A Category Theoretic Analysis of Predicative Type Theory

Michael I. Schwartzbach

TR 86-793
(Ph.D. Thesis)

December 1986

Department of Computer Science
Cornell University
Ithaca, NY 14853

A CATEGORY THEORETIC ANALYSIS OF PREDICATIVE TYPE THEORY

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Michael Ignatieff Schwartzbach

January, 1987

Biographical Sketch

Michael Ignatieff Schwartzbach was born on the 15th of August 1961 in Copenhagen, Denmark. A few years later he (was) moved to Århus, Denmark where he started his academic career by reading monosyllabic prose and molding clay figurines. The demands on him soon intensified; eventually Michael could spell the word *monosyllabic* and was allowed to leave high school. His great love for empty formalism in 1979 prompted him to begin studying mathematics and computer science at Aarhus University; his great production of empty formalism in 1984 prompted the university to award him a Cand.Scient. degree. Impressed by the grandiloquence of American academic titles, Michael enrolled at Cornell University to further pursue the study of computer science; he was pleased to find himself a Master of Science in the summer of 1986. Michael sincerely hopes that twenty years in school will prove sufficient.

Dedication

To Noulé.

Acknowledgements

First of all, I want to thank Prakash Panangaden for being an excellent friend and advisor from the first day we both came to Cornell. The success of this project is largely due to his optimism and inspiration. Secondly, I am grateful to Robert Constable for giving me a reason to study type theory and for showing me encouragement whenever I needed it. I thank Richard Shore for serving patiently on my committee.

I have had many interesting discussions with my collegues at Cornell. In particular, Nax Mendler has been very helpful; I thank him for teaching me type theory and for spending time on me that he should have used writing his own thesis. Charles Elkan showed his better qualities by helping me with LaTeX.

I am indebted to my many friends at Cornell for entertaining me during my stay; in particular, my playmates around the office provided lots of fun. Anne Rogers deserves special praise for putting up with me during the many two-dwarf days (Grumpy and Sleepy) I suffered while writing my thesis.

Finally, I am grateful to Erik Meineche Schmidt, Aarhus University, and the Danish Research Council for making my stay at Cornell possible in the first place.

Table of Contents

List of Figures

Chapter 1

Introduction

This thesis gives a category theoretic analysis of predicative type theories. In this chapter we shall give a brief introduction to the concept of type theories and summarize our methods and results.

1.1 What are Type Theories?

The word *type* has acquired numerous different meanings in computer science and logic; it is sometimes hard to see any connection between various accepted usages. The notion of type one usually thinks of in the context of formal systems is used to classify and restrict the terms of a theory; in particular, we want to consider computational theories.

The purpose of restriction is to obtain guarantees of certain semantic properties. In logic one typically wants to rid oneself of logical paradoxes. Bertrand Russell is credited for the first use of types, when he defined his version of set theory. The paradox he fought arose through *impredicativity*. A definition is deemed impredicative when it employs structures that have not yet been fully

defined and which include the structure being defined, e.g. *the set of all sets* is an impredicative concept. In logic the problem arises with quantification. Russell defined a so-called *predicative* hierarchy of sets, where the sets at any level could only be constructed of sets defined at previous levels.

In programming languages one important semantic property is the absence of run time errors. Traditional programming languages were designed with this in mind. For example, the PASCAL language allows the programmer to assign types to variables and procedures, which impose restrictions on their use. These restrictions guarantee the absence of obvious misuses. Another vital semantic property is *termination*, which is not adressed by the type systems of most programming languages. The problem is that in PASCAL one can assign a type to *any* procedure — even one that diverges. The type theories we are interested in do not type all diverging programs.

It is well-known that no computational theory can capture all the total recursive functions and nothing else. The usual way around this is to include all the partial recursive functions and argue that individual programs are actually total. Another approach is to include only a subset of the total recursive functions and argue that the selection is sufficiently large; this is the motivation behind type theories. The classic example of this process comes from the λ-calculus.

The untyped λ-calculus [2,11] presents exactly the partial recursive functions. By adding a system of finite types one obtains the (simple) typed λ^{τ}-calculus, which contains no diverging terms. Unfortunately, in this system one can only represent extended polynomials, which is a sadly small class of computable functions. An entire branch of type theory has been devoted to capturing larger and

larger classes of total recursive functions as typed λ-terms. This is done by employing ever more complex and sophisticated type systems and by adding typed term constants. By merely adding an iteration combinator to the λ^{τ}-calculus, one can represent the functions definable through primitive recursion. By adding recursion combinators for all types one obtains a far larger class of functions.

On the forefront of this process we find the second-order typed Λ^{τ}-theory [8]. This is an extension of typed λ-calculus, where types can serve as parameters to terms. On top of this is built the higher-order *Theory of Constructions* [6], which was intended to serve as a foundation for constructive mathematics. It is an impredicative theory, but it is still consistent. Per Martin-Löf's earliest theory was impredicative and inconsistent. One walks a delicate balance between impredicativity and inconsistency.

Another main branch of type theory was started with the introduction of Per Martin-Löf's *intuitionistic theory of types* [15], which was also motivated by the desire to define a foundation for constructive mathematics. It has roots in common with the original attempt by Church to use the λ-calculus as a basis for logic, but it is unique in using Rusell's predicative type hierarchy to form higher-order universes of types. It has been studied and extended by the νPrl project headed by Robert Constable [4].

1.2 Martin-Löf's Theory

The νPrl theory (and our work) gets its philosophical ballast from Martin-Löf's theory of types. We feel it is appropriate to outline its main features.

Types are expressions of a constructive approach to forming collections, i.e. they

are analogous to sets. Following Bishop's conception of sets, a type consists of *elements* and an *equality relation* between elements. To define a type one gives a way of recognizing its members and a way of determining when two elements are equal. Elements are represented by *terms*, which are divided into two classes: *canonical* and *non-canonical*. Canonical terms are the irreducible elements, they name the values, whereas non-canonical terms are expressions that denote values. The computational element enters as a method of *reduction*, through which non-canonical terms may be transformed into equivalent canonical terms. A term is deemed *typeable* when it can be reduced to a canonical term in some type; it is possible for a typeable term to contain untypeable subterms.

Types may be composed through *type constructors*. These are presented in a very uniform scheme defining the *formation* of a new type, the *introduction* of canonical terms, the *elimination* (or *analysis*) of terms, and the *equality* between terms. We can illustrate these concepts through a simple example: the *product type* constructor. If A and B are types we posit that we can form the product type $A \times B$. If a is a term of type A and b is a term of type B we introduce $< a, b >$ as a canonical term of type $A \times B$. If t is a term of type $A \times B$ we give the two elimination forms $prl(t)$ and $prr(t)$ as non-canonical terms of type A respectively type B. These may be transformed into canonical terms through the two reductions: $prl(< a, b >) = a$ and $prr(< a, b >) = b$. Finally, we say that two canonical terms $< a, b >$ and $< a', b' >$ are equal exactly when a and a' are equal terms of type A and b and b' are equal terms of type B.

A key point in this approach is that types may be viewed as *propositions* and their terms as *proofs*. Thus, a proposition is true exactly when its corresponding type is inhabited by some term. The product type serves as conjunction in

this context; the sum type corresponds to intuitionistic disjunction. This gives a natural intuitionistic logic and is the basis for the *propositions-as-types* and *truth-as-inhabitation* slogans, where a proposition is realized as a type that is inhabited exactly when the proposition is provable. In this setting, implication is a mechanism for transforming proofs; a proof of $A \Rightarrow B$ maps proofs of A to proofs of B, i.e. it is an element of the function type $A \rightarrow B$. The important Σ and Π type constructors are the type equivalents of existential and universal quantifiers. For example, the elements of $\Sigma(x \in A, B(x))$ are pairs of the form $< a, B(a) >$ where a is an element of A; this just means that a proof of an existential $\exists(x \in A, B(x))$ consists of a witness a and a proof of $B(a)$.

1.3 Categorical Theories

The above mentioned examples of type theories arose in much the same manner. Starting with a computational theory, most often based on the λ-calculus, one annotated the terms by types and restricted the reduction rules to fit this framework. Hence, the typed terms form a subset of the full set of computational terms. In the theories inspired by Martin-Löf it is possible for typed terms to contain untypeable subterms; this can be considered an undesirable feature, as it makes it almost impossible to give a compositional theory of meaning. There is, however, a completely different approach to defining type theories.

This method is extensively treated by Lambek and Scott [13]. Given any class of categories with some amount of structure one can define its *internal language*, which is essentially a syntactic representation of the objects and morphisms that are forced to exists in *all* of these categories. This may be viewed as a type theory, where objects act as types and morphisms act as terms. By proving soundness of certain reduction rules one obtains a computational theory; an important dif-

ference is that no untyped terms exist. The classes of categories considered by Lambek and Scott [13] are cartesian categories, cartesian closed categories, and toposes, with or without natural numbers objects.

Sometimes these two methods coincide. It is a well-known result that the internal language of a cartesian closed category is exactly the simple typed λ-calculus [12,7]. In other cases the similarities are deceptive; the internal language of a topos is called *intuitionistic type theory* [13]. It is, however, very hard to compare to the Martin-Löf theories, but it is certainly quite different. A closer match is found in the work of Seely [18], where locally cartesian closed categories are shown to give rise to parts of the standard Martin-Löf type theory, but without the predicative hierarchy or the computational aspects.

An advantage of the categorical approach is an instant model theory; any category in the class selected is a model of the corresponding type theory. Also, it seems that one makes fewer arbitrary choices when defining the type structure, but that may just be an illusion brought on by the familiarity of certain classes of categories.

Our goal is to derive a hybrid approach and obtain a type theory that is categorically inspired and which captures the eloquence of the νPrl theory, including the full predicative hierarchy and the plentitude of type constructors. When defining our class of categories, we are willing to import an external computational theory, but the typed terms of our system will *not* be a subset of the computational terms and *no* typed term will contain untyped subterms; furthermore, the meaning of terms and types is given in a purely compositional manner. We view this as a cleaner theory than Martin-Löf's original.

1.4 Summary

In this section we outline our categorical approach. The basic realization that sparked this project was that any universe level of the νPrl theory could be viewed as a category. The objects are the types and the morphisms are the typed terms. Clearly, composition is associative and there is an identity morphism for every object. This would have been an only mildly interesting observation if we had not further realized that some of the fundamental type constructions could be expressed as standard categorical constructions. For example, the type *Void* is the initial object, and the product and sum constructions are the categorical product and coproduct. With even greater interest we observed that the Π and Σ constructors looked like large products and coproduct of infinite discrete diagrams. One by one, the different type constructors revealed their categorical nature. It became appearant that the connection was very strong.

We wanted to demonstrate that it is possible in categories with sufficient structure to emulate a type theory that contains the fundamental properties of the νPrl theory, i.e. the same type constructors and the predicative hierarchy.

The first problem was to find the appropriate class of categories. We seemed to need something larger than a topos, since we had to construct limits and colimits of infinite diagrams. Since countable diagrams clearly would be sufficient we thought of countably bicomplete categories. This, however, gave rise to some problems. The syntax of our type theory should arise as the *internal language* of our categories, and with uncountably many diagrams we would be hard pressed to get a canonical finitary syntax. It was clear though that we would never need a diagram that could not be obtained as the range of a partial recursive function. This was the final insight needed.

It remained a problem to classify such categories. Completeness is most often restricted by cardinality, or some specific set of diagram shapes is given. It is not possible to capture recursively enumerable diagrams in this manner. We decided to axiomatize the internal language directly and define our class of categories as those in which this language could be sensibly interpreted. This is less backwards than it may seem; the class of cartesian closed categories is really equationally presented (as are most others).

We have to import some external notion of computation to define the r.e. diagrams, but it has no importance which exact syntax we choose. What we ended up with is a method that, given a set of *postulated* objects and morphisms, constructed a *term category* with an appropriate amount of structure. The countably many objects and morphisms of this category are equivalence classes of terms. A term category is finitely bicomplete, and has exponents and a subobject classifier, i.e. it is a topos. But beyond that, it is what we could call *r.e. bicomplete*, that is every r.e. diagram has a (co)limit that is (co)universal among all r.e. (co)cones. This, for example, immediately gives us a natural numbers object as the colimit of the diagram $1,\ 1+1,\ 1+1+1, \ldots$ A natural numbers object is often explicitly introduced to provide recursion on arbitrary types. Term categories are analogous to term algebras in many ways; for example, a term algebra may be viewed as an actual algebra or as a language describing a class of algebras.

Given the term category construction it is easy to form a predicative hierarchy of categories that can mimic the universe hierarchy of νPrl. The lowest level is given as the term category with no constants. A sucessor level is obtained by explicitly introducing a universe object whose elements are encodings of the

objects in the previous level, and then applying the term category construction again to this (large) set of constants. This would not have worked if we had used countably bicomplete categories.

Having constructed this inclusive hierarchy of term categories we are left wondering if they perhaps all are trivial, i.e. if all the objects are isomorphic. We can rule out this possibility by obtaining a structure preserving functor from each term category into the category of sets and functions and, thus, achieve a consistency result. This construction is inductive and relies on the natural well-foundedness of the term categories.

We can now show how the type constructors of the νPrl theory can be emulated in these categories, mostly as a simple limit or colimit of some diagram. In this manner, we develop the propositional type constructors: product, sum, and exponent, the Σ and Π constructors, subtypes and quotient types, and the constructors for recursive and infinite types. Furthermore, we can propose *any* diagram as a potential type constructor, and we give generic introduction and elimination rules for such types. A standard categorical result then tells us that this plentitude of type constructors may be reduced to just four, two of which are new to type theories (but which can be obtained from the existing ones).

Chapter 2

Categorical Type Theory

Categories are out. Inventors of categories shall find blight .

— Linda Ronstadt .

This chapter describes the syntax and the equality relations of our *categorical type theory.* As its construction is heavily inspired by standard category theoretic concepts, we will first provide a brief introduction to category theory. This exposition is limited to the concepts we shall need and is primarily intended to establish our terminology; a comprehensive introduction and tutorial may be found in [1,10,14,13]. We then give a general rule based method for constructing relations of terms; the method is similar to algebraic specifications and to first-order theories. The relations we can construct are all recursively enumerable. We apply this method to construct a *term category*, which is essentially a semantic meta-language describing a class of categories. With this in hand we proceed to present our predicative type theory as an inclusive hierarchy of term categories.

2.1 Concepts from Category Theory

Definition 2.1.1 *A* **category K** *consists of a class* $Obj(\mathbf{K})$*, called the* **objects** *of* $\mathbf{K}$*, together with, for each pair* A, B *of objects of* $\mathbf{K}$*, a set* $\mathbf{K}(A,B)$*, called the* **morphisms** *of* $\mathbf{K}$ *from* A *to* B*. We write* $f : A \rightarrow B$ *to indicate that* f *is a morphism from* A *to* B*. We require that the following conditions are satisfied:*

For any three objects $A, B,$ *and* C *of* $\mathbf{K}$*, there is a composition operation* $\cdot : \mathbf{K}(A,B) \times \mathbf{K}(B,C) \rightarrow \mathbf{K}(A,C)$ *which satisfies the associativity axiom*

$$f \cdot (g \cdot h) = (f \cdot g) \cdot h$$

For any object A *of* $\mathbf{K}$*, the set* $\mathbf{K}(A,A)$ *contains a special morphism* Id_A *that satisfies the identity axioms*

$$f \cdot Id_A = f, \quad Id_A \cdot g = g$$

There are numerous examples of categories in the literature. The canonical example is *SET*, where the objects are all sets and the morphisms from A to B are the functions between the sets. A contrasting example is any poset $(P, \sqsubseteq)$, where the objects are the elements of P and the morphisms from p to q is the singleton set $\{\bullet\}$ if $p \sqsubseteq q$ and $\emptyset$ otherwise. We can capture most of the usual set theoretic notions in the categorical setting and, thus, generalize them to apply to other categories. This work has been taken furthest through the study of *toposes* [9,13].

Definition 2.1.2 *An* **isomorphism** *between two objects* A *and* B *in a category is a pair of morphisms* $f : A \rightarrow B$ *and* $g : B \rightarrow A$ *such that* $g \cdot f = Id_A$ *and* $f \cdot g = Id_B$*. A category in which all objects are isomorphic is called* **degenerate**.

In *SET* this just establishes a bijection; in a poset only identical elements are isomorphic.

At this point it is worth pointing out some concepts that are often employed in discussions of category theory. *Duality* is used both formally and informally to describe situations that differ only in that the direction of morphisms is reversed. For instance, cones and cocones described below are dual concepts. Some concepts are self-dual, e.g. isomorphisms. An object A is deemed *universal* when it is special amongst objects with a certain property in the manner that whenever any other object B has the same property, there is induced a *unique* morphism from B to A making a certain diagram commute. For this reason universal objects are unique up to isomorphism, since uniqueness forces the induced morphisms between them to be isomorphisms. The dual situation is called *couniversality*. As described below, limits are universal and colimits are couniversal.

