Categorical Type Theory

Prakash Panangaden
Michael I. Schwartzbach

TR 85-716
December 1985

Department of Computer Science
Cornell University
Ithaca, NY 14853

Categorical Type Theory

Prakash Panangaden
Michael Ignatieff Schwartzbach

Cornell University
December 1985

Abstract

This paper examines the connections between intuitionistic type theory and category theory. A version of type theory is developed in a category theoretic framework by interpreting types as objects and type connectives as constructions in categories. This yields a theory that uniformly models type inhabitation, proof theory, type universes, syntactic forms, equality and computation.

Categorical Type Theory

by
Prakash Panangaden
and
Michael Ignatieff Schwartzbach

Introduction

In this paper we examine the connections between intuitionistic type theory and category theory. Both of these topics have begun to have significant impact on computer science, but the relationship between them has not been much explored. Historically, type theory and category theory have shared foundational considerations, so the connections between them have long been known at an intuitive level. Our goal is to show how a slightly modified version of traditional type theory can naturally be developed in a category theoretic context. This defines a notion of models for type theory that may serve as a basis for metamathematical reasoning. A further attraction to this approach is that it may help to make type theory more accessible to an audience with a background in category theory [1, 8].

The program we describe in this paper owes much to the pioneering work of F.W. Lawvere, J. Lambek and D. Scott. Lawvere and his followers showed how topos theory could serve as a category theoretic formulation of logic [2, 10, 11]. Our work departs from topos theory in that we take type inhabitation as the fundamental component of our semantics rather than truth. Lambek has discussed the relation between cartesian closed categories and the λ-calculus [9], and the connection with proof theory has been thoroughly examined by Szabo [18]. Our work can be viewed as a generalization of this program from the λ-calculus to the much richer language of

Martin-Löf's Intuitionistic Type Theory [13]. The connections with intuitionistic logic have been discussed by Scott [15]. Similar work has also been reported by Brezau-Tannen and Mayer [3] and by Curien [6]. The semantics described in this paper has been used to explicate recursive and infinite type constructors. These are extensions to the Martin-Löf type theory carried out by the PRL group at Cornell [5, 14].

Overview of Type Theory

Type theory is a constructive theory of types that has been suggested by Per Martin-Löf [12] and further explored and modified by the PRL Project headed by Robert L. Constable [4]. As our aim is to develop a theory of types in a purely category theoretic framework, we do not want to present any particular version of type theory. We have, however, clearly been inspired by these theories, and it will serve as a motivation and a clarification, if we can convey an impression of the philosophy and content of other type theories.

A type theory consists of an countably infinite accumulative hierarchy of *type universes*. These universes are introduced to allow type quantification without impredicativity. Each universe contains a collection of *types*; with each type is associated its *inhabitants* or *terms*. Types may be combined into new types by a number of *type connectives* or *type constructors*. The inhabitants of such a derived type are constructed from the inhabitants of its component types. The theory provides *inference rules* that allow one to deduce which types are inhabited, and furthermore, these rules are *constructive*. Using constructive inference rules implies that from a proof of inhabitation of a type T, one can *extract* a concrete inhabitant of T. Finally, a *syntax of terms* is given along with inference rules to demonstrate *equality* of syntactic forms and *computation rules* to reduce syntactic expressions to normal forms. In a conservative type theory all syntactic forms correspond to inhabitants of appropriate types; this is not the case in the PRL type theory.

This presentation of the essentials of type theory is strong biased towards justifying our semantics. It is possible to maintain an almost dual viewpoint, where the syntactic forms and their computation rules are fundamental, and types are a secondary classification of terms which respects computation.

Type theory has been presented as a constructive alternative to set theory and, hence, as a foundation for constructive mathematics. It may also be viewed as an extension of logic, in which types are identified with propositions and the inhabitants of a type are the collection of constructive proofs for the corresponding proposition. In this logical interpretation the concept of *truth* is replaced by the concept of *inhabitation*, and proof rules establish inhabitation rather than truth in the logical sense. This view is taken as fundamental in our semantics, in contrast to topos theory which models truth in categories.

Another view, of particular interest to us, is to think of this concept as a *theory of correct programs* by identifying types with specifications of computations, and inhabitants of a type as programs that satisfy the corresponding specification [13]. Thus the programming process becomes the development of a proof of inhabitation of a specification, from which a correct program can be extracted. A programming system realizing this approach has been developed by the PRL Project [4].

Type Categories

We develop a theory of types within a categorical framework. More precisely, we shall interpret the basic concepts presented in the previous chapter in a certain class of categories, called *type categories*, and demonstrate how the entire theory can be evolved through standard categorical constructions. We still have occasional references to set theory (as does most of category theory), but work is in progress to completely internalize all concepts.

Definition: A *type category (TC)* is a countably bi-complete category $\underline{\mathbf{C}}$, with initial object $\mathbf{0}$, final object $\mathbf{1}$, products $\times$, co-products $+$, and exponents $\rightarrow$, that satisfies the following distributivity axiom: $A\times\Sigma(i\in I, B_i) \simeq \Sigma(i\in I, A\times B_i)$, where I is any countable index set.

This definition contains redundant criteria but clearly points out the concepts we shall use. Notice that a type category is cartesian closed. The following are examples of TCs: $\underline{\mathbf{1}}$, $\underline{\mathbf{2}}$, $\underline{\mathbf{Set}}$, $\underline{\mathbf{Set}^2}$, and $\underline{\mathbf{Sets\text{-}in\text{-}time}}$; in fact, many toposes are TCs [7], and every complete lattice can be interpreted as a skeletal TC. The following result shows that no interesting domain category with a universal object is a TC.

Theorem (Two-of-Three): In a non-degenerate cartesian closed category one can have any two, but not all three, of the following constructs: an initial object, a final object, and a universal object.

