A Knowledge Level Analysis of Explanation-Based Learning

Masayuki Numao*
CSLI
Stanford University
Stanford, CA 94305-4115 U.S.A.
numao@csli.stanford.edu

Masamichi Shimura
Department of Computer Science
Tokyo Institute of Technology
2-12-1 O-okayama, Meguro, Tokyo 152 Japan
shimura@cs.titech.ac.jp

Abstract

Although Explanation-Based Learning (EBL) has up to now been used only for deductive learning that improves execution speed, chunking in Soar, which is closely related to EBL, was demonstrated to acquire new knowledge. We first analyze such *knowledge level learning* in EBL, by showing that a rule set is specialized when rules in it are replaced by their composition, and is generalized when a rule is replaced by its decomposition. Counting on this discussion, we propose a method to learn generalized rules by making a decomposition of instances. Since this method acquires knowledge that is deduced from domain theory and induced from instances, it is a natural method for *combining empirical and explanation-based learning*. We demonstrate deductive and inductive aspects of our method by examples of logic circuit design and geometric analogy.

1 Introduction

As shown in the solution of *explanation-based paradox* [5], Explanation-Based Learning (EBL) has up to now been used only for deductive learning that improves execution speed. As opposed to this, chunking in Soar, which is closely related to EBL, was demonstrated to acquire new knowledge [19]. The reason why EBL has only analyzed at the *symbol level* [3, 13] is that generalization or specialization has been viewed in each rule or concept. First, we introduce hypotheses into domain theory and define specialization and generalization at the level of rule set or deductive closure. This enables an analysis of EBL at the *knowledge level*. Based on this discussion, we propose a method to learn at the knowledge level by extracting only composed rules verified by explanations. Since this method acquires knowledge that is deduced from domain theory and induced from instances, it is a natural method for *combining empirical and explanation-based learning*[1]. We have some demonstrations to see deductive and inductive aspects of our method.

*On leave from Tokyo Institute of Technology.

2 Knowledge Level Learning

According to Dietterich [3], knowledge level learning is defined as follows:

Definition 1 (knowledge level learning) *A system is said to exhibit* knowledge level learning *(KLL) if it exhibits a positive change in its knowledge level description over time. In other words, suppose we observe a system at time T_1 and attribute to it knowledge K_1. At some later time T_2, suppose we observe the system again and attribute knowledge K_2. If $K_2 > K_1$, then we say that the system has learned at the knowledge level.*

In Explanation-Based Learning [2, 11], the learner is given a *training example* and *domain theory*, and it determines a generalization of the training example by the following two steps:[2]

1. *Explain:* Construct an *explanation* that proves the *training example* in terms of the *domain theory*.
2. *Generalize:* Determine a set of sufficient conditions under which the explanation holds.

To learn at the knowledge level, we propose to add some *hypothesis generators* to domain theory. A hypothesis generator is defined as a rule that may derive

[1] See papers by the study group of combining empirical and explanation-based learning[20].

[2] This definition of EBL is simplified to clarify the successive discussion.

incorrect examples, and is used to acquire new knowledge verified by instances. The system not only improves execution speed but also learns at the knowledge level by:

1. using such hypothesis generators,
2. separating the knowledge base of learned rules from that of domain theory,
3. extracting only composed rules verified by instances.

We call EBL in this setting *KL-EBL* (*k*nowledge-*l*evel *EBL*), where we show that a rule set is specialized when rules in it are replaced by their composition, and is generalized when a rule is replaced by its decomposition. This means that *Explanation-Based Learning* $\neq$ *Partial Evaluation* as opposed to the argument by van Harmelen and Bundy [22].

3 Composition and Decomposition of Rules

We discuss learning in a rule-based system, which is given a *rule set* and a *problem*, and infers an *answer* to the problem.

A rule shows that its LHS (Left Hand Side) derives its RHS (Right Hand Side). The following rule shows the commutative law of addition:

$$comm : (*x + *y) \rightarrow (*y + *x).$$

comm is the rule name. This rule shows that $(*x + *y)$ derives $(*y + *x)$. A symbol with $*$ denotes a variable.