We can define a very general notion of constructions in categories. This requires some initial machinery.

Definition 2.1.3 *A* **diagram** D *in a category* $\mathbf{K}$ *is a directed graph* (V_D, E_D) *where* $V_D \subset Obj(\mathbf{K})$ *and* $E_D(A,B) \subset \mathbf{K}(A,B)$. *The diagram is said to* **commute** *if the compositions of morphisms along any two paths with the same end-points yield the same results.*

Thus, a diagram is a directed graph built out of objects and morphisms. In a poset diagrams always commute; this is not the case for *SET*.

Definition 2.1.4 *A* **cone** $\triangle$ *over a diagram* D *in a category* $\mathbf{K}$ *is a family of morphisms* $m_A^{\triangle} : C \to A$, *for some fixed* $O(\triangle) \in Obj(\mathbf{K})$ *and all objects* $A \in V_D$, *such that for every* $f \in E_D(A,B)$ *we have* $f \cdot m_A^{\triangle} = m_B^{\triangle}$. *Dually, a* **cocone** $\triangledown$ *over* D *is a family of morphisms* $m_A^{\triangledown} : A \to O(\triangledown)$, *for some fixed* $O(\triangledown) \in Obj(\mathbf{K})$ *and all objects* $A \in V_D$, *such that for every* $f \in E_D(A,B)$ *we have* $m_A^{\triangledown} = m_B^{\triangledown} \cdot f$.

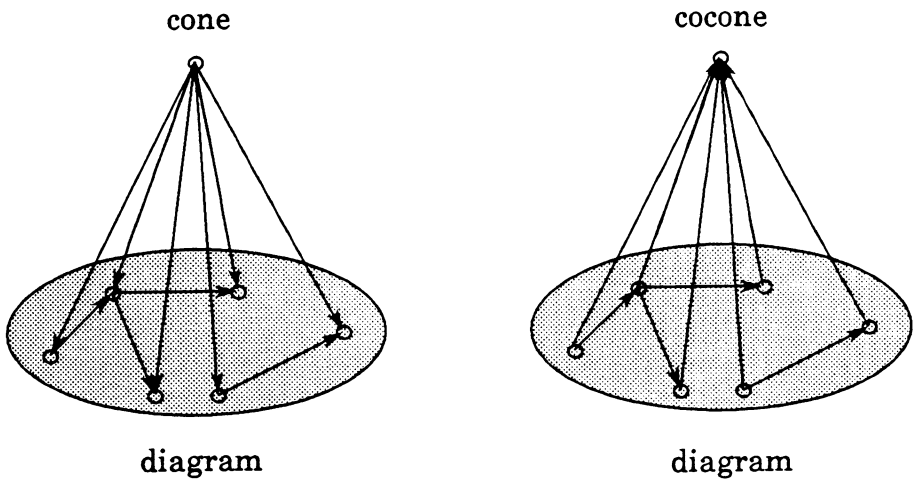

Figure 2.1: Cones and Cocones

Cones and cocones are sketched in figure 2.1.

Definition 2.1.5 *A* **limit** *over a diagram* D *in a category* $\mathbf{K}$ *is a cone* Δ *over* D *such that for any other cone* Δ' *over* D *there is induced a* **unique** *morphism* $\mu(\Delta', \Delta) : O(\Delta') \rightarrow O(\Delta)$ *such that for all* $A \in V_D$: $m_A^{\Delta'} = m_A^{\Delta} \cdot \mu(\Delta', \Delta)$. *Dually, a* **colimit** *over* D *is a cocone* ∇ *over* D *such that for any other cocone* ∇' *over* D *there is induced a* **unique** *morphism* $\mu(\nabla, \nabla')$ *such that for all* $A \in V_D$: $m_A^{\nabla'} = \mu(\nabla, \nabla') \cdot m_A^{\nabla}$.

These concepts are sketched in figure 2.2. Limits and colimits, which are clearly unique up to isomorphism, produce many of the interesting constructions in different categories. In *SET* limits and colimits of arbitrary diagrams exist; in a poset they correspond to the infimum respectively the supremum of the nodes of the diagram and, thus, may or may not exist. As an example we can look at the binary discrete diagram $(\{A, B\}, \emptyset)$, i.e. the graph with two nodes and no edges. In *SET* the limit of this diagram is the cartesian product $A \times B$ and the colimit is the disjoint union $A + B$; in a poset the limit is the greatest lower bound $A \sqcap B$

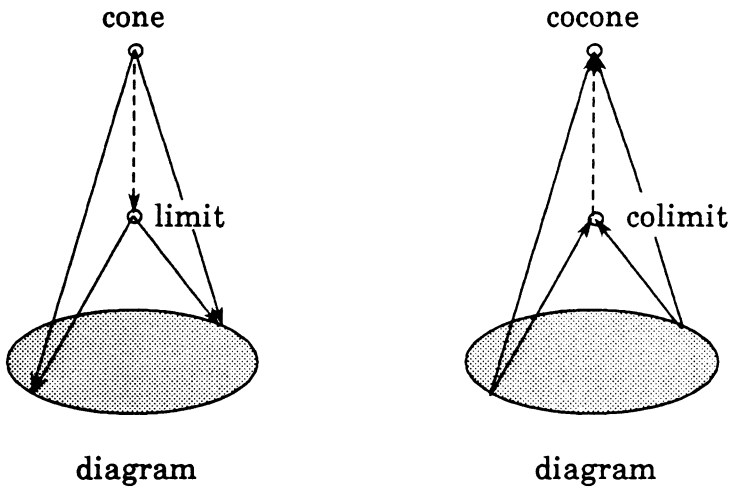

Figure 2.2: Limits and Colimits

and the colimit is the least upper bound $A \sqcup B$. If D is a collection of diagrams in a category **K**, we say that **K** is D-**complete** if the limits of all diagrams in D exist in **K**. Similarly, **K** is D-**cocomplete** if all colimits exist. **K** is called D-**bicomplete** if it is D-complete and D-cocomplete.

Definition 2.1.6 *The limit of the empty diagram is called the* **final** *object 1. The colimit of the empty diagram is called the* **initial** *object 0.*

Hence, for any object A there are unique arrows $! : A \to 1$ and $! : 0 \to A$. In *SET* the initial object is $\emptyset$ and the final object is a singleton set. In a poset the initial object is $\bot$ and the final object is $\top$, if they exist.

We can use the final object to define a concept of elements of objects, as it is done in topos theory.

Definition 2.1.7 *An* element *of an object A is a morphism from* 1 *to A.*

The elements of a *SET*-object is just its set-theoretic elements; in a poset only $\top$ has an element.

We need to introduce a novel extension of these concepts and define constructions that are *almost* limits and colimits.

Definition 2.1.8 *Let ϕ be some property of cones and D a diagram. Call the cones for which ϕ holds the* ϕ-**cones**. *Then a* ϕ-**limit** *of D is a universal ϕ-cone over D.*

Clearly, if $\phi_1 \Rightarrow \phi_2$, then a ϕ_2-limit is also a ϕ_1-limit; hence, a limit is a ϕ-limit for any ϕ. We can give a similar definition of *ϕ-cocones.*

One very important construction does not arise as the limit or colimit of any diagram. It is, however, defined as an object with a certain universal property.

Definition 2.1.9 *If A and B are objects we define their* **exponent** B^A *to be an object with a special morphism $eval : A \times B^A \rightarrow B$, such that whenever we have an object C and a morphism $f : A \times C \rightarrow B$ there is induced a* **unique** *morphism $curry(f) : C \rightarrow B^A$ such that $(Id_A \times curry(f)) \cdot eval = f$.*

In *SET* the exponent B^A is just the set of functions from A to B; the eval-map is application and the curry-map is just that. A poset that is finitely bicomplete and has exponents is called a *Heyting algebra* [9]. The existence of finite bilimits state that finite sets have *least upper bounds* and *greatest lower bounds.* The exponent of two elements a and b is called the *relative pseudo-complement* and is the greates element of the set $\{x : a \cap x \sqsubseteq b\}$.

Category theory arose as a framework for relating different topics in mathematics, e.g. algebra and topology. This is done by defining a concept of mappings between categories.

Definition 2.1.10 *A* **covariant functor** F *from* K_1 *to* K_2 *maps objects to objects and morphisms to morphisms such that for any object* $A \in Obj(K_1)$ *we have*

$$F(Id_A) = Id_{F(A)}$$

and for any $A, B \in Obj(K_1)$ *and* $f, g \in K_1(A, B)$ *we have*

$$F(f \cdot g) = F(f) \cdot F(g)$$

A **contravariant functor** *is defined similarly, except that is satisfies the equation*

$$F(f \cdot g) = F(g) \cdot F(f)$$

A function that embeds a well-founded poset into the powerset of some set may be viewed as a covariant functor from that poset to *SET*.

We are often interested in functors with special properties.

Definition 2.1.11 *A functor that is injective on objects and morphisms is called an* **embedding functor**. *A functor is said to* **preserve** *limits, colimits or exponents if it commutes with their construction, e.g.* F *preserves exponents iff* $F(B^A) = F(B)^{F(A)}$, $F(eval \cdot (a \times g)) = eval \cdot (F(a) \times F(g))$, *and* $F(curry(f)) = curry(F(f))$.

The previous example functor is an embedding that preserves exponent, limits, and colimits.

Definition 2.1.12 *A* **monic** *is a morphism* f *that is left-cancellable; that is, for any two morphisms* h *and* k *if* $f \cdot h = f \cdot k$ *then* $h = k$. *Dually, an* **epic** *is a morphism that is right-cancellable, so that for any two morphisms* h *and* k *if* $h \cdot f = k \cdot f$ *then* $h = k$.

In *SET* a monic is an injection and an epic is a surjection; in a poset all morphisms are monics and epics. Quite often we will not have monics when we need them, but we can obtain the monic part of any morphism in the following manner.

Definition 2.1.13 *An* **epi-mono factorisation** *of a morphism* $f : A \to B$ *is an object* $f(A)$*, a monic* $im(f) : A \to f(A)$*, and an epic* $f^* : f(A) \to B$*, such that* $f^* \cdot im(f) = f$*. Factorizations are unqiue up to isomorphism and always exist in finitely bicomplete categories.*

In *SET* $f(A)$ is the set $\{f(a) \mid a \in A\}$, $im(f)$ is the obvious inclusion and f^* is f restricted to $f(A)$.

The final concept we need allows us to form subcollections in categories. It uses the traditional comprehension principle expressed categorically.

Definition 2.1.14 *A* **subobject classifier** *is an object* Ω *with a special morphism* $\top : 1 \to \Omega$*, such that for any monic* $f : A \to B$ *there is induced a* **unique** *morphism* $\chi_f : B \to \Omega$ *such that* A *is the limit of the diagram with morphisms* $\top : 1 \to \Omega$ *and* $\chi_f : B \to \Omega$*, and the limit morphism from* A *to* B *is* f*.* χ_f *is called the character of* f*. We define* $\bot = \chi_! : 1 \to \Omega$*, where* $! : 0 \to 1$*. We call* $\top$ *and* $\bot$ **truth values** *of* Ω*.*

In *SET* the subobject classifier is the set $\{T, F\}$ and the character of a monic f is the characterictic function of $im(f)$. In a poset the subobject classifier is not a meaningful concept. Notice that for all monics f we have that $\chi_f = \chi_{im(f)}$.

A *topos* is a finitely bicomplete category with exponents and a subobject classifier. In a topos one can interpret notions such as *sets*, *predicates*, and *logic connectives*. The object Ω is seen as a collection of truth values; in particular, it has the elements $\top$ and $\bot$, but there may be many more. It is possible to

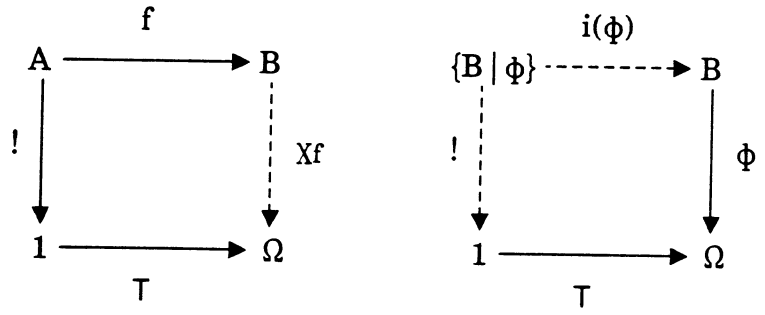

Figure 2.3: Characterization and Comprehension in a Topos

define logic connectives in this setting; for example, conjunction is a morphism $\wedge : \Omega \times \Omega \rightarrow \Omega$, such that $\wedge \cdot < \top, \top >= \top$. It can be defined as the character of the monic $< \top, \top >: 1 \rightarrow \Omega \times \Omega$. Objects can be interpreted as sets; their elements are the categorical elements, i.e. the morphisms from 1. A predicate on an object $\boldsymbol{A}$ is interpreted as a morphism $\boldsymbol{A} \rightarrow \Omega$. An object $\boldsymbol{A}$ is a *subobject* of an object $\boldsymbol{B}$ iff there exists a monic $f : \boldsymbol{A} \rightarrow \boldsymbol{B}$; we may think of f as the inclusion of $\boldsymbol{A}$ into $\boldsymbol{B}$. The key observation, and the fact that we will be using, is the duality between *characters* and *subobjects*, i.e. any subobject of $\boldsymbol{B}$ yields a unique character on $\boldsymbol{B}$, and every character on $\boldsymbol{B}$ yields a unique (up to isomorphism) subobject of $\boldsymbol{B}$. This is summed up in the topos theoretic *comprehension principle*, which is illustrated in figure 2.3. Given any subobject $f : \boldsymbol{A} \rightarrow \boldsymbol{B}$ we obtain its character $\chi_f : \boldsymbol{B} \rightarrow \Omega$; given any predicate $\phi : \boldsymbol{B} \rightarrow \Omega$ we obtain the subobject $\{\boldsymbol{B} \mid \phi\}$ and the *inclusion* monic $i(\phi) : \{\boldsymbol{B} \mid \phi\} \rightarrow \boldsymbol{B}$. The character is unique by definition of the subobject classifier and the subobject is unique up to isomorphism, since it is a limit. This gives us the fundamental connection, viz. $\chi_{i(\phi)} = \phi$. It is now possible to define the usual set operations, such as intersection, union, and complement on subobjects. The method is very simple: to obtain the intersection of two subobjects we form the comprehension of the

conjuction of their two characters. A further development makes it possible to handle also quantifiers. The logic of a topos is in general *intuitionistic*, i.e. the law of excluded middle does not hold. This is true in e.g. the topos SET^2 of pairs of sets and functions; the topos SET is classical. This approach is quite different from the *propositions-as-types* philosophy, where actual truth values are absent and a proposition is viewed as the collection of its proofs; our work will contain a categorical version of the propositions-as-types method.

2.2 Relational Closures

This section presents a rule based method of defining relations over terms in a language. We shall need this tool to define the terms of our theory. The method seems quite versatile and is presented in its fullest generality. It is simpler than and similar to the definition of first-order languages. We present a simple sequent calculus of relations of strings over some alphabet. Next we introduce inference rules and proofs. Finally, the closure is defined as the sets of tuples for which membership can be proven. The main point of this technique is to formalize what is meant by saying that a system obeys certain rules.

We assume that we are given a collection of *relation symbols* of various arities $\{R_1, R_2, \ldots\}$, some alphabet Γ, and a set of variables X. We call the strings in $\Gamma^\star$ *terms* and the strings in $(\Gamma \cup X)^\star$ *terms with variables.*

Definition 2.2.1 *A* **structure** Θ *assigns for each* n*-ary relation symbol* R *a concrete relation* $\Theta[R] \subseteq (\Gamma^\star)^n$. *Structures are ordered by inclusion, where we say* $\Theta_1 \subseteq \Theta_2$ *if for all* R *we have* $\Theta_1[R] \subseteq \Theta_2[R]$.

We now present an inference system that allows us to reason about strings belonging to relations.

Definition 2.2.2 *A* **sentence** *is an n-ary relation symbol and n terms with variables*

$$R(t_1, t_2, \ldots, t_n)$$

A **sequent** *is a pair consisting of a finite list of sentences and a sentence*

$$s_1, s_2, \ldots, s_k \vdash s$$

The s_i's are called the **hypothesis** *of the sequent and s the* **conclusion** *of the sequent.*

If $\sigma = s_1, s_2, \ldots, s_k \vdash s$ is a sequent, and $\{x_1, x_2, \ldots, x_n\}$ is the set of variables in σ, we think of it as corresponding to the formula

$$\forall x_1, x_2, \ldots, x_n \in \Gamma^*. s_1 \wedge s_2 \wedge \ldots \wedge s_n \Rightarrow s$$

When the hypothesis is empty, we write s rather that $\vdash s$.

