE={ANY, LARGE}). From this example, the following rejection rules are learned:

```
(GOOD=TRUE, MOBILITY=WHEELED) -->
                                 (MOBILITY=WHEELED)-
(GOOD=TRUE, SHAPE=POLYGON) --> (SHAPE=POLYGON)-
(GOOD=TRUE, SHAPE=REGULAR) --> (SHAPE=REGULAR)-
(GOOD=TRUE, SHAPE=SQUARE) --> (SHAPE=SQUARE)-
```

The next time the same goal concept is seen, all five learned rules fire, yielding the following updated version space: (MOBILITY={ANY, MOBILE}, SHAPE=ANY, SIZE={ANY, LARGE}).

Learning from negative examples introduces an additional issue: because all of the knowledge learned from positive examples is cued off of

[3]With a suitable training order, far misses are not needed for convergence (Bundy, Silver, & Plummer, 1985).

[4]This example, and all later ones have been implemented in Soar (Version 5.0.2), and verified to run correctly.

(GOOD=TRUE), it will not be retrieved automatically for a negative example, where (GOOD=FALSE) (the converse is also true). The solution to this is to transiently cue with both TRUE and FALSE, thus retrieving all of the knowledge so far learned about the concept, and then to proceed to process the example while maintaining only the correct value (FALSE, in this case). Then, if the example is a near-miss — that is, if it mismatches in at most one attribute — those classes of the mismatched attribute that match are rejected from the version space. If the example is a far miss, it is ignored.

As an illustration, suppose the third example is the negative example (MOBILITY=STATIONARY, SHAPE=RECTANGLE, SIZE=LARGE). This example is elaborated with superclass information to become (MOBILITY={ANY, STATIONARY}, SHAPE={ANY, POLYGON, IRREGULAR, RECTANGLE}, SIZE={ANY, LARGE}), and then the existing information about the concept definition is retrieved. The mismatched attribute is MOBILITY, and the class that matches for that attribute is ANY. The rule learned for rejecting this class is:

```
(GOOD=FALSE, MOBILITY=ANY) --> (MOBILITY=ANY)-
```

The resulting concept version space is (MOBILITY=MOBILE, SHAPE=ANY, SIZE={ANY, LARGE}).

These examples demonstrate that Soar can be used not only as the basis for rote memorization (deductive KLL), but also for induction (nondeductive KLL). Inductive augmentations of the EBL mechanism are not required, because the induction occurs directly in using the domain theory. As suggested by the integrated-learning hypothesis, rote learning and induction are distinguished by differences in the rationalization process. However, contrary to the intent of the hypothesis, the difference is in terms of what is rationalized rather than how it is rationalized (forced by the decision to store the concept definition in actions rather than conditions). In the induction case, rather than directly rationalizing the concept name in terms of the training example, it is rationalized in terms of the version space (or changes to it). This choice of what to rationalize is essentially a choice of what to learn. Here, this choice was based on the instance, the generalization hierarchies, the previous version space, and knowledge about how to induce. Bringing other knowledge to bear should allow additional useful variations on this choice.

Explanation-Based Learning

Using the chunking/EBL mechanism to perform explanation-based learning — that is, the standard form of symbol level learning — supports the third, and final leg of the integrated learning hypothesis. However, this needs no explicit demonstration here, as it is the foundational result of EBL. Instead, what is of interest here is the extent to which, in practice, this use of EBL can be integrated with the induction process described in the previous section. In this section we examine three successively weaker versions of this question. The first version is whether the direct use of EBL in induction, as described in the previous section, provides the requisite form of symbol level learning — that is, is EBL itself performing significant acts of "justifiable" generalization during induction? The answer, upon inspection, is "no". EBL is storing the results of inductive processing, but it is not itself contributing to their level of generalization. This is forced by the decision to store the concept definition in rule actions.

The second version is whether the standard use of EBL to perform symbol level learning — that is, with a classical believed domain theory — can help the inductive process described in the previous section (which is independently using EBL). The answer to this version of the question is once again "no". To see this, consider a domain theory with the following two believed rules.

```
(MOBILITY=WHEELED) --> (FAST=TRUE)
(FAST=TRUE, SHAPE=CONIC) --> (GOOD=TRUE)
```

If these rules are used to relate the training example to the goal concept, the following rule is learned.

```
(MOBILITY=WHEELED, SHAPE=CONIC) --> (GOOD=TRUE)
```

This is exactly what EBL should learn. However, it is difficult to use in the induction process described in the previous section because the generalized example is in the conditions of the rule — thus the rule retrieves the goal concept when a matched training example is present, rather than retrieving the generalized example when the goal concept is present. This failure is disturbing because this is the type of gain achieved by other hybrid approaches, such as (Flann & Dietterich, 1989; Hirsh, 1989; Sarrett & Pazzani, 1989). In these other approaches, this problem is solved by enabling the induction process to directly access the explanation, its operational fringe, or the resulting rule. In the present approach, the rule can be fired, but neither it nor any part of the explanation can be directly examined.

