THEORY OF PAIRS,
PART I: PROVABLY RECURSIVE FUNCTIONS

by

Paul J. Voda

Technical Report 84-25

December, 1984

Theory of Pairs, Part I: Provably Recursive Functions.

Paul J. Voda

Department of Computer Science, The University of British Columbia,
6356 Agricultural Road, Vancouver, B.C. Canada V6T 1W5.

1. Introduction.

Although this paper is technically a purely logical paper, its intent is to explicate notions of (partial recursive) functions, functionals, data and function types, as well as of modules as used in modern (i.e. declarative) programming languages. The close connection between logic and semantics of certain programming languages has been recognized for some time now (see for instance Kowalski [6]). In a previous paper [17] we have explained declarative programming languages as terms and formulas of a first order Theory of Pairs (TP) which is developed in detail in this paper. Computation was understood as a proof of the program in a deductively restricted subtheory of TP. The restriction of the deductive power is necessary so the proof can be efficiently carried out by a computing machine. Proofs of correctness of programs amount to the, possibly computer-aided, proofs of theorems in TP. The problem of termination of programs amounts to the proof-theoretical question of existence of proofs in subtheories of TP. These can be tackled within a computer-assisted system provided one arithmetizes the proof theory in a suitable formal theory, say TP itself.

It would seem that we need either a high order theory to accomodate the hierarchies of functions and types (such as Martin-Lof's theory of constructive types [8]) or a powerful first order theory (some extension of lambda calculus, set theory). But such theories are not completely satisfactory because it is absolutely essential, in the parlance of programming languages, to be able to view functions and types as data and vice-versa. This amounts to the necessity of Gödel numbering of functions, types, and proofs. Any successful formal theory for programming languages will have to use Gödel numbers to explain compilers, interpreters, and operating systems. All of these work on representations (Gödel numbers) of functions and types. If we realize this, we suddenly see that we do not need a strong formal theory. Actually the weakest of the interesting theories - Peano Arithmetic (or some of its equivalents) - would do. Functions, functionals, and types can be explained by Gödel numbers. This is essentially what we attempt to do in this paper

Logicians are not particulary interested in the concrete formal proofs in a concrete formal theory. They are interested in the questions of existence of proofs. Neither are they interested in the details of particular encoding into Gödel numbers. They are satisfied with the knowledge that it is possible. Computer scientists care about the details of coding, they are writing programs which must be executed. They also should care about formal proofs of programs which can be computer generated or at least computer checked. On the other hand, in order to master the complexity of the task, computer scientists were forced to design readable notations. This is where logic can possibly benefit from computer science.

Theory of pairs, as presented and developed in this paper, concerns objects freely generated from 0 by the operation of pairing. A slightly more complicated domain called *symbolic expressions* is used as the universe of the programming language Lisp [10]. We prefer to call these objects pairs because this is what they are mathematically. Pairs are superior to natural numbers in encoding lists, trees, functions, formulas, proofs, etc. Although Lisp has beeen around for a quarter of a century and McCarthy and Peter [9,12] developed the theory of recursive functions over pairs informally, only in recent years has there been a serious interest in the formalization of pairs. Boyer and Moore [1] were amongst the first to formalize domains similar to pairs. The present

author developed his formalization at the beginning of 1984 without knowing about the work done by Sato [13] and Feferman [2] in this area. The domain of Sato is slightly more complicated than ours. Feferman works with second order theories.

To axiomatize pairs is only the starting point. What matters is how one proceeds in introducing partial recursive functions, functionals and types into the theory. This is, in our opinion, the main contribution of the present paper. We start with a notation for partial recursive functionals over pairs. We define the notion of computation of functionals (section 2). In section (3) we present the Theory of Pairs together with some initial development. In section (4) we formalize the notion of computation by introducing a predicate of computability (reducibility) *Comp* similar to $\exists p T(i, a, p)$ of Kleene [5]. With the help of *Comp* we shall interpret the functionals into TP. Section (5) develops the properties of *Comp* and culminates in the proof of a very general theorem which allows to prove the recursive equations for the recursive functions over pairs. The theorem is similar to the set-theoretical theorems of Von Neumann and Tarski (see for instance [7]) about functions defined by transfinite resp. well-founded inductions. In section (6) we develop a notation for ordinals less than ϵ_0. Our ordinals include natural numbers instead of their being encoded into natural numbers. Section (7) introduces the four basic arithmetic operations over ordinals. Since the natural numbers are ordinals the arithmetic operations over ordinals retain the expected properties for numbers. Finally, in section (8) we prove the transfinite induction for all ordinals less than ϵ_0 and syntactically introduce the recursive functions defined by transfinite induction as the special case of the main theorem.

We plan in the second part of the development of TP to generalize the finite types of Gödel [4]. We intend to develop a system of types finer than the finite types. We shall include cartesian products, unions, and generalize the so called polymorphic (parameterized) types of Milner [11]. Our approach to functionals over finite types differs from the standard one in that we *attribute* different types to the same functionals rather than to have a different functional for each type. We then prove the recursive equations for the class of well-typed functionals. The theory of types will be developed in TP in the detail sufficient for the semantics of typed programming languages.

In the third part we plan to utilize the results of the first two parts to give a simple proof of the constructive consistency of TP using our functionals. We shall interpret TP into a quantifier free subtheory of TP and prove the consistency of the subtheory. This is essentially the Gödel's Dialectica proof [4], but our notation permits the proof to be carried out in a simpler way without the detours through auxiliary theories. As another byproduct, we prove the incompleteness of TP. Finally we intend to show how to replace functionals by partial recursive functions and prove that all recursive functions which can be proven total in TP (*provably* recursive functions) can be introduced by the transfinite induction up to ϵ_0. This is a known result of Gentzen [3], but again we want to present a simpler proof. Thus the third part will be devoted to the proof-theoretical questions of TP and perhaps the theory of proofs [see for instance Schuette, Takeuti] can benefit from our notations.

2. Recursive Functionals over Pairs.

We are interested in computable functions and functionals over the domain of pairs. Pairs are freely generated from 0 by the operation of pairing $[\_, \_]$. Thus 0 is differrent from any pair of the form $[a, b]$. Two pairs $[a, b]$ and $[a', b']$ are equal iff a is equal to a' and b is equal to b'. We shall write $[a, b, c]$ as the abbreviation for $[a, [b, c]]$. Similarly for more elements. From the definition of pairs it should be obvious that every pair is either 0 or is uniquely of the form

$$[a_1, a_2, \ldots, a_n, 0] \tag{1}$$

for $n \geq 1$. Thus every pair is either the empty list (0) or is a non-empty list of the form (1). the pairs a_i for $1 \leq i \leq n$ are said to be *elements* (members) of the list (1). The empty list 0 has no elements. Let us abbreviate the pair $[0, 0]$ by 1, the pair $[0, 1]$ by 2, the pair $[0, 2]$ by 3, etc.

There is a well known one-to-one correspondence between pairs and natural numbers given by the computable bijection m.

$$m(0) = 0$$

$$m([x, y]) = \frac{(m(x)+m(y)+1)(m(x)+m(y))}{2}+m(x)+1$$

and thus we can expect computable functions over pairs to be the same as the recursive functions (over natural numbers). Since we are interested only in pairs we shall henceforth drop the qualification "over pairs" and speak simply of recursive functionals and functions.

Recursive functionals will be defined with the help of certain terms denoting pairs. We shall call such terms *recursive terms*, or shortly R-terms. R-terms are composed from the constant symbol 0, the (only) variable *n*, binary operation of pairing [_ , _], ternary operation **if** _ **then** _ **else** _, unary operations _.h and _.t of projection, as well as of the binary operation _ • _ of function application. Each R-term is composed by a finite number of applications of the formation rules i through vi.

i The constant 0 is an R-term.

ii The variable *n* is an R-term.

iii If **a** and **b** are R-terms, so is [**a**, **b**].

iv If **a** is an R-term, so are (**a**.h) and (**a**.t).

v If **a** and **b** are R-terms, so is (**a** • **b**).

vi If **a**, **b**, and **c** are R-terms, so is (**if a then b else c**).

The superfluous parentheses will be dropped whenever possible. Somewhat against the standard practice we write **a** • **b** • **c** as a shorthand for **a** • (**b** • **c**).

R-terms composed only by the formation rules i and iii are called *literals*. Literals denote the correspondingly constructed pairs from the domain of pairs. We shall write $\overline{p}$ for the literal denoting the pair *p*. An R-term is a *selection* from the argument iff it is obtained from the R-term *n*.t at most by the application of the formation rules iv. R-terms composed without the rule ii are called *closed* R-terms. *Open* R-terms contain at least one occurrence of the variable *n*. For a possibly open R-term **a** and a closed R-term **b** we shall denote by **a**{*n*:=**b**} the closed R-term obtained from **a** by replacing all occurrences of the variable *n* by the term **b**. An R-term **a** will be called a *function* R-term, or shortly RF-term, iff for all subterms of **a** of the form **b** • **c** the term **b** is either *n*.h or a literal.

In order to define computations of R-terms we assign names (Gödel numbers) to R-terms. This is done by a mapping from R-terms to literals. The name of an R-term **a** is written as ⌜**a**⌝. The following symbols will be used as abbreviations for literals 1 through 7 in that order.

zero, var, head, tail, pair, if, app (2)

Naming is defined by the recursion on the construction of R-terms as follows (≡ is read as " is the same term").

⌜0⌝ ≡ [*zero*, 0]
⌜*n*⌝ ≡ [*var*, 0]
⌜[**a**, **b**]⌝ ≡ [*pair*, ⌜**a**⌝, ⌜**b**⌝]
⌜**a**.h⌝ ≡ [*head*, ⌜**a**⌝]
⌜**a**.t⌝ ≡ [*tail*, ⌜**a**⌝]
⌜**if a then b else c**⌝ ≡ [*if*, ⌜**a**⌝, ⌜**b**⌝, ⌜**c**⌝]
⌜**a** • **b**⌝ ≡ [*app*, ⌜**a**⌝, ⌜**b**⌝]

The notation λ**a**, where **a** is an R-term is introduced as a shorthand for the literal ⌜**a**⌝. A functional is intended to denote a *partial recursive* functional over pairs. Because of pairing, all functions and functionals can be treated as one argument only. Multiple arguments can be obtained from the only one argument by selections.

Our functions and functionals are not independent entities but rather names of R-terms. This is consistent with the standard practice of programming languages derived from Lisp. Functionals accept and yield names of functionals as arguments and results. The operation of application **a** • **b** expects the term **a** to denote a name of a R-term.

A closed R-term will be called *computable* (*reducible*) if there is a *computation tree* encoding its computation. Computation trees will be given as literals. We shall define the (informal) relation $Ctree(\overline{p}, \overline{q}, \overline{t})$ holding whenever $\overline{t}$ is a computation tree for the literal $\overline{p}$ in the *environment* $\overline{q}$. We say that a closed R-term $\mathbf{a}\{n{:=}\overline{q}\}$ is *computable* and *reduces* to the literal $\overline{t_1}$ - shortly $\mathbf{a}\{n{:=}\overline{q}\} \Rightarrow \overline{t_1}$ - iff

$$Ctree(\lceil \mathbf{a} \rceil, \overline{q}, [\overline{t_1}, \overline{t_2}])$$

holds for some literal $\overline{t_2}$. The literal $\overline{t_2}$ contains the computation trees for the subterms of **a**. If $\mathbf{a} \Rightarrow \overline{t_1}$ we say that the term **a** *denotes* the pair t_1. The relation *Ctree* is defined by the following clauses where the variables x, r, s, t (possibly subscribed) range over pairs.

$$Ctree([zero, 0], e, [0, 0]) \tag{D1}$$

$$Ctree([var, 0], e, [e, 0]) \tag{D2}$$

$$Ctree([pair, x_1, x_2], e, [[s_1, t_1], [s_1, s_2], t_1, t_2]) \text{ if } Ctree(x_1, e, [s_1, s_2]) \text{ and } Ctree(x_2, e, [t_1, t_2]) \tag{D3}$$

$$Ctree([head, x], e, [0, 0, s]) \text{ if } Ctree(x, e, [0, s]) \tag{D4}$$

$$Ctree([head, x], e, [t_1, [t_1, t_2], s]) \text{ if } Ctree(x, e, [[t_1, t_2], s]) \tag{D5}$$

$$Ctree([tail, x], e, [0, 0, s]) \text{ if } Ctree(x, e, [0, s]) \tag{D6}$$

$$Ctree([tail, x], e, [t_2, [t_1, t_2], s]) \text{ if } Ctree(x, e, [[t_1, t_2], s]) \tag{D7}$$

$$Ctree([if, x_1, x_2, x_3], e, [t_1, [0, s], t_1, t_2]) \text{ if } Ctree(x_1, e, [0, s]) \text{ and } Ctree(x_2, e, [t_1, t_2]) \tag{D8}$$

$$Ctree([if, x_1, x_2, x_3], e, [t_1, [[s_1, s_2], s], t_1, t_2]) \text{ if } Ctree(x_1, e, [[s_1, s_2], s]) \text{ and } Ctree(x_3, e, [t_1, t_2]) \tag{D9}$$

$$Ctree([app, x_1, x_2], e, [t_1, [r_1, r_2], [s_1, s_2], t_1, t_2]) \text{ if } Ctree(x_1, e, [r_1, r_2]) \text{ and } Ctree(x_2, e, [s_1, s_2]) \text{ and } Ctree(r_1, [r_1, s_1], [t_1, t_2]) \tag{D10}$$

We can easily convince ourselves that the truth value of the predicate $Ctree(\overline{p}, \overline{q}, \overline{s})$ is computable for any pairs p, q and s. Namely, if $\overline{p}$ and $\overline{s}$ do not have the form of any of the left-hand sides of clauses then the predicate is false. If, on the other hand, they have the form of one of the clauses then the truth value of $Ctree(\overline{p}, \overline{q}, \overline{s})$ is reduced to that of the predicates on the right-hand side of the corresponding clause. In the case of the match with the clauses (D1) and (D2) the predicate is true. Third arguments of all predicates *Ctree* on right-hand sides are always proper parts of third arguments on left-hand sides. It is impossible to decrease the computation trees forever, so the recursion must always terminate.

