J. Symbolic Computation (1992) **14**, 71–84

Constructing Type Systems over an Operational Semantics

ROBERT HARPER

Carnegie-Mellon University, Pittsburgh, PA 15213, *USA*

(*Received November* 1991)

Type theories in the sense of Martin-Löf and the NuPRL system are based on taking as primitive a type-free programming language given by an operational semantics, and defining types as partial equivalence relations on the set of closed terms. The construction of a type system is based on a general form of inductive definition that may either be taken as acceptable in its own right, or further explicated in terms of other patterns of induction. One such account, based on a general theory of inductively-defined relations, was given by Allen. An alternative account, based on an essentially set-theoretic argument, is presented.

1. Introduction

Research in type theory may be classified into three broad categories:

1. Proof theory. The emphasis here is on studying function calculi representing formal proofs in systems of natural deduction. On the syntactic side, strong normalization results are of particular interest (Tait, 1967; Girard, 1972; Martin-Löf, 1975; Coquand & Huet, 1985, 1988), particular since they entail the consistency of the type system as a constructive logic. On the semantic side, the definition and construction of models has been important (Troelstra, 1973; Martin-Löf, 1975; Donahue, 1979; McCracken, 1979; Bruce *et al.*, 1987; Hyland & Pitts, 1989). Well-known examples of type theories that fall into the proof-theoretic tradition are the simply typed λ-calculus (Barendregt, 1984; Hindley & Seldin, 1986), the second-order λ-calculus (Girard's System F) (Reynolds, 1974; Girard, 1972), Martin-Löf's early (Martin-Löf, 1975) (and most recent) type theories, and the Calculus of Constructions (Coquand & Huet, 1985; Coquand, 1986; Coquand & Huet, 1988).

2. Type assignment. Here the principal concern is with typeability of untyped λ-terms in a variety of type disciplines. On the syntactic side, the emphasis is on isolating interesting type disciplines (such as first- and second-order functional types (Hindley, 1969; Curry *et al.*, 1972; Mitchell, 1984), intersection types (Coppo & Dezani, 1978), type containment (Mitchell, 1984)), on characterizing the typeable terms in a given discipline (Hindley, 1969; Milner, 1978; Damas & Milner, 1982; Damas, 1985; Coppo & Dezani, 1978; Coppo *et al.*, 1980; Coppo & Giovanetti, 1983; Mitchell, 1984; della Rocca, 1987; Giannini & della Rocca, 1988), and on type inference algorithms (Hindley, 1969; Milner, 1978; Damas & Milner, 1982; della Rocca, 1987; Giannini & della Rocca, 1988). On the semantic side, the emphasis is on characterizing the theories of certain classes of models based on untyped λ-interpretations (Coppo *et al.*, 1980; Coppo & Giovanetti, 1983; Hindley, 1983; Barendregt *et al.*, 1983; Mitchell, 1986).

3. Realizability. Here the emphasis is on viewing types as predicates about a programming language defined by an operational semantics. Of particular interest is the development of type disciplines that are sufficiently rich to serve as specification languages for

0747-7171/92/070071+14 $03.00/0

programs, and the development of formal systems in which to conduct proofs of correctness. There is no clear proof theory/model theory distinction in these systems since the assertions are interpreted as statements about a fixed operational semantics, rather than as formal assertions subject to a variety of interpretations. The roots of this approach may be traced back to the Brouwer/Heyting semantics of intuitionistic logic (Brouwer, 1975; Heyting, 1956; Dummett, 1977) and to Kleene's realizability semantics (Kleene, 1952) (and its many descendants (Beeson, 1985; Troelstra, 1973)). Examples from contemporary computer science are Martin-Löf's type theory (Martin-Löf, 1982), the Göteborg type theory (Nordström *et al.*, 1988), and the NuPRL type theory (Constable *et al.*, 1986).

Such a broad categorization is necessarily an over-simplification. It serves, however, to place the subject of this paper in context.

The purpose of this paper is to give a set-theoretic account of Martin-Löf's semantics for a predicative type theory that includes dependent types and universes. Of course, the use of a set-theoretic construction robs the approach of any foundational significance, and would not be of any use to an intuitionist. However, if we ignore philosophical issues, and concentrate on type theory as a programming logic, then a set-theoretic explanation is less inappropriate, and provides a predagogically useful basis for introducing some of the central ideas of Martin-Löf's type theory. Both Allen and Mendler at Cornell developed a rigorous account of the inductive character of the definition (Allen, 1987*a*, *b*; Mendler, 1987). Allen's approach is based on an intuitionistically acceptable theory of inductively-defined relations, while Mendler's is a thoroughly set-theoretic account (essentially equivalent to ours). Allen's thesis provides an extremely elegant and careful analysis of a number of issues in the semantics of type theory, paying careful attention to philosophical as well as practical concerns. Mendler's account extends both Allen's and the present account by considering inductive and co-inductive types.

Two closely-related constructions are Aczel's construction of a Frege structure from a model of the untyped λ-calculus (Aczel, 1980; Aczel, 1983) and Beeson's realizability interpretation of type theory (Beeson, 1982; Beeson, 1985). Aczel's construction, which partly inspired the present approach, is based on a set-theoretic argument, but he conjectures that it could be made intuitionistically acceptable (Allen's work may be construed as bearing this out.) Beeson's is based on the (constructively unacceptable) device of inductively defining both a relation and its formal complement, then proving that they are complementary relations.

Responding to a similar impulse to cast the semantics of type theory in an independently-acceptable setting, Smith provides an interpretation of type theory in a logical theory of constructions (Smith, 1984). Unlike our account, Smith does not present a type system as an inductive definition, and may therefore be considered to be more faithful to the "open-ended" character of type theory. More recently, Aczel and Mender have developed these ideas further by considering a form of iterated inductive definition in LTC (Mendler & Aczel, 1988; Aczel & Carlisle, 1990; Mendler, 1990).

2. Preliminaries

The semantics for type theory that we shall develop is based on an inductive construction of a system of relations between terms interpreted by an operational semantics. Since the terminology and notation for the relations that we shall consider are not well-established, we set down our definitions here.

