Inductive Methods for Reasoning about Abstract Data Types *

Stephen J. Garland and John V. Guttag †
MIT Laboratory for Computer Science
545 Technology Square
Cambridge, MA 02139

Abstract

Rewriting techniques have been used to reason about a variety of topics related to programming languages, e.g., abstract data types, Petri Nets, FP programs, and data bases. They have also been used in the implementation and definition of a variety of programming languages.

At the 1980 POPL Conference, David Musser proposed a new method of proving inductive properties of abstract data types. Since that time, this method, which came to be called inductionless induction, has attracted considerable attention. Numerous applications and improvements have been proposed and several implementations described. However, little or no work has appeared that questions the basic utility of the idea.

The thesis of this paper is that while induction using equational term-rewriting holds great promise, inductionless induction does not. More specifically, we argue that for reasoning about abstract data types traditional inductive methods are usually superior.

---

*This paper appears in the *Proceedings of the 15th ACM Conference on Principles of Programming Languages*, January 1988.

†This research was supported in part by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-83-K-0125, and by the National Science Foundation under grant DCR-8411639.

1 Introduction

Many properties associated with abstract data types can be axiomatized by sets of (implicitly) universally quantified equations, e.g.,

```
isEmpty(empty) = true
isEmpty(singleton(e)) = false
isEmpty(union(s1, s2)) = isEmpty(s1) & isEmpty(s2)
subset(empty, s) = true
subset(s, empty) = isEmpty(s)
subset(singleton(e1), singleton(e2)) = (e1 = e2)
subset(singleton(e), union(s1, s2)) =
     subset(singleton(e), s1) | subset(singleton(e), s2)
subset(union(s1, s2), s3) =
     subset(s1, s3) & subset(s2, s3)

AC:     &    |    union
```

(The last line abbreviates equations asserting that `&`, `|`, and `union` are associative and commutative.) Equational axiomatizations are attractive because of their simplicity and the ease with which they can be combined with other specification formalisms [14]. Furthermore, equational term-rewriting provides a straightforward mechanism (reduction to normal form) for proving equational consequences.

These facts have motivated several efforts to build systems for equational reasoning using term-rewriting [24] or equational term-rewriting [8, 22]. In addition to abstract data types, rewriting techniques have been used to reason about a variety of other topics related to programming languages, e.g., Petri Nets [6], FP programs [2], and data bases [7]. They have also been used in the implementation and definition of a variety of programming languages, e.g., [1, 9, 10].

However, there are often infinitely many properties of an abstract data type that do not follow from any finite set of equations. For example, the equation

```
subset(s, s) = true
```

does not follow from the above axioms. The problem is that the axioms do not imply that all sets are generated by `empty`, `singleton`, and `union`. Equations such as

```
subset(strangeset, strangeset) = false
```

where `strangeset` is a new constant, are consistent with the axioms, but not with the desired conclusion.

This problem has led designers of specification languages to include extra-equational features [5, 13]. It has led those interested in reasoning about abstract data types to go beyond equational theories and to consider various kinds of inductive theories.

At the 1980 POPL Conference, David Musser proposed a new method of proving inductive properties of abstract data types [23]. Since that time, this method, which came to be called inductionless induction, has attracted considerable attention, e.g., [11, 16, 17, 18, 20, 21, 25]. Numerous applications and improvements have been proposed and several implementations described. However, little work has appeared that questions the basic utility of the idea.

The thesis of this paper is that while induction using equational term-rewriting holds great promise, inductionless induction does not. More specifically, we argue that for reasoning about abstract data types traditional inductive methods are usually superior.

Section 2 discusses the role of induction in reasoning about abstract data types and the somewhat different logical bases of inductionless and traditional induction. Section 3 takes a brief look at induction in LP, a theorem prover based on rewrite rules; it emphasizes proofs by traditional induction, since the method of inductionless induction method available in LP is described elsewhere in the literature [16]. Section 4 discusses the relative advantages of traditional induction for reasoning about abstract data types.

2 Abstract data types, equational logic, and inductive theories

Various sources of incompleteness in logic make it important to exercise some care in the use of the word "consequence." Something that is true in a particular mathematical structure may not be a consequence of particular axioms about that structure (because the axiomatization is incomplete). Something that is a semantic consequence (i.e., true in all models) of a set of axioms may not be provable using particular rules of inference (because the rules of inference are incomplete, e.g., because a complete inference

system does not exist). Hence we take care to distinguish what it means for an equation to be true in an inductively-generated structure from what it means for that equation to be provable by traditional or inductionless induction.

