SELF-CORRECTING GENERALIZATION

Stephen B. Whitehill
Department of Information and Computer Science
University of California at Irvine
Irvine, Ca 92717

ABSTRACT

A system is described which creates and generalizes rules from examples. The system can recover from an initially misleading input sequence by keeping evidence which supports (or doesn't support) a given generalization. By undoing over-generalizations, the system maintains a minimal set of rules for a given set of inputs.

I GENERALIZATION

Many programs have been written which generalize examples into rules. Soloway[5] generalizes the rules of baseball from examples. Hayes-Roth[3] and Winston[8] generalize common properties in structural descriptions. Vere[6] has formalized generalization for several applications.

If a program maintains a current hypothesis about the rule set as it sees new examples it is said to generalize incrementally. A program that incrementally forms generalizations may be sensitive to the order in which examples are presented. If exceptional examples are encountered first, the program may over-generalize. If the program is to recover and undo the over-generalization it must have a certain amount of knowledge about why the over-generalization was made.

The system to be described here has this type of self-knowledge. By associating positive and negative evidence with each generalization, the system is able to reorganize its rules to recover from over-generalizations. Even if it is initially misled by an unusual sequence of inputs it still discovers the most reasonable set of rules.

II THE PROBLEM DOMAIN

The problem domain chosen for the system is learning language morphology from examples. For example, from the words "jumped", "walked" and "kissed" we can deduce the rule that the English past tense is formed by adding "ed".

The Concept Description Language consists of a set of rules. Each rule is a production consisting of a pair of character strings. When the left-hand side is matched the right-hand side is returned. The left-hand string may optionally contain a '*' which will match zero or more characters. In this case the right hand character string may contain a '*' and the value the star on the left hand side which was matched is substituted for the star on the right hand side. For example the production for the example above looks like: *->*ED. This *->*ED rule does not always work. From "baked", "related" and "hoped" we see that for words ending in "e" we need only to add an "d". This rule is written as *E->*ED. For this domain, the problem bears a resemblence to the Grammatical Inference Problem[4].

III RELATIONSHIPS BETWEEN RULES

Rule Containment. Given rules P1 and P2, let S1 be the set of strings which match the left-hand side of P1 and S2 the set of strings which match left-hand side of P2. If S1 is a subset of S2 then we say that P1 is contained by P2. This forms a partial ordering of the rules.

The Is-a-generalization-of Operator. Given rules P1 and P2, let S1 be the set of strings which match the left-hand side of P1 and S2 the set which match the left-hand side of P2. If P1 contains P2 and if P1 and P2 produce the same result for every element in S2 then P1 is a generalization of P2. This is also a partial ordering.

Note the distinction between the containment operator and the is-a-generalization-of operator. Basically, containment deals with the left-hand side of a rule. Is-a-generalization-of deals with both sides. An example will clarify this:

*->*S contains *K->*KS
*->*S is a generalization of *K->*KS
*->*S contains *CH->*CHES
*->*S and *CH->*CHES are unrelated by generalization

By definition, if P1 is a generalization of P2, P1 contains P2. The converse is not necessarily true.

If P1 is a generalization of P2 and P1 is a generalization of P3 then P1 is a common generalization of P2 and P3. P1 is a maximal common generalization[6] if P1 is a generalization of no other common generalization. Roughly, the maximal common generalization is the one which captures all common features of the rules being generalized. For example, given WALK->WALKED and TALK->TALKED, possible generalizations are: *->*ED, *K->*KED, *LK->*LKED and *ALK->*ALKED. The last one, *ALK->*ALKED is the maximal one. In the concept description language we are using all common generalizations are related on the is-a-generalization-of operator. Therefore in our domain the maximal common generalization is unique.

IV ORGANIZATION OF RULES

The rules and their evidence are organized in a tree structure. At the top level the rules are organized as a rule list. A rule list is a list of rules partially ordered on the containment operator. No rule may be contained by a rule which precedes it in the list. Associated with most rules is some evidence which is itself in the form of another rule list. The only rules without evidence are the example pairs whose evidence is fact. These correspond to terminal nodes in the evidence tree. If a rule R1 would contain a rule R2 which follows it then R1 is marked as being blocked by R2. If R1 blocks R2 then evidence for R1 is negative evidence for R2.

The positive evidence consists of those rules which were generalized into the current generalization. Negative evidence for a generalization G is all the evidence of generalizations that are blocked by G. Thus when *->*ES blocks (*N->*NS + *K->*KS ==> *->*S) that is negative evidence for *->*ES. The evidence described here is much like evidence used for explanation in [1] or to maintain beliefs in [2]. In our system the evidence is used for reorganization, but it could be used for these other purposes as well.

Rule Application and Conflict Resolution. When the rule interpreter produces a response, it is as if it finds all rules which match the given input and then uses the one which doesn't contain any of the others (informally, the one with the least general left-hand side). In reality the rules and rule interpreter are organized so that the first rule that matches is the desired one.

Inserting New Rules. If a rule has produced the correct result, the new example pair is inserted into the evidence list for the rule. If the rule system has not produced the correct result the rule is inserted in the main rule list before the first rule with which it will generalize. If it will not generalize with any rule, it is inserted before the first rule that contains it. The same rule insertion algorithm is used to insert new rules or evidence. This means that generalizations take place in an evidence list in the same way that they do in the main rule list.

V SYSTEM REORGANIZATION

Each blocked generalization has knowledge about which generalization is blocking it. Whenever evidence for a blocked generalization G1 is entered into the rule structure, we check to see if there is now more evidence for G1 than for the blocking generalization G2. If so, G2 is moved to the position in the rule list immediately preceding G1, G2 is marked as being blocked by G1 and G1 is no longer marked as being blocked.

