Abstract

Several inductive provers have been developed to automate inductive proofs (see for instance: Nqthm, RRL INKA, LP, SPIKE, CLAM-Oyster, ...). However, inductive theorem provers very often fail to terminate. A proof to go through requires either additional lemmas, a generalization, a suitable induction variable to induce upon, or a case split.

The aim of this paper is to present a simple and powerful heuristic that allows to overcome, in many cases, the divergence of induction provers when working with conditional theories. We first provide a new definition of induction variables and then formalize a new transition rule for induction (named CGT-rule). The essential idea behind it is to propose a generalized form of the conclusion just before another induction is attempted and failure begins. This generalized form is based on the induction hypothesis and the current goal.

CGT-rule enables to prove many theorems completely automatically from the functions definitions alone. We illustrate computer applications to the correctness proof of the insertion sorting algorithm and other programs computing on lists and numbers. All of them have never proved before without user-provided generalizations and/or lemmas.

(Content Areas: Automated Reasoning, Theorem Proving)

Introduction

In this section, we first state the problem of induction, then explain our interest in automated induction, and finally discuss the aims of the present paper and related work.

The Problem

We assume familiarity with the basic notions of equational logic and rewrite systems (see for instance (Dershowitz and Jouannaud 91)). For simplicity of notation, we assume have only one sort; all the results carry over to many-sorted case without difficulty.

Let \( \mathbb{A} \) be a set of conditional equations, i.e., expressions of the form \( e_1 \wedge ... \wedge e_n \Rightarrow e_{n+1} \), where \( e_i \) are equations which are usually written as \( t = s, e_1, ..., e_n \) are the premises and \( e_{n+1} \) is the conclusion of \( e_1 \wedge ... \wedge e_n \Rightarrow e_{n+1} \). A clause is an expression of the form \( \forall e_1 \wedge ... \forall e_n \vee e_{n+1} \vee e_{n+2} \) where \( e_1, ..., e_n, e_{n+1}, ..., e_{m} \) are equations. We shall sometimes write this expression in the following equivalent way: \( e_1 \wedge ... \wedge e_n \Rightarrow e_{n+1} \vee ... \vee e_m \). We identify a conditional equation and its corresponding representation as a Horn clause. A clause \( \phi \) is an inductive consequence (inductively valid) of a set \( \mathbb{A} \) of axioms if it is valid in the initial model. This will be denoted by \( \mathbb{A} \models_{\text{ind}} \phi \). Inductive consequences are related to deductive consequences as follows: \( \mathbb{A} \models_{\text{ind}} \neg e_1 \vee ... \vee \neg e_n \vee e_{n+1} \vee \ldots \vee e_m \) if and only if for any ground substitution \( \sigma \), (for all \( i : \mathbb{A} \models_{\text{ind}} e_i \sigma \)) implies (there exists \( j \) such that \( \mathbb{A} \models_{\text{ind}} e_j \sigma \)). For a detailed discussion of initial models, see (Padawitz 88). Since ground terms can easily be well-ordered, induction can be used as a natural technique to prove clauses in the initial model. Unfortunately, there is no a simple proof theory which captures the semantic notion of the initial model. To overcome this problem, one uses formula schemata to formulate an induction rule which is used to prove formulas in the initial model. Moreover, no (recursively axiomatizable) set of induction schemata is strong enough to constitute a complete proof theory for the initial model.

To establish inductive consequences, classical theorem proving provides either explicit induction ((Boyer and Moore 79), (Aubin 79), (Zhang, Kapur, and Krishnamoorthy 88),...), or implicit induction (Dershowitz 82), (Jouannaud and Kounalis 86 and 90), (Kounalis and Rusinowitch 90), (Reddy 90), (Bouhoula, Kounalis, and Rusinowitch 95), (Bouhoula and Rusinovitch 95), (Bonsard, Hasker, and Reddy 96).

However, the hardest problems in using either approaches is to find an appropriate induction hypothesis and/or the lemmas needed for the proof. Consequently, proving conjectures in the initial model requires “eureka steps” such as selection of a suitable variable to induce upon, additional lemmas, a generalization, or case analysis for the proof to go through. In some cases a combination of these concepts is needed. In this paper we develop and implement a framework for automating, in some cases, these "eureka steps".

