SYNTHESE LIBRARY

MONOGRAPHS ON EPISTEMOLOGY,

LOGIC, METHODOLOGY, PHILOSOPHY OF SCIENCE,

SOCIOLOGY OF SCIENCE AND OF KNOWLEDGE,

AND ON THE MATHEMATICAL METHODS OF

SOCIAL AND BEHAVIORAL SCIENCES

Editors:

DONALD DAVIDSON, *Rockefeller University and Princeton University*

JAAKKO HINTIKKA, *Academy of Finland and Stanford University*

GABRIËL NUCHELMANS, *University of Leyden*

WESLEY C. SALMON, *Indiana University*

SÖREN STENLUND

COMBINATORS, λ-TERMS AND PROOF THEORY

D. REIDEL PUBLISHING COMPANY / DORDRECHT-HOLLAND

CHAPTER 4

COMPUTABLE FUNCTIONALS OF FINITE TYPE

1. INTRODUCTION

1.1. The subject matter of the present chapter originated with Gödel 1958. The second incompleteness theorem showed that there is no hope of obtaining a consistency proof of extensions of arithmetic by strictly finitary methods. The problem still remained of obtaining a proof theoretical reduction and a consistency proof of extensions of arithmetic by methods which, although not finitary, could be considered more justified from some epistemological point of view than the methods formalized in these extensions of arithmetic.

Gentzen showed that the consistency problem of arithmetic was reducible by means of finitary methods to the problem of the well-foundedness of a primitive recursive ordering relation of order type ε_0. Another suggestion was given by Gödel 1958, who showed how to obtain a proof theoretical reduction of arithmetic by methods obtained by adjoining to the finitary methods of primitive recursive arithmetic some *abstract notions*, namely a certain notion of *constructive function*.

1.2. The constructive functions introduced by Gödel are called the *computable functionals of finite type*, where the computable functionals of lowest type are the natural numbers and the computable functionals of type $\tau \to \sigma$ are rules or operations, which when applied to computable functionals of type τ can be constructively proved always to yield computable functionals of type σ as values. This notion is to be understood as a primitive, immediately understood notion, and the only notions and methods to be used in the consistency proof beyond those of primitive recursive arithmetic, are the abstract notion of computable func-

tional of finite type and certain elementary construction principles for such functionals expressed by defining equations. For the consistency proof to be reductive, the use of the notion of a computable functional should not involve some abstract notion of proof. The strength of the reasoning is to come from *principles of definition* or *of construction* rather than from rules of inference.

1.3. Gödel described an equation calculus **T** intended to codify those ideas and showed how to interpret **HA** (Heyting arithmetic) in **T**. The formulas of **T** are equations, $a=b$, between terms a and b of the same finite type. If a and b have the type of the numbers, then $a=b$ means that a and b denote the same number. If a and b are of higher type over the numbers, then $a=b$ means *intensional equality*.[1] That is, the rules of correspondence codified in the terms a and b are one and the same, or the terms a and b have the same *meaning* relative to the meaning given to the terms by the defining equations.

It is through the interpretation of equality between terms of higher types as intensional equality between rules that a statement in **T** about computable functionals of higher types does not involve some abstract notion of proof.

So, from the philosophical point of view of Gödel 1958, we must admit such abstract objects as computable functionals of higher types as certain rules of correspondence, being the objects in the intended model. These objects are to be considered as being on the same ontological level as such abstract objects as meaningful assertions, proofs, etc. For the consistency proof intended by Gödel, it is not necessary to have complete knowledge of the intended model; it is only important to accept the *notion* of a computable functional of finite type, because whatever the totality of these functionals are they must satisfy the elementary axioms of **T**.

That abstract objects of this kind are justified from a constructive point of view was first pointed out explicitly by Gödel 1958, although they had been used implicitly in informal intuitionistic reasoning.

1.4. In this chapter we shall codify the computable functionals of an

[1] We follow here the terminology in Gödel 1958. The qualification 'intensional' means (for us) only that we do not expect extensionality to hold.

extension of **T** using λ-terms which are assigned types, and certain recursion combinators. The usefulness of the λ-terms for this purpose is due to the fact that λ-terms codifies rules. A λ-term does not just name a rule; it also expresses the corresponding computation process. It codifies the definition or construction of the rule. For example, if we understand the λ-operator, then by looking at the term $\lambda x.x$ we see how the computation process of the corresponding rule works.

Rules tie together to form other rules by means of application and functional abstraction in the λ-calculus, and the choice of those two notions for the analysis of the general notion of a rule does not seem to be arbitrary. Rules are thought of as operations that can be applied to certain objects as arguments, so any theory of rules should make explicit mention of application.

The most general way of obtaining rules is by explicit definition: A linguistic expression $a(x)$ containing an indeterminate x determines a rule which when applied to an argument b has $a(b)$ as a value. In the λ-calculus this is the role of λ-abstraction or functional abstraction.

It is true that functional abstraction could be replaced by and defined in terms of some specific rules, i.e. combinators. But it seems more natural here to take it as a primitive notion, since, as already pointed out, rules are usually expressed by linguistic expressions containing variables in informal reasoning.

Application and λ-abstraction can be said to play the same role for rules as the logical operations play for propositions. They are appropriate for expressing the logically significant properties of rules. The Dialectica interpretation of Gödel 1958, can be said to show how to reduce the logical operations of **HA** to the "logic of rules" in **T**.

The λ-terms are also useful for the reason that we obtain a natural way of representing the computation processes of the computable functionals, namely as reduction rules. This was first realised by Tait 1965 and 1967*a* who calls them conversion rules. The particular reduction procedures in the λ-calculus are interesting also for the reason that they are isomorphic to reduction procedures in systems of natural deduction, as will be seen in the next chapter.

1.5. In the next section we shall describe an extension **T′** of **T**. **T′** is a simplification of the system $\mathbf{\Gamma}_4$ introduced by Howard 1963. Using the

Dialectica interpretation, Howard showed how to interpret a system of intuitionistic analysis with bar-recursion of type 0 in $\mathbf{\Gamma}_4$. The consistency of intuitionistic analysis is thus finitarily reducible to that of $\mathbf{\Gamma}_4$. Also, the consistency of the classical system $\mathbf{Z}_1$ of elementary analysis of well-founded relations of Howard and Kreisel 1966 is reducible to that of $\mathbf{\Gamma}_4$.

