A Non-Type-Theoretic Definition of Martin-Löf's Types*

Stuart Allen

87-832

April 1987

Department of Computer Science
Cornell University
Ithaca, New York 14853-7501

*This is a modified version of a paper that is to appear in *The Proceedings of the Second Annual Symposium on Logic in Computer Science*, IEEE, 1987.

A Non-Type-Theoretic Definition of Martin-Löf's Types*

Stuart Allen
Computer Science Dept.
Cornell University

April 27, 1987

Abstract

It is possible to make a natural non-type-theoretic reinterpretation of Martin-Löf's type theory. This paper presents an inductive definition of the types explicitly defined in Martin-Löf's paper, *Constructive Mathematics and Computer Programming.* The definition is set-theoretically valid, and probably will be convincing to intuitionists as well. When this definition is used with methods set out in the author's thesis, the inference rules presented in Martin-Löf's paper can be shown to be valid under the non-type-theoretic interpretation. This interpretation is non-trivial, that is, there are both inhabited types and empty types, and so, validity entails simple consistency. Finally, Michael Beeson has defined some recursive realizability models which we shall compare with the term model presented here, and we shall compare the methods of definition.

---

*This is a modified version of a paper that is to appear in *The Proceedings of the Second Annual Symposium on Logic in Computer Science*, IEEE, 1987.

Introduction

There is some interest among computer scientists in using Martin-Löf's intuitionistic theory of types [Martin-Löf] as a basis for programming [Constable *et al.*, Nordstrom, Petersson & Smith]. The work presented in this paper is a piece of a larger effort (see [Allen]) to provide a non-type-theoretic semantics for the expressive machinery of Martin-Löf's theory. The primary goal of this reinterpretation is to make most of the practice of mathematics that is based on intuitionistic type theory available to those who do not subscribe to the theory itself.

The reinterpretation of the theory of [Martin-Löf] proceeds in three phases. First, instead of using terms constructed from an open-ended body of term constructors, we use a fixed class of terms and an evaluation relation. In [Martin-Löf], a term constructor is introduced by specifying its argument places and variable binding structure, and also how one evaluates the terms with that constructor outermost. Second, instead of an open-ended body of type definitions, we use a property of terms T **type**, which we may call *typehood*, and a three-place relation on terms $t = s \in T$, which we may call *member equality*. In [Martin-Löf], a type is *defined* by choosing a canonical term T, the type, then specifying which canonical terms are to be *members* of T, and then specifying an equivalence relation on the canonical members of T as the *equality* on T. Finally, we reinterpret the judgements of [Martin-Löf] purely in terms of the property typehood, the relation member equality, and the syntax of the forms of judgement. The syntax of the judgement forms may be inductively defined from terms.

Is this reinterpretation possible for a "real" system of types, such as is defined in [Martin-Löf]? Clearly, we can define by induction the terms built using the term constructors actually introduced in [Martin-Löf], and we can define by induction the evaluation relation between those terms. It should also be easy for those familiar with [Martin-Löf] to believe that, given the property T **type** and the relation $t = s \in T$, the judgements of [Martin-Löf] may be reinterpreted appropriately. The issue then is whether the property T **type** and the relation $t = s \in T$ can be defined so that they capture the types defined in [Martin-Löf][1].

[1] or rather, the types that are definable using only the type constructors defined in [Martin-Löf]. I think we may consider a type constructor definition to be a procedure for

Anatomy of the Type Constructors

The canonical type constructors that are defined in [Martin-Löf] are conveniently divisible into three groups. The constituent types of a type are those upon whose definition the type construction depends.

- N_n N $\mathrm{I}(A,a,b)$ $A+B$. Each of these defines a type from finitely many, explicitly given, constituent types.
- $(\Sigma x\in A)B$ $(\Pi x\in A)B$ $(\mathrm{W}x\in A)B$. Each of these defines a type from an explicitly given family of constituent types that is indexed by an explicitly given constituent type.
- U_n. The equal members of universe U_n are the extensionally equal types[2] that are definable from the type constructors other than U_{n+i}. Although we might have included universes in the first group as having no constituent types, it seems clearer to conceive of them as resulting from a second order type constructor that builds a universe from a level number and a collection of ordinary first order type constructors.