For concreteness, consider the arithmetic of the natural numbers, that is, the structure $\mathbf{N} = \langle N, s, +, *, 0, 1\rangle$, where N is the set of nonnegative integers and s is the successor function. This structure is a model of the set E_N of equations

$$\begin{array}{ll} x + 0 = x & x * 0 = 0 \\ x + s(y) = s(x + y) & x * s(y) = (x * y) + x \\ 1 = s(0) & \end{array}$$

which define $+$, $*$, and 1 in terms of 0 and s. The definitions of $+$ and $*$ are *inductive* in that they define unique functions $+$ and $*$ over any model $\langle X, s, 0\rangle$ of the second-order induction axiom I_2

$$\forall P[P(0)\&\forall x[P(x) \supset P(s(x))] \supset \forall x P(x)].$$

Furthermore, if A_{1-1} is the axiom asserting that s is one-to-one, then $E_N \cup \{I_2, A_{1-1}\}$ entails precisely those formulas true in $\mathbf{N}$ because $\mathbf{N}$ is its only model (up to isomorphism).

Despite its strength, the induction axiom I_2 has a major drawback. Since the first-order theory of $\mathbf{N}$ is undecidable (by Church's theorem), there is no effective way to enumerate the consequences of this axiom. For this reason, I_2 is generally replaced by an infinite set I_1 of axioms obtained by substituting an arbitrary first-order formula ϕ for P in I_2. By Godel's Completeness Theorem, there is an effectively enumerable proof relation $\vdash$ such that ψ is a semantic consequence of T if and only if $T \vdash \psi$; hence there is an effective way to enumerate the consequences of I_1. However, $E_N \cup I_1 \cup \{A_{1-1}\}$ no longer characterizes $\mathbf{N}$; in fact, by Godel's Incompleteness Theorem, there are formulas ψ that are true of $\mathbf{N}$, but cannot be proved from this or any other recursive set of axioms true in $\mathbf{N}$. So, in going from second-order logic to first-order logic, one gives up expressive power in exchange for an effective notion of proof.

Equational logic is the subset of first-order logic in which the only formulas are equations $t_1 = t_2$ between two terms t_1 and t_2. Here one again trades expressive power for a simpler notion $\vdash_e$ of equational proof. By Birkhoff's Completeness Theorem, an equation e is true in all models of a set E of equations if and only if it can be derived from E by substituting equals

for equals. Furthermore, one can "compile" decision procedures for some interesting equational theories, e.g., by using the Knuth-Bendix completion procedure [19] to produce a convergent term-rewriting system.

Again one has given up something in exchange for a simpler notion of proof. In first-order logic, the induction axioms I_1 enable us to prove equations such as $x*y = y*x$, which do not follow from E_N alone. In equational logic, one cannot formulate induction as an axiom or as a set of axioms. Instead, to prove equations such as $x*y = y*x$, one must formulate induction as a rule of inference, defining a notion $\vdash_i$ of inductive proof that extends the notion of $\vdash_e$ of equational proof. An obvious rule would be to infer an equation $t_1(x) = t_2(x)$ from the infinitely many equations $t_1(0) = t_2(0)$, $t_1(s(0)) = t_2(s(0))$, $t_1(s(s(0))) = t_2(s(s(0)))$, ... However, this rule is not effective since it has infinitely many premises. An effective rule, sometimes known as *structural induction* [4], concludes

$$E \vdash_i t_1(x) = t_2(x)$$

from the *basis step*

$$E \vdash_i t_1(0) = t_2(0)$$

and the *induction step*

$$E \cup \{t_1(c) = t_2(c)\} \vdash_i t_1(s(c)) = t_2(s(c))$$

where the *induction hypothesis* $t_1(c) = t_2(c)$ is formed using a constant, c, not appearing in E, t_1, or t_2.

With this definition of $\vdash_i$, it is easy to give inductive proofs of the associativity and commutativity of $+$ and $*$, as well as the distributivity of $*$ over $+$. In fact, any equation true in $\mathbf{N}$ can be derived from E_N using $\vdash_i$ (since any equation true in $\mathbf{N}$ is a consequence of the axioms for commutative rings [15]). However, it can be shown that the equational theories of many other interesting abstract data types are undecidable. Hence no effective rule of inductive inference will enable us to prove all equations in these theories. So, as in first-order logic, one is forced to live with incompleteness. Of course, this does not stop us from trying to prove equations using induction. However, one must be careful to distinguish between the inductive theory of a set of equations and the inductive theorems that can be proved from those equations using particular inductive rules of inference.

