LFCS

Laboratory for Foundations of Computer Science
Department of Computer Science - University of Edinburgh

Constructing Type Systems over an Operational Semantics

Constructing Type Systems over an Operational Semantics

by

Robert Harper

ECS-LFCS-88-59

LFCS Report Series

(also published as CSR-271-88)

LFCS
Department of Computer Science
University of Edinburgh
The King's Buildings
Edinburgh EH9 3JZ

July 1988

Constructing Type Systems over an Operational Semantics

Robert Harper

1 Introduction

Research in type theory may be classified into three traditions:

- The proof-theoretic tradition. 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 [Tai67, Gir72, Mar73, CH85], 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 [Tro73, Mar73, Don79, McC79, BMM87, HP88]. Well-known examples of type theories that fall into the proof-theoretic tradtion are the simply typed λ-calculus [Bar84, HS86], the second-order λ-calculus (Girard's System F) [Rey74, Gir72], Martin-Löf's early [Mar73] (and most recent) type theories, and the Calculus of Constructions. [CH85, Coq86].

- The λ-calculus tradition. 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 [Hin69, CHS72, Mit84], intersection types [CD78], type containment [Mit84]), on characterizing the typeable terms in a given discipline [Hin69, Mil78, DM82, Dam85, CD78, CDCV80, CG83, Mit84, dR87, GdR88], and on type inference algorithms [Hin69, Mil78, DM82, dR87, GdR88]. On the semantic side, the emphasis is on characterizing the theories of certain classes of models based on untyped λ-interpretations [CDCV80, CG83, Hin83, BCDC83, Mit86].

- The computer science tradition. 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 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 most important examples of research in this direction

are Martin-Löf's type theory [Mar82], the Göteborg type theory [NPS86], and the NuPRL type theory [Con86].

Of course, such a categorization is in many ways inaccurate, emphasizing as it does only certain aspects of the research in each area. However, it doesn't seem to be an overly procrustean classification, and does serve to place the subject of this report in context.

The purpose of this note is to illustrate the construction of a small type theory in the computer science tradition. The idea 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 pedagogically useful basis for introducing some of the central ideas of Martin-Löf's type theory.

The account given here grew out of the author's attempt to understand why Stuart Allen's definition of the semantics of type theory in the NuPRL book [Con86] defined anything at all. For on close inspection, the definition does not immediately fall into any standard format for inductive definitions, and hence it is unclear at first sight whether the type and member equality relations are well-defined. Meanwhile, both Allen and Mendler at Cornell developed a rigorous account of the inductive character of the definition [All87a, All87b, Men87]. Allen's approach is based on an intuitionistically acceptable theory of inductively-defined relations, while Mendler's is a thoroughly set-theoretic account (and is essentially equivalent to ours). Two closely-related constructions are Aczel's construction of a Frege structure from a model of the untyped λ-calculus [Acz80, Acz83] and Beeson's realizability interpretation of type theory [Bee82, Bee85]. 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 (Allens' work may be viewed 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 [Smi84]. 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. However, it appears that Smith's approach cannot be extended to handle W types, or Mendler's recursive and co-recursive types.

I am grateful to Peter Aczel, Stuart Allen, Furio Honsell, John Mitchell, Nax Mendler, and David Walker for discussions about this account of type theory.

2 Preliminaries

The semantics for type theory that we shall developed 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 (p.e.r.)*. The *field* of E is defined by $|E| = \{ x \in X \mid E(x,x) \}$. It is easy to see that a p.e.r. 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 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.

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 $\bot$, and every non-empty directed subset $D \subseteq X$ has a supremum, $\bigsqcup D$, in X. A function $f : X \to Y$ between posets 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 poset 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: $x_0 = \bot$, $x_{\alpha+1} = f(x_\alpha)$, 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.

The set $X \rightharpoonup Y$ is the set of partial functions from X to Y. When working with partial functions, we adopt an (informal) logic of partial terms, writing $e \downarrow$ to mean that the expression e is defined (has a value), and $e \simeq e'$ for Kleene equality. Quantification is only over values — there are no "undefined objects."