Proof: Suppose we have $\mathbf{0}$ initial, $\mathbf{1}$ final, and U universal. Then for any object A we get morphisms ϕ_A: $A \rightarrow U$ and ψ_A: $U \rightarrow A$, such that $\psi_A \circ \phi_A = id_A$. In particular, we get the composite $\psi_0 \circ \phi_1$: $\mathbf{1} \rightarrow \mathbf{0}$. But in any cartesian closed category the existence of a morphism from the final to the initial object implies degeneracy. To see that a combination of any two is possible, we need only look at the categories **Set** (initial and final), **CPO** (final and universal), and **DP2** [16] (initial and universal). □

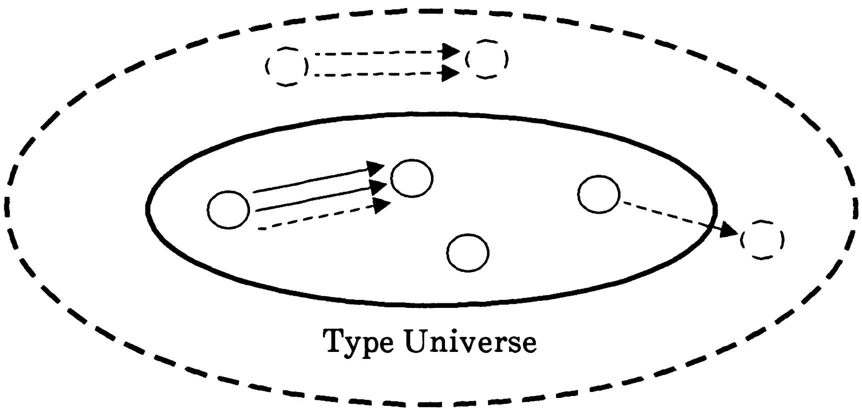
We intend to use the theory of *all* type categories, as given by the axioms for basic categories and the particular constructions we require. Hence, we shall only use results that are valid in *every* type category. Any particular TC may be viewed as a *model* of this theory. From type categories we derive *type universes*, and it is our claim that type theory is exactly the theory of all type universes.

In a TC we interpret the objects as *types*. The *type universe,* TU, contained in a TC is the collection of types constructed from **0** and **1** using countably many applications of the available type connectives. The collection of *terms* associated with a type T is the subset of $hom(1,T)$ for which we can prove membership, i.e. they must arise as morphisms being extracted from proofs; the notion of *proof* will be formally defined below. Hence, a type T is inhabited iff we can prove that there is a morphism t: $1 \rightarrow T$. This also implies that the terms of T may be viewed as merely the collection of (equivalence classes of) proofs of inhabitation of T. We see that inhabitation of **0** implies degeneracy of the category, and that 1 always has the unique inhabitant id_1, i.e. 1 is a singleton type. These definitions can be partly justified by noting that in **Set**, $hom(1,T)$ is isomorphic to T. This is also the concept of *membership* used in topos theory.

We define an *open term* of type T to be a morphism from a finite product $T_1 \times T_2 \times \ldots \times T_n$ to T, again arising from the proof rules, or equivalently a term in $hom(1,T^{T_1 \times T_2 \times \ldots \times T_n})$. The intuitive meaning of an open term, t, is that given terms of types $T_1, \ldots, T_n$ it will produce a term of type T. If t_i is a term $1 \rightarrow T_i$ we get from universality of the product a term ϕ: $1 \rightarrow T_1 \times T_2 \times \ldots \times T_n$; composition then yields the term $t \circ \phi$: $1 \rightarrow T$. Since the empty product is the final element 1, we see that a *closed term* (i.e. the case where $n = 0$) is simply an ordinary term.

The basic type constructors we introduce are product ($\times$), sum ($+$), and exponentiation ($\rightarrow$). These are realized in a TC as the corresponding categorical constructions. More powerful constructors will be introduced in subsequent chapters, as their definition requires higher universes. These type connectives are interesting because they mimic the usual logic connectives when inhabitation is interpreted as truth. For example, the type $T_1 + T_2$ is inhabited exactly when T_1 is inhabited *or* T_2 is inhabited and, hence, behaves like the logic connective $\vee$. It is worthwhile to compare this simple construction with the topos theoretic interpretation of $\vee$.

The universe of types forms a subcategory of the TC when we define the collection of morphisms between types A and B to be the set of open terms $A \rightarrow B$.



Type Category

This subcategory is often more well-behaved than the original category. In the type universe we only have countably many objects and hom-sets are likewise countable. To see this we need only observe that the set of type expressions in a TU form a context-free language over a finite alphabet, and that there are only countably many proofs available. All the type constructions can be carried out in the TU subcategory, and they yield the same result as in the TC. This is because our type constructions only involve the terms of a type and not all the morphisms from 1. Another interesting fact is that all empty objects in the universe of types are initial. As a contrast, the (type) category $\underline{\mathbf{Set}}^2$ has the object $<\{\bullet\}, \{\}>$ which is empty but not initial. Once the type constructors and their rules have been given, the claim that emptiness implies initiality could be verified inductively in the type structure.

It is important to notice that a type category only defines a single type universe and not the entire hierarchy. Later on we shall give a uniform construction that will raise a universe to the next higher universe level.

Proof Theory

We must develop a constructive proof theory of inhabitation of types. This will turn out to be similar to a refinement style constructive proof theory of logic, but in this context all concepts are given meaning in a categorical framework.

Inhabitation of a type T is the existence of a morphism from 1 to T. We expect a proof of inhabitation of T to actually construct such a morphism, which we shall call a *witness* for T, or a *term* of T.