Since our focus here is on abstract types, we work in a multisorted equational logic that enables us, for example, to axiomatize lists of integers. All

functions have signatures, and all variables are sorted. We identify a set G of functions as *generators*, and we call a sort *generated* if at least one generator has that sort as its range. When considering **N**, one might identify 0 and s as generators; alternatively, one might identify 0, 1, and + as generators.

This approach accommodates a wide range of proofs by induction. It disentangles using a sort for sort checking from proving properties about subsets of the values of a particular sort. Generators need not generate an entire sort; thus, one can choose 1 and s as generators if one wishes to prove that an equation is true over the positive integers. Not every sort need be generated, and generators of a sort S can have an arbitrary number of arguments of any sort. For example, lists can be generated using the functions `null` and `cons`, without having to specify generators for the element sort, and sets can be generated by `empty`, `insert`, and `union`.

The *equational theory* of a set E of equations consists of those equations true in all models of E) (we assume that any model contains at least one element of each sort). To define the inductive theory of E, we require that there be at least one G-closed term of each generated sort, where a term is *G-closed* if all of its function symbols are generators in G and none of its variables are from generated sorts. The *inductive theory* of E with respect to G consists of those equations true in all models of E when the variables of each generated sort are restricted to range over the interpretations of the G-closed terms of that sort.

The inductive theory can be characterized syntactically as well as model theoretically. We call a substitution *G-closed* if it is the identity on variables that are not from generated sorts and replaces all variables from generated sorts by G-closed terms. The inductive theory of E with respect to G consists of those equations for which every instantiation under a G-closed substitution is in the equational theory of E.

The *inductive theorems* (as opposed to theory) of E with respect to G are those that can be proved using the following rule of inference, which generalizes the induction rule for **N**. For notational simplicity, we assume that any generator of a sort S has at most one argument of sort S. This restriction is easily removed, and is not imposed on the rule implemented in LP. Let $b_1, \ldots, b_m$ be the (basis) generators of sort S that have no arguments of sort S, and let $g_1, \ldots, g_n$ be the remaining (inductive) generators of sort S. Suppose $t_1(x)$ and $t_2(x)$ are two terms involving a variable x of sort S, and let c be a constant of sort S that does not appear in t_1, t_2, or E. Then

$$E \vdash_i t_1(x) = t_2(x)$$

if

$$E \vdash_i t_1(b_k) = t_2(b_k)$$

for $k = 1, \ldots, m$, and

$$E \cup \{t_1(c) = t_2(c)\} \vdash_i t_1(g_k(c)) = t_2(g_k(c))$$

for $k = 1, \ldots, n$ (here $g_k(c)$ is the result of using c as the argument of sort S in g_k; the other arguments of b_k and g_k are filled by fresh variables).

The rule $\vdash_i$ is sound; i.e., all inductive theorems of E with respect to G are in the inductive theory of E with respect to G. This can be shown by induction, first over the number of uses of $\vdash_i$ in a proof and then over the interpretations of the G-closed terms models of E.

Inductionless induction provides another method for establishing that an equation belongs to the inductive theory of E provided that E is *Hilbert-Post complete* and consistent. This method is based on the following notion of "proof by consistency" in first-order logic. There, a set A of axioms is Hilbert-Post complete if $A \vdash \psi$ for any formula ψ consistent with A. Hence, if A is Hilbert-Post complete, one can show that $A \vdash \psi$ by showing that $A \cup \{\psi\}$ is consistent, i.e., that it does not entail $\phi \& \not\phi$ for any formula ϕ. This method of proof is almost never used in first-order logic because consistency is usually hard to prove.

Inductionless induction carries this style of proof over into equational logic, where consistency is potentially easier to show, but harder to define. By analogy, a set E of equations is Hilbert-Post complete if $t_1 = t_2$ is in the inductive theory of E whenever $E \cup \{t_1 = t_2\}$ is consistent. The difficulty is that since formulas cannot be negated in equational logic, the definition of consistency used in first-order logic is not immediately transferable. The usual algebraic definition is that a set of equations is consistent if and only if it has a nontrivial model, i.e., its initial model has more than one element. In logical terms, E is is consistent if its inductive theory does not include $x = y$, where x and y are variables. (In the multisorted case the definition becomes more complicated.)

