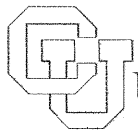
Some Learnability Results for Analogical Generalization *

Clayton Lewis

CU-CS-384-88



University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

* This research was supported by the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, under Contract No. N00014-85-K-0452, Contract Authority Identification No. 702-009. Approved for public release; distribution unlimited. Reproduction in whole or part is permitted for any purpose of the United States.

Some Learnability Results for Analogical Generalization

Clayton Lewis

CU-CS-384-88 January 1988

Department of Computer Science
Campus Box 430
University of Colorado,
Boulder, Colorado, 80309

This research was supported by the Personnel and Training Research Programs, Psychological Sciences Division, Office of Naval Research, under Contract No. N00014-85-K-0452, Contract Authority Identification No. NR 702-009. Approved for public release; distribution unlimited. Reproduction in whole or part is permitted for any purpose of the United States.

Some Learnability Results for Analogical Generalization

Clayton Lewis
Department of Computer Science and
Institute of Cognitive Science
Campus Box 430
University of Colorado
Boulder CO 80309
(303) 492 6657
clayton at boulder on csnet

January 13, 1988

Topic keywords: concept learning, analogical reasoning, theoretical analysis

Abstract.
Progress has been made in characterizing formally the capabilities and performance of inductive learning algorithms. Similar characterizations are needed for recently-proposed methods that produce generalizations from small numbers of analyzed examples. I consider one class of such methods, based on the analogical generalization technique in Anderson and Thompson's PUPS system. It might appear that some to-be-learned structures can be learned by analogy, while others are too chaotic or inconsistent. I show that this intuition is correct for a simple form of analogical generalization, so that there are learnable and unlearnable structures for this method. In contrast, I show that for PUPS-style generalization analogical structure can be imposed on an arbitrary system (within a broad class I call command systems.) It follows that the constraints on the PUPS-style method lie not in any structural condition on a to-be-learned system but rather in obtaining the knowledge needed to impose analogical structure.

Acknowledgments.
This research was supported by the Office of Naval Research, Contract No. N00014-85-K-0452. I am grateful to John Anderson and David Haussler for useful discussions.

Learnability analysis and analysis-based generalization methods.

Formal analysis of inductive learning mechanisms has succeeded in determining the applicability and performance of various learning alogorithms to various classes of learning tasks (Angluin and Smith 1983; Valiant 1984; Haussler 1987; Kearns, Pitt, and Valiant 1987). Such characterizations are not yet available for recently-proposed methods, including explanation-based learning (Mitchell , Keller,and Kedar-Cabelli 1986, De Jong and Mooney 1986), analogical generalization (Anderson and Thompson 1986), and synthetic generalization of procedures (Lewis 1986), which rely on having an analysis of to-be-generalized examples that includes some indication (different for different methods) of why the example belongs to the concept.

These methods, which can be called analysis-based methods (Lewis, in press), do not fit directly into the framework used to characterize inductive methods, because the input to the generalization process includes information other than the identity of the examples themselves. Further, the generalization process has access to background information not associated with individual examples, such as the domain theory used in explanation-based generalization. Nevertheless, as a start on learnability analysis for these methods one can ask whether these methods are applicable to arbitrary concepts, or whether some concepts are learnable using a given analysis-based method while others are not.

Analogy-based learning might work for some systems and not for others.

This paper aims to investigate this issue for one class of analysis-based methods: analogical generalization. Intuitively, it may appear that the nature of analogical generalization is such that it is applicable only to concepts satisfying some kind of regularity or consistency conditions.

To investigate this intuition I will consider a single kind of to-be-learned concept,

which I will call a command system. A command system consists of a collection of objects called commands, each of which has an associated object called a result. Learning a command system requires being able to supply a command which is associated with a specified result.

While I will discuss only command systems, the formal structure I describe is much more general. Any system which can be described by pairs of associated objects, for example sentences and meanings, or programs and functions they compute, can be subjected to the same analysis I give here.

To apply analogical generalization to the task of learning a command system we will attempt to generalize a single example of a command-result pair in such a way as to allow us to supply the commands that are paired with any other result. Intuitively, it appears that this approach will work for some command systems, which I will call analogical, but not for others, whose structure would be too chaotic and inconsistent. I will show that this intuition is correct for a very simple form of analogical generalization, but not for a more powerful (and plausible) form.