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 simply-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 $\mathbf{0}$, 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 a component, leaving an incomplete entity that may be "applied" by filling in the hole with an entity of the appropriate kind (arity).

nLet $\mathcal{X}$ be a denumerable arity-sorted set of variables such that $\mathcal{X}_\alpha \cap \mathcal{X}_\beta = \emptyset$ whenever α is distinct from β. Let x, y, and z range over $\mathcal{X}$. Let $\mathcal{C}$ be a denumerable arity-sorted set of constants, disjoint from the variables. Let c and d range over $\mathcal{C}$. A *signature* σ is a finite subset of $\mathcal{C}$. The set of terms generated by a signature σ, $\mathcal{T}(\sigma)$, is the least arity-sorted set $\mathcal{T}$ such that $\mathcal{X} \subseteq \mathcal{T}$, $\sigma \subseteq \mathcal{T}$, $a(a_1, \ldots, a_k) \subseteq \mathcal{T}$ if $a \in \mathcal{T}_{(\alpha_1,\ldots,\alpha_k)}$ and, for $1 \leq i \leq k$, $a_i \in \mathcal{T}_{\alpha_i}$, and $x_1, \ldots, x_k.a \in \mathcal{T}_{(\alpha_1,\ldots,\alpha_k)}$ if, for $1 \leq i \leq k$, $x_i \in \mathcal{X}_{\alpha_i}$ and $a \in \mathcal{T}_{\mathbf{0}}$. The metavariables A, B, C, a, b, c, f, g range over $\mathcal{T}(\sigma)$.

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 $\mathcal{T}$ is a set of terms, then the set of *closed* terms is the subset $\mathcal{T}^0 \subseteq \mathcal{T}$ consisting of those terms a such that $\mathrm{FV}(a) = \emptyset$. A closed term of ground arity is said to be *saturated*; let $\mathcal{S} = \mathcal{T}_{\mathbf{0}}^0$ 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

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

In the sequel we work with a fixed language $\mathcal{T}$ generated by the signature appearing in Table 1. The table is divided into three columns, labeled 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 Eval mapping program expressions to values. In the present situation the language is $\mathcal{T}$, defined in the previous section, the

Canonical Type Forms		*Canonical Object Forms*		*Non-Canonical Forms*	
Form	*Arity*	*Form*	*Arity*	*Form*	*Arity*
U_i $(i \in \omega)$	$\mathbf{0}$				
nat	$\mathbf{0}$	0	$\mathbf{0}$	rec	$(\mathbf{0},\mathbf{0},(\mathbf{0},\mathbf{0}))$
		s	$(\mathbf{0})$		
I	$(\mathbf{0},\mathbf{0},\mathbf{0})$	ax	$\mathbf{0}$		
$\times$	$(\mathbf{0},\mathbf{0})$	pair	$(\mathbf{0},\mathbf{0})$	split	$(\mathbf{0},(\mathbf{0},\mathbf{0}))$
$+$	$(\mathbf{0},\mathbf{0})$	inl	$(\mathbf{0})$	case	$(\mathbf{0},(\mathbf{0}),(\mathbf{0}))$
		inr	$(\mathbf{0})$		
$\rightarrow$	$(\mathbf{0},\mathbf{0})$	λ	$((\mathbf{0}))$	ap	$(\mathbf{0},\mathbf{0})$
Π	$(\mathbf{0},(\mathbf{0}))$				
Σ	$(\mathbf{0},(\mathbf{0}))$				

Table 1: Signature of a Small Language

program expressions are the saturated terms, and the set of values $\mathcal{V}$ (ranged over by v and w) is the set of 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, and hence we may define Eval(a) to be the unique value v (if one exists) such that $a \Rightarrow v$.[1] We abuse notation by writing $a \downarrow$ for Eval(a) $\downarrow$, and $a \simeq b$ for Eval(a) $\simeq$ Eval(b).

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 as the image of the saturated terms under evaluation, rather than by a prior definition on the basis of the shape of the term. It seems plausible that the type system construction described below goes through for an arbitrary evaluator, but the author has not investigated this possibility in detail.

