An Iterative and Bottom-up Procedure for Proving-by-Example

Masami Hagiya

Department of Information Science, University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo 113, JAPAN
hagiya@is.s.u-tokyo.ac.jp

Abstract. We give a procedure for generalizing a proof of a concrete instance of a theorem by recovering inductions that have been expanded in the concrete proof. It consists of three operations *introduction, extension* and *propagation,* and by iterating these operations in a bottom-up fashion, it can reconstruct nested inductions. We discuss how to use EBG for identifying the induction formula, and how EBG must be modified so that nested inductions can be reconstructed.

1 Introduction

When a teacher explains a theorem in mathematics to a student, he often picks up an appropriate instance of the theorem and explains why the theorem is true using that instance. In other words, he gives a proof of an instance of a theorem instead of proving its general case. A proof of a concrete instance often contains enough information to reconstruct a proof of the general case.

Explanation-based generalization (*EBG* for short), which has been formulated in the field of *deductive learning* or *explanation-based learning,* is a method for generalizing a proof as discussed above [2], though pure EBG generalizes a proof by simply replacing a term in a proof with a variable.

The problem we study in this paper, on the other hand, is that of generalizing a proof of an instance of a theorem and obtaining an inductive proof of the general case of the theorem by automatically recovering inductions that have been expanded in a proof of an instance. Some researchers in the field of explanation-based learning have also studied this problem under the name of *generalizing number* [?, 5], but there can be found very few foundational studies on the method that are based on formal logic except those by the author [3, 4].

Assume that we are given a concrete proof, i.e., a proof of a concrete instance of a theorem, as depicted in the left-hand side of Figure 1. In the figure, $P(i)$ denotes a proposition on natural number i, Π a proof of $P(0)$, Φ_0 a proof of $P(1)$ from $P(0)$, and Φ_1 a proof of $P(2)$ from $P(1)$.

If one can generalize proofs Φ_0 and Φ_1 to a single general proof Φ of $\forall i(P(i) \rightarrow P(S(i)))$, then one can obtain an inductive proof as depicted in the right-hand side of Figure 1. Note that S denotes the successor function and numerals 1, 2, etc. are considered as abbreviations of terms $S(0)$, $S(S(0))$, etc. The conclusion

$$\begin{array}{c} \Pi \\ P(0) \\ \Phi_0 \\ P(1) \\ \Phi_1 \\ P(2) \end{array} \qquad \qquad \dfrac{\begin{array}{cc} \Pi & \Phi \\ P(0) & \forall i(P(i) \rightarrow P(S(i))) \end{array}}{P(2)}$$

Fig. 1. A Concrete Proof and an Inductive Proof

$P(2)$ is considered to be derived from the universal formula $\forall i P(i)$. In proof Φ, $P(i)$ is called the *induction formula* and i the *induction variable.*

The procedure for generalizing a concrete proof as explained above must consist of those steps of finding the parts Π, Φ_0 and Φ_1, fixing the induction formula $P(i)$ and generalizing Φ_0 and Φ_1 to Φ.

The purpose of this paper is to formulate a generalization procedure that can iterate the above steps and reconstruct inductions that are nested within one another. We discuss how to use EBG for identifying the induction formula, and how EBG must be modified so that nested inductions can be reconstructed.

In this paper, we do not care whether a concrete proof is written by hand or automatically generated by a theorem prover. An SLD-trace of a logic program is a typical example of the latter case. As is explained in the next section, our procedure can reconstruct inductions whose induction formula $P(i)$ is of form $\exists x Q(i, x)$. By applying it to an SLD-trace of a logic program, one can obtain a proof that guarantees the existence of x for any i such that $Q(i, x)$ holds, i.e., the termination of the logic program with respect to the input-output relation $Q(i, x)$, where i is an input and x is an output.

In the deductive approach for program synthesis, the relation $Q(i, x)$ is given in advance and then the formula $\forall i \exists x Q(i, x)$ is proved. The generalization procedure of this paper, on the other hand, is given an SLD-trace of a logic program, and generalizes the trace to an inductive proof. The relation $Q(i, x)$ is not given in advance but is identified during generalization.