A general framework for modificational analogy.

Analogy can be used in more than one way to solve generalization problems. In structure mapping (Gentner 1983) the analogy A : B :: C : X is solved by determining the relevant relationships between A and B and imposing them on C and X. The unknown X is determined by the requirement that it satisfy these relationships to C. Anderson and Thompson's (1986) PUPS system uses a different approach, which I will call modificational analogy (Lewis, in press). Here X is constructed by modifying B. The modification to apply is determined by finding a modification than transforms A into C. I will use modificational analogy in this discussion.

How does one apply modificational analogy to learning a command system? If c_1 is a

command, and r_1 is its result, and we wish to obtain a new result r_2, we proceed as follows. We find a modification m, drawn from some specified class of functions, for which $m(r_1) = r_2$. We then produce $c_2 = m(c_1)$. The system is analogical if for any pair $[c_1,r_1]$, and any new result r_2, the c_2 we construct in this way has r_2 as its result. Thus we can learn an analogical system from a single example pair.

On the face of it it appears that some command systems are analogical in this sense, and others, perhaps most, are not.

Simple substitution analogy works for some systems but not for others.

The modificational analogy scheme just described behaves differently for different classes of modification functions. Suppose commands and results are sequences of words from some vocabulary, and that the permitted modifications are simply substitutions that replace words by other words. It is easy to see that some command systems are analogical under this scheme while others are not.

Consider first a system containing the pairs [delete eggplant, remove file eggplant] and [delete broccoli, remove file broccoli]. Given the first pair as an example, substitution analogy can correctly determine the command that has as result "remove file broccoli": the substitution that transforms "remove file eggplant" to "remove file broccoli" just replaces "eggplant" by "broccoli". Applying this substitution to "delete eggplant" produces "delete broccoli", which is the correct command.

Now suppose the system contains the pair [delete eggplant, remove file eggplant], as before, but also contains the pair [delete carrot, remove file broccoli]. Substitution analogy now fails to produce the correct command for the

result "remove file broccoli". The process just described gives the command "delete broccoli", as before, rather than the correct form "delete carrot". So this second command system is not analogical under substitution analogy.

Simple substitution analogy is too limited.

Simple substitution analogy fails to capture many situations in which intuitively satisfying analogies can be found. Consider a command system with the pairs [delete e, remove file eggplant], [save e, backup file eggplant], [delete b, remove file broccoli], and [save b, backup file broccoli]. Suppose that the first three of these pairs are presented as examples. It seems that the command to backup broccoli should be derivable from the others by analogy, but simple substitution is inadequate to do this.

There is more than one way to attempt to derive the correct command, but all fail in the same way. If we try to use [save e, backup file eggplant] as our base example, we find that mapping "backup file eggplant" to the desired "backup file broccoli" requires substituting "broccoli" for "eggplant". But "eggplant" does not occur in the command "save e", so the substitution cannot be carried out. Similarly, starting from the pair [delete b, remove file broccoli] leads to the vain attempt to substitute "save" for "remove" in the command "delete b".

Pupstitution extends simple substitution.

Anderson and Thompson (1986) developed an elaboration of substitution which gets over this obstacle. Their idea is that an analogy like that we have been considering requires a representation of examples that includes not simply the surface forms of objects but also an interpretation of their parts. For example the representation of the

command "save e" could include, as a kind of annotation, the information that "e" is "the first letter of eggplant". If the substitution of "broccoli" for "eggplant" is applied to this elaborated representation, we see that "the first letter of eggplant" becomes "the first letter of broccoli". We have background knowledge that this is not "e" but "b", so we conjecture that the desired command is "save b" instead of "save e".

This extended form of substitution, which I will call pupstitution, after PUPS, Anderson and Thompson's production system that incorporates it, requires a more complicated description than simple substitution, since it requires interpretations of parts of objects, and a supply of background knowledge, in order to work. The following account is based on Anderson and Thompson's (1986) scheme but deviates from it in detail and terminology.

