Design by Analogy Using Plan Abstractions

Ramón D. Acosta and Michael N. Huhns

Microelectronics and Computer Technology Corporation
VLSI CAD and Advanced Computer Architecture Programs
3500 West Balcones Center Drive
Austin, TX 78759
(512) 338-3673 or acosta@mcc.com

Abstract

An important problem-solving strategy used in many design domains is to build upon the experience of previous design efforts as an aid to solving new design problems. This can be viewed as problem solving by analogy. Two concerns for constructing systems capable of analogical reasoning are analogy recognition (finding the most relevant past experiences) and analogy transformation (adapting previously acquired knowledge to new problem-solving situations). This paper explores how the use of abstraction, particularly as related to learning problem-solving plans, can serve as a useful foundation for the use of analogy in design domains.

1 Introduction

Analogical reasoning is an important problem-solving strategy being actively investigated by a number researchers [2,3,5,6,9,15]. The establishment of analogies as a problem-solving aid is characterized by a variety of useful features, including

- a smaller search space
- a more sophisticated reuse of learned knowledge (in contrast to simple playback of problem-solving episodes)
- the application of experience via causal connections from "more-familiar" previously solved problems to "less-familiar" new problems

© AAAI-88 Workshop on AI and Design

Design, being a knowledge-intensive and search-intensive activity, is a domain for which the reusability afforded by using analogies might prove to be particularly beneficial [11,13]. For example, because design is often hierarchical, transformations and decompositions that can be applied to functional specifications typically lead to increasingly simpler, and often independent, subproblems. Further, even if a design system lacks the previous design knowledge to solve an entire new problem, analogically transforming partial design plans previously generated by the system can be an effective approach to yielding a complete design.

A vital issue in building design systems capable of exploiting analogies is the representation of design plans from which analogies are to be drawn. This representation not only must be expressive enough to explicitly represent design decisions, but also must be storable and retrievable in order to facilitate the recognition and transference of relevant analogies to future problems. Our belief is that abstractions of plan structures are the key to the successful use of analogy in design domains.

This paper explores various issues related to the reuse of learned knowledge based on analogies with previous problem-solving efforts. Section 2 defines some important concepts, including exact versus inexact analogies, analogy recognition, and analogy transformation. Sections 3 and 4 concentrate on the representation and abstraction of design plans for the purposes of analogical reasoning. Conclusions are presented in Section 5. An indication of how these ideas have been implemented in the Argo system [1,7,8] is presented throughout the paper.

2 Analogy Recognition and Transformation

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 either by executing the old plan or by using the old solution. Where there is not an exact match, an inexact analogy exists and the two tasks that arise are 1) *analogy recognition*: finding the most similar past experience, and 2) *analogy 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 [5,15], or the same purpose as the new problem-solving situation [9]. The second task, 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 intervention [12].

Our approach to these tasks is based on the following fundamental hypothesis: inexact analogies at one level of abstraction become exact analogies at a higher level of abstraction. Commitment to this hypothesis suggests the need to develop plan representations that lend themselves to automatic storage, abstraction, and retrieval. We have developed these representations and incorporated them into Argo, a tool for building knowledge-based systems that integrates techniques for reasoning and learning by analogy to aid in solving search-intensive problems, such as those in design domains.

3 Design Plans

Analogical reasoning systems can employ a variety of techniques for solving problems. Argo employs the following control strategy for solving a design problem P, based on ordering these techniques according to the amount and specificity of domain knowledge they require [3]:

1. If knowledge of an artifact that satisfies P is available, then this solution is directly instantiated.

2. If a plan for solving a problem that is exactly analogous to P is available, then it is directly executed.

3. If a plan for solving P' is available, where P' is "similar" (*i.e.*, inexactly analogous) to P, then it is analogically transformed to synthesize an artifact for P.

4. If past experience is unavailable, then weak methods such as heuristic search or means-ends analysis are employed.

In addition to the planning requirements imposed by analogical reasoning in step 3, design systems must employ hierarchical plan representations that are easily constructed and executed. Consequently, an attractive representation for a plan is a tree that depicts the goal structure or goal-operator structure involved in design construction [13]. A goal structure representation has a number of advantages, such as the explicitness with which design decisions are recorded, and the corresponding ease of replaying the design process. Note that a goal structure is not necessarily isomorphic to a design decomposition tree.

