# Syntax and Semantics of Dependent Types

Martin Hofmann

## 1 Introduction

Dependent types are types which unlike simple types, such as products, function spaces, or natural numbers, depend on or vary with values. An example of a dependent type is a type of vectors or arrays $Vec_\sigma(M)$ of a given length $M : \mathbf{N}$ over some type $\sigma$. Its objects are $nil_\sigma : Vec_\sigma(0)$ and $Cons_\sigma(U,V) : Vec_\sigma(Suc(M))$ where $U : \sigma$ and $V : Vec_\sigma(M)$. We can now consider a function which given $x{:}\,\mathbf{N}$ returns a vector (over $\mathbf{N}$) of length $x$ and all entries 0. This function then has the type $\Pi x{:}\,\mathbf{N}.\,Vec_{\mathbf{N}}(x)$ — a type of functions with the property that the type of the result depends on the argument. The same algorithm could also be typed as $\mathbf{N} \rightarrow List(\mathbf{N})$. The point of the dependent typing is that it reveals more information about the function. Another example of this kind is the exception-free head function for vectors

$$Hd : \Pi x{:}\,\mathbf{N}.\,Vec_\sigma(Suc(x)) \rightarrow \sigma$$

which yields the first entry of a vector. The typing prevents the unwanted case that $Hd$ gets applied to the empty list. In this way the dependent typing circumvents the need for partial functions in certain cases.

Another source for type dependency comes from type variables and type universes. For instance, the type of monoids with carrier $X$

$$MON(X) \stackrel{\text{def}}{=} ((X \times X) \rightarrow X) \;\times\; X$$

is a type depending on $X$. Type variables such as $X$ above can be treated as ordinary variables using universes, that is types containing (names for) other types as members. The function constructing the free monoid on a type $X$ would then be given the type

$$\Pi X{:}\, U.MON(List(X))$$

where $U$ is such a universe.

The third important source for dependent types comes from the propositions-as-types analogy under which propositions (in constructive logic) are seen as types, namely the type of their proofs. For instance, a proof of $\phi \wedge \psi$ consists of a proof of $\phi$ and a proof of $\psi$; a proof of $\phi \supset \psi$ consists of a procedure which transforms a hypothetical proof of $\phi$ into a proof of $\psi$. Therefore, conjunction and implication can be identified with cartesian product and function space, respectively. Under this correspondence, predicates, i.e. propositional functions become dependent types. We will describe how to view the atomic equality predicate and universal and existential quantification as types and type formers. In fact universal quantification corresponds to the Π-type introduced above.

If propositions are ordinary types they can be part of other types. For instance, we can enrich the above-defined type of monoids of monoids by a proposition stating associativity and neutrality:

$$\begin{array}{l} MON'(X) \stackrel{\mathrm{def}}{=} \\ \Sigma\circ : (X \times X) \to X. \\ \Sigma e : X. \\ \quad (\forall x, y, z{:}\, X. \circ(\circ(x, y), z) = \circ(x, \circ(y, z))) \ \wedge \ (\forall x{:}\, X. \circ(e, x) = x \wedge \circ(x, e) = e) \end{array}$$

The Σ-type former is a generalisation of the cartesian product $\times$ to dependent types and corresponds under the propositions-as-types analogy to existential quantification. An object of the above generalised signature thus consists of

- an object $\circ$ of type $(X \times X) \to X$
- an object $e$ of type $X$
- a proof that $\circ$ is associative and $e$ is neutral

The main aim of this article is not so much to explain how to use dependent types to formalise constructive mathematics or to do program development and specification, but rather to introduce the reader to a tool for the meta-theoretic study of dependent type theory: category-theoretic semantics.

By semantics we understand a compositional assignment of mathematical objects to syntactic objects, for instance sets or sets to types and set-theoretic functions to (open) terms of the types. Such interpretation is performed with the aim of establishing consistency or conservativity of certain type-theoretic constructs, or simply in order to explain, motivate, and justify them. Due to type dependency the verification that such an interpretation indeed validates all the rules of type theory can be quite involved which is why it has proven useful to define a general abstract notion: the category-theoretic semantics, which is proven sound and complete once and for all. Then in order to obtain an interpretation of type theory one "only" needs to check that one has an instance of the semantic notion.

The semantics is not surprising (like maybe the set-theoretic semantics for first-order logic) in that we have an almost trivial completeness property like in the case of Heyting algebra semantics for intuitionistic logic. The point is that the soundness theorem is non-trivial and therefore some work can be saved when presenting a translation of the syntax as a model construction.

## 1.1 Overview

In the next section we give the syntax for an extensible calculus of dependent types which encompasses various "named" type theories like Martin-Löf's type theory or the Calculus of Constructions. In §§2.4 & 2.3 we introduce pre-syntax and syntactic context morphisms. Both are auxiliary syntactic notions required later to define the interpretation function and to construct a term model. §3 contains the material on category-theoretic semantics. It introduces categories with families which provide a category-theoretic counterpart of type dependency and form the backbone of the semantics. We compare this notion to related concepts and identify the additional structure required to model the type and term formers. Section 3 ends with the interpretation of the syntax in the semantic structures. Finally, §4 is devoted to an extended application of the material: we give an interpretation of types as "variable sets" (presheaves) and use it to establish conservativity of Martin-Löf's Logical Framework over ordinary type theory.

Every section or larger subsection except the last one ends with several exercises which either contain definitions or proofs which are similar to previously given ones and are required later, or contain applications of the material. The last section contains instead an overview of the literature on applications of semantic methods to dependent type theory as a suggestion for further reading.

This article is self-contained except for the presupposition of some very basic category theory in §§3 & 4 which have not been included as very good introductions are readily available. The required notions are summarised in the beginning of §3.

## 2 Formal systems for dependent types

A theory of dependent types is a formal system which mainly allows one to derive judgements of the form $M : \sigma$ (the term $M$ has type $\sigma$) and $\sigma$ type ($\sigma$ is a type). As types may contain terms, typing affects typehood and both kinds of judgements must be defined simultaneously. For instance, $\mathit{Vec}_\sigma(M)$ is a type if $\sigma$ is a type and $M : \mathbf{N}$. Furthermore, we usually want a notion of definitional equality to be built into the theory, for example we wish to consider $0 : \mathbf{N}$ and $0 + 0 : \mathbf{N}$ as (definitionally) equal terms and hence $\mathit{Vec}_\sigma(0)$ and $\mathit{Vec}_\sigma(0 + 0)$ as definitionally equal types: if $M : \mathit{Vec}_\sigma(0)$ then also $M : \mathit{Vec}_\sigma(0+0)$. This leads to two more forms of judgement: $M = N : \sigma$ ($M$ and $N$ are definitionally equal terms of type $\sigma$) and $\sigma = \tau$ type ($\sigma$ and $\tau$ are definitionally equal types). Finally, we must keep track of the types of the free variables occurring in a judgement; we cannot assert $x + y : \mathbf{N}$ unless we know that $x{:}\,\mathbf{N}$ and $y{:}\,\mathbf{N}$. Since such declarations may depend on each other like in $x{:}\,\mathbf{N}, y{:}\,\mathit{Vec}_{\mathbf{N}}(x)$ it is convenient to make all judgements relative to a list of variable declarations including at least the free variables occurring inside the judgement. Such lists of declarations are called contexts and sometimes also type assignments. Intuitively, a context $x_1{:}\,\sigma_1, \dots, x_n{:}\,\sigma_n$ is well-formed if each $\sigma_i$ is a type in the context $x_1{:}\,\sigma_1, \dots x_{i-1}{:}\,\sigma_{i-1}$ and the $x_i$ are pair-wise distinct. So typehood (and thus typing) affects context validity and we finally arrive at six kinds of judgements:

| | |
|---|---|
| $\vdash \Gamma\ \mathit{ctxt}$ | $\Gamma$ is a valid context |
| $\Gamma \vdash \sigma$ type | $\sigma$ is a type in context $\Gamma$ |
| $\Gamma \vdash M : \sigma$ | $M$ is a term of type $\sigma$ in context $\Gamma$ |
| $\vdash \Gamma = \Delta\ \mathit{ctxt}$ | $\Gamma$ and $\Delta$ are definitionally equal contexts |
| $\Gamma \vdash \sigma = \tau$ type | $\sigma$ and $\tau$ are definitionally equal types in context $\Gamma$ |
| $\Gamma \vdash M = N : \sigma$ | $M$ and $N$ are def. equal terms of type $\sigma$ in context $\Gamma$. |

Well-formedness and equality of contexts can be defined in terms of the other judgements, but it is technically easier to include them as primary notions.

Regardless of which rules we later introduce to describe particular type and term formers such as the natural numbers or Π-types we always have the following structural rules.

- Rules for context formation:

$$\frac{}{\vdash \diamond\ \mathit{ctxt}}\ \text{C-Emp} \qquad \frac{\Gamma \vdash \sigma \text{ type}}{\vdash \Gamma, x{:}\,\sigma\ \mathit{ctxt}}\ \text{C-Ext}$$

$$\frac{\vdash \Gamma = \Delta\ \mathit{ctxt} \qquad \Gamma \vdash \sigma = \tau \text{ type}}{\vdash \Gamma, x{:}\,\sigma = \Delta, y{:}\,\tau\ \mathit{ctxt}}\ \text{C-Ext-Eq}$$

  The variables $x$ and $y$ in rules C-Ext and C-Ext-Eq are assumed to be fresh.

- The variable rule

$$\frac{\vdash \Gamma, x{:}\,\sigma, \Delta\ \mathit{ctxt}}{\Gamma, x{:}\,\sigma, \Delta \vdash x : \sigma}\ \text{Var}$$

- Rules expressing that definitional equality is an equivalence relation:

$$\frac{\vdash \Gamma\ \mathit{ctxt}}{\vdash \Gamma = \Gamma\ \mathit{ctxt}}\ \text{C-Eq-R}$$

$$\frac{\vdash \Gamma = \Delta\ \mathit{ctxt}}{\vdash \Delta = \Gamma\ \mathit{ctxt}}\ \text{C-Eq-S}$$

$$\frac{\vdash \Gamma = \Delta\ \mathit{ctxt} \qquad \vdash \Delta = \Theta\ \mathit{ctxt}}{\vdash \Gamma = \Theta\ \mathit{ctxt}}\ \text{C-Eq-T}$$

$$\frac{\Gamma \vdash \sigma \text{ type}}{\Gamma \vdash \sigma = \sigma \text{ type}} \quad \text{Ty-Eq-R}$$

$$\frac{\Gamma \vdash \sigma = \tau \text{ type}}{\Gamma \vdash \tau = \sigma \text{ type}} \quad \text{Ty-Eq-S}$$

$$\frac{\Gamma \vdash \sigma = \tau \text{ type} \qquad \Gamma \vdash \tau = \rho \text{ type}}{\Gamma \vdash \sigma = \rho \text{ type}} \quad \text{Ty-Eq-T}$$

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash M = M : \sigma} \quad \text{Tm-Eq-R}$$

$$\frac{\Gamma \vdash M = N : \sigma}{\Gamma \vdash N = M : \sigma} \quad \text{Tm-Eq-S}$$

$$\frac{\Gamma \vdash M = N : \sigma \qquad \Gamma \vdash N = O : \sigma}{\Gamma \vdash M = O : \sigma} \quad \text{Tm-Eq-T}$$

- Rules relating typing and definitional equality:

$$\frac{\Gamma \vdash M : \sigma \qquad \vdash \Gamma = \Delta \ \mathit{ctxt} \qquad \Gamma \vdash \sigma = \tau \text{ type}}{\Delta \vdash M : \tau} \quad \text{Tm-Conv}$$

$$\frac{\vdash \Gamma = \Delta \ \mathit{ctxt} \qquad \Gamma \vdash \sigma \text{ type}}{\Delta \vdash \sigma \text{ type}} \quad \text{Ty-Conv}$$

- For convenience (cf. E2.7 below) we also introduce the following weakening and substitution rules where $\mathcal{J}$ ranges over one of the judgements $M : \sigma$, $\sigma$ type, $M = N : \sigma$, $\sigma = \tau$ type.

$$\frac{\Gamma, \Delta \vdash \mathcal{J} \qquad \Gamma \vdash \rho \text{ type}}{\Gamma, x{:}\, \rho, \Delta \vdash \mathcal{J}} \quad \text{Weak}$$

$$\frac{\Gamma, x{:}\, \rho, \Delta \vdash \mathcal{J} \qquad \Gamma \vdash U : \rho}{\Gamma, \Delta[U/x] \vdash \mathcal{J}[U/x]} \quad \text{Subst}$$

Here $\mathcal{J}[U/x]$ (and similarly $\Delta[U/x]$) denotes the capture-free substitution of $U$ for $x$ in $\mathcal{J}$. This means that bound variables in $\mathcal{J}$ are systematically renamed so as to prevent any free variables in $U$ from becoming bound in $\mathcal{J}[U/x]$. We will henceforth consider all contexts, types, and terms as equal if they agree up to names of bound variables and assume the existence of a capture-free substitution function on these equivalence classes. One can use a de Bruijn style presentation of the syntax to avoid this identification. A good reference is (Huet 1990). The de Bruijn presentation gives rise to canonical representatives of the equivalence classes and yields an algorithm implementing capture-free substitution.

## 2.1 Type formers

Type and term formers are introduced by formation, introduction, elimination, and equality rules. There is no definitive set of type formers and new ones can be invented as needed. We present several of them to give an idea of the general pattern.