A symmetric and transitive binary relation E on a set X is called a *partial equivalence relation* (*per*). The *field* of E is defined by $|E| = \{x \in X \mid E(x, x)\}$. It is easy to see that a per is an equivalence relation on its field. The equivalence class under E of an element $x \in X$, is defined by $E[x] = \{y \in X \mid xEy\}$. Note that $E[x] \subseteq |E|$, and that if $x \notin |E|$, then $E[x] = \emptyset$. The *quotient* of X by E, X/E is defined to be the set of non-empty equivalence classes of elements of X under E. If E is a per, then for every x, y, and z in $|E|$, if $x \in E[y]$, then $y \in E[x]$, and if $x \in E[y]$ and $y \in E[z]$, then $x \in E[z]$. Conversely, E is determined by specifying $|E|$ and, for each $x \in |E|$, a set $E[x] \subseteq |E|$ satisfying these conditions.

Let $(X, \sqsubseteq)$ be a partially-ordered set. A subset $D \subseteq X$ is *directed* iff every pair of elements in D has an upper bound in D; in particular, every chain (linearly-ordered subset) is directed. The poset X is a *complete pointed partial order*, or *cppo*, iff it has a least element $\perp$, and every non-empty directed subset $D \subseteq X$ has a supremum, $\bigsqcup D$, in X. A function $f: X \to Y$ between cppos is *monotone* (or *order preserving*) iff $f(x) \sqsubseteq f(y)$ whenever $x \sqsubseteq y$. A monotone function is *continuous* if it preserves countable directed suprema.

Every monotone map on a cppo X has a least fixed point. To see this, construct the sequence $\langle x_\alpha \rangle$ of elements of X indexed by ordinals as follows: $x_0 = \perp$, $x_{\alpha+1} = f(x_\alpha)$, and $x_\lambda = \bigsqcup_{\alpha<\lambda} x_\alpha$. It is easy to show by transfinite induction that at each stage α, the initial segment of the sequence determined by α is directed, and so the required suprema exist. Since X is a set, and each x_α is an element of X, there must be a stage λ at which $x_{\lambda+1} = x_\lambda$, for otherwise there would be a bijection between ON and X, which is impossible. Now x_λ is a fixed point of f, since $x_{\lambda+1} = f(x_\lambda)$. Furthermore, if y is any fixed point of f, then $x_\alpha \sqsubseteq y$ for each α, and so $x_\lambda \sqsubseteq y$, completing the proof. Note that for continuous maps it is not necessary to appeal to a cardinality argument to establish the fixed-point property since the sequence closes off at ω.

Let S be a set (of *sorts*). An S-sorted set X is a family of sets $X = \langle X_s \rangle_{s \in S}$ indexed by sorts. An S-sorted relation R between S-sorted sets X and Y is an S-indexed family of relations $R = \langle R_s \rangle_{s \in S}$ such that for each $s \in S$, $R_s \subseteq X_s \times Y_s$. An S-sorted partial function f between S-sorted sets X and Y is an S-sorted relation between X and Y that is a partial function at each sort. Relations between and operations on S-sorted sets are defined "sort-wise," so that, for example, $X \subseteq Y$ iff $X_s \subseteq Y_s$ for each $s \in S$. If x is a variable ranging over an S-sorted set X, then, by convention, x_s ranges over X_s, and the subscripts are dropped whenever they are clear from context.

3. Language

In this section we define the syntax of a small programming language that we shall use as the basis for illustrating the construction of type systems. Since several of the program forms are binding operators, it is convenient to present the language as a set of expression constructors using the system of *arities* introduced by Martin-Löf.

An *arity* α is an n-tuple of arities, for $n \geq 0$. The terms of the arity calculus are similar to those of the simple-typed λ-calculus with only one base type. A closed term of arity $(\alpha_1, \ldots, \alpha_n)$ is to be thought of as a term with n "holes," with the ith hole awaiting a term of arity α_i. A closed term of ground arity, ( ), or 0, is called a "saturated" or "completed" term since it has no holes. The inspiration for this view of expressions comes from Frege's conception of functions as arising from completed entities by striking out

Table 1. Signature of a small language

Canonical type forms		Canonical object forms		Non-canonical forms	
Form	Arity	Form	Arity	Form	Arity
U_i $(i \in \omega)$	0				
nat	0	0	0	rec	(0, 0, (0, 0))
		s	(0)		
I	(0, 0, 0)	ax	0		
×	(0, 0)	pair	(0, 0)	split	(0, (0, 0))
+	(0, 0)	inl	(0)	case	(0, (0), (0))
		inr	(0)		
→	(0, 0)	λ	((0))	ap	(0, 0)
Π	(0, (0))				
Σ	(0, (0))				

a component, leaving an incomplete entity that may be "applied" by filling in the hole with an entity of the appropriate kind (arity).

Let $\mathscr{X}$ be a denumerable arity-sorted set of variables such that $\mathscr{X}_\alpha \cap \mathscr{X}_\beta = \emptyset$ whenever α is distinct from β. Let x, y, and z range over $\mathscr{X}$. Let $\mathscr{C}$ be a denumerable arity-sorted set of constants, disjoint from the variables. Let c and d range over $\mathscr{C}$. A *signature* σ is a finite subset of $\mathscr{C}$. The set of terms, $\mathscr{T}(\sigma)$, generated by a signature σ is the least arity-sorted set $\mathscr{T}$ such that $\mathscr{X} \subseteq \mathscr{T}$, $\sigma \subseteq \mathscr{T}$, $a(a_1, \ldots, a_k) \subseteq \mathscr{T}$ if $a \in \mathscr{T}_{(\alpha_1, \ldots, \alpha_k)}$ and, for $1 \leq i \leq k$, $a_i \in \mathscr{T}_{\alpha_i}$, and $x_1, \ldots, x_k \,.\, a \in \mathscr{T}_{(\alpha_1, \ldots, \alpha_k)}$ if, for $1 \leq i \leq k$, $x_i \in \mathscr{X}_{\alpha_i}$ and $a \in \mathscr{T}_0$. The metavariables a, b, c, f, g range over $\mathscr{T}(\sigma)$.

The notions of free and bound variables, and capture-avoiding substitution are defined in the usual way, provided that we take $x_1, \ldots, x_k$ to be bound in a in $x_1, \ldots, x_k \,.\, a$. Write $\mathrm{FV}(a)$ for the set of free variables in a and $[a/x]b$ for substitution of a for free occurrences of x in b. If $\mathscr{T}$ is a set of terms, then the set of *closed* terms is the subset $\mathscr{T}^0 \subseteq \mathscr{T}$ consisting of those terms a such that $\mathrm{FV}(a) = \emptyset$. A closed term of ground arity is said to be *saturated*; let $\mathscr{S} = \mathscr{T}_0^0$ be the set of saturated terms.