$\mathbf{T}'$ is also closely related to the system $\mathbf{T}_1$ introduced by Tait 1967*a*. But we have followed Howard 1963 by working with two basic types, one for natural numbers and one for the Brouwer ordinals. $\mathbf{T}'$ will be presented as an equation calculus for deriving equations between terms of the same finite type. In section 7 we present the reduction rules and use them to give a different characterization of intensional equality. In section 8 we establish the consistency of the reduction rules of $\mathbf{T}'$ using the computability method introduced by Tait 1967*a*. We will in fact prove something more, namely a strong normalization theorem (in the terminology of Prawitz 1970) to the effect that each reduction sequence is finite. While Tait 1967*a* proves that each closed term is normalizable, we will prove the strong normalizability of all terms. This is needed for the results in section 9, where we introduce the notion of a model for $\mathbf{T}'$ and prove the completeness of the axioms and rules for intensional equality.

2. FINITE TYPES AND TERMS OF FINITE TYPES

2.1. The *finite types* are defined inductively,

(i) 0 is a finite type,

(ii) 1 is a finite type,

(iii) if σ and τ are finite types, then $(\sigma\to\tau)$ is a finite type.

The types of clause (i) and (ii) are the *atomic types* and the types of clause (iii) are the *higher types*. We will write $\sigma_1\to\ldots\to\sigma_n\to\tau$ for $(\sigma_1\to\ldots\to(\sigma_n\to\tau)\ldots)$. Each finite type is then uniquely of the form $\sigma_1\to\ldots\to\sigma_n\to\tau$, where τ is 0 or 1 and $n\geqslant 0$. Arbitrary types will be denoted by $\sigma, \tau, \varrho, \ldots$

The intended interpretation of the finite types is this: The objects of type 0 are the natural numbers and the objects of type 1 are wellfounded trees, i.e. Brouwer ordinals. The objects of type 0 and 1 are said to be computable functionals. The objects of type $\sigma\to\tau$ are computable functionals, which are defined to be rules or functions, which, when applied to computable functionals of type σ always yield computable functionals of type τ as values. The universal quantifier 'always' is here to be understood in the constructive sense.

2.2. *Terms of finite types.* We first introduce some *constants*, each with a fixed type.

2.2.1. 0_0 is a constant of type 0, intended to denote the number 0.

2.2.2. 0_1 is a constant of type 1, intended to denote the ordinal 0.

2.2.3. S_0 is a constant of type $0\to 0$, intended to denote the successor operation on natural numbers.

2.2.4. S_1 is a constant of type $(0\to 1)\to 1$, intended to denote the supremum operation on sequences of ordinals.

2.2.5. For each type τ, there is a constant R_0^τ of type

$$\tau\to(0\to\tau\to\tau)\to 0\to\tau,$$

intended to denote the primitive recursion functional.

2.2.6. For each type τ, there is a constant R_1^τ of type

$$\tau\to((0\to 1)\to(0\to\tau)\to\tau)\to 1\to\tau,$$

intended to denote the ordinal recursion functional.

The meaning of the recursion functionals will be further explained through their defining equations in section 3.

The terms of finite types are then defined by the following inductive definition.

(i) Each constant of type τ is a term of type τ.

(ii) For each type τ, there is a denumerably infinite list of variables of type τ and each variable of type τ is a term of type τ. The variables will be denoted $x^\tau, y^\tau, z^\tau, \ldots$

(iii) If a is a term of type τ and x is a variable of type σ, then $\lambda x.a$ is a term of type $\sigma \to \tau$. (Functional abstraction.)

(iv) If a is a term of type $\sigma \to \tau$ and b is a term of type σ, then (ab) is a term of type τ. (Application.)

Terms will be denoted by $a^\tau, b^\tau, c^\tau, \ldots$. Sometimes we will denote a term of type 0 or $0 \to 1$ by t. Free and bound variables are defined as usual and we will identify terms that differ only in their bound variables. A substitution $a(b)$ of b for x in $a(x)$ is also defined as in chaper 1 with the exception that now, x and b must have the same type. Assuming that types are always properly chosen, we shall generally omit type superscripts. We write as usual $a_1 \ldots a_n$ for $(\ldots (a_1 a_2) \ldots a_n)$ and $\lambda x_1 \ldots x_n . a$ for $\lambda x_1 . \lambda x_2 . \ldots \lambda x_n . a$.

The *numerals* are the terms 0_0, $S_0 0_0$, $S_0(S_0 0_0)$, ... We denote them by $\bar{0}$, $\bar{1}$, $\bar{2}$, ..., respectively.

3. THE EQUATION CALCULUS

We shall present an equation calculus **T′** by giving axioms and inference rules for deriving equations of the form

$$a =_t b,$$

where a and b are terms of the same finite type, and for deriving equations of the form

$$a = b,$$

where a and b are terms, both of type 0 or both of type 1.

3.1. *Axioms and rules for intensional equality.*

3.1.1 *Equality axioms.*

(i 1) $\quad a =_t a$

(i 2) $\quad \dfrac{a =_t b}{b =_t a}$

(i 3) $\quad \dfrac{a =_t b, b =_t c}{a =_t c}$

(i 4) $\quad \dfrac{a =_t b, c =_t d}{ac =_t bd}$

(i 5) $\quad \dfrac{a =_t b}{\lambda x . a =_t \lambda x . b}$

3.1.2. *Axioms for constants and λ-abstraction.*

(i 6) $\quad R_0 ab\bar{0} =_t a$
$\quad R_0 ab(S_0 t) =_t bt(R_0 abt)$

(i 7) $\quad R_1 ab\,0_1 =_t a$
$\quad R_1 ab(S_1 t) =_t bt(\lambda x . R_1 ab(tx))$

(i 8) $\quad (\lambda x . a(x))b =_t a(b)$

(i 9) $\quad \lambda x . ax =_t a$

(In the axioms (i 7) and (i 9), x is not free in a, b and t)

3.2. *Axioms and rules for equality of type 0 or 1.* The terms are assumed to have types such that the left and right hand sides in the axioms and in the conclusions of the rules are of type 0 or 1.