Composition and decomposition of rules are the following manipulation:

Definition 2 (composition and decomposition) *If an application of rule* $r1$ *has the same effect as the successive applications of firstly* $r2$ *and then* $r3$, $r1$ *is called a* composition *of* $r2$ *and* $r3$. *Conversely, the set* $\{r2, r3\}$ *is a* decomposition *of* $r1$.

The composition of rules is the same as *MACROP* in STRIPS [1] and the result of generalization in Explanation-Based Learning [12]. The inverse manipulation generates a decomposition, whose detail is discussed later.

The system is given problems and their answers, and acquires a new set of rules by which correct answers can be obtained to the similar problems. Such a pair $(problem, answer)$ is called an *instance*. Our goal is to generalize such a given instance, where generalization is defined as follows:

Definition 3 (generalization) *Let* $I1$ *and* $I2$ *be instance sets derived from rule set* $S1$ *and* $S2$, *respectively. Rule set* $S1$ *is a* generalization *of rule set* $S2$, *denoted* $S1 \preceq S2$, *iff* $I1 \supseteq I2$. *Conversely,* $S2$ *is a* specialization *of* $S1$.

The following theorems are proved by using the above definitions:

Theorem 1 *If* $r1$ *is the composition of* $r2$ *and* $r3$ *then* $\{r2, r3\} \preceq \{r1\}$.

Proof: Let i be any instance derived from $\{r1\}$. By definition 2, i is also derived from $\{r2, r3\}$. Therefore, $\{r2, r3\} \preceq \{r1\}$. In general, $\{r2, r3\} \not\preceq \{r1\}$, since some instances derived from $r2$ are not derived from $r1$. □

Theorem 2 *If* $S1 \preceq S2$ *and* $S3 \preceq S4$ *then* $(S1 \cup S3) \preceq (S2 \cup S4)$.

Proof: Suppose $S2 \cup S4$ derives i. Consider a rule in $S2$ is applied at a step in the derivation. Since $S1 \preceq S2$, $S1$ derives the same step. Similarly when a rule in $S4$ is applied, $S3$ derives the same step. Since $S1 \cup S3$ derives each step of the derivation, $S1 \cup S3$ derives i, by which we conclude that $(S1 \cup S3) \preceq (S2 \cup S4)$. □

In other words, when two rules are replaced by their composition, the rule set is specialized, and when a rule is replaced by its decomposition, it is generalized. Thus, we can generalize (specialize) a given set of rules by making their decomposition (composition).

4 A Method of KL-EBL

In this section, we propose a method of KL-EBL. We call an instance with a correct answer a *positive instance*, and one with an incorrect answer a *negative instance*. An instance is assumed to be a rule whose LHS is its problem and RHS its answer. Our goal is to find a decomposition of the positive instances that does not derive any negative instance.

4.1 Explanation

To make a decomposition, an *explanation* shown in Figure 1 is constructed. It is a directed graph whose nodes are rule names, and gives an analysis of an instance in terms of domain theory, i.e., how the instance is generated as a composition of rules in the domain theory. The arc which connects *rule*1 and *rule*2 shows that these rules make a composition at the *position*, which is a selector, such as *car*, *cdr* or *cadr*, with a sign. If the sign is $+$, *rule*2 is attached on a subexpression of *rule*1's RHS specified by the selector. If the sign is $-$,

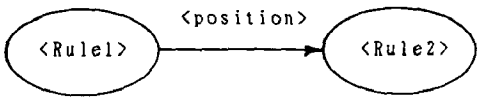

Figure 1: Explanation.

*rule*1 is attached on a subexpression of *rule*2's LHS. If the selector is omitted, *rule*2 is attached on the whole expression of *rule*1's RHS.

The explanation is constructed by the following steps:

1. Find the shortest path (or, when *cost* is attached to each rule, the minimum cost path) that shows how the answer is inferred from the problem by using a rule-based system equipped with the domain theory.

