

CM-STRATEGY: A METHODOLOGY FOR INDUCTIVE THEOREM PROVING OR CONSTRUCTIVE WELL-GENERALIZED PROOFS.

Marta FRAŇOVÁ

L.R.I., Bât. 490, 91405 Orsay Cedex, France

ABSTRACT

The main problem, when automatically proving theorems by induction is the problem of strategy, or, how to automatically direct deductions. This is not trivial, and, at present, only a mixture of complicated strategies have been investigated. The essential contribution of this paper is therefore the proposing of a new strategy for inductive theorem proving, inspired by a new mechanism called Constructive Matching (CM), and used for automatic programming [f04].

We also propose a new method for the recognition of predicates and functions, necessary to prove a theorem by our approach, that are not defined in the knowledge-base ("invention" of new operators). Finally, we illustrate the obtainment of a suitable generalized lemma necessary for the proof.

INTRODUCTION

One of the earliest techniques for program synthesis, the automated construction of computer programs, has been the *deductive approach* [m05], in which the program is developed by proving a theorem corresponding to the given specification.

The special techniques needed for the fulfilment of this deductive approach have inspired us to develop a constructive methodology for inductive proofs. For this, we determined, step by step, all the tools we needed for inductive *automatic* theorem proving, i.e.,

(i) we determined in which "data-types", proofs by induction can be performed automatically (requirements on axioms, definitions of functions and predicates, ...), (but we do not treat the problem of how to transform "bad" data into "good" data);

(ii) we determined by what the choice of (a scheme of) the induction principle is influenced, and (because we find it possible), we formulated an induction principle which helps us to *automatically* "generate" induction hypotheses (in the form) that are *necessary* for the proof of a given theorem;

(iii) we determined *how* to proceed from given data (axioms + induction hypotheses) to the given theorem (i.e. the *strategy*).

In the present paper only (ii) and (iii) are treated.

The novelty of our approach, and a comparison with already existing inductive theorem proving systems is exemplified in [f08], and therefore not explained here. But, let us point out an essential difference: We *construct directly* (without transformations) the desired formula. This difference appears to be very important, as soon as one realizes, that in our approach

- the application of induction hypotheses is not one of "most difficult points", as it is, for instance in [b12], pg. 90;
- special heuristics for transforming a given formula into an other, to which an induction hypothesis can be applied, are not needed as in [m05].

Due to a lack of space we are forced to present our system from a methodological (nevertheless correct) point of view rather than to give its complete algorithmic description. Such a description requires the introduction of notions that are not published elsewhere (but with which our system works) and therefore an algorithmic description without specifying these notions would be confusing.

The organization of the paper is as follows. We start with a motivation, i.e. a presentation of our methodology in an intuitive way. We also answer the question: Why can proofs be directed by CM-strategy? Section 2 presents our formulation of the Induction Principle, and indicates the links between information included explicitly, or implicitly, in given (to be proved) theorems and information one can express explicitly when one has a "good" formulation of the structural induction principle. Section 3 (the most important from the methodological point of view) gives the formal definition of Constructive Matching. In this section, we also describe how one can recognize subproblems with regard to a given theorem and our knowledge base. In the conclusion we explain why we were motivated by Beth's method of semantic tableaux and why we did not use them in their original form. APPENDIX I contains a list of axioms used in the paper. APPENDIX II shows our methodology working in automatic programming. When we refer to appendices, we write $\Rightarrow$ I, and $\Rightarrow$ II respectively.

1. MOTIVATION

Let us start with an example which is nothing more than a presentation of our methodology when proving theorems by induction.

The reader is asked to work through this example with us carefully, because it will then allow a better understanding of the theoretical explanation which follows in next sections.

Let us suppose that we want to prove

$$\forall x (EQL (REV x) (FOO x \text{ null})),$$

noted $\forall x Q(x)$, where x is of the type LIST-of-NAT given by the family of constructors {null, unit, append}. NAT is the type of natural numbers. EQL is the equality on LIST-of-NAT. FOO and REV are functions

FOO: LIST-of-NAT $\times$ LIST-of-NAT $\rightarrow$ LIST-of-NAT,

REV: LIST-of-NAT $\rightarrow$ LIST-of-NAT, defined by following axioms:

$$A_1: (FOO \text{ null } l) = l$$

$$A_2: (FOO (\text{unit } a) l) = (\text{append } (\text{unit } a) l)$$

$$A_3: (FOO (\text{append } (\text{unit } a) l) L) = (FOO l (\text{append } (\text{unit } a) L))$$

$$A_4: (REV \text{ null}) = \text{null}$$

$$A_5: (REV (\text{unit } a)) = (\text{unit } a)$$

$$A_6: (REV (\text{append } (\text{unit } a) l)) = (\text{append } (REV l) (\text{unit } a)).$$

Consider the family of constructors (here {null, unit, append}). It decomposes the induction proof into cases, each possible valuation of the induction variable x producing a case. Since this valuation is a kind of equality, we shall

denote it by EQT.

Our example has three cases:

case(i) (EQT x null), prove Q(x);

case(ii) (EQT x (unit a)), prove Q(x);

case(iii) (EQT x (append (unit a) l)), suppose Q(l) and prove Q(x).

Let us note (Valuat x) a chosen valuation of x. Let σ_x be {x $\leftarrow$ (Valuat x)}.

Intuitive description of CM-strategy:

We describe Q(x) by its "pattern" PTQ. As an illustration, let us suppose that it is PTQ = (H EXPR1 EXPR2). In the above example, H = EQL, EXPR1 = (REV x), EXPR2 = (FOO x null).

Using the valuation of x in each case, (by the evaluation of H with regard to EXPR1 and with regard to the given valuation, and by an application of induction hypotheses, if it is the case) one deduces a valid formula which has the pattern PTQ' = (H EXPR1 EXPR1'). In this step, the value of EXPR2 is not taken into account.

The next step is the constructive one: We try to find valid transformations of EXPR1' into EXPR2, i.e. we transform EXPR1' into EXPR1'', so that EXPR1'' = σ_x (EXPR2).

It may happen that a direct transformation of EXPR1' into EXPR2 is not possible. We then evaluate EXPR2 for the current valuation. This leads to EXPR2'. Then, either EXPR1' can be transformed into EXPR2', or the equality EXPR1' = EXPR2' becomes a new theorem, to be treated as before. The last case may well generate an infinite sequence of theorems to be proven.