Bruynooghe, De Raedt and De Schreye in [1] use a concrete proof to guide unfold/fold program transformation, but a concrete proof is not directly generalized to an inductive proof.

In the next section, we introduce the recursion operator and formulate the induction schema. We then describe the operations of the generalization procedure. The extension of EBG is finally discussed.

2 Generalization Procedure

The generalization procedure of this paper reconstructs inductions whose induction formula $P(i)$ is of form $\exists x Q(i, x)$, where $Q(i, x)$ is an atomic formula. In general, x is a vector of variables, but for simplicity we assume that x is a single variable in this paper.

Formulas of form $\forall i \exists x Q(i,x)$ are called $\forall\exists$-*specifications,* and (constructively) proving a formula of this kind corresponds to synthesizing a program satisfying the input-output relation $Q(i,x)$, where i is considered as an input and x an output.

Since it is not easy to directly handle the existential quantifier $\exists$, we introduce the *recursion operator,* denoted by r, and express the output explicitly in term of the input and r. If n is a term denoting a natural number, $f(i,x)$ is a function whose first argument i is a natural number, and t is a term, then $r(n,f,t)$ is also a term. The operator r satisfies the following reduction rules.

$$\begin{aligned} r(0,f,t) &= t \\ r(S(n),f,t) &= f(n, r(n,f,t)) \end{aligned}$$

After introducing r, we can formulate the following inference rule, which we call Σ_1-*induction.*

$$\frac{\begin{matrix} \Pi \\ P(0,t) \end{matrix} \qquad \begin{matrix} \Phi \\ \forall i \forall x (P(i,x) \rightarrow P(S(i), f(i,x))) \end{matrix}}{P(n, r(n,f,t))}$$

This rule says that if Π is a proof of $P(0,t)$ and Φ is a proof of $\forall i \forall x(P(i,x) \rightarrow P(S(i), f(i,x)))$, then one can conclude $P(n, r(n,f,t))$ for any integer n. We also write the above inductive proof as $R(n,f,\Phi,t,\Pi)$, i.e., $R(n,f,\Phi,t,\Pi)$ is a proof of $P(n,r(n,f,t))$. In Φ, i is called the induction variable, $P(i,x)$ the induction formula and $f(i,x)$ the induction term.

The generalization procedure consists of the following three operations.

- Introducing an induction — If a proof of $P(1,u)$ from $P(0,t)$ can be generalized to an induction, replace it with an inductive proof.
- Extending an induction — If $P(S(n),v)$ is derived from $P(n,u)$ whose proof is inductive, extend the induction to $P(S(n),v)$.
- Propagating an induction — If there has been obtained an inductive proof of $P(n,u)$ and there exists a proof of $P(0,t)$ elsewhere, replace the latter with the inductive proof.

The procedure iteratively applies these operations from the inner subproofs to the outer ones, i.e., it applies the operations in a bottom-up fashion.

2.1 Introducing Inductions

In order to obtain a Σ_1-induction by generalization, we first search in the given concrete proof for a subproof of the following form.

$$\begin{matrix} \Pi \\ A_0 \\ \Phi_0 \\ A_1 \end{matrix}$$

A_0 and A_1 are atomic formulas sharing the same predicate symbol. Π is a proof of A_0, and Φ_0 is a proof of A_1 from A_0, where the assumption A_0 is used exactly once in Φ_0.

The identification of the induction formula $P(i,x)$ and the induction term $f(i,x)$ is the most difficult step of this operation. Here is one of the possible methods.

We first apply the following EBG procedure on Φ_0, where A_0 is considered as the only *operational* atom.

1. For each maximal term appearing in Φ_0 (except those appearing in the axioms), introduce a new variable and replace the term with the new variable.
2. Do unification in each inference step of Φ_0 so that the inference step becomes valid.

