MCC Technical Report Number ACA-AI/CAD-038-88

Formulating and Retrieving Knowledge through Abstraction Extensions to Explanation-Based Learning

Michael N. Huhns and Ramón D. Acosta

January 1988

Microelectronics and Computer Technology Corporation
Advanced Computer Architecture and VLSI CAD Programs
3500 West Balcones Center Drive
Austin, TX 78759
(512) 343-0978

1 Introduction and Background

Explanation-based learning (EBL) [4,11] is a knowledge-intensive analytic technique by which learning systems can capture and generalize problem-solving experience from single training examples. Unfortunately, EBL generalizations are limited in that they arbitrarily give equal weight to all portions of the examples, without regard to whether each portion is relevant or important to solving future problems. To overcome this limitation, several extensions to EBL have been incorporated into Argo, a tool for building knowledge-based systems that reason analogically and nonmonotonically and improve with use. Argo extends EBL by 1) learning several new rules, at different levels of abstraction, from each training example, 2) using rules at these different levels of abstraction to solve new problems that are not necessarily identical, but just analogous to those it has solved previously, and 3) providing an abstraction-based strategy for efficiently retrieving acquired knowledge.

Argo transfers experience from previous problem-solving efforts to new problems via analogical reasoning methods. An analogy is a mapping from a base domain to a target domain that allows the sharing of features between these domains. We classify analogies as being either *exact* or *inexact*. Where there is an exact match between a past experience and a new problem-solving situation, an exact analogy exists and the new problem can be solved by reexecuting the solution plan from the past experience. Where an exact match does not exist, the two problems that arise are 1) *analogy recognition*: finding the most similar past experience, and 2) *analogical transformation*: adapting this experience to the new problem situation.

Several techniques have been suggested for automatically recognizing the most similar past experience. These include finding a past experience with either an identical first stage [3], the same causal connections among its components [5,16,17], or the same purpose as the new problem-solving situation [9]. The second problem, the adaptation of old experiences to new problem situations, has been attempted previously by employing heuristically-guided incremental perturbations according to primitive transformation steps [2], heuristic-based analogical inference [6], and user interaction [13].

A fundamental hypothesis of our work is that inexact analogies at one level of abstraction become exact analogies at a higher level of abstraction. Thus, we have developed techniques within Argo for automatically computing and storing increasingly abstract versions of plans and subsequently employing them in solving new problems. The use of plan abstractions provides Argo with an effective means for analogy transformation and recognition, and enables it to improve its problem-solving performance as it is used.

2 Problem-Solving in Argo

Argo is a generic development environment for the use of analogical reasoning and learning in solving problems, particularly in search-intensive domains such as design [1,7,8]. Derived from the Proteus expert system tool [14], it represents knowledge using a combination of predicate logic and frames within a justification-based truth-maintenance system. Argo's primary inference mechanisms are forward chaining, backward chaining, multiple inheritance through the frame system, truth maintenance, and contradiction resolution.

Argo executes a problem-solving strategy with two major phases: a problem-solving phase followed by a learning phase. The control strategy for the problem-solving phase is that of a standard production-system interpreter, modified for analogical reasoning by requiring that only the most specific rules from a partial order of forward rules (based on the *abstraction* relation defined below) be matched and considered for execution.

3 Learning in Argo

Learning typically occurs after problem-solving has been successful. This prevents the learning of results that might be subsequently invalidated due to nonmonotonic reasoning triggered by dependency-directed backtracking. Thus, plans that are learned do not incorporate failed lines of reasoning [10].

A problem-solving plan in Argo is the explanation for a training example. It is represented by a rule-dependency graph: a directed acyclic graph having nodes corresponding to forward rules and edges indicating deductive dependencies between the rules of the plan. Argo implements a type of derivational analogy [3,12] to solve new problems by making use of abstracted rule-dependency graphs from previous problem-solving experiences. In this vein, the primary function of the system's learning phase is to compute and store abstractions for the plan of a solved problem.

A number of domain-dependent and domain-independent techniques for automatically generating plan abstractions are possible. These include deleting rules from a plan, replacing a rule by a more general rule that refers to fewer details of a problem (as in ABSTRIPS [15]), and generalizing a macrorule for the plan without reference to the plan itself. Argo abstracts a plan by deleting all of its leaf rules, which are those having no outgoing dependency edges. For many domains, the leaf rules trimmed from a plan tend to be those that deal with details at the plan's level of abstraction. Increasingly abstract versions of a plan are obtained by iteratively trimming it until either one or zero nodes remain.