Terms are identified up to the α, β, and η conversion, defined as the smallest congruence relation $\equiv$ containing all instances of

1. $x_1, \ldots, x_k \,.\, a \equiv y_1, \ldots, y_k \,.\, [y_1, \ldots, y_k / x_1, \ldots, x_k]a$,
2. $(x_1, \ldots, x_k \,.\, a)(a_1, \ldots, a_k) \equiv [a_1, \ldots, a_k / x_1, \ldots, x_k]a$, and
3. $x_1, \ldots, x_k \,.\, a(x_1, \ldots, x_k) \equiv a$.

In the sequel we work with a fixed language $\mathscr{T}$ generated by the signature appearing in Table 1. The table is divided into three columns, labelled by headings that suggest the role of the term formation operators in the operational semantics (canonical/non-canonical) and in the type system (type/object forms). A *canonical form* is a saturated term whose outermost constructor is labelled canonical in Table 1.

4. Operational Semantics

A programming language is defined by a syntax of expressions, a distinguished set of program expressions (usually closed terms, possibly with other restrictions), and an operational semantics defining a partial function mapping program expressions to values.

In the present situation the expressions are the members of the set $\mathscr{T}$ defined in the previous section, the program expressions are the members of the set $\mathscr{S}$ of saturated terms, and values $\mathscr{V}$ (ranged over by v and w) are the saturated terms in canonical form. The evaluation function is defined by an inductive definition of its graph, the relation $a \Rightarrow v$, which is the smallest relation closed under the rules of Figure 1. It is easy to see that $a \Rightarrow v$ is single-valued. We write $a\downarrow$ to mean that there exists v such that $a \Rightarrow v$, and $a \simeq b$ to mean that a and b evaluate to the same value.

The evaluator defined by the rules of Figure 1 is Martin-Löf's weak head reduction evaluator, for which the property of being a value is determined only by its outermost form. For other evaluation strategies it may be more convenient to define the set of values

$$\mathrm{U}_i \Rightarrow \mathrm{U}_i$$

$$\mathrm{nat} \Rightarrow \mathrm{nat} \qquad\qquad 0 \Rightarrow 0$$

$$\mathrm{s}(a) \Rightarrow \mathrm{s}(a)$$

$$\frac{a \Rightarrow 0 \quad b \Rightarrow c}{\mathrm{rec}(a,b,f) \Rightarrow c} \qquad\qquad \frac{a \Rightarrow \mathrm{s}(a') \quad f(a', \mathrm{rec}(a',b,f)) \Rightarrow c}{\mathrm{rec}(a,b,f) \Rightarrow c}$$

$$\mathrm{I}(a,b,c) \Rightarrow \mathrm{I}(a,b,c) \qquad\qquad \mathrm{ax} \Rightarrow \mathrm{ax}$$

$$a \times b \Rightarrow a \times b \qquad\qquad \langle a,b\rangle \Rightarrow \langle a,b\rangle$$

$$\frac{a \Rightarrow \langle a', a''\rangle \quad f(a',a'') \Rightarrow b}{\mathrm{split}(a,f) \Rightarrow b}$$

$$a + b \Rightarrow a + b \qquad\qquad \mathrm{inl}(a) \Rightarrow \mathrm{inl}(a)$$

$$\mathrm{inr}(a) \Rightarrow \mathrm{inr}(a)$$

$$\frac{a \Rightarrow \mathrm{inl}(a') \quad f(a') \Rightarrow b}{\mathrm{case}(a,f,g) \Rightarrow b} \qquad\qquad \frac{a \Rightarrow \mathrm{inr}(a') \quad g(a') \Rightarrow b}{\mathrm{case}(a,f,g) \Rightarrow b}$$

$$a \to b \Rightarrow a \to b \qquad\qquad \lambda(f) \Rightarrow \lambda(f)$$

$$\frac{a \Rightarrow \lambda(f) \quad f(b) \Rightarrow c}{\mathrm{ap}(a,b) \Rightarrow c}$$

$$\Pi(a,f) \Rightarrow \Pi(a,f) \qquad\qquad \Sigma(a,f) \Rightarrow \Sigma(a,f)$$

Figure 1. Operational semantics.

as those terms that evaluate to themselves, rather than giving a separate definition as we have done here (following Martin-Löf). It seems plausible that the type system construction described below goes through for an arbitrary evaluator, but this has not been investigated in detail.

5. Partial Equivalence Relations on Saturated Terms

In Martin-Löf's type theory a type is determined by defining those values that are to serve as elements and defining when two such values are to be equal. The presence of dependent types, and the ability to define type-valued functions on a type, makes it impossible to separate types from objects. The types themselves are therefore drawn from the collection of values, and it is part of the definition of a type system to define when two types are to be considered equal. Thus both the definition of the collection of types and the elements of each type have both a "collecting" and a "quotienting" aspect which is conveniently captured by using partial equivalence relations.

The pers that we shall consider are over the set $\mathscr{S}$ of saturated terms. It might seem at first sight that we could first consider pers over the subset $\mathscr{V} \subseteq \mathscr{S}$ of values, then extend to all saturated terms "at the end." But since the evaluator that we are considering is not compositional, we are forced to "interleave" evaluation within the definition of the pers representing type and member equality. One of the benefits of the construction to follow is that it provides a compositional way to reason about the non-compositional evaluator.

Viewed as a programming logic, type theory is a logic of total correctness in that a saturated term may serve as a type or a member of a type only if it has a value under the operational semantics. In particular, we shall see that a term inhabits a function type $A \to B$ only if it carries elements of A to elements of B, and hence is a *total* function. (It is possible to consider partial functions (and, more generally, partial objects) in this setting (Constable & Smith, 1987, 1988).) The formal realization of this aspect of type theory is the notion of an "evaluation-respecting" per.

Let PER denote the set of partial equivalence relations on the set of saturated terms $\mathscr{S}$. (Hereafter, we use "per" to refer only to elements of PER.) A per E *respects evaluation* iff

1. E relates only defined terms: if $a \in |E|$, then $a\downarrow$.
2. E respects Kleene equality: if $a \simeq a'$ and $b \simeq b'$, then $E(a, b)$ iff $E(a', b')$.