### 2.1.1 Dependent function space

The dependent function space also called dependent product or Π-type corresponds to the set-theoretic notion of cartesian product over a family of sets $\Pi_{i \in I} B_i$ which has as elements functions mapping an index $i$ to an element of the corresponding set $B_i$. In type theory this is expressed as follows:

$$\frac{\Gamma \vdash \sigma \text{ type} \qquad \Gamma, x{:}\,\sigma \vdash \tau \text{ type}}{\Gamma \vdash \Pi x{:}\,\sigma.\tau \text{ type}} \; \Pi\text{-F}$$

$$\frac{\Gamma \vdash \sigma_1 = \sigma_2 \text{ type} \quad \Gamma, x{:}\,\sigma \vdash \tau_1 = \tau_2 \text{ type}}{\Gamma \vdash \Pi x{:}\,\sigma_1.\tau_1 = \Pi x{:}\,\sigma_2.\tau_2 \text{ type}} \; \Pi\text{-Eq}$$

The first rule expresses that a dependent function space consists of a type $\sigma$ (possibly depending on other types recorded in $\Gamma$) and a type depending on $\sigma$ (and $\Gamma$), viz. $\tau$. The rule Π-Eq expresses that definitional equality is respected by the Π-former. The variable $x$ becomes bound in $\Pi x{:}\,\sigma.\tau$ and thus this type is subject to the convention on renaming of variables set out above. To form elements of the Π-type we have the introduction rule with associated congruence rule:

$$\frac{\Gamma, x{:}\,\sigma \vdash M : \tau}{\Gamma \vdash \lambda x{:}\,\sigma.M^{\tau} : \Pi x{:}\,\sigma.\tau} \; \Pi\text{-I}$$

$$\frac{\Gamma, x{:}\,\sigma \vdash M_1 = M_2 : \tau \quad \Gamma \vdash \sigma_1 = \sigma_2 \text{ type} \quad \Gamma, x{:}\,\sigma \vdash \tau_1 = \tau_2 \text{ type}}{\Gamma \vdash \lambda x{:}\,\sigma_1.M_1^{\tau_1} = \lambda x{:}\,\sigma_2.M_2^{\tau_2} : \Pi x{:}\,\sigma_1.\tau_1} \; \Pi\text{-I-Eq}$$

So to give an element of $\Pi x{:}\,\sigma.\tau$ one must give an element of $\tau[x]$ in the presence of a variable $x$ of type $\sigma$. The congruence rule Π-I-Eq expresses that definitional equality preserves Π-introduction ($\lambda$). We will henceforth refrain from writing down congruence rules. Such rules are silently understood for every type and term former we will introduce.

Elements of a Π-type are consumed using application like in the set-theoretic situation where an element of $\Pi_{i \in I} B_i$ and a specific $i_0 \in I$ gives an element of $B_{i_0}$:

$$\frac{\Gamma \vdash M : \Pi x{:}\,\sigma.\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash App_{[x:\sigma]\tau}(M, N) : \tau[N/x]} \; \Pi\text{-E}$$

Notice that the square brackets in the typing annotation $[x\colon\sigma]\tau$ are an integral part of the term former for application and indicate the binding of $x$ in $\tau$. Now we encounter a source for definitional equality: applying a function $\lambda x\colon\sigma.M^\tau$ to a term $N:\sigma$ results in $M$ with $x$ replaced by $N$:

$$\frac{\Gamma\vdash\lambda x\colon\sigma.M^\tau:\Pi x\colon\sigma.\tau\quad\Gamma\vdash N:\sigma}{\Gamma\vdash \mathit{App}_{[x:\sigma]\tau}(\lambda x\colon\sigma.M^\tau,N)=M[N/x]:\tau[N/x]}\ \Pi\text{-C}$$

Notice that substitution plays a more prominent role in dependently typed calculi as it is needed to formulate even the typing rules, not only the conversion rules (as in the case of simply-typed lambda calculus).

The attribute "definitional" for en equation like $\Pi$-C is certainly arguable. It is motivated by the understanding of $\lambda x\colon\sigma.M^\tau$ as a canonical element of $\Pi x\colon\sigma.\tau$ and application as a derived concept defined by equation $\Pi$-C. This view becomes important if one wants to see type theory as a foundation of constructive mathematics which accordingly is to be justified by a philosophical argument rather than via an interpretation in some other system, see (Martin-Löf 1975;(1984)). For us the distinction between canonical and non-canonical elements is not important. However, we will use it to motivate further definitional equalities.

A more pragmatic explanation for $\Pi$-C is that in applications one often uses abstraction as a means for making definitions and application to instantiate a definition. Thus when the $\Pi$-type is used in this way then both sides of $\Pi$-C are indeed definitionally equal in the proper sense of the word.

### 2.1.2 Dependent sum

The next type former we introduce is the $\Sigma$-type (or dependent sum) corresponding to disjoint union in set theory. If we are given a family of sets $(B_i)_{i\in I}$ we can form the set $\Sigma_{i\in I}B_i \overset{\text{def}}{=} \{(i,b)\mid i\in I\wedge b\in B_i\}$ whose elements consist of an index $i$ and an element of the corresponding set $B_i$. In type theory the corresponding formation and introduction rules look as follows.

$$\frac{\begin{array}{c}\Gamma\vdash\sigma\ \text{type}\\ \Gamma,x\colon\sigma\vdash\tau\ \text{type}\end{array}}{\Gamma\vdash\Sigma x\colon\sigma.\tau\ \text{type}}\ \Sigma\text{-F}\qquad\frac{\begin{array}{c}\Gamma\vdash M:\sigma\\ \Gamma\vdash N:\tau[M/x]\end{array}}{\Gamma\vdash \mathit{Pair}_{[x:\sigma]\tau}(M,N):\Sigma x\colon\sigma.\tau}\ \Sigma\text{-I}$$

The elimination rule looks a bit complicated at first sight:

$$\frac{\begin{array}{c}\Gamma,z\colon\Sigma x\colon\sigma.\tau\vdash\rho\ \text{type}\\ \Gamma,x\colon\sigma,y\colon\tau\vdash H:\rho[\mathit{Pair}_{x:\sigma.\tau}(x,y)/z]\\ \Gamma\vdash M:\Sigma x\colon\sigma.\tau\end{array}}{\Gamma\vdash\mathsf{R}^{\Sigma}_{[z:\Sigma x:\sigma.\tau]\rho}([x\colon\sigma.y\colon\tau]H,M):\rho[M/z]}\ \Sigma\text{-E}$$

Here the variable $z$ in $\rho$ and the variables $x,y$ in $H$ become bound inside $\mathsf{R}^\Sigma$ as indicated by the square brackets. The idea behind $\mathsf{R}^\Sigma$ is that in order to give a (possibly dependent) function out of $\Sigma x\colon \sigma.\tau$, it is enough to specify it on canonical elements, viz. the pairs. This is expressed by the following definitional equality

$$\frac{\Gamma \vdash \mathsf{R}^{\Sigma}_{[z:\Sigma x:\sigma.\tau]\rho}([x\colon \sigma, y\colon \tau]H, \mathit{Pair}_{[x:\sigma]\tau}(M,N)) : \rho[\mathit{Pair}_{[x:\sigma]\tau}/z]}{\begin{array}{r}\Gamma \vdash \mathsf{R}^{\Sigma}_{[z:\Sigma x:\sigma.\tau]\rho}([x\colon \sigma, y\colon \tau]H, \mathit{Pair}_{[x:\sigma]\tau}(M,N)) = \\ H[M/x, N/y] : \rho[\mathit{Pair}_{[x:\sigma]\tau}/z]\end{array}} \quad \Sigma\text{-C}$$

which says that a function on $\Sigma x\colon \sigma.\tau$ defined using the eliminator $\mathsf{R}^{\Sigma}$ behaves on canonical elements as specified by the argument $H$. As an example we show how to define projections for the $\Sigma$-type. Assume $\Gamma \vdash \sigma$ type, $\Gamma, x\colon \sigma \vdash \tau$ type, and $\Gamma \vdash M : \Sigma x\colon \sigma.\tau$. We define

$$M.1 \overset{\text{def}}{=} \mathsf{R}^{\Sigma}_{[z:\Sigma x:\sigma.\tau]\sigma}([x\colon \sigma, y\colon \tau]x, M) : \sigma$$