Pupstitutions operate not on sequences of words but rather on more complex structures, which I will call interpreted structures. An interpreted structure contains a sequence of words, but also may contain interpretations of these words, individually or in groups. Formally, an interpreted structure is a sequence of components. A component is either a sequence of words or a pair consisting of a sequence of words and an interpreted structure. For example, the command "save e" discussed above could be represented by an interpreted structure whose first component is the word "save", and whose second component is a pair consisting of the word "e" and the interpreted structure which has one component, the sequence of words "the first letter of eggplant". We can write this structure out as (save, [e, (the first letter of eggplant)]). Figure 1a shows a diagrammatic representation that may be clearer.

This definition permits interpretations or parts of them to be assigned further interpretations (though I will not need to do this in the present argument.) For

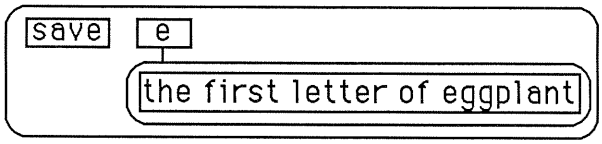


Figure 1a: An interpreted structure shown diagrammatically.

example, the interpreted structure associated with "e" in this example could instead have three components, "the", a pair consisting of "first" and "position used to abbreviate names of files", and the sequence of words "letter of eggplant". The more complex interpreted structure incorporating this further interpretation we can write as (save, [e, (the, [first, (position used to abbreviate names of files)], letter of eggplant)]). It is shown diagrammatically in Figure 1b.

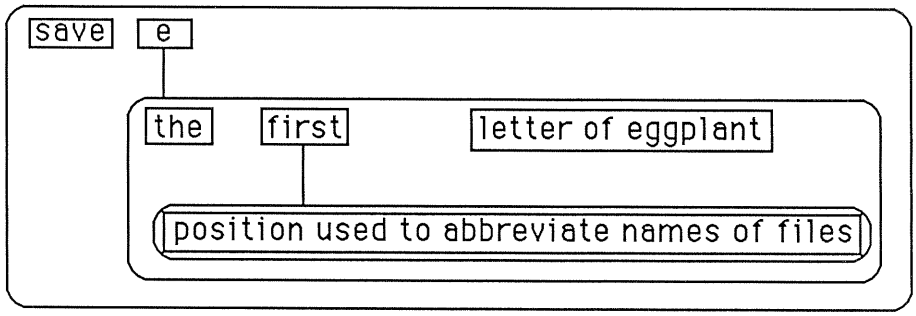


Figure 1b: A nested interpreted structure.

The sequence of words found in the top components of an interpreted structure is called the content of the structure. The content of either example in Figure 1 is just "save e".

The transformation that a pupstitution carries out on an interpreted structure is specified by an ordinary substitution, which I will call the base substitution of the pupstitution, indicating that certain words are to be replaced by others, and by background knowledge, a collection of pairs of sequences of words. The first object in a background knowledge pair will be called an instance, and the second object will be called the interpretation. For example, background knowledge could include the pair [b, the first letter of broccoli].

A pupstitution transforms an interpreted structure into an ordinary sequence of words. It operates component by component, as follows. If the component is an ordinary sequence of words the base substitution is applied to it. If the component is a pair, the pupstitution is applied to the interpreted structure in the pair, producing a sequence of words. This sequence of words is looked up in background knowledge. If it appears in background knowledge as the interpretation of a pair, the instance of the pair replaces the original component. If no such pair is found the result of the pupstitution is undefined. (Various dispositions are possible, including leaving the original component unchanged, or, as Anderson and Thompson 1986 do, invoking analogical generalization to construct an instance of the desired interpretation. What choice is made does not affect the present argument.)

The above operations produce a sequence of words for each component in the original interpreted structure. These sequences are simply concatenated to give the result of the pupstitution.

I illustrate this process by applying a pupstitution whose base substitution replaces "eggplant" by "broccoli", and whose background knowledge includes the pair [b, the first letter of broccoli], to the structure (save, [e, the first letter of eggplant]). The component "save" is unaffected by the base substitution. Processing the component [e, the first letter of eggplant] entails applying the pupstitution

to the structure (the first letter of eggplant). Its sole component is an ordinary sequence of words, so I simply apply the base substitution, getting the sequence "the first letter of broccoli". The pair [b, the first letter of broccoli] occurs in background knowledge, so the original component [e, the first letter of eggplant] is replaced by "b". So the resulting sequence of words is "save b".