Let VPER denote the set of all pers that respect evaluation.

Taken together, these conditions amount to requiring that $E(a, b)$ hold iff $a \Rightarrow v$ and $b \Rightarrow w$ and $E(v, w)$: an evaluation-respecting per ("vper") is determined by its behaviour on values. Hence if E is a per on $\mathscr{V}$, there is a unique extension of E to a vper E^* on $\mathscr{S}$ obtained by defining $E^*(a, b)$ to hold iff there exists v and w such that $a \Rightarrow v$ and $b \Rightarrow w$ and $E(v, w)$. (This is essentially Martin-Löf's method of defining a type in terms of its canonical members.)

PER forms a cppo under the ordering $E \sqsubseteq E'$ iff $|E| \subseteq |E'|$ and for all $a \in |E|$, $E[a] = E'[a]$. Note that this is strictly stronger than requiring $E \subseteq E'$, which entails only that $E[a] \subseteq E'[a]$ for each $a \in |E|$. It is easy to see that this defines a partial ordering, with the empty relation as least element. Let $\mathscr{D}$ be a directed set of pers. The supremum $\bigsqcup \mathscr{D}$ of $\mathscr{D}$ is given by the per D such that $a \in |D|$ iff $a \in |E|$ for some $E \in \mathscr{D}$, in which case $D[a]$ is defined to be $E[a]$. This is well-defined since $\mathscr{D}$ is directed: if $a \in |E'|$ for some other $E' \in \mathscr{D}$, then $E[a] = E'[a]$ since they have an upper bound in $\mathscr{D}$. It is easy to see

that **VPER** is a sub-cppo of **PER** under the above ordering: we need only observe that the supremum of a directed set of vpers is itself a vper.

The following operations on vpers, inspired by Plotkin's "logical relations" (Plotkin, 1980) are used in the construction of type systems:

$$
\begin{aligned}
N &= \{(\bar{m}, \bar{m}) \mid m \in \omega\} \\
E \times F &= \{(\langle a, b\rangle, \langle a', b'\rangle) \mid E(a, a') \wedge F(b, b')\} \\
E + F &= \{(\mathrm{inl}(a), \mathrm{inl}(a')) \mid E(a, a')\} \cup \{(\mathrm{inr}(b), \mathrm{inr}(b')) \mid F(b, b')\} \\
E \to F &= \{(\lambda(f), \lambda(f')) \mid \forall a, a' \,.\, E(a, a') \supset F(f(a), f'(a'))\} \\
I(a, b, E) &= \{(\mathrm{ax}, \mathrm{ax}) \mid E(a, b)\} \\
\Pi(E, \Phi) &= \{(\lambda(f), \lambda(f')) \mid \forall a, a' \,.\, E(a, a') \supset \Phi(a)(f(a), f'(a'))\} \\
\Sigma(E, \Phi) &= \{(\langle a, b\rangle, \langle a', b'\rangle) \mid E(a, a') \wedge \Phi(a)(b, b')\}
\end{aligned}
$$

In the definitions of $\Sigma(E, \Phi)$ and $\Pi(E, \Phi)$, the choice of argument to Φ is immaterial, provided that Φ respects E. We shall only be interested in these operations when this is the case.

6. Type Systems

A type system may be thought of as a family of partial equivalence relations, one for type membership and equality, and one for the membership and equality of each type. More precisely, *type system* is a pair $\tau = (E, \Phi)$, where E is a vper and $\Phi : \mathscr{S}/E \to \mathbf{VPER}$ is a function assigning a vper to each $a \in |E|$ in such a way that if $E(a, b)$, then $\Phi(a) = \Phi(b)$. The relation E is called the type equality relation for τ, and, for each $a \in |E|$, the relation $\Phi(a)$ is called the member equality relation of type a in τ. Let $\mathbf{TS} = \Sigma_{E \in \mathbf{VPER}} (\mathscr{S}/E \to \mathbf{VPER})$ be the set of type systems.

Type systems are constructed as fixed points of monotone operators on type systems. We therefore need to consider a notion of approximation of one type system by another. The ordering we consider here regards a type system $\tau = (E, \Phi)$ to be an approximation of $\tau' = (E', \Phi')$ iff every type in E is a type in E', and its equivalents in E are the same as those in E', and, for every type a of E, the relation assigned to a by Φ' is the same as assigned to a by Φ. Formally, the approximation ordering on type systems is given by

$$(E, \Phi) \sqsubseteq (E', \Phi') \Leftrightarrow E \sqsubseteq E' \wedge \forall a \in |E| \,.\, \Phi(a) = \Phi'(a).$$

It is easy to see that **TS** is partially ordered by this relation, with $(\varnothing, \varnothing)$ as least element. Let $\mathscr{D} = \langle (E_i, \Phi_i) \rangle_{i \in I}$ be a directed set of type systems. The supremum of $\mathscr{D}$ is the type system $\bigsqcup \mathscr{D} = (E, \Phi)$, where $E = \bigsqcup_{i \in I} E_i$, and, for each $a \in |E|$, $\Phi(a)$ is $\Phi_i(a)$ for any $i \in I$ such that $a \in |E_i|$. Some such i must exist, by the definition of E, and any two choices agree since $\mathscr{D}$ is directed. It remains to show that Φ respects the equality relation given by E. But $E(a, b)$ holds only if $E_i(a, b)$ for some $i \in I$, in which case $\Phi(a) = \Phi_i(a) = \Phi_i(b) = \Phi(b)$, since each (E_i, Φ_i) is a type system.

7. A Fragment of Martin-Löf's System

In this section we construct a type system for a fragment of Martin-Löf's type theory without universes. The type system includes dependent product and sum types, and the equality type, and hence illustrates some of the characteristic features of type theory. Treatment of universes is deferred to the next section. This type system shall be obtained as the least fixed point of a monotone operator on **TS**. The construction may be motivated

by considering the iterative construction of fixed points described in section 2. Beginning with the empty type system, we have, at each stage α, a "partial" type system $\tau_\alpha = (E_\alpha, \Phi_\alpha)$. A type is said to "exist" at stage α iff it is a member of the field of E_α. Since τ_α is a type system, if $a \in |E_\alpha|$, then $\Phi_\alpha(a)$ is defined. At successor stages, the set of types is extended to include some set of "new" types constructed from the types existing at the previous stage. For example, if a and b are types existing at stage α, then the type $a \times b$ exists at stage $\alpha + 1$. At limit stages we simply collect together everything that has been constructed at earlier stages.