2. Analyze dependencies among the rule applications, and represent them as a graph.

NAT (NonAlgorithmic Translator) [15] is implemented in Utilisp for this purpose. NAT has two working memories, Problem WM and Answer WM. Initially, a problem and an answer are given in Problem WM and Answer WM, respectively. Until both the WMs match, forward reasoning rules are applied to Problem WM, and backward reasoning rules to Answer WM. We adopt the two WMs for bi-directional search, which enables efficient search of the path. To find the shortest path, a rule is selected in breadth first order by conflict resolution after each rule application. The WMs employ *UNIFORM* (UNIfied FORM) [14, 15], which packs results of conflicting rule applications into one representation, and reduces memory and time space to be used. Finally, the learning module analyzes dependencies among the rule applications, and represents them as an explanation.

4.2 Learning Algorithm

Our algorithm makes a decomposition of a positive instance by dividing its explanation into *subgraphs*, and making a composition of each subgraph. To determine a decomposition that does not derive any negative instance, the algorithm compares each subgraph with explanations of negative instances. We use the following definition of equality for the comparison:

Definition 4 (equality of subgraphs) *Subgraphs X and Y are equal ($X = Y$), if they satisfy the following conditions:*

1. *They are isomorphic.*
2. *Corresponding nodes have the same rule name, and corresponding arcs have the same* position *tag.*

A node which is a composition of a subgraph is compared after it is replaced by the original subgraph.

$n \in G$ denotes that the node n is in the graph G, and $X \subseteq Y$ denotes that X is a subgraph of Y. First, choose a node out of an explanation of each negative instance, and term it H_{Nj}, such that G_s shown below becomes smaller to generate more generalized rules. Then, divide an explanation of each positive instance into subgraphs G_s that satisfy the following condition:

$$G_s \neq G$$

where,

$$H_{Nj} \in G \subseteq G_{Nj}.$$

The composition of rules in each G_s gives a generalization which derives positive instances but not negative instances.

The algorithm is as follows:

Algorithm 1

begin
Explain positive instances $P_i (i = 1, ..., m)$ and negative instances $N_j (j = 1, ..., n)$, and term these explanations G_{Pi} and G_{Nj}, respectively;
For *each j,* **do** *select a minimum subgraph g_{Nj} that satisfies:*

$$(g_{Nj} \subseteq G_{Nj}) \wedge \forall i (g_{Nj} \not\subseteq G_{pi})$$

Choose a node out of the nodes in g_{Nj}, and term it H_{Nj};
Initialize the rule set to empty;
For *each node n in each G_{Pi}* **do begin**
Select a subgraph G_s which satisfies:

$$(n \in G_s) \wedge \forall j ((H_{Nj} \in G \subseteq G_{Nj}) \wedge (G_s \neq G))$$

Add a composition of rules in G_s to the rule set;
Replace subgraph G_s by a node
end
end.

Although this algorithm is nondeterministic at some steps, a correct generalization is obtained by whatever choices occur at those steps.

Instead of negative instances, a decomposition may be determined based on incorrect firings of rules, i.e. firings that are not on the shortest path to solve each problem:

Algorithm 2

begin

Explain positive instances $P_i (i = 1, ..., m)$, *and term these explanations* G_{Pi};

Let G_{Ni} *be a concatenation of* G_{Pi} *and incorrect firings;*

Let H_{Nj} *be the first firing of each sequence of incorrect firings;*

Initialize the rule set to empty;

For *each node n in each* G_{Pi} **do begin**

Select a subgraph G_s *which satisfies:*

$$(n \in G_s) \wedge$$

$$\forall i, j(\neg((H_{Nj} \in G \subseteq G_{Ni}) \wedge (G_s = G)))$$

Add a composition of rules in G_s *to the rule set;*

Replace subgraph G_s *by a node*

end

end.

Since a sequence of incorrect firings may be infinite, NAT traces incorrect firings one by one when they are required.

Algorithm 1 requires negative instances and works even if there are more than one answers to a problem. In contrast, Algorithm 2 requires no negative instances by assuming different answers are incorrect.

5 Deductive and Inductive Aspects of KL-EBL

KL-EBL acquires knowledge that is deduced from domain theory and induced from instances. It has been applied to some domains by giving an appropriate domain theory, such as logical circuit design, analogy intelligence tests, translation of natural language [17], program synthesis [18, 16] and translation of programs [14]. In this paper, we will demonstrate the first two examples to discuss deductive and inductive aspects of KL-EBL.