Now, in the particular case where $M$ is canonical, i.e. $M \equiv \mathit{Pair}_{[x:\sigma]\tau}(U,V)$ the rule $\Sigma$-C gives $\Gamma \vdash M.1 = U : \sigma$ as expected. A second projection can be defined similarly:

$$M.2 \overset{\text{def}}{=} \mathsf{R}^{\Sigma}_{[z:\Sigma x:\sigma.\tau]\tau[z.1/x]}([x\colon \sigma, y\colon \tau]y, M) : \tau[M.1]$$

Notice that the definiens (the right-hand side) is well-typed by virtue of rules $\Sigma$-C and Ty-Conv which allow us to conclude

$$\Gamma, x\colon \sigma, y\colon \tau \vdash y : \tau[(\mathit{Pair}_{[x:\sigma]\tau}(x,y).1)/x]$$

One can restrict the elimination operator $\mathsf{R}^{\Sigma}$ to those cases where the type $\rho$ does not depend on $\Sigma x\colon \sigma.\tau$. One can then still define the first projection, but no longer the second one. This is called weak $\Sigma$-elimination, see (Luo 1994).

Important special cases of $\Pi x\colon \sigma.\tau$ and $\Sigma x\colon \sigma.\tau$ arise when $\tau$ does not actually depend on $\sigma$. In this case, i.e. when $\Gamma \vdash \sigma$ type and $\Gamma \vdash \tau$ type we write

$$\sigma \to \tau \overset{\text{def}}{=} \Pi x\colon \sigma.\tau$$

and

$$\sigma \times \tau \overset{\text{def}}{=} \Sigma x\colon \sigma.\tau$$

indicating that in these cases the $\Pi$- and $\Sigma$-types correspond to ordinary non-dependent function space and cartesian product, respectively.

A (constructive) proof of an existential statement $\exists x\colon \sigma.P(x)$ consists of an element $M$ of $\sigma$ (the witness) together with a proof that $P(M)$, that is an element of $P(M)$. Thus under the propositions-as-types analogy the $\Sigma$-type is the counterpart to existential quantification.

### 2.1.3 Natural numbers

An example of a basic type is provided by the type of natural numbers given by the rules

$$\frac{\vdash \Gamma\ \mathit{ctxt}}{\Gamma \vdash \mathbf{N}\ \text{type}}\ \text{N-F} \qquad \frac{\vdash \Gamma\ \mathit{ctxt}}{\Gamma \vdash 0 : \mathbf{N}}\ \text{N-I-0} \qquad \frac{\Gamma \vdash M : \mathbf{N}}{\Gamma \vdash \mathit{Suc}(M) : \mathbf{N}}\ \text{N-I-S}$$

The elimination rule is similar to the one for $\Sigma$-types; in order to define a (possibly dependent) function on $\mathbf{N}$ it is enough to give it on the canonical elements 0 and $Suc(M)$. In the case of $Suc(M)$ the function may be called (primitive) recursively for $M$.

$$\frac{\begin{array}{c}\Gamma, n{:}\,\mathbf{N} \vdash \sigma\ \text{type} \\ \Gamma \vdash H_z : \sigma[0/n] \\ \Gamma, n{:}\,\mathbf{N}, x{:}\,\sigma \vdash H_s : \sigma[\mathit{Suc}(n)/n] \\ \Gamma \vdash M : \mathbf{N}\end{array}}{\Gamma \vdash \mathsf{R}^{\mathbf{N}}_{[n:\mathbf{N}]\sigma}(H_z, [n{:}\,\mathbf{N}, x{:}\,\sigma]H_s, M) : \sigma[M/n]}\ \text{N-E}$$

The primitive-recursive behaviour of $\mathsf{R}^{\mathbf{N}}$ is expressed by the following two rules for definitional equality:

$$\frac{\Gamma \vdash \mathsf{R}^{\mathbf{N}}_{[n:\mathbf{N}]\sigma}(H_z, [n{:}\,\mathbf{N}, x{:}\,\sigma]H_s, 0) : \sigma[0/n]}{\Gamma \vdash \mathsf{R}^{\mathbf{N}}_{[n:\mathbf{N}]\sigma}(H_z, [n{:}\,\mathbf{N}, x{:}\,\sigma]H_s, 0) = H_z : \sigma[0/n]}\ \text{N-C-0}$$

$$\frac{\Gamma \vdash \mathsf{R}^{\mathbf{N}}_{[n:\mathbf{N}]\sigma}(H_z, [n{:}\,\mathbf{N}, x{:}\,\sigma]H_s, \mathit{Suc}(M)) : \sigma[\mathit{Suc}(M)/n] \qquad \text{N-C-S}}{\begin{array}{l}\Gamma \vdash \mathsf{R}^{\mathbf{N}}_{[n:\mathbf{N}]\sigma}(H_z, [n{:}\,\mathbf{N}, x{:}\,\sigma]H_s, \mathit{Suc}(M)) = \\ \qquad\qquad H_s[M/n, \mathsf{R}^{\mathbf{N}}_{[n:\mathbf{N}]\sigma}(H_z, [n{:}\,\mathbf{N}, x{:}\,\sigma]H_s, M)/x] : \sigma[\mathit{Suc}(M)/n]\end{array}}$$

The elimination rule for natural numbers allows for both the definition of functions by primitive recursion and proof of properties of the natural numbers by mathematical induction. For instance, we can define addition as follows

$$M + N \overset{\text{def}}{=} \mathsf{R}^{\mathbf{N}}_{[n:\mathbf{N}]\mathbf{N}}(N, [n{:}\,\mathbf{N}, x{:}\,\mathbf{N}]\mathit{Suc}(x), M) : \mathbf{N}$$

and—writing $\bar{n}$ for the closed term $\underbrace{\mathit{Suc}(\dots \mathit{Suc}(}_{n\ \text{times}} 0) \dots)$—we have

$$\diamond \vdash \bar{n} + \bar{m} = \overline{m + n} : \mathbf{N}$$

for all (set-theoretic) natural numbers $m$, $n$ by $m$-fold application of rule N-C-S followed by N-C-Z. We will see an example of the use of mathematical induction below.

### 2.1.4 Notation

We will henceforth freely suppress type annotations if this increases readability. For instance, we may write $\lambda x{:}\,\sigma.M$ or even $\lambda x.M$ instead of $\lambda x{:}\,\sigma.M^{\tau}$. We sometimes omit a prevailing context $\Gamma$ and thus write $\vdash \mathcal{J}$ instead of $\Gamma \vdash \mathcal{J}$. We write $\diamond \vdash J$ if we want to emphasise that a judgement holds in the empty context. If $\sigma$ contains among others the free variable $x$ then we can write $\sigma[x]$ to emphasise this and use the notation $\sigma[M]$ for $\sigma[M/x]$ in this case.