Suppose we were to define T **type** and $t = s \in T$ by mutual recursion of some sort. What might the clauses defining the canonical types be? Let $s \leftarrow t$ mean that t is a closed term that evaluates to s. The defining clauses for the first group of constructors are simple, those for $A+B$ being

$$A+B \textsf{ type iff } A \textsf{ type} \;\&\; B \textsf{ type}$$

and

$$\begin{aligned} t = t' \in A+B \text{ iff } A+B \textsf{ type} \;\&\; \exists\, a\, a'.\ \mathrm{i}(a)\leftarrow t \;\&\; \mathrm{i}(a')\leftarrow t' \;\&\; a = a' \in A \\ \text{or } \exists\, b\, b'.\ \mathrm{j}(b)\leftarrow t \;\&\; \mathrm{j}(b')\leftarrow t' \;\&\; b = b' \in B, \end{aligned}$$

where $\mathrm{i}(t)$ and $\mathrm{j}(t)$ are canonical term constructors used for injection.

The second group of type constructors requires a certain kind of specification for families of types, namely, a term B_x and a type A over which B_x

defining a type from its constituent types, the procedure being applicable to types not yet defined.

[2]Types are extensionally equal when they have the same membership and the same equality between members.

is *type functional*. We may say that a term B is *type functional over A in x* when substitution of equal members of A for x in B results in extensionally equal types. Suppose we included a clause defining $T = S$ as extensional type equality.

$$T = S \text{ iff } T \ \textsf{type} \ \& \ S \ \textsf{type} \ \& \ \forall\, t\, s.\ t = s \in T \text{ iff } t = s \in S.$$

Then the clauses defining $(\Pi x \in A)B$, typical of the second group, might be

$$(\Pi x \in A)B \ \textsf{type} \text{ iff } A \ \textsf{type} \ \& \ \forall\, a\, a'.\ B[a/x] = B[a'/x] \text{ if } a = a' \in A$$

and

$$\begin{aligned} t = t' \in (\Pi x \in A)B \text{ iff } & (\Pi x \in A)B \ \textsf{type} \\ & \& \ \exists\, u\, b\, u'\, b'.\ (\lambda u)b \leftarrow t \ \& \ (\lambda u')b' \leftarrow t' \\ & \qquad \& \ \forall\, a\, a'.\ b[a/u] = b'[a'/u'] \in B[a'/x] \\ & \qquad\qquad \text{if } a = a' \in A. \end{aligned}$$

The clause admitting U_n as a type would be simply

$$\mathrm{U}_n \ \textsf{type}.$$

To define equality on U_n, we need only define the membership of U_n, and then define the equality to be extensional type equality on the members.

$$t \in T \text{ iff } t = t \in T.$$

$$T = S \in \mathrm{U}_n \text{ iff } T \in \mathrm{U}_n \ \& \ S \in \mathrm{U}_n \ \& \ T = S.$$

To define the membership of U_n, we might duplicate the clauses defining typehood, except that instead of the clause U_n **type** we would use

$$\mathrm{U}_m \in \mathrm{U}_n \text{ iff } m < n,$$

and in the other typehood defining clauses we would replace T **type** by $T \in \mathrm{U}_n$.

Finally, in addition to the clauses defining the other canonical types, we would include clauses defining the non-canonical types.

$$T \ \textsf{type} \text{ iff } \exists S \leftarrow T.\ S \ \textsf{type}.$$
$$t = s \in T \text{ iff } \exists S \leftarrow T.\ t = s \in S.$$

While a putative definition of T **type** and $t = s \in T$ having the form just outlined may well serve as a convincing definition when properly read, the reading is not as straightforward as possible. If one inspects the right-hand sides of the clauses, one finds various negative, as well as positive, occurrences of the three-place relation $t = s \in T$;[3] there are no strongest T **type** and $t = s \in T$ satisfying those clauses. Our aim will be to express more explicitly, by means of an appropriate monotonic operator, what would be meant by the recursive definition sketched above.

Extensional Type Systems

The method we shall use for defining type systems is more easily applied to type systems with extensional type equality than to those with intensional equality. Since the only type systems to be defined here are extensional, we shall restrict our consideration to such.

By $s \leftarrow t$, let us mean that term t evaluates to term s. For our purposes, what is important about evaluation is that

if $s \leftarrow t$ then t is closed & $s \leftarrow s$

and

if $s \leftarrow t$ & $s' \leftarrow t$ then s is s',

that is, $s \leftarrow t$ defines a partial function in t. A *canonical* term is one that is its own value.

Let us say that a *possible type system* is (or is represented by) a relation $\tau T \phi$ between terms (T) and two-place relations on terms (ϕ). The intention is that when τ is a type system, $\tau T \phi$ is true just when T is a type and ϕ is the equality between members of T.[4] Let us define four relations which should make clear the intended use of type systems.