5 P.e.r.'s 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" aspects

[1] Strictly speaking, Eval is a partial function of the $\equiv$ class of a, and hence we should stipulate that $a \Rightarrow v$ respects $\equiv$ in both positions.

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

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

$$\mathrm{s}(M) \Rightarrow \mathrm{s}(M')$$

$$\frac{M \Rightarrow 0 \quad N \Rightarrow P}{\mathrm{rec}(M, N, F) \Rightarrow P} \qquad\qquad \frac{M \Rightarrow \mathrm{s}(M') \quad F(M', \mathrm{rec}(M', N, F)) \Rightarrow P}{\mathrm{rec}(M, N, F) \Rightarrow P}$$

$$\mathrm{I}(M, N, P) \Rightarrow \mathrm{I}(M, N, P) \qquad\qquad \mathrm{ax} \Rightarrow \mathrm{ax}$$

$$M \times N \Rightarrow M \times N \qquad\qquad \langle M, N \rangle \Rightarrow \langle M, N \rangle$$

$$\frac{M \Rightarrow \langle M', M'' \rangle \quad F(M', M'') \Rightarrow N}{\mathrm{split}(M, F) \Rightarrow N}$$

$$M + N \Rightarrow M + N \qquad\qquad \mathrm{inl}(M) \Rightarrow \mathrm{inl}(M)$$

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

$$\frac{M \Rightarrow \mathrm{inl}(M') \quad F(M') \Rightarrow N}{\mathrm{case}(M, F, G) \Rightarrow N} \qquad\qquad \frac{M \Rightarrow \mathrm{inr}(M') \quad G(M') \Rightarrow N}{\mathrm{case}(M, F, G) \Rightarrow N}$$

$$M \to N \Rightarrow M \to N \qquad\qquad \lambda(F) \Rightarrow \lambda(F)$$

$$\frac{M \Rightarrow \lambda(F) \quad F(N) \Rightarrow P}{\mathrm{ap}(M, N) \Rightarrow P}$$

$$\Pi(M, F) \Rightarrow \Pi(M, F) \qquad\qquad \Sigma(M, F) \Rightarrow \Sigma(M, F)$$

Figure 1: Operational Semantics

which is conveniently captured by using partial equivalence relations.

The p.e.r.'s that we shall consider are over the set $\mathcal{S}$ of saturated terms. It might seem at first sight that we could first consider p.e.r.'s over the subset $\mathcal{V} \subseteq \mathcal{S}$ of values, then extend to all saturated terms "at the end." But since the evaluator that we are considering is not compositional (and it is hard to see how one could be made to be so in the presence of binding operators), we are forced to "interleave" evaluation within the definition of the p.e.r.'s representing type and member equality. One way to view the construction to follow is as a way of providing a compositional way of reasoning about the non-compositional evaluator; interlacing evaluation with the definition of the relations is crucial to achieving this end.

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 $A \rightarrow B$ 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 [CS87, CS88]. The formal realization of this aspect of type theory is the notion of a "value-respecting" p.e.r.

Let PER denote the set of partial equivalence relations on the set of saturated terms $\mathcal{S}$. Hereafter, we use "p.e.r." to refer only to elements of PER. A p.e.r. 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 be the set of all value-respecting p.e.r.'s (v.p.e.r.'s). If $\Phi : \mathcal{S} \rightarrow \mathsf{VPER}$, then we extend Φ to closed terms of arity $(\mathbf{0})$ by taking $\Phi(f)(a) = \Phi(fa)$.

A v.p.e.r. E is determined by its behavior on values: $E(a,b)$ iff there exists $v, w \in V$ such that $a \Rightarrow v$, $b \Rightarrow w$, and $E(v,w)$. Therefore one way to define a v.p.e.r. is to first define a p.e.r. E on $\mathcal{V}$, then extend to a v.p.e.r. E^* by pre-evaluation: $E^*(a,b)$ iff there exists v and w such that $a \Rightarrow v$, $b \Rightarrow w$, and $E(v,w)$. Note that E^* is the unique v.p.e.r. agreeing with E on values. (This is essentially Martin-Löf's method of defining types by defining their canonical members, then extending membership to all saturated terms that evaluate to canonical members.)