Now the example will show how we construct this equality, and how some infinite sequences are reduced to a generalized theorem.

case(i):

(EQT x null), σ_x is {x $\leftarrow$ null}. Here,

PTQ' = (EQL (REV x) null) because of A_4 . We now try to find the axioms that can transform EXPR1' (= null) into EXPR2 (= (FOO ...)). The axiom A_1 is the only one which can be used, because it may have instances of the form null = (FOO ...). We obtain the equality (which is an instance of A_1): null = (FOO null null), i.e. we have succeeded in obtaining a transformation of EXPR1' (= null) into EXPR1'' (= (FOO null null)). Moreover, EXPR1'' = σ_x (EXPR2). This completes the proof of case (i).

case(ii):

(EQT x (unit a)), σ_x is {x $\leftarrow$ (unit a)}. Here,

PTQ' = (EQL (REV x) (unit a)). We therefore try to transform EXPR1' = (unit a) into an expression of the form (FOO ...). We have no space here to explain why, but only two axioms, which may have instances of the form (unit a) = (FOO ...), can be used: A_1 and A_2 . Using A_1 , one finds that EXPR1' (= (unit a)) can be transformed into EXPR1'' (= (FOO null (unit a))). But

EXPR1'' $\neq$ σ_x (EXPR2), therefore this possibility is rejected.

Because (unit a) is (append (unit a) null), using the instance of A_2 (i.e. (unit a) = (FOO (unit a) null)), one finds that EXPR1'' is (FOO (unit a) null).

EXPR1'' = σ_x (EXPR2) is satisfied, and this completes the proof of case(ii).

case(iii):

(EQT x (append (unit a) l)), σ_x is {x $\leftarrow$ (append (unit a) l)}. In this case we have at our disposal the induction hypothesis:

(EQL (REV l) (FOO l null)).

By the evaluation of (REV x) with regard to the given valuation of x, and an immediate application of the induction hypothesis, we obtain

PTQ' = (EQL (REV x) (append (FOO l null) (unit a))).

As there are no axioms that have an instance matching (append (FOO l null) (unit a)) = (FOO ...), we must evaluate EXPR2. We obtain EXPR2' = (FOO l (unit a)). We want to prove EXPR1' = EXPR2', i.e.

$Q_1(l, a)$: (append (FOO l null) (unit a)) = (FOO l (unit a)).

This equality becomes a new theorem to be proven:

$\forall l \forall a Q_1(l, a)$.

We skip the application of our methodology to this new theorem. It "fails" again by leading to a new EXPR1' and EXPR2' that are different, as before. The new theorem reads:

$Q_2(l', a, b)$:

(append (append (FOO l' null) (unit b)) (unit a)) = (FOO l' (append (unit b) (unit a))).

Continuing this way we obtain, that for proving Q_2 we need to prove Q_3 , for proving Q_3 we need to prove Q_4 , ... After several steps we try to see, whether it is possible to find a generalized theorem determined by $Q_1, Q_2, Q_3, Q_4, \dots$

The reader can see that the structures of Q_1 and Q_2 are very similar, being of the form

$Q^*(l, L)$: $\forall l \forall L$ (EQL (append (FOO l null) L) (FOO l L)).

Therefore Q^* will be this generalized theorem.

In this case, the automation of the common structure of Q_1 and Q_2 is trivial since Q^* is simply the least generalization of Q_1 and Q_2 (more details can be found in [k31]).

We have undertaken the proof (still at a conjectural state) that provable theorems can be expressed as a sequence of theorems which can be generalized.

The proof of $Q^*(l, L)$ is not difficult, and left to the reader.

Why could we conjecture such a "simple" form of the sequence of constructed theorems? Why can proofs always follow the same scheme (evaluation of an expression, application of the induction hypothesis, directed transformation of an expression into another one, ...)? The reasons are simple:

1) The form of a given theorem indicates which is the formula we have to obtain for a given valuation of the induction variable. This directs the construction of expressions necessary for a construction of this formula.

2) Functions and predicates are defined *recursively*, and *with regard to a given family of constructors*. This facilitates a search for links among expressions, and together with the form of the formula we want to obtain, it indicates missing lemmas.

3) Moreover, a good formulation of the structural induction principle points at the elements for which we should explicitly express the induction hypothesis, and therefore an application of induction hypotheses no longer is (contrary to [b12]) one of the most difficult problems when proving theorems by induction.

These arguments are only consequences of next sections.

2.1. PROOFS BY INDUCTION

In practice, when we try to prove by induction a theorem $\forall x A(x)$ with x of the type T we know (or should know) some *primitive functions* (or constructors) by which we can obtain elements of the type T by a simple combination of these primitive functions. Functions with a codomain T will be called *generators* of T.

Moreover, we suppose that we know *selectors*, i.e. functions which for an element x of the type expressed by some primitive function f as (f y) gives the result y.

Finally, we suppose that we have at our disposal *predicates* which take the value TRUE only if an element of the type has been constructed using the given primitive function.

By these predicates $p_1, \dots, p_k$ we can define the equivalence relation $\sim$ on the type T: for x, y of the type T x $\sim$ y iff there is p_i such that ($p_i x$) = ($p_i y$) = TRUE.

We can then choose the representative rep_i of the class

$P_i = \{ x; (p_i x) = \text{TRUE} \}, i=1, \dots, k.$

For instance, for $T = \text{LIST-of-NAT}$ we have the predicates null?, unit?, append? and the representatives of the appropriate classes are null, (unit a), (append (unit a) l).

So, when we want to prove $\forall x A(x)$ of the type T it is enough to take all the representatives $rep_1, \dots, rep_k$, and to prove $A(rep_i)$ for each $i = 1, \dots, k$.

Now we introduce the notion of the valuation of one element x as the choice of one rep_i which will be noted $(EQT x rep_i)$, and very often we say that x is represented by rep_i . So EQT is in a sense an equality on T . It is not an equality throughout the proof but only in the considered case (for a given representative). When we speak about one possible not-explicitly determined valuation of x , we will write $(\text{Valuat } x)$, i.e. $(\text{Valuat } x) \in \{rep_1, \dots, rep_k\}$.

2.2. THE STRUCTURAL INDUCTION PRINCIPLE - FORMULATION

Let us first introduce some notions and notations. Let us denote by FGT the family of generators of some well-founded type, and by FCT the family of constructors of the type T , i.e. $FCT \subset FGT$. We classify elements of FCT in the following way:

- constants of the type T (i.e. $f_j: \rightarrow T$)
- basic- T -operators (i.e. $f_j: D_1^1 \times \dots \times D_{n_j}^1 \rightarrow T$, such that there is no $m \in \{1, \dots, n_j\}$ such that D_m^1 is T);
- general- T -functions (i.e. $f_j: D_1^1 \times \dots \times D_{n_j}^1 \rightarrow T$, such that there is at least one $m \in \{1, \dots, n_j\}$ such that D_m^1 is T).