Motivation

The need to be able to prove inductive theorems appears in many applications including number theory, program verification, and program synthesis. For example:

Conditional equations may be used to define arithmetic functions and express suitable properties about these functions. Consider, for instance, the theory \( \mathbb{A} \) of nonnegative_integers with addition, multiplication and exponentiation: \( \mathbb{A} = \{ x + 0 = x, x + S(y) = S(x + y), x * 0 = 0, x * S(y) = x + (x * y), \exp(x, 0) = S(0), \exp(x, S(y)) = x * \exp(x, \ldots \} \).
Outline of our Approach: an Example

To illustrate the essential ideas behind our method let us outline it on a rather complicated conjecture. Suppose our goal is to prove the reflexivity property of the set inclusion relation,

\[ (*) D(L) = true \Rightarrow SUB(L, L) = true \]

This is part of a simple program verification problem. Consider the following axiomatization \( \mathcal{A} \) of subset \( (\subseteq) \), element inclusion \( (\in) \), all different elements \( (D) \), wherein a set is represented as a list \( L \): 

\[
\begin{align*}
\epsilon(x, L) &= true \Rightarrow SUB(x, L) = SUB(L, x) \\
\epsilon(x, L) &= false \Rightarrow SUB(x, L) = false \\
SUB(\emptyset, L) &= true \\
E(x, c) &= true \Rightarrow \epsilon(x, L) = true \\
E(x, c) &= false \Rightarrow \epsilon(x, L) = false \\
\epsilon(x, L) &= true \Rightarrow D(c, L) = false \\
\epsilon(x, L) &= false \Rightarrow D(c, L) = D(L) \\
D(\emptyset) &= true \\
EQ(x, x) &= true \\
true &= false \Rightarrow false \\
false &= true \Rightarrow false 
\end{align*}
\]

Here, EQ is the identity comparison operator for elements. Note that this simple conjecture is problematic for all automated proof systems (e.g., Nyqthm, RRL, INKA, LP, SPIKE, CLAM-Oyster, etc...). None of them can prove it when given just the above definitions. Of course, with the addition of some lemmas all provers are able to prove this. Also, this example cannot be treated by any of the heuristics proposed in heuristic (Kapur and Subramaniam 96), (Ireland and Bundy 96 and 96a), (Walsh 94), (Basin and Walsh 92), (Bundy et al. 93). Let see how our method allows an automatic proof of it.

To carry out a proof of \( (*) \) we first supply a well-founded ordering \( > \) on terms (i.e., there is no endless descending chain \( t_1 > t_2 > \ldots \)) of terms. For example, we may compare terms by using the lexicographic path order \( > \) generated by the precedence \( SUB >_p D >_p \in >_p \). Then \( SUB(L, L) > true \) holds in the lexicographic path order even though the two terms presumably have the same semantics (see Dershowitz 87). Using a monotonic and stable well-founded ordering \( > \) on terms we may apply a (conditional) equation for simplification.

We then compute a Test Set \( TS(\mathcal{A}) \) for \( \mathcal{A} \), i.e., a finite set of terms which, in essence, is a finite description of the initial model of \( \mathcal{A} \) (see for instance: Kounalis and Rusinovich 90), (Kounalis 92)). For example, the set \( TS(\mathcal{A}) = \{x, L, D, true, false\} \) constitutes a suitable test set for \( \mathcal{A} \). Note that every ground term is equal to a term over \( SUB, \in, D, \emptyset, true, false \). The reason for considering test sets is to instantiate their elements to variables in a conjecture in order to create the induction shemes for the proof of a conjecture.