PER forms a cppo under the ordering $E \sqsubseteq E'$ iff for all $a \in |E|$, $E[a] = E'[a]$. It is easy to see that this defines a partial ordering, with the empty relation as least element. Let $\mathcal{D}$ be a directed set of p.e.r.'s. The supremum $\bigsqcup \mathcal{D}$ of $\mathcal{D}$ is given by the p.e.r. D such that $a \in |D|$ iff $a \in |E|$ for some $E \in \mathcal{D}$, in which case $D[a]$ is defined to be $E[a]$. This is well-defined since $\mathcal{D}$ is directed: if $a \in |E'|$ for some other $E' \in \mathcal{D}$, then $E[a] = E'[a]$ since they have an upper bound in $\mathcal{D}$. 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 v.p.e.r.'s is itself a v.p.e.r.

The following operations on v.p.e.r.'s, inspired by Plotkin's "logical relations" [Plo80],

are used in the construction of type systems.

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

Note the use of the "evaluation closure" operation on the result of each operation. The effect of the closure may be illustrated by observing that, for example, $I(a,b,E)(c,d)$ iff $c \Rightarrow \mathrm{ax}$, $d \Rightarrow \mathrm{ax}$, and $E(a,b)$. 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 v.p.e.r. and $\Phi : \mathcal{S}/E \to \mathsf{VPER}$ is a function assigning a v.p.e.r. to each $a \in |E|$ in such a way that if $E(a,b)$, then $\Phi(a) = \Phi(b)$. The relation E is the type equality relation for τ, and, for each $a \in |E|$, the relation $\Phi(a)$ is the member equality for type a in type system τ. Let $\mathsf{TS} = \Sigma_{E \in \mathsf{VPER}}(\mathcal{S}/E \to \mathsf{VPER})$ be the set of type systems.

TS forms a cppo under the ordering

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

Informally, τ is less than τ', iff τ' has at least as many types as τ, and they agree on the equality relation assigned to the types that they have in common. It is easy to see that TS is partially ordered by this relation, with $(\emptyset,\emptyset)$ as least element. Let $\mathcal{D}\langle (E_i,\Phi_i)\rangle_{i\in I}$ be a directed set of type systems and let $\tau_i = (E_i,\Phi_i)$ and $\tau_j = (E_j,\Phi_j)$ be two elements of $\mathcal{D}$. Therefore, the supremum of $\mathcal{D}$ is given by $\bigsqcup\mathcal{D} = (E,\Phi)$, where $E = \bigsqcup_{i\in I} E_i$, and $\Phi(a)$ is defined iff $\Phi_i(a)$ is defined for some $i \in I$, in which case $\Phi(a) = \Phi_i(a)$. (That this is well-defined follows from the fact that $\mathcal{D}$ is directed: if $a \in |E_i| \cap |E_j|$, then $\Phi_i(a) = \Phi_j(a)$, since τ_i and τ_j have an upper bound in $\mathcal{D}$.)

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, a cppo. 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. The crucial point is that whenever a type is introduced, its membership equality relation is defined and remains fixed for all future stages. Were this not the case, the construction process would not be monotone, and we would not be guaranteed to reach a fixed point.

To make these ideas precise, we define an operator $\mathbf{T} : \mathsf{TS} \to \mathsf{TS}$ by $\mathbf{T}(E, \Phi) = (F^*, \Psi)$, where