PRINCIPLE OF STRUCTURAL INDUCTION

Let us suppose that we want to prove that $A(a)$ holds for an arbitrary element a of the type T . $A(a)$ holds for all a of the type T provided that the following hold:

- $A(f_c)$ holds for every constant f_c of FCT ,
- $A(f_j(x_1^j, \dots, x_{n_j}^j))$ can be proved for every basic- T -constructor f_j of FCT with the arbitrary arguments $x_1^j, \dots, x_{n_j}^j$ (x_m^j is of the type D_m^1 for $m \in \{1, \dots, n_j\}$) considered, during the proof of $A(f_j(x_1^j, \dots, x_{n_j}^j))$, as local constants (i.e. they do not change their value during the proof-procedure),
- $A(f_j(x_1^j, \dots, x_{n_j}^j))$ can be proved for every general- T -constructor f_j , with $x_1^j, \dots, x_{n_j}^j$ considered during the proof as local constants, and starting the proof procedure by supposing

- (i) explicit validity of $A(x_{p_q}^j)$ for each local constant $x_{p_q}^j$ of the type T ($x_{p_q}^j \in \{x_1^j, \dots, x_{n_j}^j\}$)
- (ii) implicit validity of $A(p_{p_q})$ for any $p_{p_q} < x_{p_q}^j$, where $x_{p_q}^j$ is a local constant and p_{p_q} belongs to the well-founded ordering of T , called $D_{p_q}^1$.

Here, implicit means that during the proof procedure a new parameter p_{p_q} of T can be constructed, and p_{p_q} is smaller than $x_{p_q}^j$. We can then suppose $A(p_{p_q})$ to be valid. Naturally, p_{p_q} will not occur explicitly in $A(f_j(x_1^j, \dots, x_{n_j}^j))$, which is the last formula of the deduction.

- (iii) implicit validity of $A(q)$ for any $q < f_j(x_1^j, \dots, x_{n_j}^j)$.

As one can easily notice, this principle of structural induction cuts a proof of $\forall x A(x)$ into two cases (distinguished by the existence of induction hypotheses).

2.3. STRUCTURAL INDUCTION AND STRUCTURE OF FORMULAE

In this section we explain that the structure of a formula A (when proving $\forall x A(x)$) can lead to some heuristics when using the structural induction principle during the proof of the desired theorem.

Remark 2.3.1.:

It may happen that we do not want to prove $\forall x A(x)$ for all elements of the type T , but only for elements which satisfy some condition P , i.e. we want to prove $\forall x (P(x) \Rightarrow A(x))$.

The predicate P in our (inductive) approach cannot be arbitrary, because we must be able to determine constructors of the type $T_P = \{x | x \in T \wedge P(x) \text{ holds}\}$. Then instead of proving $\forall x (P(x) \Rightarrow A(x))$, we prove the theorem $\forall x A(x)$ in T_P .

Example 2.3.1.: Let us suppose that in NAT we want to prove the theorem:

$\forall y (\text{odd } y) \Rightarrow [(\text{2div } y) = (\text{Suc } (\text{2div } (\text{substr1 } y)))]$, where 2div is the integer-division-by-two function, the meaning of odd , Suc and substr1 is clear.

The predicate odd determines the type NAT_{odd} in which each element is either $(\text{Suc } 0)$ or can be written as $(\text{Suc } (\text{Suc } a))$ for a $a \in \text{NAT}_{\text{odd}}$. It means that we will try to prove $\forall y [(\text{2div } y) = (\text{Suc } (\text{2div } (\text{substr1 } y)))]$ in the domain NAT_{odd} .

Remark 2.3.2.:

In $A(x)$ may also occur some functions or predicates which are defined with the help of a "selective" function df of the type T (i.e. $df: T_1 \times T \rightarrow T$, where T_1 can be T and for $hp \in T_1$ ($df \ hp \ q$) is an element of T smaller than q).

In a such case, when we want to prove $A(x)$ for x given by some representative of T expressed by a general- T -constructor, we include in our induction hypotheses also a hypothesis $\forall hp A((df \ hp \ x))$. One can see that hp here represents all elements p of T_1 for which $A((df \ p \ x))$ is satisfied. We call such a hp the help-parameter. A predicate A defined with the help of a selective function will be called the predicate depending on a help-parameter, which will be written: A depends on hp .

Example 2.3.2.: Let us suppose that we want to prove $\forall x \exists z Q(x, z)$, where $Q(x, z)$ is $(\text{PERMUT } x \ z) \wedge (\text{ORDERED } z)$.

The definition of PERMUT , given by (AL23)-(AL27) in [1], contains a "selective" function DELETE . When we have to prove $\exists z Q((\text{append } (\text{unit } a) \ l), z)$, we use the induction hypothesis $\forall p \exists z' Q((\text{append } (\text{unit } a) (\text{DELETE } p \ l)), z') \wedge (\text{MEMBER } p \ l)$ as well as the more classical one $\exists z' Q(l, z')$. One cannot know in advance which one is to be used (may be both). In general one has to use all the possible induction hypotheses induced by the "selective" function.

What is gained by using of our structural induction principle as a method of proving theorems? Nothing more than

- the knowledge of the form (or pattern) of the formula which we want to obtain, i.e., $A(f_j(x_1^j, \dots, x_{n_j}^j))$ from given axioms and formulae $A(p_{p_q}) A(x_{p_q}^j)$ for elements p_{p_q} and $x_{p_q}^j$ described in the formulation of the structural induction principle, and
- the generation of induction hypotheses in the required form.

3. CONSTRUCTIVE MATCHING

The idea of constructive matching comes from realizing that a given theorem $\forall x A(x)$, to be proved in a theory F , expresses a form $A(x)$ of a formula B which should be proved from axioms and hypotheses, i.e. we would like to construct a B , valid in F , such that there is a substitution σ such that $\sigma A(x) = B$.

3.1. DEFINITION AND EXAMPLES

Definition 3.1.1.: Formula B matches formula A iff there is a substitution σ such that σA is identical to B (i.e. $\sigma A = B$).

Example 3.1.1.:

Let B be $((\text{Suc } a) = x_2 * u_1 + (\text{Suc } u_2)) \wedge (\text{Suc } u_2) < x_2$, let A be $(x_1 = x_2 * z_1 + z_2) \wedge (z_2 < x_2)$. Then, B matches A with the substitution $\{x_1 \leftarrow (\text{Suc } a), z_1 \leftarrow u_1, z_2 \leftarrow (\text{Suc } u_2)\}$.

Definition 3.1.2.: If formulae A and B do not match, but from the theory F and B we can prove B' which matches A, then the process of finding B' is called *Constructive Matching* (CM) of A and B.

When proving the theorem $\forall x A(x)$, using the structural induction principle, we have to prove the validity $A(\text{rep}_i)$ for $i \in \{1, \dots, k\}$, for all the representatives $\text{rep}_1, \dots, \text{rep}_k$. Let us note σ_i the substitution $\{x \leftarrow \text{rep}_i\}$. Then, in the case (EQT $x \text{ rep}_i$) we have to obtain the formula $\sigma_i(A(x))$. One can see that (EQT $x \text{ rep}_i$) and $A(x)$ do not match, but, if $\forall x A(x)$ is provable in F, we can prove $A(\text{rep}_i)$ from theory F (extended by possible induction hypotheses), and (EQT $x \text{ rep}_i$).