One possible drawback of Argo's automatic abstraction scheme is that deleting all leaf rules might eliminate potentially useful abstract plans in which only some of the leaf rules should be deleted. Except for the very smallest plans, however, it is clearly not practicable to generate macrorules for all possible subgraphs of the rule-dependency graph. Also, the system can always start with a plan's previously computed abstraction, followed by instantiations of some of the trimmed rules, to obtain the appropriate "abstraction" required to solve a new problem.

Argo computes abstractions during its learning phase—after a problem is solved. In contrast, it is possible to save a plan and only compute abstractions when necessary, *i.e.*, when solving new problems in which an abstract version of an original plan is applicable. There are difficulties with using this approach, including identification of the most suitable previous plan using some type of partial match procedure and analogical transformation of the selected plan based upon the partial match results. Consequently, Argo uses an *a priori* approach to generating abstractions.

During this learning phase, the plan or abstract plans for a solved problem are not explicitly learned by the system. Instead, an explanation-based scheme is used to regress through the component rules of each plan, resulting in a set of *macrorules* that embody the relevant preconditions and postconditions of the plan. Applying a macrorule yields the same result as executing a plan; in addition, a macrorule is more efficient and eliminates the correspondence problem for plans described in [13]. Further, it is difficult to index plans so that their appropriateness can be determined, whereas macrorules can be integrated with the original rules in the system in such a way that they are applied always and only when appropriate.

Argo's use of rule-dependency graphs to represent plans contrasts with the explanation-based learning mechanism in [11], in which explanations consist of proof trees having edges between individual antecedents and consequents of dependent rules. While only one macrorule is computed for the technique presented in [11], Argo computes a set of one or more macrorules for a given explanation. Although harder to compute, these macrorules can be applied to situations differing structurally from the original problem.

The macrorules are organized into a partial order based on a relation called *abstraction*. A plan P_i is a mapping from a domain D_i, determined by the antecedents of the macrorule for P_i, to a range R_i, determined by the consequents of the macrorule for P_i. Intuitively, one plan is more abstract than another if it applies to more situations and if its execution results in fewer commitments. More precisely,

$$P_i \sqsupset P_j \Leftrightarrow (D_i \succ D_j) \wedge (R_i \succ R_j)$$

where $\sqsupset$, the *abstraction* relation, is to be read "is an abstraction of," and where

Definition 1 $S_i \succ S_j \Leftrightarrow$ *the set of possible worlds in which* S_j *is true is a subset of the set of possible worlds in which* S_i *is true.*

Abstraction is a transitive, reflexive, and antisymmetric relation: it thus induces a partial order on a set of rules.

This partial order facilitates locating and applying previous problem-solving experience. If a problem is given to the system that is exactly analogous to an old problem, then the most specific macrorule is applied to completely solve it. Alternatively, if the new problem is inexactly analogous to an old one, the system follows specialization paths in the partial order of forward rules in order to choose the least abstract macrorule that is applicable, *i.e.*, one that instantiates the largest number of details without making incorrect inferences. By successively selecting the least abstract rules, the system typically finds the shortest path to a valid solution.

4 Conclusions

The work reported here is based on developing the fundamental methodology for a system, Argo [7], that reasons and learns by analogy for solving problems in design. This methodology includes the use of problem-solving plans to effect the analogical transfer of knowledge from a base problem to a target problem, the use of abstract plans to allow the transfer of experience to inexactly analogous target problems, an algorithm for calculating macrorules for a plan that allows the plan to be retrieved and applied efficiently, and the formal definition of an abstraction relation for partially ordering plans.

Design is a knowledge-intensive problem-solving activity characterized by a very large, incompletely-specified search space with many alternative solutions and no metric. Optimal design is thus typically intractable. The learning techniques in Argo are useful for such an intractable domain because they dramatically reduce the size of the search space and obviate the need for exhaustive search. Traditional EBL techniques presume that the same problems will be encountered again—an unlikely possibility in such a large space. Argo's abstract macrorules apply to general problem classes and are much more likely to be reused.