There are several choices on how to compare positive and negative evidence. The one chosen is to count how much direct evidence there is for a rule. Direct evidence is that evidence found in the top level rule list in the evidence tree. Another method which was rejected for this application is to count the total number of pieces of evidence in the evidence tree. The first method was chosen because *CH->*CHES and *X->*XES are exceptions to *->*S (rather than *A->*AS, *B->*BS, *C->*CS, etc. being exceptions to *->*ES) because there is more direct evidence for *->*S (rules like *A->*AS) than for *->*ES. Even if half the words in English used *CH->*CHES this would still be an exception to *->*S.

This method produces the most reasonable set of rules. The system has been tested on pluralizing French adjectives. French has a much more complicated morphology than English, having not only exceptions to rules but also exceptions to exceptions. The system was found to produce the same rules for pluralizing French adjectives as those found in a French-English dictionary . A detailed example of this appears in [7].

VI UNDOING GENERALIZATIONS - AN EXAMPLE

The system is written in MLISP and runs on UCI LISP. The following example was annotated from a trace of the system in operation. The rules and evidence are printed as a tree. The evidence for a node is indented from the parent node in the printout.

```
INPUT WORD? church
WHAT IS RESPONSE? churches
INPUT WORD? match
WHAT IS RESPONSE? matches
```

```
INPUT WORD? bus
WHAT IS RESPONSE? buses
RULES:
 *->*ES     BUS->BUSES
            *CH->*CHES   MATCH->MATCHES
                         CHURCH->CHURCHES
```

At this point we have over-generalized. We will find this out later. The only rule seen by the rule interpreter is *->*ES. BUS->BUSES and *CH->*CHES are evidence for *->*ES. MATCH->MATCHES and CHURCH->CHURCHES are evidence for the *CH->*CHES rule (which is itself evidence).

```
INPUT WORD? book
IS RESPONSE BOOKES? n
WHAT IS RESPONSE? books
INPUT WORD? back
IS RESPONSE BACKES? n
WHAT IS RESPONSE? backs
RULES:
 B*K->B*KS  BACK->BACKS
            BOOK->BOOKS
 *->*ES     BUS->BUSES
            *CH->*CHES   MATCH->MATCHES
                         CHURCH->CHURCHES
```

What should be regular cases are treated as exceptions.

```
INPUT WORD? car
IS RESPONSE CARES? n
WHAT IS RESPONSE? cars
RULES:
 (*->*S)    CAR->CARS
            B*K->B*KS    BACK->BACKS
                         BOOK->BOOKS
 *->*ES     BUS->BUSES
            *CH->*CHES   MATCH->MATCHES
                         CHURCH->CHURCHES
```

At this point we want to make the generalization *->*S but this generalization is blocked by *->*ES. We make the generalization but mark it as blocked. The parentheses indicated that the rule is blocked. The only productions seen by the production system are: CAR->CARS, B*K->B*KS and *->*ES. The blockage of *->*S is negative evidence for *->*ES. The system will detect that the rules are in the wrong order when there is more evidence for *->*S (and hence against *->*ES) than there is for *->*ES. At this point there is just as much negative evidence as positive (looking down one level in the evidence tree).

```
INPUT WORD? bat
IS RESPONSE BATES? n
WHAT IS RESPONSE? bats
RULES:
 (*->*ES)   BUS->BUSES
            *CH->*CHES   MATCH->MATCHES
                         CHURCH->CHURCHES
 *->*S      CAR->CARS
            B*K->B*KS    BACK->BACKS
                         BOOK->BOOKS
            BAT->BATS
```

This addition negative evidence for *->*ES has caused a reordering of the rules. *->*ES is now blocked by *->*S (as it should be).

```
INPUT WORD? house
IS RESPONSE HOUSES? y
INPUT WORD? bunch
IS RESPONSE BUNCHES? y
```

The system now has a properly ordered rule set and can handle both regular and irregular cases.

VII CONCLUSIONS

By giving a generalization program some self-knowledge it can recover from initially misleading input sequences. This introspection can be achieved by associating positive and negative evidence with generalizations. Without knowledge about what led to a generalization, it is not possible to undo the generalization. The system described here discovers the most reasonable set of morphological rules for a given language construct (the set found in a dictionary) regardless of the input sequence. The choice of language morphology as the problem domain was arbitrary. Any domain with a concept description language whose maximal common generalization is unique would serve just as well. Further work is needed for concept description languages whose maximal common generalization is not necessarily unique. Any incremental generalization program could improve its ability to recover from misleading input by applying the techniques described.

REFERENCES

[1] Bechtel, R., Morris, P. and Kibler, D., "Incremental Deduction in a Real-time Environment", Canadian Society for the Computational Studies of Intelligence (May 1980).

[2] Doyle, J., "A Glimpse of Truth Maintenance", 6-IJCAI (1979), 232-237.

[3] Hayes-Roth, F. and McDermott, J., "Knowledge Acquisition from Structural Descriptions", Department of Computer Science, Carnegie-Mellon Univ. (1976).

[4] Hunt, E. "Artificial Intelligence" 1975

[5] Soloway, E. and Riseman, E., "Levels of Pattern Description in Learning", 5-IJCAI (1977), 801-811.

[6] Vere, S., "Induction of Relational Productions in the Presence of Background Information", 5-IJCAI (1977), 349-355.

[7] Whitehill, S., "Self-correcting Generalization". U.C. Irvine C.S. Tech Report no. 149. (June 1980).

[8] Winston, P., "Learning Structural Descriptions From Examples", MIT-AI Technical Report 231, (1970).