[3] For example, the right-hand sides of the Π type formation and equality clauses contain explicitly negative occurrences of $() = () \in ()$. Wherever $() = ()$ occurs, we may say that $() = () \in ()$ implicitly occurs both positively and negatively. Occurrences of $() = ()$ are to be found in the clauses for Π type formation and U_n equality.

[4] If we wanted to include type systems with intensional type equality, we would characterize type systems by three-place relations $\tau T T' \phi$, such that $\tau T T' \phi$ holds when T and T' are equal types with member equality ϕ.

$T =_\tau S$ iff $\exists\phi.\,\tau T\phi$ & $\tau S\phi$.

T type_τ iff $T =_\tau T$.

$t = s \in_\tau T$ iff $\exists\phi.\,\tau T\phi$ & $t\phi s$.

$t \in_\tau T$ iff $t = t \in_\tau T$.

A(n extensional) *type system* has certain properties which we now define.

Fun(τ) iff $\forall\, T\ \phi\ \phi'.\ \phi$ is ϕ' if $\tau T\phi$ & $\tau T\phi'$,

that is, $\tau T\phi$ defines a partial (relation valued) function in T.

TyVal(τ) iff $\forall\, T\ \phi.\ \tau T\phi$ iff $\exists T'{\leftarrow}T.\,\tau T'\phi$,

that is, to be a type is to evaluate to a(n equal) type.

TrSy(τ) iff $\forall\, T\ \phi.\ \phi$ is transitive and symmetric if $\tau T\phi$,

thus, member equality is an equivalence relation on members.

Val(τ) iff $\forall\, T\ \phi.$ if $\tau T\phi$ then $\forall\, t\ s.\ t\phi s$ iff $\exists t'{\leftarrow}t.\,t'\phi s$,

and so, in view of TrSy(τ), to be a member is to evaluate to a(n equal) member. A possible type system τ is a type system, or ETS(τ), just when it has these four properties.

This characterization of type systems has been chosen because our central method of definition will be inductive definition of particular relations of the kind we have called possible type systems.

The Inductive Definition

We shall define our principal type system as the strongest possible type system closed under the type constructors. Defining a universe U_n requires selecting the strongest possible type system closed under the type constructors other than U_{n+i}. These definitions might be treated uniformly if we had a method for defining the strongest possible type system closed under a given collection of universe constructors as well as under the non-universe type constructors.

Defined below is a function TyF which is a two-place operator on possible type systems. For possible type systems σ and τ, $\mathrm{TyF}(\sigma;\tau)T\phi$ holds when type T (with equality ϕ) is a type of σ (with equality ϕ) or else it is constructed from the types of τ by a non-universe constructor. The parameter σ provides a means of including base types, and by this means we will introduce universes.

We will define a one-place possible type system operator $\mu(\sigma)$ such that $\mu(\sigma)$ is the strongest possible type system closed under $\mathrm{TyF}(\sigma;)$. But is such a definition valid? That depends upon TyF and upon the reader's understanding of inductive definition. There are three kinds of readers whom I would address. Those who subscribe to set theory, those whose ontology includes extensional properties and relations, and intuitionists or other constructivists.

Set theoretically, the definition is valid if $\mathrm{TyF}(\sigma;)$ may be construed as a monotonic operator on the subsets of some set. Clearly, the terms can be represented as members of a set $\mathcal{T}$. If $\mathrm{TyF}(\sigma;)$ is monotonic on the subsets of $\mathcal{T} \times Pow(\mathcal{T} \times \mathcal{T})$ then

$$\mu(\sigma) \text{ is } \bigcap \{\tau \subseteq \mathcal{T} \times Pow(\mathcal{T} \times \mathcal{T}) \mid \mathrm{TyF}(\sigma;\tau) \subseteq \tau\}.$$

If relations are essentially extensional,[5] and if they are ontologically real,[6] then the definition is valid if $\mathrm{TyF}(\sigma;\tau)T\phi$ is monotonic in τ, since μ is explicitly definable by

$$\mu(\sigma)T\phi \text{ iff } \forall\tau.\ \tau T\phi \text{ if } \forall\, S\,\psi.\ \tau S\psi \text{ if } \mathrm{TyF}(\sigma;\tau)S\psi.$$

Standard intuitionistic theory of inductive definition directly licenses inductive definitions of

the strongest P such that $\forall\overline{x}.\ P\overline{x}$ if $\theta(\overline{x};P)$,

where $\theta(\overline{x};P)$ is a relation between individuals and properties of individuals that is (strictly) positive in P. The definition of $\mu(\sigma)$ does not quite

[5]as opposed to extensional equality between relations being a relation between intensional relations,

[6]rather than *façons de parler*,

conform to this standard since it is not a relation between individuals, but rather, for each σ, $\mu(\sigma)$ is a relation between individuals and two-place relations between individuals. Still, the intuitionist might be convinced of the validity of our definition if $\mathrm{TyF}(\sigma;\tau)T\phi$ is strictly positive in τ.