Definition 2.2.3 *An* **assignment** *to a set of variables $V \subseteq X$ is a function $\alpha : V \to (\Gamma \cup X)^*$. If t is a term with variables, we define $t[\alpha]$ to be t with each occurrence of a variable $v \in V$ replaced by $\alpha(v)$.*

This defines a homomorphism from $(\Gamma \cup X)^*$ to $(\Gamma \cup X)^*$.

Definition 2.2.4 *A* **rule** *is of the form*

$$\frac{\sigma}{\sigma_1; \sigma_2; \ldots; \sigma_n}$$

where σ and the σ_i's are sequents.

The rule is written in *refinement style*; hence, it appears upside-down when compared to ordinary sequent calculus rules.

If we are given a structure K we want to construct its *closure* with respect to a collection of rules, by adding terms to the relations as demanded by the rules.

Definition 2.2.5 *If K is a structure and ρ is a collection of rules, we define a ρ, K-***proof** *to be a finite tree whose nodes are sequents, and which satisfies the following requirements: each node and its descendants are related through the image of a rule by some assignment,* **or** *related through an application of the* **implication rule**

$$\frac{H \vdash C}{H \vdash S;\ H, S \vdash C}$$

for any sentence S, **or** *related through an application of the* **universal specialization rule**

$$\frac{H \vdash C}{H' \vdash C'}$$

where H is the image of H' and C is the image of C' under some assignemnt, **or** *the node is a leaf of the form $S_1, S_2, \ldots, S_n \vdash S_i$,* **or** *the node is a leaf of the form $H \vdash R(t_1, t_2, \ldots, t_n)$, where the t_i's are terms (without variables) and $< t_1, t_2, \ldots, t_n > \in K[R]$ (we call such leaves K-***axioms**).

We shall write $H \vdash_{\rho,K} S$ to indicate the existence of a ρ, K-proof with root $H \vdash S$. This is not an unusual notion of proof; they are in fact just proofs in a first-order theory, when we restrict ourselves to the simple sequent formulas.

Definition 2.2.6 *If K is a structure and ρ is a collection of rules, we define $\rho(K)$ to be the structure where*

$$\rho(K)[R] = \{< t_1, t_2, \ldots, t_n > \mid \vdash_{\rho,K} R(t_1, t_2, \ldots, t_n)\}$$

We call $\rho(K)$ the ρ-**closure** *of K.*

We can give a formal justification for calling this a closure.

Theorem 2.2.7 *The ρ-construction is a closure operation on structures in the sense of [13]; that is, $K \subseteq \rho(K)$, $K \subseteq \mathcal{L} \Rightarrow \rho(K) \subseteq \rho(\mathcal{L})$, and $\rho(\rho(K)) \subseteq \rho(K)$.*

Proof: These are all trivial consequences of the definition of ρ, K-proof. □

We can observe a useful fact about the closure, which will prove vital at a later stage; it is basically a restatement of the fact that first-order theories have r.e. theorems.

Theorem 2.2.8 *If K is r.e. then $\rho(K)$ is r.e.*

Proof: Clearly, the collection of ρ, K-proofs is r.e. if we ignore the validity of K-axioms. Hence, an r.e. computation can generate all such pseudo-proofs and for each of them, using concurrency or so-called *dove-tailing*, verify the finitely many K-axioms at its leaves. Hence, the collection of ρ, K-proofs is r.e. when K is r.e. But then to test membership $< t_1, t_2, \ldots, t_n > \in \rho(K)$ we can generate all ρ, K-proofs and look for one whose root is $\vdash R(t_1, t_2, \ldots, t_n)$. □

We shall illustrate this method by constructing the integers. The relation symbols we need are $\square \in Int$ and $\square = \square \in Int$ (the $\square$'s are place-holders). We define the following rules; variables are written in bold-face.

(1) $\dfrac{0 \in Int}{}$ (2) $\dfrac{\mathbf{sn} \in Int}{\mathbf{n} \in Int}$ (3) $\dfrac{\mathbf{pn} \in Int}{\mathbf{n} \in Int}$

(4) $\dfrac{\mathbf{n} = \mathbf{n} \in Int}{\mathbf{n} \in Int}$ (5) $\dfrac{\mathbf{m} = \mathbf{n} \in Int}{\mathbf{n} = \mathbf{m} \in Int}$ (6) $\dfrac{\mathbf{k} = \mathbf{m} \in Int}{\mathbf{k} = \mathbf{n} \in Int;\ \mathbf{n} = \mathbf{m} \in Int}$

(7) $\dfrac{\mathbf{sn} = \mathbf{sm} \in Int}{\mathbf{n} = \mathbf{m} \in Int}$ (8) $\dfrac{\mathbf{pn} = \mathbf{pm} \in Int}{\mathbf{n} = \mathbf{m} \in Int}$

(9) $\dfrac{\mathbf{psn} = \mathbf{n} \in Int}{\mathbf{n} \in Int}$ (10) $\dfrac{\mathbf{spn} = \mathbf{n} \in Int}{\mathbf{n} \in Int}$

Let Θ be the closure of the empty structure with respect to these rules. The way Θ defines the integers is as the set of equivalence classes of $\Theta(\Box \in \mathit{Int})$ under $\Theta(\Box = \Box \in \mathit{Int})$. We can illustrate the concept of proofs by proving that for any integer x we have $ppsx = pspx$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{x} \in \mathit{Int} \vdash \mathit{ppsx} = \mathit{pspx} \in \mathit{Int}}}{\mathbf{x} \in \mathit{Int} \vdash \mathit{psx} = \mathit{spx} \in \mathit{Int}}}{\mathbf{x} \in \mathit{Int} \vdash \mathit{psx} = \mathbf{x} \in \mathit{Int}}}{\mathbf{x} \in \mathit{Int} \vdash \mathbf{x} \in \mathit{Int}} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\mathbf{x} \in \mathit{Int} \vdash \mathbf{x} = \mathit{spx} \in \mathit{Int}}}{\mathbf{x} \in \mathit{Int} \vdash \mathit{spx} = \mathbf{x} \in \mathit{Int}}}{\mathbf{x} \in \mathit{Int} \vdash \mathbf{x} \in \mathit{Int}}}$$

This construction seems similar to that of a term algebra, but as this example illustrates, it is a more powerful specification tool than equational algebras. The cost is, naturally, that the relations specified in this manner often will be undecidable.

Theorem 2.2.9 *For any algebraic specification with positive equations, its term model can be obtained as a relational closure.*

Proof: For every sort S we define relation symbols for equality and membership: $\Box \in S$ and $\Box = \Box \in S$. For every function symbol f with arity $S_1, S_2, \ldots, S_n \rightarrow S$ we add the rules

(formation) $$\dfrac{f(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}) \in S}{\mathbf{x_1} \in S_1; \mathbf{x_2} \in S_2; \ldots, \mathbf{x_n} \in S_n}$$

(congruence) $$\dfrac{f(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}) = f(\mathbf{y_1}, \mathbf{y_2}, \ldots, \mathbf{y_n}) \in S}{\mathbf{x_1} = \mathbf{y_1} \in S_1; \mathbf{x_2} = \mathbf{y_2} \in S_2; \ldots, \mathbf{x_n} = \mathbf{y_n} \in S_n}$$

For each equality relation we give rules for reflexivity, symmetry and transitivity, and for each algebraic equation $\phi_1 = \phi_2$ between terms of sort S with free variables $\mathbf{z_i}$ of sorts S_i for $i \in \{1, 2, \ldots, n\}$ we add the rule

(equation) $$\dfrac{\phi_1 = \phi_2 \in S}{\mathbf{z_1} \in S_1; \mathbf{z_2} \in S_2; \ldots, \mathbf{z_n} \in S_n}$$

It is now easy to see that the term model is obtained by taking the quotient of each sort relation in the closure with respect to the corresponding equality relation. □

2.3 A Categorical Closure

This section presents a closure construction of a category of terms. We want to obtain a category that contains a given set of *postulated* objects and morphisms, and which is closed under the formation of exponents and (almost) limits and colimits of certain diagrams. A crucial decision to make is which diagrams to include; the collection must be countable for naming purposes and it must be easy to reason about. Diagrams that in a certain sense are *computable* seem to be the natural choice. For reasons to be disclosed later we also want a subobject classifier. The technique of relational closures is just what we need to do this.

We give a particular set of relation symbols over a standard alphabet Γ; we use $\Box$ as a place-holder.

$$\begin{array}{ll} \Box\mathit{Obj} & \Box = \Box\mathit{Obj} \\ \Box : \Box \to \Box\mathit{Mor} & \Box = \Box : \Box \to \Box\mathit{Mor} \\ \Box\mathit{Dia} & \Box = \Box\mathit{Dia} \\ \Box(\Box) \in \Box & \Box : \Box(\Box) \to \Box(\Box) \in \Box \end{array}$$

Apart from these we need to define a theory of *computable* functions that allows us to form infinite diagrams. This takes us beyond the theory of toposes, which is the basis of type theory in [13]. Infinity never arises without being explicitly postulated; in [13], a natural numbers object is introduced. We, however, are

interested in *expressiveness* rather than *minimality* and need a wide variety of infinite diagrams and accompaning infinite objects. We will not be specific about which exact formalism we use for this purpose. Any standard notation will do; what we actually write down will be informal descriptions of clearly computable schemes. Our programs compute on strings in Γ^*. We introduce the following relation symbols:

$$\Box Comp \qquad \Box(\Box) \downarrow= \Box Comp$$

and give rules establishing that $f\ Comp$ holds iff f represents a computable function as described above, and $f(t) \downarrow= s$ iff f on input t terminates and equals s. We shall not write these rules in details as they would be tedious and quite standard. An expression of the form $f(t)$ is called a *redex*. We shall allow redices to mix with the other relations in the following manner: for each n-ary relation symbol R we introduce a new n-ary relation on redices $\hat{R}$ with the following rule

$$\text{(computation)} \qquad \frac{\hat{R}(\alpha_1, \alpha_2, \ldots, \alpha_n)}{R(t_1, t_2, \ldots, t_n),\ \alpha_i \downarrow= t_i\ Comp}$$

This simply means that when redices are present in rules, we must proceed by evaluating them, if possible, and then apply the relation. Thus, non-termination of a redex will prevent a relation from holding.

The rules we require follow here; variables are written in bold-face. In this presentation we take a rule

$$\frac{\sigma_1; \sigma_2; \ldots; \sigma_k}{\Psi}$$

to abbreviate the rules

$$\frac{\sigma_1}{\Psi} \quad \frac{\sigma_2}{\Psi} \quad \cdots \quad \frac{\sigma_k}{\Psi}$$

and we will not distinguish between R and $\hat{R}$.

Equality Rules.

(reflexivity) $$\frac{}{\mathbf{x} = \mathbf{x}\ Obj}$$

(symmetry) $$\frac{\mathbf{y} = \mathbf{x}\ Obj}{\mathbf{x} = \mathbf{y}\ Obj}$$

(transitivity) $$\frac{\mathbf{x} = \mathbf{z}\ Obj}{\mathbf{x} = \mathbf{y}\ Obj;\ \mathbf{y} = \mathbf{z}\ Obj}$$

(stability) $$\frac{\mathbf{m} : \mathbf{a_2} \to \mathbf{b_2}\ Mor}{\mathbf{m} : \mathbf{a_1} \to \mathbf{b_1}\ Mor;\ \mathbf{a_1} = \mathbf{a_2}\ Obj;\ \mathbf{b_1} = \mathbf{b_2}\ Obj}$$

We include similar rules for the other equality relations.

Category Rules.

(identity) $$\frac{id(\mathbf{x}) : \mathbf{x} \to \mathbf{x}\ Mor}{\mathbf{x}\ Obj}$$

(composition) $$\frac{(\mathbf{g} \cdot \mathbf{f}) : \mathbf{x} \to \mathbf{z}\ Mor}{\mathbf{f} : \mathbf{x} \to \mathbf{y}\ Mor;\ \mathbf{g} : \mathbf{y} \to \mathbf{z}\ Mor}$$

(cancellation) $$\frac{\mathbf{f} \cdot id(\mathbf{x}) = \mathbf{f} : \mathbf{x} \to \mathbf{y}\ Mor}{\mathbf{f} : \mathbf{x} \to \mathbf{y}\ Mor}$$

(cancellation) $$\frac{id(\mathbf{y}) \cdot \mathbf{f} = \mathbf{f} : \mathbf{x} \to \mathbf{y}\ Mor}{\mathbf{f} : \mathbf{x} \to \mathbf{y}\ Mor}$$

(associativity) $$\frac{(\mathbf{f} \cdot (\mathbf{g} \cdot \mathbf{h})) = ((\mathbf{f} \cdot \mathbf{g}) \cdot \mathbf{h}) : \mathbf{x} \to \mathbf{w}\ Mor}{\mathbf{f} : \mathbf{x} \to \mathbf{y}\ Mor;\ \mathbf{g} : \mathbf{y} \to \mathbf{z}\ Mor;\ \mathbf{h} : \mathbf{z} \to \mathbf{w}\ Mor}$$

(equality) $$\frac{\mathbf{f} \cdot \mathbf{g} = \mathbf{f}' \cdot \mathbf{g}' : \mathbf{a} \to \mathbf{c}\ Mor}{\mathbf{f} = \mathbf{f}' : \mathbf{b} \to \mathbf{c}\ Mor;\ \mathbf{g} = \mathbf{g}' : \mathbf{a} \to \mathbf{b}\ Mor}$$

We shall from here on suppress the parentheses around compositions.

There are some subtle points concerning the representation of diagrams that

must be discussed. We wish to denote a diagram by a list of morphisms; discrete objects would be represented by their identity morphisms. This, however, implies an unfortunate ambiguity. The list

$$\{f : A \to B, g : B \to A\}$$

has four different interpretations as a diagram, depending on which objects we choose to identify. We shall remove this confusion by adding identification tags to the objects, so that objects with the same tag are to be considered identical. In this notation the four interpretations of the above list are denoted as follows:

$$\{f : A(1) \to B(1), g : B(1) \to A(1)\} \quad \{f : A(1) \to B(1), g : B(1) \to A(2)\}$$
$$\{f : A(1) \to B(1), g : B(2) \to A(1)\} \quad \{f : A(1) \to B(1), g : B(2) \to A(2)\}$$

We shall use this in-line notation for diagrams throughout the text. For a subtle technical reason explained in chapter 4 we shall use morphisms as identification tags rather than integers. The use of integers in the text can be thought of as some preselected set of different morphisms (cf. theorem 3.1.4).

Diagram Rules.

(formation) $$\frac{[\mathbf{r}]\ \mathit{Dia}}{\mathbf{r} : \mathit{Comp};\ \mathbf{r(n)} \downarrow = < \mathbf{m, a, b, i, j} >\ \mathit{Comp} \vdash \mathbf{m} : \mathbf{a} \to \mathbf{b}\ \mathit{Mor}}$$

(membership) $$\frac{\mathbf{m} : \mathbf{a(i)} \to \mathbf{b(j)} \in [\mathbf{r}];\ \mathbf{a(i)} \in [\mathbf{r}];\ \mathbf{b(j)} \in [\mathbf{r}]}{[\mathbf{r}]\ \mathit{Dia};\ \mathbf{r(n)} \downarrow = < \mathbf{m, a, b, i, j} >\ \mathit{Comp}}$$

(equality) $$\frac{\mathbf{d_1} = \mathbf{d_2}\ \mathit{Dia}}{\mathbf{d_1}\ \mathit{Dia};\ \mathbf{d_2}\ \mathit{Dia};\ \mathit{Sub}(\mathbf{d_1}, \mathbf{d_2});\ \mathit{Sub}(\mathbf{d_2}, \mathbf{d_1})}$$

where $\mathit{Sub}(\mathbf{x}, \mathbf{y})$ abbreviates

$$\mathbf{m} : \mathbf{a(i)} \to \mathbf{b(j)} \in \mathbf{x} \vdash \mathbf{m} : \mathbf{a(i)} \to \mathbf{b(j)} \in \mathbf{y}$$

The equality on diagrams is quite weak. This just means that we will have several isomorphic but different objects in our category. If we insisted on having a

skeletal category, we could add a rule identifying isomorphic objects.

Limit Rules.