By the second step, we obtain the most general substitution for the newly introduced variables such that the result is a valid proof.

Assume that by EBG, we have obtained a proof Φ_0' of A_1' from A_0'. Since Φ_0 is an instance of Φ_0', there exists a substitution θ for the newly introduced variables such that $\theta(\Phi_0') = \Phi_0$. Let A_0' be of form $P(x_1, \cdots, x_n)$, where $x_1, \cdots, x_n$ are newly introduced variables that were not instantiated during EBG. We then check the following condition: *A_1' is an instance of $P(x_1, \cdots, x_n)$, i.e., there exist terms $u_1, \cdots, u_n$ such that $A_1' = P(u_1, \cdots, u_n)$.*

If the above condition is satisfied, we divide the variables $x_1, \cdots, x_n$ into the following three sets:

1. the set of x_j such that $u_j = S(x_j)$ and $\theta(x_j) = 0$.
2. the set of x_j such that $u_j = x_j$, and
3. the set of others.

We then introduce a new variable i, which is intended to become the induction variable, and replace each variable x_j in Set 1 with i. Each variable x_j in Set 2 is instantiated simply with $\theta(x_j)$. Each variable z that is not among $x_1, \cdots, x_n$ is also instantiated with $\theta(z)$.

In the following explanation, we assume that Set 3 consists of a single variable x and u is the corresponding term. We then define the function f by $f(i,x) = u$. As a result, we obtain a proof Φ of $P(S(i), f(i,x))$ from $P(i,x)$. Since i and x are arbitrary, it can be considered as a proof of $\forall i \forall x (P(i,x) \rightarrow P(S(i), f(i,x)))$.

We can finally obtain the inductive proof $R(1, f, \Phi, t, \Pi)$ of $P(1, f(0,t))$, because $A_0 = P(0,t)$ for some term t. The original proof of A_1 is then replaced with $R(1, f, \Phi, t, \Pi)$.

Let us give a simple example. Assume that we have the following axioms for the predicates add and mul. These axioms are exactly the definite clauses of the logic program defining add and mul.

$$\forall i\, \mathtt{add}(0,i,i)$$
$$\forall i \forall j \forall k (\mathtt{add}(i,j,k) \rightarrow \mathtt{add}(S(i),j,S(k)))$$
$$\forall i\, \mathtt{mul}(0,i,0)$$
$$\forall i \forall j \forall k \forall l (\mathtt{mul}(i,j,k) \wedge \mathtt{add}(k,j,l) \rightarrow \mathtt{mul}(S(i),j,l))$$

The proof of mul$(2,2,4)$, which uses these axioms, takes the form of the tree in the following figure.

```
                                add(0,2,2)
                                         |
     mul(0,2,0)      add(0,2,2)    add(1,2,3)
              \          /               |
           mul(1,2,2)               add(2,2,4)
                     \                 /
                        mul(2,2,4)
```

The proof can be considered as an SLD-trace of the logic program.

The operation explained so far can be applied on the subproof whose conclusion is add$(1,2,3)$. Applying EBG on the proof of add$(1,2,3)$ from add$(0,2,2)$, we obtain a proof of add$(S(i),y,S(x))$ from add(i,y,x). By the above method, y is instantiated with 2, and we obtain the induction formula add$(i,2,x)$, and the induction term $S(x)$. Function f is defined by $f(i,x)=S(x)$. We finally obtain the proof $R(1,f,\Phi,2,\Pi)$ of add$(1,2,3)$, where Π is the proof of add$(0,2,2)$ and Φ is the proof of $\forall i \forall x($add$(i,2,x) \rightarrow$ add$(S(i),2,S(x))$.

2.2 Extending Inductions

If n is a numeral and $P(S(n),f(n,u))$ is proved from $P(n,u)$, whose proof is of form $R(n,f,\Phi,t,\Pi)$, then one can extend the inductive proof $R(n,f,\Phi,t,\Pi)$ to $R(S(n),f,\Phi,t,\Pi)$, because $u=r(n,f,t)$.