The relation of reducibility induces the relation $=$ of identity over closed R-terms.

$\mathbf{a} = \mathbf{b}$ iff for some pair p we have $\mathbf{a} \Rightarrow \overline{p}$ and $\mathbf{b} \Rightarrow \overline{p}$

We should be able to demonstrate that the following holds for any reducible closed R-terms **a**, **b**, **c**, and **d**:

$\mathbf{a}=\mathbf{a}$	(3)
$\mathbf{a}=\mathbf{b} \rightarrow \mathbf{b}=\mathbf{a}$	(4)
$\mathbf{a}=\mathbf{b}\ \&\ \mathbf{b}=\mathbf{c} \rightarrow \mathbf{a}=\mathbf{c}$	(5)
$\mathbf{a}=\mathbf{b} \rightarrow \mathbf{c}=\mathbf{d}$ where **d** is **c** with some occurrences of **a** replaced by **b**	(6)
$[\mathbf{a}, \mathbf{b}].\mathbf{h} = \mathbf{a}$	(7)
$[\mathbf{a}, \mathbf{b}].\mathbf{t} = \mathbf{b}$	(8)
$\mathbf{0}.\mathbf{h} = \mathbf{0}$	(9)
$\mathbf{0}.\mathbf{t} = \mathbf{0}$	(10)

if 0 **then** **c** **else** **d** = **c** (11)

if [**a**, **b**] **then** **c** **else** **d** = **d** (12)

$(\lambda\mathbf{a}) \bullet \mathbf{b} = \mathbf{a}\{n{:=}[\lambda\mathbf{a}, \mathbf{b}]\}$ provided the R.H.S reduces (13)

The idea of the variable *n* should be now obvious. The selection *n*.**t** refers to the only argument of a functional. The R-term *n*.**h** recursively selects the functional $\lambda\mathbf{a}$.

A partial recursive functional $\lambda\mathbf{a}$ is said to be *defined* for the pair p to yield the pair q iff we have

$$(\lambda\mathbf{a}) \bullet \overline{p} \Rightarrow \overline{q} \tag{14}$$

The partial recursive functional $\lambda\mathbf{a}$ is called a *partial recursive function* iff **a** is an RF-term. Partial recursive functions do not permit as left arguments of applications such R-terms as computed functions (for instance another applications) and selections from the argument. A partial recursive function $\lambda\mathbf{a}$ is called a (*total*) *recursive* function iff (14) holds for all pairs p. Note that our definition permits the partial recursive functions to invoke functionals. Recursive functions may invoke partial functionals provided they use them only in the defined points. Our definition is a more liberal one than the standard definition of recursive functions and in this respect it resembles the definition of Turing computability. Obviously, we do not gain any additional power from our definitions. This can be demonstrated by a standard proof of equivalence, say, of Turing and our computability via mutual encodings.

If the reader objects to our identification of partial recursive functions with names of certain R-terms, and thinks it too intensional, we suggest that he defines a function $f(x)$ to be recursive iff there is a pair p such that $f(\overline{q}) = \overline{p} \bullet \overline{q}$ for all pairs q. Then $\overline{p}$ can be vieved both as one of the possible names for f and as an algorithm (program) for the computation of f. In other words, a function is recursive if there is a program computing it.

Let us give some examples of recursive functions. At the same time we devise a notation to improve the readability. Consider the literal abbreviated as *len*

$$len \equiv \lambda\ \mathbf{if}\ n.\mathbf{t}\ \mathbf{then}\ 0\ \mathbf{else}\ [0,\ n.\mathbf{h} \bullet n.\mathbf{t}.\mathbf{t}] \tag{15}$$

Provided that *len* is a recursive function (as it is) we can employ the reduction property (13) to derive

$$len \bullet \mathbf{a} = \mathbf{if}\ \mathbf{a}\ \mathbf{then}\ 0\ \mathbf{else}\ [0,\ len \bullet \mathbf{a}.\mathbf{t}]$$

It is easy to see that *len* computes the length of the list **a**. As a slight notational improvement we can give a name to the recursive invocation and the argument and write

$$\lambda len,\ x\text{: }\mathbf{if}\ x\ \mathbf{then}\ 0\ \mathbf{else}\ [0,\ len \bullet x.\mathbf{t}]$$

This is just a shorthand for the term at the right-hand-side of (15). The next example is a two argumented function concatenating two lists. literal

$$cat \equiv \lambda cat,\ a\text{: }\mathbf{if}\ a.\mathbf{h}\ \mathbf{then}\ a.\mathbf{t}\ \mathbf{else}\ [a.\mathbf{h}.\mathbf{h},\ cat \bullet [a.\mathbf{h}.\mathbf{t},\ a.\mathbf{t}]] \tag{16}$$

In order to escape the tedious selections from the argument we can name the arguments explicitly.

$$\lambda cat,\ [x,\ y]\text{: }\mathbf{if}\ x\ \mathbf{then}\ y\ \mathbf{else}\ [x.\mathbf{h},\ cat \bullet [x.\mathbf{t},\ y]]$$

This is again just a shorthand for the right-hand-side of (16). This notation can be naturally extended to more arguments or even to more complicated forms of arguments, for instance $[[x, y], z]$. Next example is the function *rev* reversing a list. It uses the function *cat*.

$$rev \equiv \lambda rev,\ x\text{: }\mathbf{if}\ x\ \mathbf{then}\ 0\ \mathbf{else}\ cat \bullet [rev \bullet x.\mathbf{t},\ x.\mathbf{h},\ 0] \tag{17}$$

As a further notational convenience we can drop the name of recursive invocations *n*.**h** if a function is explicitly defined.

$$transpose \equiv \lambda[x,\ y]\text{: }[y,\ x] \equiv \lambda[n.\mathbf{t}.\mathbf{t},\ n.\mathbf{t}.\mathbf{h}]$$

We should now prove the identities (3) through (13). We would then obtain an informal theory of recursive functionals over pairs. This "natural" development is, however, not acceptable to us for two reasons.

1) We have on mind explication of programming languages by *formal* semantics. In particular we want be able to reason about our programs in full predicate logic rather than to be restricted to identities among terms. Predicate logic will permit a natural discussion not only of closed R-terms but also of terms containing variables. We also insist on a formal theory because we foresee computer-assisted proofs and transformations of programs.

2) We are not satisfied with the *intensional* character of our identity $=$. It is defined only among reducible terms. The natural property $\mathbf{a}=\mathbf{a}$ does not hold if $\mathbf{a}$ is not reducible.

We shall develop the theory of recursive functionals within the formal framework of the first order Theory of Pairs. We shall introduce R-terms as a subset of terms of a conservative extension of TP. Thus R-terms obtain denotations from the standard model of TP rather than from the notion of reduction. The identity relation over terms which is now induced by the reduction will be interpreted as the natural identity among pairs.

We shall introduce the predicate of computability *Comp* into TP as the arithmetization of the informal notion of computability. We show that all partial recursive functionals can be (numeral-wise) represented within TP. We shall investigate two classes of functionals which can be introduced into TP, i.e. the defining equations for functionals can be proven with variables instead of literals. These are a subclass of recursive functions and of functionals of finite types. We shall not investigate all functionals in TP for two reasons. The first reason is a pragmatic one. The above two classes seem to be sufficient for the definition of declarative programming languages. The second reason is more serious. Because of the inclompleteness of TP we will not be able to introduce all recursive functions into TP. We will be restricted only to the subclass of provably recursive functions.

3. Theory of Pairs as a Formal First Order Theory.

The domain of the intended interpretation of TP is the set of pairs. TP is set up as a first order theory using the terminology and notation of Shoenfield [15]. In order to keep the proofs of formal theorems short we shall present the proofs in a natural deduction style. Such proofs can be readily converted into formal proofs. The language of TP contains - in addition to the symbols for logical connectives, quantifiers, and identity - the constant symbol 0 denoting the pair 0, the binary function symbol $[\_, \_]$ denoting the operation of pairing, and the binary predicate symbol $\_ \in \_$ denoting the relation of being an element of a list.

Formation rules for terms are as follows.

a) constant 0 is a term,

b) individual variables $a, b, \ldots, m, n, \ldots, x, y, z$, possibly primed, are terms,

c) if $\mathbf{a}$ and $\mathbf{b}$ are terms, so is $[\mathbf{a}, \mathbf{b}]$.

For any terms $\mathbf{a}$ and $\mathbf{b}$ the atomic formulas of TP are $\mathbf{a}=\mathbf{b}$ and $\mathbf{a} \in \mathbf{b}$. Other formulas are formed from atomic formulas with the help of connectives and quantifiers in the usual way. We shall use some obvious abbreviations, as for instance, $\mathbf{a} \neq \mathbf{b}$ for $\neg\, \mathbf{a}=\mathbf{b}$ and $\mathbf{a} \notin \mathbf{b}$ for $\neg\, \mathbf{a} \in \mathbf{b}$. The notion of literals and abbreviations for pairs is the same as in section (2). The bold faced variables are syntactic meta-variables ranging over variables ($\mathbf{x}, \mathbf{y}, \ldots$), terms ($\mathbf{a}, \mathbf{b}, \ldots$), and formulas ($\mathbf{A}, \mathbf{B}, \ldots$). We let $\mathbf{p}\{\mathbf{x}:=\mathbf{a}\}$ to stand for the term or formula obtained from the term or formula $\mathbf{p}$ by replacing all free occurrences of the variable $\mathbf{x}$ by the term $\mathbf{a}$ subject to the restriction on the free variables becoming bound in the process of substitution.

The non-logical axioms of TP are the following ones.

$$[x, y] = [x', y'] \rightarrow x = x' \,\&\, y = y' \qquad \text{(Un)}$$

$$x \notin 0 \qquad \text{(Mem1)}$$

$$x \in [y, z] \leftrightarrow x = y \vee x \in z \qquad \text{(Mem2)}$$

$$\mathbf{A}\{\mathbf{x}:=0\} \,\&\, \forall \mathbf{x} \forall \mathbf{y} (\mathbf{A} \,\&\, \mathbf{A}\{\mathbf{x}:=\mathbf{y}\} \rightarrow \mathbf{A}\{\mathbf{x}:=[\mathbf{x}, \mathbf{y}]\}) \rightarrow \mathbf{A} \qquad \text{(Ind)}$$

In (Ind) the variable $\mathbf{y}$ does not occur in the formula $\mathbf{A}$. The axiom (Un) establishes the uniqueness of pairing. Note that the implication in the other direction is a logical axiom of equality. The Axioms (Mem1) and (Mem2) give the properties of the list membership predicate. The

schema of axioms of induction says that all pairs are generated from 0 by pairing in finite number of steps. Recalling the informal discussion of pairs in section (2) we can (non-constructively) see that TP is consistent because its intended interpretation is a model satisfying the non-logical axioms.

Note. The operation of pairing can be introduced into Peano Arithmetic. Thus TP can be interpreted in PA. We shall interpret PA into TP in section (7). Thus we gain only a notational convenience by starting with pairs instead of natural numbers. Notational convenience is what the art of computer programming is about. We hope that we succeed in convincing a logician reader that a good notation for Gödel numbering and for - what computer scientists call data structures - trees, lists, proofs, ordinal numbers etc. can be used with advantage in logic. Although we prefer our axiomatization of pairs for technical reasons, we could have stressed the connection to arithmetic by presenting TP as a *Generalized Arithmetic.* In such a view we start with the number 0 and have infinitely many successors. We write $succ_z(y)$ instead of $[z, y]$. Instead of $\in$ we could have taken the partial order $<$ denoting the relation of "being a predecessor" as primitive.

$$x < [x_1, x_2, \ldots, x]$$

The axioms (Un) and (Ind) would remain and instead of the membership axioms we would have two axioms for $<$ which in PA express the same property.

$$\neg\, x < 0$$
$$x < succ_y(z) \leftrightarrow x \leq z$$

Both Generalized Arithmetic and TP are equivalent since $\in$ has the following explicit definition.

$$x \in y \leftrightarrow \exists z\, [z, z] \leq y$$

On the other hand, $<$ can be introduced into TP by a slightly more complicated definition which we do not give here.

TP will be developped by conservative extensions. We shall extend TP by new predicate and function symbols. This extends the formation rules. We also add new equality axioms for the introduced symbols. The properties of introduced symbols are given by *defining axioms* which are either explicit or contextual (see [15]).