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

and

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

Theorem 7.1

1. $\mathbf{T}$ *is monotone.*

2. $\mathbf{T}$ *is not continuous.*

Proof *Let* $\tau_1 = (E_1, \Phi_1)$ *and let* $\tau_2 = (E_2, \Phi_2)$ *be type systems such that* $\tau_1 \sqsubseteq \tau_2$. *Let* $\tau_1' = (E_1', \Phi_1') = \mathbf{T}(\tau_1)$ *and* $\tau_2' = (E_2', \Phi_2') = \mathbf{T}(\tau_2)$. *We are to show that* $\tau_1' \sqsubseteq \tau_2'$, *i.e., that* $E_1' \sqsubseteq E_2'$ *and for every* $a \in |E_1'|$, $\Phi_1'(a) = \Phi_2'(a)$. *Let* $a \in |E_1'|$. *Then* a *must evaluate to some value* v *such that* $v \in |E_1'|$. *Consider the case* $v \equiv b \times c$. *Then since* $v \in |E_1'|$, *we must have* $b \in |E_1|$ *and* $c \in |E_1|$. *But then* $b \in |E_2|$ *and* $c \in |E_2|$ *since* $E_1 \sqsubseteq E_2$,

and so $v \in |E_2'|$ as well, and therefore $a \in |E_2'|$. By similar reasoning, $E_1'[a] = E_2'[a]$ for all $a \in |E_1|$, and so $E_1' \sqsubseteq E_2'$. To see that $\Phi_1'(a) = \Phi_2'(a)$, we consider the unique v such that $a \Rightarrow v$, and proceed by case analysis on the form of v. For $v \equiv b \times c$, $\Phi_1'(a) = \Phi_1(b) \times \Phi_1(c)$, but this is just $\Phi_2(b) \times \Phi_2(c) = \Phi_2'(b \times c) = \Phi_2'(a)$.

To see that continuity fails, consider the term $a \equiv \Pi(N, x.\,\mathrm{rec}(n; N; u, v.N \times v))$. At each finite stage i, only the type $N \times \cdots \times N$ (i times) exists, and hence a exists only at stage ω, and therefore more types exist at subsequent stages. □

Let τ_0 be the least fixed point of $\mathbf{T}$. 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 $F(a, b)$ whenever $E_\nu(a, b)$, and $\Phi(a) = \Phi_\nu(a)$ whenever the latter is defined. It is important that ν be a universe system here, for otherwise this may not be well-defined.

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 since the definition of $\mathbf{T}$ makes no reference to universes and ν is a universe system. The verification that it is monotone is similar to that for Ty 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 τ_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

1. $E_{\nu_{i+1}} = \{ (\mathrm{U}_j, \mathrm{U}_j) \mid 0 < j \leq i+1 \}^*$;
2. For each $0 < j \leq i+1$, $\Phi_{\nu_{i+1}}(\mathrm{U}_j) = E_{j-1}$.

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 $\tau_\omega = \bigsqcup_{i\in\omega} \tau_i$.

9 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 $\tau = (E, \Phi)$, $\tau \models J$, by cases on the form of J as follows:

- $\tau \models A$ type iff $E(A, A)$;
- $\tau \models A = B$ iff $E(A, B)$;
- $\tau \models a \in A$ (where $\tau \models A$ type) iff $\Phi(A)(a, a)$;
- $\tau \models a = b \in A$ (where $\tau \models A$ type) iff $\Phi(A)(a, b)$.

In the case of the membership judgements, the relation $\tau \models J$ is defined only under the indicated presuppositions of typehood. Here we adopt Martin-Löf's presuppositions, but note that there are alternatives (see [All87b] 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_i's. 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 first by cases on the form of J, and then by induction on n. Furthermore, the definition is made only under presuppositions that express the sequential functionality of each of the A_i's 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 [Mar82]. Let $\tau = (E, \Phi)$ be a type system, and let A be such that $\tau \models A$ type. Define $\tau \models (x \in A)\ J$ as follows:

- $\tau \models (x \in A)\ B(x)$ type iff for every a and b such that $\Phi(A)(a, b)$, $\tau \models B(a) = B(b)$.[2]
- $\tau \models (x \in A)\ B(x)) = C(x)$ (where $\tau \models B$ type and $\tau \models C$ type) iff for every a and b such that $\Phi(A)(a, b)$, $\tau \models B(a) = C(b)$.
- $\tau \models (x \in A)\ c(x) \in C(x)$ (where $\tau \models (x \in A)\ C(x)$ type) iff for every a and b such that $\Phi(A)(a, b)$, $\tau \models c(a) = c(b) \in C(a)$.
- $\tau \models (x \in A)\ c(x) = d(x) \in C(x)$ (where $\tau \models (x \in A)\ c(x) \in C(x)$ and $\tau \models (x \in A)\ d(x) \in C(x)$) iff for every a and b such that $\Phi(A)(a, b)$, $\tau \models c(a) = c(b) \in C(a)$.