HOW TO PERFORM THE CONSTRUCTIVE MATCHING

Let $Q: T_1 \times T_2 \rightarrow \text{BOOL}$ be a recursive predicate, defined with respect to representatives of T_1 . Let rep be a representative given by a general- T_1 -constructor. Then the part of the definition Q with respect to rep can be expressed symbolically as follows:

$Q(x_1, x_2)$ holds
if $\begin{cases} x_1 = \text{rep} \wedge Q(\text{srep}_1, f_1^1(x_2)) \wedge P_1(f_1^2(\text{rep}), f_1^3(x_2)) \\ x_1 = \text{rep} \wedge Q(\text{srep}_2, f_2^1(x_2)) \wedge P_2(f_2^2(\text{rep}), f_2^3(x_2)) \\ \dots \\ x_1 = \text{rep} \wedge Q(\text{srep}_m, f_m^1(x_2)) \wedge P_m(f_m^2(\text{rep}), f_m^3(x_2)), \end{cases}$

where, for $j \in \{1, \dots, m\}$, the following are satisfied: srep_j is obtained from rep by some selector, for any y $f_j^2(y) < y$; P_j are already defined predicates, f_j^i are already defined functions, moreover, for any rep and any x_2 , the following condition, called CONDP,

$$P_1(f_1^2(\text{rep}), f_1^3(x_2)) \vee P_2(f_2^2(\text{rep}), f_2^3(x_2)) \vee \dots \vee P_m(f_m^2(\text{rep}), f_m^3(x_2))$$

is TRUE.

Let us call the j -th line of the definition Q for x represented by rep , the formula

$$Q(\text{srep}_j, f_j^1(x_2)) \wedge P_j(f_j^2(\text{rep}), f_j^3(x_2)).$$

On each j -th line x_2 is considered as completely (symbolically) determined by $f_j^1(x_2)$ and $f_j^3(x_2)$, i.e. there is a "function" $G_j(f_j^1, f_j^3)$ such that $G_j(f_j^1(m), f_j^3(m)) = m$ for any m .

Let EXPR be an unquantified expression depending on one variable only, say x , and let $\forall x Q(x, \text{EXPR})$ be the theorem to be proved using the structural induction principle.

$Q(x_1, x_2)$ is defined with regard to x_1 . Therefore to prove $\forall x Q(x, \text{EXPR})$ means, for an arbitrary x , to prove that EXPR belongs to the class, say C_x , of all x_2 for which $Q(x, x_2)$ holds.

Let us consider the case, where x is represented by a general- T_1 -constructor rep . Then we have to consider the above mentioned part of the definition of Q. Let us note τ_j the substitution $\{x \leftarrow \text{srep}_j\}$.

If x is represented by rep , the class C_{rep} is implicitly determined by the definition of Q: If $Q(\text{srep}_j, q_1) \wedge P_j(f_j^2(\text{rep}), q_2)$ holds for some q_1, q_2 , then we know that $G_j(q_1, q_2) \in C_{\text{rep}}$, i.e. there exists an x_3 from C_{rep} such that $f_j^1(x_3) = q_1$ and $f_j^3(x_3) = q_2$.

Moreover, $x_2 \in C_{\text{rep}}$ only if there is a j -th line such that

$$(**) Q(\text{srep}_j, f_j^1(x_2)) \wedge P_j(f_j^2(\text{rep}), f_j^3(x_2))$$

is satisfied. This is to say that if $Q(\text{rep}, x_2)$ is valid, then there is a j -th line such that (**) is satisfied. Let us call (**) the valid part of $Q(\text{rep}, x_2)$ (corresponding to j -th line). One can see that as soon as the valid part (**) of the formula

$Q(\text{rep}, x_2)$ was obtained, in our consideration we can replace (**) by $Q(\text{rep}, x_2)$.

In our approach, finding out whether or not $\text{EXPR} \in C_{\text{rep}}$, for the given representation of x by rep , is performed by taking an $x_2 \in C_{\text{rep}}$, and verifying whether or not x_2 is equal to (or can be transformed into) EXPR, i.e. we "construct" the formula $Q(\text{rep}, \text{EXPR})$ in the following way:

We take an x_2 for which $Q(\text{rep}, x_2)$ is known. Then we verify the possibility of a transformation of x_2 into EXPR (for the given representation of x by rep).

The choice of x_2 is not arbitrary. We only choose an x_2 which has some links with EXPR. The links between elements of C_{rep} and EXPR are expressed in induction hypotheses.

We will show, how these links may be explicated:

Let us suppose, that Q does not depend on hp (see Remark 2.3.2.), i.e. that srep_j does not depend on hp for $j = 1, 2, \dots, m$. Then, we have at our disposal the induction hypothesis $Q(\text{srep}_j, \tau_j(\text{EXPR}))$, where τ_j is $\{x \leftarrow \text{srep}_j\}$. Because we want EXPR to be from C_{rep} , for some j must $\tau_j(\text{EXPR})$ be $f_j^1(\text{EXPR})$.

Moreover, for the same j , $P_j(f_j^2(\text{rep}), f_j^3(\text{EXPR}))$ must be satisfied.

Now, if srep_j depends on hp for some j , i.e. $\text{srep}_j = (df \ t_1 \ \text{rep})$ for some t_1 , let us note ρ_j the replacing of t_1 by hp. Then, we have at our disposal the induction hypothesis

$$(H_{hp}) \quad \forall hp \ \rho_j(Q(\text{srep}_j, \tau_j(\text{EXPR}))).$$

τ_j is $\{x \leftarrow \text{srep}_j\}$. Therefore $\tau_j(\text{EXPR})$ depends on hp, as well. But, if srep_j depends on hp, it means that in the definition of Q the expression t_1 is determined as $g(f_j^2(x_2))$ for some function g . Therefore instead of taking H_{hp} we use $\rho_j(\tau_j(Q(x, \text{EXPR})))$ with $\rho_j = \{t_1 \leftarrow g(f_j^2(\text{EXPR}))\}$.

Because we want EXPR to be from C_{rep} , for some j , $\rho_j(\tau_j(\text{EXPR}))$ must be $f_j^1(\text{EXPR})$ (or, if they are not same, $\tau_j(\text{EXPR})$ must be transformable into $f_j^1(\text{EXPR})$).

Moreover, $P_j(f_j^2(\text{rep}), f_j^3(\text{EXPR}))$ must be satisfied.

Notice, that it may happen that while $\tau_j(\text{EXPR})$ and $f_j^1(\text{EXPR})$ may not be the "same" expressions (for instance $(\text{Suc } (+ a \ b))$ and $(+ a \ (\text{Suc } b))$ are not the same expressions), $\tau_j(\text{EXPR})$ can be transformed into $f_j^1(\text{EXPR})$.