The basic unit of proof theory is a *sequent*:

$$T_1, T_2, ..., T_n \vdash T$$

which we read as follows: *if inhabitation of T_1 through T_n is given, we can conclude that T is inhabited.* In a type category this is exactly realized by the existence of a morphism:

$$m: T_1 \times T_2 \times ... \times T_n \rightarrow T$$

called the *validation* of the sequent. If we are given witnesses w_i: $1 \to T_i$ then universality of the product induces a unique morphism ψ: $1 \to T_1 \times T_2 \times \ldots \times T_n$. But then composition yields $m \circ \psi$: $1 \to T$, which is a witness for T. These morphisms from the TC are used to *bootstrap* the terms in the induced TU.

We say that a sequent is *valid* iff it has a validation in every TC. The product $T_1 \times T_2 \times \ldots \times T_n$ is called the *hypothesis* of the sequent. An example of a valid sequent is

$$A, B \vdash A$$

since in every TC we have the projection morphism π_A: $A \times B \to A$. This nicely confirms the intuition of *projection of information* contained in the same sequent in logic.

The next level of proof theory introduces *inference rules*:

$$\frac{S}{S_1, S_2, \ldots, S_n}$$

where S and the S_i's are sequents. The inference rule reads as follows: *if all the S_i's are valid, we can conclude that S is valid.* Since we want a *constructive* proof theory this translates in our interpretation to: *from validations for the S_i's we can construct a validation for S*. An inference rule is called *sound* if this is possible in every TC.

Such rules are normally organized into *introduction* and *elimination* rules for each type constructor. It turns out that all these rules in a natural way can be derived from the various universality and co-universality properties of the categorical constructions. The following rules arise from the basic type connectives.

$\times$-*introduction*: $$\frac{H \vdash A \times B}{H \vdash A \quad H \vdash B}$$

Here H is any hypotesis, i.e. a product of types, and A and B are arbitrary types. To see that this rule is sound we can assume that we are given the validations a: H $\rightarrow$A and b: H $\rightarrow$B. Universality of the product induces a unique morphism ψ: H $\rightarrow$A$\times$B, that serves as a the validation for H$\vdash$A$\times$B.

$$\times\text{-}elimination: \quad \frac{H\vdash A}{H\vdash A\times B} \qquad \frac{H\vdash B}{H\vdash A\times B}$$

These rules are seen to be sound, since a validation v: H$\rightarrow$A$\times$B can be composed with either π_A: A$\times$B $\rightarrow$A or π_B: A$\times$B $\rightarrow$B to form a validation for H$\vdash$A or H$\vdash$B.

$$+\text{-}introduction: \quad \frac{H\vdash A+B}{H\vdash A} \qquad \frac{H\vdash A+B}{H\vdash B}$$

These are sound rules, since a validation a: H $\rightarrow$A can be composed with the canonical injection in_A: A $\rightarrow$A+B to form a validation for H$\vdash$A+B. Similarly for the other case.

$$+\text{-}elimination: \quad \frac{H\vdash C}{H\vdash A+B \quad H,A\vdash C \quad H,B\vdash C}$$

Given validations v: H $\rightarrow$A+B, a: H$\times$A $\rightarrow$C, and b: H$\times$B $\rightarrow$C we can demonstrate soundness of this rule as follows. From co-universality is induced a unique morphism ψ: (H$\times$A)+(H$\times$B) $\rightarrow$C. Universality gives us $id_H\times v$: H $\rightarrow$H$\times$(A+B). Distributivity provides a morphism ϕ: H$\times$(A+B) $\rightarrow$(H$\times$A)+(H$\times$B), from which we can construct the validation $\psi \circ \phi \circ (id_H\times v)$: H $\rightarrow$C for the sequent H$\vdash$C. If H was empty we would not need distributivity

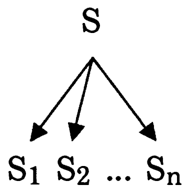
$$\rightarrow\text{-}introduction: \quad \frac{H\vdash A\rightarrow B}{A,H\vdash B}$$

This rule is sound since a validation v: $A \times H \to B$ and cartesian closure immediately yields a validation $curry(v)$: $H \to B^A$.

$\to$-elimination: $$\frac{H \vdash B}{H \vdash A \to B \quad H \vdash A}$$

Soundness of this rule again follows from cartesian closure. If we have the validations f: $H \to B^A$ and a: $H \to A$ we use universality to get $a \times f$: $H \to A \times B^A$ and by composition we have the validation $eval \circ (a \times f)$: $H \to B$.

A *proof tree* (which is not yet a *proof*) is a tree of finite height where the nodes are sequents and a node and its descendants:



are related through a sound inference rule:

$$\frac{S}{S_1 \; S_2 \; \dots \; S_n}$$

A proof of (the inhabitation of) a type T is a proof tree with root:

$\vdash T$

i.e. a sequent which concludes T from an empty hypothesis, and whose leaves are all of the following kinds:

$T_1, T_2, \dots, T_n \vdash T_i$ *(projection)*

$H \vdash 1$ (*trivial*)

$0 \vdash T$ (*absurd*)

i.e. sequents which are all trivially valid. Clearly validations can be constructed from the leaves and upwards through the tree, the final result being a validation of the root sequent $\vdash T$; but this is just a morphism from 1 (the empty product) to T, or a *witness* for T. Hence a proof of inhabitation of T constructively yields an inhabitant of T. The relationship that t is a morphism $1 \rightarrow T$ originating as the witness of a proof for T is written $t \in T$.

Later on we shall introduce more sophisticated proofs by allowing dependencies in the sequents.

A Hierarchy of Universes

A TC forms a type universe which we shall call U_1. We wish to extend this to a hierarchy of type universes, $\{U_k\}_{k \in N}$, all of which arise from TCs. Furthermore, we want the hierarchy to be accumulative in the sense that U_k must appear as a type in U_{k+1} and every type in U_k must also be a type in U_{k+1}. This, together with new type constructors, allows us to perform quantification over types without the risk of impredicativity.