It will be seen that TyF does indeed meet these various criteria for the validity of the definition of μ.

We begin the definition by setting out the type formation methods. This is done by defining the relations $\widehat{\mathrm{N}}_?$, $\widehat{\mathrm{N}}$, $\widehat{\mathrm{I}}$, $\widehat{+}$, $\widehat{\Sigma}$, $\widehat{\Pi}$ and $\widehat{\mathrm{W}}$. Each of these is an operator on possible type systems,[7] whose value (a possible type system) has only the types that evaluate to a certain form.

The types N_n and N have no constituent types.

$$\widehat{\mathrm{N}}_?T\phi \text{ iff } \exists n.\ \mathrm{N}_n{\leftarrow}T \ \&\ \forall\, a\, b.\ a\phi b \text{ iff } \exists m < n.\ m_n{\leftarrow}a, b.$$

Define N-equality by

$$\begin{array}{l}\text{Neq is the strongest } \phi \text{ such that}\\ \forall\, a\, b.\ a\phi b \text{ if } 0{\leftarrow}a, b \text{ or } \exists\, a'\, b'.\ \mathrm{suc}(a'){\leftarrow}a \ \&\ \mathrm{suc}(b'){\leftarrow}b \ \&\ a'\phi\, b'.\end{array}$$

$$\widehat{\mathrm{N}}T\phi \text{ iff } \mathrm{N}{\leftarrow}T \ \&\ \phi \text{ is Neq.}$$

The rest of the non-universe type constructors have constituent types, and so the type formation operators need, as a parameter, a possible type system from which to get these constituent types. In the definitions of $\widehat{\mathrm{I}}$ and $\widehat{+}$, α and β range over two-place relations on terms.

$$\begin{array}{l}\widehat{\mathrm{I}}(\tau)T\phi \text{ iff } \exists\, A\, \alpha\, a\, b.\ \mathrm{I}(A,a,b){\leftarrow}T \ \&\ \tau A\alpha \ \&\ a\alpha a \ \&\ b\alpha b\\ \qquad\qquad \&\ \forall\, t\, t'.\ t\phi t' \text{ iff } \mathrm{r}{\leftarrow}t, t' \ \&\ a\alpha b.\end{array}$$

$$\begin{array}{l}\widehat{+}(\tau)T\phi \text{ iff } \exists\, A\, \alpha\, B\, \beta.\ A{+}B{\leftarrow}T \ \&\ \tau A\alpha \ \&\ \tau B\beta\\ \qquad\qquad \&\ \forall\, t\, t'.\ t\phi t' \text{ iff } \exists\, a\, a'.\ \mathrm{i}(a){\leftarrow}t \ \&\ \mathrm{i}(a'){\leftarrow}t' \ \&\ a\alpha a'\\ \qquad\qquad\qquad \text{or } \exists\, b\, b'.\ \mathrm{j}(b){\leftarrow}t \ \&\ \mathrm{j}(b'){\leftarrow}t' \ \&\ b\beta b'.\end{array}$$

Now we proceed with the second group of type constructors, those having families of constituent types. In the definitions below, α ranges over two-place relations between terms and γ ranges over three-place relations between terms. The application of γ to terms is indicated by $t\gamma_a s$.

[7] We may consider possible type systems to be zero-place operators.

$$\begin{aligned}\mathrm{Fam}(\tau;A;\alpha;x;B;\gamma) \text{ iff } \tau A\alpha \ \&\ \forall\, a\, a'.\ \text{if } a\alpha a' \text{ then } & \gamma_a \text{ is } \gamma_{a'}\\ & \&\ \tau\, B[a/x]\, \gamma_a\\ & \&\ \tau\, B[a'/x]\, \gamma_{a'}.\end{aligned}$$

When τ is a type system, i.e., when $\mathrm{ETS}(\tau)$, then B is type functional over A in x if and only if there are α and γ such that $\mathrm{Fam}(\tau;A;\alpha;x;B;\gamma)$. The definition of the type functionality in the previous section is easily seen to identify the type functionality of B over A in x with

$$\mathrm{Fam}(\tau;A;\ a,a'.(a = a' \in_\tau A)\,;x;B;\ a,b,b'.(b = b' \in_\tau B[a/x])\,).$$