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

$$\frac{J_1 \quad \ldots \quad J_n}{J}$$

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. In reconstructing the suppressed premises of the inference rules, we have preferred to err on the side of conservativity since we are not concerned here with minimality or convenience.

[2]Note that $B(x)$, $B(a)$, and $B(b)$ are instances of application in the arity system!

Consider the rule of substitutivity of equality:

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

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 $\Phi_\omega(\mathrm{U}_{i+1})$ since $\Phi_\omega(\mathrm{U}_i) = E_i \sqsubseteq E_{i+1} = \Phi_\omega(\mathrm{U}_{i+1})$.

Consider the rule of product introduction:

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

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:

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

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 that fact that $\Phi_\omega(\Pi(A, B))$ is value-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.

References

[Acz80] Peter Aczel. Frege structures and the notions of truth, proposition, and set. In J. Keisler, J. Barwise, and K. Kunen, editors, *The Kleene Symposium*, pages 31–59. North Holland, 1980.

[Acz83] Peter Aczel. Frege structures revisited. In B. Nordström and J. Smith, editors, *Proceedings of the 1983 Marstrand Workshop*, 1983.

[All87a] Stuart Allen. A non–type–theoretic definition of Martin-Löf's types. In *Proceedings of the Symposium on Logic in Computer Science*, Ithaca, New York, June 1987.

[All87b] Stuart Allen. *A Non-Type-Theoretic Semantics for Type-Theoretic Language.* PhD thesis, Cornell University, 1987.

[Bar84] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*, volume 103 of *Studies in Logic and the Foundations of Mathematics.* North-Holland, Amsterdam, revised edition, 1984.

[BCDC83] H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4), 1983.

[Bee82] Michael Beeson. Recursive models for constructive set theories. *Annals of Mathematical Logic*, 23:127–178, 1982.

[Bee85] Michael J. Beeson. *Foundations of Constructive Mathematics*, volume 6 of *Ergebnisse der Mathematik und ihrer Grenzgebiete.* Springer-Verlag, Heidelberg, 1985.

[BMM87] Kim Bruce, Albert Meyer, and John C. Mitchell. The semantics of second-order lambda calculus. *Information and Computation*, 1987. To appear.

[CD78] Mario Coppo and Mariangiola Dezani. A new type assignment for lambda terms. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 19:139–156, 1978.

[CDCV80] M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Principal type schemes and lambda calculus semantics. In *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 535–560, New York, 1980. Academic Press.

[CG83] M. Coppo and E. Giovanetti. Completeness results for a polymorphic type system. In G. Ausiello, editor, *CAAP '83*, volume 159 of *Lecture Notes in Computer Science*, pages 179–190. Springer-Verlag, 1983.

[CH85] Thierry Coquand and Gérard Huet. Constructions: A higher–order proof system for mechanizing mathematics. In B. Buchberger, editor, *EUROCAL '85: European Conference on Computer Algebra*, volume 203 of *Lecture Notes in Computer Science*, pages 151–184. Springer-Verlag, 1985.

[CHS72] H. B. Curry, J. R. Hindley, and J. P. Seldin. *Combinatory Logic, Volume 2.* North-Holland, Amsterdam, 1972.

[Con86] Robert L. Constable, *et. al. Implementing Mathematics with the NuPRL Proof Development System.* Prentice–Hall, Englewood Cliffs, NJ, 1986.

[Coq86] Thierry Coquand. An analysis of Girard's paradox. In *Proc. of the Symposium on Logic in Computer Science*, pages 227–236, Boston, June 1986.

[CS87] Robert L. Constable and Scott Fraser Smith. Partial objects in constructive type theory. In *Proceedings of the Second Annual Symposium on Logic in Computer Science*, pages 183–193, June 1987.