(e 1) $\quad a = a$

(e 2) $\quad \dfrac{a = b}{b = a}$

(e 3) $\quad \dfrac{a = b, b = c}{a = c}$

(e 4) $\quad \dfrac{c = d}{ac = ad}$ (c and d of type 0 or 1)

(e 5) $\quad \dfrac{a = b}{(\lambda x . a)c = (\lambda x . b)c}$ (c of type 0 or 1)

$$(e\,6)\qquad \frac{a =_i b}{a = b}$$

$$(\mathbf{IR})\qquad \frac{f\bar{0} = a,\ f(S_0 t) = bt(ft)}{ft = R_0 abt}$$

3.4. Those are all the axioms and rules of inference of **T′**. It is not necessary to postulate a transfinite induction rule of the form

$$(TI)\qquad \frac{f0_1 = a,\ f(S_1 t) = bt(\lambda x\,.\,f(tx))}{fc = R_1 abc}$$

for, as shown by Howard 1963 and 1968*a*, it can be derived by the method of Kreisel 1959.

By using two notations $=_i$ and $=$ for equality, we do not want to suggest that there are different kinds of identity with respect to a certain interpretation. On the intended interpretation $=_i$ denotes identity between objects of higher types, and $=$ denotes identity between objects of type 0 or 1, and in the latter case $=_i$ is simply a subrelation of $=$. The index on the relation $=_i$ functions only as a deduction technical device. It imposes a restriction on the inference rule (*i* 5), which would not be valid for the intended interpretation if all indices *i* were omitted, as will be argued below.

With respect to the classical intended interpretation (where the objects of higher types are functions in extension) one can, of course, think of $=_i$ as denoting a subrelation of identity at all types. The axioms for $=_i$ are then all properties needed of identity to prove the Dialectica interpretation of the axioms of intuitionistic analysis in **T′**. But from the point of view of sections 1.1, 1.2 and 1.3 in the introduction, the classical model is of course uninteresting.

3.5. *Truth-functions.* This equation calculus is (with trivial modifications) an extension of primitive recursive arithmetic in the sense of Goodstein 1957. Therefore we can define closed terms $+$, $\dot{-}$, $\cdot$, all of type $0\to0\to0$ such that (where $a+b$ means $+ab$ etc.)

$$a+\bar{0} = a$$
$$a+(S_0 b) = S_0(a+b)$$

$$\bar{0}\dot{-}\bar{1} = \bar{0} \qquad a\dot{-}\bar{0} = a$$
$$(S_0 a)\dot{-}1 = a \qquad a\dot{-}(S_0 b) = (a\dot{-}b)\dot{-}\bar{1}$$
$$a\cdot\bar{0} = \bar{0}$$
$$a\cdot(S_0 b) = (a\cdot b)+a.$$

Using the induction rule (**IR**), we can prove the usual properties of those functions as in recursive arithmetic. For x and y of type 0, we can define the constant E of type $0\to0\to0$ by

$$E \equiv \lambda xy\,.\,(x\dot{-}y)+(y\dot{-}x),$$

and prove that for all terms a and b of type 0,

$$Eab = \bar{0} \quad \text{iff} \quad a = b.$$

It was Gödel's intention that formulas of **T** should be built up by means of propositional truth-functions. Having the equality function E, we can define the truth-functions as in recursive arithmetic.

$$\neg\,(a = b) \quad \text{means} \quad (\bar{1}\dot{-}Eab) = \bar{0}$$

and

$$(a = b)\to(c = d) \quad \text{means} \quad (\bar{1}\dot{-}Eab)\cdot(Ecd) = \bar{0}.$$

In this way, all tautologies of the classical propositional calculus becomes provable in **T′**. Furthermore, using the induction rule we can prove the following formulas

$$\neg\,(S_0 t = \bar{0})$$
$$(S_0 t = S_0 t')\to t = t',$$

i.e. the Peano third and fourth axioms.

Tait 1967 seems, however, to suggest that the formulas of **T** are to be built up by means of propositional truth-functions from equations between terms of the same arbitrary finite type. And for the interpretation of Heyting arithmetic of all finite types $\mathbf{HA}^\omega$ or $\mathbf{IDB}^\omega$ in the sense of Troelstra 1970, one has to form formulas between terms of higher types.

One way to do this within an equation calculus like T′, is by introducing equality functionals E^τ of type $\tau\to\tau\to 0$ with the axioms

$$\frac{a=_t b}{Eab=\bar{0}} \qquad \text{If } a \text{ and } b \text{ are distinct closed normal}^1 \text{ terms, infer } Eab=\bar{1}.$$

But the functionals E^τ seem to be of an unnatural kind compared to the ones explained in section 2.1. In a certain sense, the functionals E^τ operate on terms. So we prefer not to use them.

The best way to proceed if one wants to have formulas built up from equations between terms of higher types, seems to be to go outside the equation calculus and take the propositional operations $\neg$, $\to$, etc. as primitives.

4. THE ROLE OF THE INDUCTION RULE

4.1. In some presentations of (extensions of) Gödel's **T**, eg in Kreisel 1970, the induction rule is formulated such that it applies to terms of arbitrary finite type, i.e. the terms in the conclusion $ft=R_0abt$ of **(IR)** may have the same arbitrary finite type. In other formulations of (extensions of) **T**, an extensional notion of equality between terms of higher types is used (cf. Spector 1962). For the purpose of obtaining the consistency proof of **HA** intended by Gödel 1958, this seems not to be quite correct. We think that it is not in agreement with the philosophical position of section 1.2 and 1.3.

First of all, the induction rule for terms of higher types is not needed for the proof in **T** of the Dialectica interpretation of the axioms of **HA**. The induction rule enters only in the proof in T of the Dialectica interpretation of the induction axioms of **HA**, and for this it suffices to have **(IR)** of type 0. All axioms of **HA** are transformed into equations between terms of type 0 by means of the Dialectica interpretation. And intensional equality is used through our rule (*e* 6) to reduce the logical operations of **HA** to the computable functionals of **T**.

Secondly, (and more important) an equation between terms of higher types means (intensional) equality between rules of correspondence

[1] For a definition of the normal terms see section 7.

and it would be unjustified to use the principles of proof **(IR)** or extensionality to infer such a statement.

Consider for example the term

$$R_0ab$$

where a is a closed term of higher type τ and $b\equiv\lambda x^0.\lambda y^\tau.a$. Then, by the induction rule and (*i* 5) without restriction, we could prove

$$\lambda x.a=\lambda x.R_0abx,$$

where x is a variable of type 0. But the terms $\lambda x.a$ and $\lambda x.R_0abx$ are not intensionally equal as we understand this notion; they express different definitions. The equation is justified by means of proof by mathematical induction and not by the meaning given to R_0 by its defining equation only.