Lists of TP are actually finite sets where the order and repetition of elements matters. Thus some set-theoretical notions are adaptable to lists. For instance, the predicate of being a *sublist* has the following definition.

$$x \subset y \leftrightarrow \forall z(z \in x \rightarrow z \in y) \qquad \text{(D1)}$$

Because of order and repetition there is not a unique union of lists. However, the existence of union lists can be demonstrated.

$$\vdash \exists z \forall w(w \in z \leftrightarrow w \in x \vee w \in y) \qquad \text{(T1)}$$

The proof is by induction on x using (T1) as the induction formula A. In order to prove $A\{x{:=}0\}$ we take $z = y$. In order to prove $A\{x{:=}[x_1, x_2]\}$ we use I.H. to obtain as z_1 a union of the lists x_2 and y. Take $z = [x_1, z_1]$. We have

$$w \in z \leftrightarrow w \in [x_1, z_1] \leftrightarrow w{=}x_1 \vee (w \in x_2 \vee w \in y) \leftrightarrow w \in [x_1, x_2] \vee w \in y.$$

Let us introduce the relation of *being a part* of a pair. It is the natural partial order on pairs introduced with the help of the auxiliary predicate *Cl.*

$$Cl(m) \leftrightarrow \forall v \forall w([v, w] \in m \rightarrow v \in m \,\&\, w \in m) \qquad \text{(D2)}$$
$$x \trianglelefteq y \leftrightarrow \forall m\{Cl(m) \,\&\, y \in m \rightarrow x \in m\} \qquad \text{(D3)}$$

The reflexivity and transitivity of $\trianglelefteq$ follows directly from the definition.

$$\vdash x \trianglelefteq x \qquad \text{(T2)}$$
$$\vdash x \trianglelefteq y \,\&\, y \trianglelefteq z \rightarrow x \trianglelefteq z \qquad \text{(T3)}$$

The relation of being a proper part is introduced as follows.

$$x \triangleleft y \leftrightarrow x \trianglelefteq y \,\&\, x \neq y \qquad \text{(D4)}$$

As the immediate consequence of (D4) and (T2) we obtain

$$\vdash x \trianglelefteq y \leftrightarrow x \lhd y \vee x = y. \tag{T4}$$

Before we can prove the antisymmetry of $\trianglelefteq$ in (T9) we have to prove some auxiliary theorems.

$$\vdash \neg\, x \lhd 0 \tag{T5}$$

Proof: Observe that $Cl([0,0]) \;\&\; (z \in [0,0] \rightarrow z = 0)$. Thus $x = 0 \vee \neg\, x \trianglelefteq 0$, i.e. $\neg\, x \lhd 0$. QED.

$$\vdash x \lhd [y, z] \rightarrow x \trianglelefteq y \vee x \trianglelefteq z \tag{T6}$$

Proof:. Assume $\neg\, x \trianglelefteq y \;\&\; \neg\, x \trianglelefteq z$. Then there are lists m_1 and m_2 such that

$$Cl(m_1) \;\&\; y \in m_1 \;\&\; x \notin m_1 \;\&\; Cl(m_2) \;\&\; z \in m_2 \;\&\; x \notin m_2$$

Denote by $\hat{m}$ a union of lists m_1 and m_2. Consider $m = [[y, z], \hat{m}]$. Clearly $Cl(m)$. Since $x \notin \hat{m}$ we have

$$x = [y, z] \vee \neg\, x \trianglelefteq [y, z]$$

i.e. $\neg\, x \lhd [y, z]$. QED.

$$\vdash x{=}0 \vee \exists y \exists z\; x = [y, z] \tag{T7}$$

The proof is by a straightforward induction on x.

$$\vdash x \lhd [x, y] \;\&\; \neg\, [x, y] \lhd x \;\&\; x \lhd [y, z] \;\&\; \neg\, [y, z] \lhd x \tag{T8}$$

Directly from (D3) we have $x \trianglelefteq [x, y] \;\&\; x \trianglelefteq [y, x]$. Thus we have to prove

$$x \neq [x, y] \;\&\; \neg\, [x, y] \trianglelefteq x \;\&\; x \neq [y, x] \;\&\; \neg\, [y, x] \trianglelefteq x \tag{1}$$

This is proven by induction on x taking $\mathbf{A} \equiv \forall y\,(1)$ and using (T5) and (T6).

$$\vdash x \trianglelefteq y \;\&\; y \trianglelefteq x \rightarrow x = y \tag{T9}$$

Proof: Clearly it suffices to show $\neg\, (x \lhd y \;\&\; y \lhd x)$. By (T7) we can consider two cases. a) $y = 0$. Then (T9) follows from (T5). b) $y = [v, w]$. If also $x \lhd y$ then from (T6) we have $x \trianglelefteq v \vee x \trianglelefteq w$. Since by (T8) $\neg\, y \trianglelefteq v \;\&\; \neg\, y \trianglelefteq w$ we obtain $\neg\, y \trianglelefteq x$ from (T3). This also means $\neg\, y \lhd x$. QED.

Provided **y** does not occur in the formula **A** we have the principle of complete induction over $\lhd$

$$\vdash \forall \mathbf{x}\{\forall \mathbf{y}\,(\mathbf{y} \lhd \mathbf{x} \rightarrow \mathbf{A}\{\mathbf{x}{:=}\mathbf{y}\}) \rightarrow \mathbf{A}\} \rightarrow \mathbf{A}. \tag{T10}$$

Proof: Assume the antecedent. Using the schema of induction (Ind) with the formula

$$\forall \mathbf{y}\,(\mathbf{y} \trianglelefteq \mathbf{x} \rightarrow \mathbf{A}\{\mathbf{x}{:=}\mathbf{y}\}) \tag{2}$$

one proves (2) and thus also **A** because of (T2).

Functions **if _ then _ else _**, **_.h**, and **_.t** are introduced by the following contextual defining axioms.

$$(x = 0 \;\&\; x.\mathbf{h} = 0) \vee \exists y\; x = [x.\mathbf{h}, y] \tag{D5}$$
$$(x = 0 \;\&\; x.\mathbf{t} = 0) \vee \exists y\; x = [y, x.\mathbf{t}] \tag{D6}$$
$$\{x = 0 \;\&\; (\mathbf{if}\ x\ \mathbf{then}\ y\ \mathbf{else}\ z) = y\} \vee \{x \neq 0 \;\&\; (\mathbf{if}\ x\ \mathbf{then}\ y\ \mathbf{else}\ z) = z\} \tag{D7}$$

We leave it to the reader to prove the existence and uniqueness conditions justifying the contextual definitions. Theorems (T11) through (T16) are obvious consequences of defining axioms.

$$\vdash [x, y].\mathbf{h} = x \tag{T11}$$
$$\vdash [x, y].\mathbf{t} = y \tag{T12}$$
$$\vdash 0.\mathbf{h} = 0 \tag{T13}$$
$$\vdash 0.\mathbf{t} = 0 \tag{T14}$$
$$\vdash (\mathbf{if}\ 0\ \mathbf{then}\ z\ \mathbf{else}\ w) = z \tag{T15}$$
$$\vdash (\mathbf{if}\ [x, y]\ \mathbf{then}\ z\ \mathbf{else}\ w) = w \tag{T16}$$

We introduce now the concept of *provably well-founded* relations which play a fundamental role in our introduction of recursive functions over pairs. A binary relation $\ll$ is said to be provably

well-founded (or shortly just well-founded) if the schema (Wfind) of induction over the relation $\ll$ can be demonstrated for every formula **A** where **y** does not occur.

$$\vdash \forall x\{\, \forall y\,(y \ll x \rightarrow A\{x:=y\}) \rightarrow A \,\} \rightarrow A \qquad \text{(Wfind)}$$

We had to adopt this definition instead of the standard set-theoretical one (saying that every non-empty set (list) has a minimal element in the relation) because we cannot quantify over infinite lists.

One can alternatively introduce into TP a unary predicate symbol U (undefined) characterized only by the equality axiom.

$$x{=}y \;\&\; U(x) \rightarrow U(y)$$

This is clearly a conservative extension. The relation $\ll$ can then be defined as well-founded whenever the following is a theorem of TP.

$$\vdash \forall x\{\, \forall y(y \ll x \rightarrow U(y)) \rightarrow U(x) \,\} \rightarrow U(x)$$

The proof of schema (Wfind) can be obtained from the proof of this formula by consistently replacing occurrences of $U(x)$ by **A** in the proof.

Schema of theorems (T10) demonstrates that $\lhd$ is a well-founded relation. The following theorem will be used later.

> **Theorem** (T17). If $\ll$ is a well-founded relation and **a** is a term containing free at most the variable **x** then the relation
>
> $$y \ll' x \leftrightarrow a\{x:=y\} \ll a$$
>
> is well-founded.

Proof: Assume

$$\forall x\{\, \forall y\,(y \ll' x \rightarrow A\{x:=y\}) \rightarrow A \,\}$$

for a formula **A** not containg the variable **y**. By the well-founded induction on $\ll$, variable **y**, and the formula

$$\forall x(a \ll y \vee a = y \rightarrow A) \qquad (3)$$

one can easily prove (3) and also **A** after dropping the quantifier and setting **y**:=**a**.

4. Computation Trees and Function Aplication.

In the previous section we have introduced all functions but the application which are needed to interpret R-terms into TP. In this section we define application. We take the operation of naming from section (2) without any changes. In section (3) we have proven by (T3.11) through (T3.16) the identities (2.7) through (2.12). The reader will note that we interpret the intensional identity between R-terms by the extensional identity of TP. Thus the properties (2.3) through (2.6) hold in TP because of the equality axioms and theorems (see [15]). Before we can introduce the function $\_ \bullet \_$ of application, and demonstrate (2.13) by (T5.19), we have to interpret in TP the notion of computations over R-terms.

We shall now arithmetize the informal predicate of being a computation tree as defined in section (2). A computation tree for the R-term **a** in the environment $\overline{p}$ supplying value for the variable n will have the form

$$[\lceil a \rceil, \overline{p}, [\overline{q}_1, \overline{q}_2]] \qquad (1)$$

where $Ctree(\lceil a \rceil, \overline{p}, [\overline{q}_1, \overline{q}_2])$ holds. An irreducible term has no computation tree. In order to escape the inherent recursiveness in the definition of the predicate *Ctree* we shall introduce the predicate $Trees(m)$ satisfied if (i) m is a list of computation trees of the form (1) and also (ii) if (1) is in m then all computation trees for the terms required for the computation of **a** are also in m.

The predicate *Trees* has the explicit definition

$$Trees(m) \leftrightarrow \forall x\{x \in m \rightarrow (x.\mathbf{h}.\mathbf{h} = zero \;\&\; Zero(x, m) \vee$$

$$x.\mathbf{h}.\mathbf{h} = var \;\&\; Var(x, m) \vee$$
$$x.\mathbf{h}.\mathbf{h} = head \;\&\; Head(x, m) \vee$$
$$x.\mathbf{h}.\mathbf{h} = tail \;\&\; Tail(x, m) \vee$$
$$x.\mathbf{h}.\mathbf{h} = pair \;\&\; Pair(x, m) \vee$$
$$x.\mathbf{h}.\mathbf{h} = if \;\&\; If(x, m) \vee$$
$$x.\mathbf{h}.\mathbf{h} = app \;\&\; App(x, m)\ )\ \} \tag{D1}$$

where the auxiliary predicates are introduced as follows.

$$Zero(x, m) \leftrightarrow \exists y\ x = [[zero, 0], y, [0, 0]] \tag{D2}$$

$$Var(x, m) \leftrightarrow \exists y\ x = [[var, 0], y, [y, 0]] \tag{D3}$$

$$Head(x, m) \leftrightarrow \exists x_1\ \exists y\ \exists z\{x = [[head, x_1], y, [x.\mathbf{h}.\mathbf{h}, z]] \;\&\; [x_1, y, z] \in m\} \tag{D4}$$

$$Tail(x, m) \leftrightarrow \exists x_1\ \exists y\ \exists z\{x = [[tail, x_1], y, [x.\mathbf{h}.\mathbf{t}, z]] \;\&\; [x_1, y, z] \in m\} \tag{D5}$$

$$Pair(x, m) \leftrightarrow \exists x_1\ \exists x_2\ \exists y\ \exists w\ \exists z\{x = [[pair, x_1, x_2], y, [[w.\mathbf{h}, z.\mathbf{h}], w, z]] \;\&\; [x_1, y, w] \in m \;\&\; [x_2, y, z] \in m\} \tag{D6}$$

$$If(x, m) \leftrightarrow \exists x_1\ \exists x_2\ \exists x_3\ \exists y\ \exists w\ \exists z\{x = [[if, x_1, x_2, x_3], y, [z.\mathbf{h}, w, z]] \;\&\; [x_1, y, w] \in m \;\&\; [\mathbf{if}\ w.\mathbf{h}\ \mathbf{then}\ x_2\ \mathbf{else}\ x_3, y, z] \in m\} \tag{D7}$$

$$App(x, m) \leftrightarrow \exists x_1\ \exists x_2\ \exists y\ \exists v\ \exists w\ \exists z\{x = [[app, x_1, x_2], y, [z.\mathbf{h}, v, w, z]] \;\&\; [x_1, y, v] \in m \;\&\; [x_2, y, w] \in m \;\&\; [v.\mathbf{h}, [v.\mathbf{h}, w.\mathbf{h}], z] \in m\} \tag{D8}$$

Note that all seven auxiliary predicates are monotonic in the second argument. For instance, the predicate *If* satisfies the following.

$$\vdash\ m \subset n \;\&\; If(x, m) \rightarrow If(x, n) \tag{T1}$$

Similar theorems can be proven for the other predicates. Following is an obvious theorem.

$$\vdash\ Trees(m) \;\&\; [x, y, z] \in m \rightarrow z \neq 0 \tag{T2}$$

We could now introduce the predicate *Ctree* into TP by the following explicit definition.

$$Ctree(x, n, t) \leftrightarrow \exists m(Trees(m) \;\&\; [x, n, t] \in m)$$

The predicate *Ctree* would be the pair-theoretic counterpart of Kleene's number-theoretic predicate T [5]. In case the reader wonders whether the predicate *Ctree*, defined with an apparently unbounded existential quantification, is decidable let us just note that the list m can be always constructed from the tree t (see the proof of the theorem (T5.7)). Instead of the predicate *Ctree* we shall introduce the semi-decidable predicate of the computability *Comp* obtained by unbounded quantification.

$$Comp(x, n) \leftrightarrow \exists\boxplus m(Trees(m) \;\&\; [x, n, t] \in m) \tag{D9}$$

Note that x is supposed to denote a name (Gödel number) of an R-term.