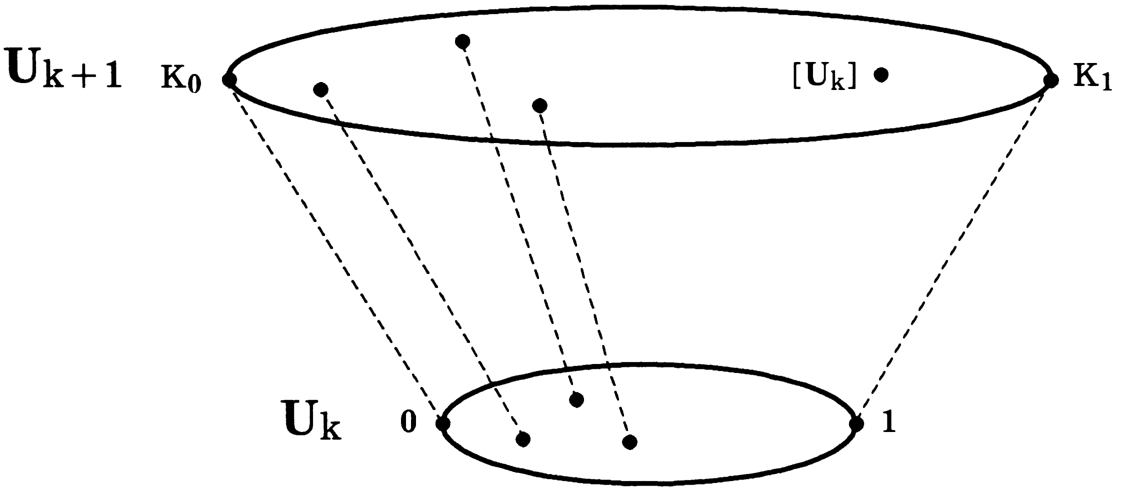
If $\underline{C}$ is a type category forming a type universe U_k we define $F(\underline{C})$ to be the category whose objects are covariant endofunctors $F: \underline{C} \rightarrow \underline{C}$ and whose morphisms are natural transformations. The following result tells us that we can form a new type universe in $F(\underline{C})$:

Theorem: $\mathbf{F}(\underline{\mathbf{C}})$ is a type category.

Proof: All countable limits and co-limits can be constructed pointwise in $\mathbf{C}$. It follows that $\mathbf{F}(\underline{\mathbf{C}})$ is countably bi-complete, which is sufficient to carry out the type constructions. Distributivity similarly follows from pointwise distributivity. □

In particular in $\mathbf{F}(\underline{\mathbf{C}})$ we get that K_0 is initial and K_1 is final, where K_A is the constant functor such that for every object O we have $K_A(O)=A$ and for every morphism m we have $K_A(m) = id_A$. We define $\mathbf{U}_{k+1}$ to contain K_T for every $T \in \mathbf{U}_k$ and to be closed under countable applications of type constructors. We must now show that this hierarchy is accumulative. $\mathbf{U}_k$ is embedded into $\mathbf{U}_{k+1}$ as follows:

- $T \in \mathbf{U}_k$ is embedded as $K_T \in \mathbf{U}_{k+1}$
- $\mathbf{U}_k$ itself is embedded as $[\mathbf{U}_k] = \Sigma(T \in \mathbf{U}_k, K_1) \in \mathbf{U}_{k+1}$



The following results shows that this is really what we want.

Theorem: $hom(K_A, K_B) \simeq hom(A, B)$

Proof: This is a consequence of Yoneda's lemma but can be easily demonstrated by the following elementary argument: Let m: X →Y be any morphism in $\underline{C}$ and ψ a natural transformation from K_A to K_B. We then have the following commuting diagram:

$$\begin{tikzcd}
K_A(X) \arrow[r, "K_A(m)"] \arrow[d, "\psi X"'] & K_A(Y) \arrow[d, "\psi Y"] \\
K_B(X) \arrow[r, "K_B(m)"] & K_B(Y)
\end{tikzcd}$$

which reduces to the diagram:

$$\begin{tikzcd}
A \arrow[r, "id_A"] \arrow[d, "\psi X"'] & A \arrow[d, "\psi Y"] \\
B \arrow[r, "id_B"] & B
\end{tikzcd}$$

From this it follows that ψ must be constant and that any choice of ψ exactly corresponds to a morphism from A to B. □

The function from hom(A, B) to $hom(K_A, K_B)$ that is implied in this result is called the *embedding transformation.*

Our choice of $[U_k]$ is also correct, as the collection of terms of $[U_k]$ is easily seen to be isomorphic to the collection of types in U_k.

It is also worth noticing that if U_k is countable then so is U_{k+1}; hence, we have constructed a countable hierarchy of countable universes. Strictly speaking, if $\underline{C}$ is a category then $\mathbf{F}(\underline{C})$ is a *quasi-category*, as the collection of morphisms may no longer be a set. This indicates, however, more than anything else a weakness in the definition of categories. In general, each each application of F(__) brings us one step higher in Russell's type hierarchy. The need for a hierarchy of category universes has been proposed by Grothendieck.

Given the faithfulness of the universe embedding we shall allow ourselves the notational convenience of being fairly lax about the exact universe level in which we operate, e.g. we will write A instead of K_A.

We need new inhabitation rules to enable us to move between universes. To indicate in which universe level we are working, we shall equip sequents with a universe index:

$$H \vdash_i T$$

If no index is present the sequent may be interpreted in any universe.

$$\textit{embedding:} \qquad \frac{H \vdash_{i+j} T}{H \vdash_i T}$$

This rule tells us that a validation can be lifted to any higher universe level; this happens by j applications of the embedding transformation.

$$\textit{inclusion}\ T \in U_k: \qquad H \vdash_{k+1} [U_k]$$

This is a family of valid sequents indexed by all the types in a universe. They allow one to reason about inhabitation of a universal type. These sequents may also appear as leaves in proofs.