5.1 Deductive Aspect

As an example of deductive learning, we demonstrate learning of logic circuit design. The system learns generalized design rules from design examples as shown in Figure 2.

Domain theory is as follows, in which a rule with $\Rightarrow$ instead of $\rightarrow$ denotes a forward-reasoning rule, and a rule with $\Leftarrow$ denotes a backward-reasoning rule:

1. *Gate specification:* Rules whose RHS and LHS are a gate and its specification, respectively.

Specification:

OUTPUT = (and (or INPUT1 INPUT2) (or INPUT3 INPUT4))

Designed Circuit:

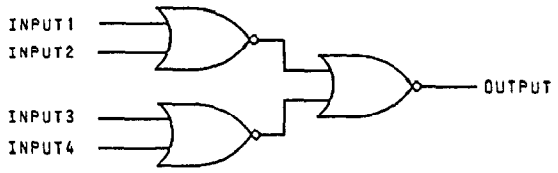

Figure 2: An Example of Circuit Design

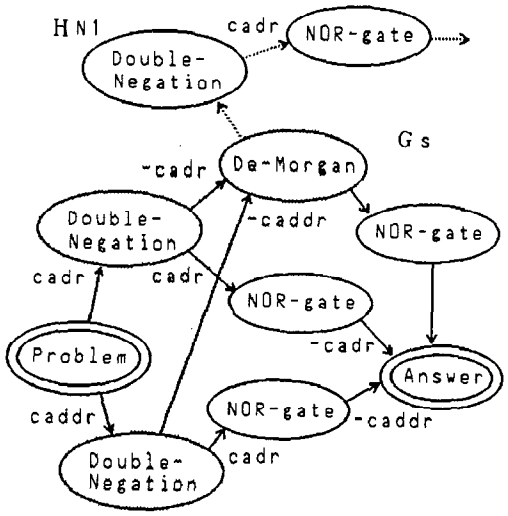

Figure 3: An Explanation of Circuit Design

*NOR-gate: (not (or *x *y)) $\Leftarrow$ (NOR *x *y).*

2. *Logical reduction rules:* De Morgan's law and a rule for removing a double negation.

De-Morgan:
*(and (not *x) (not *y)) $\Leftarrow$ (not (or *x *y)).*
*Double-Negation: *x $\Leftarrow$ (not (not *x)).*

Since the above domain theory is true with regard to logic circuits, the learning process is deductive, i.e., is truth-preserving.

The positive instance shown in Figure 2 is as follows:

instance1: (and (or INPUT1 INPUT2) (or INPUT3 INPUT4))
$\rightarrow$ (NOR (NOR INPUT1 INPUT2) (NOR INPUT3 INPUT4)).

Figure 3 shows an explanation of the above instance, where a sequence of incorrect firings is shown in dotted lines.

Let us apply Algorithm 2 to this example. Since H_{Nj} is the first firing of each sequence of incorrect firings, H_{N1} =*Double-Negation* as indicated in the figure. Thus, for a node n =*Double-Negation*, G_s = {*Double-Negation, De-Morgan*}. Furthermore, for another *Double-Negation* in the graph, G_s = {*Double-Negation, Double-Negation, De-Morgan*} as marked in the figure. For n =*NOR-gate*, G_s = {*NOR-gate*}.

Thus, the learned rules are as follows:

1. A composition of two *Double-Negations* and *De-Morgan*:

   Double-NegationDouble-NegationDe-Morgan:
   *(and *x1 *y) ⇒ (not (or (not *x1) (not *y))).*

2. A forward reasoning version of *NOR-gate*:

   forward-NOR-gate:
   *(not (or *x *y)) ⇒ (NOR *x *y).*

These rules derive the correct circuit from the given specification, i.e., the system learns that a specification involving *and* has to be translated into that involving *not*. This reflects a design principle that NOR and NAND gates should be preferred to AND or OR gates. The system learns no new knowledge on logic circuits, since the domain theory is already true. However, the domain theory involves no design principle, which is acquired as a composition.