Finally, a set of transitions rules (see subsection 4.1) is applied to the conjectures; The generate rule that allows to derive the induction schemes by instantiating an induction variable (see subsection 4.2) with elements of \( TS(\mathcal{A}) \). The case

Outline of our Approach: an Example

To illustrate the essential ideas behind our method let us outline it on a rather complicated conjecture. Suppose our goal is to prove the reflexivity property of the set inclusion relation,

\[ (*) D(L) = true \Rightarrow SUB(L, L) = true \]

This is part of a simple program verification problem. Consider the following axiomatization \( \mathcal{A} \) of subset \( (\subseteq) \), element inclusion \( (\in) \), all different elements \( (D) \), wherein a set is represented as a list \( L \):

\[
\begin{align*}
\epsilon(x, L) &= true \Rightarrow SUB(x, L) = SUB(L, x) \\
\epsilon(x, L) &= false \Rightarrow SUB(x, L) = false \\
SUB(\emptyset, L) &= true \\
E(x, c) &= true \Rightarrow \epsilon(x, L) = true \\
E(x, c) &= false \Rightarrow \epsilon(x, L) = false \\
\epsilon(x, L) &= true \Rightarrow D(c, L) = false \\
\epsilon(x, L) &= false \Rightarrow D(c, L) = D(L) \\
D(\emptyset) &= true \\
EQ(x, x) &= true \\
true &= false \Rightarrow false \\
false &= true \Rightarrow false 
\end{align*}
\]

Here, EQ is the identity comparison operator for elements. Note that this simple conjecture is problematic for all automated proof systems (e.g., Nyqthm, RRL, INKA, LP, SPIKE, CLAM-Oyster, etc...). None of them can prove it when given just the above definitions. Of course, with the addition of some lemmas all provers are able to prove this. Also, this example cannot be treated by any of the heuristics proposed in heuristic (Kapur and Subramaniam 96), (Ireland and Bundy 96 and 96a), (Walsh 94), (Basin and Walsh 92), (Bundy et al. 93). Let see how our method allows an automatic proof of it.

To carry out a proof of \( (*) \) we first supply a well-founded ordering \( > \) on terms (i.e., there is no endless descending chain \( t_1 > t_2 > \ldots \)) of terms. For example, we may compare terms by using the lexicographic path order \( > \) generated by the precedence \( SUB >_p D >_p \in >_p \). Then \( SUB(L, L) > true \) holds in the lexicographic path order even though the two terms presumably have the same semantics (see Dershowitz 87). Using a monotonic and stable well-founded ordering \( > \) on terms we may apply a (conditional) equation for simplification.

We then compute a Test Set \( TS(\mathcal{A}) \) for \( \mathcal{A} \), i.e., a finite set of terms which, in essence, is a finite description of the initial model of \( \mathcal{A} \) (see for instance: Kounalis and Rusinovich 90), (Kounalis 92)). For example, the set \( TS(\mathcal{A}) = \{x, L, D, true, false\} \) constitutes a suitable test set for \( \mathcal{A} \). Note that every ground term is equal to a term over \( SUB, \in, D, \emptyset, true, false \). The reason for considering test sets is to instantiate their elements to variables in a conjecture in order to create the induction shemes for the proof of a conjecture.

Finally, a set of transitions rules (see subsection 4.1) is applied to the conjectures; The generate rule that allows to derive the induction schemes by instantiating an induction variable (see subsection 4.2) with elements of \( TS(\mathcal{A}) \). The case
There are two fundamental ideas behind our proof strategy. The first idea is to apply the generate rule on an induction variable (see definition 3). The second idea is to apply (if possible) the CGT-rule just before another induction (generate rule) is attempted. In our example the proof starts with the goal:

$(*) \neg D(l) = true \lor \text{SUB}(L,L) = true$

To handle clause $(*)$ we can use the generate rule: we first instantiate the induction variable $L$ with elements of the test set, $(x, \emptyset, true, false)$, and then simplify (using the rules of simplify and/or case analysis) the resulted instances. The first step gives us two distinct clauses,

1. $\neg D(\emptyset) = true \lor \text{SUB}(\emptyset, \emptyset) = true$
2. $\neg D(x, l) = true \lor \text{SUB}(x, l, l) = true$

Notice that the clause, $\in(x, x.l) = true \lor \in(x, x.l) = false$, is inductive valid in $A$. So the generate rule gives,

3. $\neg true = true \lor true = true$
4. $\in(x, x.l) = true \Rightarrow (\neg D(x,l) = true \lor \text{SUB}(x,l, l) = true)$
5. $\in(x, x.l) = false \Rightarrow (\neg D(x,l) = true \lor false = true$)

To make clauses more readable when the case analysis rule applies we use the notation $p \Rightarrow q$ where $p$ is the rule premise and $q$ the initial clause to which one simplification step was performed.