Dependent Types

Through higher universes we can define new type constructors, the so called *dependent types*. These types are proposed by Martin-Löf to enable the typing of structures where the types of some components depend on the types of others, e.g. it allows us to express the type of all finite automata, where for example the type of the transition function depends on the type of the input alphabet. We shall carry out these constructions in U_{k+1}; this is where we hope to achieve more internalized constructs in the future.

If A is a type in U_k, B is a term $A \to [U_k]$, and x is a term $1 \to A$ we can define the composition $B \circ x$: $1 \to [U_k]$. Since $hom(1, [U_k])$ is isomorphic to U_k in the type universe this corresponds to a type $|B \circ x| \in U_k$. We can now define the *dependent product and sum* types as follows.

dependent product: $\Pi(A.\,B)$ denotes the type $\Pi(x \in A, |B \circ x|)$

dependent sum: $\Sigma(A.\,B)$ denotes the type $\Sigma(x \in A, |B \circ x|)$

where the latter occurrences of Π and Σ are the categorical product and sum respectively. This shows why we need TCs to be countably bi-complete. We also depend on the fact that types in a TU only have countably many terms; these constructions may not be possible in a general TC.

It is interesting to notice that in the case where B is a constant K_C with $C \in U_k$ we get:

$$\Pi(A.\,B) \simeq A \to C \quad \text{and} \quad \Sigma(A.\,B) \simeq A \times C$$

Thus, curiously enough, the dependent product generalizes the exponent and the dependent sum generalizes the product. (In the PRL type theory the dependent product is called *dependent function space* and the dependent sum is called *dependent product.*)

Extended Proof Theory

The dependent type constructors lead us to define a more powerful proof theory. We need to generalize the notion of a sequent to allow for dependencies. The most general form of a sequent is:

$$A \vdash B$$

where A is the list:

$$x_1\colon A_1, x_2\colon A_2(x_1), \ldots, x_n\colon A_n(x_1, x_2, \ldots, x_{n-1})$$

and B is the list:

$$y_1\colon B_1(x_1, \ldots, x_n), y_2\colon B_2(x_1, \ldots, x_n, y_1), \ldots, y_m\colon B_m(x_1, \ldots, x_n, y_1, y_2, \ldots, y_{m-1})$$

A sequent like this translates to the type $\Pi(S_n.\, T_m)$ where we define:

$$S_1 = A_1,\ S_{i+1} = \Sigma(S_i.\, A_{i+1})$$

and:

$$T_1 = B_1, T_{i+1} = \Sigma(T_i.\, B_{i+1})$$

The sequent is *valid* iff in every TC there is a morphism $1 \to \Pi(S_n.\, T_m)$. In the case where we have no dependencies in the sequent this reduces to the original definition of a valid sequent. When working with sequents we shall exploit the absence dependencies and reduce them to their simplest form; thus the sequent

$$A, B \vdash x{:}C, D(x)$$

will be assigned the type $A \times B \to \Sigma(C.\ D)$ rather than the type $\Pi(\Sigma(A.\ B).\ \Sigma(C.\ D))$. It is perhaps less obvious why this is the appropriate definition. An example is the sequent

$$x{:}A, B(x) \vdash C$$

with the type $\Sigma(A.\ B) \to C$. If we have morphisms from 1 into the elements of the hypothesis, a: $1 \to A$ and b: $1 \to |B \circ a|$, we can use the canonical injection in_a to obtain the morphism $in_a \circ b$: $1 \to \Sigma(A.\ B)$, which composed with a validation for the sequent would yield a morphism $1 \to C$. In this way the definition of these generalized sequents can be motivated as in the simpler case.

Proof trees are defined as before, but we must introduce a new inference rule to split conclusions:

$$\textit{split}: \qquad \frac{H \vdash x{:}A, B(x)}{\vdash A \quad H \vdash \Phi}$$

We then restrict the notion of a proof to be a proof tree where in applications of this rule it is true that $\Phi = |B \circ a|$ where a is the validation of $\vdash A$. By virtue of the canonical injection morphism this is a sound rule. The *split* rule allows us to perform two independent subproofs for a dependent conclusion, provided that they can be unified later.

Rules for Dependent Types

We can now present inhabitation rules for the dependent types. To prove soundness we shall need the following:

Lemma: $\Pi(A \times B.\ C) \simeq \Pi(A.\ \Pi(B.\ C))$

Proof: This is just a generalized version of currying that follows from the isomorphism $A \times B \to [U_k] \simeq A \to (B \to [U_k]))$. □

The rules for Σ and Π types follow a beautifully symmetric pattern.

$$\Sigma\textit{-introduction:} \qquad \frac{H \vdash \Sigma(A.\ B)}{H \vdash x{:}A,\ B(x)}$$

This rule is trivially sound since the morphism we are given immediately serves as a validation.

$$\Sigma\textit{-elimination:} \qquad \frac{H \vdash C}{H \vdash \Sigma(A.\ B) \quad H,\ x{:}A \vdash B(x) \to C}$$

Demonstrating soundness is more complicated for this rule. We are given the morphisms f: $H \to \Sigma(A.\ B)$ and g: $1 \to \Pi(H \times A.\ B \to C)$. From this we are able to construct $h = id_H \times (g \circ !)$: $H \to H \times \Pi(H \times A.\ B \to C)$. Various isomorphisms yield the morphism k: $H \to H \times (H \to \Pi(A.\ B \to C))$ from which we use properties of the exponent to get $m = eval \circ k$: $H \to \Pi(A.\ B \to C)$. This gives us $m \times f$: $H \to \Pi(A.\ B \to C) \times \Sigma(A.\ B)$. Using distributivity we end up with ϕ: $H \to \Sigma(A.\ \Pi(A.\ B \to C) \times B)$. For every $a \in A$ we can construct $eval \circ (\pi_a \times id_{|B \circ a|})$: $\Pi(A.\ B \to C) \times |B \circ a| \to C$; co-universality then induces a unique ψ: $\Sigma(A.\ \Pi(A.\ B \to C) \times B) \to C$, so by a final composition we have constructed the validation $\psi \circ \phi$: $H \to C$.