In a type system, the type determines a unique member equality, so, we can eliminate the occurrences of member equality by dropping all reference to it, thus:

$$\exists\, \alpha\, \gamma.\ \mathrm{Fam}(\tau;A;\alpha;x;B;\gamma).$$

Note that $\mathrm{Fam}(\tau;A;\alpha;x;B;\gamma)$ is strictly positive in τ.

$$\begin{aligned}\widehat{\Sigma}(\tau)T\phi \text{ iff } \exists\, A\ \alpha\ x\ B\ \gamma.\ & (\Sigma x\in A)B \leftarrow T \ \&\ \mathrm{Fam}(\tau;A;\alpha;x;B;\gamma)\\ & \&\ \forall\, t\, t'.\ t\phi t' \text{ iff } \exists\, a\ b\ a'\ b'.\ (a,b)\leftarrow t \ \&\ (a',b')\leftarrow t'\\ & \qquad\qquad \&\ a\alpha a' \ \&\ b\,\gamma_a b'.\end{aligned}$$

$$\begin{aligned}&\widehat{\Pi}(\tau)T\phi\\ &\text{iff } \exists\, A\ \alpha\ x\ B\ \gamma.\ (\Pi x\in A)B\leftarrow T \ \&\ \mathrm{Fam}(\tau;A;\alpha;x;B;\gamma)\\ &\qquad \&\ \forall\, t\, t'.\ t\phi t' \text{ iff } \exists\, u\ b\ u'\ b'.\ (\lambda u)b\leftarrow t \ \&\ (\lambda u')b'\leftarrow t'\\ &\qquad\qquad \&\ \forall\, a\, a'.b[a/u]\,\gamma_a\, b'[a'/u']\\ &\qquad\qquad\qquad \text{if } a\alpha a'.\end{aligned}$$

Let us define the equality for W types.

$$\begin{aligned}&\mathrm{Weq}(\alpha;\gamma) \text{ is the strongest } \phi \text{ such that}\\ &\forall\, t\, t'.\ t\phi t' \text{ if } \exists\, a\ f\ u\ s\ a'\ f'\ u'\ s'.\ \mathrm{sup}(a,f)\leftarrow t \ \&\ (\lambda u)s\leftarrow f\\ &\qquad \&\ \mathrm{sup}(a',f')\leftarrow t' \ \&\ (\lambda u')s'\leftarrow f'\\ &\qquad \&\ a\alpha a' \ \&\ \forall\, b\, b'.\ s[b/u]\,\phi\, s'[b'/u'] \text{ if } b\,\gamma_a b'.\end{aligned}$$

$$\begin{aligned}&\widehat{\mathrm{W}}(\tau)T\phi\\ &\text{iff } \exists\, A\ \alpha\ x\ B\ \gamma.\ (\mathrm{W}x\in A)B\leftarrow T \ \&\ \mathrm{Fam}(\tau;A;\alpha;x;B;\gamma) \ \&\ \phi \text{ is } \mathrm{Weq}(\alpha;\gamma).\end{aligned}$$

It may be of interest that in each of these definitions of type formation, the definition of the member equality of a type depends only on the member equalities of its constituent types.

We may now define type formation under these constructors plus any base types.

$$\mathrm{TyF}(\sigma;\tau)T\phi \text{ iff } \sigma T\phi \text{ or } \widehat{\mathrm{N}}_? T\phi \text{ or } \widehat{\mathrm{N}} T\phi \text{ or } \widehat{\mathrm{I}}(\tau)T\phi \text{ or } \widehat{+}(\tau)T\phi$$
$$\text{or } \widehat{\Sigma}(\tau)T\phi \text{ or } \widehat{\Pi}(\tau)T\phi \text{ or } \widehat{\mathrm{W}}(\tau)T\phi.$$

The relation $\mathrm{TyF}(\sigma;\tau)T\phi$ is strictly positive, hence monotonic, in τ. Clearly, the relation $s \leftarrow t$ can be graphed as a subset of $\mathcal{T} \times \mathcal{T}$, and therefore, $\mathrm{TyF}(\sigma;\tau)$ is a two-place operator on the subsets of $\mathcal{T} \times Pow(\mathcal{T} \times \mathcal{T})$, which are the graphable possible type systems. Hence, the definition of $\mu(\sigma)$ will be valid for graphable σ. Let us introduce a convenient notation for closure under $\mathrm{TyF}(\sigma;)$.