Of course one may be interested in (extensions of) **T** for other reasons than obtaining the consistency proof intended by Gödel, and then there may be some interest in seeing what happens when our restrictions on **(IR)** is dropped. Tait's 1967*a* and Howard's 1968*b* analysis of the terms for **T** without this restriction on **(IR)**, establish its consistency by the same methods that are needed to prove the consistency of the system with **(IR)** of type 0 only. This is so since, in their systems, the two equality relations coincide for closed terms of the same arbitrary type.

4.4. The induction rule **(IR)** is equivalent to the induction rule in the following, more usual, form:

$$\textbf{(IR)}' \qquad \frac{a(\bar{0})=b(\bar{0}),\ a(t)=b(t)\to a(S_0t)=b(S_0t)}{a(t)=b(t)}$$

This fact is proved as in recursive arithmetic (cf. Goodstein 1957). In **(IR)**′ it is of course assumed that $a(x)$ and $b(x)$ are of atomic type. If we added to **T**′ an equality functional E^τ of type $\tau\to\tau\to 0$ for all types τ with the rule

$$(E) \qquad a=b \text{ iff } Eab=\bar{0}, \text{ for terms } a \text{ and } b \text{ of the same finite type (even when } Eab=\bar{0} \text{ follows by (IR)),}$$

as in Troelstra 1970, then we could derive **(IR)′** from **(IR)** even if we permitted induction of all finite types. But this result can be sharpened. In the presence of rule (E), **(IR)′** of type 0 implies **(IR)′** of all finite types. For suppose that $a(x)$ and $b(x)$ are terms of type τ and x a variable of type 0 and assume that we have

$$a(\bar{0}) = b(\bar{0})$$

and

$$a(t) = b(t) \to a(S_0 t) = b(S_0 t).$$

Let

$$f \equiv \lambda x.\, Ea(x)b(x)$$

Then by (E) we have

$$f\bar{0} = \bar{0} \quad \text{and} \quad ft = \bar{0} \to f(S_0 t) = \bar{0}.$$

By **(IR)′** of type 0 we infer $ft = \bar{0}$, i.e. $Ea(t)b(t) = \bar{0}$, so the result $a(t) = b(t)$ follows by (E).

However, the equality relation between terms of higher types obtained by adjoining (E) will not be further considered here.

5. SOUNDNESS OF THE AXIOMS

5.1. As already pointed out, a statement $a =_i b$ means that we are able to see immediately by inspection of the linguistic expression a and b that they express the same rule of correspondence or have the same meaning.

It is then immediate that the equality axioms $(i\,1), \ldots, (i\,5)$ are justified. $(i\,1)$, $(i\,2)$, $(i\,3)$ for the reason that we conceive of the relation 'having the same meaning' as being an equivalence relation. $(i\,4)$ and $(i\,5)$ are justified by the following semantical principle:

> A substitution of an expression a for an expression b with the same meaning as a in an expression c does not change the meaning of c.

This principle is implicit in the relation of having the same meaning. It could be considered as an adequacy condition for this relation. Since we also substitute open terms, as in the case of $(i\,5)$, the question arises of what it means to say that two open terms have same meaning, or, in other words, what it means for an equation between open terms to be valid in the sense of intensional equality. One might think that $(i\,5)$ involves proof. This is not the case since the premiss of $(i\,5)$ requires that we should see simply by inspection of the terms $a(x)$ and $b(x)$ that they will have the same meaning *uniformly* in any meaning assigned to x.

5.2. The axioms and rules for the recursion constants are valid by assumption. They *give* the constants their meaning. The axiom $(i\,8)$ is valid for the same reason. It is the defining equation for λ-abstraction. The axioms $(i\,6)$, $(i\,7)$ an $(i\,8)$ express the principles of definition of computable functionals that we allow ourselves to use.

5.3. The axiom $(i\,9)$ is the crucial one. Let r denote a rule that assigns objects of type σ to objects of type τ and consider the rule r' described as follows:

> r': To an object x of type τ assign the object obtained by applying the rule r to x.

The axiom $(i\,9)$ asserts that r and r' denote the same rule. Thinking of a rule as a process whereby one steps from argument to value this appears not to be true. The process r' is almost the same as r but it seems to contain an additional initial step. One might however think that this additional step is redundant and does not reflect any significant difference between the rules r and r', or, in other words, that r' is just a more complicated way of communicating the rule r.

However, $(i\,9)$ cannot be justified only by the meaning given to λ-abstraction by its defining equation. Using this meaning only, we can see that $\lambda x.ax$ and a will aways have the same value uniformly in their argument, i.e. that

$$(\lambda x.ax)x =_i ax$$

but to conclude that therefore $\lambda x.ax =_i a$, requires an additional assumption. And to adopt $(i\,9)$ would be to make this assumption

implicit in the primitive notion of a computable functional. I think that this is a reasonable assumption concerning the notion of function of the present chapter, since we can still maintain the philosophical point of view explained in the introduction. By adding (*i* 9) a statement $a =_i b$ still does not involve any abstract notion of proof, because we are able to decide wether a and b express the same rule simply by inspection of the terms a and b. This will be verified in section 8.

It should also be noted that (*i* 9) asserts something new only when a is not equal to a term of the form $\lambda y.a'(y)$. For suppose that a is equal to a term of this form, then we have

$$\lambda x.ax = \lambda x.(\lambda y.a'(y))x$$
$$= \lambda x.a'(x) = a.$$

So when the term a in (*i* 9) is equal to a term of the mentioned form, then (*i* 9) is justified by the meaning given to λ-abstraction by its defining equation.

Another way of expressing (*i* 9) in T′ is as follows:

(*i* 9)′ If a and b are terms not containing x free and $ax =_i bx$, then $a =_i b$.

This says that two computable functionals that can be seen to have the same values uniformly for the same arguments are identical.

The rule (*i* 9)′ is not to be confused with the principle of extensionality in T′, of which there are two versions. The principle of *weak extensionality*, first introduced by Spector 1962, is this:

$$\frac{ax_1 \ldots x_n = bx_1 \ldots x_n}{a = b}$$

where $x_1, \ldots, x_n$ are variables not occurring free in a and b and such that the left and right hand sides in the premiss have atomic type. Equality in the premiss is as in section 3.