The literal into which a term reduces should be uniquely determined. We prove a slightly more general uniqueness theorem.

$$\vdash\ Trees(m) \;\&\; [x, y, s] \in m \;\&\; [x, y, s'] \in m \rightarrow s = s' \tag{T3}$$

The proof is by the complete induction on $\lhd$ taking $\mathbf{A} \equiv (\forall x \forall y \forall s'\ T3)$ and $\mathbf{x} \equiv s$. Let us therefore assume $\mathbf{A}\{x{:=}w\}$ for all w s.t. $w \lhd s$. Also assume the antecedent of (T3). We have seven possible values (2.2) for $x.\mathbf{h}$. Let us consider just the case $x.\mathbf{h}{=}app$ here. By $App([x, y, s], m)$ and $App([x, y, s'], m)$ we have for some pairs x_1, x_2, v, w, z, v', w', and z':

$$x = [app, x_1, x_2] \;\&\; s = [z.\mathbf{h}, v, w, z] \;\&\; s' = [z'.\mathbf{h}, v', w', z'] \tag{1}$$

We also have

$$[x_1, y, v], [x_1, y, v'] \in m$$

Now, $v \lhd s$ and by I.H. we have $v=v'$. We also have

$$[x_2, y, w], [x_2, y, w'] \in m$$

Again, $w \lhd s$ and by I.H. we have $w=w'$. Finally, we have

$$[v.\mathbf{h}, [v.\mathbf{h}, w.\mathbf{h}], z], [v'.\mathbf{h}, [v'.\mathbf{h}, w'.\mathbf{h}], z'] \in m$$

Since $v=v'$, $w=w'$, and $z \lhd s$, the induction hypothesis applies and we have $z=z'$. By (1) also $s=s'$. This terminates the proof of the case $x.\mathbf{h}=app$. The other cases are proven similarly.

Next theorem says that for any two lists satisfying *Trees* there is a superlist also satisfying the predicate.

$$\vdash \mathit{Trees}(m) \,\&\, \mathit{Trees}(n) \rightarrow \exists p(m \subset p \,\&\, n \subset p \,\&\, \mathit{Trees}(p)) \qquad \text{(T4)}$$

Proof: Assume the antecedent and take as p any union of lists m and n guaranteed to exist by (T3.1). Assume also $x \in p$. Then $x \in m \vee x \in n$. In any case $x.\mathbf{h}.\mathbf{h}$ must be one of seven constants (2.2), say that $x.\mathbf{h}.\mathbf{h}=if$. Then we have $\mathit{If}(x, m) \vee \mathit{If}(x, n)$. Since we have $m, n \subset p$ we have in any case $\mathit{If}(x, p)$ by the monotonicity theorem (T1). Other cases are similar.

$$\vdash \exists m(\mathit{Trees}(m) \,\&\, [x, y, z] \in m) \,\&\, \exists m'(\mathit{Trees}(m') \,\&\, [x, y, z'] \in m') \rightarrow z = z' \qquad \text{(T5)}$$

Proof: Assume the conditions of theorem. By (T4) there is a list p such that $m, m' \subset p$ and $\mathit{Trees}(p)$. Then $[x, y, z], [x, y, z'] \in p$ and by the uniqueness theorem (T3) we have $z = z'$.

Having demonstrated the uniqueness of computation trees we are now in a position to define the valuation function $\mathit{val}(x, n)$ denoting the unique value obtained by the computation of the R-term named by x in the environment n. Function *val* will be introduced by a contextual definition using the formula (2).

$$(\neg\, \mathit{Comp}(x, n) \,\&\, y=0) \vee \exists m \exists t\{\mathit{Trees}(m) \,\&\, [x, n, [y, t]] \in m\} \qquad (2)$$

It is easy to see that for any x, and n there is at least one y satisfying (2). That there is also at most one is obvious in case $\neg\, \mathit{Comp}(x, n)$. We intend to set the value of $\mathit{val}(x, n)$ artificially to 0 if there is no computation tree for it. In the case when $\mathit{Comp}(x, n)$ is satisfied, i.e. if there is a computation tree for the term named by x, (T5) guarantees that there is at most one y. Thus both the existence and uniqueness conditions for y in (2) are satisfied and we are justified in defining the valuation function *val* by the contextual definition (D10).

$$(\neg\, \mathit{Comp}(x, n) \,\&\, \mathit{val}(x, n)=0) \vee \exists m \exists t\{\mathit{Trees}(m) \,\&\, [x, n, [\mathit{val}(x, n), t]] \in m\} \qquad \text{(D10)}$$

As an immediate consequence we have

$$\vdash \mathit{Trees}(m) \,\&\, [x, n, t] \in m \rightarrow t.\mathbf{h} = \mathit{val}(x, n) \qquad \text{(T6)}$$

We are now in a position to introduce the operation of application by the explicit definition (D11).

$$x \bullet y = \mathit{val}(x, [x, y]) \qquad \text{(D11)}$$

Intuitively, when $x \bullet y$ is computable then $x = \lambda\mathbf{a}$ for an R-term $\mathbf{a}$ and there are lists m and t such that

$$\mathit{Trees}(m) \,\&\, [\lceil \mathbf{a} \rceil, [\lambda\mathbf{a}, y], t] \in m$$

The function named by x is computable (defined) in argument y if $\mathit{Comp}(x, [x, y])$ is satisfied. We introduce a predicate for that.

$$\mathit{Def}(x, y) \leftrightarrow \mathit{Comp}(x, [x, y]) \qquad \text{(D12)}$$

The relation of reduction $\Rightarrow$ of closed R-terms can be introduced as follows.

$$x \Rightarrow y \leftrightarrow \mathit{Comp}(x, 0) \,\&\, y = \mathit{val}(x, 0) \qquad \text{(D13)}$$

Note that x is supposed to denote a name of a closed R-term whereas y directly denotes the reduced value rather than the name of the corresponding literal.

5. Properties of Computable R-terms.

Here are the basic properties of the predicate *Comp*.

$$\vdash Comp([zero, 0], n) \quad (T1)$$
$$\vdash Comp([var, 0], n) \quad (T2)$$
$$\vdash Comp([head, x], n) \leftrightarrow Comp(x, n) \quad (T3)$$
$$\vdash Comp([tail, x], n) \leftrightarrow Comp(x, n) \quad (T4)$$
$$\vdash Comp([pair, x, y], n) \leftrightarrow Comp(x, n) \;\&\; Comp(y, n) \quad (T5)$$
$$\vdash Comp([if, x, y, z], n) \leftrightarrow Comp(x, n) \;\&\; Comp(\textbf{if } val(x, n) \textbf{ then } y \textbf{ else } z, n) \quad (T6)$$
$$\vdash Comp([app, x, y], n) \leftrightarrow Comp(x, n) \;\&\; Comp(y, n) \;\&\; Def(val(x, n), val(y, n)) \quad (T7)$$

Let us prove just (T7). The other proofs are similar.

Proof of ($\rightarrow$): Assume $Comp([app, x, y], n)$. Then for certain lists m and t we have

$$Trees(m) \;\&\; [[app, x, y], n, t] \in m$$

But then for certain lists v, w, and z we have $t = [z.\mathbf{h}, v, w, z]$ and also

$$[x, n, v], \quad [y, n, w], \quad [v.\mathbf{h}, [v.\mathbf{h}, w.\mathbf{h}], z] \in m$$

Thus also

$$Comp(x, n) \;\&\; Comp(y, n) \;\&\; Comp(v.\mathbf{h}, [v.\mathbf{h}, w.\mathbf{h}])$$

The last conjunct means $Def(v.\mathbf{h}, w.\mathbf{h})$, but $v.\mathbf{h} = val(x, n)$ and $w.\mathbf{h} = val(y, n)$, so we have $Def(val(x, n), val(y, n))$.

Proof of ($\leftarrow$): Assume the R.H.S. of (T7). Then for certain lists m_1, v, m_2, w, m_3, and z we have the following.

$$Trees(m_1) \;\&\; [x, n, v] \in m_1$$
$$Trees(m_2) \;\&\; [y, n, w] \in m_2$$
$$Trees(m_3) \;\&\; [val(x, n), [val(x, n), val(y, n)], z] \in m_3$$

Note that $v.\mathbf{h} = val(x, n)$ and $w.\mathbf{h} = val(y, n)$. Applying (T4.4) twice we obtain a list $\hat{m}$ such that

$$m_1, m_2, m_3 \subset \hat{m} \;\&\; Trees(\hat{m})$$

Consider the list

$$m = [[[app, x, y], n, [z.\mathbf{h}, v, w, z]], \hat{m}]$$

It is easy to see that $Trees(m)$. Thus $Comp([app, x, y], n)$. QED.

Since the operation of taking one of the union lists is constructive, the list m in the proofs of converse implications ($\leftarrow$) is obtained constructively. This is used in the proof of the following theorem.

Theorem (T8): If $Ctree(\overline{p}, \overline{q}, \overline{l})$ holds then a pair m such that

$$\vdash Trees(\overline{m}) \;\&\; [\overline{p}, \overline{q}, \overline{l}] \in \overline{m}$$

can be constructively found.

The theorem is proven by the informal induction on the construction of the literal $\overline{l}$. (T8) says that the informal notion of computability is representable within TP. On the other hand, since we hope that TP is consistent, everything provable in TP holds in the standard model. So we can generalize (T8) to (T9).

Representation Theorem (T9): $\vdash Comp(\overline{p}, \overline{q}) \;\&\; val(\overline{p}, \overline{q}) = \overline{l_1}$ iff there is a pair t_2 such that $Ctree(\overline{p}, \overline{q}, [\overline{l_1}, \overline{l_2}])$ holds.

The Representation theorem says that each partial recursive functional is representable in TP. More specifically, the computation of $(\lambda \mathbf{a}) \bullet \overline{p}$ terminates iff $\vdash Def(\lambda \mathbf{a}, \overline{p})$. All partial recursive functionals are literalwise representable in TP. Only a subclass of provably recursive functionals can be introduced into TP with variables instead of literals which amounts by (T19) to proving the functionals total in TP.

The weaker notion of representability is sufficient in logic to prove such proof-theoretical results as incompleteness etc. A computer scientist insists on the introduction of functionals because only then is he able to reason formally about programs and also to transform less efficient programs (called in computer science *specifications*) into equivalent programs which compute faster. Indeed, as we see it, the main difference between the use of recursive functions in logic and in computer science is that a computer scientist, instead of being satisfied with any definition of a recursive function, is always looking for such a formulation of the function which can be computed as efficiently as possible.

The following theorems about the valuation function *val* are proven similarly as the theorems (T1) through (T7).

$$\vdash val([zero, 0], n) = 0 \tag{T10}$$
$$\vdash val([var, 0], n) = n \tag{T11}$$
$$\vdash Comp([head, x], n) \rightarrow val([head, x], n) = val(x, n).\mathbf{h} \tag{T12}$$
$$\vdash Comp([tail, x], n) \rightarrow val([tail, x], n) = val(x, n).\mathbf{t} \tag{T13}$$
$$\vdash Comp([pair, x, y], n) \rightarrow val([pair, x, y], n) = [val(x, n), val(y, n)] \tag{T14}$$
$$\vdash Comp([if, x, y, z], n) \rightarrow val([if, x, y, z], n) = \textbf{if } val(x, n) \textbf{ then } val(y, n) \textbf{ else } val(z, n) \tag{T15}$$
$$\vdash Comp([app, x, y], n) \rightarrow val([app, x, y], n) = val(x, n) \bullet val(y, n) \tag{T16}$$

The following theorem expresses the effects of the repeated use of properties of *Comp* and *val*.

Reflexion Theorem. For any R-term **a** we have

$$\vdash Comp(\lceil \mathbf{a} \rceil, n) \rightarrow val(\lceil \mathbf{a} \rceil, n) = \mathbf{a} \tag{T17}$$

The proof is by the induction on the construction of the R-term **a**. Let us demonstrate just the case when $\mathbf{a} \equiv$ **if b then c else d**.

If $Comp(\lceil \mathbf{a} \rceil, n)$ then by (T6) $Comp(\lceil \mathbf{b} \rceil, n)$ and by I.H. $val(\lceil \mathbf{b} \rceil, n) = \mathbf{b}$. By (T15)

$$val(\lceil \mathbf{a} \rceil, n) = \textbf{if b then } val(\lceil \mathbf{c} \rceil, n) \textbf{ else } val(\lceil \mathbf{d} \rceil, n)$$

If $\mathbf{b} = 0$ then by (T6) $Comp(\lceil \mathbf{c} \rceil, n)$ and by I.H. we have $val(\lceil \mathbf{c} \rceil, n) = \mathbf{c}$. The case $\mathbf{b} \neq 0$ is similar. Thus in any case we have

$$val(\lceil \mathbf{a} \rceil, n) = (\textbf{if b then c else d}) = \mathbf{a}$$

Directly from the Reflexion theorem we obtain the Reduction theorem.

Reduction Theorem: For any closed R-term **a**

$$\vdash \lceil \mathbf{a} \rceil \Rightarrow x \rightarrow \mathbf{a} = x \tag{T18}$$

The counterpart of (2.13) in TP is the theorem about domains of partial recursive functionals.