$\mathrm{CTyF}(\sigma;\tau)$ iff $\forall\, T\ \phi.\ \tau T\phi$ if $\mathrm{TyF}(\sigma;\tau)T\phi$.

Now we define μ.

$\mu(\sigma)$ is the strongest τ such that $\mathrm{CTyF}(\sigma;\tau)$.[8]

To complete the definition, we need only define the universes and add them as base types. We shall define the hierarchy HAN_n of type systems that are reflected in universes. The types of the type system spine_n are the universes of HAN_n.[9]

HAN_n is $\mu(\mathrm{spine}_n)$.

$\mathrm{spine}_n T\phi$ iff $\exists m < n.\ \mathrm{U}_m \leftarrow T\ \&\ \phi$ is $=_{\mathrm{HAN}_m}$.

We now define the principal type system via the union of spines.[10]

HAN_ω is $\mu(\mathrm{spine}_\omega)$.

$\mathrm{spine}_\omega T\phi$ iff $\exists m.\ \mathrm{U}_m \leftarrow T\ \&\ \phi$ is $=_{\mathrm{HAN}_m}$.

It remains to be seen that these possible type systems are indeed type systems.

---

[8]This is set theoretically valid when σ is graphable, and this is sufficient for our purpose.

[9]HAN_n and spine_n are graphable.

[10]HAN_ω and spine_ω are graphable.

Adequacy of the Definition

Is HAN_ω a reasonable reinterpretation of the type constructors defined in [Martin-Löf]? It seems to me that the definition is posed in a way that would make the adequacy of HAN_ω obvious if only we should ascertain that

- HAN_ω and each HAN_n are extensional type systems, and
- each type of HAN_n is also a type of HAN_{n+i} and of HAN_ω, with the same member equality.

If the theorems that follow are to be read set theoretically, the variables σ and τ should be restricted to graphable possible type systems.

Theorem $\mu(\sigma)$ is $\mathrm{TyF}(\sigma;\mu(\sigma))$.
Since $\mathrm{CTyF}(\sigma;\ \mathrm{TyF}(\sigma;\mu(\sigma))\,)$,
since $\mathrm{TyF}(\sigma;\tau)$ is monotonic in τ, and $\mathrm{CTyF}(\sigma;\mu(\sigma))$.

In fact, $\mu(\sigma)$ is the strongest fixed point of $\mathrm{TyF}(\sigma;)$.

It will be convenient to make the following crude abstraction from the spines.

$\mathrm{Uonly}(\sigma)$ iff $\forall\, T\ \phi.$ if $\sigma T \phi$ then $\exists n.\ \mathrm{U}_n \leftarrow T$.

Lemma $\mathrm{Fun}(\mu(\sigma))$ if $\mathrm{Fun}(\sigma)$ & $\mathrm{Uonly}(\sigma)$.
Since $\mathrm{Fun}(\mu(\sigma))$ iff $\forall\, T\ \phi.$ if $\mu(\sigma)T\phi$ then $\forall \phi'.\ \phi$ is ϕ' if $\mu(\sigma)T\phi'$,
and if $\mathrm{Fun}(\sigma)$ & $\mathrm{Uonly}(\sigma)$ then
$\mathrm{CTyF}(\ \sigma\ ;\ T,\phi.\ \forall\phi'.\ \phi$ is ϕ' if $\mu(\sigma)T\phi'\ )$.

Lemma $\mathrm{TyVal}(\mu(\sigma))$ if $\mathrm{TyVal}(\sigma)$.
Since $\forall\tau.\ \mathrm{TyVal}(\mathrm{TyF}(\sigma;\tau))$ if $\mathrm{TyVal}(\sigma)$.

Lemma $\mathrm{TrSy}(\mu(\sigma))$ if $\mathrm{TrSy}(\sigma)$.
Since if $\mathrm{TrSy}(\sigma)$ then $\mathrm{CTyF}(\ \sigma\ ;\ T,\phi.\ \phi$ is transitive and symmetric$\,)$.

Lemma $\mathrm{Val}(\mu(\sigma))$ if $\mathrm{Val}(\sigma)$.
Since if $\mathrm{Val}(\sigma)$ then $\mathrm{CTyF}(\ \sigma\ ;\ T,\phi.\ \forall\, t\ s.\ t\phi s$ iff $\exists t' \leftarrow t.\ t'\phi\, s\ )$.