A notion of extensionality, $a =_e b$, mentioned for example by Kreisel 1965, is defined as follows: (i) For terms a and b of the same atomic type $a =_e b$ means $a = b$ as in section 3. (ii) Suppose that we have defined $a =_e b$ for terms of type σ and τ, then for terms a and b of type $\sigma \to \tau$, $a =_e b$ means that for all terms c and d of type σ such that $c =_e d$ we have $ac =_e bd$.

In the presence of (*i* 9), it is possible to derive simultaneous recursion in T′ as in Stenlund 1971. This is needed for the Dialectica interpretation of the induction axioms of intuitionistic analysis. But even with the weaker notion of intensional equality, without (*i* 9), it is possible to derive simultaneous recursion.[1]

6. DEFINING AXIOMS AND UNIQUENESS RULES

6.1. It is instructive to consider the relationship between the axioms and the inference rules of T′ and the classification of axioms for theories in so called standard formalization. We have first the equality axioms. The remaining axioms fall into two groups: the non-logical and the logical axioms. In T′ the axioms and rules for the primitive constants play the role of the former, while the axioms and rules for application and λ-abstraction play the role of the latter. As pointed out in the introduction application and λ-abstraction are our means for expressing the "logic of rules of correspondence" in T′.

In this section we shall call attention to another classification of the axioms and rules of T′. This classification emphasizes the fact that some of the axioms and rules of T′ (to the left below) express the defining properties of the primitive operations, while others, the *uniqueness rules* (to the right below), assert a uniqueness property of the primitive operations with respect to the intended interpretation.

6.2. For the constants R_0, R_1 and λ-abstraction the defining axioms and the uniqueness rules are the following:

$R_0 ab\bar{0} =_i a$
$R_0 ab(S_0 t) =_i bt(R_0 abt)$

$\dfrac{f\bar{0} = a,\ f(S_0 t) = bt(ft)}{ft = R_0 abt}$

$R_1 ab0_1 =_i a$
$R_1 ab(S_1 t) =_i bt(\lambda x.R_1 ab(tx))$

$\dfrac{f0_1 = a,\ f(S_1 t) = bt(\lambda x.f(tx))}{fc = R_1 abc}$

$(\lambda x.a(x))b =_i a(b)$

$\dfrac{bx =_i (\lambda x.ax)x}{b =_i \lambda x.ax}$ (x not free in a and b)

[1] This was shown to me by Hindley, who learned it from Schütte.

The defining equations to the left give the meaning to the recursion constants and λ-abstraction, and the uniqueness rules to the right assert the uniqueness of the functions defined by these operations.

There is an analogy with the situation for systems of natural deduction (cf. Prawitz 1970), where there are two rules for each primitive logical operation; an introduction and an elimination rule. Also in analogy with the situation for systems of natural deduction, the uniqueness rules and the defining equations may be said to be *inverses* of one another. For example, an inference by the induction rule can be thought of as "the inverse" of a computation according to the defining equation of the recursion functional R_0.

As explained by Prawitz 1970, in systems of natural deduction an introduction rule for a logical operation can be interpreted as giving the meaning to that operation, and the corresponding elimination rule can be seen to be justified by this meaning. Here the analogy seems to end. Under the intensional interpretation of equality, it is the defining equations (and the formation rules) that give the primitive operations their meaning, and as we have already pointed out, the uniqueness rules assert something more of the intended interpretation.

6.3. The defining properties of the primitive constants 0_0, 0_1, S_0, S_1, are simply those implicit in the formation rules for terms of $\mathbf{T}'$. 0_0 is a constant of type 0 and S_0 is a constant of type $0\to 0$. The defining properties are thus that 0_0 denotes an object of type 0 and that S_0 is a rule that assigns objects of type 0 to objects of type 0. Their uniqueness properties seem to be those expressed in the third and fourth Peano axioms, i.e.

$$\neg\,(0_0 = S_0 t)$$

$$S_0 t = S_0 t' \to t = t'.$$

As in the case of R_0 and R_1, the uniqueness properties of 0_0 and S_0 cannot be said to be justified by their defining properties. The 3rd and 4th Peano axioms express something more about the intended standard model. The situation is similar in the case of 0_1 and S_1.

7. REDUCTION RULES

7.1. As pointed out in the introduction, the defining equations of the primitive operations can be thought of as *computation rules* for the computable functionals. On the formal level these computation rules are represented as reduction rules. This leads to the notion of *definitional equality*: Two terms are definitionally equal if they reduce to a common term.

The notions of intensional and definitional equality have usually been used as synonyms. We prefer here to use the latter terminology when we think of the defining equations as computation rules for the computable functionals and not only as assertions about them. As will be proved below, the two notions are (extensionally) equivalent.

7.2. We describe the redexes and their corresponding contracta in the following table.

Redex	*Contractum*
$R_0 ab\bar{0}$	a
$R_0 ab(S_0 t)$	$bt(R_0 abt)$
$R_1 ab0_1$	a
$R_1 ab(S_1 t)$	$bt(\lambda x.R_1 ab(tx))$
$(\lambda x.a(x))b$	$a(b)$
$\lambda x.ax$	a

In the last redex and in the fourth contractum it is assumed, as usual, that x is a new variable.

When a does not reduce to a term of the form $\lambda y.a'$, the reduction $\lambda x.ax$ *red.* a seems not to fit in with the interpretation of the reduction rules as computation rules. A better name for it would perhaps be *the simplification rule.*

A *contraction* of a redex in a term a is defined as usual as the replacement of a subterm of a which is a redex by its contractum. A term a *reduces* to a term b, in symbols $a \geqslant b$, if b is obtained from a by repeated (possibly zero) contractions. A term is *normal* or *in normal form* if it does not contain a subterm which is a redex.

The redexes of the first four kinds are R-redexes in the sense of chapter 2. The following result is therefore proved as in chapter 2.

7.2.1. THEOREM. (The Church–Rosser property) *If $a \geqslant b$ and $a \geqslant c$, then there is a term d such that $b \geqslant d$ and $c \geqslant d$.*

From this theorem we infer the transitivity of the relation of definitional equality, hence we have the following result.

7.2.2. COROLLARY. *Definitional equality is an equivalence relation.*

The following theorem shows that our notions of intensional and definitional equality are equivalent.

7.2.3. THEOREM. *Two terms a and b are definitionally equal if and only if $a =_i b$ is derivable in* T'.

Proof. That the definitional equality of a and b implies that $a =_i b$ is derivable in the equation calculus is obvious.