With regard to preceeding remarks, the construction of $Q(x, \text{EXPR})$, for the given representation of x by rep , is as follows:

Let ξ be a symbol representing elements $x_2 \in C_{\text{rep}}$. Let us note (*) the formula $Q(x, \xi)$. As mentioned above, (*) for x represented by rep holds, only if there is a j -th line such that (**) is satisfied.

We therefore take the definition of Q on j -th line ($j = 1, \dots, m$). We write

$$(1j) \quad Q(\text{srep}_j, f_j^1(\xi)) \wedge P_j(f_j^2(\text{rep}), f_j^3(\xi)).$$

$\text{srep}_j < \text{rep}$, therefore we have at our disposal the induction hypothesis $Q(\text{srep}_j, \tau_j(\text{EXPR}))$, resp. $\rho_j(Q(\text{srep}_j, \tau_j(\text{EXPR})))$ if srep_j depends on hp. The induction hypothesis allows us to replace $f_j^1(\xi)$ in (1j) by $\tau_j(\text{EXPR})$, resp. by $\rho_j(\tau_j(\text{EXPR}))$.

So we have constructed

$$(2j) \quad Q(\text{srep}_j, \tau_j(\text{EXPR})) \wedge P_j(f_j^2(\text{rep}), f_j^3(\xi)).$$

resp. (2j') $\rho_j(Q(\text{srep}_j, \tau_j(\text{EXPR}))) \wedge P_j(f_j^2(\text{rep}), f_j^3(\xi)).$

Now, we verify whether or not

$$P_j(f_j^2(\text{rep}), f_j^3(\text{EXPR}))$$

is satisfied. If it is, we verify whether or not

$$G_j(\tau_j(\text{EXPR}), f_j^3(\text{EXPR})),$$

resp. $G_j(\rho_j(\tau_j(\text{EXPR})), f_j^3(\text{EXPR}))$ is equal to (or can be transformed into) EXPR.

If $P_j(f_j^2(\text{rep}), f_j^3(\text{EXPR}))$ is not satisfied for the j -th line, we call this line a total-failure-line, because there is no sense in looking for some strategy by which we could succeed in constructing $Q(x, \text{EXPR})$ on this line. But if the theorem is provable, then, with respect to the condition CONDP, there must

be at least one line which is not a total-failure-line, and for which $G_j(\tau_j(\text{EXPR}), f_j^i(\text{EXPR}))$ can also be transformed into EXPR.

It may also happen that $G_j(\tau_j(\text{EXPR}), f_j^i(\text{EXPR}))$ cannot be transformed directly into EXPR. Then, we look for a rule (a lemma) of the form:

$$Q(\text{rep}, G_j(\tau_j(\text{EXPR}), f_j^i(\text{EXPR}))) \wedge \dots \Rightarrow Q(\text{rep}, \text{EXPR}).$$

In such a way, if we succeed in constructing a valid part of $Q(x, \text{EXPR})$, for the given representation of x by rep , we can conclude that we have constructed the formula $Q(x, \text{EXPR})$, valid for the given representation of x by rep .

Remark: One can argue that we should first verify the condition P_j , rather than immediately apply induction hypotheses. This objection is not valid if theorems $\forall x \exists z Q(x, z)$ are also treated (see II).

In order to link this discussion with our intuitive description of the CM-strategy (section 1), one can notice that H is Q here, EXPR1 is x , EXPR2 is EXPR . $\text{EXPR1}'$ is $G_j(\tau_j(\text{EXPR}), f_j^i(\text{EXPR}))$, resp. $G_j(\rho_j(\tau_j(\text{EXPR})), f_j^i(\text{EXPR}))$.

Let us suppose that $P_j(f_j^i(\text{rep}), f_j^i(\text{EXPR}))$ is satisfied. The English commentary produced during the proof is then: By the evaluation of Q with regard to x represented by rep , and with regard to ξ representing the class C_{rep} we obtain $Q(x, G_j(f_j^i(\xi), f_j^i(\xi)))$. Then by the application of the corresponding induction hypothesis, we obtain the formula

$Q(x, G_j(\rho_j(\tau_j(\text{EXPR})), f_j^i(\text{EXPR})))$, resp. $Q(x, G_j(\rho_j(\tau_j(\text{EXPR})), f_j^i(\text{EXPR})))$, valid for the representation of x by rep .

Finding the valid transformation of $G_j(\tau_j(\text{EXPR}), f_j^i(\text{EXPR}))$ into EXPR by ... (axioms and rules giving this transformation are mentioned), the formula $Q(x, \text{EXPR})$, valid for the representation of x by rep , is considered to be constructed.

Example: Let us suppose that we want to prove

$\forall x (\text{PERMUT } x (\text{append} (\text{unit} (\text{last } x)) (\text{DELETE} (\text{last } x) x)))$, the definition of PERMUT, in I, leads us, for x represented by $(\text{append} (\text{unit } a) l)$, to consider the following two lines of the definition of PERMUT:
(PERMUT $x_1 \ x_2$) holds

$$\text{if } \begin{cases} x_1 = (\text{append} (\text{unit } a) l) \wedge (\text{PERMUT } l (\text{CDR } x_2)) \\ \quad \wedge (a = (\text{CAR } x_2)) \\ x_1 = (\text{append} (\text{unit } a) l) \wedge \\ \quad (\text{PERMUT } (\text{DELETE} (\text{CAR } x_2) (\text{append} (\text{unit } a) l)) \\ \quad (\text{CDR } x_2)) \wedge (a \neq (\text{CAR } x_2)) \end{cases}$$

We have: f_1^i is CDR, f_2^i is CAR, f_3^i is CAR, rep_1 is l , rep_2 is $(\text{DELETE} (\text{CAR } x_2) (\text{append} (\text{unit } a) l))$. PERMUT is defined with regard to the selective function DELETE. For the given EXPR, (CAR EXPR) is (last x).

The corresponding induction hypotheses are therefore

(H_1) (PERMUT $l (\text{append} (\text{unit} (\text{last } l)) (\text{DELETE} (\text{last } l) l)))$, τ_1 is $\{x \leftarrow l\}$.

(H_2)

(PERMUT
(DELETE (last x) (append (unit a) l))
(append
(unit (last (DELETE (last x) (append (unit a) l)))
(DELETE (last (DELETE (last x) (append (unit a) l)))
(DELETE (last x) (append (unit a) l))))))
 $\wedge$ (MEMBER (last x) (append (unit a) l)),
 $\tau_2 = \{x \leftarrow (\text{DELETE} (\text{last } x) (\text{append} (\text{unit } a) l))\}$.

We want to construct $Q(x, \text{EXPR})$ for x represented by $(\text{append} (\text{unit } a) l)$.

Let us take the 1-st line of the definition of PERMUT for x represented by $(\text{append} (\text{unit } a) l)$. We write
(PERMUT $x \ \xi$) holds only if

$$(1_1) (\text{PERMUT } l (\text{CDR } \xi)) \wedge a = (\text{CAR } \xi)$$

We verify whether or not $a = (\text{CAR } \text{EXPR})$, (i.e. $a = (\text{last } x)$) is satisfied for the given representation of x .