Argo acquires problem-solving experience in the form of problem-solving plans represented by rule-dependency graphs (RDGs). An RDG is a directed acyclic graph having nodes corresponding to forward rules and edges indicating deductive dependencies between the rules. Thus, we generalize the notion of a goal-operator tree in order to account for interaction among subgoals.

4 Learning Plan Abstractions

A number of domain-dependent and domain-independent techniques for automatically generating plan abstractions are possible. For a given plan represented by an RDG, these techniques include:

1. Deleting a rule having no outgoing edges, *i.e.*, one upon which no other rule in the plan is dependent. For design domains, such rules typically instantiate details; it seems plausible that deleting these rules will yield plan abstractions because the resultant plans will make fewer commitments to implementation details.

2. Replacing a rule by a more general rule that refers to fewer details of a problem. A more general rule might be one having fewer antecedents or consequents, fewer constants, more variables, more general domain constants, etc. As achieved in ABSTRIPS [14], these rules can be generated from the initial domain knowledge using criticality measures.

3. Replacing a sequence of rules or a subplan by a single, more general rule.

4. Generalizing a computed macrorule (see below) for the plan without reference to the components of the original plan.

Some of the questions that must be carefully considered in choosing an appropriate abstraction scheme include the following:

1. How independent is the technique from the application domain?

2. Are the abstractions generated within the deductive closure of the system?

3. How automatic is the technique?

4. How easy is it to implement?

5. How useful are the abstractions for solving analogous problems?

In keeping with these guidelines, the abstraction scheme currently employed in Argo is a variation of the first option listed above. It involves automatically formulating a plan abstraction by deleting all of its leaf rules, which are those having no outgoing dependency edges. For many design domains, the leaf rules trimmed from a plan tend to be those that deal with design details at the plan's level of abstraction. In effect, deletion of leaf rules generates a reusable *plan kernel* at a given level of abstraction. Thus, 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 part of the leaf rules should be deleted. Except for the very smallest plans, however, it is clearly not practicable to automatically capture abstractions for all possible subgraphs of the RDG, although these would be valid and potentially useful. In addition, although additional forward chaining might be required, it is always possible for the system to start with a previously computed abstract plan kernel, followed by instantiations of the relevant 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.

From increasingly abstract plan kernels, Argo uses an explanation-based mechanism [4,10] to calculate sets of *macrorules*. These macrorules are organized according to an abstraction relation for plans into a partial order, from which Argo can efficiently retrieve the most specific plan applicable for solving a new problem. Thus, the use of plan abstraction provides Argo with an effective means for analogy transformation and recognition, enabling systems built with Argo to improve their problem-solving performance as they are used.

5 Conclusions

In this paper we have described a number of topics relevant to using analogical reasoning in the design process. Fundamental to any system that uses analogy for solving problems are the issues of analogy recognition and analogy transformation. In design domains, suitable representations for plans are critical to the effectiveness of analogical reasoning. Goal-structured hierarchical trees or graphs are representations worth investigating because they capture the expressiveness and explicitness of decisions in the design process. Techniques for plan abstraction are the key to success in recognition and transformation of analogies in design systems.

The work on Argo outlined here is based on developing the fundamental methodology for a system that reasons and learns by analogy for solving problems in design. This methodology includes the use of design 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 design plan that allows the plan to be retrieved and ap-

plied efficiently, and the formal definition of an abstraction relation for partially ordering plans. Thus, Argo implements a type of derivational analogy [3,11]. Readers are referred to the works on Argo cited above for more detailed descriptions of the system and the motivation behind our approach.

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] 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.

[11] J. Mostow, "Automated Replay of Design Plans: Some Issues in Derivational Analogy," to appear in *Artificial Intelligence*, 1988.

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

[13] J. Mostow, "Toward Better Models of the Design Process," *AI Magazine*, vol. 6, no. 1, Spring 1985.

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

[15] 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.