Conjecture 3 is a tautology, so is deleted. For conjectures 4 and 5 the case analysis rule gives,

41. $\neg D(x, y) = true \Rightarrow (\neg true = true \lor \neg D(x, l) = true \lor \text{SUB}(x, l, l) = true)$
42. $\neg D(x, y) = false \Rightarrow (\neg false = true \lor \neg D(x, l) = true \lor \text{SUB}(x, l, l) = true)$
43. $\neg D(x, y) = true \Rightarrow (\neg false = true \lor \neg D(x, l) = true \lor false = true)$
44. $\neg D(x, y) = false \Rightarrow (\neg true = true \lor \neg D(x, l) = true \lor false = true)$

The premise $\neg D(x, y) = false$ in clauses 42 and 52 is simplified using $A$ to $true = false$ and the result reduces by $A$ to a tautology by $A$. Clause 51 is a tautology. Clause 41 can be rewritten using the case analysis rule to

41.1. $\neg D(x, y) = true \Rightarrow (\neg false = true \lor \text{SUB}(x, l, l) = true)$
41.2. $\neg D(x, y) = false \Rightarrow (\neg D(l) = true \lor \text{SUB}(l, l, l) = true)$

Now clause 411 is reduced $A$ to a tautology. Application of CGT-rule to 412 gives (see definition 4),

6. $\neg D(x, W) = true \lor \neg D(W) = true \lor \text{SUB}(W, x, l) = \text{SUB}(W, l)$

This is the only clause proposed by CGT-rule. Roughly speaking CGT-rule works in two steps: at the first step it takes as argument an induction hypothesis and a derived clause from it (w.r.t. generate and simplification), and produces a new clause. This new clause is generated in such a way that both sides of a positive literal share non-trivial common subterms. Further, at least one of these subterms is at a position which is a suffix of inductive position of $R$ (see definition 1). At the second step the GGT-rule generalizes these subterms using a fresh variable as well as all occurrences of these subterms in the whole clause.

To handle clause 6 we can use the generate rule. The first step gives us two distinct clauses,

7. $\neg D(x, \emptyset) = true \lor \neg D(\emptyset, x) = true \lor \text{SUB}(\emptyset, x, l) = \text{SUB}(\emptyset, l)$
8. $\neg D(x, y, Z) = false \lor \neg D(y, Z) = true \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l)$

which are rewritten using $A$. Simplification of clause 7 gives a tautology. Simplification of clause 8 gives,

81. $\neg D(x, y) = true \Rightarrow (\neg true = false \lor \neg D(y, Z) = true \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l))$
82. $\neg D(x, y) = false \lor \neg D(y, Z) = true \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l))$

Now clause 81 is reduced by $A$ to a tautology. Using Case Analysis, clause 82 gives

821. $\in(y, x, l) = true \Rightarrow (\neg D(y, Z) = true \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l))$
822. $\in(y, x, l) = false \Rightarrow (\neg D(y, Z) = true \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l))$

Clauses 821 and 822 can be rewritten (case analysis), using $A$, to

821. $\neg D(y, Z) = true \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l)$
822. $\neg D(y, Z) = false \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l)$

Clauses 822 is a tautology. Using Case Analysis, we can reduce Clause 821 to

8211. $\in(y, Z) = true \Rightarrow (\neg D(y, Z) = true \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l))$
8212. $\in(y, Z) = false \Rightarrow (\neg D(y, Z) = true \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l))$

The first clause is reduced by $A$ to a tautology. Using the induction hypothesis (6), 8212 is reduced to

$\neg D(y, Z) = true \lor \neg D(y, Z) = false \lor \neg D(y, Z) = true \lor \text{SUB}(y, Z, x, l) = \text{SUB}(y, Z, l)$

which is a tautology. This completes the proof of initial conjecture.