In the previous example, this operation can be applied on the subproof of add$(2,2,4)$. As a result, we obtain the proof $R(2,f,\Phi,2,\Pi)$ of add$(2,2,4)$ from the inductive proof $R(1,f,\Phi,2,\Pi)$ of add$(1,2,3)$.

2.3 Propagating Inductions

Assume that there exists a proof $R(n,f,\Phi,t,\Pi)$ of $P(n,u)$, where Φ is a proof of $P(S(i),f(i,x))$ from $P(i,x)$. Assume also that there exists a proof of $P(0,t')$ somewhere else. Then one can replace the proof of $P(0,t')$ with $R(0,f,\Phi,t',\Pi)$. We call this operation *propagation* of an induction.

This operation can be applied on add$(0,2,2)$ above mul$(1,2,2)$. As a result, the proof of add$(0,2,2)$ is replaced with the inductive proof $R(0,f,\Phi,2,\Pi)$.

One may generalize $R(n,f,\Phi,t,\Pi)$ before applying propagation. This makes the generalization procedure more complete.

2.4 EBG on Inductions

Since the procedure is iterated to reconstruct nested inductions, EBG may be applied on an inductive proof of form $R(n,f,\Phi,t,\Pi)$. Special care must be taken when applying EBG on Φ because Φ is a proof of $P(S(i),f(i,x)))$ from $P(i,x)$. Since i and x are universal variables, terms containing i or x must not be replaced with a new variable during EBG. This means that terms that are maximal among those that do not contain i or x are replaced with new variables.

If the result of generalizing Φ is a proof Φ' of $P'(S(i), f'(i,x))$ from $P'(i,x)$ and that of generalizing Π is a proof Π' of P'', then one must introduce another new variable y and unify $P'(0,y)$ and P'' to make the induction valid.

In the previous example, EBG is applied on an inductive proof while the following subproof is being generalized.

```
mul(0,2,0)     add(0,2,2)
         \         /
          mul(1,2,2)
```

The proof $R(0, f, \Phi, 2, \Pi)$ of $\mathsf{add}(0,2,2)$, which was obtained by propagation, is generalized to the proof $R(y, f, \Phi', z, \Pi')$, whose conclusion is $\mathsf{add}(y, z, r(y, f, z))$. Therefore we obtain a proof of $\mathtt{mul}(S(i), 2, r(x, f, 2))$ from $\mathtt{mul}(i, 2, x)$. Thus we can introduce an induction by the introduction operation and reconstruct a nested induction. Note that the induction term of the outer induction is $r(x, f, 2)$.

Since the conclusion of an induction contains the recursion operator r, one must in general do unification between terms containing r during EBG. The details are not discussed here due to the space limitation.

References

1. Bruynooghe,M., De Raedt,L., De Schreye,D.: Explanation based program transformation, *Proceedings of IJCAI 89,* 1989, pp.407-412.
2. Ellman,T.: Explanation-based learning: A survey of programs and perspectives, *ACM Computing Surveys,* Vol.21, No.2 (1989), pp.163-221.
3. Hagiya,M.: Programming by example and proving by example using higher-order unification, *10th International Conference on Automated Deduction* (Stickel,M., ed.), Lecture Notes in Artificial Intelligence, Vol.449 (1990), pp.588-602.
4. Hagiya,M.: From programming-by-example to proving-by-example, *Theoretical Aspects of Computer Software* (Ito, T., Meyer,A.R., eds.), Lecture Notes in Computer Science, Vol.526 (1991), pp.387-419.
5. Shavlik,J.W., DeJong,G.F.: An explanation-based approach to generalizing number, *Proceedings of IJCAI 87,* 1987, pp.236-238.
6. Shavlik,J.W., DeJong,G.F.: Acquiring general iterative concepts by reformulating explanations of observed examples, *Machine Learning Volume III* (Kodratoff,Y., Michalski,R., eds.), 1990, pp.302-350.