The converse is proved by induction on the derivation of $a =_i b$. In the cases in which the equation is derived by $(i1)$, $(i2)$ or $(i3)$, the result follows by corollary 7.2.2, and hence by the Church–Rosser theorem. The remaining cases follow immediately by the induction hypothesis.

8. COMPUTABILITY AND NORMAL FORM

8.1. Let $a > b$ mean that $a \geqslant b$ but $a \not\equiv b$. A term a is *strongly normalizable* (a is SN) if each reduction sequence

$$a > a_1 > a_2 > \ldots$$

starting with a is finite. In other words, a is SN if a is normalizable and each reduction sequence starting with a ends in the normal form of a. The uniqueness of the normal form of a normalizable term follows by the Church–Rosser theorem as in chapter 1.

In the type-free λ-calculus, there are terms which are normalizable but not SN. An example is the term

$$(\lambda x.y)((\lambda x.xx)(\lambda x.xx)).$$

All terms of T' are SN, as we shall prove now. We shall do this by defining what it means for a term to be *computable* and then prove that

(i) each computable term is SN,

(ii) each term is computable.

8.2. *Computability.* By induction over the type τ of a term a, we define what it means for a to be computable.

8.2.1. If a has type 0, then a is computable if a is SN.

8.2.2. If a has type 1, then a is computable provided that a is SN and that t is computable if a reduces to a term of the form $S_1 t$.

8.2.3. If a has type $\tau \to \sigma$, then a is computable if ab is a computable term of type σ for all computable b of type τ.

Remark. If $a \geqslant b$, and a is computable, then b is computable. This is obvious for terms of atomic type. For terms of higher types it follows from the following fact: A term a is computable if $aa_1 \ldots a_n$ is computable for all computable $a_1, \ldots, a_n$ such that $aa_1 \ldots a_n$ has atomic type.

Then we proceed to the proof of (i) and (ii).

8.2.4. THEOREM. *Each term of type τ and of the form $xa_1 \ldots a_n$, $n \geqslant 0$, with all of $a_1, \ldots, a_n$* SN *is computable, and each computable term of type τ is* SN.

Proof. The proof is by induction on the complexity of τ.

Case 1. τ is atomic. Then each computable term of type τ is SN by definition. If all of $a_1, \ldots, a_n$ are SN, then clearly so is $xa_1 \ldots a_n$ because each reduction of this term must proceed within the terms $a_1, \ldots, a_n$. It can't reduce to a term of the form $S_1 t$, so by 8.2.1 and 8.2.2 the result follows.

Case 2. Let a be a computable term of type $\tau \to \sigma$ and x a variable of type τ. By the induction hypthesis (with $n = 0$) x computable, and since a is computable, we conclude by 8.2.3 that ax is computable. By the induction hypothesis ax is SN and hence, so is a.

To prove the remaining part of the theorem, let $xa_1 \dots a_n$ be a term of type $\tau\to\sigma$ with all of $a_1, \dots, a_n$ SN. Let b a computable term of type τ. By the induction hypothesis b is SN. Hence, all of $a_1, \dots, a_n, b$ are SN, so, applying the induction hypothesis again, we conclude that $xa_1 \dots a_n b$ is computable, and by 8.2.3 so is $xa_1 \dots a_n$.

This completes the proof.

8.2.5. LEMMA. *The constants 0_0, 0_1, S_0 and S_1 are computable.*

Proof. Since 0_0 and 0_1 are normal terms and the latter not of the form $S_1 t$, they are computable by definition.

To see that S_0 is computable, let b be a computable term of type 0. By theorem 8.2.4 it follows that b, and hence $S_0 b$ is SN. By 8.2.1 this means that $S_0 b$ is computable and by 8.2.3 it follows that S_0 is computable.

The computability of S_1 follows similarly using 8.2.2. This completes the proof.

8.2.6. LEMMA. *If $a(b)$ is computable, then $(\lambda x.a(x))b$ is computable, provided that b is computable if x is not free in $a(x)$.*

Proof. Let $a_1, \dots, a_n$ be computable terms such that

(1) $\qquad a(b)a_1 \dots a_n$

has atomic type. Since $a(b)$ is computable, it follows by 8.2.3 that (1) is computable and hence SN. The lemma will follow by 8.2.3 if we prove that the term

(2) $\qquad (\lambda x.a(x))ba_1 \dots a_n$

is computable. If (2) reduces to a term of the form $S_1 t$, then so does (1) so t is computable, and it remains only to prove that (2) is SN. Since (1) is SN, so are all its subterms $a(b)$, b, $a_1, \dots, a_n$, and b is SN also if it does not occur in a. Therefore, an infinite reduction of (2) must be an infinite reduction of $\lambda x.a(x)$. Such a reduction must either yield a reduction of the form

$$a(x) > a'(x) > a''(x) > \dots$$

but then,

$$a(b) > a'(b) > a''(b) \dots$$

would be an infinite reduction of $a(b)$, which is impossible, or else, it is like this

$$\lambda x.a(x) \geqslant \lambda x.ax > a' > a'' \dots$$

with x not free in a, a', a'', ... But then the reduction

$$a(b) \geqslant a'b > a''b \dots$$

would be an infinite reduction of $a(b)$, which is still impossible. This completes the proof.

8.2.7. LEMMA. *The constants R_0 are computable.*

Proof. By 8.2.3 it is sufficient to prove that R_0abc is computable for all computable a, b and c of appropriate types. We do this by induction on the number of successor constants S_0 in c. Let $a_1, \dots, a_n$ be computable terms such that $R_0abca_1 \dots a_n$ has atomic type.

Case 1. c does not reduce to a term of the form 0_0 or $S_0 t$. Then each reduction of $R_0abca_1 \dots a_n$ must proceed within the terms a, b, c, a_1, ..., a_n. By theorem 8.2.4 they are all SN and hence, so is $R_0abca_1 \dots a_n$. Since the latter term cannot reduce to a term of the form $S_1 t$, it follows by 8.2.1 and 8.2.2 that it is computable, and by 8.2.3 so is R_0abc.

