TRIVIALIZING THE PROOF OF TRIVIAL THEOREMS

Y. Kodratoff and J. Castaing

LRI, Bat. 490 - Universite de Paris-Sud - F. 91405 ORSAY CEDEX

ABSTRACT

Besides a definition of "trivial" theorem, this paper presents a sketch of our methodology for the generalization of recurrence proofs on "trivial" theorems but that lead, in a context of automatic theorem proving, to very lengthy (or even impossible to achieve) proofs.
This paper reduces to a description of a detailed example, we hope to make clear that our methodology is of a much wider field of application.

I INTRODUCTION

In the field of theorem proving by induction, the need for an efficient generalization system has been expressed several times [2,3,5]. The methodology presented here differs from the one already used in existing systems [2,4]by two main features. First, our heuristics are driven by example proofs run on particular values of the variables (in this, we follow [6,7]. Second, instead of a progressive generalization expected to eventually reach a state where the theorem is provable, we go the other way round: we "savagely" generalize the theorem into an expression which is (in general) FALSE, and use a progressive particularization expected to eventually reach a state where the theorem is TRUE (and provable). This coincidence of truth and provability of a progressively particularized expression is implicitly part of our definition of "triviality".

II OUR DEFINITION OF A TRIVIAL THEOREM

II.1 Sketch of our methodology

We suppose that we are in an environment of recursive definitions as in [1] (Burstall 69) and that we have at our disposal a well-founded ordering. Let t(x) be a theorem to be proven by induction on x. Let s(x) be the successor of x in the well-founded ordering. An induction proof contains two steps. The first one is the basis case (which we consider here as already proven). The second tries to prove that t(x) implies t(s(x)). Let R be the set of rules we have at our disposal (section 3), and let $tc(x) =_R t(s(x))$ be the expression one obtains after having applied R to t(s(x)).

Definition: We say that t(x) r-matches tc(x) iff
1- there exists a substitution σ such that $\sigma_0 t(x) = tc(x)$
2- $\sigma_0 x$ is less than or equal to x in the well-founded ordering.

Remark: This definition implies that the substitution $x \leftarrow s(x)$ is a failure of the r-matching.

It is clear that the r-matching of t(x) towards tc(x) proves that the implication $t(x) \rightarrow t(s(x))$ is valid in the theory. If the r-matching of t towards tc fails, we analyse the conditions at which it could succeed. These conditions are considered as recursively generated new theorems to be proven by the same methodology.

II.2 Definition of "trivial"

The theorems we are able to prove are trivial in the following sense:
a- If the proposition to be proven is FALSE, it must "quickly" evaluate to FALSE for the first particular values of its variables. In the list domain these values are NIL, (CONS A NIL), (CONS B (CONS A NIL)),... where A, B are atoms.
b- When the matching of t towards tc fails then it often happens that the conditions which express how to avoid this failure are a system of equations (of the diophantine type) among a set of new variables. "Trivial" means also here that this system is "easy" to solve. The words quoted above, i.e, "quickly" and "easy" can have different definitions. In our system, we have chosen the following: "quickly" is "at once" (an untrue expression must evaluate to FALSE for the first element of the well-founded ordering) and "easy" means that only equality relationships linking the variables directly are allowed. Notice that even with that ground definition of trivial, we are able to prove (trivial) theorems the proof of which is very difficult.

III THE REWRITING SYSTEM

The functions we are to use are given below under a case representation (Burstall 69).

```
(EQL x y ) type : boolean
(EQL NIL   NIL  ) = TRUE
(EQL (CONS A x) NIL) = FALSE
(EQL NIL (CONS A y)) = FALSE
(EQL (CONS A x) (CONS B y)) =
               (AND (EQN A B ) (EQL x y))

(AND x y ) type : boolean
(AND TRUE y) = y
(AND FALSE y) = FALSE

(APP x y) type : list
(APP NIL y) =y
(APP (CONS A x) y) = (CONS A (APP x y))

(REV x) type : list
(REV NIL) = NIL
(REV (CONS A x)) = (APP (REV x) (CONS A NIL))
```

(F00 x y) type : list
(F00 NIL y) - y
(FOO (CONS A x)y) - (FOO x (CONS A y))