The third version is whether some form of explanation-based learning can lead to generalizations that are useful in the induction process described in the previous section. The answer here is finally "yes". However, it requires augmenting the domain theory itself with the ability to interpret rules and to generate and process explanations. These rules are not Soar's native rules, but declarative structures of limited expressibility that are stored in the actions of Soar's rules. These rules are retrieved as needed to support a backward-chaining process that starts with the goal concept and ends when it grounds out in attribute-value pairs contained in the elaborated example (the operational predicates). The operational fringe of the explanation derived from this process is a generalization of the example. Based on the approach in (Hirsh, 1989), this generalized example is used in induction by converting it into an explicit, factored version space — by assigning values to unmentioned attributes ({ANY} for positive examples, and the values in the concept version space for negative examples) and then elaborating it with superclasses — and then merging it with the current concept version space.

As illustration, consider the positive example (MOBILITY=WHEELED, SHAPE=CIRCLE, SIZE=SMALL), which becomes upon elaboration (MOBILITY={ANY, MOBILE, WHEELED}, SHAPE={ANY, CONIC, CIRCLE}, SIZE={ANY, SMALL}). If the domain theory consists of the two rules above, backward chaining yields an operational fringe of (MOBILITY=WHEELED, SHAPE=CONIC), which is more general than the original example because it ignores SIZE, and generalizes SHAPE from CIRCLE to CONIC. When this generalized example is extended to cover the unmentioned attributes, and elaborated, it becomes (MOBILITY={ANY, MOBILE, WHEELED}, SHAPE={ANY, CONIC}, SIZE=ANY). When this description is then merged with the concept version space, the result is (MOBILITY=MOBILE, SHAPE=ANY, SIZE=ANY). The rule learned from this processing is:

```
(GOOD=TRUE) --> (SIZE=LARGE)-
```

This same general approach can be used to incorporate other forms of knowledge into the induction process. So far, we have partial implementations of the use of irrelevance knowledge (Subramanian & Genesereth, 1987) and determinations (Davies & Russell, 1987; Mahadevan, 1989; Russell, 1988; Widmer, 1989) in the induction process.

When taken together, the answers to the three versions of the question reveal that explanations can be effectively combined with induction in this approach, but that this is achieved only by building additional declarative rule interpretation and EBL mechanisms into the domain theory. The native mechanisms are not usable because there is no way to access the rules (or explanations) they create as declarative structures, as required by the induction process.

The question this raises is whether this is evidence that the Soar architecture needs to be changed or is evidence that some of our preconceived notions about induction, and its interaction with chunking/EBL, need to be changed. While the former is a distinct possibility, the utility of architectures as theories of intelligence stems in large part from their ability to predict unexpected but important phenomena. If the architecture is changed whenever one of its consequences violates preconceived notions, this benefit is lost. Also potentially lost are the positive consequences of the way the changed component currently works. The component is usually the way it is for good reason, which in this case is the ability to model basic aspects of human memory. Thus it is useful, before jumping in and changing the architecture, to first consider the possibility that Soar is revealing something important here. When this is done, at least one intriguing speculation arises — that chunking and EBL, though quite similar in mechanism (both compile dependency structures), are really distinct capabilities. Chunking is an automatic architectural process (it learns for every result of every subgoal, and does not compete for cognitive resources with performance), of fixed capability (how it works is not affected by what the system knows), which compiles recognition-driven procedures (productions) from experience. It is an appropriate, and effective, generalized long-term caching mechanism; but it really is a low-level mechanism that is in some ways more analogous to neural-network learning algorithms than to EBL. However, an intelligent system also needs to be able to deliberately create and utilize declarative explanations of new phenomena. This is where EBL, as used here in concept learning, comes in. It is a deliberate cognitive process, of open capability, which processes and creates declarative structures that can be used in induction, and which can also yield behavior, but only indirectly, through interpretation.

Conclusions

By making a distinction between what is in the domain theory (either implicitly or explicitly) and what is believed, it is possible to distinguish the symbol level and knowledge level uses of EBL — symbol level uses make implicit knowledge explicit, while knowledge level uses make unbelieved knowledge believed. This idea has been explored here as the foundation for chunking(EBL)-based rote memorization (deductive KLL) and induction (nondeductive KLL) capabilities. Utilizing unbelieved knowledge enables induction to be performed in the domain theory itself, rather than as a post hoc process. Chunking is used in this induction process to store the initial version space, and to record modifications to it that are required by new instances of the concept. These capabilities demonstrate why EBL is not simply reducible to partial evaluation — the training examples are essential.