Since the usual algebraic notion of consistency is hard to check directly, various methods of inductionless induction generally employ stronger notions. They require that certain "forbidden" equations among generators not follow equationally, and they use completion procedures similar to Knuth-Bendix to detect these equations. As the number of forbidden consequences

increases, the notion of consistency becomes stronger and Hilbert-Post completeness becomes easier to show; but the number of inductive theorems decreases because consistency is rarer.

Demonstrations of Hilbert-Post completeness usually involve two components: first, showing that every variable-free term is provably equal to a variable-free term that contains only generators, i.e., that E is *sufficiently complete* for G [12]; and second, using the chosen definition of consistency to show that the equations in E involving only generators are Hilbert-Post complete (with respect to equations involving only generators). Huet and Hullot [16], for example, forbid any equation relating generators, thus trivializing the second component. Paul [25] and Lazrek, Lescanne and Thiel [21] take a more general approach. They allow the equational theory of E to contain equations relating generators, but forbid any such equations that are not already equational consequences of E. Jouannaud and Kounalis [17] replace the requirement of Hilbert-Post completeness by a related, but more flexible, requirement of inductive reducibility.

Section 4 provides more details about how the completeness and consistency checks needed in proofs by inductionless induction are done in practice.

3 Induction in LP

LP is a theorem prover based primarily on equational term-rewriting. It departs from conventional equational term-rewriting in two important ways.

- It supports a variety of proof methods beyond normalization and completion, e.g., induction, contradiction, and case analysis.
- It allows users to supply non-equational rules of inference. Some of these are applied automatically and some of them are used in conjunction with various proof methods.

In this paper, we discuss only two of LP's proof methods, traditional and inductionless induction. We also limit our discussion of user-supplied rules of inference to rules pertaining to induction.

LP supports two methods of proof by induction, traditional and inductionless induction. The method of inductionless induction was inherited from LP's predecessor, REVE [8, 22], and is based upon the work described in [16]. The method of traditional induction is explained in this section.

The basis for proofs in LP is a set of rewrite rules together with assertions about the operators appearing in them, e.g., that + is associative-commutative. (Logically, these assertions are merely abbreviations for equations. Operationally, they are treated quite differently. Equational term-rewriting is used to avoid nonterminating rules such as $x + y \rightarrow y + x$.) Frequently the rules have been obtained by ordering and completing a set of equations, but the soundness of proofs in LP (other than by inductionless induction) does not require that the rules be either terminating or Church-Rosser. Thus our proofs deal with the consequences of a set of rules ($\rightarrow^*$) rather than a set of equations ($=^*$).

For proofs by induction, one lists the generators of those sorts, together with their signatures, over which one may want to do induction. This information has no impact on sort checking or the rewriting theory, but does define the inductive theory as described in Section 2. For example, to prove theorems about lists, one might start with the following generators and rewrite rules.

```
Generators for sort 'list':
Basis:
       null: -> list
Inductive:
       cons: elem list -> list
Rewrite rules:
1.  append(null, x) -> x
2.  append(cons(x, y), z) -> cons(x, append(y, z))
3.  rev(null) -> null
4.  rev(cons(x, y)) -> append(rev(y), cons(x, null))
```

To carry out a proof by induction, one enters an equation together with the name and sort of the variable over which induction is to be done. LP then pushes this equation onto a stack of proofs-in-progress, generates the appropriate equations for the basis step in the inductive proof, pushes them onto the stack, and attempts to prove the top equation on the stack. After the validity of the basis equations has been shown, LP generates the appropriate structural induction hypotheses as described in Section 2 (there will be more than one if a generator has two arguments of the generated sort). If an induction hypothesis can be ordered, it is added to the set of rules; otherwise it is left as an equation. Then LP pushes onto the stack of proofs-in-progress the equations that must be proved in the induction

step and attempts to prove them. After the validity of these equations has been shown, LP deletes any rules and equations that involve the constants added to form the induction hypothesis, and it asks whether the user wants the now-established inductive theorem added to the rewriting system. The following edited transcript shows how this mechanism can be used to prove a simple theorem about lists (the >> in the transcript is a prompt issued by LP).

```
>> prove append(x, null) = x  by induction x list

Basis step [proved]:
   append(null, null) = null

Induction hypothesis: append(c0, null) = c0
Ordered into: append(c0, null) -> c0

Induction step [proved]:
   append(cons(v0, c0), null) = cons(v0, c0)

The equation  append(x, null) = x
IS an inductive theorem.
```

In this proof, equational reasoning suffices to establish the basis and induction steps. Frequently, some other form of reasoning is required (e.g., case analysis or a nested induction), or a lemma must be established for the proof to go through. LP's proof management facility allows for this.