Π-*introduction*: $$\frac{H \vdash \Pi(A.\ B)}{H, x{:}A \vdash B(x)}$$

Here soundness follows from a simpler argument. From f: $1 \to \Pi(H \times A.\ B)$ we can construct $id_H \times (f \circ !)$: $H \to H \times \Pi(H \times A.\ B)$. From the isomorphism in the lemma we get a morphism ϕ: $H \to H \times (H \to \Pi(A.\ B))$, and properties of the exponent give us the validation $eval \circ \phi$: $H \to \Pi(A.\ B)$.

Π-*elimination*: $$\frac{H \vdash C}{H \vdash \Pi(A.\ B) \quad H \vdash x{:}A, B(x) \to C}$$

This rule is sound since from f: $H \to \Pi(A.\ B)$ and g: $H \to \Sigma(A.\ B \to C)$ we can construct the morphism $f \times g$: $H \to \Pi(A.\ B) \times \Sigma(A.\ B \to C)$. From distributivity it follows that we have ϕ: $H \to \Sigma(A.\ \Pi(A.\ B) \times (B \to C))$. For every $a \in A$ we are now able to construct the morphism $eval \circ (\pi_a \times id_{|B \circ a| \to C})$: $\Pi(A.\ B) \times (|B \circ a| \to C) \to C$; co-universailty then induces a unique ψ: $\Sigma(A.\ \Pi(A.\ B) \times (B \to C)) \to C$. By a final composition we get the validation $\psi \circ \phi$: $H \to C$.

Subtype Construction

A final type connective is a subtype constructor. If A is a type in U_k and B is a term $A \to [U_k]$, then the subtype

$\{A \mid B\}$

is defined to be the limit in the following pullback diagram:

$$\begin{tikzcd}
\{A \mid B\} \arrow[r, dashed, "\sigma"] \arrow[d, dashed, "!"'] & A \arrow[d, "B^*"] \\
1 \arrow[r, "in_1"'] & 1+1
\end{tikzcd}$$

B^* is defined in the following manner; for every $T \in U_k$ define the map I_T: $1 \rightarrow 1+1$ to be in_1 if T is inhabited and to be in_2 otherwise. Then co-universality of $[U_k]$ induces a unique map I^*: $[U_k] \rightarrow 1+1$. Now we can define $B^* = I^* \circ B$. This is very similar to the definition of subobjects in topos theory [7]. Note that we always use $1+1$ even though it may not be a proper subobject classifier. This just means that we lose the uniqueness of characters for a given monic, but as we start by defining the character (i.e. B^*) this is immaterial. We get the following inhabitation rules for subtypes:

{}-*introduction*: $$\frac{H \vdash \{A \mid B\}}{H \vdash x{:}A,\, B(x)}$$

This is a sound rule. Given f: $H \rightarrow \Sigma(A.\ B)$ we first see that for every $a \in A$ we get the morphism $a \circ !$: $|B \circ a| \rightarrow A$. Co-universality then induces a unique ϕ: $\Sigma(A.\ B) \rightarrow A$; we also have the unique $!$: $\Sigma(A.\ B) \rightarrow 1$, and we see that $B^* \circ \phi = in_1 \circ !$. Now universality of the pullback induces a unique ψ: $\Sigma(A.\ B) \rightarrow \{ A \mid B \}$, and composition yields the validation $\psi \circ f$: $H \rightarrow \{ A \mid B \}$.

{}-*elimination*: $$\frac{H \vdash A}{H \vdash \{A \mid B\}}$$

Soundness of this rule is easily seen. If we have a morphism f: H $\rightarrow$ { A | B } we merely use composition to get the validation $\sigma \circ f$: H $\rightarrow$ A.

Derivation of Syntax

Having described the class of models we consider and given a number of inhabitation rules that are proven sound, we are now in a position to *derive* a syntax of terms. This is an approach that is radically different from the PRL type theory [4] where syntax and computation are first defined, and types are introduced as secondary classifications of syntactic forms which respect the computation rules. It corresponds, however, perfectly to the philosophy of F.W. Lawvere [10], who advocates that syntax is a derived concept born from a theory out of a desire to perform calculations. This is indeed how the syntax of terms will serve us: it forms a basis for defining *computation*.

In our formalism a syntactic form is a name for a morphism. Thus when giving inference rules from which these morphisms are constructed, we can specify how the name of a resulting validation is given from the names of the assumed validations. This gives us the notion of an *annotated* inference rule, in which syntactic forms are specified:

$$\frac{S \text{ "}\sigma\text{"}}{S_1 \text{ "}\sigma_1\text{"}, S_2 \text{ "}\sigma_2\text{"}, \ldots, S_n \text{ "}\sigma_n\text{"}}$$

Here the various σ's are syntactic forms providing names for the corresponding validation morphisms. An example of such an annotation is the rule for product introduction:

$$\frac{H \vdash A \times B \text{ "}<a, b>\text{"}}{H \vdash A \text{ "}a\text{" } \quad H \vdash B \text{ "}b\text{"}}$$

Our ability to define a uniform notation for these terms highlights the constructive nature of the inhabitation rules.

A complete set of annotated inference rules is presented in the next section. This serves as a definition of the *language of terms*, TL.