In implementations of type theory many more such conventions are being used and sometimes they are even made part of the official syntax. It is always an important question whether such shorthands should be treated formally or informally. Here we have decided to have a syntax as explicit as possible so as to facilitate its meta-theoretic study. For doing proofs within the theory obviously the syntactic sugar is unavoidable.

### 2.1.5 Identity types

As we have explained, definitional equality is the congruence generated by the computational equations like N-C-Z and $\Pi$-C. Its main purpose is to facilitate the construction of inhabitants of types; in some examples, like the definition of the second projection for $\Sigma$-types above, its use is unavoidable. However, definitional equality is merely a judgement, and not a type, that is, not a proposition, and therefore cannot be established by induction, i.e., using $\mathsf{R}^{\mathbf{N}}$ or $\mathsf{R}^{\Sigma}$. Also, we cannot have definitional equalities as assumptions in a context. In order to enable equality reasoning inside type theory one is therefore lead to introduce a type corresponding to equality—the identity type. For every two terms of the same type we have a (not necessarily inhabited) type of proofs of their equality

$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \mathit{Id}_\sigma(M, N) \text{ type}} \quad \text{Id-F}$$

and the identity types have canonical inhabitants corresponding to reflexivity

$$\frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathit{Refl}_\sigma(M) : \mathit{Id}_\sigma(M, M)} \quad \text{Id-I}$$

We call two terms $\Gamma \vdash M, N : \sigma$ propositionally equal if the type $\Gamma \vdash \mathit{Id}_\sigma(M, N)$ is inhabited. By the (implicit) congruence rules and rule Ty-Conv propositional equality extends definitional equality, that is, we have $\Gamma \vdash \mathit{Refl}_\sigma(M) : \mathit{Id}_\sigma(M, N)$ provided $\Gamma \vdash M = N : \sigma$.

So far we only know that propositional equality is a reflexive relation. The further properties like symmetry, transitivity, Leibniz' principle are all consequences of the following elimination rule for identity types

$$\frac{\begin{array}{l}\Gamma \vdash \sigma \text{ type} \\ \Gamma\,,\, x{:}\,\sigma, y{:}\,\sigma\,,\, p{:}\, \mathit{Id}_\sigma(x,y) \vdash \tau \text{ type} \\ \Gamma, z{:}\,\sigma \vdash H : \tau[z/x, z/y, \mathit{Refl}_\sigma(z)/p] \\ \Gamma \vdash M{:}\,\sigma \qquad \Gamma \vdash N{:}\,\sigma \\ \Gamma \vdash P : \mathit{Id}_\sigma(M,N)\end{array}}{\Gamma \vdash \mathsf{R}^{\mathit{Id}}_{[x:\sigma,y:\sigma,p:\mathit{Id}_\sigma(x,y)]\tau}([z{:}\,\sigma]H\,,\, M, N, P) : \tau[M/x, N/y, P/p]} \quad \text{Id-E}$$

and the associated equality rule

$$\frac{\Gamma \vdash \mathsf{R}^{\mathit{Id}}_{[x:\sigma,y:\sigma,p:\mathit{Id}_\sigma(x,y)]\tau}([z{:}\,\sigma]H\,,\, M, M, \mathit{Refl}_\sigma(M)) : \tau[M/x, M/y, \mathit{Refl}_\sigma(M)/p]}{\begin{array}{l}\Gamma \vdash \mathsf{R}^{\mathit{Id}}_{[x:\sigma,y:\sigma,p:\mathit{Id}_\sigma(x,y)]\tau}([z{:}\,\sigma]H\,,\, M, M, \mathit{Refl}_\sigma(M)) \\ \qquad = H[M/z] : \tau[M/x, M/y, \mathit{Refl}_\sigma(M)/p]\end{array}} \quad \text{Id-C}$$

The eliminator $\mathsf{R}^{\mathit{Id}}$ is an induction principle like $\mathsf{R}^{\mathbf{N}}$ and $\mathsf{R}^{\Sigma}$ which roughly states that every element of an indentity type behaves as if it were a canonical one of the form $\mathit{Refl}_\sigma(M)$. We demonstrate how to derive Leibniz' principle from $\mathsf{R}^{\mathit{Id}}$: Suppose that $x{:}\,\sigma \vdash \rho[x]$ type, and that we are given two propositionally equal terms of type $\sigma$, i.e., $\vdash M, N : \sigma$ and $\vdash P : \mathit{Id}_\sigma(M,N)$. If $\vdash H : \rho[M]$ then we can construct an element $\mathit{Subst}_{[x:\sigma]\rho}(P,H)$ of $\rho[N]$ as follows. We define

$$\tau[x{:}\,\sigma, y{:}\,\sigma, p{:}\,\mathit{Id}_\sigma(x,y)] \stackrel{\text{def}}{=} \rho[x] \Rightarrow \rho[y]$$

(Recall that $\phi \Rightarrow \psi$ abbreviates $\Pi x{:}\,\phi.\psi$.) Now $\lambda h{:}\,\rho[x].h$ is an inhabitant of $x{:}\,\sigma \vdash \tau[x, x, \mathit{Refl}_\sigma(x)]$, so

$$\mathit{Subst}_{[x:\sigma]\rho}(P,H) \stackrel{\text{def}}{=} \mathit{App}(\mathsf{R}^{\mathit{Id}}_{[x:\sigma,y:\sigma,p:\mathit{Id}(x,y)]}([x{:}\,\sigma]\lambda h{:}\,\rho[x].h\,,\, M, N, P)\,,\, H) : \rho[N]$$

and from Id-C and $\Pi$-C we get the derived rule $\mathit{Subst}_{[x:\sigma]\rho}(\mathit{Refl}_\sigma(M), H) = H : \rho[M]$.

From *Subst* we can derive symmetry, transitivity, and congruence properties of propositional equality in the usual way. For example, if $x{:}\,\sigma \vdash U[x] : \tau$ and $\vdash P : \mathit{Id}_\sigma(M,N)$ then

$$\begin{array}{l}\mathit{Resp}_{\sigma,\tau}([x{:}\,\sigma]U\,,\, P) \stackrel{\text{def}}{=} \\ \qquad \mathit{Subst}_{[x:\sigma]\mathit{Id}_\tau(U[M],U[x])}(P, \mathit{Refl}_\tau(U[M])) : \mathit{Id}_\tau(U[M], U[N])\end{array}$$

We can derive a similar congruence property in the case that $\tau$ depends on $x$; for this and other derived properties and combinators for propositional equality we refer to (Nordström, Petersson, and Smith 1990; Streicher 1993; Hofmann 1995a).