We can now use pupstitution to solve analogies in command systems. We first associate an interpreted structure with c_1 and r_1. We do this in any way we wish, as long as the content of each interpreted structure agrees with the object with which it is associated. These associations embody the analysis of the example needed to support the generalization process. We now define a pupstitution (if there is one) which will transform the structure associated with r_1 into r_2. We then apply this pupstitution to the structure associated with c_1. We propose the resulting sequence of words as c_2. Figure 2 shows the entire process applied to the "backup file broccoli" example discussed above.

---

c1: "save e"

r1: "backup file eggplant"

r2: "backup file broccoli"

Interpretation of c1: (save, [e, (the first letter of eggplant)])

Interpretation of r1: (backup file eggplant)

Background knowledge for pupstitution: [b, first letter of broccoli]

Base substitution for pupstitution: "broccoli" for "eggplant"

Pupstitution carries interpretation of r1 to r2, and carries interpretation of c1 to "save b", as required.

Figure 2: Using pupstitution to construct a command.

Pupstitution analogy can always be made to work.

I now return to our central concern, determining what constraints govern the applicability of analogical generalization. For simple substitution analogy we saw that some command systems are analogical but others are not. What is the situation for analogies using pupstitution? Given appropriate interpretations of commands and results, and appropriate background knowledge, any command system whatsoever can be generalized from a single example, under pupstitution.

Let {... $[c_i, r_i]$ ...} be any command system. Suppose we are given as an example that the result of c_1 is r_1, and we are asked to determine what command will produce any other result, say r_j. For each i we select some unique key word k_i, and we construct the background knowledge {...[c_i,command k_i]... [r_i,result k_i] ...}.We assign the interpreted structure ([c_1,command k_1]) to c_1, and the structure ([r_1,result k_1]) to r_1. We construct a pupstitution whose base substitution replaces k_1 with k_j. This will suffice to transform our interpretation of r_1 to r_j. If we now apply this pupstitution to our interpretation of c_1 we obtain c_j, as we require. So this command system, about which we assumed nothing, is analogical. Figure 3 applies this method to the inconsistent "delete carrot" example which I showed was not analogical under simple substitution analogy.

This formal argument may well be unsatisfying, because the required background knowledge transparently includes complete knowledge of the command system. But similar situations can occur in realistic cases of analogy. Suppose we are confronting a desk calculator for the first time. We are shown that pressing the key marked ÷

c1: "delete eggplant"

r1: "remove file eggplant"

r2: "remove file broccoli"

Interpretation of c1: ([delete eggplant,(command word1)])

Interpretation of r1: ([remove file broccoli,(result word1)])

Background knowledge for pupstitution:

[delete carrot, command word2],

[remove file broccoli, result word2]

... plus other pairs ...

Base substitution for pupstitution: "word2" for "word1"

Pupstitution carries interpretation of r1 to r2, and carries interpretation of c1 to "delete carrot", as required.

Figure 3: Application of pupstitution to inconsistent command system.

---

makes the calculator divide. We have background knowledge that ÷ is the sign conventionally associated with division. We conjecture, by analogy, that the + key makes the calculator add. Lacking the background knowledge about arithmetic signs, or failing to relate this knowledge to the calculator, we would be unable to determine what key to press. With this knowledge, appropriately linked to the calculator, we can. The calculator is analogical just if we know and use the right background knowledge.

A more fanciful, but still possible, case is the following. Pat is learning to use a new command language on a computer. On being shown an example of a command and its result, Pat notes the remarkable fact that the command, and a description of its result, occurred in consecutive lines of a nursery rhyme learned years ago.

Given a new result to obtain, Pat notes that a description of this result also appears in the rhyme, and tries the command mentioned in the previous line. It works. Knowing the rhyme, and seeing its connection to the system, has allowed Pat to use an analogy to solve a novel problem.

The point of this example is that for any system whatever that Pat might wish to learn, there exists some rhyme that would allow the system to be learned in this same way. If Pat knew the rhyme, and applied it in the right way, the system would be analogical.

Constraints in analogy lie in getting and applying background knowledge.