Domain Theorem. For any functional $\lambda\mathbf{a}$ we have

$$\vdash Def(\lambda\mathbf{a}, x) \rightarrow (\lambda\mathbf{a}) \bullet x = \mathbf{a}\{n:=[\lambda\mathbf{a}, x]\} \tag{T19}$$

Proof: Assume $Def(\lambda\mathbf{a}, x)$. Then $Comp(\lceil \mathbf{a} \rceil, [\lambda\mathbf{a}, x])$. By (D4.11) and Reflection theorem we have

$$(\lambda\mathbf{a}) \bullet x = val(\lceil \mathbf{a} \rceil, [\lambda\mathbf{a}, x]) = \mathbf{a}\{n:=[\lambda\mathbf{a}, x]\}.$$

The next theorem expresses the sufficient conditions for a functional to be provably recursive.

Theorem on Provably Recursive Functionals: For any R-term **a** and a provably well-founded relation $\ll$ the following holds.

$$\vdash \forall n\{\forall y(y \ll n.\mathbf{t} \rightarrow Def(n.\mathbf{h}, y)) \rightarrow Comp(\lceil \mathbf{a} \rceil, n)\} \rightarrow Def(\lambda\mathbf{a}, x) \tag{T20}$$

Proof: Take an R-term **a** and a provably well-founded relation $\ll$ and assume the antecedent. Instantiating it with $n:=[\lambda\mathbf{a}, x]$ and introducing $\forall x$ yields

$$\forall x\{\forall y(y \ll x \rightarrow Def(\lambda\mathbf{a}, y)) \rightarrow Def(\lambda\mathbf{a}, x)\}.$$

$Def(\lambda\mathbf{a}, x)$ follows from a suitably instantiated schema of (3.Wfind) for the well-founded relation

$\prec$. QED.

(T20) is a powerful theorem but a proof of $Comp(\ulcorner \mathbf{a} \urcorner, n)$ can be quite tedious. The Application theorem (T21) makes the proof easier. We need first some auxiliary notions.

We say that the formula $\mathbf{c}=0$ ($\mathbf{c} \neq 0$) *governs* an occurrence of the subterm **b** of an R-term **a** if the occurrence of **b** occurs in the term **d** (**e**) such that **if c then d else e** is a subterm of **a**. The *governing formula* for an occurrence of the subterm **b** of an R-term **a** is a conjunction of all formulas governing that occurrence of **b**. The order in which the conjunctions are taken is not important but we have to settle for a particular one. We take the conjuncts in the order in which the above terms **c** occur in **a**. If one term includes another one then the smaller one is taken first. On the other hand, if there are no formulas governing the occurrence of a subterm **b** we take the formula $0 = 0$ as its governing formula.

For an R-term **a** let us denote by $Alldef_{\mathbf{a}}$ the conjunction (in the same order as above) of all formulas $G \rightarrow Def(\mathbf{b}, \mathbf{c})$ where $\mathbf{b} \bullet \mathbf{c}$ is a subterm of **a** and G is its governing formula. Set $Alldef_{\mathbf{a}} \equiv (0=0)$ if there are no applications in **a**. Note that $Alldef_{\mathbf{a}}$ contains at most the variable n free.

The Application theorem says that an R-term is computable iff all of its applications are defined.

Application Theorem. For any R-term **a** we have

$$\vdash Comp(\ulcorner \mathbf{a} \urcorner, n) \leftrightarrow Alldef_{\mathbf{a}} \qquad \text{(T21)}$$

The proof is by the induction on the construction of the term **a**. The cases 0, n, **b.h**, and **b.t** are straightforward. We prove the last three cases.

$\mathbf{a} \equiv [\mathbf{b}, \mathbf{c}]$. It is easy to see that $Alldef_{\mathbf{a}} \leftrightarrow Alldef_{\mathbf{b}} \,\&\, Alldef_{\mathbf{c}}$. By (T5) and I.H. we have

$$Comp(\ulcorner \mathbf{a} \urcorner, n) \leftrightarrow Comp(\ulcorner \mathbf{b} \urcorner, n) \,\&\, Comp(\ulcorner \mathbf{c} \urcorner, n) \leftrightarrow Alldef_{\mathbf{b}} \,\&\, Alldef_{\mathbf{c}} \leftrightarrow Alldef_{\mathbf{a}}$$

$\mathbf{a} \equiv$ **if b then c else d**. After some reflexion we can see that

$$Alldef_{\mathbf{a}} \leftrightarrow Alldef_{\mathbf{b}} \,\&\, (\mathbf{b} = 0 \rightarrow Alldef_{\mathbf{c}}) \,\&\, (\mathbf{b} \neq 0 \rightarrow Alldef_{\mathbf{d}})$$

Using (T6), Reflexion theorem, and I.H. we have

$$Comp(\ulcorner \mathbf{a} \urcorner, n) \leftrightarrow Comp(\ulcorner \mathbf{b} \urcorner, n) \,\&\, Comp(\ulcorner \textbf{if b then } \mathbf{c} \textbf{ else } \mathbf{d} \urcorner, n) \leftrightarrow$$
$$Alldef_{\mathbf{b}} \,\&\, (\mathbf{b} = 0 \rightarrow Comp(\ulcorner \mathbf{c} \urcorner, n) \,\&\, (\mathbf{b} \neq 0 \rightarrow Comp(\ulcorner \mathbf{d} \urcorner, n) \leftrightarrow$$
$$Alldef_{\mathbf{b}} \,\&\, (\mathbf{b} = 0 \rightarrow Alldef_{\mathbf{c}}) \,\&\, (\mathbf{b} \neq 0 \rightarrow Alldef_{\mathbf{d}}) \leftrightarrow Alldef_{\mathbf{a}}$$

$\mathbf{a} \equiv \mathbf{b} \bullet \mathbf{c}$. We can easily see that

$$Alldef_{\mathbf{a}} \leftrightarrow Alldef_{\mathbf{b}} \,\&\, Alldef_{\mathbf{c}} \,\&\, Def(\mathbf{b}, \mathbf{c})$$

Using (T7), Reflexion Theorem, and I.H. we have

$$Comp(\ulcorner \mathbf{a} \urcorner, n) \leftrightarrow Comp(\ulcorner \mathbf{b} \urcorner, n) \,\&\, Comp(\ulcorner \mathbf{c} \urcorner, n) \,\&\, Def(\mathbf{b}, \mathbf{c}) \leftrightarrow$$
$$Alldef_{\mathbf{b}} \,\&\, Alldef_{\mathbf{c}} \,\&\, Def(\mathbf{b}, \mathbf{c}) \leftrightarrow Alldef_{\mathbf{a}}$$

For an R-term **a** and a binary relation $\prec$ let us denote by $Down_{\mathbf{a}}^{\prec}$ the conjunction (in the usual order) of all formulas $G \rightarrow \mathbf{b} \prec n.\mathbf{t}$ where the term $n.\mathbf{h} \bullet \mathbf{b}$ occurs in **a** and G is its governing formula. Set $Down_{\mathbf{a}}^{\prec} \equiv 0 = 0$ if there are no such terms. $Down_{\mathbf{a}}^{\prec}$ says that recursive calls in the term **a** use as arguments values less than $n.\mathbf{t}$ in the relation $\prec$. We are now able to formulate a theorem simplifying the proofs of provability of recursive functions.

Theorem on Provably Recursive Functions. For any RF-term **a** and a provably well-founded relation $\prec$ such that all functionals called in **a** are provably recursive (i.e. for any literal **b** such that $\mathbf{b} \bullet \mathbf{c}$ occurs in **a** we have $\vdash Def(\mathbf{b}, x)$) and the arguments of recursion are driven down in the relation $\prec$ (i.e. $\vdash Down_{\mathbf{a}}^{\prec}$), the function $\lambda\mathbf{a}$ is provably recursive, i.e. the following holds.

$$\vdash Def(\lambda\mathbf{a}, x)$$
$$\vdash (\lambda\mathbf{a}) \bullet x = \mathbf{a}\{n:=[\lambda\mathbf{a}, x]\} \qquad \text{(T22)}$$

Proof: Assume the conditions of the theorem. If G is the governing formula of an occurrence of the term $\mathbf{b} \bullet \mathbf{c}$ in **a** the following holds.

$$\vdash \forall y(y \ll n.\mathbf{t} \rightarrow Def(n.\mathbf{h}, y)) \;\&\; G \rightarrow Def(\mathbf{b}, \mathbf{c}) \qquad (1)$$

This is because there are two possibilities.

$\mathbf{b} \equiv n.\mathbf{h}$. Then from $\vdash Down_{\mathbf{a}}^{\ll}$ we have $\vdash G \rightarrow \mathbf{c} \ll n.\mathbf{t}$. (1) directly follows from this.

$\mathbf{b} \not\equiv n.\mathbf{h}$. Then $\mathbf{b}$ must be a literal and (1) follows directly from $\vdash Def(\mathbf{b}, x)$.

Since we have proven (1) for all occurrences of subterms $\mathbf{b} \bullet \mathbf{c}$ in $\mathbf{a}$, we also have

$$\vdash \forall y(y \ll n.\mathbf{t} \rightarrow Def(n.\mathbf{h}, y)) \rightarrow Alldef_{\mathbf{a}}$$

Note that this also includes the case when there are no applications in $\mathbf{a}$. Application theorem now applies and after the introduction of $\forall n$ we obtain.

$$\vdash \forall n\{\forall y(y \ll n.\mathbf{t} \rightarrow Def(n.\mathbf{h}, y)) \rightarrow Comp(\lceil \mathbf{a} \rceil, n)\}$$

The theorem is now proven by a simple application of theorems (T20) and (T19). QED.

We shall now demonstrate that the functions *len*, *cat*, and *rev* defined in section (2) are provably recursive. The formula

$$Down_{\mathbf{a}}^{\lhd} \equiv n.\mathbf{t} \neq 0 \rightarrow n.\mathbf{t}.\mathbf{t} \lhd n.\mathbf{t} \qquad (2)$$

is the same one for both $\mathbf{a} \equiv len$ and $\mathbf{a} \equiv rev$ and is provable by (T3.8). The relation $\lhd$ is of course well-founded (T3.10). The are no auxiliary functions invoked in *len*. Thus (T22) applies to yield

$$\vdash len \bullet x = \textbf{if } x \textbf{ then } 0 \textbf{ else } [0, len \bullet x.\mathbf{t}]. \qquad (T23)$$

We have

$$Down_{cat}^{\ll} \equiv n.\mathbf{t}.\mathbf{h} \neq 0 \rightarrow [n.\mathbf{t}.\mathbf{h}.\mathbf{t}, n.\mathbf{t}.\mathbf{t}] \ll n.\mathbf{t} \qquad (3)$$

If we take $x \ll y \leftrightarrow x.\mathbf{h} \lhd y.\mathbf{h}$ we can easily prove (3). By (T3.17) we know that $\ll$ is well-founded. There are no auxiliary functions called in *cat*. Thus (T22) applies to yield after setting $x{:=}[x, y]$

$$\vdash Def(cat, x) \qquad (T24)$$
$$\vdash cat \bullet [x, y] = \textbf{if } x \textbf{ then } y \textbf{ else } [x.\mathbf{h}, cat \bullet [x.\mathbf{t}, y]]. \qquad (T25)$$

In order to demonstrate

$$\vdash rev \bullet x = \textbf{if } x \textbf{ then } 0 \textbf{ else } cat \bullet [rev \bullet x.\mathbf{t}, [x.\mathbf{h}, 0]] \qquad (T26)$$

we note that for $\mathbf{a} \equiv rev$ the property (2) holds and that the function *cat* invoked in *rev* is recursive by (T24).

Typical applications of the theorem (T22) will contain only invocations of already proven recursive functions. Thus one will only have to point out the measure of the argument (in the case of *cat* _.**h**) and a well-founded relation in which the measure is driven down by the recursive calls. More often than not it will be the well-founded ordering $<$ of ordinals defined in next section.

6. Ordinal Numbers.

In order to introduce provably recursive functionals into TP in Part II we need provably well-founded relations admitting descending sequences substantially longer than the ones obtained from the relation $\lhd$. Ordinals less than ϵ_0 will supply such relations. We shall now introduce these ordinals into TP. The list $[a_1, a_2, \ldots, a_n, 0]$ will denote the ordinal

$$\omega^{a_1} + \omega^{a_2} + \cdots + \omega^{a_n}$$

provided all a_i are ordinals and $a_{i+1} \leq a_i$ for all $1 \leq i < n$.

Ordinals will be introduced with the help of a couple of recursive functions. The functions will actually be characteristic functions of predicates we want to introduce into TP. In order to make the constant transfer between predicates and their characteristic functions easier we shall adopt certain conventions. Let us denote by $\dot{\mathbf{p}}$ the characteristic function of an n-ary predicate $\mathbf{p}$. We say that the predicate $\mathbf{p}$ is *introduced by its characteristic function* if $\dot{\mathbf{p}}$ is a provably recursive function and the defining axiom for $\mathbf{p}$ is as follows.

$$\mathbf{p}(x_1, x_2, \ldots, x_n) \leftrightarrow \dot{\mathbf{p}} \bullet [x_1, x_2, \ldots, x_n] = 0 \tag{1}$$

Some binary predicates (and functions), as for instance $<$, will be introduced as infix predicates. Then we shall write not only $\mathbf{a}<\mathbf{b}$ instead of $<(\mathbf{a}, \mathbf{b})$ but also $\mathbf{a} \dot{<} \mathbf{b}$ instead of $\dot{<} \bullet [\mathbf{a}, \mathbf{b}]$.