LP also allows multilevel induction. Consider, for example, the natural numbers. Some theorems are most easily proved using both $t_1(c) = t_2(c)$ and $t_1(s(c)) = t_2(s(c))$ as induction hypotheses. In first-order logic, such properties can be proved by using axioms of "complete" induction. To use n-level induction in LP to prove a theorem $\phi(x)$, where x is of sort N, one first proves $\phi(0)$, ..., $\phi(s^{n-1}(0))$ as the basis of the induction and then assumes $\phi(c)$, ..., $\phi(s^{n-1}(c))$ as induction hypotheses for proving $\phi(s^n(c))$.

In many ways, traditional induction in LP resembles the way induction is handled by the Boyer-Moore Theorem Prover [3]. The context in which it is used, however, is rather different. Boyer-Moore uses various heuristics to derive subgoals automatically from the conjecture to be proved. In contrast, LP incorporates almost no heuristics and relies largely on forward rather than backward inference. As discussed in Section 4.6, the user rather than

the program is responsible for inventing useful lemmas. Another important distinction is that while both rely heavily on rewriting, the rewriting technology used is quite different. LP is built around state-of-the-art equational term-rewriting machinery. Particular attention has been devoted to the application of termination orderings, completion procedures, and combining equational theories. The rewriting machinery in Boyer-Moore antedates much important work in these areas.

4 A comparison of traditional induction with inductionless induction

While it is dangerous to draw conclusions from limited data, we believe that in some respects traditional induction is clearly superior to inductionless induction. Some of the advantages of traditional induction are discussed in the remainder of this section. This discussion includes a number of examples, all of which were run using LP.

4.1 Simpler theoretical basis

In traditional induction, the soundness of different rules of induction is generally clear; at the very least, it is easy to prove soundness by straightforward inductive arguments. Furthermore, if two rules of induction are separately sound, it is sound to use them together.

In inductionless induction, the soundness of proofs based on various notions of consistency is less clear. Inductionless induction is less logical than traditional induction, i.e., soundness is usually established by algebraic techniques. For certain sets E of equations and G of generators, the inductive theory of E with respect to G is just the set of equations true in the initial model of E. This happens when every closed term is variable-free and E is sufficiently complete for G. For other sets of equations and generators the soundness of inductionless induction is harder to show. Furthermore, it is not so clear how to use different formulations of inductionless induction in conjunction with one another.

Another indication of their simple theoretical basis is that traditional inductive proofs are monotonic with respect to axiomatizations. If P is provable from E, then P is provable from any superset of E. Inductionless induction is not monotonic in this respect. The reason for this is discussed in Section 4.3.

4.2 Completeness of presentation not required for soundness

No notion of completeness is required to ensure the soundness of traditional induction. In contrast, the soundness of inductionless induction depends upon a rather strong notion of completeness.

Consider an equational theory E and an equation $t_1 = t_2$. In inductionless induction one demonstrates the validity of $t_1 = t_2$ by showing that it is consistent with E. The soundness of this method depends upon the Hilbert-Post completeness of the inductive theory of E, i.e., on there being no independent variable-free equation that can be added consistently to E (see Section 2). This presents two problems: E may not be Hilbert-Post complete; and, even if it is, one may have to go to some trouble to prove it.

It is frequently useful to reason about equational theories that are not Hilbert-Post complete. In specifying abstract types, for example, it is good practice to to avoid placing unnecessary constraints on implementations, e.g., on the properties of elements in a set abstraction. Therefore, much of the time one reasons from specifications that are not complete in any logical sense. The following example was extracted from such a specification.