We have now collected enough material to carry out the promised example of a proof by induction. We wish to construct an element of the type

$$m{:}\,\mathbf{N} \vdash \mathit{Id}_{\mathbf{N}}(m+0, m) \text{ type}$$

where $+$ is the addition operation defined above. Notice that we have $n{:}\,\mathbf{N} \vdash \mathit{Refl}_{\mathbf{N}}(n) : \mathit{Id}_{\mathbf{N}}(0+n, n)$ immediately by N-E-Z and the definition of $+$. Let us define $\sigma[m{:}\,\mathbf{N}] \stackrel{\text{def}}{=} \mathit{Id}_{\mathbf{N}}(m+0, m)$. So $\sigma$ is now a type with a distinguished free variable $m$. By N-C-Z we have

$$\vdash \mathit{Refl}_{\mathbf{N}}(0) : \sigma[0]$$

Now by N-E-S we have

$$m{:}\,\mathbf{N} \vdash \sigma[\mathit{Suc}(m)] = \mathit{Id}_{\mathbf{N}}(\mathit{Suc}(m), \mathit{Suc}(m+0)) \text{ type}$$

Therefore,

$$m{:}\,\mathbf{N}, h{:}\,\sigma[m] \vdash \mathit{Resp}_{\mathbf{N},\mathbf{N}}([x{:}\,\mathbf{N}]\mathit{Suc}(x)\ ,\ h) : \sigma[\mathit{Suc}(m)]$$

and we can finally conclude

$$\begin{aligned} m{:}\,\mathbf{N} \vdash \mathsf{R}^{\mathbf{N}}_{[m:\mathbf{N}]\sigma}(\ & [t]\mathit{Refl}_{\mathbf{N}}(0), \\ & [m{:}\,\mathbf{N}, h{:}\,\sigma]\mathit{Resp}_{\mathbf{N},\mathbf{N}}([x{:}\,\mathbf{N}]\mathit{Suc}(x)\ ,\ h), \\ & m) : \sigma[m] \end{aligned}$$

Again, we refer to (Nordström, Petersson, and Smith 1990) for more examples of this kind. The metatheory and the strength of the present and other formulations of the identity type have been analysed in (Hofmann 1995a;(1996);Streicher 1993).

We remark that propositional equality does not affect the definitional one which even in the presence of identity types remains confined to intensional equality. Therefore, type theory together with identity types as defined here is called intensional type theory, see (Martin-Löf 1982). There exists another formulation of identity types in which one may conclude $\Gamma \vdash M = N : \sigma$ from $\Gamma \vdash P : \mathit{Id}_{\sigma}(M, N)$. This rule is called equality reflection and makes it possible to derive "definitional" equalities by induction and thus makes it extensional. Therefore, type theory with equality reflection is called extensional type theory. Since the proof $P$ is discarded upon application of this rule, definitional equality (then rather called judgemental equality) and thus typing become undecidable. See also E3.30.

### 2.1.6 Universes

A universe is a type containing codes for types. This is expressed by the following two rules

$$\frac{\vdash \Gamma\ \mathit{ctxt}}{\Gamma \vdash U \text{ type}}\ \text{U-F} \qquad \frac{\Gamma \vdash M : U}{\Gamma \vdash \mathit{El}(M) \text{ type}}\ \text{El-F}$$

So if $M : U$ is such a "code" then we can form the type associated to $M$, namely $\mathit{El}(M)$. So far the universe does not contain any closed codes. This may be achieved by stipulating that the universe be closed under certain type formers. For instance, closure under Π-types is expressed by

$$\frac{\Gamma \vdash S : U \qquad \Gamma, s{:}\, El(S) \vdash T : U}{\Gamma \vdash \hat{\Pi}(S, [s{:}\, El(S)]T) : U} \quad \text{U-}\Pi$$

$$\frac{\Gamma, s{:}\, El(S) \vdash M : El(T)}{\Gamma \vdash \hat{\lambda} s{:}\, El(S).M^{El(T)} : El(\hat{\Pi}(S, [s{:}\, El(S)]T))} \quad \text{U-}\Pi\text{-I}$$

$$\frac{\Gamma \vdash M : El(\hat{\Pi}(S, [s{:}\, El(S)]T)) \qquad \Gamma \vdash N : El(S)}{\Gamma \vdash \hat{App}_{[s:El(S)]El(T)}(M, N) : El(T[N/s])} \quad \text{U-}\Pi\text{-E}$$

$$\frac{\Gamma \vdash \hat{App}_{[s:El(S)]El(T)}(\hat{\lambda} s{:}\, El(S).M^{El(T)}, N) : El(T[N/s])}{\Gamma \vdash \hat{App}_{[s:El(S)]El(T)}(\hat{\lambda} s{:}\, El(S).M^{El(T)}, N) = M[N/s] : El(T[N/s])} \quad \text{U-}\Pi\text{-C}$$

A more economic syntax for universes closed under Π-types is obtained if we replace the last three above rules by a single new type equality

$$\frac{\Gamma \vdash \hat{\Pi}(S, [s{:}\, El(S)]T) : U}{\Gamma \vdash El(\hat{\Pi}(S, [s{:}\, El(S)]T)) = \Pi s{:}\, El(S).El(T) \text{ type}} \quad \text{U-}\Pi\text{-Ty}$$

which states that $El(\hat{\Pi}(S, [s{:}\, El(S)]T))$ is the product of the $El(T)$ rather than behaving like it. One does not need the new application and abstraction operators $\hat{\lambda}$ and $\hat{App}$ then. This syntax, which in fact is often used in the literature, has the disadvantage that it is no longer the case that equal types share the same outermost type former. This makes it more difficult to show that the type formers are injective; an auxiliary property required to establish the subject reduction property for an untyped rewrite system derived from definitional equality. Also, in many models rule U-Π-Ty is not valid under the canonical interpretation of Π, see (Streicher 1991) and the example following Def. 3.20.

Closure under natural numbers is described by

$$\frac{\vdash \Gamma \ ctxt}{\Gamma \vdash \hat{\mathbf{N}} : U} \quad \text{U-N}$$

and further rules introducing term formers $\hat{0}$, $\hat{Suc}$, and $\hat{\mathsf{R}}^{\mathbf{N}}$ witnessing that $El(\hat{\mathbf{N}})$ behaves like $\mathbf{N}$. Again we could instead impose the equality $\Gamma \vdash El(\hat{\mathbf{N}}) = \mathbf{N}$ type if the type theory already contains natural numbers.

In a similar way closure under other type formers including another universe can be stipulated. A final important closure property for universes is impredicative quantification:

$$\frac{\Gamma \vdash \sigma \text{ type} \qquad \Gamma, x{:}\, \sigma \vdash T : U}{\Gamma \vdash \forall x{:}\, \sigma.T : U} \quad \text{U-}\forall$$

$$\frac{\Gamma, x{:}\,\sigma \vdash M : \mathit{El}(T)}{\Gamma \vdash \hat{\lambda} x{:}\,\sigma.M^{\mathit{El}(T)} : \mathit{El}(\forall x{:}\,\sigma.T)} \quad \text{U-}\forall\text{-I}$$

$$\frac{\Gamma \vdash M : \mathit{El}(\forall x{:}\,\sigma.T) \qquad \Gamma \vdash N : \sigma}{\Gamma \vdash \hat{App}_{[x:\sigma]\mathit{El}(T)}(M, N) : \mathit{El}(T[N/x])} \quad \text{U-}\forall\text{-E}$$

$$\frac{\Gamma \vdash \hat{App}_{[x:\sigma]\mathit{El}(T)}\hat{\lambda} x{:}\,\sigma.M^{\mathit{El}(T)}, N) : \mathit{El}(T[N/x])}{\Gamma \vdash \hat{App}_{[x:\sigma]\mathit{El}(T)}(\hat{\lambda} x{:}\,\sigma.M^{\mathit{El}(T)}, N) = M[N/x] : \mathit{El}(T[N/x])} \quad \text{U-}\forall\text{-C}$$

The difference to closure under Π-types is that the "domain-type" $\sigma$ is arbitrary and not confined to a "small type" of the form $\mathit{El}(S)$. In particular $\sigma$ can be $U$ itself and we can form terms like

$$\mathit{polyone} = \forall c{:}\, U.\forall s{:}\, \mathit{El}(c).c$$

where $\mathit{El}(\mathit{polyone})$ has the closed inhabitant

$$\diamond \vdash \hat{\lambda} C{:}\, U.\hat{\lambda} x{:}\, \mathit{El}(C).x : \mathit{El}(\mathit{polyone})$$

—the polymorphic identity function known from polymorphic lambda calculus.