We shall also introduce characteristic functions $\ddot{\vee}$, $\dot{\&}$, $\dot{\neg}$, and $\dot{=}$ for the corresponding connectives as well as identity. Since these symbols are already in TP we have to correlate them to the characteristic functions by theorems rather than by definitions.

$$\dot{\&} \equiv \lambda[x, y]: \textbf{if } x \textbf{ then } y \textbf{ else } 1 \tag{D1}$$
$$\ddot{\vee} \equiv \lambda[x, y]: \textbf{if } x \textbf{ then } 0 \textbf{ else } y \tag{D2}$$
$$\dot{\neg} \equiv \lambda x: \textbf{if } x \textbf{ then } 1 \textbf{ else } 0 \tag{D3}$$
$$\dot{=} \equiv \lambda \dot{=}, [x, y]: \textbf{if } x \textbf{ then } (\textbf{if } y \textbf{ then } 0 \textbf{ else } 1) \textbf{ else if } y \textbf{ then } 1 \textbf{ else } x.\mathbf{h} \dot{=} y.\mathbf{h} \dot{\&} x.\mathbf{t} \dot{=} y.\mathbf{t} \tag{D4}$$

The characteristic functions of connectives are obviously recursive. The function $\dot{=}$ drives the argument of recursion down in the relation $\lhd$ (the measure is $\_.\mathbf{h}$). We have the following theorems.

$$\vdash (x \dot{\&} y) = 0 \leftrightarrow x = 0 \,\&\, y = 0 \tag{T1}$$
$$\vdash (x \ddot{\vee} y) = 0 \leftrightarrow x = 0 \vee y = 0 \tag{T2}$$
$$\vdash (\dot{\neg} x) = 0 \leftrightarrow \neg x = 0 \tag{T3}$$
$$\vdash (x \dot{=} y) = 0 \leftrightarrow x = y \tag{T4}$$

The first three theorems are obvious, (T4) is proven by the ordinary induction on x using $(\mathbf{A} \equiv \forall y T4)$.

We can generalize the use of these theorems and the property (1) of characteristic functions. Let $\mathbf{A}$ be a formula without quantifiers consisting at most of the connectives $\&$, $\vee$, $\neg$, and only such predicates which have been introduced by their characteristic functions. We call the term $\mathbf{a}$ the *characteristic term* of the formula $\mathbf{A}$ if $\mathbf{a}$ is as $\mathbf{A}$ with the connectives and predicates replaced by their dotted counterparts. It is easy to see that $\vdash \mathbf{A} \leftrightarrow \mathbf{a} = 0$.

Let us introduce the binary infix predicate $<_{lex}$ of lexicographic ordering of pairs by its characteristic function.

$$\dot{<}_{lex} \equiv \lambda \dot{<}_{lex}, [x, y]: \textbf{if } y \textbf{ then } 1 \textbf{ else if } x \textbf{ then } 0 \textbf{ else } x.\mathbf{h} \dot{<}_{lex} y.\mathbf{h} \ddot{\vee} (x.\mathbf{h} \dot{=} y.\mathbf{h} \dot{\&} x.\mathbf{t} \dot{<}_{lex} y.\mathbf{t}) \tag{D5}$$

The function $\dot{<}_{lex}$ is recursive because it uses only recursive functions and the argument of recursion is driven down by the same measure as the argument of $\dot{=}$. As can be easily seen the relation $<_{lex}$ totally orders pairs. It would be nice if the predicate were a well-ordering. Unfortunately this is not the case as can be seen from the following infinite descending chain.

$$\cdots <_{lex} [0, 0, 0, \omega] <_{lex} [0, 0, \omega] <_{lex} [0, \omega] <_{lex} \omega \tag{2}$$

where $\omega \equiv [1, 0] \equiv [[0, 0], 0]$ (see below). Fortunately, the situation can be remedied by lexicographically ordering only a subset of pairs called *ordinals less than* ϵ_0, or shortly just *ordinals*. Ordinals are pairs containing a subpair $[a, b, c]$ only if b is not lexicographically greater than a. Thus, with the exception of ω, no pairs in the chain (2) are ordinals. The unary predicate *Ord* of being an ordinal is introduced by the characteristic function $\dot{Ord}$ which is easily seen to be recursive.

$$\dot{Ord} \equiv \lambda \dot{Ord}, x: \textbf{if } x \textbf{ then } 0 \textbf{ else } \dot{Ord}(x.\mathbf{h}) \dot{\&} \dot{Ord}(x.\mathbf{t}) \dot{\&} \dot{\neg} x.\mathbf{h} \dot{<}_{lex} x.\mathbf{t}.\mathbf{h} \tag{D6}$$

The binary infix relations $<$ and $\leq$ of lexicographic ordering restricted to the ordinals are defined by their characteristic (and obviously recursive) functions.

$$\dot{<} \equiv \lambda[x, y]: \dot{Ord}(x) \dot{\&} \dot{Ord}(y) \dot{\&} x \dot{<}_{lex} y \tag{D7}$$
$$\dot{\leq} \equiv \lambda[x, y]: x < y \ddot{\vee} x \dot{=} y \tag{D8}$$

We have the following theorems.

$$\vdash Ord(0) \tag{T5}$$
$$\vdash Ord([x, y]) \leftrightarrow Ord(x) \,\&\, Ord(y) \,\&\, y.\mathbf{h} \leq x \tag{T6}$$

$$\vdash \neg\, x<0 \tag{T7}$$
$$\vdash 0<[x, y] \leftrightarrow Ord([x, y]) \tag{T8}$$
$$\vdash [x, y]<[x', y'] \leftrightarrow Ord([x, y]) \;\&\; Ord([x', y']) \;\&\; (x<x' \vee x = x' \;\&\; y<y') \tag{T9}$$
$$\vdash x\leq y \leftrightarrow x<y \vee x = y \tag{T10}$$
$$\vdash x<y \rightarrow \neg\, y\leq x \tag{T11}$$
$$\vdash x<y \;\&\; y<z \rightarrow x<z \tag{T12}$$
$$\vdash Ord(x) \;\&\; Ord(y) \rightarrow x<y \vee y<x \vee x=y \tag{T13}$$

Proofs: All theorems but (T11) and (T13) are straightforward consequences of definitions. (T11) and (T13) are proven by induction (3.Ind) on y using ($\forall x$ T11), resp. ($\forall x$ T13) as induction formulas. The reader will note that the auxiliary predicate $<_{les}$ is needed to escape the mutual recursiveness of predicates *Ord* and $<$.

The relation $<$ is a well-founded relation over pairs, but unfortunately it is not provably well-founded. The schema of well-founded induction

$$\forall \mathbf{x}\{\forall \mathbf{y}(\mathbf{y}<\mathbf{x} \rightarrow \mathbf{A}\{\mathbf{x}:=\mathbf{y}\}) \rightarrow \mathbf{A}\} \rightarrow \mathbf{A} \tag{Wford}$$

is true in the standard model but unprovable in TP. As will be seen in section (8) (Wford) is provable for each literal $\bar{p}$ substituted for **x**. However, if we restrict the relation $<$ to ordinals less than certain ordinal by the definition

$$x <_p y \leftrightarrow x<y \;\&\; y<p \tag{D9}$$

then (Wford) with $<_{\bar{p}}$ instead of $<$ is provable for any ordinal $\bar{p}$. This is the main theorem (T8.10) proven in section (8).

7. Ordinal Arithmetic.

We introduce in this section the operations of ordinal addition, subtraction, multiplication, and division. The operations specialize to natural numbers. Instead of giving the full proofs of the expected properties of arithmetic operations which are mostly done by the ordinary induction (3.Ind) we only sketch the proofs by indicating the induction variables and formulas. Since our ordinals are represented in a variant of a Cantor's normal form, the properties of arithmetic operations have to be proven by a rather tedious case analysis. This can be compared to the proofs of properties of arithmetic operations from their decimal notation rather than from the successor representation of natural numbers in PA.

If the reader finds this section not sufficiently detailed he is referred to the standard treatments of ordinals in the arithmetic as given in the texts of Takeuti and Schuette [16,14]. He will find that their treatment is even sketchier. On the other hand, our constructive treatment of ordinals should not be compared to a standard set-theoretical treatment which is more elegant because the transfinite induction on ordinals in set theory is readily available from the Axiom of Foundation. We are trying to establish the transfinite induction using the ordinary induction over pairs. We have done all the proofs in this section in detail and we urge a logician reader to attempt at least a couple of proofs to obtain an insight into the typical work of a computer scientist trying to prove properties of his programs. The tedious case analysis of such proofs comes from the fact that the programs are highly optimized and consequently contain many **if – then – else** operators. Indeed our insistence on a formal development of TP is motivated by our desire to have the computers perform the tedious work of the mostly mechanical case analysis.

The infix operation of ordinal addition is introduced as follows.

$$+ \equiv \lambda +, [x, y]: \textbf{if } x \doteq 0 \;\dot{\vee}\; x.\mathbf{h} \dot{<} y.\mathbf{h} \textbf{ then } y \textbf{ else } [x.\mathbf{h},\, x.\mathbf{t}+y] \tag{D1}$$

It is obvious that the operation is recursive.

$$\vdash 0+x = x \tag{T1}$$
$$\vdash x<x.\mathbf{h} \rightarrow [x, y]+z = z \tag{T2}$$
$$\vdash x.\mathbf{h}\leq x \rightarrow [x, y]+z = [x, y+z] \tag{T3}$$

These theorems follow directly from the definition of addition.

$\vdash Ord(x) \,\&\, Ord(y) \to Ord(x+y)$ (T4)
$\vdash Ord(x) \to x+0 = x$ (T5)
$\vdash Ord(x) \,\&\, Ord(y) \,\&\, Ord(z) \to (x+y)+z = x+(y+z)$ (T6)
$\vdash x<x' \,\&\, Ord(y) \to y+x<y+x'$ (T7)
$\vdash Ord(x) \,\&\, Ord(y) \to x\leq y+x$ (T8)
$\vdash x<x' \,\&\, Ord(y) \to x+y\leq x'+y$ (T9)
$\vdash Ord(x) \,\&\, Ord(y) \to x<y+1 \leftrightarrow x\leq y$ (T10)

(T4) is proven by the induction on x with $\mathbf{A} \equiv (T4)$ using the easy to prove lemma

$\vdash Ord(x) \,\&\, Ord(y) \to (x+y).\mathbf{h} = x.\mathbf{h} \vee (x+y).\mathbf{h} = y.\mathbf{h}.$ (1)

(T5) is proven by the induction on x with $\mathbf{A} \equiv (T5)$. (T6) is proven with the help of (1) by the induction on x with $\mathbf{A} \equiv (T6)$. (T7) is proven by the induction on y with $\mathbf{A} \equiv (T7)$. (T8) is proven by the induction on y with $\mathbf{A} \equiv (T8)$. (T9) is proven by the induction on x with $\mathbf{A} \equiv (\forall x'\ T9)$. (T10) is proven by the induction on y with $\mathbf{A} \equiv (\forall x\ T10)$.

The infix operation of ordinal subtraction is introduced as follows.

$- \equiv \lambda -, [x, y]$: **if** $x \doteq 0 \vee y \doteq 0 \vee \dot{\neg}\, x.\mathbf{h} \doteq y.\mathbf{h}$ **then** x **else** $x.\mathbf{t}-y.\mathbf{t}$ (D2)

Subtraction is obviously a recursive function.

$\vdash x-0 = x \,\&\, 0-x = 0$ (T11)
$\vdash [x, y]-[x, z] = y-z$ (T12)
$\vdash x_1 \neq y_1 \to [x_1, x_2]-[y_1, y_2] = [x_1, x_2]$ (T13)

These theorems follow directly from the definition.

$\vdash Ord(x) \,\&\, Ord(y) \to Ord(x-y)$ (T14)
$\vdash y\leq x \to y+(x-y) = x$ (T15)

(T14) is proven by the induction on x with $\mathbf{A} \equiv (T14)$. For the proof of (T15) we need the lemma

$\vdash Ord(x) \,\&\, Ord(y) \to (x-y).\mathbf{h}\leq x.\mathbf{h}$ (2)

proven by the induction on x with $\mathbf{A} \equiv \forall y\,(2)$. The proof of (T15) is by the induction on y with $\mathbf{A} \equiv (\forall x\ T15)$.

The infix operation of ordinal multiplication is introduced as follows.

$\times \equiv \lambda \times, [x, y]$:
if $x \doteq 0 \vee y \doteq 0$ **then** 0 **else** $[x.\mathbf{h}+y.\mathbf{h},$ **if** $y.\mathbf{h}$ **then** $x.\mathbf{t}$ **else** $0] + x\times y.\mathbf{t}$ (D3)

Multiplication is obviously a recursive function.

$\vdash x\times 0 = 0 \,\&\, 0\times x = 0$ (T16)
$\vdash x\times[0, y] = x+x\times y$ (T17)
$\vdash 0<x \,\&\, 0<y \to x\times[y, z] = [x.\mathbf{h}+y, 0]+x\times z$ (T18)

These theorems follow directly from the definition.

$\vdash Ord(x) \,\&\, Ord(y) \to Ord(x\times y)$ (T19)
$\vdash Ord(x) \,\&\, Ord(y) \,\&\, Ord(z) \to x\times(y+z) = x\times y + x\times z$ (T20)
$\vdash x<x' \,\&\, 0<y \to y\times x<y\times x'$ (T21)
$\vdash x<x' \,\&\, Ord(y) \to x\times y\leq x'\times y$ (T22)
$\vdash Ord(x) \,\&\, Ord(y) \,\&\, Ord(z) \to (x\times y)\times z = x\times(y\times z)$ (T23)

(T19) is proven by the induction on y with $\mathbf{A} \equiv (T19)$. For the proof of (T20), (T21), and (T23) we need the lemma

$\vdash 0<x \,\&\, 0<y \to (x\times y).\mathbf{h} = x.\mathbf{h}+y.\mathbf{h}$ (3)

which is proven by the induction on y with $\mathbf{A} \equiv$ (3). (T20) is proven by the induction on y with $\mathbf{A} \equiv (T20)$. (T21) is proven by the induction on x with $\mathbf{A} \equiv (\forall x'\ T21)$. (T22) is proven by the induction on y with $\mathbf{A} \equiv (T22)$. The proof of (T23) requires (3), (T20), and (T21); it is by the induction on z with $\mathbf{A} \equiv (T23)$.