Our earlier discussion of analogy using simple subsitution confirmed the intuition that some structures are analogical, that is, generalizable by analogy, while others are not. But the conclusion for pupstitution is that analogical structure can be imposed on any command system, no matter how seemingly chaotic or inconsistent. Therefore the constraints on generalization by analogy using pupstitution cannot lie in the structural requirements of analogy: under pupstitution, structures *in themselves* are neither analogical or not, as they are under simple subsitution. Rather, any constraints on the applicability of generalization by pupstitution must lie in the process of obtaining the needed background knowledge, and seeing its application to the system at hand. In the terms of the nursery rhyme example, how does Pat come to know the right rhyme? Given the rhyme, how can Pat reliably determine the relationship between the rhyme and the system? After all, Pat may know many rhymes, and even many rhymes in which commands and outcomes appear in different associations. The pursuit of constraints on analogical generalization must shift to these questions.

References.

Anderson, J.R. and Thompson, R. (1986). Use of analogy in a production system architecture. Paper presented at the Illinois Workshop on Similarity and Analogy, Champaign-Urbana, June, 1986.

Angluin, D. and Smith, C.H. (1983) Inductive inference: Theory and methods. *Computing Surveys*, **15**, 237-269.

DeJong, G. and Mooney, R. (1986) Explanation-based learning: An alternative view. *Machine Learning*, **1**.

Gentner, D. (1983) Structure mapping: A theoretical framework for analogy. *Cognitve Science*, **7**, 155-170.

Haussler, D. (1987) Bias, version spaces, and Valiant's learning framework. In P. Langley (Ed.) *Proc. Fourth International Workshop on Machine Learning*, Los Altos, CA: Morgan Kaufmann.

Kearns, M., Li, M., Pitt, L. and Valiant, L. Recent results on boolean concept learning. In P. Langley (Ed.) *Proc. Fourth International Workshop on Machine Learning*, Los Altos, CA: Morgan Kaufmann.

Lewis, C.H. (1986) A model of mental model construction. In *Proceedings of CHI'86 Conference on Human Factors in Computer Systems*. New York: ACM, 306-313.

Lewis, C.H. (in press) Why and how to learn why: Analysis-based generalization of procedures. *Cognitive Science*. (Earlier version available as Technical Report CS-CCU-347-86, Department of Computer Science, University of Colorado, Boulder CO.)

Valiant, L.G. (1984) A theory of the learnable. *Communications of the ACM*, **27**, 1134-1142.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CU-CS-384-88	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION University of Colorado 6b. OFFICE SYMBOL (*If applicable*)	7a. NAME OF MONITORING ORGANIZATION Cognitive Science Program
6c. ADDRESS (*City, State, and ZIP Code*) Department of Computer Science CB 430 Boulder, CO 80309-0430	7b. ADDRESS (*City, State, and ZIP Code*) Office of Naval Research (Code 1142PT) 800 N. Quincy St. Arlington, VA 22217-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER N00014-85-K-0452
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 61153N; PROJECT NO. RR04206; TASK NO. RR04206-0C; WORK UNIT ACCESSION NO. R702-009

11. TITLE (*Include Security Classification*)
Some Learnability Results for Analogical Generalization

12. PERSONAL AUTHOR(S)
Clayton Lewis

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*Year, Month, Day*)	15. PAGE COUNT
Technical	FROM _____ TO _____	88-1-16	14

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (*Continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUB-GROUP	Concept learning, analogical reasoning, theoretical analysis
05	08		

19. ABSTRACT (*Continue on reverse if necessary and identify by block number*)

Progress has been made in characterizing formally the capabilities and performance of inductive learning algorithms. Similar characterizations are needed for recently-proposed methods that produce generalizations from small numbers of analyzed examples. I consider one class of such methods, based on the analogical generalization technique in Anderson and Thompson's PUPS system. It might appear that some to-be-learned structures can be learned by analogy, while others are too chaotic or inconsistent. I show that this intuition is correct for a simple form of analogical generalization, so that there are learnable and unlearnable structures for this method. In contrast, I show that for PUPS-style generalization analogical structure can be imposed on an arbitrary system (within a broad class I call command systems.) It follows that the constraints on the PUPS-style method lie not in any structural condition on a to-be-learned system but rather in obtaining the knowledge needed to impose analogical structure.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS		
22a. NAME OF RESPONSIBLE INDIVIDUAL Susan Chipman	22b. TELEPHONE (*Include Area Code*) (202) 696-4318	22c. OFFICE SYMBOL ONR 1142CS

DD Form 1473, JUN 86 *Previous editions are obsolete.* SECURITY CLASSIFICATION OF THIS PAGE