Lemma ETS($\mu(\sigma)$) if ETS(σ) & Uonly(σ).

Lemma If Fun(τ) then $=_\tau$ is transitive and symmetric.

Lemma If TyVal(τ) then $\forall\, t\, s.\ t =_\tau s$ iff $\exists t' \leftarrow t.\ t' =_\tau s$.

Theorem ETS(HAN_n) & ETS(spine_n), by induction on n.
Since $\forall n$. Uonly(spine_n).

Theorem ETS(HAN_ω).
Since ETS(spine_ω) & Uonly(spine_ω).

Lemma $\mu(\sigma)$ is monotonic in σ.
Since assuming σ to be as strong as σ', CTyF($\sigma; \mu(\sigma')$),
since CTyF($\tau; \tau'$) is antimonotonic in τ, and CTyF($\sigma'; \mu(\sigma')$).

Theorem HAN_n is a subrelation of HAN_{n+i} and HAN_ω.
Since spine_n is a subrelation of spine_{n+i} and spine_ω, and μ is monotonic.

Formal Proof — Consistency

Let us call the signs expressing the judgements of [Martin-Löf] *sequents*. Sequents can be defined by induction from the terms.[11] In [Allen], the reinterpretation of the judgements of [Martin-Löf] is rather directly effected by defining a property, Fn, of sequents in terms of the property T **type** and the relation $t = s \in T$. There are methods set out in [Allen] by means of which one can prove all the inference rules of [Martin-Löf] to be valid under the reinterpretation, that is, the conclusion of each rule is a sequent that satisfies Fn if the premises do.[12] Thus, every sequent provable using only those rules satisfies Fn.

For any terms T and t, there is a sequent, let us write $t \in T$, that satisfies Fn just when $t \in_{\mathrm{HAN}_\omega} T$. If T is an empty type of HAN_ω then the sequent $t \in T$ is not derivable, for any t, using the rules of [Martin-Löf].

[11] Here we include only the terms built using a fixed collection of term constructors.

[12] Martin-Löf intentionally elides some premises from the presentation of some inference rules in [Martin-Löf], so these must be made explicit in order to show these rules to be valid.

We may consider this as a form of simple consistency when propositions are represented by types, the representation being such that a proposition is true just when the type representing it is inhabited.

Comparison with Beeson's Recursive Models

We shall compare the type systems defined in this paper with Beeson's models for type theory, and we shall also compare the methods of definition. The difference between Beeson's models and the type systems presented here are of some practical interest, and there is some theoretical significance to the difference between the definitions. In [Beeson], Beeson presents a sequence $M_n W$ of models for the sequence $\mathrm{ML}_n\mathrm{W}$ of formal systems. The system $\mathrm{ML}_n\mathrm{W}$ consists of the inference rules given in [Martin-Löf], excluding those about universes U_{n+i}.[13] The model $M_n W$, which may be viewed as a recursive realizability interpretation, corresponds to the type system HAN_n, which may be viewed as a term model.

The basic difference between HAN_n and $M_n W$ is that HAN_n is a term model based upon the so-called lazy evaluation procedure described in [Martin-Löf], whereas $M_n W$ uses numbers instead of terms, and uses call-by-value computation semantics. Beeson codes the canonical term constructors, except for lambda, and uses $\{m\}(n)$ instead of the application of one term to another.

By means of a recursive definition, to be discussed below, Beeson defines some relations between numbers, including $M_n W \models i = j$ and $M_n W \models i = j \in k$. The terms of [Martin-Löf] are made to correspond to numbers by an effective *partial* function on closed terms, $\hat{t}$, also written $t\hat{}$. The connection with [Martin-Löf] is made by the following facts.

If $\mathrm{ML}_n\mathrm{W}$ proves $A = B$ then $M_n W \models \hat{A} = \hat{B}$,

and

[13] Actually, Beeson adds the rule

$$\frac{x \in \mathrm{N}}{\mathrm{I}(\mathrm{N}, \mathrm{suc}(x), 0) = \mathrm{N}_0}$$

which is derivable, but the derivation uses a universe. Beeson needs this rule to establish certain facts about $\mathrm{ML}_0\mathrm{W}$, but these are of no importance to us.

if $\mathrm{ML}_n\mathrm{W}$ proves $a = b \in A$ then $M_nW \models \hat{a} = \hat{b} \in \hat{A}$.