To make these ideas precise, we define an operator $\mathbf{T}: \mathsf{TS} \to \mathsf{TS}$ by $\mathbf{T}(E, \Phi) = (E'^*, \Phi'^*)$, where

$$
\begin{aligned}
E' = &\{(\text{nat}, \text{nat})\} \\
&\cup \{(a_1 \times a_2, a_1' \times a_2') \mid E(a_1, a_1') \wedge E(a_2, a_2')\} \\
&\cup \{(a_1 + a_2, a_1' + a_2') \mid E(a_1, a_1') \wedge E(a_2, a_2')\} \\
&\cup \{(a_1 \to a_2, a_1' \to a_2') \mid E(a_1, a_1') \wedge E(a_2, a_2')\} \\
&\cup \{(\mathrm{I}(a_1, a_2, a_3), \mathrm{I}(a_1', a_2', a_3')) \mid E(a_3, a_3') \wedge \Phi(a_3)(a_1, a_1') \wedge \Phi(a_3)(a_2, a_2')\} \\
&\cup \{(\Pi(b, f), \Pi(b', f')) \mid E(b, b') \wedge \forall a, a' \,.\, \Phi(b)(a, a') \supset E(f(a), f'(a'))\} \\
&\cup \{(\Sigma(b, f), \Sigma(b', f')) \mid E(b, b') \wedge \forall a, a' \,.\, \Phi(b)(a, a') \supset E(f(a), f'(a'))\}
\end{aligned}
$$

and

$$
\Phi'(a) = \begin{cases}
N & \text{if } a \equiv \text{nat} \\
\Phi(a_1) \times \Phi(a_2) & \text{if } a \equiv a_1 \times a_2 \wedge a_1, a_2 \in |E| \\
\Phi(a_1) + \Phi(a_2) & \text{if } a \equiv a_1 + a_2 \wedge a_1, a_2 \in |E| \\
\Phi(a_1) \to \Phi(a_2) & \text{if } a \equiv a_1 \to a_2 \wedge a_1, a_2 \in |E| \\
\mathrm{I}(a_1, a_2, \Phi(a_3)) & \text{if } a \equiv \mathrm{I}(a_1, a_2, a_3) \wedge a_3 \in |E| \\
\Pi(\Phi(b), \Phi(f)) & \text{if } a \equiv \Pi(b, f) \wedge b \in |E| \wedge \forall a \in |\Phi(b)| \,.\, f(a) \in |E| \\
\Sigma(\Phi(b), \Phi(f)) & \text{if } a \equiv \Sigma(b, f) \wedge b \in |E| \wedge \forall a \in |\Phi(b)| \,.\, f(a) \in |E|
\end{cases}
$$

Here we define $\Phi(f)(a) = \Phi(fa)$, for f of arity (0), and $\Phi^*(a) = \Phi(v)$ whenever $a \Rightarrow v$.

THEOREM 7.1.

1. **T** *maps type systems to type systems.*
2. **T** *is monotone.*
3. **T** *is not continuous.*

PROOF.

1. Suppose that $\tau = (E, \Phi)$ is a type system. It is easy to see that E'^* is a vper, and that Φ'^* assigns a vper to each $a \in |E'^*|$. It remains to show that Φ'^* respects E'^*. But this follows easily from the fact that Φ respects E and from the definitions of the operations on pers.

2. Let $\sigma = (E, \Phi)$ and $\tau = (F, \Psi)$ be type systems, and suppose that $\sigma \sqsubseteq \tau$. Let $\mathbf{T}(\sigma) = (E', \Phi')^*$ and let $\mathbf{T}(\tau) = (F', \Psi')^*$. We are to show that $E'^* \sqsubseteq F'^*$, and that $\Phi'^*(a) \sqsubseteq \Psi'^*(a)$ for each $a \in |E'^*|$.

Suppose that $a \in |E'^*|$. Then there exists a unique v such that $a \Rightarrow v$ and $v \in |E'|$. By definition of **T**, the value v must have one of seven possible forms; we consider two cases here. Suppose that $v = a_1 \times a_2$, with a_1 and a_2 elements of $|E|$. By supposition $|E| \subseteq |F|$, so that both a_1 and a_2 are in $|F|$, and hence, by the definition of **T**, $v = a_1 \times a_2 \in |F'|$, and hence $a \in |F'^*|$. To take another example, suppose that $v = \Pi(b, f)$, where $b \in |E|$ and for every $c \in |\Phi(b)|, f(c) \in |E|$. Now by supposition $|E| \subseteq |F|$, and hence $b \in |F|$, and $\Phi(b) = \Psi(b)$. But then $v = \Pi(b, f) \in |F'|$, and therefore $a \in |F'^*|$. By similar reasoning the equivalence class $E'^*[a]$ is equal to the equivalence class $F'^*[a]$ for each $a \in |E'^*|$.

Suppose that $a \in |E'|$, so that $a \Rightarrow v$ for some value v such that $v \in |E'|$. We are to show that $\Phi'(v) = \Psi'(v)$, from which the result follows directly. We proceed by cases on the possible form of v, based on the definition of **T**. Suppose that $v = a_1 \times a_2$ with $a_1 \in |E|$ and $a_2 \in |E|$. Then $a_1 \in |F|$ and $a_2 \in |F|$, and hence $\Psi(a_1) = \Phi(a_1)$ and $\Psi(a_2) = \Phi(a_2)$. But then

$$\begin{aligned}\Psi'(v) &= \Psi'(a_1 \times a_2)\\ &= \Psi(a_1) \times \Psi(a_2)\\ &= \Phi(a_1) \times \Phi(a_2)\\ &= \Phi'(a_1 \times a_2)\\ &= \Phi'(v)\end{aligned}$$

The other cases are handled similarly.

3. Consider the term $a \equiv \Pi(N, x.\, \mathrm{rec}(n; N; u, v.\, N \times v))$. At each finite stage i, only the i-fold product $N \times \cdots \times N$ exists, so that a exists only at the transfinite stage ω.