[CS88] Robert L. Constable and Scott Fraser Smith. Computational foundations of basic recursive function theory. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 360–371, July 1988.

[Dam85] Luis Manuel Martins Damas. *Type Assignment in Programming Languages.* PhD thesis, Edinburgh University, 1985.

[DM82] Luis Damas and Robin Milner. Principal type schemes for functional programs. In *Proceedings of the 9th ACM Symposium on the Principles of Programming Languages*, pages 207–212, 1982.

[Don79] James E. Donahue. On the semantics of data type. *SIAM Journal on Computing*, 8:546–560, 1979.

[dR87] Simona Ronchi della Rocca. A unification semi-algorithm for intersection type schemes. In Hartmut Ehrig, Robert Kowalski, Giorgio Levi, and Ugo Montanari, editors, *TAPSOFT '87*, pages 37–51, March 1987.

[GdR88] Paola Giannini and Simona Ronchi della Rocca. Characterization of typings in polymorphic type discipline. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 61–71, July 1988.

[Gir72] Jean-Yves Girard. *Interprètation fonctionelle et élimination des coupures dans l'arithmètique d'ordre supérieure.* PhD thesis, Université Paris VII, 1972.

[Hin69] J. R. Hindley. The principal type scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–40, 1969.

[Hin83] J. R. Hindley. The completeness theorem for typing λ terms. *Theoretical Computer Science*, 22:127–134, 1983.

[HP88] J. Martin E. Hyland and Andrew M. Pitts. The theory of constructions: Categorical semantics and topos-theoretic models. In *Proceedings of the Boulder Conference on Categories in Computer Science*, 1988. To appear.

[HS86] J. Roger Hindley and Jonathan P. Seldin. *Introduction to Combinators and λ-Calculus*, volume 1 of *London Mathematical Society Student Texts*. Cambridge University Press, 1986.

[Mar73] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Shepherdson, editors, *Logic Colloquium, '73*, pages 73–118, Amsterdam, 1973. North-Holland.

[Mar82] Per Martin-Löf. Constructive mathematics and computer programming. In *Sixth International Congress for Logic, Methodology, and Philosophy of Science*, pages 153–175, Amsterdam, 1982. North-Holland.

[McC79] Nancy McCracken. *An Investigation of a Programming Language with a Polymorphic Type Structure*. PhD thesis, Syracuse University, Syracuse, New York, 1979.

[Men87] Paul Mendler. *Recursive Definition in Type Theory*. PhD thesis, Cornell University, 1987.

[Mil78] Robin Milner. A theory of type polymorphism in programming languages. *Journal of Computer and System Sciences*, 17:348–375, 1978.

[Mit84] John C. Mitchell. Type inference and type containment. In G. Kahn, D. MacQueen, and G. Plotkin, editors, *Semantics of Data Types*, volume 173 of *Lecture Notes in Computer Science*, pages 257–278. Springer-Verlag, 1984.

[Mit86] John C. Mitchell. A type–inference approach to reduction properties and semantics of polymorphic expressions. In *ACM Conference on LISP and Functional Programming*, pages 308–319, August 1986.

[NPS86] Bengt Nordström, Kent Petersson, and Jan Smith. *An Introduction to Martin-Löf's Type Theory*. University of Göteborg, Göteborg, Sweden, 1986. Preprint.

[Plo80] Gordon Plotkin. Lambda-definability in the full type hierarchy. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 363–373, New York, 1980. Academic Press.

[Rey74] John C. Reynolds. Towards a theory of type structure. In *Colloq. sur la Programmation*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer-Verlag, 1974.

[Smi84] Jan Smith. An interpretation of Martin-Löf's type theory in a type-free theory of propositions. *Journal of Symbolic Logic*, 49(3):730–753, September 1984.

[Tai67] William W. Tait. Intensional interpretation of functionals of finite type. *Journal of Symbolic Logic*, 32(2):187–199, June 1967.

[Tro73] Ann S. Troelstra, editor. *Metamathematical Investigation of Intuitionistic Arithmetic and Analysis*, volume 344 of *Lectures Notes in Mathematics*. Springer-Verlag, Heidelberg, 1973.