This model deviates immediately from the semantics even when restricted to the terms and type constructors explicitly given in [Martin-Löf]. For example, if t is $(\lambda z)(\,z(z), 0\,)$ then $t(t)$ evaluates to $(t(t), 0)$, hence $(\mathrm{E}\,x, y)(\,t(t), 0\,)$ evaluates to 0, and so the judgement $(\mathrm{E}\,x, y)(\,t(t), 0\,) = 0 \in \mathrm{N}$ is valid. But, $t(t)\hat{}$ is undefined, hence, $(\mathrm{E}\,x, y)(\,t(t), 0\,)\hat{}$ is also undefined, and so, it is false that $M_nW \models (\mathrm{E}\,x, y)(\,t(t), 0\,)\hat{} = \hat{0} \in \hat{\mathrm{N}}$.

While there may be no theoretically significant difference between the realizability model M_nW and the term model HAN_n, the latter is a more direct reinterpretation of the semantics of type definition given in [Martin-Löf]. Also, HAN_n is more convenient to use when one is concerned directly with the terms.

Now we turn to the definitions. In [Beeson], M_nW is defined, given M_mW for $m < n$, by mutual recursion between five properties and relations on numbers:

$M_nW \models i$ **type**,
$M_nW \models i = j \in k$,
$M_nW \models i \in j$,
i is a family of types in M_nW over j,
$M_nW \models i = j$.

The definition could have been accomplished by defining $M_nW \models i$ **type** and $M_nW \models i = j \in k$ by mutual recursion, and then explicitly defining each of the others in terms of these. To simplify our discussion we shall suppose the definition to have been given in this way.

This definition is of essentially the same form as the one sketched in this paper in the section headed *Anatomy of the Type Constructors.* The form is not the standard one for inductive definitions because of negative occurrences of $M_nW \models i = j \in k$ on the right-hand sides of some clauses of the definition. Nevertheless, Beeson explains, the definition is valid because all the members of a type are added at the same time as the type is introduced as such. Beeson then indicates how to define the model by standard means of induction. He applies the following *device* (*trick*). Define, by mutual recursion, not only $M_nW \models i$ **type** and $M_nW \models i = j \in k$, but also a relation $M_nW \not\models i = j \in k$. When $M_nW \models k$ **type**, $M_nW \not\models i = j \in k$

is to be defined to be the negation of $M_nW \models i = j \in k$. Wherever, in the clauses of the earlier definition, $M_nW \models i = j \in k$ occurs negatively on the right-hand side, replace that occurrence by the negation of $M_nW \not\models i = j \in k$. The defining clauses for $M_nW \not\models i = j \in k$ are made by imitating those for $M_nW \models i = j \in k$ except that the right-hand sides are negated (classically) in such a way that occurrences of $M_nW \models k$ **type** are positive, and then negative occurrences of $M_nW \models i = j \in k$ are eliminated as just suggested. The resulting definition consists of clauses whose right-hand sides are positive, though not strictly positive, in the definienda. Beeson says that, classically, it can be shown that $M_nW \models i = j \in k$ and $M_nW \not\models i = j \in k$ are complementary when $M_nW \models k$ **type**.

The similar definition of T **type** and $t = s \in T$ which was sketched in this paper has the same negative occurrences of the definienda on the right-hand sides of some clauses. However, here we proceeded to improve the definition by making explicit Beeson's explanation for the validity of the original definition, namely that the membership (and equality) of a type is defined when the type is. The validity of the definition does not depend upon defining the complement of $t = s \in T$ and so no appeal to non-intuitionistic principles is needed to justify the reformulation of the definition. More significantly, the resulting definition is, in this author's opinion, far more perspicuous than either the original one or the one achieved by the device utilized in [Beeson], as is evidenced by the immediate availability of induction over types.

References

[Allen] Stuart F. Allen. Doctoral Dissertation, Computer Science Department, Cornell University, 1987 (expected).

[Beeson] Michael Beeson. Recursive Models for Constructive Set Theories. *Annals of Mathematical Logic,* v. 23 (1982).

[Constable et al.] Robert Constable *et al. Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

[Martin-Löf] Per Martin-Löf. Constructive Mathematics and Computer Programming. *Sixth International Congress for Logic, Methodology, and Philosophy of Science.* North-Holland, Amsterdam, 1982.

[Nordstrom] Bengt Nordstrom. Programming in constructive set theory: some examples. *Proceedings 1981 Conference on Functional Programming Languages and Computer Architecture.* Portsmouth, England, 1981.

[Petersson & Smith] Kent Petersson and Jan Smith. Program Derivation in Type Theory: The Polish Flag Problem. Computer Science Department, University of Göteborg/Chalmers, Göteborg, Sweden, January 1985.