In our approach, this is performed by checking whether or not a can be transformed into the form (last x) for x represented by $(\text{append} (\text{unit } a) l)$. Therefore we look at the definition of the function last, and we see that a can be transformed into (last L) for $L = (\text{unit } a)$

or $L = (\text{append} (\text{unit } b) (\text{unit } a))$ for some b . Both possibilities are rejected, and this failure is registered as a total failure TF_1 .

Therefore, we take the second line of the definition of PERMUT for x represented by $(\text{append} (\text{unit } a) l)$. We write

$$(1_2) (\text{PERMUT } (\text{DELETE} (\text{CAR } \xi) (\text{append} (\text{unit } a) l)) (\text{CDR } \xi)) \wedge a \neq (\text{CAR } \xi).$$

By the application of H_2 we obtain

$$(2_2) (\text{PERMUT } (\text{DELETE} (\text{last } x) (\text{append} (\text{unit } a) l)) (\text{append} (\text{unit} (\text{last} (\text{DELETE} (\text{last } x) (\text{append} (\text{unit } a) l))) (\text{DELETE} (\text{last} (\text{DELETE} (\text{last } x) (\text{append} (\text{unit } a) l))) (\text{DELETE} (\text{last } x) (\text{append} (\text{unit } a) l)))))) \wedge a \neq (\text{CAR } \xi).$$

We now verify whether or not $a \neq (\text{CAR } \xi)$, i.e.

$a \neq (\text{CAR} (\text{append} (\text{unit} (\text{last } x)) (\text{DELETE} (\text{last } x) x)))$ for the given representation of x by $(\text{append} (\text{unit } a) l)$. (CAR EXPR) is (last x), therefore $a \neq ((\text{CAR } \text{EXPR})$ is verified as (NOT ($a = (\text{last } x)$)). We verify, therefore, whether or not a can be transformed into (last x) for the given representation of x . This is not possible, and we conclude that $a \neq (\text{CAR } \text{EXPR})$ is satisfied.

Because $a \neq (\text{CAR } \xi)$ is in (2_2) satisfied, (2_2) is nothing but the valid part of (PERMUT $x \ G_2(\tau_2(\text{EXPR}), f_2^i(\text{EXPR}))$).

The last step therefore is to verify whether or not $G_2(\tau_2(\text{EXPR}), f_2^i(\text{EXPR}))$ can be transformed into EXPR, i.e.

whether or not
(append (unit (last x))
(append (unit (last (DELETE (last x) (append (unit a) l)))
(DELETE (last (DELETE (last x) (append (unit a) l)))
(DELETE (last x) (append (unit a) l))))))

can be transformed into
(append (unit (last x)) (append (DELETE (last x) x))).

The equality of these two expressions is not possible, therefore we are looking for a rule of the form

$$(\text{PERMUT } A \ B) \wedge \dots \Rightarrow (\text{PERMUT } A \ C).$$

We obtain that the rule we look for is the transitivity of PERMUT. Then, using the rule

$$((\text{CAR } A) = (\text{CAR } B)) \wedge (\text{PERMUT } (\text{CDR } A) (\text{CDR } B)) \Rightarrow (\text{PERMUT } A \ B)$$

together with H_2 , completes the proof. For lack of space, the detailed description of this last part is not given here.

It is very difficult to explain such constructions without being too formal. We hope that the examples given in this section, and section 1. as well, help us to be as illustrative as possible.

3.2. SUBPROBLEMS WITH REGARD TO CM-PROCEDURE

During the process of CM we can use only functions and predicates defined by axioms. But it may happen that the proof $\forall x A(x)$ needs more functions and predicates than those currently available.

This situation is similar to the decomposition of the problem into subproblems (see [s04]), but with a small difference: we do not ask "how to decompose a problem into subproblems", but rather: How to determine that, for a proof of our theorem $\forall x A(x)$, we will need some new function or predicate which is not explicitly defined in our system?

The aim of this becomes clear, when one realizes that these new functions will help us to explicit some parts of the representatives ξ (see preceding section and $\equiv$ II).

Definition 3.2.1.:

Let M be a predicate $(:T_1 \times T_2 \rightarrow \text{BOOL})$ such that there exists a function $\varphi_M: T_1 \rightarrow T_2$ and three predicates $Q_1: T_1 \times T_2 \rightarrow \text{BOOL}$, $Q_2: T_1 \times T_3 \rightarrow \text{BOOL}$, $Q_3: T_3 \times T_2 \rightarrow \text{BOOL}$, such that

1. φ_M is the function represented by the Specification Theorem
 $\forall x (P_1(x) \Rightarrow \exists z Q_1(x, z) \wedge (\forall y Q_2(x, y) \Rightarrow Q_3(y, \varphi_M x)))$,
 where P_1 characterizes the input domain;
2. $M(p, (\varphi_M p))$ is true iff $\forall y Q_2(p, y) \Rightarrow Q_3(y, (\varphi_M p))$;
3. Q_2 is not a kind of equality;
4. $Q_3 \neq M$;

Then, we call M -problem the synthesis of φ_M from the Specification Theorem. Any M for which we can define an M -problem will be said to have the ∇ -property).

Example 3.2.1.: To the predicate LTL defined in $\equiv$ I, we can associate the theorem (LTL-problem):

$\forall l ((\text{NOT}(\text{EQL } l \text{ null})) \Rightarrow (\exists z (\text{MEMBER } z \text{ } l) \wedge (\forall y (\text{MEMBER } y \text{ } l) \Rightarrow (< z y))))$
 which defines MIN (minimum) of elements of l . For all u of the type NAT, (LTL u l) holds only if $(\leq u \text{ } (\text{MIN } l))$.

Definition 3.2.2.:

Let C be a unary predicate. We call a formula SD_C with the free-variable x , such that $(C(x) \Leftrightarrow SD_C(x))$ holds for any x , a semantic definition of a condition $C(x)$.

Example 3.2.2.: $\forall y (C_1(y) \Rightarrow C_2(x, y))$, where C_1, C_2 are two known predicates, is a semantic definition of C such that $C(x)$ iff $\forall y C_1(y) \Rightarrow C_2(x, y)$.

Definition 3.2.3.:

C is a Trivial Condition if its semantic negation does not contain existential quantifiers. If this is not true, C is called a Non-trivial Condition.

Example 3.2.3.: $(\text{MEMBER } a \text{ } l)$ is a Trivial Condition because its semantic negation is $\forall x (\text{MEMBER } x \text{ } l) \Rightarrow (\neq x a)$.

The predicate LTL: $\text{NAT} \times \text{LIST-of-NAT} \rightarrow \text{BOOL}$ the semantic definition of which is (LTL $x \text{ } l$) iff

$[\forall y ((\text{MEMBER } y \text{ } l) \Rightarrow (< x y))]$ is Non-trivial Condition. By analogy, (PRIME x) iff

$\forall y ((y < x) \wedge (y \text{ divides } x) \Rightarrow (y = 1))$ is a Non-trivial Condition.