(EQN x y) type : boolean
(EQN ZERO ZERO) = TRUE
(EQN (SUCC x) ZERO) = FALSE
(EQN ZERO (SUCC y)) = FALSE
(EQN (SUCC x) (SUCC y)) = (EQN x y)

IV DESCRIPTION OF OUR METHODOLOGY

It consists of four steps, the first two of which will be exemplified by the proof of:
t= (EQL (APP (REV x) (FOO x IIIL))
(FOO (APP x x) NIL)).
Assuming that the basis case has been proven, a recurrence proof of t consists of an induction step done by the substitution x<-(CONS A x) and the rewritings of section III. This leads to:
tc= (EQL (APP (APP (REV x) (CONS A NIL))
(FOO x (CONS A NIL))) (FOO (APP x (CONS A x))
(CONS A NIL)))
and the proof of tc under the hypothesis that t is TRUE is not at all trivial.

IV.1 Step one

We "savagely generalize the variables of t, giving a different name to all of them. The obtained expression, named TI, is generally wrong. We particularize TI so that it will eventually take the form tl which evaluates to TRUE (for the first values of its variables). As in section II.1, we compute tcl and if there exists a substitution a: 0o tl = tcl, the problem is solved. Otherwise, we have to use step 2.

Example: The theorem t given above is savagely generalized to:
TI- (EQL (APP (REV xl) (FOO x2 NIL))
(FOO(APP x3 x4) NIL)).
Let xil, l<i<4, be an atom, we give to each variable xi the value (CONS xil NIL) and look for the conditions on the xil insuring that TI takes the value TRUE. The expression of T1((C0NS xil NIL)) is equal to:
(EQL (APP (REV (CONS xil NIL))
(FOO (CONS x21 NIL) NIL)) (FOO (APP (CONS x31 NIL)
(CONS x41 NIL)) NIL)).
A call - by - name evaluation of the definitions given in section 3 leads to evaluate first the underlined functional symbols, so that

TI ((CONS xil NIL)) becomes

\- (EQL (APP (APP (REV NIL) (CONS xil NIL))
(FOO (C0NS,x2I NIL~CTIL))
(FOO (CONS x31 (APP NIL (CONS x41 NIL))) NIL)).

\- (EQL (APP (APP NIL (CONS xil NIL))
(FOO (CONS x2 NIL) NIL))
(FOO (APP NIL (CONS x4 1 NIL)) (CONS x31 NIL))).

\- (EQL (APP (CONS xil NIL) (FOO(CONS x21 NIL) NIL))
(FOO (CONS x41 NIL) (CONS x31 NIL))).

= (EQL (CONS xll (APP NIL (FOO (CONS x21 NIL) NIL)))
(FOO NIL (CONS x41 (CONS x31 NIL)))).

\- (EQL (CONS xll (APP NIL (FOO (CONS x21 NIL) NIL)))
(CONS x41 (CONS x31 NIL))).

\- (AND (EQN xll x41)
(EQL (APP NIL (FOO (CONS x21 NIL) NIL))
(CONS x31 NIL))).
The evaluation stops at this point since xll and x41 are variables and we cannot evaluate (EQN xll x41). We therefore state that (EQN xl 1 x41) is a condition for TI ((CONS xil NIL)) evaluating to TRUE. We replace (EQN xil x41) by TRUE and the evaluation proceeds on (and is left to the reader) up to the result:
\- (AND (EQN x21 x31 (EQL NIL NIL)).
In the same way as above, we must have (EQN x21 x31).From these conditions, we deduce that xl = x4 x2 = x3 which are put in TI in order to obtain tl:
tl- (EQL (APP (REV y) (FOO z NIL))
(FOO (APP z y) NIL)).
We leave to the reader to verify that tl cannot be trivially proven by induction on y (or z). We therefore proceed on to step 2.

IV.2 Step two:Obtaining new theorems.

Broadly speaking, we apply again the same strategy which is too much generalizing and then finding for particular values of the variables the conditions which bring the generalization to TRUE. In this step the generalization is made according to the following heuristics: the matching of tl towards tcl fails, and we mark the terms in tl that do not contain the recurrence variable and fail to match with tel. Each marked term is generalized to a different variable vl,...vi and tl takes the form T2. In T2, we give particular values to the variables different from the vi's, and find conditions on the vi's so that these particular expressions evaluate to TRUE (as in step 1). Our triviality condition "insures" that the system of equations linking the vi's is easy to solve. The solution is put into T2 which becomes t2. If t2 does not match tc2, we recursively apply step 2 to t2 (obtaining t3...). This step stops in two cases: either the theorem is proven or the failure of the matching is the same in ti and ti+1. In the last case we proceed on to step 3.