In *verification-based learning* [7], which is a kind of of Explanation-Based Learning, the system first verifies, i.e., explains a given circuit, and acquires a generalization based on the verification by using constraint propagation, which amounts to making a composition of the explanation. A composed subgraph G_s is determined such that it involves only the logical reduction rules. In contrast, our method determines G_s referring to incorrect firings, i.e., since the gate specification rules are not incorrectly fired, they are separated from the composition of logical reduction rules.

5.2 Inductive Aspect

To show an inductive aspect of KL-EBL, we have a demonstration of analogical inference. Analogical inference is inference based on an analogy between a target and a base. In the demonstration, the system explains an analogy between the target and the base to find an appropriate relation. Then, by using Algorithm 2, it extracts rules, which perform analogical inference based on the analogy.

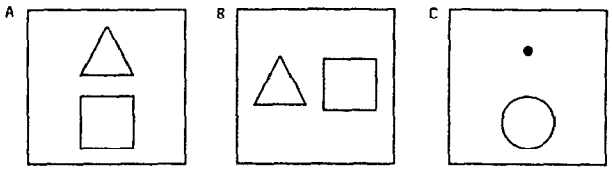

Figure 4: An Analogy Intelligence Test

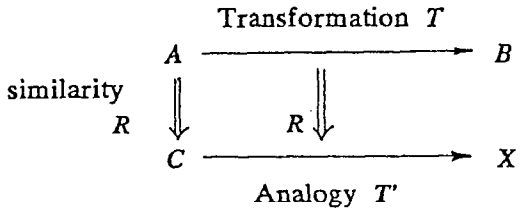

Figure 5: Analogical Inference

A geometric analogy problem as shown in Figure 4 typically appears in human intelligence tests [24, 23]. The problem is to infer an appropriate figure, X, such that A is to B as C is to X. In other words, we want to find the rule describing how C becomes some X, which most closely matches the rule describing how A becomes B.

Figure 5 illustrates the inference. The analogy T' is obtained from the transformation T and the similarity between A and C. Therefore, by making a decomposition of a rule R_{init}: $A \rightarrow C$, the system finds a rule set denoting the similarity R, and, by using this, infers an analogy T'. The representation of Figure 4 is as follows:

A: *(:relation: above (:object: triangle) (:object: square)).*
B: *(:relation: left (:object: triangle) (:object: square)).*
C: *(:relation: above (:object: dot) (:object: circle)).*

The rule R_{init} is:

R_{init}: *(AB (:relation: above (:object: triangle) (:object: square)))*
→ *(CX (:relation: above (:object: dot) (:object: circle))),*

where AB and CX are labels to distinguish A from C. A decomposition of R_{init} gives similarity R.

The domain theory contains three kinds of rules. We attach *cost* to each rule, by which Nat finds the minimum cost path representing the closest match of two

figures. The following are the *equivalence transformation rules*:

encode-relation: *(AB (:relation: *relation *x *y))*
⇒ *(:relation: *relation (AB *x) (AB *y))*
cost: 1.

encode-object: *(AB (:object: *object))*
⇒ *((:object: *object))*
cost: 1.

decode-relation: *(:relation: *relation (CX *x) (CX *y))*
⇐ *(CX (:relation: *relation *x *y))*
cost: 1.

decode-object: *((:object: *object))*
⇐ *(CX (:object: *object))*
cost: 1.

If both sides of R_{init} are the same, the explanation contains only these rules.

Other two kinds of rules, *abstraction* and *instantiation* rules, are the hypothesis generators, which are used to find the correspondence between A and C. The abstraction rules abstract the LHS of R_{init}. A triangle is abstracted to an arbitrary object by using the following rule:

abstract-triangle:
(AB (:object: triangle)) ⇒ *((:object: arbitrary))*
cost: 100.

Conversely, the next rule instantiates the object:

inst-triangle:
((:object: arbitrary)) ⇐ *(CX (:object: triangle))*
cost: 100.

We call it an *instantiation rule*, since it would instantiate the triangle to the abstracted object if it reasoned forward, although, in reality, this backward-reasoning rule abstracts the RHS of R_{init}.