(formation) $$\frac{\mathit{lim}(\mathbf{d})\ \mathit{Obj}}{\mathbf{d}\ \mathit{Dia}}$$

(equality) $$\frac{\mathit{lim}(\mathbf{d_1}) = \mathit{lim}(\mathbf{d_2})\ \mathit{Obj}}{\mathbf{d_1} = \mathbf{d_2}\ \mathit{Dia}}$$

(limit cone) $$\frac{L(\mathbf{d},\mathbf{a},\mathbf{i}) : \mathit{lim}(\mathbf{d}) \to \mathbf{a}\ \mathit{Mor}}{\mathbf{d}\ \mathit{Dia};\ \mathbf{a}(\mathbf{i}) \in \mathbf{d}}$$

(commutativity) $$\frac{\mathbf{m} \cdot L(\mathbf{d},\mathbf{a},\mathbf{i}) = L(\mathbf{d},\mathbf{b},\mathbf{j}) : \mathit{lim}(\mathbf{d}) \to \mathbf{b}\ \mathit{Mor}}{\mathbf{d}\ \mathit{Dia};\ \mathbf{m} : \mathbf{a}(\mathbf{i}) \to \mathbf{b}(\mathbf{j}) \in \mathbf{d}}$$

(universality) $$\frac{\mathit{univ}(\mathbf{x},\mathbf{k},\mathbf{d}) : \mathbf{x} \to \mathit{lim}(\mathbf{d})\ \mathit{Mor}}{\mathit{Cone}(\mathbf{x},\mathbf{k},\mathbf{d})}$$

(commutativity) $$\frac{\mathit{Lcomm}(\mathbf{x},\mathbf{k},\mathbf{d},\mathit{univ}(\mathbf{x},\mathbf{k},\mathbf{d}))}{\mathit{Cone}(\mathbf{x},\mathbf{k},\mathbf{d})}$$

(identification) $$\frac{L(\mathbf{d},\mathbf{a},\mathbf{i}) = L(\mathbf{d},\mathbf{c},\mathbf{j}) : \mathit{lim}(\mathbf{d}) \to \mathbf{a}\ \mathit{Mor}}{\mathbf{d}\ \mathit{Dia};\ \mathbf{a}(\mathbf{i}) \in \mathbf{d};\ \mathbf{c}(\mathbf{j}) \in \mathbf{d};\ \mathbf{a} = \mathbf{c}\ \mathit{Obj};\ \mathbf{i} = \mathbf{j} : \mathbf{p} \to \mathbf{q}\ \mathit{Mor}}$$

(uniqueness) $$\frac{\mathbf{f} = \mathit{univ}(\mathbf{x},\mathbf{k},\mathbf{d}) : \mathbf{x} \to \mathit{lim}(\mathbf{d})\ \mathit{Mor}}{\mathit{Cone}(\mathbf{x},\mathbf{k},\mathbf{d});\ \mathbf{f} : \mathbf{x} \to \mathit{lim}(\mathbf{d})\ \mathit{Mor};\ \mathit{Lcomm}(\mathbf{x},\mathbf{k},\mathbf{d},\mathbf{f})}$$

where $\mathit{Cone}(\mathbf{x},\mathbf{k},\mathbf{d})$ abbreviates

$$\begin{aligned}
&\mathbf{x}\ \mathit{Obj};\ \mathbf{k}\ \mathit{Comp};\ \mathbf{a}(\mathbf{i}) \in \mathbf{d} \vdash \mathbf{k}(\mathbf{a},\mathbf{i}) : \mathbf{x} \to \mathbf{a}\ \mathit{Mor};\\
&\mathbf{m} : \mathbf{a}(\mathbf{i}) \to \mathbf{b}(\mathbf{j}) \in \mathbf{d} \vdash \mathbf{m} \cdot \mathbf{k}(\mathbf{a},\mathbf{i}) = \mathbf{k}(\mathbf{b},\mathbf{j}) : \mathbf{x} \to \mathbf{b}\ \mathit{Mor};\\
&\mathbf{a}(\mathbf{i}) \in \mathbf{d},\ \mathbf{c}(\mathbf{j}) \in \mathbf{d},\ \mathbf{a} = \mathbf{c}\ \mathit{Obj},\ \mathbf{i} = \mathbf{j} : \mathbf{p} \to \mathbf{q}\ \mathit{Mor} \vdash\\
&\qquad \mathbf{k}(\mathbf{a},\mathbf{i}) = \mathbf{k}(\mathbf{a},\mathbf{j}) : \mathbf{x} \to \mathbf{a}\ \mathit{Mor}
\end{aligned}$$

and $\mathit{Lcomm}(\mathbf{x},\mathbf{k},\mathbf{d},\mathbf{f})$ abbreviates

$$\mathbf{a}(\mathbf{i}) \in \mathbf{d} \vdash \mathbf{k}(\mathbf{a},\mathbf{i}) = L(\mathbf{d},\mathbf{a},\mathbf{i}) \cdot \mathbf{f} : \mathbf{x} \to \mathbf{a}\ \mathit{Mor}$$

The definition of a cone is complicated; it states that the cone must provide a morphism to every object in the diagram, that each little triangle involving a

morphism of the diagram must commute, and that on equal objects with equal tags the cone morphisms must be equal, which has the effect of essentially removing duplicates from the diagram. Only some cones can be proven to satisfy this property; those that can we call *p-cones* (provable cones). Similarly, a *p-limit* is a universal p-cone. Clearly, any cone over a finite diagram is a p-cone, since it is a finite collection of morphisms that can be trivially computed. This is an instance of the notion of ϕ-cones we introduced in section 2.1.

Colimit Rules.

(formation) $$\frac{\mathit{colim}(\mathbf{d})\ \mathit{Obj}}{\mathbf{d}\ \mathit{Dia}}$$

(equality) $$\frac{\mathit{colim}(\mathbf{d_1}) = \mathit{colim}(\mathbf{d_2})\ \mathit{Obj}}{\mathbf{d_1} = \mathbf{d_2}\ \mathit{Dia}}$$

(colimit cone) $$\frac{C(\mathbf{d},\mathbf{a},\mathbf{i}) : \mathbf{a} \rightarrow \mathit{colim}(\mathbf{d})\ \mathit{Mor}}{\mathbf{d}\ \mathit{Dia};\ \mathbf{a}(\mathbf{i}) \in \mathbf{d}}$$

(commutativity) $$\frac{C(\mathbf{d},\mathbf{a},\mathbf{i}) = C(\mathbf{d},\mathbf{b},\mathbf{j}) \cdot \mathbf{m} : \mathbf{a} \rightarrow \mathit{colim}(\mathbf{d})\ \mathit{Mor}}{\mathbf{d}\ \mathit{Dia};\ \mathbf{m} : \mathbf{a}(\mathbf{i}) \rightarrow \mathbf{b}(\mathbf{j}) \in \mathbf{d}}$$

(identification) $$\frac{C(\mathbf{d},\mathbf{a},\mathbf{i}) = C(\mathbf{d},\mathbf{a},\mathbf{j}) : \mathbf{a} \rightarrow \mathit{colim}(\mathbf{d})\ \mathit{Mor}}{\mathbf{d}\ \mathit{Dia};\ \mathbf{a}(\mathbf{i}) \in \mathbf{d};\ \mathbf{c}(\mathbf{j}) \in \mathbf{d};\ \mathbf{a} = \mathbf{c}\ \mathit{Obj};\ \mathbf{i} = \mathbf{j} : \mathbf{p} \rightarrow \mathbf{q}\ \mathit{Mor}}$$

(couniversality) $$\frac{\mathit{couniv}(\mathbf{x},\mathbf{k},\mathbf{d}) : \mathit{colim}(\mathbf{d}) \rightarrow \mathbf{x}\ \mathit{Mor}}{\mathit{Cocone}(\mathbf{x},\mathbf{k},\mathbf{d})}$$

(commutativity) $$\frac{\mathit{Ccomm}(\mathbf{x},\mathbf{k},\mathbf{d},\mathit{couniv}(\mathbf{x},\mathbf{k},\mathbf{d}))}{\mathit{Cocone}(\mathbf{x},\mathbf{k},\mathbf{d})}$$

(uniqueness) $$\frac{\mathbf{f} = \mathit{couniv}(\mathbf{x},\mathbf{k},\mathbf{d}) : \mathit{colim}(\mathbf{d}) \rightarrow \mathbf{x}\ \mathit{Mor}}{\mathit{Cocone}(\mathbf{x},\mathbf{k},\mathbf{d});\ \mathbf{f} : \mathit{colim}(\mathbf{d}) \rightarrow \mathbf{x}\ \mathit{Mor};\ \mathit{Ccomm}(\mathbf{x},\mathbf{k},\mathbf{d},\mathbf{f})}$$

where $\mathit{Cocone}(\mathbf{x},\mathbf{k},\mathbf{d})$ abbreviates

$$\begin{aligned}
&\mathbf{x}\ \mathit{Obj};\ \mathbf{k}\ \mathit{Comp};\ \mathbf{a}(\mathbf{i}) \in \mathbf{d} \vdash \mathbf{k}(\mathbf{a},\mathbf{i}) : \mathbf{a} \rightarrow \mathbf{x}\ \mathit{Mor};\\
&\mathbf{m} : \mathbf{a}(\mathbf{i}) \rightarrow \mathbf{b}(\mathbf{j}) \in \mathbf{d} \vdash \mathbf{k}(\mathbf{a},\mathbf{i}) = \mathbf{k}(\mathbf{b},\mathbf{j}) \cdot \mathbf{m} : \mathbf{a} \rightarrow \mathbf{x}\ \mathit{Mor};\\
&\mathbf{a}(\mathbf{i}) \in \mathbf{d},\ \mathbf{c}(\mathbf{j}) \in \mathbf{d},\ \mathbf{a} = \mathbf{c}\ \mathit{Obj},\ \mathbf{i} = \mathbf{j} : \mathbf{p} \rightarrow \mathbf{q}\ \mathit{Mor} \vdash\\
&\qquad \mathbf{k}(\mathbf{a},\mathbf{i}) = \mathbf{k}(\mathbf{a},\mathbf{j}) : \mathbf{a} \rightarrow \mathbf{x}\ \mathit{Mor}
\end{aligned}$$

and $Ccomm(\mathbf{x},\mathbf{k},\mathbf{d},\mathbf{f})$ abbreviates

$$\mathbf{a(i)} \in \mathbf{d} \vdash \mathbf{k(a,i)} = \mathbf{f} \cdot C(\mathbf{d},\mathbf{a},\mathbf{i}) : \mathbf{a} \rightarrow \mathbf{x}\ Mor$$

The cocones defined in this manner we call *p-cocones.* As before, we define a *p-colimit* to be a couniversal *p*-cocone.

Exponent Rules.

(formation) $$\frac{exp(\mathbf{a},\mathbf{b})\ Obj}{\mathbf{a}\ Obj;\ \mathbf{b}\ Obj}$$

(equality) $$\frac{exp(\mathbf{a_1},\mathbf{b_1}) = exp(\mathbf{a_2},\mathbf{b_2})\ Obj}{\mathbf{a_1} = \mathbf{a_2}\ Obj;\ \mathbf{b_1} = \mathbf{b_2}\ Obj}$$

(evaluation) $$\frac{eval(\mathbf{a},\mathbf{b}) : \mathbf{a} \times exp(\mathbf{a},\mathbf{b}) \rightarrow \mathbf{b}\ Mor}{\mathbf{a}\ Obj;\ \mathbf{b}\ Obj}$$

(currying) $$\frac{curry(\mathbf{f}) : \mathbf{c} \rightarrow exp(\mathbf{a},\mathbf{b})\ Mor}{\mathbf{f} : \mathbf{a} \times \mathbf{c} \rightarrow \mathbf{b}\ Mor}$$

(commutativity) $$\frac{Ecomm(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{f},curry(\mathbf{f}))}{\mathbf{f} : \mathbf{a} \times \mathbf{c} \rightarrow \mathbf{b}\ Mor}$$

(uniqueness) $$\frac{\mathbf{g} = curry(\mathbf{f}) : \mathbf{c} \rightarrow exp(\mathbf{a},\mathbf{b})\ Mor}{\mathbf{f} : \mathbf{a} \times \mathbf{c} \rightarrow \mathbf{b}\ Mor;\ \mathbf{g} : \mathbf{c} \rightarrow exp(\mathbf{a},\mathbf{b});\ Ecomm(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{f},\mathbf{g})}$$

where $Ecomm(\mathbf{a},\mathbf{b},\mathbf{c},\mathbf{f},\mathbf{g})$ abbreviates

$$eval(\mathbf{a},\mathbf{b}) \cdot (id(\mathbf{a}) \times \mathbf{g}) = \mathbf{f} : \mathbf{a} \times \mathbf{c} \rightarrow \mathbf{b}\ Mor$$

and $\times$ abbreviates well-known concepts from the product.

Morphism Abstraction Rules.

We also want a way of introducing morphisms directly through well-defined computations.

(abstraction) $$\frac{morph(\mathbf{f}) : \mathbf{a} \rightarrow \mathbf{b}\ Mor}{\mathbf{f}\ Comp;\ \mathbf{a}\ Obj;\ \mathbf{b}\ Obj;\ func(\mathbf{f},\mathbf{a},\mathbf{b})}$$

where $func(f, a, b)$ abbreviates

$$\mathbf{x} = \mathbf{y} : \mathbf{1} \to \mathbf{a}\ Mor \vdash \mathbf{f}(\mathbf{x}) = \mathbf{f}(\mathbf{y}) : \mathbf{1} \to \mathbf{b}\ Mor$$

i.e. f transforms elements of a into elements of b and respects equality.

(application) $$\frac{morph(\mathbf{f}) \cdot \mathbf{x} = \mathbf{f}(\mathbf{x}) : \mathbf{1} \to \mathbf{b}\ Mor}{morph(\mathbf{f}) : \mathbf{a} \to \mathbf{b}\ Mor;\ \mathbf{x} : \mathbf{1} \to \mathbf{a}\ Mor}$$

The functions we provide morphism abstractions for we call *p-functions* (provable functions).

Image Rules

To facilitate the construction of a subobject classifier we shall give notation for images, i.e. the monic part of epi-mono factorizations. These rules are really redundant; images are consequences of finite limits and colimits.

(image) $$\frac{\mathbf{f}(\mathbf{a}) = lim(\Delta_2)\ Obj;\ im(\mathbf{f}) = L(\Delta_2, \mathbf{b}, 1) : \mathbf{f}(\mathbf{a}) \to \mathbf{b}}{\mathbf{f} : \mathbf{a} \to \mathbf{b}\ Mor}$$

where Δ_2 represents the diagram

$$\{C(\Delta_1, \mathbf{b}, 1) : \mathbf{b}(1) \to colim(\Delta_1)(1),\ C(\Delta_1, \mathbf{b}, 2) : \mathbf{b}(1) \to colim(\Delta_1)(1)\}$$

and Δ_1 represents the diagram

$$\{\mathbf{f} : \mathbf{a}(1) \to \mathbf{b}(1),\ \mathbf{f} : \mathbf{a}(1) \to \mathbf{b}(2)\}$$

This is very straight-forward: Δ_1 is a pushout diagram and Δ_2 is an equalizer diagram. The construction of images is illustrated in figure 2.4.

Subobject Classifier Rules.

(classifier) $$\frac{}{\Omega\ Obj}$$

(T) $$\frac{}{\mathsf{T} : 1 \to \Omega\ Mor}$$

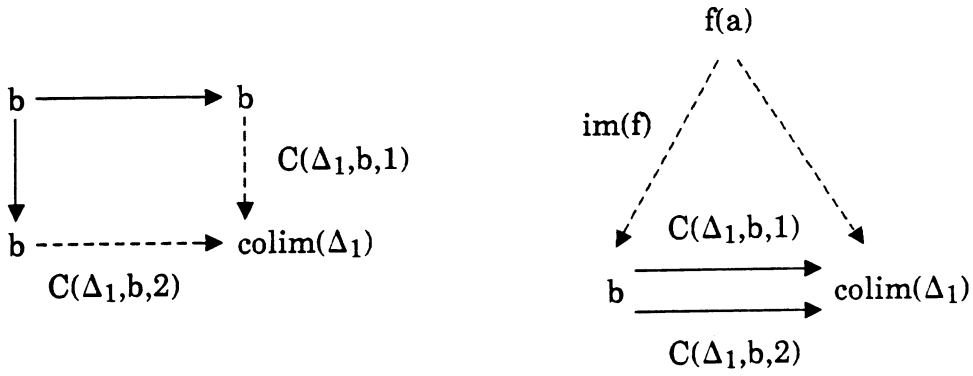

Figure 2.4: The Construction of Images

(character) $$\frac{\boldsymbol{char(im(f))} : \mathbf{b} \rightarrow \Omega\, \boldsymbol{Mor}}{\mathbf{f} : \mathbf{a} \rightarrow \mathbf{b}\, \boldsymbol{Mor}}$$

(pullback) $$\frac{\boldsymbol{Pullback(im(f), f(a), b, char(im(f)))}}{\mathbf{f} : \mathbf{a} \rightarrow \mathbf{b}\, \boldsymbol{Mor}}$$

(uniqueness) $$\frac{\mathbf{g} = \boldsymbol{char(im(f))} : \mathbf{b} \rightarrow \Omega\, \boldsymbol{Mor}}{\mathbf{f} : \mathbf{a} \rightarrow \mathbf{b}\, \boldsymbol{Mor};\ \mathbf{g} : \mathbf{b} \rightarrow \Omega\, \boldsymbol{Mor};\ \boldsymbol{Pullback(im(f), f(a), b, g)}}$$

where $\boldsymbol{Pullback}(\mathbf{h}, \mathbf{c}, \mathbf{d}, \mathbf{g})$ abbreviates

$$\boldsymbol{lim}(\Delta) = \mathbf{c}\ \boldsymbol{Obj};\ \boldsymbol{L}(\Delta, \mathbf{d}, 1) = \mathbf{h} : \mathbf{c} \rightarrow \mathbf{d}\ \boldsymbol{Mor}$$

and Δ represents the diagram $\{\top : 1(1) \rightarrow \Omega(1), g : \mathbf{d}(1) \rightarrow \Omega(1)\}$. This situation is illustrated in figure 2.4. Notice, that the subobject classifier only provides characters for images of morphisms. This is because a naïve definition of monics would only yield characters for *provable* monics, which would leave us with a deficient subobject classifier. In this manner we avoid such problems, since we can easily construct the image of every morphism, and $\chi_f = \chi_{im(f)}$ when f is a monic.