**Induction in Conditional Theories : the Machinery**

In this section we first provide a set of transition rules for induction with an extension of the case analysis rule, then properly define the notion of induction variable, and finally
Transition Rules for Conditional Theories

The induction procedure is formalized by the following set of transitions rules (Bouhoua, Kounalis, and Rusinowich 95), (Bouhoula and Rusinowitch 95) applied to pairs of the form (E,H), where E is the set of clauses which are conjectures to be proved and H the set of inductive hypotheses. The initial set of conditional equations A is oriented with a well-founded ordering to a rewrite system R. The inference system for induction contains the transition rules given below. Roughly speaking, the generate rule allows to derive the induction schemes by instantiating an induction variable (see next subsection with elements of TS(R). These induction schemes are then simplified by the set of inductive hypotheses. The initial set of conditional rules is a kind of conjecture disprover. It is very important, for any function f with boolean values, the type a 1=b1 & ... & an=bn = s=r where for all i in [1..n], bi ∈ [true, false]. Conjectures can also be boolean clauses i.e., clauses whose negative literals are of the type ¬a=b where be [true, false]. If we assume that any defined function is completely defined, for any function f with boolean values, the following is inductively valid: f(t1,...,tn) = true. We can reformulate the complement rule as follows: COMPLEMENT: (E ∪ {¬(a=b) ∨ r }, H) |¬ (E ∪ {a=b ∨ r }, H)
if b ∈ [true, false].

Designing Good Induction Schemes

The generate rule captures the essence of induction. It allows to obtain the induction schemes that entails the proof of the conjecture under consideration. An induction scheme must be formed in the way that has a good chance of success. In some sense, the success of a proof is strongly depends to what variable the generate rule applies. Then a question naturally arises: What is the best variable to replace in order to be able to apply a definition and eventually the induction hypothesis? For example consider the following set R of conditional axioms that defines the intersection of two lists,

The induction procedure is formalized by the following set of transitions rules (Bouhoua, Kounalis, and Rusinowich 95), (Bouhoula and Rusinowitch 95) applied to pairs of the form (E,H), where E is the set of clauses which are conjectures to be proved and H the set of inductive hypotheses. The initial set of conditional rules is a kind of conjecture disprover. It is very important, when a convergent set of axioms exist for ground terms, to guard against false lemmas. We often work with boolean axiomatizations, that is conditional rules with boolean preconditions over free constructors: Every rule in R is of the form a1=b1 ∧ ... ∧ an=bn ⇒ s=r where for all i in [1..n], bi ∈ [true, false]. Conjectures can also be boolean clauses i.e., clauses whose negative literals are of the type ¬a=b where be [true, false]. If we assume that any defined function is completely defined, for any function f with boolean values, the following is inductively valid: f(t1,...,tn) = true. We can reformulate the complement rule as follows: COMPLEMENT: (E ∪ {¬(a=b) ∨ r }, H) |¬ (E ∪ {a=b ∨ r }, H)
if b ∈ [true, false].

The generate rule captures the essence of induction. It allows to obtain the induction schemes that entails the proof of the conjecture under consideration. An induction scheme must be formed in the way that has a good chance of success. In some sense, the success of a proof is strongly depends to what variable the generate rule applies. Then a question naturally arises: What is the best variable to replace in order to be able to apply a definition and eventually the induction hypothesis? For example consider the following set R of conditional axioms that defines the intersection of two lists,
Definition 1. (inductive position set). If $R$ is a set of axioms and $f$ a function symbol in $F$, then $\text{IP}(R,f) = \{p \mid p \not\in \varepsilon \text{ and } \exists (P \Rightarrow l \rightarrow r) \text{ in } R \text{ such that } l(e) = f \text{ and } l(p) \in F\}$ is the \textbf{inductive position set of function} $f$ \textbf{w.r.t.} $R$. $\text{IP}(R)$ denotes the set $\bigcup_{f\in F} \{\text{IP}(R,f)\}$ and is the \textbf{inductive position set of $R$}. 

Example. For the set $R$ of rules that define the intersection of two lists, we get $\text{IP}(R,\text{INT}) = \{1\}$, $\text{IP}(R,\varepsilon) = \{2\}$, $\text{IP}(R,\text{EQ}) = \{1, 2\}$. 