The system finds the similarity or the correspondence between A and C by transforming A into C by using these rules, and constructs an explanation as shown in Figure 6, which describes the minimum cost path of transformation. Since the application cost of the equivalence rule is low and that of the abstraction and instantiation rule is high, the minimum cost path gives the maximal similarity. According to the explanation, the similarity is detected by pairing the triangle with the dot, and the rectangle with the circle, by assuming them as abstracted objects.

Some incorrect firings are shown by the dotted line in Figure 6(b). The result of decomposition is:

encode-relation: *(AB (:relation: *relation *x *y))*
⇒ *(:relation: *relation (AB *x) (AB *y)).*

Forward-decode-relation:
*(:relation: *relation (CX *x) (CX *y))*
⇒ *(CX (:relation: *relation *x *y)).*

abstract-triangle: *(AB (:object: triangle))*
⇒ *((:object: arbitrary)).*

abstract-square: *(AB (:object: square))*
⇒ *((:object: arbitrary)).*

triangle-dot:
(AB (:object: triangle)) ⇒ *(CX (:object: dot)).*
(The composition of *abstract-triangle* and *inst-dot*.)

square-circle:
(AB (:object: square)) ⇒ *(CX (:object: circle)).*
(The composition of *abstract-square* and *inst-circle*.)

These rules preserve the relation between objects, and transform the triangle into the dot, and the square into the circle.

By applying the above rules to the transformation T,

T: *(AB (:relation: above (:object: triangle) (:object: square)))*
→ *(AB (:relation: left (:object: triangle) (:object: square))),*

the analogy T' is obtained as follows:

T': *(CX (:relation: above (:object: dot) (:object: circle)))*
→ *(CX (:relation: left (:object: dot) (:object: circle))).*

Notice that, our method newly constructs the answer, although Evan's geometric analogy program [24] selects the answer from given choices.

6 Conclusion

We have analyzed EBL at the knowledge level, presented a generalization method by decomposition of rules, and demonstrated its advantages by using some learning examples.

In EGGS [12] and verification-based learning [7, 10], the system explains a given instance by using backward reasoning rules, and makes a composition as a forward reasoning rule. This means that its learning ability is limited to instances explained by the backward reasoning rules, and that it cannot use rules that delete terms. In contrast, by employing bi-directional search, our method expands the learning ability.

In addition, EGGS and the verification-based learner usually make a composition of the entire explanation,