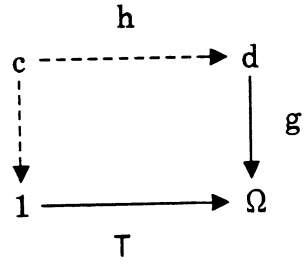

Figure 2.5: Pullback Situation

Definition 2.3.1 *If K is a structure in this context, we call its relational closure with respect to these rules the* **categorical closure** *of K, $CC(K)$.*

We shall construct the *term category generated by K*. Define

$$O(K) = CC(K)(\Box\ Obj)/CC(K)(\Box = \Box\ Obj)$$
$$M(K) = CC(K)(\Box : \Box \to \Box\ Mor)/CC(K)(\Box = \Box : \Box \to \Box\ Mor)$$
$$D(K) = CC(K)(\Box\ Dia)/CC(K)(\Box = \Box\ Dia)$$

Definition 2.3.2 *$TC(K)$ is the category with objects $O(K)$ and morphisms $M(K)$, where the identity of $[A] \in O(K)$ is $[id(A)] \in M(K)$ and we define composition by $[f] \cdot [g] = [f \cdot g]$.*

We have constructed a quite interesting category.

Theorem 2.3.3 *$TC(K)$ is finitely bicomplete and cartesian closed, has a subobject classifier and a natural numbers object; hence, it is in particular a topos in the sense of [13].*

Proof: Every diagram $[d] \in D(K)$ has a p-limit $[lim(d)]$ and a p-colimit $[colim(d)]$. Similarly, any two objects $[a], [b] \in O(K)$ has the exponent $[exp(a, b)]$. The category is finitely bicomplete since every finite diagram is computable, every finite cone is a p-cone, and every finite cocone is a p-cocone. The subobject classifier is $[\Omega]$; this works because monics and images are the same concept in a category with pushouts and equalizers, so for any monic $[f]$ we have

$\chi_{[f]} = [char(im(f))]$. We can also construct a natural numbers object. Define $N_0 = 1$ and $N_{i+1} = N_i + 1$, and let N be the colimit of the diagram $\Delta = \{in_l : N_i \to N_{i+1}\}$. Define $\overline{k}$ to be the colimit morphism $N_k \to N$. Our zero morphism $0 : 1 \to N$ is $\overline{0}$. Construct the Δ-cocone $\{\overline{k+1} \cdot in_l : N_k \to N\}$. The induced morphism is our successor morphism $S : N \to N$. Suppose $1 \xrightarrow{f} A \xrightarrow{g} A$ is given. Let $\phi_0 = f$ and $\phi_{i+1} = \phi_i + g \cdot \phi_i$, and let $\phi : N \to A$ be induced by the Δ-cocone $\{\phi_i : N_i \to A\}$. Then clearly $\phi \cdot 0 = f$ and $g \cdot \phi = \phi \cdot S$. If also $\psi \cdot 0 = f$ and $g \cdot \psi = \psi \cdot S$ then $\psi \cdot \overline{k} = \phi_k$, so uniqueness of the induced morphism ensures that $\phi = \psi$. Hence, N is a natural numbers object. □

This result does not reveal a very important property of the term category, namely that it has *computable* limits and colimits of *computable* diagrams.

We want to argue inductively about the objects and morphisms in the category. For example, to verify some property of the limit of a diagram, we want to first look at the components of the diagram, and then argue that the limit operation preserves that property. To know that this process will terminate we need to establish some notion of well-foundedness.

Definition 2.3.4 *If x and y are morphisms or objects in the category, we write $x \sqsubset y$ if x is a component of the construction of y. Examples are: if a is an object or a morphism in the diagram d then $a \sqsubset lim(d)$; both a and b are smaller that $exp(a, b)$; if m is a morphism in the cone (x, k) over the diagram d, then $m \sqsubset univ(x, k, d)$.*

The nature and well-definedness of this ordering should be intuitively obvious.

Theorem 2.3.5 *The ordering $\sqsubset$ on $\mathcal{TC}(K)$ is well-founded.*

Proof: This amounts to observing that $\mathcal{TC}(K)$ is really defined through an inductive process, where we start with K and at successor stages add the terms

formed by using the proof rules on previously defined terms. This is an inductive definition that will close below ω_1^{CK}. Now, the ordering $t_1 \sqsubset t_2$ merely states that t_1 is introduced at an earlier stage than t_2. Hence, $\sqsubset$ is well-founded. □

2.4 The Categorical Type Theory

We propose $\mathcal{TC}(\emptyset)$, the category generated by the empty structure, as a type theory. The objects are *types* and the morphisms are *terms*. We shall later see that we may think of the diagrams as *type constructors*. From the result in theorem 2.3.3 we may expect this to be a rich theory; chapter 4 is dedicated to demonstrating its usefulness.

We call $\mathcal{TC}(\emptyset)$ our *core theory*; it contains all the types and terms of ultimate interest. At a glance, it may seem that $\mathcal{TC}(\emptyset)$ is empty but this is not so. Clearly, the empty diagram is computable, so we immediately get the initial and final objects, 0 and 1. From there on, it is obvious that we can build more complicated objects and morphisms.

To accomodate a higher-order predicative theory as described in [4], we shall construct a stratified version that will give us a hierarchy of categories. Define $H_1 = \mathcal{TC}(\emptyset)$ and $K_1 = \emptyset$. We construct successor levels as follows: Define $K_{i+1} = CC(K_i) \cup UNI_i$, where UNI_i is the structure containing a special object U_i and a morphism $\mu_i(T) : 1 \rightarrow U_i$ for every object T in H_i. Now we can define $H_{i+1} = \mathcal{TC}(K_{i+1})$. The H_i's form an inclusive hierarchy; if $i < j$ there is an obvious inclusion functor $I_{ij} : H_i \rightarrow H_j$. This is a very simple and straight-forward way of obtaining a predicative hierarchy; each object in H_i becomes an element

of $\mathcal{U}_i$. Iterating this process gives us the hierarchy with which we shall match the universe hierarchy of the νPrl theory.

Definition 2.4.1 *We call the sequence* $\{H_i\}_{i\geq 1}$ **the hierarchy of categorical types.**

In chapter 4 we shall show how these categories embodies sufficient structure to represent the types and terms of the νPrl theory.

Theorem 2.4.2 *Each* H_i *is r.e.*

Proof: Clearly, $\emptyset$ is r.e., and theorem 2.2.8 tells us that every iteration preserves the r.e. property, since each UNI_i is also r.e. □

This is an important property, which we shall use to argue that certain diagrams providing the link between H_i and H_{i+1} are computable.

We could continue this construction through higher ordinals. For example, define $K_\omega = \bigcup_i K_i \cup UNI_\omega$ and $H_\omega = \mathcal{TC}(K_\omega)$ where UNI_ω is the structure with a special object $\mathcal{U}_\omega$ and a morphism $\mu_\omega(T) : 1 \rightarrow \mathcal{U}_\omega$ for every object T in $\bigcup_i H_i$. These extensions are mostly curiosities at this point; one might idly wonder if the inductive limit of this process would yield an impredicative theory.

Chapter 3

Consistency

A foolish consistency is the hobgoblin of small minds .

— Ralph W. Emerson .

The *consistency* of our theory is the subject of this chapter. We suggest a formal definition and give a model theoretic proof that it holds for our theory. Finally, we point out that extensions of the free category construction may render the result inconsistent.

3.1 The Definition

In the previous chapter we defined an inclusive hierarchy of categories $\{H_i\}$ and proposed it as a type theory. But nothing we have shown so far precludes that some or all of the H_i's are degenerate. We might proceed to build higher-level concepts in this theory, only to find out later that we have been talking about a degenerate category, which, on the surface is a very rich theory, indeed.

Definition 3.1.1 *A category is* **consistent** *if the objects* 0 *and* 1 *are not isomorphic.*

This a reasonable concept; the 0 and 1 objects may be viewed as a variation of *true* and *false* when we employ the propositions-as-types principle, so this definition merely emulates the usual notion of consistency. We are not surprised to obtain the following result.

Theorem 3.1.2 *A topos is consistent iff it is non-degenerate.*

Proof: Let H be any topos. If it is degenerate then clearly it is inconsistent. Now, suppose $\phi : 1 \cong 0$ is given. Let A be any object. The unique morphism $! : A \rightarrow 1$ induces a morphism $\phi \cdot ! : A \rightarrow 0$. For any object B we have that $H(0, B^A) \cong H(0 \times A, B)$, so $0 \times A$ is also initial; hence, $0 \cong 0 \times A$. The morphisms $\phi \cdot ! : A \rightarrow 0$ and $Id_A : A \rightarrow A$ induces a unique morphism $< \phi \cdot !, Id_A >: A \rightarrow 0 \times A$. Now $< \phi \cdot !, Id_A > \cdot \pi_A : 0 \times A \rightarrow 0 \times A$ is unique due to initiality, so $< \phi \cdot !, Id_A > \cdot \pi_A = Id_{0 \times A}$ and $\pi_A : 0 \times A \cong A$. Hence, $0 \cong A$. Since this holds for any A, we conclude that all objects are isomorphic. □

Corollary 3.1.3 *A topos is inconsistent iff it has a morphism* $1 \rightarrow 0$.

Inconsistency turns out to be equivalent to the identification of the topos theoretic truth values.

Theorem 3.1.4 *In a topos,* $\top = \bot$ *iff it is inconsistent.*

Proof: Suppose the category is inconsistent and, hence, degenerate. Then $\Omega \cong 1$, so there is only one morphism $1 \rightarrow \Omega$, but then $\top = \bot$. Suppose now that $\top = \bot$. By definition 0 is the limit of the diagram

$$\Delta = \{\top : 1(1) \rightarrow \Omega(1), \bot : 1(2) \rightarrow \Omega(1)\}$$

Since, $\top = \bot$ we see that

$$\{Id : 1(3) \rightarrow 1(1), Id : 1(3) \rightarrow 1(2), \top : 1(3) \rightarrow \Omega(1)\}$$

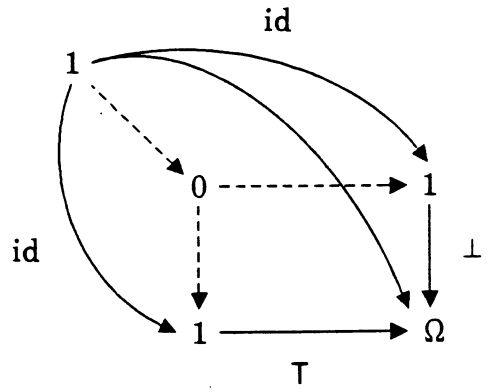

Figure 3.1: $\top = \bot$ induces morphism from 1 to 0

is a Δ-cone, so universality of the limit induces a morphism $1 \rightarrow 0$, and the category is inconsistent. The situation is shown in figure 3.1. □

It may appear that consistency is too weak to ensure that the objects correspond to our intuition of interesting types. The following result indicates the depth of our definition.

Theorem 3.1.5 *If $\mathcal{TC}(K)$ is consistent then its natural numbers object N has $\aleph_0$ distinct elements, viz. $\{S^i 0\}_{i \geq 0}$.*

Proof: Suppose that $m < n$ but $S^m 0 = S^n 0$. Given any $g : 1 \rightarrow A$ and $f : A \rightarrow A$ there is induced a unique $\phi : N \rightarrow A$ so that $\phi \cdot S^i 0 = f^i g$. Now let $A = \Omega \times \Omega \times \ldots \times \Omega$ be the n-fold product of Ω's. Define $g : 1 \rightarrow A$ to be the morphism induced by the diagram

$$\{\top : 1 \rightarrow \Omega(1), \bot : 1 \rightarrow \Omega(i) \ (i = 2, \ldots n)\}$$

i.e., intuitively the tuple $< \top, \bot, \bot, \ldots, \bot >$. Define $f : A \rightarrow A$ to be the morphism induced by the diagram

$$\{\pi_{i+1 \bmod n} : A \rightarrow \Omega(i) \ (i = 1, 2, \ldots, n)\}$$

i.e., intuitively a cyclic right-shift. Now from the assumption we infer that $f^m g = f^n g$ and, hence, that $\pi_1 f^m g = \pi_1 f^n g$, from which we conclude that $\top = \bot$, so $\mathcal{TC}(K)$ is inconsistent, which is a contradiction. □

In a similar way, we are able to convince ourselves that we can satisfy any requirement of non-identification that our intuition demands.

It remains to prove consistency of the H_i's. One very hard way to do this would be to proof-theoretically show that 0 and 1 are non-isomorphic in each H_i. This would involve quantification over every possible pair of morphisms between 0 and 1, which hardly seems practical. We are led to a model-theoretic argument.

3.2 Consistency of the Hierarchy

We can demonstrate consistency through the following argument.

Theorem 3.2.1 *A topos T is consistent iff there is a consistent topos M and a functor $F : T \to M$ that preserves limits and colimits.*

Proof: If T is consistent we can use the identity functor $T \to T$. Conversely, let $F : T \to M$ be as described and suppose that T is inconsistent, i.e. there exists a morphism $m : 1_T \to 0_T$. This gives us a morphism $F(m) : F(1_T) \to F(0_T)$. But since F preserves limits and colimits we have $F(1_T) = 1_M$ and $F(0_T) = 0_M$. Hence, $F(M) : 1_M \to 0_M$ is a morphism and M is inconsistent, which is a contradiction. □

The target category we shall use is *SET*.

Observation 3.2.2 *SET is a consistent topos.*

Proof: In section 2.1 it is argued that *SET* is a topos. Since in *SET* we have $0 = \emptyset$, $1 = \{\bullet\}$ and isomorphism is bijection it is clearly consistent. □

We now show how to construct the required functors.

Theorem 3.2.3 *For each H_i there is a functor $F_i : H_i \rightarrow SET$ preserving limits, colimits, exponents, and subobject classifier.*

Proof: First build F_1 inductively as follows. Map Ω to the subobject classifier of *SET*, i.e. a set with two elements $\{tt, ff\}$; let $F_1(\top) : 1 \rightarrow \{tt, ff\}$ select the element tt. Map identity morphisms to identity functions and compositions to compositions. Given any diagram d we can get a corresponding diagram in *SET* by interpreting each component of the diagram. We define F_1 to map $lim(d)$ to the *SET*-limit and $colim(d)$ to the *SET*-colimit of the corresponding *SET*-diagram. The construction of *SET*-bilimits is detailed in [1]. Given a p-cone (x, k) over d it can be interpreted in set as a cone over $F_1(d)$, so we map $univ(x, k, d)$ to the induced morphism in *SET*; similarly for p-cocones. The exponent $exp(a, b)$ is mapped to the set of functions $F_1(a) \rightarrow F_1(b)$; we define $eval(a, b)$ as $F_1(eval(a,b))(f, x) = f(x)$, and $curry(f)$ as $F_1(curry(f))(c)(x) = f(c, x)$. If f is a computation transforming elements of a into elements of b we define $F_1(morph(f))(x) = F_1(y)$, where $f(x) = y$. Let $f : a \rightarrow b$; in *SET* $F_1(im(f))$ is the inclusion function from $F_1(f)(F_1(a))$ to $F_1(b)$; $F_1(char(im(f)))$ is the characteristic function $\chi : F_1(b) \rightarrow \{tt, ff\}$ where $\chi(x) = tt$ iff $x \in F_1(f)(F_1(a))$. It is immediate that F_1 is a well-defined functor preserving the required constructions. Now assume we have a functor $F_i : H_i \rightarrow SET$. We can extend F_i to the functor F_{i+1} in the following way: first extend it to cover the universe object by defining $F_{i+1}(U_i) = \{i\} \times Obj(H_i)$ and $F_{i+1}(\mu_i(T)) = < i, T >$; next extend it to the new constructions in $H_{i+1} \setminus H_i$ in a manner analogous to the definition of F_1. □

This tells us that we can interpret the hierarchy in *SET* in an interesting and nontrivial way.