Definition 1 allows to select among the set of variables of an equation, a subset which is suitable for the application of generate rule. However, we may cut down this set by discarding variables judged useless since they lead to goals to which definitions fail to apply. Let $C = a_1 = b_1 \lor \cdots \lor a_n = b_n \lor x_{n+1} = b_{n+1} \lor \cdots \lor v_m = b_m$. We denote by $\text{comp}(C)$ the set of atoms of $C$: $\{t \mid t$ is either $a_i$ or $b_j$ for $i = 1, 2, \ldots, m\}$. 

Definition 2. Suppose $R$ is a left-linear set of conditional rules, and $C$ is a clause. $U_X$ is defined to be the set of positions $u$ in $t$ labelled by $x$ such that whenever there is a rule $P \Rightarrow l \rightarrow r$ in $R$ such that $l$ unifies with $t/v$ for some prefix $v$ of $u$ then, $[u/v] \in D(R)$ and there is a position $p$ in $\text{IP}(R,l)$ such that $p$ is a prefix of $u$. The multiset $U_X(C)$ of variable $x$ in $C$ is then defined as $\{\{u \mid u \in U_X(t) \text{ and } t \in \text{comp}(C)\}\}$. 

Example. Suppose $C_1 = (x, L) = true \land \varepsilon (x, S) = true \Rightarrow \varepsilon (x, \text{INT}(L, S)) = true$. Then, $U_X(\varepsilon (x, L)) = \emptyset$, $U_Y(\varepsilon (x, L)) = \{2\}$ (with $v = \varepsilon$) $U_X^1(\varepsilon (x, S)) = \emptyset$, $U_Y(\varepsilon (x, S)) = \{2\}$, $U_X(\varepsilon (x, \text{INT}(L, S))) = \emptyset$, $U_Y(\varepsilon (x, \text{INT}(L, S))) = \{2\}$ (here $v = 2$). Therefore $U_X(C_1) = \{\emptyset\}$, $U_Y(C_1) = \{\{2\}\}$ and $U_{L}(C_1) = \{\{2, 21\}\}$. Suppose now $C_2 = \text{INT}(x, \text{INT}(Y, Z)) = \text{INT}(\text{INT}(X, Y), Z)$. Then, $U_X(\text{INT}(X, \text{INT}(Y, Z))) = \{1\}$, $U_Y(\text{INT}(X, \text{INT}(Y, Z))) = \{1\}$, $U_X(\text{INT}(X, \text{INT}(Y, Z))) = \{2\}$, $U_Y(\text{INT}(X, \text{INT}(Y, Z))) = \{2\}$, and $U_{Z}(\text{INT}(X, \text{INT}(Y, Z))) = \{2\}$. Therefore $U_X(C_2) = \{\{1, 1\}\}$, $U_Y(C_2) = \{\{1\}\}$ and $U_Z(C_2) = \{\emptyset\}$. 

The main reason for considering the set $U_X(C)$ is just to select the best variables in a clause to induce upon. The motivation behind the requirements of definition 2 is the following. First, the condition of unifiability is well-understood since if a term $t/v$ does not unify with $R$, any instance of $t/v$ using the elements of a test set could not be an instance of $R$. Second, assume that $t/v$ unifies with $R$. If a variable $x$ is at position $u$ in $t/v$ and the length of $u$ is greater than or equal to the length of the variable positions in $R$, then further instantiation of $x$ cannot create instances of $R$ since $R$ is assumed to be left linear so eventual substitutions are too shallow to matter. Further, if $u$ is not a suffix of an induction position, then instantiation of this variable cannot give further information about $t/v$. The following definition defines formally the notion of induction variables in our setting:

Definition 3 (induction variables): Variable $x$ in $C$ is a \textbf{induction variable} if for all variables $y$ in $C$ either $x$ is a generalized variable and $y$ is not a generalized variable or $|U_Y(C)| < |U_X(C)|$. A variable $x$ is said to be generalized if $x$ is the variable introduced by the CGT-rule below, i.e. if $x$ is the variable obtained by replacing common non-trivial subterms in a clause by $x$. Notice that this requirement is consistent with definition 2 since the CGT-rule generalizes subterms that are at positions suffixes of positions in the induction position set of a function. Notice also that in the case when $|U_Y(C)| = |U_X(C)|$ either $x$ or $y$ may be used to induce upon. 