Universes are employed for modularisation and abstraction. For instance, they permit the definition of a type of a certain algebraic structure. In this way the type of semigroups with carrier $X : U$ can be defined as

$$\begin{aligned}\mathit{SEM}(X) &\overset{\text{def}}{=} \Sigma\circ{:}\, \mathit{El}(X) \times \mathit{El}(X) \to \mathit{El}(X).\\ &\qquad \Pi x{:}\, \mathit{El}(X).\Pi y{:}\, \mathit{El}(X).\Pi z{:}\, \mathit{El}(X).\mathit{Id}_{\mathit{El}(X)}(\circ(\circ(x,y),z)\ ,\circ(x,\circ(y,z)))\end{aligned}$$

An element of type $\mathit{SEM}(X)$ consists of a binary function on $\mathit{El}(X)$, and a proof that this function is associative. We can now write a function $F : \Pi X{:}\, U.\mathit{MON}(X) \to \mathit{SEM}(X)$ which "forgets" the neutral element. We can also form $\Sigma X{:}\, U.\mathit{SEM}(X)$; the type of semigroups. More complex examples of this kind may be found in (Luo 1991). An application of this pattern to semantics of modules in functional languages is (Harper and Mitchell 1993).

Under the propositions-as-types analogy we can view a universe also as a type of propositions. For instance, the type $\sigma \to U$ can be viewed as an analogue to the power-set of $\sigma$.

### 2.1.7 Miscellaneous types

A counterpart to absurdity in logic is the following empty type:

$$\frac{\vdash \Gamma\ \mathit{ctxt}}{\Gamma \vdash \mathbf{0}\ \text{type}} \quad \text{0-F} \qquad \frac{\Gamma \vdash \sigma\ \text{type} \qquad \Gamma \vdash M : \mathbf{0}}{\Gamma \vdash \mathsf{R}^{\mathbf{0}}_{\sigma}(M) : \sigma} \quad \text{0-E}$$

There are no canonical elements in $\mathbf{0}$ so there is no "computation rule" like N-C-Z. It is sometimes useful to have a type with a single canonical element corresponding to the true proposition:

$$\frac{\vdash \Gamma\ ctxt}{\Gamma \vdash \mathbf{1}\ \text{type}}\ \text{1-F} \qquad\qquad \frac{\vdash \Gamma\ ctxt}{\Gamma \vdash \star : \mathbf{1}}\ \text{1-I}$$

$$\frac{\Gamma, x{:}\,\mathbf{1} \vdash \sigma\ \text{type} \qquad \Gamma \vdash H : \sigma[\star/x] \qquad \Gamma \vdash M : \mathbf{1}}{\Gamma \vdash \mathsf{R}^{\mathbf{1}}_{[x:\mathbf{1}]\sigma}(H, M) : \sigma[M/x]}\ \text{1-E}$$

$$\frac{\Gamma \vdash \mathsf{R}^{\mathbf{1}}_{[x:\mathbf{1}]\sigma}(H, \star) : \sigma[\star/x]}{\Gamma \vdash \mathsf{R}^{\mathbf{1}}_{[x:\mathbf{1}]\sigma}(H, M) = H : \sigma[M/x]}\ \text{1-C}$$

There are a number of other types considered in the literature like co-product types corresponding to binary disjoint union, finite types with $n$ elements for each natural number $n$, types of well-founded trees, subset types, and quotient types to name the most important ones. In implementations like Lego and Coq new type formers can be defined "on the fly" by giving the rules for their canonical elements (like *Suc* and 0 in the case of $\mathbf{N}$). The elimination rules are then generated automatically. In Alf this is also possible, but elimination rules are replaced by the more general device of pattern-matching on the form of the constructors.

Finally, we can consider arbitrary theories of dependent types defined by type symbols, constants, and equations. This is described in (Pitts 1997).

## 2.2 Examples of type theories

In this section we briefly describe some "named" type theories and how they fit into the formal framework described here.

### 2.2.1 Martin-Löf's type theory

This is a collective name for type theories containing several of the above-described type formers, but not a universe closed under impredicative quantification. A characteristic feature of Martin-Löf's type theory is the presence of identity types either with or without equality reflection. Martin-Löf invented his type theories with the aim of extending the propositions as types correspondence to predicate logic and to provide a universal language for constructive mathematics (Martin-Löf 1984; Martin-Löf 1975). A standard reference on Martin-Löf's type theories is (Nordström, Petersson, and Smith 1990). An implementation of extensional Martin-Löf type theory is the Nuprl system (Constable et al. 1986).

### 2.2.2 The Logical Framework

Martin-Löf's Logical Framework (LF), see Part IV of (Nordström, Petersson, and Smith 1990), is a type theory with $\Pi$-types and a universe. Its intended use is to define theories, in particular Martin-Löf type theory, as extensions of the LF by constants and equations.

The idea is that types and type formers are declared as constants in the universe and that term-formers are declared as constants of the appropriate *El*-types. To distinguish from object level type formers some different notation is used: the $\Pi$-type of the framework is written $(x\colon\sigma)\tau$ instead of $\Pi x\colon\sigma.\tau$ and $(\sigma)\tau$ instead of $\sigma\to\tau$. Iterated $\Pi$-types are written $(x_1\colon\sigma_1,\dots,x_n\colon\sigma_n)\tau$ instead of $\Pi x_1\colon\sigma_1.\dots\Pi x_n\colon\sigma_n.\tau$. Abstraction is written as $[x\colon\sigma]M$ instead of $\lambda x\colon\sigma.M^\tau$ and application is written $M(N)$ instead of $App_{[x:\sigma]\tau}(M,N)$.[1] Iterated abstractions and applications are written $[x_1\colon\sigma_1,\dots,x_n\colon\sigma_n]M$ and $M(N_1,\dots,N_n)$, respectively. The lacking type information can be inferred. The universe is written *Set* instead of $U$. The *El*-operator is omitted.