Consider the inductive theory with the following presentation.

```
min(0, x) -> 0
min(s(x), s(y)) -> s(min(x, y))
least(insert(empty, e)) -> e
least(insert(insert(s, e), e1)) ->
           min(least(insert(s, e)), e1)

Commutative: min

Generators:  empty  : -> set
             insert : set nat -> set
```

It is easy to prove the theorem

```
least(insert(s, 0)) = 0
```

by induction over `s`. The same theorem can also be proved by inductionless induction, but the proof would not be sound, since the equations are not Hilbert-Post complete. Proofs of `least(empty) = 0` would also "work" by inductionless induction, as would proofs of `least(empty) = s(0)`.

4.3 Convergent rewriting systems not required

Proving consistency is the heart of inductionless induction. In practice this means that one must be able to transform the original set of equations plus the theorem to be proved into a convergent term-rewriting system, i.e., a set of rewrite rules that is both terminating (Noetherian) and Church-Rosser. While programs for doing this are getting ever more sophisticated, it can still be a nontrivial task. Furthermore, there are interesting equational theories for which convergent term-rewriting systems do not exist, e.g., any undecidable theory.

The requirement of convergence leads inductionless induction to behave nonmonotonically. It may be possible to prove an equation e from a set E of equations, but not from $E \cup \{e'\}$ because the augmented system cannot be translated into a convergent term-rewriting system.

By contrast, when using traditional induction it is only necessary to order the original equations, or even a subset of the original equations, into rewrite rules. It is not necessary that the system be convergent. We have frequently taken advantage of this, e.g., when it has been impractical, if not impossible, to derive convergent systems from the axioms supplied.

4.4 Nonterminating rewriting systems not fatal

Inductionless induction cannot be used unless $\rightarrow^*$ is Noetherian, since the soundness of the Knuth-Bendix procedure depends upon this property. The soundness of traditional induction does not depend on the Noetherian property. Its utility, however, frequently depends upon the rewriting strategy used, e.g., upon whether the rewriting mechanism explores fairly the different ways of rewriting terms.

When $\rightarrow^*$ is Noetherian, one still has to worry about whether the original system plus the equation to be proved yields a Noetherian system. In inductionless induction, if the conjecture cannot be ordered in such a way that the original system plus the equation to be proved forms a Noetherian system, one cannot proceed at all.

In traditional induction, there are two ways theorems may be proved when the induction hypothesis cannot be ordered to yield a Noetherian system. First, there are theorems whose proof requires induction but not explicit use of the induction hypothesis. For example, using the rules for sets from Section 1 one can use a nested induction to prove (by contradiction) that the equation

```
subset(x, y) = subset(y, x)
```

is not valid. (That inductionless induction can be used for showing invalidity is an oft-touted virtue. This example illustrates that traditional induction can also be used for this purpose, and that it can be used in situations where unorderable equations prevent the use of inductionless induction.)

Second, there are theorems that can be proved by ordering and using the induction hypothesis despite the fact that it yields a nonterminating rewriting system. This can occur, for example, when a fair rewriting strategy is used. The rewriting strategy used in LP is not fair, but does ensure that the induction hypothesis is among the last rules tried.

In addition to supporting several methods for automatically ordering equations into terminating rewriting systems, LP allows the user to create potentially nonterminating sytems with the "left-to-right" ordering. As the following edited transcript illustrates, this enables one to use induction to prove properties such as the commutativity of + (rule 3 in that transcript was proved by induction from rules 1 and 2).

```
Generators for sort 'N':
Basis:
       0: -> N
Inductive:
       s: N -> N
Rewrite rules:
1.  x + 0 -> x
2.  x + s(y) -> s(x+y)
3.  0 + x -> x

>> set ordering left-to-right

>> prove x + y = y + x by induction y N

Basis step [proved]:
   x + 0 = 0 + x

Induction hypothesis: x + c1 = c1 + x
Ordered into: x + c1 -> c1 + x

Induction step [to prove]:
```

```
   x + s(c1) = s(c1) + x

>> resume by induction x N

Basis step [proved]:
   0 + s(c1) = s(c1) + 0

Induction hypothesis: c2 + s(c1) = s(c1) + c2
Ordered into: s(c1 + c2) -> s(c1) + c2

Induction step [proved]:
   s(c2) + s(c1) = s(c1) + s(c2)

The induction step  x + s(c1) = s(c1) + x
HAS been proved.

The equation  x + y = y + x
IS an inductive theorem.
```

4.5 Proofs easier to follow

In traditional induction, all steps are deductive and contribute to the proof in an obvious fashion. In contrast, an inductionless induction proof is not deductive. It succeeds by failing to derive a contradiction. To follow the intermediate steps in a proof by inductionless induction, one must understand the completion procedure being used. In the case of equational term-rewriting, this procedure can be rather complex.