Example. Suppose that in $C_1$ no variable is generalized. Since $|U_X(C_1)| \leq |U_Y(C_1)| < |U_{L}(C_1)|$, $L$ is the induction variable in $C_1$. Assume that in $C_2$ no variable is generalized. Since $|U_Y(C_2)| < |U_Y(C_2)| < |U_X(C_2)|$, $X$ is the induction variable in $C_2$. Suppose now that in $C_3 = \text{INT}(W,X) = \text{INT}(W,X)$ $W$ is generalized and $X$ is not. Then $W$ is the induction variable in $C_3$. However, if we suppose that both $X$ and $W$ are not generalized variables, then either $W$ or $X$ may be considered as induction variable, since $|U_X(C_3)| = |U_X(C)|$. 

Finding Generalizations in Conditional Theories: the CGT-rule 

We now introduce an inference rule which turns out to be an indispensable part of practical inductive theorem prover. The essential idea behind the inference rule is to propose a generalized form of the conclusion just before another application of generate rule is attempted and failure begins. At the basis of our method for generalizing is the following
of B. Variable W is said to be the
generalized variable in B.

**CGT-rule** : \[(E \cup \{B\}, H) \vdash \neg (E \cup \{D\}, H)\] if D is a
generalized transform of C.

As we have pointed out above CGT-rules produces a
generalized clause in two steps: At the first step the condition
1 with \(\eta \theta\) create a clause D’ that contains at least an equation
(i.e. the equation \(d[s]=a[s]\)) in which both sides share the same
non-trivial subterm s. Condition 2 implies that a[s] cannot be
rewritten to d[s]. Condition 3 implies that both members of
equation \(d[s]=a[s]\) are distinct. Therefore, the generalized
clause cannot be reduced to a tautology. Condition 4 filter out
outcomes through a representative set of ground terms to
guard against over-generalization. Notice that our motivation
is to produce a generalized that is as general as possible. So if
D’ is a clause of the form \(P \lor \alpha[s] = \alpha[t]\), with \(\text{Var}(\alpha()) \cap
\text{Var}(t) = \emptyset\), \(\text{Var}(\alpha()) \cap \text{Var}(s) = \emptyset\), and \(\text{Var}(\alpha()) \cap \text{Var}(P) = \emptyset\),
P \lor \alpha[s] = \alpha[t]\) can be simplified to P \lor (s = t), since P \lor \alpha[s] = \alpha[t]\) is inductively valid if P \lor (s = t) is so.

**Example 1**

Let \(A = \text{perm}?(l, is(l))\) be true. So \(P[x] \lor \{lx\} = \text{perm}?(l, is(l)), \ r[x] = true, and P[x] is empty. Let B = \(\neg \eta[c, ins(c, is(l))] = true \lor \text{perm}?(l, del(c, ins(c, is(l))))\) be true. Then s = is(l), b = true and Q[s] = \(\neg \eta[c, ins(c, is(l))\) = true. Clearly
B is derived from A by using the substitution \{L/y, l\} and
rewriting. Then b = r = true, and substitution \eta is the
identity. Let \(\theta\) be the substitution \{L/y, l\}. Then \(\text{perm}?(\text{is}(l))\) = true \(\lessdot \text{perm}?(l, del(c, ins(c, is(l))))\) = true. Let s = is(l). Notice that is(l) is the maximal common subterm. Consider
now the clause D’ = \(\neg \eta[c, ins(c, is(l))\) = true \lor \text{perm}?(l, del(c, ins(c, W))) = \text{perm}?(l, is(l))\). Then the clause D’ = \(\neg \eta[c, ins(c, W)] = true \lor \text{perm}?(l, del(c, ins(c, W))) = \text{perm}?(l, is(l))\). Since \(\alpha[l] = \text{perm}?(l, l)\) we then have: \(\text{perm}?(l, l) \lor \text{Var}(\text{perm}?(l, del(c, ins(c, W))))\) = \(\emptyset\), \(\text{Var}(\text{perm}?(l, l) \lor \text{Var}(\neg \eta[c, ins(c, W)]\) = true\) = \(\emptyset\), \(\text{Var}(\text{perm}?(l, l) \lor \text{Var}(W))\). Therefore D’ = \(\neg \eta[c, ins(c, W)] = true \lor \text{del}(c, ins(c, W))) = W is a
generalized transform of B and W is a generalized variable.