When C is trivial, it can either be replaced by a simpler condition, or its evaluation is trivial. It can therefore be used as a predicate in the conditional part of a recursive definition. When C is not trivial, since it has the ∇ -property, we shall first try to synthesize a corresponding φ_C . Then, φ_C will help us to simplify the condition C and explicit, if necessary, some parts of ξ (see $\equiv$ II).

Example 3.2.4.: Let us suppose that we want to prove $\forall l \exists l' (\text{PERMUT } l' \text{ } l) \wedge (\text{ORDERED } l')$. We can see that ORDERED in $\equiv$ II is defined with the help of LTL, which has the ∇ -property. This is why, before starting the proof, we need to synthesize φ correspondig to the theorem
 $\forall l ((\text{NOT}(\text{EQL } l \text{ null})) \Rightarrow (\exists z (\text{MEMBER } z \text{ } l) \wedge (\forall y (\text{MEMBER } y \text{ } l) \Rightarrow (< z y))))$. (The application of this can be found in $\equiv$ II.)

One should realize that there is no general strategy for proving theorems of the type $\forall x \dots \exists y \dots \forall z \dots$. To see this try proving the well-known relative-prime-number problem (see in [g02]):

$$\forall n \exists N \forall p \forall q [\text{RP}(p, q) \wedge p, q > N \Rightarrow |\sqrt{2} - \frac{p}{q}| > \frac{1}{2 + \frac{1}{n}}],$$

where $\text{RP}(p, q)$ means that p and q are relative prime.

This is why in our system we have some heuristics in order to solve some simple such problems.

CONCLUSION

We have shown that the CM-procedure is used when we want "constructively" to prove a given theorem. It means that CM is the strategy used to orient our deduction when proving a given theorem from given axioms $A_1, \dots, A_k$.

Beth proposed a solution to the problem of finding whether or not some formula V is a logical consequence of formulae $A_1, \dots, A_k$. His solution is the method of semantic tableaux [b03], formalized by his Completeness Theorem for a system of Natural Deduction F. As Beth himself pointed out, the practical interest of his method is seriously impaired by complicated splittings of a tableau into subtableaux. By a modification of Beth's method of semantic tableaux inspired by the CM-strategy, we have obtained a method for inductive theorem proving.

Our modification consists in

- including the structural induction principle in the set of rules for the construction of tableaux relative to ST and
- orienting a development of tableaux (by CM-strategy) towards the desired goal.

We do not give here our modification of Beth's method.

Our approach is currently under implementation, but has not been yet completed. Its main difficulty is due to the generalizations which will be left to the user in this first version.

The efficiency of our methodology depends on the truth of our conjecture relative to sequence of theorems generated by the recursively generated $\text{EXPR1}'$ and $\text{EXPR2}'$ (see section 1). It may be that an elaborate strategy is needed in order to put this sequence in such a form that its generalization appears at once.

ACKNOWLEDGEMENTS

I would like to express my warmest thanks to Yves Kodratoff. I also thank Professor Georg Kreisel for his encouragement. Professor Jean-Luc Remy provided many helpful remarks.

APPENDIX I

We give here only the list of axioms explicitly used in the paper.

TYPE NAT - given by

Constructors: 0: $\rightarrow \text{NAT}$ and Suc: $\text{NAT} \rightarrow \text{NAT}$

Selectors: Pred: $\text{NAT} \rightarrow \text{NAT}$

Predicates: zero?: $\text{NAT} \rightarrow \text{BOOL}$ and Suc?: $\text{NAT} \rightarrow \text{BOOL}$.

TYPE LIST-of-NAT - given by

Constructors: null: $\rightarrow \text{LIST}$, unit: $\text{NAT} \rightarrow \text{LIST}$ and append: $\text{LIST} \times \text{LIST} \rightarrow \text{LIST}$

Selectors: CAR: $\text{LIST} \rightarrow \text{NAT}$ and CDR: $\text{LIST} \rightarrow \text{LIST}$

Predicates: null?: $\text{LIST} \rightarrow \text{BOOL}$, unit?: $\text{LIST} \rightarrow \text{BOOL}$, append?: $\text{LIST} \rightarrow \text{BOOL}$

Relations and Functions:

EQL: $\text{LIST} \times \text{LIST} \rightarrow \text{BOOL}$

MEMBER: $\text{NAT} \times \text{LIST} \rightarrow \text{BOOL}$

LTL : $\text{NAT} \times \text{LIST} \rightarrow \text{BOOL}$ defined by

- (AL8) (LTL x null)
 (AL9) (LTL x (unit y)) $\wedge (= x y)$
 (AL10) (LTL x (unit y)) $\wedge (< x y)$
 (AL11) (LTL x (append (unit y) l)) $\wedge (= x y) \wedge (\text{LTL } x \text{ l})$
 (AL12) (LTL x (append (unit y) l)) $\wedge (< x y) \wedge (\text{LTL } x \text{ l})$
 ORDERED : $\text{LIST} \rightarrow \text{BOOL}$
 (AL13) (ORDERED null)
 (AL14) (ORDERED (UNIT x))
 (AL15) (ORDERED (append (UNIT x) l))
 $\wedge (\text{LTL } x \text{ l}) \wedge (\text{ORDERED } l)$
 (AL16) (ORDERED (append (UNIT x) (append (UNIT y) l)))
 $\wedge (= x y) \wedge (\text{ORDERED } (\text{append } (\text{UNIT } y) \text{ l}))$
 (AL17) (ORDERED (append (UNIT x) (append (UNIT y) l)))
 $\wedge (< x y) \wedge (\text{ORDERED } (\text{append } (\text{UNIT } y) \text{ l}))$
 DELETE : $\text{NAT} \times \text{LIST} \rightarrow \text{LIST}$
 PERMUT : $\text{LIST} \times \text{LIST} \rightarrow \text{BOOL}$ defined by
 (AL23) (PERMUT null null)
 (AL24) (PERMUT (unit x) (unit y)) $\wedge (= x y)$
 (AL25) (PERMUT (append (unit x) (unit y))
 (append (unit y) (unit x)))
 (AL26) (PERMUT (append (unit x) l) (append (unit y) m))
 $\wedge (= x y) \wedge (\text{PERMUT } l \text{ m})$
 (AL27) (PERMUT (append (unit x) l) (append (unit y) m))
 $\wedge (\neq x y) \wedge (\text{MEMBER } y \text{ l})$
 $\wedge (\text{PERMUT } (\text{DELETE } y \text{ (append (unit x) l)) } m)$

APPENDIX II

This appendix is for the automatic programming oriented reader, and is readable only if the section 3. has already been understood.

We try to present, here, the CM-procedure as it appears in our approach to automatic programming.