Let τ_0 be the least fixed point of **T**. As we shall see below, this type system is a model for the inference rules of Martin-Löf's type theory, restricted to the type constructors that we consider. We shall also use τ_0 as the basis for the construction of a type system with universes in the next section.

8. Adding Universes

One characteristic feature of Martin-Löf's type system is the cumulative hierarchy of *universes.* A *universe* of types is a type whose members are types, whose equality relation is the restriction of type equality to its members, and which is closed under the type formation operators considered in the previous section. Since it is inconsistent to introduce a universe of *all* types (which would include the universe itself), Martin-Löf instead introduces a countable hierarchy of universe U_i $(i \in \omega)$ such that U_i is included (in a suitable sense) in U_{i+1} and, for each $j < i$, U_i contains U_j as a base type. In this section we construct a model for type theory with universes.

The idea is to construct the type system τ_ω as the limit of a countable sequence $\langle \tau_i \rangle_{i \in \omega}$ of type systems, where τ_i is a type system with the first i universes as base types. The first type system, $\tau_0 = (E_0, \Phi_0)$, was already constructed in the last section. The type system τ_1 is defined by taking U_1 as a base type equal only to itself and with E_0 as member equality relation. The type system τ_1 is a proper extension of τ_0 in the sense that $\tau_0 \sqsubseteq \tau_1$. Iterating this process we obtain a chain $\tau_0 \sqsubseteq \tau_1 \cdots$ of type systems, and take τ_ω to be its

supremum. It is important to realize that we do *not* extend the language of type theory at each stage. On the contrary, the universe symbols, and terms involving them, are available from the start, and types in τ_0 may have members involving universe symbols.

These ideas may be made precise as follows. Call a type system $\nu = (E_\nu, \Phi_\nu)$ a *universe system* iff whenever $a \in |E_\nu|$, then $a \Rightarrow \mathrm{U}_i$ for some i. We define the operator $\mathbf{T}_\nu : \mathsf{TS} \to \mathsf{TS}$ similarly to the operator $\mathbf{T}$ of the last section, except that we take $E'(a, b)$ whenever $E_\nu(a, b)$, and $\Phi'(a) = \Phi_\nu(a)$ whenever the latter is defined. In the construction ν will be a type system consisting of some initial segment of the universe hierarchy.

THEOREM 8.1. *If ν is a universe system, then $\mathbf{T}_\nu$ is a monotone operator on type systems.*

PROOF. $\mathbf{T}_\nu$ is well-defined because ν is a universe system, and universes are distinct from other types. The proof that $\mathbf{T}_\nu$ is monotone is similar to that given for $\mathbf{T}$ in the last section.

Define the sequences $\langle \nu_i \rangle_{i \in \omega}$ and $\langle \tau_i \rangle_{i \in \omega}$ simultaneously as follows. At stage 0, take ν_0 to be the empty type system (which is trivially a universe system), and let $\tau_0 = (E_{\nu_0}, \Phi_{\nu_0})$ be the least fixed point of $\mathbf{T}_{\nu_0}$. The required fixed point exists by the previous theorem, and is the same as the type system τ_0 defined in the last section since $\mathbf{T}_{\nu_0} = \mathbf{T}$. At stage $i+1$, take ν_{i+1} to be the universe system $(E_{\nu_{i+1}}, \Phi_{\nu_{i+1}})$ defined by

$$E_{\nu_{i+1}} = \{(\mathrm{U}_j, \mathrm{U}_j) \mid 0 < j \leq i+1\}^*$$

and

$$\Phi_{\nu_{i+1}}(a) = E_{j-1} \quad \text{if } a \Rightarrow \mathrm{U}_j \quad \text{with } 0 < j \leq i+1.$$

Take $\tau_{i+1} = (E_{i+1}, \Phi_{i+1})$ to be the least fixed point of $\mathbf{T}_{\nu_{i+1}}$.

THEOREM 8.2. *For each $i \in \omega$,*

1. *ν_i is a universe system;*
2. *τ_i exists;*
3. *$\tau_i \sqsubseteq \tau_{i+1}$.*

PROOF. These follow easily from the definitions.

It follows that $\langle \tau_i \rangle_{i \in \omega}$ is a chain, and hence we may define $\tau_\omega = \bigsqcup_{i \in \omega} \tau_i$.

7. Judgements and Their Correctness

There are four forms of assertion, or *judgement*, in type theory: A type, $A = B$, $a \in A$, and $a = b \in A$. The first two express typehood and type equality, and the second two express membership and member equality for a type A. Let J range over the judgement forms.

A *basic* judgement is a judgement involving only saturated terms (closed terms of ground arity). We define what it means for a basic judgement J to be *correct* in a type system $\tau = (E, \Phi)$, $\tau \vDash J$, by cases on the form of J as follows:

1. $\tau \vDash A$ type iff $A \in |E|$.
2. $\tau \vDash A = B$ iff $E(A, B)$;
3. If $\tau \vDash A$ type, then $\tau \vDash a \in A$ iff $a \in |\Phi(A)|$;
4. If $\tau \vDash A$ type, then $\tau \vDash a = b \in A$ iff $\Phi(A)(a, b)$.

In the case of the membership judgements, the relation $\tau \vDash J$ is defined only under the indicated presuppositions of typehood. Here we adopt Martin-Löf's presuppositions, but note that there are alternatives (see (Allen, 1987*b*) for a thorough discussion.)

A *hypothetical* judgement is used to express a judgement about open terms. Hypothetical judgements have the form $(x_1 \in A_1, \ldots, x_n \in A_n)J$ where the free variables occurring in J are among the x_is. Roughly speaking, such a judgement expresses a kind of universal validity of J over all terms of type $A_1, \ldots, A_n$. However, the precise meaning is complicated by the fact that hypothetical judgements also express *functionality*, which means that not only must J be universally valid, but it must "respect equality" at each of the domain types. The precise meaning of "respects equality" can be given only for each individual judgement form, and hence definition of correctness for a hypothetical judgement in a type system must be given by induction on n, with a case analysis on the form of J. Furthermore, the definition is made only under presuppositions that express the sequential functionality of each of the A_is in $x_1, \ldots, x_n$.