Corollary 3.2.4 *All the H_i's are consistent.*

Proof: This follows immediately by combining the results in this section. □

3.3 An Inconsistent Extension

The $\mathcal{TC}(\Box)$-construction is a wonderful mechanism that creates an interesting category seemingly out of thin air. One could easily be lead to assume that the closure could be extended to include most other useful constructs. This section will demonstrate that the requirement of consistency imposes severe restrictions on what can successfully be put on one's wish list.

Definition 3.3.1 *A* **retract-universal** *object in a category is an object Υ, such that for any object A there is a pair of morphisms*

$$r_A : A \to \Upsilon,\ s_A : \Upsilon \to A$$

such that $s_A \cdot r_A = Id_A$.

This is a fairly common construct; most categories of domains have universal objects, which are in particular retract-universal [19]. We could easily include such a feature in our closure, by adding the following rules.

Universal Object Rules.

(Universal) $\dfrac{}{\Upsilon\ Obj}$

(Retract Pair) $$\frac{R(\mathbf{a}) : \mathbf{a} \rightarrow \Upsilon\, Mor;\ S(\mathbf{a}) : \Upsilon \rightarrow \mathbf{a}\, Mor}{\mathbf{a}\, Obj}$$

(Retraction) $$\frac{S(\mathbf{a}) \cdot R(\mathbf{a}) = id(\mathbf{a}) : \mathbf{a} \rightarrow \mathbf{a}\, Mor}{\mathbf{a}\, Obj}$$

Unfortunately, we have now constructed an inconsistent category.

Theorem 3.3.2 *Including retract-universal objects guarantees that $\mathcal{TC}(K)$ is inconsistent.*

Proof: Just observe that $S(0) \cdot R(1)$ is a morphism $1 \rightarrow 0$ in $\mathcal{TC}(K)$, which by corollary 3.1.3 implies inconsistency. □

One could view this result a version of *Girard's Paradox* [8]: the impossibility of having a type of all types. The analogy is not strong, however, since the concept of retractions is a rather heavy-handed way of expressing the type-of-all-types notion. Universal objects could be maintained consistently at the cost of either completeness or cocompleteness, which shows that we have many choices when designing a theory in this framework.

Chapter 4

Refining the Theory

In the end everybody must understand for himself .

— Per Martin-Löf .

The theory we have defined seems to be far removed from more conventional type theories. In this chapter we shall recover many well-known concepts by demonstrating how they can be emulated in our theory. In particular, we shall attempt to represent the νPrl theory [4] as faithfully as possible. This project will not succeed entirely, in part because the νPrl theory as it stands is too voluminous to be manageable, but also because variations of the νPrl concepts often seem to emerge more naturally from the categorical concepts than the originals do. We feel, however, that we can capture the *essence* of the νPrl theory.

4.1 Sequents, Types, and Terms

Refining our theory is a process of high-lighting certain objects and morphisms that we consider to be of particular interest and showing how their special properties can yield practical rules describing their behavior. The interesting objects are called *types* and the interesting morphisms are called *terms*. It is usually

the case that if A and B are types then all the morphisms between A and B are terms, but we will not insist on this property. It is understood that if an object is not a type today, it may become a type tomorrow; all that is required is that we find a practical use for it overnight. Thus, the notion of types and terms is *open-ended*, but *bounded*: the collection of types may grow, but a type will always be an object. Of course, we are subject to the possibility of extending the term category construction and, thus, admitting many more potential types. The claim that we will not want to look for types outside our current hierarchy is a statement of faith, which we may well be forced to retract in the future.

We need to develop some notation before we can define the kind of rules we shall use. If $B : A \rightarrow \mathcal{U}_i$ is a morphism in H_{i+1}, then $X(A,B)$ is the diagram in H_i described by the following computation: On input $< a, T >$ we verify that a is a morphism $1 \rightarrow A$; we then verify that T is an object; next we verify that $B \cdot a = \mu_i(T)$; if we terminate at this point we output $< id(T) : T, T, a, a >$. This has the effect of generating the discrete diagram of all the elements of $\mathcal{U}_i$ that are selected by B and elements of A. Notice, that tagging with morphisms allows us to specify that equal objects selected by different representatives of an element of A are identified, whereas objects selected by different elements of A are kept distinct.

Definition 4.1.1 *We call $X(A,B)$ the* **indexed collection** *generated by A and B.*

Notice, that any such indexed collection already existed in H_i; the morphism B in H_{i+1} was merely used to point it out.

Definition 4.1.2 *If $B : A \rightarrow \mathcal{U}_i$ is a morphism, then $\sigma(A,B)$ is the colimit of $X(A,B)$.*

We can now define a *sequent*, our unit of inference; notice that this is entirely different from the concept of sequent we used in chapter 2.

Definition 4.1.3 *A* **sequent** *is of the form:*

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

$$b(x_1, x_2, \ldots, x_n) \in B(x_1, x_2, \ldots, x_n)$$

Define $S_0 = 1$ *and* $S_{k+1} = \sigma(S_k, A_k)$*; we say that the sequent* **holds** *if we can prove the following conditions in the closure* $CC(K_{i+1})$*:*

- $\vdash A_k : S_{k-1} \rightarrow \mathcal{U}_i\ Mor$, *for* $1 \leq k \leq n$
- $\vdash B : S_n \rightarrow \mathcal{U}_i\ Mor$
- $\vdash b\ Comp$; $\vdash B\ Comp$
- $a_0 : 1 \rightarrow 1\ Mor, P_1, P_2, \ldots, P_n, \vdash$

  $$b(a_1, a_2, \ldots, a_n) : 1 \rightarrow B(a_1, a_2, \ldots, a_n)\ Mor$$

  where P_k *is* $a_k : 1 \rightarrow T_k\ Mor, A_k \cdot a_{k-1} = \mu_i(T_k) : 1 \rightarrow \mathcal{U}_i\ Mor$
- $a_0 = c_0 : 1 \rightarrow 1\ Mor, Q_1, Q_2, \ldots, Q_n, \vdash$

  $$B(a_1, a_2, \ldots, a_n) = B(c_1, c_2, \ldots, c_n)\ Obj$$
- $a_0 = c_0 : 1 \rightarrow 1\ Mor, Q_1, Q_2, \ldots, Q_n, \vdash$

  $$b(a_1, a_2, \ldots, a_n) = b(c_1, c_2, \ldots, c_n) : 1 \rightarrow B(a_1, a_2, \ldots, a_n)\ Mor$$

  where P_k *is* $a_k = c_k : 1 \rightarrow T_k\ Mor, A_k \cdot a_{k-1} = \mu_i(T_k)$

This is merely a tedious way of spelling out the requirements for sequents with dependent hypotheses as they are encountered in predicative type theories [4,15]: Each A_k must define an object when it is provided elements from $A_1, A_2, \ldots, A_{k-1}$;

B is a computation that given elements from each A_k produces an object, b is a computation that given elements from each A_k produces an element of the corresponding B, and both b and B are required to respect the equalities of the A_k's. All these requirements allow us to think of b as defining a function in the category H_{i+1}, which may be viewed as a particularly strong existential statement. The sequents we shall use will normally be much simpler than the most general version. A simple example is a projection:

$$x : A, y : B(x) \vdash_i \phi(x, y) \in C(x, y)$$

where B and C are always yielding the object D. This sequent holds, since given input x and y we can define ϕ to output the y. Clearly, in this situation we can satisfy all the conditions above; they all become very simple. We shall allow ourselves to use a more liberal notation and exploit lack of dependencies; we will normally write the sequent as

$$x : A, y : D \vdash y \in D$$

This is far more readable and can consistently be read as short-hand for the formal notation.

We define rules in a very standard way.

Definition 4.1.4 *A* **rule** *is of the form*

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

where S and the S_i's are sequents. The rule asserts that if the S_i's hold then S will hold.

We are now ready to define some types and rules. Beware that these rules do not *define* the types, which are just suggestive names for existing objects. Each

rule must be proven *sound*, i.e. we must show that it asserts a true fact about the term categories.

When proving a rule sound we are faced with what seems to be a very large proof burden. In practice, however, it is a quite manageable task. We are partly saved by the equally complex assumptions that the rules provide us, but we can also make use of the following simple observation: If $\alpha, \beta, \ldots$ are computations that on input h are known to produce elements of $A, B, \ldots$ we can construct any well-formed morphism expression using $\alpha, \beta, \ldots$ in place of elements and know that the resulting computation on input h produces instances of the morphism expression and can be proven to respect equality. For example, if we know that given an element of H, α produces an element of $A \to B$ and β produces an element of $B \to C$, we can define the computation $\beta \cdot \alpha$ that on input h produces $\beta(h) \cdot \alpha(h)$ which is an element of $A \to C$. If now α and β are known to preserve equality of H it follows easily that so does $\beta(h) \cdot \alpha(h)$, since composition is defined to preserve equality of its arguments. Hence, we have proven soundness of the rule:

$$\frac{h : H \vdash (\beta(h) \cdot \alpha(h))(h) \in A \to C}{h : H \vdash \alpha(h) \in A \to B;\ h : H \vdash \beta(h) \in B \to C}$$

which, if we assume the convention that computations without explicit arguments depend on the entire hypothesis, we can write as

$$\frac{H \vdash \beta \cdot \alpha \in A \to C}{H \vdash \alpha \in A \to B;\ H \vdash \beta \in B \to C}$$

All simple constructions have been defined to preserve equality, and in the case of morphisms induced by universality and couniversality we get functionality from the inherent uniqueness.

4.2 Propositional Types

The propositional types are defined as follows:

- *Void* is 0, the limit of the empty diagram.
- $A \times B$ is the limit of the binary discrete diagram $\{A, B\}$.
- $A + B$ is the colimit of the binary discrete diagram $\{A, B\}$.
- $A \rightarrow B$ is the exponent of A and B.

We derive the following rules.

(*Void*-elim) $$\frac{H, x : Void \vdash any(x) \in T}{}$$

Here we define $any(x) = univ(T, \emptyset, \emptyset) \cdot x$, i.e. the composition of the uniquely induced morphism from T to 0 and x. Since composition preserves equality, we can prove that *any* is functional.

(×-intro) $$\frac{H \vdash < a, b > \in A \times B}{H \vdash a \in A;\ H \vdash b \in B}$$

Here $< a, b >= univ(1, \{a, b\}, \{A, B\})$ is defined to be the morphism induced by the cone $\{a, b\}$ over the diagram $\{A, B\}$. This is a finite cone that can trivially be expressed by a computation. Uniqueness of this construction ensures functionality.

(×-elim) $$\frac{H \vdash prl(x) \in A}{H \vdash x \in A \times B} \qquad \frac{H \vdash prr(x) \in B}{H \vdash x \in A \times B}$$

We define $prl(x) = L(\{A, B\}, A) \cdot x$, i.e. the composition of x and the left projection of the product. Again, functionality follows easily. Similarly, $prr(x)$ is x composed with the right projection of the product. Product are illustrated in figure 4.1.

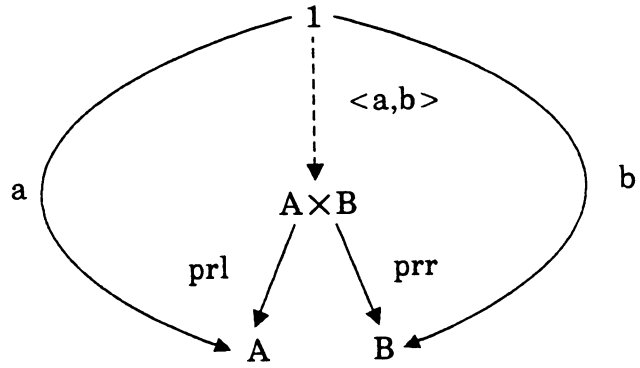

Figure 4.1: Product Types

(+-intro) $$\frac{H \vdash inl(a) \in A + B}{H \vdash a \in A} \qquad \frac{H \vdash inr(b) \in A + B}{H \vdash b \in B}$$

Here $inl(a)$ is $C(A, B, A) \cdot a$, i.e. the composition of a with the left injection into the coproduct, and $inr(b)$ is defined similarly.

(+-elim) $$\frac{H \vdash decide(f, g) \in A + B \rightarrow C}{H \vdash f \in A \rightarrow C;\ H \vdash g \in B \rightarrow C}$$

Here $decide(f, g) = couniv(1, \{f, g\}, \{A, B\})$, i.e. the unique morphism induced by the finite cocone $\{f, g\}$ over the diagram $\{A, B\}$. Coproducts are illustrated in figure 4.2.

(→-intro) $$\frac{H \vdash \lambda(f) \in A \rightarrow B}{H, x : A \vdash f(x) \in B}$$

We notice that the antecedent of this rule tells us that f can be abstracted as a morphism, so we define $\lambda(f) = curry(morph(f))$. Functionality follows since currying always preserves equality and f by assumption preserves equality.

(→-elim) $$\frac{H \vdash f(a) \in B}{H \vdash a \in A;\ H \vdash f \in A \rightarrow B}$$

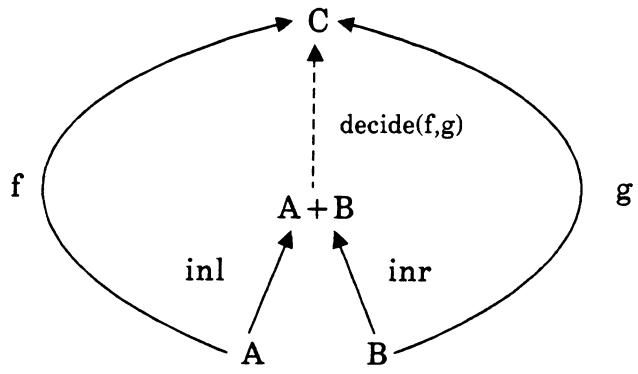

Figure 4.2: Coproduct Types

Here $f(a)$ denotes $eval(A, B) \cdot (a \times f)$, i.e. the application of f to a. Functionality follows trivially.

These types and rules are called *propositional* since they are obviously closely connected to intuitionistic propositional logic with the standard propositions-as-types interpretation.

4.3 The Integer Type

The type *Int* is the natural numbers object; it is used to perform induction.

(*Int*-intro) $$\frac{H \vdash 0 \in Int \qquad H \vdash S(n) \in Int}{H \vdash n \in Int}$$

Here, 0 and S are just the zero and successor morphisms of the natural numbers object.

(*Int*-elim) $$\frac{H \vdash iter(b, f) \in Int \to T}{H \vdash n \in Int; H \vdash b \in T; H \vdash f : T \to T}$$

We know that t and f induces a unique morphism $\phi : Int \to T$, so we can merely define $iter(t, f) = \phi$. This is a very simple induction rule; we can also allow the type to vary.

(Int-elim) $$\frac{H, n : Int \vdash ind(n, b, f) \in T(n)}{H \vdash T \in Int \to \mathcal{U}_i;\ induction(b, f, T)}$$

where $induction(b, f, T)$ abbreviates:

$$H \vdash b \in T(0);\ H, n : Int, x : T(n) \vdash f(x) \in T(S(n))$$

Here, $ind(n, b, f)$ is simply the computation that produces $f^n(b)$. Notice, that the ind-term is just a computation; we cannot define an $ind(b, f)$-term with domain Int and hope to type it.

4.4 Sigma and Pi Types

We now describe the *quantification* types Σ and Π. If A is an object and B is a morphism $A \to \mathcal{U}_i$ we define:

- $\Pi(A, B)$ is the limit (product) of the diagram $X(A, B)$.
- $\Sigma(A, B)$ is the colimit (sum) of the diagram $X(A, B)$.

This captures the intuition that Π- and Σ-types are products and sums of their components; our rules mirror this fact.

(Σ-intro) $$\frac{H \vdash in(a, b) \in \Sigma(A, B)}{H \vdash a \in A; H \vdash b \in B(a)}$$

We define $in(a, b) = C(X(A, B), B(a)) \cdot b$, i.e. the composition of b with the injection morphism indexed by a. Functionality is no problem here.