The infix operation of ordinal division is introduced as follows.

$$/ \equiv \lambda /, [x, y]:$$
$$\textbf{if } 0 \dot{<} y \;\&\; y \dot{\leq} x \textbf{ then } [x.\mathrm{h}\dot{-}y.\mathrm{h},\, 0] + (\textbf{if } x.\mathrm{h} \doteq y.\mathrm{h} \textbf{ then } x\dot{-}y \textbf{ else } x.\mathrm{t})/y \textbf{ else } 0 \qquad \text{(D4)}$$

In order to prove the division recursive we need the lemma

$$\vdash x\dot{-}y \dot{\leq} x \qquad (4)$$

which is proven by the induction on y with $\mathbf{A} \equiv \forall x\,(4)$. Next we note that

$$\vdash x \neq 0 \;\&\; y \neq 0 \;\&\; x.\mathrm{h} = y.\mathrm{h} \rightarrow x\dot{-}y \lhd x$$

Thus the recursion in the division drives the measure _.h down in the well-founded relation $\lhd$.

$$\vdash x<y \rightarrow x/y = 0 \qquad \text{(T24)}$$
$$\vdash [x, z]\leq[x, y] \rightarrow [x, y]/[x, z] = 1+(y\dot{-}z)/[x, z] \qquad \text{(T25)}$$
$$\vdash [y_1, y_2]\leq[x_1, x_2] \;\&\; y_1<x_1 \rightarrow [x_1, x_2]/[y_1, y_2] = [x_1\dot{-}y_1, 0]+x_2/[y_1, y_2] \qquad \text{(T26)}$$

These theorems follow directly from the definition of division.

$$\vdash Ord(x) \;\&\; Ord(y) \rightarrow Ord(x/y) \qquad \text{(T27)}$$
$$\vdash Ord(x) \;\&\; 0<y \rightarrow y\times(x/y)\leq x \;\&\; x<y\times(x/y+1) \qquad \text{(T28)}$$

Both theorems are proven by the complete induction over $\lhd$ (T3.10) on the variable x using the theorems themselves as induction formulas.

Ordinal notations in Peano arithmetic encode ordinals into natural numbers and thus the finite ordinals are different from the corresponding natural numbers. Our ordinals include natural numbers and the arithmetic operations over ordinals behave as expected with natural numbers. Let us define the first infinite ordinal ω and the predicate N of being a natural number as follows.

$$\omega \equiv [1, 0] \qquad \text{(D5)}$$
$$N(x) \leftrightarrow x<\omega \qquad \text{(D6)}$$

We have the following properties of natural numbers.

$$\vdash Ord(\omega) \qquad \text{(T29)}$$
$$\vdash N(0) \qquad \text{(T30)}$$
$$\vdash N([x, y]) \leftrightarrow x = 0 \;\&\; N(y) \qquad \text{(T31)}$$
$$\vdash N(x) \;\&\; N(y) \rightarrow N(x+y) \qquad \text{(T32)}$$
$$\vdash N(x) \;\&\; N(y) \rightarrow N(x\dot{-}y) \qquad \text{(T33)}$$
$$\vdash N(x) \;\&\; N(y) \rightarrow N(x\times y) \qquad \text{(T34)}$$
$$\vdash N(x) \;\&\; N(y) \rightarrow N(x/y) \qquad \text{(T35)}$$
$$\vdash N(x) \rightarrow x+1 = [0, x] \qquad \text{(T36)}$$
$$\vdash \mathbf{A}\{\mathbf{x}:=0\} \;\&\; \forall \mathbf{x}\,(N(\mathbf{x}) \;\&\; \mathbf{A} \rightarrow \mathbf{A}\{\mathbf{x}:=\mathbf{x}+1\}) \;\&\; N(\mathbf{x}) \rightarrow \mathbf{A} \qquad \text{(T37)}$$

(T29) and (T30) are obvious. For ($\rightarrow$) of (T31) we have from $N([x, y])$ immediately $x = 0$. That also $N(y)$ is proven by the induction on y. For the converse ($\leftarrow$) we observe that from $y<\omega$ we have $y.\mathrm{h} = 0$ and thus $Ord([0, y])$. (T32) is proven by the induction on x with $\mathbf{A} \equiv (T32)$. (T33) requires induction on x with $\mathbf{A} \equiv (\forall y\ T33)$. (T34) is proven by the induction on y with $\mathbf{A} \equiv (T34)$. (T35) requires the complete induction over $\lhd$ on the variable x with $\mathbf{A} \equiv (T35)$. (T36) is proven by the induction on x with $\mathbf{A} \equiv (T36)$. The schema of the induction over natural numbers (T37) is proven by the ordinary induction on $\mathbf{x}$ using the induction formula $N(\mathbf{x}) \rightarrow \mathbf{A}$.

8. Transfinite Induction and Functions Defined by Transfinite Induction.

In this section we investigate the provability of transfinite induction in TP. We show that transfinite induction is provable for any ordinal given as a literal (T9). The idea of the proof is adapted from Schuette [14].

In order to simplify the discussion let us use the following abbreviations.

$$\mathbf{A}(\mathbf{a}) \equiv \mathbf{A}\{\mathbf{x}:=\mathbf{a}\}$$
$$\mathbf{A}^*(\mathbf{x}) \equiv \forall \mathbf{y}\,(\mathbf{y}<\mathbf{x} \rightarrow \mathbf{A}(\mathbf{y}))$$
$$\mathbf{P} \equiv \forall \mathbf{x}\,(\mathbf{A}^*(\mathbf{x}) \rightarrow \mathbf{A}(\mathbf{x}))$$

$\mathbf{B} \equiv \mathit{Ord}(\mathbf{x}) \rightarrow \forall \mathbf{y}\{\mathit{Ord}(\mathbf{y}) \,\&\, \mathbf{A}^*(\mathbf{y}) \rightarrow \mathbf{A}^*(\mathbf{y}+[\mathbf{x}, 0])\}$
$\mathbf{B}(\mathbf{a}) \equiv \mathbf{B}\{\mathbf{x}:=\mathbf{a}\}$
$\mathbf{B}^*(\mathbf{x}) \equiv \forall \mathbf{y}(\mathbf{y}<\mathbf{x} \rightarrow \mathbf{B}(\mathbf{y}))$
$\mathbf{R} \equiv \forall \mathbf{x}(\mathbf{B}^*(\mathbf{x}) \rightarrow \mathbf{B}(\mathbf{x}))$

We shall also assume that all auxiliary variables **l**, **n**, **y**, **w**, **z**, etc. used in this section are all different and that they are also different from **x** and do not occur in the formula **A**.

For an ordinal n we call the formula $\mathbf{P} \rightarrow \mathbf{A}^*(n)$ the *transfinite induction* up to n. We want to show that transfinite induction holds for any ordinal less than ϵ_0 and trivially also for any literal. Preparatory to the proof of (T9) we prove the lemmas (T1) through (T8).

$$\vdash \mathbf{P} \,\&\, \mathit{Ord}(\mathbf{x}) \,\&\, \mathbf{A}^*(\mathbf{x}) \rightarrow \mathbf{A}^*(\mathbf{x}+1) \tag{T1}$$

Proof: Assume the antecedent. We immediately obtain $\mathbf{A}(\mathbf{x})$. If $\mathbf{y}<\mathbf{x}+1$ then by (T7.10) $\mathbf{y}<\mathbf{x} \vee \mathbf{y} = \mathbf{x}$. Thus in any case $\mathbf{A}(\mathbf{y})$. QED.

$$\vdash \mathbf{B}(\mathbf{x}) \rightarrow \mathbf{B}(\mathbf{x}+1) \tag{T2}$$

Proof: Assume $\mathbf{B}(\mathbf{x})$, $\mathit{Ord}(\mathbf{x})$, $\mathit{Ord}(\mathbf{y})$, and $\mathbf{A}^*(\mathbf{y})$. First of all note that if $N(\mathbf{l})$ and $\mathbf{A}^*(\mathbf{y}+[\mathbf{x}, 0]\times\mathbf{l})$ then $\mathit{Ord}(\mathbf{y}+[\mathbf{x}, 0]\times\mathbf{l})$ and from $\mathbf{B}(\mathbf{x})$ we obtain $\mathbf{A}^*((\mathbf{y}+[\mathbf{x}, 0]\times\mathbf{l})+[\mathbf{x}, 0])$, i.e. $\mathbf{A}^*(\mathbf{y}+[\mathbf{x}, 0]\times(\mathbf{l}+1))$. Since also $\mathbf{A}^*(\mathbf{y}+[\mathbf{x}, 0]\times 0)$ the induction on natural numbers (T7.37) applies to yield

$$N(\mathbf{l}) \rightarrow \mathbf{A}^*(\mathbf{y}+[\mathbf{x}, 0]\times\mathbf{l}) \tag{1}$$

In order to prove $\mathbf{A}^*(\mathbf{y}+[\mathbf{x}+1, 0])$ assume $\mathbf{z}<\mathbf{y}+[\mathbf{x}+1, 0]$. If $\mathbf{z}<\mathbf{y}$ then $\mathbf{A}(\mathbf{z})$ follows from $\mathbf{A}^*(\mathbf{y})$, assume therefore that $\mathbf{y}\leq\mathbf{z}$. Then by (T7.15)

$$\mathbf{z} = \mathbf{y}+(\mathbf{z}\text{-}\mathbf{y}) < \mathbf{y}+[\mathbf{x}+1, 0]$$

From the monotonicity of addition (T7.7) we obtain

$$(\mathbf{z}\text{-}\mathbf{y}) < [\mathbf{x}+1, 0] = [\mathbf{x}, 0]\times\omega$$

Let $\mathbf{a} \equiv (\mathbf{z}\text{-}\mathbf{y})/[\mathbf{x}, 0]$ then by (T7.28) we have

$$[\mathbf{x}, 0]\times\mathbf{a} \leq (\mathbf{z}\text{-}\mathbf{y}) < [\mathbf{x}, 0]\times(\mathbf{a}+1)$$

Since

$$[\mathbf{x}, 0]\times\mathbf{a} \leq (\mathbf{z}\text{-}\mathbf{y}) < [\mathbf{x}, 0]\times\omega$$

we use the monotonicity of multiplication (T7.21) to derive $\mathbf{a}<\omega$. Thus also $N(\mathbf{a}+1)$ and from (1) we have $\mathbf{A}^*(\mathbf{y}+[\mathbf{x}, 0]\times(\mathbf{a}+1))$. Now, from

$$\mathbf{z} = \mathbf{y}+(\mathbf{z}\text{-}\mathbf{y}) < \mathbf{y}+[\mathbf{x}, 0]\times(\mathbf{a}+1)$$

we obtain $\mathbf{A}(\mathbf{z})$. QED.

$$\vdash \mathbf{P} \rightarrow \mathbf{R} \tag{T3}$$

Proof: Assume **P**. In order to prove **R** we also assume $\mathbf{B}^*(\mathbf{x})$ for certain **x**. In order to prove $\mathbf{B}(\mathbf{x})$ we assume $\mathit{Ord}(\mathbf{x})$, $\mathit{Ord}(\mathbf{y})$, and $\mathbf{A}^*(\mathbf{y})$ for certain **y**. Finally, in order to prove $\mathbf{A}^*(\mathbf{y}+[\mathbf{x}, 0])$, we assume $\mathbf{z}<\mathbf{y}+[\mathbf{x}, 0]$ for certain **z**. We have to prove $A(\mathbf{z})$. Note first that by (T1) we have $\mathbf{A}^*(\mathbf{y}+1)$. There are two cases.

a) $\mathbf{x} = 0$. Then $[\mathbf{x}, 0] = 1$. Thus $\mathbf{z}<\mathbf{y}+1$ and we have $\mathbf{A}(\mathbf{z})$ from $\mathbf{A}^*(\mathbf{y}+1)$.

b) $\mathbf{x} \neq 0$. If also $\mathbf{z}\leq\mathbf{y}$ then $\mathbf{A}(\mathbf{z})$ because of $\mathbf{A}^*(\mathbf{y}+1)$. Therefore assume that $\mathbf{y}<\mathbf{z}$. Let $\mathbf{b} \equiv \mathbf{z}\text{-}\mathbf{y}$. We have $0<\mathbf{b}$ and $\mathbf{y}+\mathbf{b} = \mathbf{z}<\mathbf{y}+[\mathbf{x}, 0]$, i.e by (T7.7) $\mathbf{b}<[\mathbf{x}, 0]$. Therefore $\mathbf{b}.\mathbf{h} < \mathbf{x}$ and from $\mathbf{B}^*(\mathbf{x})$ we obtain $\mathbf{B}(\mathbf{b}.\mathbf{h})$. From (T2) we have $\mathbf{B}(\mathbf{b}.\mathbf{h}+1)$ and since $\mathit{Ord}(\mathbf{b}.\mathbf{h}+1)$ and $\mathit{Ord}(\mathbf{y})$ also $\mathbf{A}^*(\mathbf{y}+[\mathbf{b}.\mathbf{h}+1, 0])$. Applying the monotonicity of addition (T7.7) again

$$\mathbf{z} = \mathbf{y}+\mathbf{b} < \mathbf{y}+[\mathbf{b}.\mathbf{h}+1, 0]$$

we obtain $A(\mathbf{z})$. QED.