Let Q, rep be as described in section 3. Let the part of the definition of a predicate R with regard to x represented by rep be as follows:

$R(x)$ holds if

$$\begin{cases} x = rep \wedge R(srep_1') \wedge P_1'(f_1'(rep)) \\ x = rep \wedge R(srep_2') \wedge P_2'(f_2'(rep)) \\ \dots \\ x = rep \wedge R(srep_k') \wedge P_k'(f_k'(rep)). \end{cases}$$

Let us suppose, that we want to prove a special theorem of the form $\forall x A(x)$ which is $\forall x \exists z Q(x, z) \wedge R(z)$.

If x is represented by rep , then to find z such that $Q(x, z) \wedge R(z)$ holds means:

Take an abstract representative ξ of all x_2 for which $Q(rep, x_2) \wedge R(x_2)$ holds. Then with the information included in axioms, induction hypotheses, rules, ..., try to explicitate this element. How can it be done?

The definition of Q leads us to consider m cases for the $Q(rep, \xi)$ part of the formula $Q(rep, \xi) \wedge R(\xi)$:

$$Q(srep_j, f_j^i(\xi)) \wedge P_j(f_j^i(rep), f_j^i(\xi)) \quad (j=1, \dots, m).$$

Naturally, we have at our disposal the induction hypothesis $\exists z_j Q(srep_j, z_j) \wedge R(z_j)$, resp. $\forall hp \exists z_j Q(srep_j, z_j) \wedge R(z_j)$, therefore $f_j^i(\xi)$ can be replaced by z_j , and so we have:

$$Q(rep, G_j(z_j, f_j^i(\xi))) \text{ holds} \\ \text{only if } P_j(f_j^i(rep), f_j^i(\xi)) \text{ is satisfied.}$$

Then we look at the definition of R and we look for a line, say the i -th, where $srep_i'$ and $f_i^i(\xi)$ are the "same". Because of the induction hypothesis $R(srep_i')$ can be replaced by $R(z_i)$, and therefore the condition $P_i'(f_i^i(\xi))$ together with $P_j(f_j^i(rep), f_j^i(\xi))$ will help us explicitate the $f_j^i(\xi)$ part as a function expression depending only on x , or z_j . Let $z_j'(x)$ be this explicitly expressed $f_j^i(\xi)$. Let us note that $P_j(f_j^i(rep), f_j^i(\xi) \wedge P_i'(f_i^i(\xi)))$ cannot be verified as was the case for the preceding type of theorems, but it may be reduced to the condition $C_j(f_j^i(rep), z_j'(x))$. Then, naturally, finding z , for x represented by rep , will follow the scheme:

$$\text{if } C_j(f_j^i(x), z_j'(x)) \text{ holds then } z \text{ is } G_j(z_j, z_j'(x)).$$

Example :

Let us try to prove $\forall x \exists z (\text{PERMUT } x \text{ z}) \wedge (\text{ORDERED } z)$ for x represented by (append (unit a) l) (i.e. to find a part of a program SORT when (append? x) is satisfied).

Then the structural induction principle gives the following induction hypotheses:

- (H₁) (PERMUT l v₁) $\wedge$ (ORDERED l v₁)
 (H₂) $\forall hp \exists v_2^p (\text{PERMUT } (\text{delete } hp \text{ (append (unit a) l)) } v_2^p) \wedge (\text{ORDERED } v_2^p) \wedge (\text{MEMBER } hp \text{ (append (unit a) l)}).$

Due to a lack of space, we will only consider the 1-st line of the definition of PERMUT (see section 3.). Let ξ be a symbol denoting an element for which (PERMUT x ξ) $\wedge$ (ORDERED ξ) is satisfied.

The first line of PERMUT indicates, that for the given representation of x , (PERMUT x ξ) holds only if

(PERMUT l (CDR ξ)) $\wedge$ (a = (CAR ξ)) is satisfied. By (H₁), (CDR ξ) can be replaced by v_1 . We obtain:

(PERMUT x (append (CAR ξ) v₁)) holds only if
 (PERMUT l v₁) $\wedge$ (a = (CAR ξ)) is satisfied,

i.e.

(PERMUT x (append (CAR ξ) v₁)) holds only if
 (a = (CAR ξ)) is satisfied.

Now we look at the definition of ORDERED, and, with regard to (H₁), we find that (ORDERED (append (CAR ξ) v₁)) holds only if (LTL (CAR ξ) v₁) is satisfied. We obtain, therefore:

(PERMUT x (append (CAR ξ) v₁))
 $\wedge$ (ORDERED (append (CAR ξ) v₁)) holds, only if
 (a = (CAR ξ)) $\wedge$ (LTL (CAR ξ) v₁) is satisfied. The condition (a = (CAR ξ)) gives

(PERMUT x (append (CAR ξ) v₁))
 $\wedge$ (ORDERED (append (CAR ξ) v₁)) holds, only if
 (LTL a v₁). Now, we can change this condition (see section 3.2.) to ($\leq a$ (MIN v₁)).

Because v₁ is (SORT (CDR x)) we obtain the following part of the program SORT:

if (append? x) then
 if (CAR x) < (MIN (SORT (CDR x)))
 then (append (CAR x) (SORT (CDR x))).

REFERENCES

- [b03] Beth E.W.: *The Foundations of Mathematics*; Amsterdam 1959.
 [b12] R.S.Boyer, J.S.Moore: *A Computational Logic*; Academic Press, 1979.
 [f03] M.Fraňová: Program Synthesis and Constructive proofs Obtained by Beth's tableaux, in Cybernetics and System Research 2, R. Trappl eds., North-Holland, Amsterdam 1984, pp. 715-720.
 [f04] M.Fraňová: CM - Strategy - Driven Deductions for Automatic Programming; in T.O'Shea (ed.): ECAI 84: Advances in Artificial Intelligence, Elsevier Science Publishers B.V. (North-Holland), 1984, pp. 573-576.
 [f08] M.Fraňová: A Methodology for Automatic Programming based on the Constructive Matching Strategy; to appear in: Proceedings of EUROCALL'85, Linz, April 1985.
 [f09] M.Fraňová: Inventing is Moderate Cheating or a Theory of Humble Invention; to appear: in Proceedings of Cognitiva '85, Paris, June 1985.
 [g02] Girard J.-Y.: Proof theory; to appear.
 [k31] Y.Kodratoff: Generalizing and Particularizing as the Techniques of Learning, Computers and Artificial Intelligence Vol.2, pp.417-442, 1983.
 [m05] Z.Manna and R.Waldinger: A Deductive Approach to Program Synthesis; ACM Transactions on Programming Languages and Systems, Vol. 2., No.1, January 1980, pp. 90-121.
 [s04] R.D.Smith: Top-Down Synthesis of Simple Divide and Conquer Algorithm; Technical Report NPS52-82-011, Naval Postgraduate School, Monterey, CA 93940, November 1982.