The precise definition of correctness of a hypothetical judgement in a type system may be recovered from the following explanation for the case $n = 1$, and from Martin-Löf's account (Martin-Löf, 1982). Let $\tau = (E, \Phi)$ be a type system, and let A be such that $\tau \vDash A$ type. Define $\tau \vDash (x \in A)J$ by cases on the form of J as follows:

1. $\tau \vDash (x \in A)B(x)$ type iff for every a and b such that $\Phi(A)(a, b)$, $\tau \vDash B(a) = B(b)$.
2. If $\tau \vDash B$ type and $\tau \vDash C$ type, then $\tau \vDash (x \in A)B(x) = C(x)$ iff for every a and b such that $\Phi(A)(a, b)$, $\tau \vDash B(a) = C(b)$.
3. If $\tau \vDash (x \in A)C(x)$ type, then $\tau \vDash (x \in A)c(x) \in C(x)$ iff for every a and b such that $\Phi(A)(a, b)$, $\tau \vDash c(a) = c(b) \in C(a)$.
4. If $\tau \vDash (x \in A)c(x) \in C(x)$ and $\tau \vDash (x \in A)d(x) \in C(x)$, then $\tau \vDash (x \in A)c(x) = d(x) \in C(x)$ iff for every a and b such that $\Phi(A)(a, b)$, $\tau \vDash c(a) = c(b) \in C(a)$.

10. Proof Theory

We may now verify the soundness of some of the rules of Martin-Löf's type theory with universes in the type system τ_ω. We prove, in each case, that if

$$\frac{J_1 \cdots J_n}{J}$$

is an inference rule, and for each $1 \leq i \leq n$, J_i is correct in τ_ω, then J is correct in τ_ω as well. When presented as a system of natural deduction, such an inference rule presents only those hypotheses that are active in the inference, suppressing those that remain inert. To avoid tedious details, we ignore these inactive hypotheses in the following verifications, considering only closed rule instances. The verification for the general case follows the same pattern, but is somewhat more complicated to present.

Consider the rule of substitutivity of equality:

$$\frac{A \text{ type } a = b \in A (x \in A) B(x) \text{ type}}{B(a) = B(b)}$$

Suppose that each of the premises is correct in τ_ω, so that we have

1. $E_\omega(A, A)$;
2. $\Phi_\omega(A)(a, b)$;
3. If $\Phi_\omega(A)(a, b)$, then $E_\omega(B(a), B(b))$.

from which it immediately follows that the conclusion is correct in τ_ω.

Consider the rule of cumulativity for universes:

$$\frac{a = b \in \mathrm{U}_i}{a = b \in \mathrm{U}_{i+1}}$$

If the premise is correct in τ_ω, then $\Phi_\omega(\mathrm{U}_i)(a, b)$. But then $\Phi_\omega(\mathrm{U}_{i+1})(a, b)$ since $\Phi_\omega(\mathrm{U}_i) = E_i \subseteq E_{i+1} = \Phi_\omega(\mathrm{U}_{i+1})$.

Consider the rule of product introduction:

$$\frac{A \text{ type } (x \in A) B(x) \text{ type } (x \in A) a(x) \in B(x)}{\lambda(a) \in \Pi(A, B)}$$

If the premises are correct in τ_ω, then we have

1. $E_\omega(A, A)$;
2. If $\Phi_\omega(A)(b, c)$, then $E_\omega(B(b), B(c))$;
3. If $\Phi_\omega(A)(b, c)$, then $\Phi_\omega(B(b))(a(b), a(c))$.

It follows that $\Pi(A, B) \in |E_\omega|$. To show that $\lambda(a) \in |\Phi_\omega(\Pi(A, B))|$ it suffices to show that whenever $\Phi_\omega(A)(b, c)$, $\Phi_\omega(B(b))(a(b), a(c))$. But this is precisely the third property above.

Consider the rule of product elimination:

$$\frac{\Pi(A, B) \text{ type } b \in \Pi(A, B) \; a \in A}{\mathrm{ap}(b, a) \in B(a)}$$

For the premises to be correct in τ_ω means

1. $E_\omega(\Pi(A, B), \Pi(A, B))$;
2. $\Phi_\omega(\Pi(A, B))(b, b)$;
3. $\Phi_\omega(A)(a)$.

It follows from the definition of τ_ω and the fact that $\Phi_\omega(\Pi(A, B))$ is evaluation-respecting, that there is an f such that $b \Rightarrow \lambda(f)$, with $\lambda(f) \in |\Phi_\omega(\Pi(A, B))|$. Therefore $f(a) \in |\Phi_\omega(B(a))|$. But $\mathrm{ap}(b, a) \simeq f(a)$, and so $\mathrm{ap}(b, a) \in |\Phi_\omega(B(a))|$, as desired.

The verification of the other rules follows a very similar pattern.

I am grateful to Peter Aczel, Stuart Allen, Robert Constable, Furio Honsell, John Mitchell, Nax Mendler, and David Walker for their comments and suggestions. In particular, the general pattern of the recursion was discovered in conversation with Honsell, and many of the technical details

were worked out in discussions with Mendler. I thank Robert Constable for suggesting that the resulting account be given wider circulation.

References

Aczel, P., Carlisle, D. P. The logical theory of constructions: A formal framework and its implementation. In: (Huet, G., Plotkin, G., eds) *Proceedings of The First Workshop on Logical Frameworks*, Antibes, France (to appear).

Aczel, P. (1980). Frege structures and the notions of truth, proposition, and set. In: Keisler, J., Barwise, J., Kunen, K., eds) *The Kleene Symposium*, Studies in Logic and the Foundations of Mathematics, pp. 31–59. North-Holland.

Aczel, P. (1983). Frege structures revisited. In: (Nordström, B., Smith, J., eds) *Proceedings of the 1983 Marstrand Workshop.*

Allen, S. (1987*a*). A non-type-theoretic definition of Martin-Löf's types. In: *Second Symposium on Logic in Computer Science*, Ithaca, New York.

Allen, S. (1987*b*). *A Non-Type-Theoretic Semantics for Type-Theoretic Language.* PhD thesis, Cornell University.

Barendregt, H. P. (1984). *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* North-Holland.

Barendregt, H., Coppo, M., Dezani-Ciancaglini, M. (1983). A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic* **48**(4).

Beeson, M. (1982). Recursive models for constructive set theories. *Annals of Mathematical Logic* **23,** 127–178.

Beeson, M. J. (1985). *Foundations of Constructive Mathematics*, volume 6 of *Ergebnisse der Mathematik und ihrer Grenzgebiete.* Springer-Verlag.