Example: Choosing z as the inductive variable, so that z is replaced by (CONS A z) into tl
tl= (EQL (APP (REV y) (FOO z NIL))
(FOO (APP z y) NIL)), we obtain
tcl* (EQL (APP (REV y) (FOO z (CONS A NIL)))
(FOO (APP z y) (CONS A NIL))).
The r-matching of tl and tcl fails because it would lead to the substitution NIL <-(CONS A NIL), which is forbidden since the left part of a substitution must be a variable. As above explained, we replace the two occurences of NIL by two different variables vl and v2. We have
T2- (EQL (APP (REV y) (FOO z vl)) (FOO (APPzy)v2))
Giving to y and z the value NIL we obtain (as in step 1) the condition (EQL vl v2) which is put into T2 in order to give t2:
t2 = (EQL (APP (REV y) (FOO z u)) (FOO (APPzy)u)).
The reader will find that inducting on z (which is replaced by (CONS A z) in t2), one obtains tc2 such that there exists a substitution a such that
Oot2 = tc2, 0-=-(CONS A u). This proves the induction step for t2, because u is not the induction

variable. We assume here that the basis case has already been proven, so this ends the proof of t.

IV.3 Step three: Inducting lemmata from a partial matching.

We suppose that step 1 and step 2 failed and we must now prove the theorem t' obtained after applying step 1 and step 2. We restrict ourselves to the cases where the property to be proven about t' is not a unary operator. An instance of this operator is EQL of arity 2, when t' is concerned with list equality.
Let t' = (P S1(x) ... Sn(x)) where x is the induction variable, and t'c = (P Sc1(x) ... Scn(x)) such that t' does not match t'c.
Let us suppose that one subterm Si(x) matches Sci(x) : there exists a substitution σ such that σ_0 Si (x) = Sci(x) and $\sigma_0 x \neq x$. It is evident that t'= (P S1(x) ... Sn(x)) matches t" = (Pσ_0S1(x)...σ_0Sn(x)). It follows that if we are able to prove the n-1 lemmata:
σ_0S1(x) = Sc1(x),...σ_0Sn(x) = Scn(x), we will have proven t'. This is really a heuristic since nothing proves that the n lemmata are easy ones. Our simple remark is that this is the last hint one can find and is therefore worth trying.

IV.4 Step four: Inducting lemmata from a particular values of the induction variable.

We suppose that step 3 fails because no Si(x) matches an Sci(x).
Let s(0)<s(s(0))<... be the successive canonical forms of the well founded ordering. We write:
(P S1(0),... Sn(0)), (P S1(s(0)) ... Sn(s(0))),... and evaluate all the Si's with our rewriting rules. We attempt to match the evaluated Si(0) and Si(s(0)) (as we have already seen 0←s(0) is excluded as a matching failure). If this succeeds for one i, a substitution σ is found and used as in step 3.

V. CONCLUSION

Our main goal has not been to give a syntactical characterization of the class of theorems we are able to prove in that way. Notice nevertheless, that improving our possibility to solve systems of equations and to detect our constant substitutions will extend the class of trivial theorems as defined here.

REFERENCES

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

[2] Aubin R., "Mechanizing structural induction". Ph. D. Thesis. Univ. Edinburgh (1976).

[3] Boyer R.S. and J S. Moore, "A computational logic". Academic Press (1980).

[4] Castaing J., Y. Kodratoff and P. Degano, "Theorem proving by the study of example proof traces". Proc. Int. Workshop in Program Construction Bonas, (1980).

[5] Castaing J. and Y. Kodratoff, "Generalisation de theories". Actes reunion GROSSEM, Poitiers (1980), 141-166.

[6] Degano P., J. Castaing and Y. Kodratoff, "Inductive hypothesis extracted from proof traces", Atti AICA 80, 114-116.

[7] Degano P. and F. Sirovich, "Inducting function properties from computation traces". IJCAI-79, 208-216 and "Inductive generalization and proofs of function properties", Comp. Ling. (1979), 13, 101-130.