Case 2. c reduces to 0_0. If this would cause $R_0abca_1 \dots a_n$ to have an infinite reduction, it would have to begin like this

$$R_0abca_1 \dots a_n \geqslant R_0a'b'0_0a_1' > a'a_1' \dots a_n' > \dots$$

where a, b, $a_1, \dots, a_n$ reduces to a', b', $a_1', \dots, a_n'$, respectively, in a finite number of steps. Since a, $a_1, \dots, a_n$ are all computable, so is $aa_1 \dots a_n$, and since

$$aa_1 \dots a_n \geqslant a'a_1' \dots a_n',$$

it follows that $a'a_1' \dots a_n'$, and hence $R_0abca_1 \dots a_n$ is SN and if it reduces to a term of the form $S_1 t$, then t is computable. So if c does not reduce to a term of the form $S_0 t$, it follows that R_0abc is computable, and it remains only to consider this case.

Case 3. c reduces to $S_0 t$. Since $S_0 t$ is SN, so is t which by 8.2.1

means that t is computable. By the induction hypothesis R_0abt is computable. An infinite reduction of $R_0abca_1 \dots a_n$ would have to start like this

$$R_0abca_1 \dots a_n \geqslant R_0a'b'(S_0t')a_1' \dots a_n'$$
$$> b't'(R_0a'b't')a_1' \dots a_n' \geqslant \dots$$

where a, b, t, a_1, ..., a_n reduce to a', b', t', a_1', ..., a_n', respectively, in a finite number of steps. Since all of b, t, R_0abt, a_1, ..., a_n are computable so is $bt(R_0abt)a_1 \dots a_n$, and since

$$bt(R_0abt)a_1 \dots a_n \geqslant b't'(R_0a'b't')a_1' \dots a_n'$$

it follows that the above reduction of $R_0abca_1 \dots a_n$ is finite and if it reduces to a term of the form S_1t, then t is computable.

Thus we have proved that R_0abc is computable for all computable a, b and c and the lemma follows.

8.2.8. LEMMA. *The constants R_1 are computable.*

Proof. As in the proof of lemma 8.2.7, it is sufficient to prove that R_1abc is computable for all computable a, b and c of appropriate types. This is done by induction on the number of successors in c. The cases in which c does not reduce to a term of the form S_1t are exactly like the corresponding cases in the proof of lemma 8.2.7, so let us consider the case in which c reduces to a term of this form. Since c is computable it follows by 8.2.2 that t is a computable term of type $0 \to 1$. The induction hypothesis is then that the term

$$R_1ab(te)$$

is computable for all computable e of type 0. The lemma will follow as lemma 8.2.7, if we prove that the reduction

$$R_1abca_1 \dots a_n \geqslant R_1a'b'(S_1t')a_1' \dots a_n'$$
$$> b't'(\lambda x.R_1a'b'(t'x))a_1' \dots a_n' > \dots$$

is finite and that d is computable, if it reduces to a term of the form S_1d. This follows clearly from the fact that

$$bt(\lambda x.R_1ab(tx))a_1 \dots a_n \geqslant b't'(\lambda x.R_1a'b'(t'x))a_1' \dots a_n'$$

if we prove that the left hand side of this relationship is a computable term. Now all of b, t, a_1, ..., a_n are computable, so it remains only to prove that $\lambda x.R_1ab(tx)$ is computable. By the induction hypothesis and

$$(\lambda x.R_1ab(tx))e \geqslant R_1ab(te),$$

we conclude that $(\lambda x.R_1ab(tx))e$ is computable for all computable e using lemma 8.2.6. By 8.2.3 it follows that $\lambda x.R_1ab(tx)$ is computable and the proof is complete.

We have proved the computability of the primitive constants and the variables. The computability of all terms is an immediate consequence of the following theorem.

8.2.9. THEOREM. *Let $a(x_1, \dots, x_n)$ be a term all of whose free variables are among the ones shown, and let $b_1, \dots, b_n$ be computable terms of the same types as $x_1, \dots, x_n$, respectively. Then $a(b_1, \dots, b_n)$ is computable.*

Proof. The proof is by induction of the construction of $a(x_1, \dots, a_n)$. To simplify notation we write a' for $a(b_1, \dots, b_n)$.

Case 1. If $a(x_1, \dots, x_n)$ is a variable x_i, $1 \leqslant i \leqslant n$, then a' is b_i and the result is immediate.

Case 2. If a is a variable distinct from all of $x_1, \dots, x_n$ or if a is a primitive constant, then $a' \equiv a$ and the result follows from 8.2.4, 8.2.5, 8.2.7 and 8.2.8.

Case 3. If a is a_1a_2, then a' is $a_1'a_2'$. By the induction hypothesis a_1' and a_2' are computable, and by 8.2.3 so is $a_1'a_2'$.

Case 4. a is $\lambda x.a_1(x)$. By the induction hypothesis $a_1'(b)$ is computable for all computable b. Since

$$(\lambda x.a_1'(x))b \geqslant a_1'(b),$$

the computability of $\lambda x.a_1'(x) \equiv a'$ follows by 8.2.6 and 8.2.3.

This completes the proof.

8.2.10. STRONG NORMALIZATION THEOREM. *Each term is computable and hence, strongly normalizable.*

8.3. *Consequences.* The strong normalization theorem has important consequences. Each term has precisely one normal form which can be found *mechanically* in a finite number of steps by reducing the term. Hence, it can be decided effectively whether two terms a and b are definitionally equal by checking whether they have the same normal form. Since definitional and intensional equality are equivalent relations, we have verified what one expects from the philosophical point of view of section 1.2 and 1.3:

8.3.1. THEOREM. *It is mechanically decidable whether $a =_i b$ is derivable or not in the equation calculus* **T'**.

It is easy to see what the normal terms must look like. A term a is normal in one of the following cases.

(i) a is 0_0 or 0_1,

(ii) a is S_i or $S_i b$ and b is normal, $i=0, 1$,

(iii) a is $xa_1 \dots a_n$ and $a_1, \dots, a_n$ are normal, $n \geqslant 0$,

(iv) a is $R_i a_1 \dots a_n$, $n \geqslant 0$, and $a_1, \dots, a_n$ are normal terms and a_3 is not of the form 0_i or $S_i b$,

(v) a is of the form $\lambda x.a'$, where a' is a normal term not of the form bx with x not free in b.

Since each term is strongly normalizable, we have:

8.3.2. THEOREM. *Each closed term of type* 0 *reduces to a numeral.*

Similarly, we have.