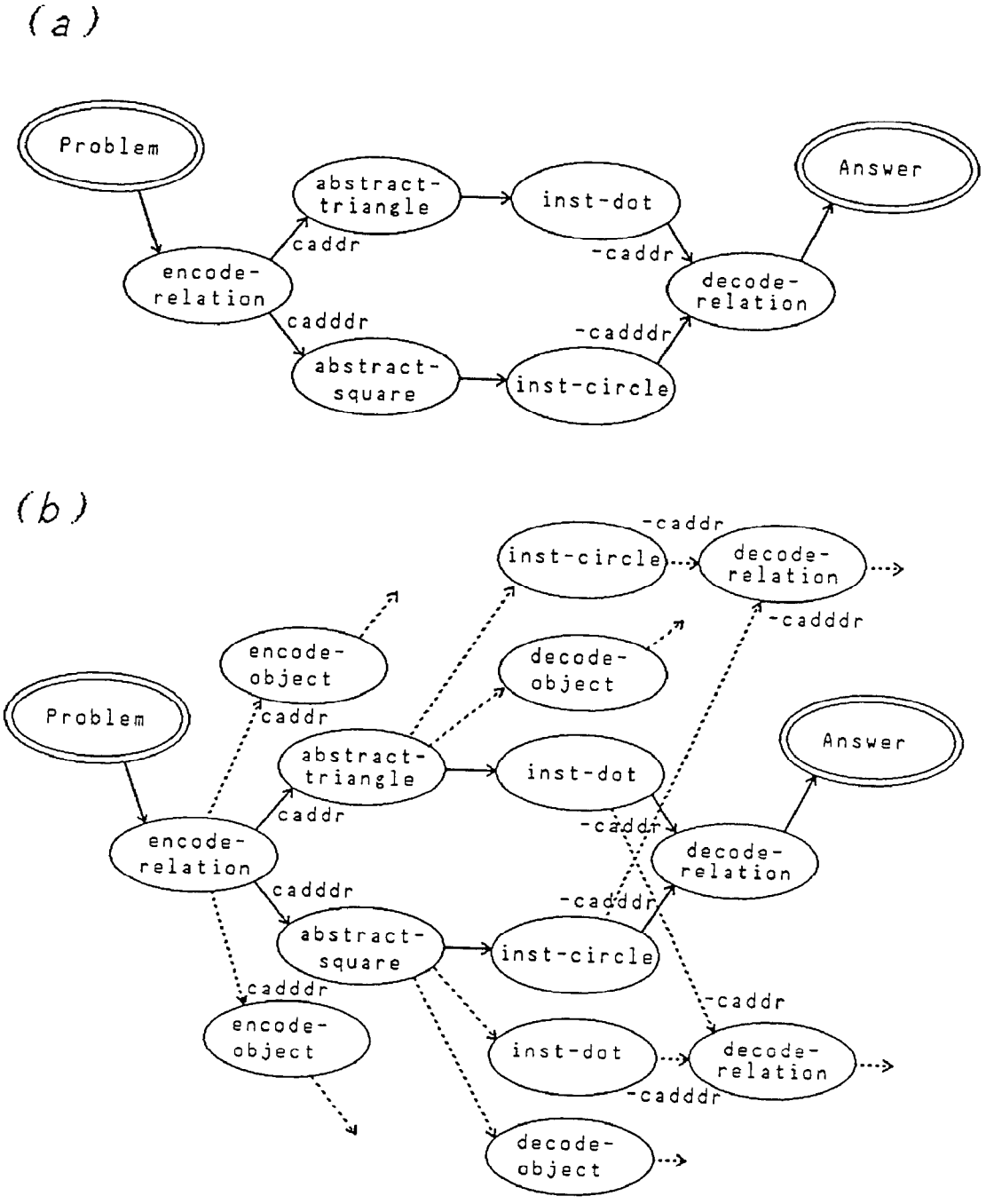

Figure 6: (a) An explanation of R_{init}; (b) that with incorrect firings.

which sometimes derives overly specific rules. In such a case, they make a composition of only a part of explanation determined in an *ad hoc* manner. As described in section 5.1, the verification-based learner makes a composition of all the rules in the explanation, except for gate specification rules. In the GENESIS example [12], any facts supporting only hierarchical class inferences (ISA inferences) are pruned to arrive at an appropriate generalization. Such a method depends on each situation, whereas our method is based on either incorrect firings or negative instances. For example, rules supporting only ISA inferences are not included in the composition, since hierarchical class inferences are always deterministic, and never become a head H_{Nj} of incorrect firings.

In PRODIGY [9], composed subgraphs are selected based on their *utility* determined by problem solving performance. In contrast, we give a method to extract subgraphs according to positive and negative instances.

According to Mitchell, Keller and Kedar-Cabelli [11], our method is categorized as *combining explanation-based and similarity-based methods*, i.e., explanations of positive instances are used for generalization, and those of negative instances for specialization. This is useful when only incomplete theories are available for generalization. Even if only one instance is available, our method is still beneficial for accurate generalization. Some results have already been achieved in combining explanation-based and similarity-based methods, including those of Lebowitz [6] who explores such a method in his UNIMEM system. His system first searches for empirical similarity-based generalizations, and then attempts to verify these empirical generalizations by their explanations. Conversely, our method first constructs explanations of instances, and then finds a similarity between explanations. Kedar-Cabelli [4] also proposes a method that makes use of multiple explanations in her Purpose-Directed Analogy, though her method does not employ negative instances or incorrect firings. DeJong and Mooney [2] call such an inter-explanation method *explanation-based refinement.*

From the viewpoint of inductive inference, our method is assumed as a procedure to search a space of possible rules until one is found that agrees with all the data so far. However, how to search the space is different from the usual *identification by enumeration*, such as Shapiro's MIS [21]. Our mechanism constructs a rule set from explanations of instances rather than finds a compatible rule set from the space. This means that it can easily learn from the small number of instances, and that it is difficult to learn by using too weak domain theory, though this problem is common in Explanation-Based Learning.