(Σ-elim) $$\frac{H \vdash decide(g) : \Sigma(A, B) \to C}{H, x : A \vdash g(x) \in B(x) \to C}$$

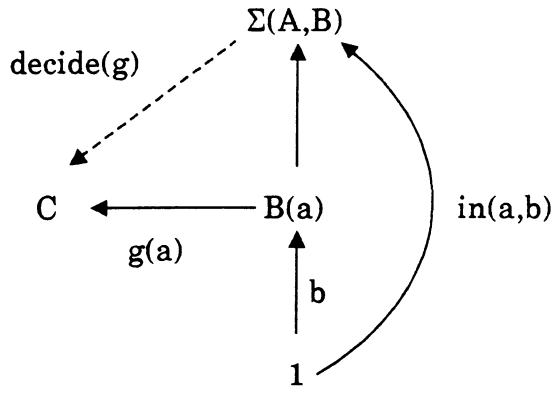

Figure 4.3: Σ Types

We first notice that g forms a cocone over $X(A,B)$; the hard part of proving this is to show that g denotes the same morphism on objects with the same tag in the diagram, but that is exactly the condition that g is functional. Hence, we can define $decide(g) = couniv(C,g,X(A,B))$, the unique morphism induced by g. Σ types are illustrated in figure 4.3. This rule is quite different from the νPrl rule, and it may not be obvious that we have defined a *strong existential type.* Recently this type constructor has attracted attention because its presence in an impredicative theory yields inconsistency. The question is whether we given an element of $\Sigma(A,B)$ can decide what its index is, i.e. if we have a morphism $proj : \Sigma(A,B) \rightarrow A$ such that for all a,b we have $proj \cdot in(a,b) = a$. We can easily get this by building the cocone (A,p) over $X(A,B)$ where p for each a contains the morphism $a \cdot couniv(B(a),\emptyset,\emptyset)$ from $B(a)$ to A; now we can define $proj = couniv(A,p,X(A,B))$.

(Π-intro) $$\frac{H \vdash \lambda(f) \in \Pi(A,B)}{H, x : A \vdash f(x) \in B(x)}$$

Again, we realize that f forms a cone over $X(A,B)$, and we can define $\lambda(f)$ to be $univ(1,f,X(A,B))$.

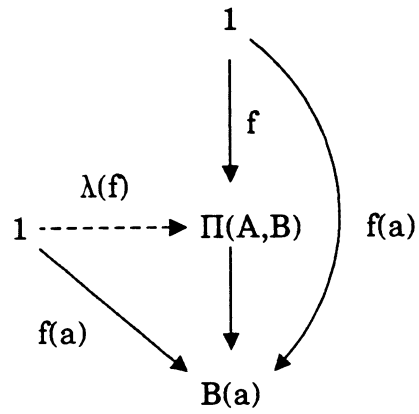

Figure 4.4: Π Types

(Π-elim) $$\frac{H \vdash f(a) \in B(a)}{H \vdash f \in \Pi(A,B);\ H \vdash a \in A}$$

Here we merely define $f(a) = L(X(A,B),B(a)) \cdot f$, i.e. f composed with the projection morphism indexed by a. Π types are illustrated in figure 4.4. Notice, how these rules are similar to the rules for the $\rightarrow$-type; this tells us that we could obtain the $\rightarrow$-type as a restricted version of the Π-type.

4.5 Subtypes and Quotient Types

To construct these types we first need an ***inhabitation predicate***, i.e. a morphism $I : \mathcal{U}_i \rightarrow \Omega$ such that for all A we have $I \cdot \mu_i(A) = \top$ iff A has an element. To get this we first build the diagram Δ described by the following computation: On input $< t, T >$ we verify that T is an object; next we verify that t is a morphism $1 \rightarrow T$; if we terminate at this point we output $< id(1), 1, 1, id(T), id(T) >$. This process defines Δ as the discrete diagram of 1's, one for each inhabited type. Now we build a cocone $(\mathcal{U}_i, k)$ over Δ, where k for each inhabited type T contains the morphism $\mu_i(T)$ from the corresponding copy of 1 to $\mathcal{U}_i$. Clearly, the induced morphism $\phi = couniv(\mathcal{U}_i, k, \Delta)$ is monic, so we can define $I = char(im(\phi))$, the

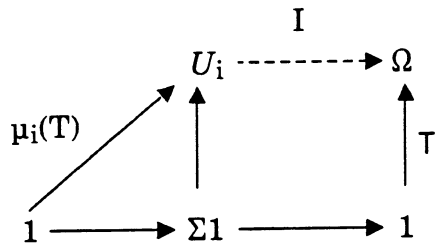

Figure 4.5: The Inhabitation Predicate

character of ϕ. This situation is illustrated in figure 4.5. With this in hand we can proceed.

If A is an object and $B : A \to \mathcal{U}_i$ is a morphism then

– $\{A \mid B\}$ is the limit of the diagram $S(A,B) = \{I \cdot B : A \to \Omega, \top : 1 \to \Omega\}$.

i.e. the subobject with character $I \cdot B$.

({}-intro) $$\frac{H \vdash sub(a) \in \{A \mid B\}}{H \vdash a \in A;\ H \vdash b \in B(a)}$$

Form the cone $c = \{proj : \Sigma(A,B) \to A, \top : \Sigma(A,B) \to \Omega\}$ over $S(A,B)$. We can now define $sub(a) = couniv(\Sigma(A,B), c, S(A,B)) \cdot a$.

({}-elim) $$\frac{H \vdash sup(x) \in A}{H \vdash x \in \{A \mid B\}}$$

We define $sup(x) = L(S(A,B),A) \cdot x$, i.e. x composed with the inclusion of the subobject into A. Subtypes are illustrated in figure 4.6. As these rules demonstrate, this subtype construction is very much like a Σ-type, except that the *proof* part has been supressed.

If A is an object and $B : A \times A \to \mathcal{U}_i$ is a morphism then

– A/B is the colimit of the diagram

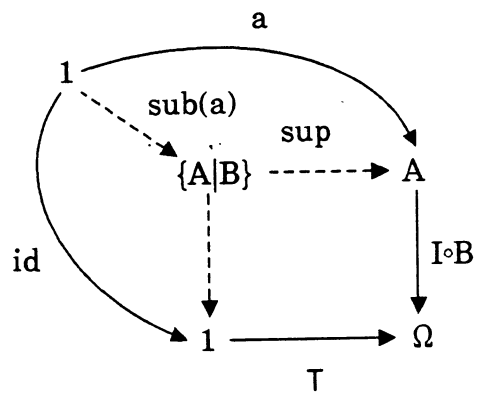

Figure 4.6: Subtypes

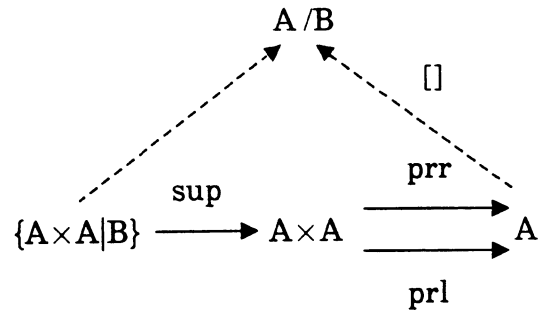

Figure 4.7: Quotient Types

$$Q(A,B) = \{sup : \{A \times A \mid B\} \rightarrow A \times A, prl, prr : A \times A \rightarrow A\}$$

We get an obvious introduction rule; each element in A has an equivalence class in A/B.

(/-intro) $$\frac{H \vdash [a] \in A/B}{H \vdash a \in A}$$

We can simply define $[a] = C(Q(A,B),A) \cdot a$. Quotient types are illustrated in figure 4.7. We would expect to get an elimination rule from the couniversal

properties of the quotient, but it turns out that we are unable to express it until we have defined *equality types*.

4.6 Recursive and Infinite Types

We now turn our attention to ω-inductively defined types, e.g. lists, trees, and streams. Recursive and infinite types in type theory are developed by Mendler, Constable, and Panangaden [5,17]. They are defined by morphisms from $\mathcal{U}_i$ to $\mathcal{U}_i$, with certain restrictions. Such a morphism maps objects of H_i to objects of H_i, but it is not defined on morphisms. We can only give meaning to morphisms $F : \mathcal{U}_i \to \mathcal{U}_i$ that can be extended to *covariant, continous or cocontinous endo-functors* on H_i; we call these morphisms *inductive*. The adjective (co)continous means that F must preserve (co)limits. This may seem like a severe requirement, but we have never encountered a common inductive (data)-type constructor that failed to meet it.

To illustrate this we focus on the most common class of inductive definitions: expressions over one variable using constants, $+$, $\times$, and $\to$. If we must insist that the variable does not occur in the domain of any $\to$-type, i.e. that the expressions must be *strongly positive*, we are guaranteed to get a covariant and continous functor. But the functor $x \mapsto (x \to A)$ is contravariant and neither continous nor cocontinous. All the strongly positive expressions, however, are continous. The $\to$-type constructor is, however, not cocontinous in its second argument either in general categories. In certain categories of domains the $\to$-functor is bicontinous in both its arguments, and one can actually solve equations such as $T = T \to T$ [19]. Thus we arrive at similar restrictions as [4], but with a completely different motivation. In [16] the requirements have been weakened to just *positive* expressions of which $x \mapsto ((x \to A) \to B)$ is an example. These

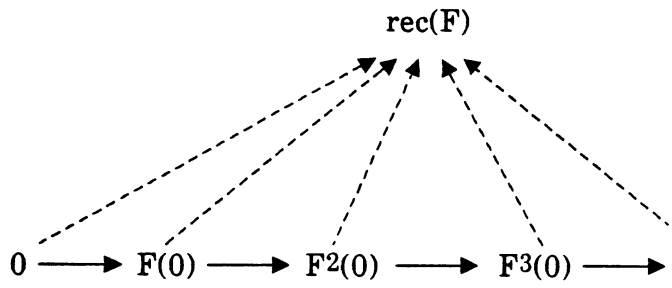

Figure 4.8: Cochain and Colimit

expressions are all covariant but demonstrably *not* continous or cocontinous; only the property of *monotonicity* is used by Mendler [16]. It is not entirely clear how we could use this broader definition; at the very least we would lose the property that an inductive type is *isomorphic* to its unfolding.

Let $F : \mathcal{U}_i \to \mathcal{U}_i$ be inductive. Then $\mathit{cochain}(F)$ is the diagram obtained by iterating F on the unique induced morphism $\mathit{couniv}(F(0), \emptyset, \emptyset) : 0 \to F(0)$, and $\mathit{chain}(F)$ is the diagram obtained by iterating F on the unique induced morphism $\mathit{univ}(F(1), \emptyset, \emptyset) : F(1) \to 1$. These diagrams are clearly both computable. We can now define:

- $\mathit{rec}(F)$ is the colimit of $\mathit{cochain}(F)$.

- $\mathit{inf}(F)$ is the limit of $\mathit{chain}(F)$.

These constructions are illustrated in figures 4.8 and 4.9. It is now clear why we must insist on covariant functors; a contravariant functor would alternate the direction of morphisms in (co)chains. When the functor is cocontinous it follows that the colimit of $F(\mathit{cochain}(F))$ is $F(\mathit{rec}(F))$; but $\mathit{cochain}(F)$ and $F(\mathit{cochain}(F))$ is essentially the same diagram, so couniversality tells us that $\mathit{rec}(F) \cong F(\mathit{rec}(F))$. Similarly, continuity yields $\mathit{inf}(F) \cong F(\mathit{inf}(F))$. However,

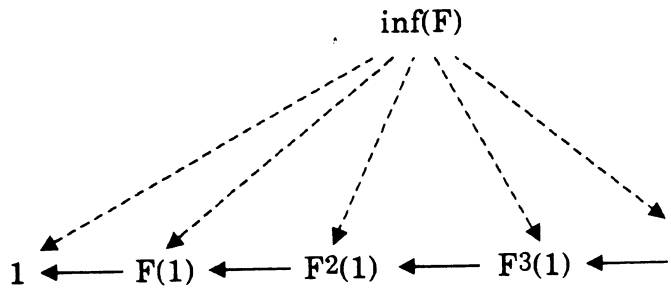

Figure 4.9: Chain and Limit

(co)continuity is not necessary to derive sensible induction rules, so it is conceivable that we could introduce a notion of monotonicity and define inductive types as in [16]. We can think of the recursive type as the *smallest* solution to $x \cong F(x)$ and correspondingly of the infinite type as the *largest* solution (the ordering being intuitive rather than formal). For example, if $F(x) = 1 + A \times x$ then $rec(F)$ defines *finite lists* and $inf(F)$ defines *streams*.

We get some very pleasing rules.

(*rec*-intro) $$\frac{H \vdash inc(n, x) \in rec(F)}{H, n : Int \vdash x \in F^n(0)}$$

Here $F^n(0)$ abbreviates $iter(F, 0)(n)$. The rule states that we can obtain an element of the recursive type by including an element from any unfolding. We can merely define $inc(n, x) = C(cochain(F), F^n(0)) \cdot x$. This introduction rules is illustrated in figure 4.10.

(*rec*-elim) $$\frac{H \vdash def(f) \in rec(F) \to B}{H \vdash f \in F(B) \to B}$$

This rule tells us how to define recursive functions on recursive types. Soundness is a little complicated to demonstrate. We clearly have the unique morphism

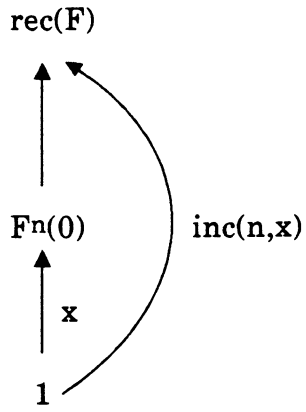

Figure 4.10: Introduction Rule for Recursive Types

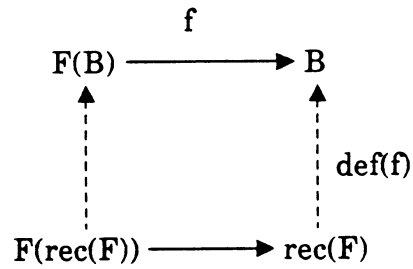

Figure 4.11: Elimination Rule for Recursive Types

from 0 to B; applying F yields a morphism from $F(0)$ to $F(B)$; composing with f we get a morphism from $F(0)$ to B. By iterating this we get a morphism from each $F^n(0)$ to B. But then we have a cocone p over $cochain(F)$ and we can define $def(f) = couniv(B, p, cochain(F))$. This elimination rule is illustrated in figue 4.11. To see how this rule works we shall define a lenght function on lists, i.e. a morphism $length : rec(F) \rightarrow Int$, where $F(x) = 1 + A \times x$. Our rule tells us to define a morphism $1 + A \times Int \rightarrow Int$; let $0 : 1 \rightarrow Int$ be the zero morphism and let $S : Int \rightarrow Int$ be the successor morphism. Then we can define $length = def(decide(0, S \cdot prr))$.

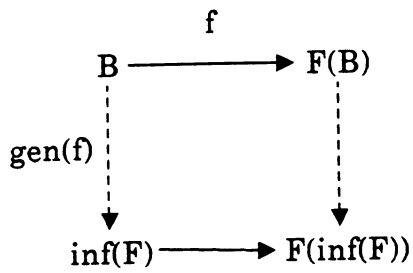

Figure 4.12: Introduction Rule for Infinite Types

The rules for infinite types are dual in nature.

(*inf*-intro) $$\frac{H \vdash gen(f) \in B \to inf(F)}{H \vdash f \in B \to F(B)}$$

We see that elements on an infinite type must be generated inductively. As before we start with the unique morphism from B to 1; applying F we get a morphism from $F(B)$ to $F(1)$; composing with f we get a morphism from B to $F(1)$. By iterating we get a morphism from B to each $F^n(1)$. But then we have a cone p over $chain(F)$ and we can define $gen(f) = univ(B, p, chain(F))$. This introduction rule is illustrated in figure 4.12.

(*inf*-elim) $$\frac{H, n : Int \vdash app(n, x) \in F^n(1)}{H \vdash x \in inf(F)}$$

The rule states that from an element of the infinite type we can get an element of any approximant. We simply define $app(n, x) = L(chain(F), F^n(1)) \cdot x$. This elimination rule is illustrated in figure 4.13.

It is tempting to speculate on what happens if we form the limit of a cochain or the colimit of a chain. This turns out not to be very interesting; regardless of what F is, we find that the colimit of $chain(F)$ is 1 and the limit of $cochain(F)$ is 0.

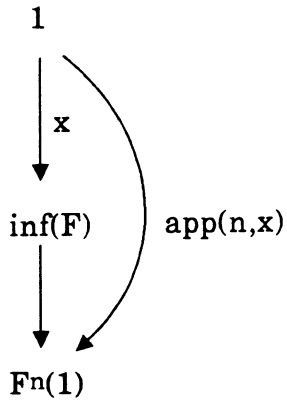

Figure 4.13: Elimination Rule for Infinite Types

It seems reasonable that we could handle transfinite recursive type definitions, too. In that case we would first form the ω-cochain as before, but then extend it with the objects $rec(F)$, $F(rec(F))$, $F^2(rec(F))$, and so on, which would produce a 2ω chain. This seems to generalize to arbitrary recursive ordinals, but we have not investigated this further.