Given some particular TC it is clear that every syntactic form denotes a unique morphism. It is equally clear that a morphism may be denoted by several syntactic forms; thus, we have defined a notion of *equality* of syntactic forms. This makes it possible to define a theory of equality by means of *equality statements* of the form:

$$\sigma_1 = \sigma_2$$

This statement would be *valid* iff σ_1 and σ_2 denoted equal morphisms in all TCs. The next step would be to introduce equality inference rules of the form:

$$\frac{\mathcal{E}}{\mathcal{E}_1\ \mathcal{E}_2 \ldots \mathcal{E}_n}$$

where the $\mathcal{E}$'s are equality statements. This rule is said to be *sound* if in all TCs the equalities $\mathcal{E}_1$ through $\mathcal{E}_n$ imply the equality $\mathcal{E}$. In the PRL Project such a theory of equality is reflected into type theory by introducing equality types like

$$\sigma_1 = \sigma_2 \ \mathit{in}\ \mathrm{T}$$

We hope that this will turn out to be a special case of a general method for reflecting theories into type theory.

Similarly, we can introduce a *reduction* on syntactic forms as a subrelation of the equality relation based on elimination rules. This allows us to define *normal forms* of syntactic forms, which provides us with means of regarding terms in their syntactic forms as *programs* that can be *executed.*

The language TL is variable-free, i.e. it is a language of combinators. As it is presented in this paper it is a rather weak language, but in a future version we will include recursive, infinite and quotient types. The technique by which these can be expressed in TCs is worked out in [5, 14]. This extension would provide us with a very powerful language which has been evolved from its own proof theory.

Rules with Syntax Annotation

In this section we shall summarize the inhabitation rules and present the syntax for terms.

projection: $\mathrm{T}_1, \mathrm{T}_2, \ldots, \mathrm{T}_n \vdash \mathrm{T}_i$ "$\mathbf{P}_{n,i}$"

absurd: $\mathbf{0} \vdash \mathrm{T}$ "$\mathbf{A}_\mathrm{T}$"

trivial: $\mathrm{H} \vdash 1$ "$\mathbf{T}_\mathrm{H}$"

$\times$*-introduction*: $\dfrac{\mathrm{H} \vdash \mathrm{A} \times \mathrm{B} \text{ "}\langle a, b\rangle\text{"}}{\mathrm{H} \vdash \mathrm{A} \text{ "}a\text{" } \mathrm{H} \vdash \mathrm{B} \text{ "}b\text{"}}$

$\times$*-elimination*: $\dfrac{\mathrm{H} \vdash \mathrm{A} \text{ "}m \downarrow 1\text{"}}{\mathrm{H} \vdash \mathrm{A} \times \mathrm{B} \text{ "}m\text{"}} \quad \dfrac{\mathrm{H} \vdash \mathrm{B} \text{ "}m \downarrow 2\text{"}}{\mathrm{H} \vdash \mathrm{A} \times \mathrm{B} \text{ "}m\text{"}}$

+ -*introduction*: $\dfrac{H \vdash A+B \text{ "}a\uparrow 1\text{"}}{H \vdash A \text{ "}a\text{"}}$ $\dfrac{H \vdash A+B \text{ "}b\uparrow 2\text{"}}{H \vdash A \text{ "}b\text{"}}$

+ -*elimination*: $\dfrac{H \vdash C \text{ "}\delta(s; f, g)\text{"}}{H \vdash A+B \text{ "}s\text{" } H, A \vdash C \text{ "}f\text{" } H, B \vdash C \text{ "}g\text{"}}$

→-*introduction*: $\dfrac{H \vdash A \rightarrow B \text{ "}\kappa(f)\text{"}}{A, H \vdash B \text{ "}f\text{"}}$

→-*elimination*: $\dfrac{H \vdash B \text{ "}f(a)\text{"}}{H \vdash A \rightarrow B \text{ "}f\text{" } H \vdash A \text{ "}a\text{"}}$

inclusion $T \in \mathbf{U}_k$: $H \vdash_{k+1} [\mathbf{U}_k] \text{ "}\mathbf{I}_T\text{"}$

embedding: $\dfrac{H \vdash_{i+j} T \text{ "}\mathbf{E}^j(m)\text{"}}{H \vdash_i T \text{ "}m\text{"}}$

split: $\dfrac{H \vdash x{:}A, B(x) \text{ "}\phi\uparrow a\text{"}}{\vdash A \text{ "}a\text{" } H \vdash \Phi \text{ "}\phi\text{"}}$

Σ-*introduction*: $\dfrac{H \vdash \Sigma(A.\ B) \text{ "}\Sigma(m)\text{"}}{H \vdash x{:}A, B(x) \text{ "}m\text{"}}$

Σ-*elimination*: $\dfrac{H \vdash C \text{ "}\Delta(s; f)\text{"}}{H \vdash \Sigma(A.\ B) \text{ "}s\text{" } H, x{:}A \vdash B(x) \rightarrow C \text{ "}f\text{"}}$

Π-*introduction*: $\dfrac{H \vdash \Pi(A.\ B) \text{ "}\Pi(m)\text{"}}{H, x{:}A \vdash B(x) \text{ "}m\text{"}}$

Π-*elimination*:
$$\frac{H \vdash C \text{ "}\nabla(p; f)\text{"}}{H \vdash \Pi(A.\ B) \text{ "}p\text{" } H \vdash x{:}A,\ B(x) \to C \text{ "}f\text{"}}$$

{}-*introduction*:
$$\frac{H \vdash \{A \mid B\} \text{ "}\lfloor m \rfloor\text{"}}{H \vdash x{:}A,\ B(x) \text{ "}m\text{"}}$$

{}-*elimination*:
$$\frac{H \vdash A \text{ "}\lceil m \rceil\text{"}}{H \vdash \{A \mid B\} \text{ "}m\text{"}}$$

Apart from these we need a number of trivial rules to actually carry out interesting proofs.

permute:
$$\frac{H_1, \ldots, H_n \vdash T \text{ "}\sigma(m)\text{"}}{H_{\sigma(1)}, \ldots, H_{\sigma(n)} \vdash T \text{ "}m\text{"}}$$

where σ is any permutation of the numbers 1, 2, ..., n.

left thin:
$$\frac{H, G \vdash T \text{ "}\mathbf{L}(m)\text{"}}{G \vdash T \text{ "}m\text{"}}$$

right thin:
$$\frac{H \vdash T \text{ "}\mathbf{R}(m)\text{"}}{H \vdash T, S \text{ "}m\text{"}}$$

These rule are obviously all sound.