When combined with the standard use of EBL for symbol level learning, these capabilities provide the three legs of support required by the integrated learning hypothesis. However, the support is weakened by the difference between rote learning and induction arising from differences in what is rationalized rather than how it is rationalized. Further weakening is engendered by the difficulty in using EBL to generalize instances for use by induction. This has been accomplished, but only by implementing an additional declarative EBL mechanism in the domain theory. Both of these weaknesses occur because of the choice to store the inductive concept definition in the actions of learned rules (rather than in the conditions), which is itself forced by the non-penetrability of Soar's rules, and the resulting difficulty in determining the contents of rule conditions. As discussed in the previous section, this may actually turn out to be appropriate, or it may reveal an aspect of Soar that should be altered.

References

Anderson, J. R. 1986. Knowledge compilation: The general learning mechanism. In R. S. Michalski, J. G. Carbonell, & T. M. Mitchell (Eds.), *Machine Learning: An Artificial Intelligence Approach, Volume II.* Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Bundy, A., Silver, B., & Plummer, D. 1985. An analytical comparison of some rule-learning programs. *Artificial Intelligence, 27*, 137-181.

Davies, T. R., & Russell, S. J. 1987. A logical approach to reasoning by analogy. *Proceedings of IJCAI-87.* Milan.

DeJong, G., & Mooney, R. J. 1986. Explanation-based learning: An alternative view. *Machine Learning, 1*, 145-176.

Dietterich, T. G. 1986. Learning at the knowledge level. *Machine Learning, 1*, 287-315.

Flann, N. S., & Dietterich, T. G. 1989. A study of explanation-based methods for inductive learning. *Machine Learning, 4*, 187-226.

Hirsh, H. 1989. Combining empirical and analytical learning with version spaces. *Proceedings of the Sixth International Workshop on Machine Learning.* Cornell.

Laird, J. E., Newell, A., & Rosenbloom, P. S. 1987. Soar: An architecture for general intelligence. *Artificial Intelligence, 33*, 1-64.

Mahadevan, S. 1989. Using determinations in EBL: A solution to the incomplete theory problem. *Proceedings of the Sixth International Workshop on Machine Learning.* Cornell.

Mitchell, T. M., Keller, R. M., & Kedar-Cabelli, S. T. 1986. Explanation-based generalization: A unifying view. *Machine Learning, 1*, 47-80.

Prieditis, A. E. 1988. Environment-guided program transformation. G. F. DeJong (Ed.), *Proceedings of the AAAI Symposium on Explanation-Based Learning.* Stanford, CA: AAAI.

Rosenbloom, P. S. 1988. Beyond generalization as search: Towards a unified framework for the acquisition of new knowledge. G. F. DeJong (Ed.), *Proceedings of the AAAI Symposium on Explanation-Based Learning.* Stanford, CA: AAAI.

Rosenbloom, P. S., & Laird, J. E. 1986. Mapping explanation-based generalization onto Soar. *Proceedings of AAAI-86.* Philadelphia.

Rosenbloom, P. S., Laird, J. E., Newell, A., & McCarl, R. 1990. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Artificial Intelligence*, . In press.

Rosenbloom, P. S., Laird, J. E., & Newell, A. 1987. Knowledge level learning in Soar. *Proceedings of AAAI-87.* Seattle.

Rosenbloom, P. S., Laird, J. E., & Newell, A. 1988. The chunking of skill and knowledge. In B. A. G. Elsendoorn & H. Bouma (Eds.), *Working Models of Human Perception.* London: Academic Press.

Rosenbloom, P. S., Newell, A., & Laird, J. E. 1990. Towards the knowledge level in Soar: The role of the architecture in the use of knowledge. In K. VanLehn (Ed.), *Architectures for Intelligence.* Hillsdale, NJ: Erlbaum. In press.

Russell, S. J. 1988. Tree-structured bias. *Proceedings of AAAI-88.* St. Paul, MN.

Sarrett, W. E. & Pazzani, M. J. 1989. One-sided algorithms for integrating empirical and explanation-based learning. *Proceedings of the Sixth International Workshop on Machine Learning.* Cornell.

Shiffrin, R. M. & Schneider, W. 1977. Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. *Psychological Review, 84*, 127-190.

Subramanian, D., & Feigenbaum, J. 1986. Factorization in experiment generation. *Proceedings of AAAI-86.* Philadephia.

Subramanian, D., & Genesereth, M. R. 1987. The relevance of irrelevance. *Proceedings of IJCAI-87.* Milan.

van Harmelen, F. & Bundy, A. 1988. Explanation-based generalization = partial evaluation. *Artificial Intelligence, 36*, 401-412.

Widmer, G. 1989. A tight integration of deductive and inductive learning. *Proceedings of the Sixth International Workshop on Machine Learning.* Cornell.

Winston, P. H. 1975. Learning structural descriptions from examples. In Winston, P. H. (Ed.), *The Psychology of Computer Vision.* New York: McGraw Hill.

Young, R. M., Plotkin, G. D., & Linz, R. F. 1977. Analysis of an extended concept-learning task. *Proceedings of IJCAI-77.* Cambridge.