4.6 Easier to see why a proof has failed

A proof by traditional induction either succeeds or halts with a finite number of irreducible equations. These equations are frequently suggestive of lemmas that will allow the proof to succeed. For example, when one attempts to use the presentation of sets appearing at the start of Section 1 and traditional induction to prove the reflexivity of the subset relation, LP reduces the proof to the equation

```
subset(c0, union(c0, c1)) &
    subset(c1, union(c0, c1)) = true
```

This suggests the lemma

```
subset(s, union(s, s1)) = true
```

which can proved by induction and is sufficient to complete the proof. As is frequently the case, the lemma is a generalization of the irreducible equation. (This is similar to the way generalizations are generated in the Boyer-Moore theorem prover.)

An attempt to prove the above theorem by inductionless induction fails by diverging. By examining the critical pairs generated by the Knuth-Bendix completion procedure, one can infer the necessity of the above lemma. In general, the problems with doing this are knowing when to halt the attempt to complete the system and then deciding which of the critical pairs generated will lead to a useful lemma. An added complication is that the critical pairs generated in a diverging proof by inductionless induction depend upon details of the completion process used. In some implementations a critical pair that is suggestive of a useful lemma may be generated quickly; in others it may not be generated at all.

4.7 Proofs easier to control

A purported advantage of inductionless induction is that it requires no user interaction. In our experience, this is not true. Starting the proof requires no user interaction, but getting it to succeed often involves considerable user interaction. Among other things the user may have to take one of the following actions.

- Interrupt the completion procedure to add a necessary lemma. Deciding when to do this can be problematic.
- Provide information about how to order a critical pair equation into a rewrite rule to preserve termination. For equational term-rewriting, in particular, this may be rather difficult.

In traditional induction, the points at which the user interacts with the proof are well defined. While the user may have to supply information about how to order an induction hypothesis, our experience indicates that this is easier than ordering critical pairs generated by the completion procedure. In addition, the user can induct over different well-founded relations without having to change the underlying term-rewriting system. To change generators in a proof by inductionless induction, one must reconvert the original

set of equations into a convergent term-rewriting system using a different ordering based on the new set of generators.

An additional problem with inductionless induction is that, before attempting to prove a lemma, one must back out of the the proof-in-progress that exposed the need for the lemma. If this isn't done, one may derive an inconsistency even though the lemma is valid, or prove the validity of an invalid lemma using equational consequences of the original conjecture. This issue does not arise in traditional induction. Any equations derived in attempting to prove a conjecture can be used safely when working on a lemma.

5 Summary

The inductive theory of a set of equations is generally larger than the set of theorems provable using any particular method of induction. The nature of the gap depends upon the set of equations and the method of proof.

For reasoning about abstract data types, traditional induction seems superior to inductionless induction. It is mathematically simpler, more generally applicable, and easier to control. When it succeeds, the proof is easier to follow. When it fails, it is easier to see why.

Recent advances suggest that equational term-rewriting may prove useful in many applications. In LP we are trying to combine these advances with traditional proof techniques. Our hope is that the basic proof techniques employed can remain orthogonal to each other and to improvements in rewriting technology. In this light, we find our experience with traditional induction very encouraging. Recently we have added several other proof techniques to LP and made significant additions to the underlying deductive mechanism. The interactions between these additions and inductionless induction raise difficult theoretical and practical problems. In contrast, they have combined easily with traditional induction to enhance the power of LP.

6 Acknowledgments

Katherine Yelick and Pierre Lescanne provided helpful advice and many interesting examples during the early stages of this work. James Horning and James Saxe used several of versions of LP, uncovering various problems and suggesting many improvements.

References

[1] Backus, J., Williams, J. H., and Wimmers, E. L. *FL Language Manual (Preliminary Version)*, IBM Almaden Research Center Research Report, 1986.

[2] Bellegarde, F. "Rewriting Systems on FP Expressions to Reduce the Number of Sequences Yielded," *Science of Computer Programming* 6 (1986), 11–34.

[3] Boyer, R. S. and Moore, J. S. *A Computational Logic*, Academic Press, 1979.

[4] Burstall, R. "Proving properties of programs by structural induction," *Computer J.* 12:1 (1969), 41–48.