Our syntax is quite different from from the terms in the PRL type theory, but most places it is just a matter of notational style. A representative term in our language is

$$<<\mathbf{T}_1, \mathbf{T}_1 \uparrow 2> \uparrow 1, \kappa(\mathbf{A}_{1 \times 0})>$$

which is a witness for the inhabitation of the type $(1\times(1+1)+0)\times(0\rightarrow(1\times 0))$. This term would be very hard to write down without a proof as guidance, but this merely emphasizes the central role of proofs in the programming process.

Conclusions

The results in this paper may be viewed as a unification and generalization of category theoretic formalizations of logic [7, 11], proof theory [18], and computational formalisms, specifically λ-calculus [3, 9, 15] and combinatory logic [6]. Our major point of departure has been to use the topos theoretic notion of membership, namely the existence of a morphism from the terminal object, as the meaning of inhabitation. Thus we do not model truth values as arrows in a category. The constructive nature of type theory allows each proof of inhabitation of a type to correspond to an explicit witness for that type.

Much work remains to be done. We are in the process of giving fully internalized definitions of our various constructions. We are also interested in seeking topological type categories so that we may build models that express computational approximation, a concept that is the key to Scott's development of programming language semantics. We are actively investigating how reflections of theories into type theory can be formalized by using suitable embedding functors. The reflection of equality reasoning in the vPRL version of type theory is an example of such a process [4]. Finally, we are seeking category theoretic proofs of interesting meta-mathematical facts; for example, we are examining the question of mimicing the proof that Martin-Löf's original type theory is inconsistent.

To summarize our results, we have:

- given a category theoretic semantics for inhabitation of a type and for type constructions;
- shown how proof rules may be justified using standard category theoretic constructions;
- shown how terms appear as morphisms and syntax appears as a notation for naming morphisms;
- modelled equality of terms as identity of morphisms and computation with terms as taking place through such equalities;
- given a uniform construction for building higher universes by using functor categories.

Acknowledgements

We have benefitted from discussions with members of the PRL group at Cornell, especially Robert Constable and Nax Mendler. The first author would like to thank Larry Paulsen and Pino Rosolini for stimulating discussions. Thanks also goes to Anne Rogers for proofreading this manuscript.

References

[1] **Arbib, M.A., Manes, E.G.** *Structures, Arrows and Functors. The Categorical Imperative.* Academic Press 1975.

[2] **Barendregt, H.** *Lambda Calculus and its Models.* In Proc. Logic Colloquium '82. North-Holland 1984.

[3] **Breazu-Tannen, V., Mayer, A.R.** *Lambda Calculus with Constrained Types.* In Proc. Logic of Programs, Lecture Notes in Computer Science Vol. 193. Springer-Verlag 1985.

[4] **Constable, R.L.** et al. *Implementing Mathematics in the NuPrl System.* To appear. Prentice-Hall 1986.

[5] **Constable, R.L., Mendler, N.P.** *Recursive Definitions in Type Theory.* In Proc. Logic of Programs, Lecture Notes in Computer Science Vol. 193. Springer-Verlag 1985.

[6] **Curien, P.-L.** *Categorical Combinatory Logic.* In Proc. ICALP '85. Lecture Notes in Computer Science Vol 194. Springer-Verlag 1985.

[7] **Goldblatt, R.** *Topoi: The Categorical Analysis of Logic.* Studies in Logic and the Foundations of Mathematics Vol 98. North-Holland 1979.

[8] **Herrlich, H., Strecker, G.E.** *Category Theory: An introduction.* Sigma Series in Pure Mathematics 1. Heldermann Verlag 1979.

[9] **Lambek, J.** *From λ-calculus to Cartesian Closed Categories.* In To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press 1980.

[10] **Lawvere, F.W.** *Introduction.* In Toposes, Algebraic Geometry and Logic. Lecture Notes in Mathematics Vol 274. Springer-Verlag 1972.

[11] **Makkai, M., Reyes, G.E.** *First Order Categorical Logic.* Lecture Notes in Mathematics Vol 611. Springer-Verlag 1977.

[12] **Martin-Löf, P.** *An Intuitionistic Theory of Types: Predicative Part.* In Proc. Logic Colloquium '73. North-Holland 1975.

[13] **Martin-Löf, P.** *Constructive Mathematics and Computer Programming.* In Proc. 6th International Congress for Logic, Method and Philosophy of Science. Hanover 1979.

[14] **Mendler, N.P., Panangaden, P., Constable, R.L.** *Lazy Types in Type Theory*. In preparation.

[15] **Scott, D.S.** *Relating Theories of the λ-calculus*. In To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism. Academic Press 1980.

[16] **Smyth, M.B., Plotkin, G.D.** *The Category Theoretic Solution of Recursive Domain Equations*. Siam Journal of Computing Vol 11 No 4. 1982.

[17] **Stenlund, S.** *Combinators, λ-terms and Proof Theory*. D.Reidel Publishing Company 1972.

[18] **Szabo, M.E.** *Algebra of Proofs*. Studies in Logic and the Foundations of Mathematics Vol 88. North-Holland 1979.