We list here a representative sample of theorems that can all
be proved from the definitions alone, using the CGT-rule.
For comparison, the automated proof systems Nqthm, RRL,
INKA, LP, SPIKE, CLAM-Oyster, etc., failed to produce a
proof for all of them when given just the above definitions.
Of course, with the addition of some lemmas all provers are able
to prove this. All conjectures proposed by the CGT-rule are
sufficiently simple to be proved automatically without
introducing fresh failure. The definitions of the functions
involved into the theorems bellow are classical and therefore
omitted.

**Example 1**

Theorem to be proved: \(\text{EQ}(y, 0) = false \land
\text{div}(x, y) = true \Rightarrow \text{div}(x^2, y) = true\). The test instance
substitution \{k/s(z)\} reduces to \(\text{EQ}(y, 0) = false \land \text{div}(x, y) = true \Rightarrow \text{div}(x^2, y) = true\). The CGT-rule then speculates thelemma
\(\text{EQ}(y, 0) = false \land \text{div}(x, y) = true \Rightarrow \text{div}(x^2, y) = true\). This lemma allows the proof of the equation,
From: Proceedings of the Twelfth International FLAIRS Conference. Copyright © 1999, AAAI (www.aaai.org). All rights reserved.

EQ(y,0) = false ∧ div(x,y) = true ⇒ div(x,k*y) = true to go through without divergence.

Example 2: Theorem to be proved: ∈ (min(L),L) = true. The test instance substitution {L/y, x} reduces to min(y,l) < x = true ⇒ ∈ (min(y,l),x,y,l) = true. The CGT-rule then speculates the lemma W < x = true ⇒ ∈ (W,x,y,l) = true. This lemma allows the proof of the equation, ∈ (min(L),L) = true to go through without divergence.

Example 3: Theorem to be proved: sorted(is(L)) = true. The test instance substitution {L/y, x} reduces to sorted(is(y, is(l))) = true The CGT-rule then speculates the lemma sorted(is(y,W)) = sorted(W). This lemma allows the proof of the equation, sorted(is(L)) = true, to go through without divergence.

Example 4: Theorem to be proved: perm?(L,is(L)) = true. The test instance substitution {L/y, x} reduces to perm?(L,del(x,ins(x,is(l)))) = true The CGT-rule then speculates the lemma perm?(L,del(x,ins(x,is(l)))) = true ⇒ del(x,ins(x,W)) = W.

Example 5: Theorem to be proved: ∈ (x,ins(x,is(l))) = true. The test instance substitution {L/y, x} reduces to ∈ (x,ins(x,ins(x,is(l)))) = true The CGT-rule then speculates the lemma ∈ (x,ins(x, ins(x,W)) = ∈ (x, ins(x, W)). This lemma (with example 4) allows the proof of the equation, Perm?(L, is(L)) = true, to go through without divergence.

Examples 3 to 5 form the first completely automatic proof of the correctness of insertion sort.

In the majority of case, the conjectures proposed by CGT-rule are optimal in a sense that are the simplest possible equations to fix divergence and sufficiently simple to be proven automatically without introducing a fresh divergence. When multiple lemmas are proposed, then any one on its own is sufficient to fix divergence. In some cases a false lemma may be also conjectured. The totality of the conditional generalized transforms, during the proof of a conjecture, form a search space. One way of organizing it is by a proof tree: The root node is conjecture C to be proved; every internal node labeled by an equation, has as children all possible generalized transforms of the clause C which is the result of an application of the transition rules to C. This allows to reject quickly all false conjectures.

Conclusion

This paper has described a simple rules which attempts to propose generalizations which may be used to inductive proofs in conditional theories. This support has the advantage of overcoming the usual failures of inductive proofs. The method seems to be very simple and powerful. The applicability of the method is also illustrated with a large number of examples.

References