In order to evaluate Argo, a knowledge-based system has been implemented for designing VLSI digital circuits. This system refines behavioral descriptions for circuits into structural descriptions by executing plans composed of transformation, instantiation, and decomposition rules. The system is typically used as follows: a designer trains it on a set of representative examples by choosing solution paths and controlling backtracking, thereby producing plans for achieving correct designs. Argo compiles these plans, at various levels of abstraction, into a set of macrorules and maintains these macrorules in its justification network. A knowledge-based contradiction-resolution mechanism revises and updates this network, yielding nonmonotonic learning. Given a behavioral description for a new design, Argo applies the macrorules that require the fewest additional details to complete the design, thus reducing the computation required to solve the problem. We have demonstrated that with use, the system accumulates design knowledge that produces better quality designs more efficiently [7].

Acknowledgement

We would like to thank Tom M. Mitchell for supplying many of the ideas and insights upon which this research is based.

References

[1] R. D. Acosta, M. N. Huhns, and S. Liuh, "Analogical Reasoning for Digital System Synthesis," *Proceedings of the IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, November 1986, pp. 173–176.

[2] J. G. Carbonell, "Learning by Analogy: Formulating and Generalizing Plans from Past Experience," in *Machine Learning, An Artificial Intelligence Approach, Vol. I*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Tioga Press, Palo Alto, CA, 1983, pp. 137–161.

[3] J. G. Carbonell, "Derivational Analogy: A Theory of Reconstructive Problem Solving and Expertise Acquisition," in *Machine Learning: An Artificial Intelligence Approach, Vol. II*, R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, eds., Morgan Kaufmann, Los Altos, CA, 1986, pp. 371–392.

[4] G. DeJong and R. Mooney, "Explanation-Based Learning: An Alternative View," *Machine Learning*, vol. 1, no. 2, 1986, pp. 145–176.

[5] D. Gentner, "Structure Mapping: A Theoretical Framework for Analogy," *Cognitive Science*, vol. 7, no. 2, April 1983, pp. 155–170.

[6] R. Greiner, *Learning by Understanding Analogies*, Ph.D. Dissertation, Stanford University, Technical Report STAN-CS-1071, Palo Alto, CA, September 1985.

[7] M. N. Huhns and R. D. Acosta, "Argo: An Analogical Reasoning System for Solving Design Problems," MCC Technical Report No. AI/CAD-092-87, Microelectronics and Computer Technology Corporation, Austin, TX, April 1987.

[8] M. N. Huhns and R. D. Acosta, "Argo: A System for Design by Analogy," *Proceedings of the Fourth IEEE Conference on Artificial Intelligence Applications*, San Diego, CA, March 1988.

[9] S. T. Kedar-Cabelli, "Formulating Concepts According to Purpose," *Proceedings of the Sixth National Conference on Artificial Intelligence*, Seattle, WA, July 1987, pp. 477–481.

[10] S. Liuh and M. N. Huhns, "Using a TMS for EBG," MCC Technical Report No. AI-445-86, Microelectronics and Computer Technology Corporation, Austin, TX, December 1986.

[11] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli, "Explanation-Based Generalization: A Unifying View," *Machine Learning*, vol. 1, no. 1, 1986, pp. 47–80.

[12] J. Mostow, "Automated Replay of Design Plans: Some Issues in Derivational Analogy," submitted to *Artificial Intelligence Journal*, March 1987.

[13] J. Mostow and M. Barley, "Automated Reuse of Design Plans," *Proceedings of the International Conference on Engineering Design*, Boston, MA, August 1987.

[14] C. J. Petrie, D. M. Russinoff, and D. D. Steiner, "PROTEUS: A Default Reasoning Perspective," *Proceedings of the 5th Generation Computer Conference*, National Institute for Software, Washington, D.C., October 1986.

[15] E. D. Sacerdoti, "Planning in a Hierarchy of Abstraction Spaces," *Artificial Intelligence*, vol. 5, no. 2, 1974, pp. 115–135.

[16] P. H. Winston, "Learning New Principles from Precedents and Exercises," *Artificial Intelligence*, vol. 19, no. 3, November 1982, pp. 321–350.

[17] P. H. Winston, "Learning by Augmenting Rules and Accumulating Censors," in *Machine Learning, An Artificial Intelligence Approach, Vol. II*, Morgan Kaufman, Los Altos, CA, 1985, pp. 45–61.