Acknowledgment

Niall Murtagh and Blair Belton have assisted editing an early draft of this paper.

References

[1] A. Barr, P.R. Cohen, and E.A. Feigenbaum, editors. *The Handbook of Artificial Intelligence, Vol.1.* William Kaufmann, Inc., Los Altos, CA, 1982.

[2] G. DeJong and R. Mooney. Explanation-based learning: An alternative view. *Machine Learning*, 1:145–176, 1986.

[3] T. G. Dietterich. Learning at the knowledge level. *Machine Learning*, 1:287–316, 1986.

[4] S. T. Kedar-Cabelli. Purpose-directed analogy. In *Proceedings of the Cognitive Science Society Conference*, pages 150–159, Irvine, CA, 1985.

[5] R.M. Keller. Defining operationality for explanation-based learning. *Artificial Intelligence*, 35:227–241, 1988.

[6] M. Lebowitz. Concept learning in a rich input domain: Generalization-based memory. *in ref. [8]*, pages 193–214, 1986.

[7] S. Mahadevan. Verification-based learning: A generalization strategy for inferring problem-reduction methods. In *IJCAI85*, pages 616–623, Los Angeles, CA, 1985.

[8] R.S. Michalski, J.G. Carbonell, and T.M. Mitchell, editors. *Machine learning: An artificial intelligence approach (Vol. 2).* Morgan Kaufmann, Los Altos, CA, 1986.

[9] S. Minton. Quantitative results concerning the utility of explanation-based learning. In *AAAI88*, pages 564–569, 1988.

[10] T. M. Mitchell, S. Mahadevan, and L. I. Steinberg. LEAP: A learning apprentice for VLSI design. In *IJCAI85*, pages 573–580, Los Angeles, CA, 1985.

[11] T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1:47–80, 1986.

[12] R. J. Mooney and S. W. Bennett. A domain independent explanation-based generalizer. In *AAAI86*, pages 551–555, Philadelphia, PA, 1986.

[13] A. Newell. The knowledge level. *AI Magazine*, 2:1–20, 1981.

[14] M. Numao and M. Shimura. Construction of a program translation system by learning (in Japanese). *Information Processing Society of Japan*, 86-AI-40, 1985.

[15] M. Numao and M. Shimura. Nat: Nonalgorithmic translator (in Japanese). *Information Processing Society of Japan*, 86-SYM-38, 1986.

[16] M. Numao and M. Shimura. Explanation-based acceleration of similarity-based learning. In *Proceedings of the Sixth International Workshop on Machine Learning*, pages 58–60, Cornell University, Ithaca, NY, 1989. Morgan Kaufmann.

[17] M. Numao and M. Shimura. A learning method based on partial structures of explanations (in Japanese). *The Transactions of the Institute of Electronics, Information and Communication Engineers*, J72-D-II(2):263–270, 1989.

[18] M. Numao and M. Shimura. Inductive program synthesis by using a reversible meta-interpreter. In *Proceedings of the Workshop on Meta-Programming in Logic*, Leuven, Belgium, 1990. to appear.

[19] P. S. Rosenbloom, J. E. Laird, and A. Newell. Knowledge level learning in Soar. In *AAAI87*, pages 499–504, 1987.

[20] A.M. Segre, editor. *Proceedings of the Sixth International Workshop on Machine Learning.* Morgan Kaufmann, San Mateo, CA, 1989.

[21] E.Y. Shapiro. *Algorithmic Program Debugging.* The MIT Press, Cambridge, MA, 1982.

[22] F. van Harmelen and A. Bundy. Explanation-based generalization = partial evaluation. *Artificial Intelligence*, 36:401–412, 1988.

[23] S.A. Vere. Induction of relational productions in the presence of background information. In *IJCAI77*, pages 349–355, 1977.

[24] P.H. Winston. *Artificial intelligence (second edition).* Addison Wesley, Reading, MA, 1984.