Brouwer, L. E. J. (1975). *Collected Works, Vol.* 1. (Troelstra, A. S., ed.) North-Holland.

Bruce K., Meyer, A., Mitchell, J. C. (to appear). The semantics of second-order lambda calculus. *Information and Computation.*

Constable, R. L., Smith, S. F. (1987). Partial objects in constructive type theory. In: *Second Symposium on Logic in Computer Science*, pp. 183–193.

Constable, R. L., Smith, S. F. Computational foundations of basic recursive function theory. In: *Third Symposium on Logic in Computer Science*, pp. 360–371.

Constable, R. L. *et al.* (1986). *Implementing Mathematics with the NuPRL Proof Development System.* Prentice-Hall.

Coppo, M., Dezani, M. (1978). A new type assignment for lambda terms. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik* **19,** 139–156.

Coppo, M., Giovanetti, E. (1983). Completeness results for a polymorphic type system. In: (Ausiello, G., ed.) *Lecture Notes in Computer Science* **159,** 179–190.

Coppo, M., Dezani-Ciancaglini, M., Venneri, B. (1980). Principal type schemes and lambda calculus semantics. In: (Seldin, J. P., Hindley, J. R., eds) *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pp. 535–560. Academic Press.

Coquand, T., Huet, G. (1985). Constructions: A higher-order proof system for mechanizing mathematics. In: (Buchberger, B., ed.) *EUROCAL '85: European Conference on Computer Algebra. Lecture Notes in Computer Science* **203,** 151–184.

Coquand, T., Huet, G. (1988). The Calculus of Constructions. *Information and Computation* **76,** 95–120.

Coquand, T. (1986). An analysis of Girard's paradox. In: *Symposium on Logic in Computer Science*, pp. 227–236, Boston.

Curry, H. B., Hindley, J. R., Seldin, J. P. (1972). *Combinatory Logic*, Volume 2. North-Holland.

Damas, L., Milner, R. (1982). Principal type schemes for functional programs. In: *Ninth ACM Symposium on Principles of Programming Languages*, pp. 207–212.

Damas, L. M. M. (1985). *Type Assignment in Programming Languages.* PhD thesis, Edinburgh University.

della Rocca, S. R. (1987). A unification semi-algorithm for intersection type schemes. In: (Ehrig, H., Kowalski, R., Levi, G., Montanari, U., eds) *TAPSOFT '87*, pp. 37–51.

Donahue, J. E. (1979). On the semantics of data type. *SIAM Journal on Computing* **8,** 546–560.

Dummett, M. (1977). *Elements of Intuitionism.* Oxford University Press.

Giannini, P., della Rocca, S. R. (1988). Characterization of typings in polymorphic type discipline. In: *Third Symposium on Logic in Computer Science*, pp. 61–71.

Girard, J.-Y. (1972). *Interprètation Fonctionelle et Élimination des Coupures dans l'Arithmétique d'Ordre Supérieure.* PhD thesis, Université Paris VII.

Heyting, A. (1956). *Intuitionism: An Introduction.* North-Holland.

Hindley, J. R., Seldin, J. P. (1986). *Introduction to Combinators and λ-Calculus*, volume 1 of *London Mathematical Society Student Texts.* Cambridge University Press.

Hindley, J. R. (1969). The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society* **146,** 29–40.

Hindley, J. R. (1983). The completeness theorem for typing λ terms. *Theoretical Computer Science* **22,** 127–134.

Hyland, J. M. E., Pitts, A. M. (1989). The Theory of Constructions: Categorical semantics and topos-theoretic models. In: (Gray, J. W., Scedrov, A., eds) *Categories in Computer Science and Logic*, volume 92 of *Contemporary Mathematics*. American Mathematical Society.

Kleene, S. C. (1952). *Introduction to Metamathematics*. van Nostrand.

Martin-Löf, P. (1975). An intuitionistic theory of types: Predicative part. In: (Rose, H. E., Shepherdson, J. C., eds) *Logic Colloquium '73*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pp. 73–118. North-Holland.

Martin-Löf, P. (1982). Constructive mathematics and computer programming. In: *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pp. 153–175. North-Holland.

McCracken, N. (1979). *An Investigation of a Programming Language with a Polymorphic Type Structure*. PhD thesis, Syracuse University, Syracuse, New York.

Mendler, P. F., Aczel, P. (1988). The notion of a framework and a framework for LTC. In: *Third Symposium on Logic in Computer Science*, pp. 392–401, Edinburgh.

Mendler, P. (1987). *Recursive Definition in Type Theory*. PhD thesis, Cornell University.

Mendler, N. (1990). A series of type theories and their interpretation in the logical theory of constructions. In: (Huet, G., Plotkin, G., eds) *Proceedings of the First Workshop on Logical Frameworks*, Antibes, France (to appear).

Milner, R. (1978). A theory of type polymorphism in programming languages. *Journal of Computer and System Sciences* **17**, 348–375.

Mitchell, J. C. (1984). Type inference and type containment. In: (Kahn, G., MacQueen, D., Plotkin, G., eds) *Semantics of Data Types. Lecture Notes in Computer Science* **173**, 257–278.

Mitchell, J. C. (1986). A type-inference approach to reduction properties and semantics of polymorphic expressions. In: *1986 Symposium on LISP and Functional Programming*, pp. 308–319.

Nordström, B., Petersson, K., Smith, J. (1988). *Programming in Martin-Löf's Type Theory*. University of Göteborg, Göteborg, Sweden, Preprint.

Plotkin, G. (1980). Lambda-definability in the full type hierarchy. In: (Seldin, J. P., Hindley, J. R., eds) *To H. B. Curry: Essays in Combinatory Logic, Lambda Calculus and Formalism*, pp. 363–373. Academic Press.

Reynolds, J. C. (1974). Towards a theory of type structure. In: *Colloq. sur la Programmation. Lecture Notes in Computer Science* **19**, 408–423.

Smith, J. (1984). An interpretation of Martin-Löf's type theory in a type-free theory of propositions. *Journal of Symbolic Logic* **49**, 730–753.

Tait, W. W. (1967). Intensional interpretation of functionals of finite type. *Journal of Symbolic Logic* **32**, 187–199.

Troelstra, A. S. (ed.) (1973). *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis. Lecture Notes in Mathematics* **344**.