4.7 Computation Rules

Till now, we have just given suggestive names to certain objects and morphisms and demonstrated how we could obtain rules similar to those of the νPrl theory. Throughout this discussion it has been tacitly assumed that we have chosen the *correct* interpretation, and we have certainly not violated out intuitive understanding of types and terms at any point. There is, however, a formal requirement that we must meet. A critical aspect of type theory is its computational content, i.e. the existence of a reduction relation on terms. The reduction relation is specified through computation rules, so to make any claim of modelling the

νPrl theory we must verify that all the reduction rules on terms are valid in our theory, i.e. they must all respect equality of morphisms. Of course, since we have different rules we will get different reductions, but they should be immediately recognizable as variations of the νPrl reduction rules. We use $\triangleright$ to denote reduction.

$\times$: $prl \cdot < a, b > \triangleright a$

$prr \cdot < a, b > \triangleright b$

$+$: $decide(f, g) \cdot inl(a) \triangleright f \cdot a$

$decide(f, g) \cdot inr(b) \triangleright g \cdot b$

$\rightarrow$: $\lambda(f)(a) \triangleright f(a)$

Σ: $decide(g) \cdot in(a, b) \triangleright g(a) \cdot b$

Π: $\lambda(f)(a) \triangleright f(a)$

$\{\}$: $sup \cdot sub(x) \triangleright x$

rec: $def(f)(inc(n, x)) \triangleright I(n) \cdot x$

$I(0) \triangleright !$

$I(S \cdot n) \triangleright f \cdot F(I(n))$

where ! is the unique morphism from 0 to B

inf: $app(0, gen(f)) \triangleright !$

$app(S \cdot n, gen(f)) \triangleright F(app(n, gen(f))) \cdot f$

where ! is the unique morphism from B to 1

Apart from these, we also have the reduction $\rhd_C$ from the underlying computation system; we use that in the following general reduction rule:

$$t \rhd_C s \Rightarrow t \rhd s$$

Notice, that since the elimination rule for the quotient type is missing, we cannot provide a reduction rule either. This hole will be filled out after we introduce equality types.

The soundness of these rules is immediate by expanding the definitions of the terms.

4.8 Equality Types

To fully express all the rules of the νPrl type constructors we must introduce a simple version of *equality types*. They are types of the form

$$\mathbf{I}(t_1, t_2, T)$$

that are inhabited iff t_1 and t_2 are equal terms of the type T. Hence, they serve to reflect part of the meta-theory of νPrl back into the theory. We have to add them explicitly, which means for each equality type finding an existing object that is equivalent to it with respect to inhabitation. Fortunately, this is very easy; we can take advantage of the *formulas-as-types* paradigm and merely spell out the equality predicates in the language of types. Thus, equality types are derived concepts in our theory. The translation is as follows.

Void: $\mathbf{I}(any(x), any(y), T) = 1$

$\times$: $\mathbf{I}(< a, b >, < a', b' >, A \times B) = \mathbf{I}(a, a', A) \times \mathbf{I}(b, b', B)$

$+$: $\mathbf{I}(inl(a), inl(a'), A + B) = \mathbf{I}(a, a', A)$

$\mathbf{I}(inr(b), inr(b'), A + B) = \mathbf{I}(b, b', B)$

$\mathbf{I}(inl(a), inr(b), A + B) = 0$

$\mathbf{I}(inr(b), inl(a), A + B) = 0$

$\rightarrow$: $\mathbf{I}(f, g, A \rightarrow B) = \Pi(a : A, I(f(a), g(a), B))$

Int: $\mathbf{I}(0, 0, Int) = 1$

$\mathbf{I}(S \cdot n, S \cdot m, Int) = \mathbf{I}(n, m, Int)$

$\mathbf{I}(0, S \cdot m, Int) = 0$

$\mathbf{I}(S \cdot n, 0, Int) = 0$

Σ: $\mathbf{I}(in(a, b), in(a', b'), \Sigma(A, B)) = \mathbf{I}(a, a', A) \times \mathbf{I}(b, b', B(a))$

Π: $\mathbf{I}(f, g, \Pi(A, B)) = \Pi(a : A, \mathbf{I}(f(a), g(a), B(a)))$

$\{\}$: $\mathbf{I}(sub(a), sub(a'), \{A \mid B\}) = I(a, a', A)$

$/$: $\mathbf{I}([a], [a'], A/B) = B(a, a')$

rec: $\mathbf{I}(inc(n, x), inc(m, y), rec(F)) = \mathbf{I}(n, m, Int) \times \mathbf{I}(x, y, F^n(0))$

inf: $\mathbf{I}(gen(f), gen(g), B \rightarrow inf(f)) = \mathbf{I}(f, g, B \rightarrow F(B))$

Furthermore, we have the following general rule for non-canonical terms:

$$t_i \rhd t_i' \Rightarrow \mathbf{I}(t_1, t_2, T) = \mathbf{I}(t_1', t_2', T)$$

We can illustrate the use of equality types by finally stating the elimination rule for the quotient type.

(/-elim)
$$\frac{H \vdash [f] : A/B \to C}{H \vdash f : A \to C; H, a : A, a' : A, B(a, a') \vdash I(f(a), f(a'), C)}$$

The content of this rule is that a function on A that respects B induces a function on the quotient. Soundness follows easily; define p to be the cocone

$$\{f \cdot prl \cdot sup : \{A \mid B\} \to C, f : A \to C\}$$

over $Q(A, B)$; commutativity follows from the assumption that f yields equal values on related elements. Now, we can define $[f] = couniv(C, p, Q(A, B))$, and we are done. The computation rule that accompanies this elimination form is

$$/: \qquad [f] \cdot [a] \rhd [f(a)]$$

Equality types in this simple form clearly do not add any thing to our theory, as they merely denote already existing types. They reflect equality statements back into the theory.

4.9 Pure Types and Terms

Earlier, we defined a *type* to be an *interesting* object. Looking back, we see that the type constructors we have described are very uniform, i.e. it is apparant that they will work on all objects and not just types. Hence, we are faced with the somewhat philosophical question whether $A \times B$ is a type even if A and B are not. Since there are no technical problems in giving an affirmative answer

we shall do so. This is merely a reflection of the fact that we have many more type constructors than we know what to do with. To remedy this situation we shall define a *pure type* to be an object that is built exclusively from the type constructors we have defined and a *pure term* to be a morphism that is built exclusively from the corresponding rules. If we only wanted to work with pure types and terms we could add that as an extra proof burden on our type constructors. These correspond to some of the *well-formedness conditions* for the νPrl theory, where many of the subgoal in formation rules are various well-formedness criteria that have been reflected back into the theory. Some of these we can avoid since we cannot have typed terms with untyped subterms, but others we have in common with νPrl, as for example functionality requirements for type constructors.

4.10 Diagrams as Type Constructors

When defining the previous type constructors it was clear that they arose in a very uniform fashion. In fact, any diagram-scheme D seems to induce two type constructors $\Lambda(D)$ and $\Xi(D)$ obtained through a limit and colimit construction over D. We can even say what their introduction and elimination rules are:

($\Lambda(D)$-intro) $$\frac{H \vdash univ(X,k,D)\cdot x \in lim(D)}{H \vdash (X,k,D)cone; H \vdash x \in X}$$

($\Lambda(D)$-elim) $$\frac{H \vdash L(D,A)\cdot y \in A}{H \vdash y \in lim(D); H \vdash A \in D}$$

($\Xi(D)$-intro) $$\frac{H \vdash C(D,A)\cdot a \in colim(D)}{H \vdash A \in D; H \vdash a \in A}$$

($\Xi(D)$-elim) $$\frac{H \vdash couniv(X,k,D)\cdot y \in X}{H \vdash (X,k,D)cocone; H \vdash y \in colim(D)}$$

These rules are not entirely formal, but they reveal a pattern that pervades throughout our work so far. Conversely, we are allowed to think of *any* diagram as defining a type constructor. It is now a deeply interesting question whether this plethora of type constructors really gives us *new* types; is it the case that some small set of type constructors is sufficient? We can provide a seemingly contradictory answer to this question: ***yes and no!***

Yes, because it is a well-know fact from category theory that with only products and equalizers we can build limits of all diagrams, and with coproducts and coequalizers we can build colimits of all diagrams. This means, that with *four* type constructors we can express all the others, at least up to isomorphism. Products and coproducts are just Σ and Π; equalizers and coequalizers are more obscure and not recognizable as existing type constructors. If we wanted to add them we could proceed as follows. Let $f, g : A \to B$ be two morphisms and let $P(f,g)$ be the *parallel-arrow* diagram consiting of f and g suspended between A and B. Now we define

- $EQ(f,g)$ is the limit of $P(f,g)$
- $COEQ(f,g)$ is the colimit of $P(f,g)$

We get the following rules.

(EQ-intro) $$\frac{H \vdash i(a) \in EQ(f,g)}{H \vdash a \in A;\ H \vdash I(f \cdot a, g \cdot a, B)}$$

(EQ-elim) $$\frac{H \vdash e(x) \in A}{H \vdash x \in EQ(f,g)}$$

($COEQ$-intro) $$\frac{H \vdash i(b) \in COEQ(f,g)}{H \vdash b \in B}$$

($COEQ$-elim) $$\frac{H \vdash e(h) \in COEQ(f,g) \to C}{H \vdash h \in B \to C;\ H \vdash I(f \cdot h, g \cdot h, A \to C)}$$

These rules are justified in the obvious way. It is clear that these types have no intuitive content, which is probably why they were not suggested in the first place.

The *no* part of our answer points out that the soul of type theory is to isolate *interesting* type constructors, for which we can give *meaningful* rules. Minimality is not a virtue in this context; recusive types may be superfluous, but it seems very unlikely that we could ever have invented useful induction rules without viewing them as constructors in their own right.

We can illustrate these ideas by constructing a new diagram and interpreting it as a type constructor. Look at the infinite, two-dimensional, grid-shaped diagram, built from the two inductive morphisms F and G, that is shown in figure 4.14; if we call the colimit of this diagram $dual(F, G)$; it is not hard to see that $dual(F, G) \cong rec(F) \times rec(G)$. Thus a seemingly new type constructor is reduced to a combination of existing ones.

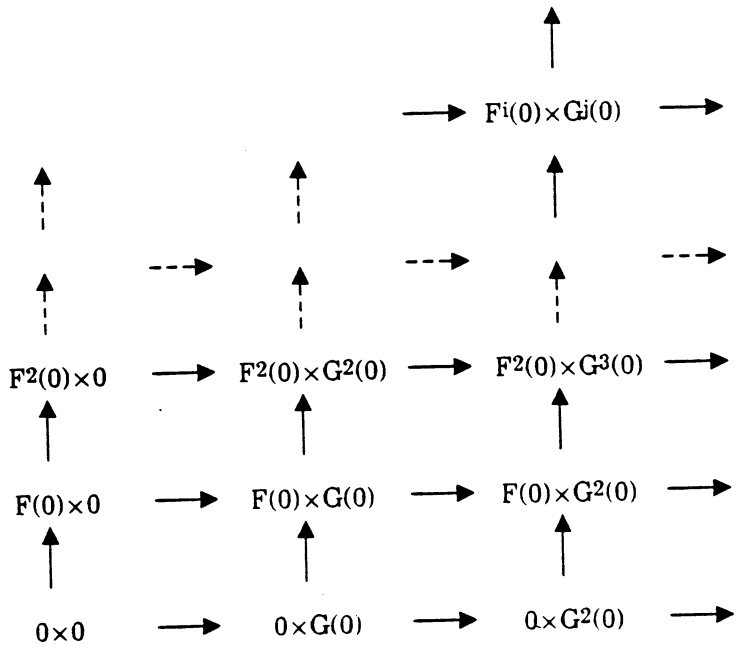

Figure 4.14: A Grid-shaped Diagram

Chapter 5

Conclusions

We now assess our achievements and point out future research directions.

5.1 The Impact of our Theory

The categorical type theory we have presented is a simplified and diagramatically inspired version of predicative type theory. Aside from some important details the theory is essentially a variant of Martin-Löf's theory and the νPrl theory. One of our principal contributions was to spell out a model theoretic notion of consistency. It argues heavily in favor of the consistency of comparable theories.

We have shown how different type constructors may be viewed as instances of simple, standard category theoretic constructions, viz. limits and colimits of computable diagrams. This realization may help to discover interesting introduction and elimination rules in a standard manner. Our consistency result immediately generalizes to all future type constructors that can be expressed in this way.

Our theory can serve as a tool for analysis of theories of this general nature;

in particular, one easily obtains results about the inconsistency of certain extensions. Thus for example, the notion *Type:Type* interpreted as the existence of a retract-universal type is easily shown to be inconsistent.

We have successfully managed to separate the notions of computations and typed terms. In the νPrl theory typed terms are obtained by typing arbitrary computations, and typed terms may contain untypeable subterms. In our theory typed terms are essentially built from typed combinators, and the computations enter as a way of describing diagrams.

5.2 Future Work

We see many interesting studies emerge from this project.

We hope to use our framework to analyze extensions of the theory; in particular, we are interested in *impredicative* higher-order constructs. These are present in the theory of constructions [6], which is built on top of the 2nd order polymorphic lambda calculus. It has a special sub-class of types called *propositions*, which possibly forms a topos. Thus, in this setting propositions are types, but some types are not propositions. Quantification over an arbitrary type yields a proposition. As the set of propositions is viewed as a type, a proposition can quantify over all propositions – a clearly impredicative phenomenon. We want to investigate whether our hierarchy of term categories can be identified as a stratification of some part of this theory.

Many meta-mathematical questions naturally arise: We would like to obtain some normalization results about our categorical theory. We expect to have certain *universe relativity* results, i.e. the validity of certain proofs in arbitrary

universe levels. Such results have been obtained for the νPrl theory by S. F. Allen in his upcoming thesis.

It is not known how to weaken the notion of continuity to allow for transfinite recursive types. We are at the present limited to such recursive definitions that reach their closure after ω many unfoldings. The work of N. P. Mendler [16] employs a notion of *monotonicity* that we aspire to express categorically.

Finally, we hope to find interesting models that can reveal more than just consistency. The set valued model we have provided merely formalizes our intuition that types may be viewed as sets with special properties. A topological model could perhapsm allow us to regain the notion of computational *approximations* that is at the heart of the Scott semantics. It is already the case that that elements of infinite types must be understood through their computational approximations.

Bibliography

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

[2] H. P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Volume 103 of *Studies in Logic*, North-Holland, 2nd and revised edition edition, 1984.

[3] R. Burstall and D. Rydeheard. Computational category theory. 1985. Typeset Manuscript.

[4] R. L. Constable et al. *Implementing Mathematics with the Nuprl Proof Development System.* Prentice-Hall, 1986.

[5] R. L. Constable and N. P. Mendler. Recursive definitions in type theory. In *Proceedings of Logics of Programs '85*, Springer-Verlag, 1985.

[6] T. Coquand and G. Huet. Constructions: a higher order proof system for mechanizing mathematics. In *Proceedings of EUROCAL85, Linz*, Springer-Verlag, 1985.

[7] P.-L. Curien. *Categorical Combinators, Sequential Algorithms, and Functional Programming.* Pitman-Wiley, 1986.

[8] J.-Y. Girard. *Interprétation fonctionelle et elimination des coupures dans l'arithmetique d'order superieur.* Ph.D. thesis, Paris, 1972.

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

[10] H. Herrlich and G. E. Strecker. *Category Theory: An Introduction.* Volume 1 of *Sigma Series in Pure Mathematics*, Heldermann Verlag, 1979.

[11] J. R. Hindley and J. P. Seldin. *Introduction to Combinators and Lambda Calculus.* Volume 1 of *London Mathematics Society Student Texts*, Cambridge University Press, 1986.

[12] J. Lambek. From lambda calculus to cartesian closed categories. In *To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, Academic Press, 1980.

[13] J. Lambek and P. J. Scott. *Introduction to Higher Order Categorical Logic.* Volume 7 of *Cambridge Studies in Advanced Mathematics*, Cambridge University Press, 1986.

[14] S. MacLane. *Categories for the Working Mathematician.* Volume 5 of *Graduate Texts in Mathematics*, Springer-Verlag, 1971.

[15] P. Martin-Löf. An intuitionistic theory of types: predicative part. In *Proceedings of Logic Colloquium '73*, North-Holland, 1975.

[16] N. P. Mendler. *Inductive Definition and Infinite Objects in Constructive Type Theory.* Ph.D. thesis, Cornell, 1986. In preparation.

[17] N. P. Mendler, P. Panangaden and R. L. Constable. Infinite objects in type theory. In *Proceedings of Symposium on Logic in Computer Science '86*, IEEE Computing Society Press, 1986.

[18] R. A. G. Seely. Locally cartesian closed categories and type theory. *Mathematical Proceedings of Cambridge Philosophical Society*, 95, 1984.

[19] M. B. Smyth and G. D. Plotkin. The category theoretic solution of recursive domain equations. *Siam Journal of Computing*, 11(4), 1982.