8.3.3. THEOREM. *Each closed term of type* 1 *either reduces to 0_1 or to a term of the form $S_1 c$, where c is a closed normal term of type* $0 \to 1$. *Given two terms a and b of type* 1 *and a numeral $\bar{n}$, it can be decided whether or not a is the n:th predecessor of b.*

The next theorem yields the consistency of the equality relation between terms of type 0 and 1.

8.3.4. THEOREM. *Let a and b be closed terms of the same atomic type, then $a=b$ is derivable in* **T'** *if and only if $a =_i b$ is derivable.*

Proof. $a =_i b$ implies $a=b$ by (e 6). If $a=b$ has been derived without using (**IR**), then it is easy to see that $a =_i b$ is also derivable.

We complete the proof by showing how to remove each application of (**IR**) from a proof of an equation $a=b$ with a and b closed. First, we note that if there occurs a free variable (other than as a t in the premiss of (**IR**)) in the derivation of $a=b$, we can replace it by a suitable closed term and still obtain a derivation of $a=b$. Hence, we may assume that in each application of (**IR**), the terms in the conclusion are closed. The t in the conclusion hence reduces to a numeral $\bar{n}$. We prove by induction on n that this application of (**IR**) can be removed.

If $n=0$, then the conclusion $ft=R_0abt$ follows from the axiom $R_0ab\bar{0}=a$ and the premiss $f\bar{0}=a$ and the fact that $t=\bar{0}$.

Suppose that $t \geqslant S_0\bar{n}$ and that

$$f\bar{n} = R_0ab\bar{n}$$

can be derived without using (**IR**), then so does

$$b\bar{n}(f\bar{n}) = b\bar{n}(R_0ab\bar{n}) = R_0ab(S_0\bar{n}).$$

By the second premiss of (**IR**), we have

$$f(S_0\bar{n}) = b\bar{n}(f\bar{n}),$$

and since $t=S_0\bar{n}$, the conclusion $ft=R_0abt$ follows without this application of (**IR**) and this completes the proof.

This theorem yields the consistency of **T'**. Two numerals cannot be proved to be equal unless they are identical. As a consistency proof of intuitionistic analysis (via the Dialectica interpretation), this proof is, of course, of no significance from the point of view of reductive proof theory, because the proof of the strong normalization theorem uses in an essential way the reasoning formalized in intuitionistic analysis with bar-recursion of type 0.

The consistency of **T'** is of course immediate from the philosophical position explained in the introduction, and our purpose has not been

so much to establish consistency results as to analyse the computation rules of the computable functionals as represented by the reduction rules of **T**′.

8.4. As remarked in the introduction the computability method was introduced by Tait 1966 and 1967*a*. In the latter paper, Tait proves a normalization theorem for the closed terms of T, extended with a schema of bar-recursion of type 0. The reduction relation in Tait 1967*a* is different from ours. It is weaker than the one in section 7, and Tait 1967*a* considers only a specific deterministic way of reducing a term, while we have considered all possible ways of reducing terms.

A normalization theorem for the terms of **T** without bar-recursion which also covers open terms, was proved by Sanchis 1967.

The proof of the (strong) normalization theorem by the computability method can be given in different forms. The definition of computability can be given different but extensionally equivalent forms. We feel that the definition and proof of the present section have the simplest forms. The definition of computability reflects in a simple way the conceptual content of the computability notion and is largely independent of our particular formalism, in contrast, for example, to the notion of regularity in Sanchis 1967.

A quite different normalization proof was given by Howard 1970. He assigns ordinals to the terms such that the normalization theorem follows within primitive recursive arithmetic extended by transfinite induction up to Bachmann's ordinal $\varphi_c(0)$, where $c=\varepsilon_{\Omega+1}$.

All of the above mentioned proofs establish normalization theorems only, and none of their authors considers the reductions $\lambda x.ax \geqslant a$, where x is not free in a.

9. INTERPRETATION OF TYPES AND TERMS

In this section we shall introduce the notion of a *structure* and a *model* for **T**′. We consider different kinds of models for **T**′ and prove the completeness of the axioms and rules for intensional equality with respect to the intended interpretation of application.

9.1. *Structures.* Let V be the set of all types of **T**′. A *structure*

$$\Sigma = \langle \Sigma^\tau, \doteq^\tau, ap^{\tau\to\sigma} \rangle_{\tau,\sigma\in V}$$

for **T**′, consists of the following things:

9.1.1. For each $\tau \in V$ a non-empty set Σ^τ.

9.1.2. For each $\tau \in V$, $\doteq^\tau$ is the identity relation on Σ^τ.

9.1.3. For all $\tau, \sigma \in V$, a mapping

$$ap^{\tau\to\sigma}: \Sigma^{\tau\to\sigma} \times \Sigma^\tau \to \Sigma^\sigma.$$

We will use $a, b, c, \ldots$ to denote elements of a structure, although we have reserved them for terms. It will be clear from the context if we are talking about terms or about elements of a structure.

To simplify the notation we write

$$ab \text{ for } ap^{\tau\to\sigma}(a,b).$$

We will in general also omit type superscripts.

9.2. *Combinators.* To arrive at the notion of a model for **T**′, we consider a structure Σ for **T**′ satisfying the following conditions:

9.2.1. For all $\tau, \sigma \in V$, $\Sigma^{\tau\to\sigma\to\tau}$ contains an element $\bar{K}$ such that for all $a \in \Sigma^\tau$ and all $b \in \Sigma^\sigma$ we have

$$\bar{K}ab \doteq a.$$

(Note that this is an abbreviation of $ap^{\sigma\to\tau}(ap^{\tau\to\sigma\to\tau}(\bar{K}, a), b) \doteq^\tau a$)

9.2.2. For all $\tau, \sigma, \varrho \in V$, the set $\Sigma^{(\tau\to\sigma\to\varrho)\to(\tau\to\sigma)\to(\tau\to\varrho)}$ contains an element $\bar{S}$ such that for all $a \in \Sigma^{\tau\to\sigma\to\varrho}$, $b \in \Sigma^{\tau\to\sigma}$ and $c \in \Sigma^\tau$ we have

$$\bar{S}abc \doteq ac(bc).$$

9.2.3. The combinatory axioms of section 3.7 of chapter 1 are assumed to hold, when we understand $\bar{K}$ as the combinator **K**, $\bar{S}$ as the combinator **S**, $\bar{S}\bar{K}\bar{K}$ as **I** and juxtaposition and η-equality of chapter 1 as $ap(a, b)$ and $\doteq$, respectively.