[5] Burstall, R. and Goguen, J. "Putting Theories Together to Make Specifications," *Proceedings Fifth IJCAI*, 1977.

[6] Choppy, C. and Johnen, C. "Petrireve: Proving Petri Net Properties with Rewriting Systems," *Proc. First Int'l Conf. on Rewriting Techniques and Applications, Lecture Notes in Computer Science* 202, Springer-Verlag, 1985, 271–286.

[7] Cosmadakis, S. S. and Kannellakis, P. C. "Two Applications of Equational Theories to Data Base Theories," *Proc. First Int'l Conf. on Rewriting Techniques and Applications, Lecture Notes in Computer Science* 202, Springer-Verlag, 1985, 107–123.

[8] Forgaard, R. and Guttag, J. V. "REVE: A Term Rewriting System Generator with Failure-Resistant Knuth-Bendix," *Proceedings of an NSF Workshop on the Rewrite Rule Laboratory*, General Electric Corporate Research and Development Report No. 84GEN008, Schenectady, NY, April 1984, 5–31.

[9] Fribourg, L. "Oriented Equational Clauses as a Programming Language," *Journal of Logic Programming* 1 (1984), 165–177.

[10] Futatsugi, K., Goguen, J. A., Jouannaud, J.-P., and Meseguer, J. "Principles of OBJ2," *Proceedings of the 12th ACM Conference on Principles of Programming Languages*, 1985.

[11] Goguen, J. A. "How to Prove Algebraic Inductive Hypotheses without Induction, with Applications to the Correctness of Data Type Implementations," *Proc. of the Fifth Conference on Automated Deduction*, Les Arcs, France, July 1980, *Lecture Notes in Computer Science* 87, New York, Springer-Verlag, 356–373.

[12] Guttag, J. V. and Horning, J. J. "The Algebraic Specification of Abstract Data Types," *Acta Informatica* 10 (1978), 27–52.

[13] Guttag, J. V. and Horning, J. J. "Report on the Larch Shared Language" and "A Larch Shared Language Handbook," *Science of Computer Programming* 6:2 (Mar. 1986), 103–157.

[14] Guttag, J. V., Horning, J. J., and Wing, J. M. "An Overview of the Larch Family of Specification Languages," *IEEE Software* 2:5 (Sept. 1985), 24–36.

[15] Henkin, L. "The Logic of Equality," *American Mathematical Monthly* 84 (1977), 597–612.

[16] Huet, G. and Hullot, J. M. "Proofs by Induction in Equational Theories with Constructors," *J. Computer and System Sciences* 25:2 (Oct. 1982), 239–266.

[17] Jouannaud, J.-P. and Kounalis, E. "Proof by Induction in Equational Theories without Constructors," *Proc. Second IEEE Symposium on Logic in Computer Science*, Cambridge, MA, June 1986, 358–366.

[18] Kapur, D. and Musser, D. R. "Proof by Consistency," *Artificial Intelligence* 31 (1987), 125–157.

[19] Knuth, D. E. and Bendix, P. B. "Simple Word Problems in Universal Algebras," in *Computational Problems in Abstract Algebra*, J. Leech (ed.), Pergamon Press, Oxford, 1969, 263–297.

[20] Lankford, D. S. "A Simple Explanation of Inductionless Induction," MTP-14, Louisiana Technical University, 1981.

[21] Lazrek, A., Lescanne, P., and Thiel, J.-J. "Proving Inductive Equalities: Algorithms and Implementation," Report 86-R-087, Centre Recherche en Informatique de Nancy, France, September 1986.

[22] Lescanne, P. "Computer Experiments with the REVE Term Rewriting System Generator," *Proceedings of the Tenth ACM Conference on Principles of Programming Languages*, Austin, January 1983, 99–108.

[23] Musser, D. R. "On Proving Inductive Properties of Abstract Data Types," *Proceedings of the Seventh ACM Conference on Principles of Programming Languages*, Las Vegas, January 1980, 154–162.

[24] Musser, D. R. "Abstract Data Type Specification in the AFFIRM System," *IEEE Transactions on Software Engineering* 6 (1980).

[25] Paul, E. "Proof by Induction in Equational Theories with Relations between Constructors," *Proceedings of the Ninth Colloquium on Trees in Algebra and Programming*, Bordeaux (1984), Cambridge University Press, 210–225.