Let us define functions *om* and *maj* as follows.

$$\mathit{om} \equiv \lambda \mathit{om}, x.\ \textbf{if } x \textbf{ then } 0 \textbf{ else } [\mathit{om} \bullet x.\mathbf{t}, 0] \tag{D1}$$

$maj \equiv \lambda maj,\ x.$ **if** x **then** 1 **else** $maj \bullet x.\mathbf{h}{+}1$ (D2)

Both functions are obviously recursive. We shall abbreviate $om \bullet x$ by ω_x. We have the following properties.

$\vdash\ \omega_0 = 0$ (T4)
$\vdash\ N(i) \rightarrow \omega_{i+1} = [\,w_i\ 0\,]$ (T5)
$\vdash\ N(i) \rightarrow Ord(\omega_i)$ (T6)
$\vdash\ Ord(x) \rightarrow N(maj \bullet x)\ \&\ x{<}\omega_{maj \bullet x}$ (T7)

Proof: (T4) and (T5) follow directly from definitions. (T6) is proven by the induction over natural numbers (T7.37) on the variable x with $\mathbf{A} \equiv Ord(\omega_i)$. (T7) is proven by the ordinary induction over pairs (3.Ind) on the variable x with $\mathbf{A} \equiv (T7)$.

Note that by the representation theorem (T5.9) a pair p is an ordinal (natural number) iff $Ord(\overline{p})$ reduces to 0, i.e. $\vdash\ Ord(\overline{p})$; ($\overline{p} < \omega$ reduces to 0, i.e. $\vdash\ N(\overline{p})$).

Theorem: For any formula **A** and any numeral $\overline{n}$ we have

$\vdash\ \mathbf{P} \rightarrow \mathbf{A}^{\bullet}(\omega_{\overline{n}})$ (T8)

The proof is by the induction on the construction of numerals. If $\overline{n} \equiv 0$ then $\vdash\ w_{\overline{n}} = 0$ and

$\vdash\ \mathbf{P} \rightarrow \mathbf{A}^{\bullet}(\omega_{\overline{n}})$ (2)

holds trivially. Assume therefore (2) for any formula **A** and for a numeral $\overline{n}$. We have $\vdash\ N(\overline{n})$. Substituting **B** for **A** in (2) yields

$\vdash\ \mathbf{R} \rightarrow \mathbf{B}^{\bullet}(\omega_{\overline{n}})$.

Since

$\vdash\ \mathbf{R}\ \&\ \mathbf{B}^{\bullet}(\omega_{\overline{n}}) \rightarrow \mathbf{B}(\omega_{\overline{n}})$

we also have

$\vdash\ \mathbf{R} \rightarrow \mathbf{B}(\omega_{\overline{n}})$

and by (T3) also

$\vdash\ \mathbf{P} \rightarrow \mathbf{B}(\omega_{\overline{n}})$. (3)

Now from (T6) we have $\vdash\ Ord(\omega_{\overline{n}})$ and since also $\vdash\ Ord(0)\ \&\ \mathbf{A}^{\bullet}(0)$ we have

$\vdash\ \mathbf{B}(\omega_{\overline{n}}) \rightarrow \mathbf{A}^{\bullet}(0{+}[\,\omega_{\overline{p}}\ 0\,])$.

From (T5) we have

$\vdash\ \mathbf{B}(\omega_{\overline{n}}) \rightarrow \mathbf{A}^{\bullet}(\omega_{\overline{n}+1})$.

Combining this with (3) yields

$\vdash\ \mathbf{P} \rightarrow \mathbf{A}^{\bullet}(\omega_{\overline{n}+1})$.

Noting that $\vdash\ \overline{n}{+}1 = [0,\ \overline{n}]$ terminates the proof.

Theorem on Transfinite Induction. For any formula **A** and any literal $\overline{p}$ the following holds.

$\vdash\ \mathbf{P} \rightarrow \mathbf{A}^{\bullet}(\overline{p})$ (T9)

Proof: If p is not an ordinal then $\vdash\ \neg\ x{<}\overline{p}$ and (T9) holds trivially. If p is an ordinal then by (T7)

$\vdash\ N(maj \bullet \overline{p})\ \&\ \overline{p}{<}\omega_{maj \bullet \overline{p}}$

Because TP is consistent $maj \bullet \overline{p}$ reduces to a numeral $\overline{n}$ such that $\vdash\ \overline{p}{<}\omega_{\overline{n}}$ (T9) now follows from $\vdash\ \mathbf{A}^{\bullet}(w_{\overline{n}}) \rightarrow \mathbf{A}^{\bullet}(\overline{p})$ and (T8). QED.

It is easy to see that

$\vdash\ \forall \mathbf{x}(\mathbf{P} \rightarrow \mathbf{A}^{\bullet}(\mathbf{x})) \leftrightarrow \forall \mathbf{x}(\mathbf{P} \rightarrow \mathbf{A}(\mathbf{x}))$

The formula $\mathbf{P} \rightarrow \mathbf{A}(\mathbf{x})$ is transfinite induction up to ϵ_0, it is also the formula for the well-founded induction (6.Wford). As mentioned in section (6) it is true in the standard interpretation yet unprovable in TP. This was established by Gentzen [3] for PA. The unprovability in TP is a consequence of the equivalence of TP and PA. In the Part III we shall demonstrate the unprovability directly.

The following theorem relates the transfinite and well-founded inductions.

Theorem (T10): For any literal $\bar{p}$ the relation $<_{\bar{p}}$ is provably well-founded.

Proof: Take a literal $\bar{p}$ and any formula **A**. We want to prove $\mathbf{P}_{\bar{p}} \rightarrow \mathbf{A}(\mathbf{x})$ where $\mathbf{P}_{\bar{p}}$ is as follows.

$$\mathbf{P}_{\bar{p}} \equiv \forall \mathbf{x}\{\forall \mathbf{y}\,(\mathbf{y} <_{\bar{p}} \mathbf{x} \rightarrow \mathbf{A}(\mathbf{y})) \rightarrow \mathbf{A}(\mathbf{x})\}$$

Since we have

$$\vdash \mathbf{P}_{\bar{p}} \leftrightarrow \forall \mathbf{x}\{(\mathbf{x} < \bar{p} \rightarrow \mathbf{A}^*(\mathbf{x})) \rightarrow \mathbf{A}(x)\}$$
$$\vdash \mathbf{P}_{\bar{p}} \leftrightarrow \forall \mathbf{x}(\neg\, \mathbf{x} < \bar{p} \rightarrow \mathbf{A}(\mathbf{x})) \,\&\, \mathbf{P}$$

we can use (T9) to obtain

$$\vdash \mathbf{P}_{\bar{p}} \rightarrow \forall \mathbf{x}(\neg\, \mathbf{x} < \bar{p} \rightarrow \mathbf{A}(\mathbf{x})) \,\&\, \forall \mathbf{x}(\mathbf{x} < \bar{p} \rightarrow \mathbf{A}(\mathbf{x}))$$

thus in any case $\vdash \mathbf{P}_{\bar{p}} \rightarrow \mathbf{A}(\mathbf{x})$. QED.

The theorem on recursive functions (T5.22) allows to show functions recursive provided one proves that the argument of the recursion is driven down in a well-founded relation. We would like to characterize a class of provably recursive functions syntactically without relying on proofs in TP. The class of *functions defined by transfinite induction for ordinals less than* ϵ_0, or shortly T-functions is such a class.

We need an auxiliary definition. For any RF-term **a** and literal $\bar{m}$ let us denote by **a'** the RF-term obtained by the following term identities.

$$n' \equiv n$$
$$0' \equiv 0$$
$$[\mathbf{b}, \mathbf{c}]' \equiv [\mathbf{b}', \mathbf{c}']$$
$$(\mathbf{b}.\mathbf{h})' \equiv \mathbf{b}'.\mathbf{h}$$
$$(\mathbf{b}.\mathbf{t})' \equiv \mathbf{b}'.\mathbf{t}$$
$$(\textbf{if } \mathbf{b} \textbf{ then } \mathbf{c} \textbf{ else } \mathbf{d})' \equiv \textbf{if } \mathbf{a}' \textbf{ then } \mathbf{b}' \textbf{ else } \mathbf{c}'$$
$$(\mathbf{b} \bullet \mathbf{c})' \equiv \mathbf{b} \bullet \mathbf{c}' \quad \text{where } \mathbf{b} \text{ is a literal}$$
$$(n.\mathbf{h} \bullet \mathbf{c})' \equiv \textbf{if } \bar{m} \bullet \mathbf{c}' \dot{<} \bar{m} \bullet n.\mathbf{t} \textbf{ then } n.\mathbf{h} \bullet \mathbf{c}' \textbf{ else } 0$$

T-functions are defined by the following inductive definition.

1) Functions *len* and $\dot{<}$ are T-functions.

2) Let **a** be an RF-term where in all subterms $\mathbf{b} \bullet \mathbf{c}$ with **b** a literal, the term **b** is a T-function. If the term **a** does not contain subterms of the form $n.\mathbf{h} \bullet \mathbf{c}$ then the function $\lambda\mathbf{a}$ is an (explicitly defined) T-function. If, on the other hand, **a** contains subterms of the form $n.\mathbf{h} \bullet \mathbf{c}$ then for a T-function $\bar{m}$ and a literal $\bar{p}$ the function

   $\lambda\, \textbf{if } n.\mathbf{t} \dot{<} \bar{p} \textbf{ then } \mathbf{a}' \textbf{ else } 0$

   is a T-function.

3) T-functions are obtained only on the account of the rules 1) and 2).

We have the following theorem.

Theorem (T11): All T-functions are provably recursive.

The proof is by the induction on the construction of T-functions. Functions *len* and $\dot{<}$ are provably recursive. For the term **a** from the point 2) if $\lambda\mathbf{a}$ is an explicit function then take $\ll \equiv \dot{<}$, otherwise let us define

$$x \ll y \leftrightarrow \bar{m} \bullet x <_{\bar{p}} \bar{m} \bullet y$$

In the former case $\dot{<}$ is provably well-founded by (T3.10) in the latter case the relation $\ll$ is

provably well-founded by (T3.17) and (T10). In any case $\vdash Down_{\mathbf{b}}^{\prec}$ where **b** is either **a** or λ **if** $n.\mathbf{t} \dot{<} \overline{p}$ **then a' else 0** and thus the theorem (T5.22) applies. QED.

The function $\overline{m}$ supplies an (ordinal) measure in which arguments of recursive functions descend. The literal $\overline{p}$ guarantees that we have a provably well-founded relation in which the measure descends.

The question whether there are provably recursive functions $\overline{p}$, i.e. such functions that $\vdash Def(\overline{p}, x)$, which are not T-functions is answered negatively by pointing out that *cat* is not a T-function. In the part III we plan to demonstrate that we do not gain any extra functions, just different programs, because for any provably recursive function $\overline{p}$ there is a T-function $\overline{q}$ such that $\vdash \overline{p} \bullet x = \overline{q} \bullet x$.

9. Conclusion.

We have presented what, we think, is a simple definition of partial recursive functionals over pairs. The elegance of pairs permits a very simple form of Gödel numbering and consequently a simple formulation of the predicate *Comp*. The simplicity of *Comp* and *val* allows a straightforward formulation of theorems about computability as well as a simple interpretation of partial recursive functionals into TP. Pairs also permit a natural notation for ordinals less than ϵ_0.

The author would like to thank his colleagues Karl Abrahamson and Akira Kanda from the Department of Computer Science and especially Andrew Adler from the Department of Mathematics for long and fruitful series of discussions and suggestions about the domain of pairs.

[1] Boyer R., Moore J., A Computational Logic; Academic press, New York 1979.

[2] Feferman S., Inductively Presented Systems and the Formalization of Meta-Mathematics, in Logic Colloquium '80; North Holland, 1982.

[3] Gentzen G., Beweisbarkeit und Unbeweisbarkeit von Anfangsfaellen der transfiniten Induktion in der reinen Zahlentheorie; Mathematische Annalen, vol.119, p140-161, 1943.

[4] Gödel K., Ueber eine bisher noch nicht benuetzte Erweiterung des finiten Standpunktes; Dialectica, vol. 12, p280-287, 1958.

[5] Kleene S., Introduction to Meta-Mathematics; North Holland, 1971.

[6] Kowalski R., Logic for Problem Solving; North Holland, Amsterdam 1979.

[7] Levy A., Basic Set Theory; Springer-Verlag, Berlin 1979.

[8] Martin-Lof P. An Intuitionistic Theory of Types; in Logic Colloquium '73 (Rose ed.), North Holland, Amsterdam, 1973.

[9] McCarthy J., A basis for a mathematical theory of computation; in Computer Programming and Formal Systems (P. Braffort ed.), North Holland, Amsterdam 1963.

[10] McCarthy J. et al., LISP 1.5 Programmer's Manual; MIT Press, Cambridge Mass., 1965.

[11] Milner R., A Theory of Type Polymorphism in Programming; Journal of Computer and System Sciences, vol. 17, p348-375, 1978.

[12] Peter R., Recursive Functions in Computer Theory; Ellis Horwood, Chichester 1981.

[13] Sato M., Theory of Symbolic Expressions, II; Technical Report 84-04, Department of Information Science, University of Tokyo, Tokyo 1984.

[14] Schuette K., Proof Theory; Springer Verlag, Berlin 1977.

[15] Shoenfield J., Mathematical Logic, Addison-Wesley, Reading Mass., 1967.

[16] Takeuti G., Proof Theory; North Holland, Amsterdam, 1975.

[17] Voda P., A View of Programming Languages as Symbiosis of Meaning and Computations; appears in New Generation Computing, Tokyo, February 1985.