For example the $\Pi$-type is described by the following constant and equality declarations (understood in every valid context):

$$\begin{array}{l}
\vdash \Pi : (\sigma\colon Set, \tau\colon(\sigma)Set)Set\\
\vdash App : (\sigma\colon Set, \tau\colon(\sigma)Set, m\colon\Pi(\sigma,\tau), n\colon\sigma)\tau(m)\\
\vdash \lambda : (\sigma\colon Set, \tau\colon(\sigma)Set, m\colon(x\colon\sigma)\tau(x))\Pi(\sigma,\tau)\\
\sigma\colon Set, \tau\colon(\sigma)Set, m\colon(x\colon\sigma)\tau(x), n\colon\sigma \vdash\\
\qquad\qquad App(\sigma,\tau,\lambda(\sigma,\tau,m),n) = m(n)
\end{array}$$

Notice, how terms with free variables are represented as framework abstractions (in the type of $\lambda$) and how substitution is represented as framework application (in the type of App and in the equation).

In this way the burden of dealing correctly with variables, substitution, and binding is shifted from the object language to the Logical Framework and so can be handled once and for all.

Of course, the LF can also cope with type formers other than the dependent function space. Since we refer to it later in §4, we consider here an ad hoc type former (creating a copy of its argument) defined by the two rules

$$\frac{\Gamma\vdash\sigma\ \text{type}}{\Gamma\vdash\mathbf{L}(\sigma)\ \text{type}}\qquad\frac{\Gamma\vdash M:\sigma}{\Gamma\vdash\mathbf{l}(M):\mathbf{L}(\sigma)}$$

In LF it would have to be rendered by two constants $\mathbf{L} : (Set)Set$ and $\mathbf{l} : (\sigma\colon Set,\sigma)\mathbf{L}(\sigma)$.

The Alf system (Magnusson and Nordström 1994) is based on the Logical Framework. It allows for the definition of types in *Set* simply by giving their

[1]In loc. cit. and in the Alf system the type annotations in functional abstractions are omitted. We include them for the sake of consistency.

constructors. Functions on the types are then defined by pattern-matching over the constructors as needed.

The Logical Framework can also be used to encode the syntax of other logical systems such as predicate logic and modal logic. The interested reader is referred to (Harper, Honsell, and Plotkin 1993).

### 2.2.3 The Calculus of Constructions

The Calculus of Constructions (CC) (Coquand and Huet 1988) is a type theory with Π-types and a universe closed under impredicative quantification (U-Impr). The universe is traditionally denoted by *Prop* and the corresponding *El*-operator is either written $\mathit{Prf}(-)$ or omitted. The idea is that the universe *Prop* corresponds to a type of propositions and that $\mathit{Prf}(-)$ associates the type of proofs to a proposition. Originally, it was intended that *Prop* not only contains propositions, but also datatypes like the natural numbers which are definable by their "impredicative encodings", for instance one has

$$\mathit{Nat} \stackrel{\text{def}}{=} \mathit{Prf}(\forall c{:}\,\mathit{Prop}.\forall z{:}\,\mathit{Prf}(c).\forall s{:}\,\mathit{Prf}(c) \to \mathit{Prf}(c).c)$$

and for this type constants 0 and $\mathit{Suc}(-)$ can be defined as well as an operator permitting definition of functions by primitive recursion. The point is that *Nat* itself is of the form $\mathit{Prf}(-)$ and so can serve as the argument $c$ to an element of *Nat*. Also other inductive datatypes like lists or trees can be defined in this way. Similarly, logical connectives can be defined on the type *Prop* by their usual higher-order encodings (Coquand and Huet 1988).

The encoding of datatypes inside *Prop* proved insufficient as no internal induction principles (like $\mathsf{R}^{\mathbf{N}}$) are available for these. This gave rise to two extensions of the pure Calculus of Constructions: Luo's Extended Calculus of Constructions (ECC) implemented in the Lego system (Luo 1994; Luo and Pollack 1992) and the Calculus of Inductive Definitions (CID) implemented in the Coq system (Coquand and Paulin-Mohring 1989; Dowek et al. 1991). ECC extends the Calculus of Constructions by a sequence of universes $U_0$, $U_1, \ldots$ where each $U_{i+1}$ contains a code for $U_i$ and *Prop* is contained in $U_0$. Datatypes reside in $U_0$ and are given by inductive rules like the ones for $\mathbf{N}$. The higher universes are used for modularisation as hinted at in the example above. *Prop* is used for propositions only.

In the CID we have two universes both closed under impredicative quantification, *Set* and *Prop*. The datatypes reside in *Set* and are given by inductive rules as in ECC. The implementation of the CID, Coq, comes with a program extraction facility which extracts executable ML programs from derivations in CID essentially by removing all terms and types coming from the universe *Prop*, see also (Paulin-Mohring 1989).

## 2.3 Pre-syntax

The syntax of types, terms, and contexts has been given together with the typing and equality rules. For certain purposes it is convenient to have a simpler inductive definition of possibly non well-typed terms, out of which the actual ones are singled out by the rules. For instance, we might want to consider $App_{[x:\mathbf{N}]\mathbf{N}}(0,0)$ as a term albeit not a well-typed one. These pre-terms, -types, and -contexts have been used to give semantics to type theory in terms of untyped computation (Allen 1987; Martin-Löf 1984); we will use them as an auxiliary device in the definition of the interpretation of type theory in semantic structures and also in the definition of context morphisms below.

The pre-contexts ($\Gamma$), pre-types ($\sigma$, $\tau$), and pre-terms ($M$, $N$) for a type theory with $\Pi-$, $\Sigma-$, identity types, and natural numbers are given by the following grammar.

$$\begin{array}{lcl}
\Gamma & ::= & \diamond \\
 & & \mid \Gamma, x{:}\,\sigma \qquad \text{provided } x \text{ is not declared in } \Gamma \\[2ex]
\sigma,\tau & ::= & \Pi x{:}\,\sigma.\tau \mid \Sigma x{:}\,\sigma.\tau \mid Id_\sigma(M,N) \mid \mathbf{N} \\[2ex]
M,N,H,P & ::= & x \mid \lambda x{:}\,\sigma.M^\tau \mid App_{[x:\sigma]\tau}(M,N) \mid \\
 & & Pair_{[x:\sigma]\tau}(M,N) \mid \mathsf{R}^{\Sigma}_{[z:(\Sigma x:\sigma.\tau)]\rho}([x{:}\,\sigma, y{:}\,\tau]H, M) \mid \\
 & & Refl_\sigma(M) \mid \mathsf{R}^{Id}_{[x:\sigma,y:\sigma,p:Id_\sigma(x,y)]|\tau}([z{:}\,\sigma]H\ , M, N, P) \mid \\
 & & 0 \mid Suc(M) \mid \mathsf{R}^{\mathbf{N}}_{[n:\mathbf{N}]\sigma}(H_z, [n{:}\,\mathbf{N}, x{:}\,\sigma]H_s, M)
\end{array}$$

Capture-free substitution and identification of terms with different bound variables can then be dealt with on the level of the pre-syntax. We will use the predicates well-formed or valid for those pre-terms/-types/-contexts which actually occur in derivable judgements. Note that the variables declared in a pre-context are pairwise distinct.

## Exercises

E2.1 Construct an inhabitant of the type

$$m{:}\,\mathbf{N}, n{:}\,\mathbf{N} \vdash Id_{\mathbf{N}}(m+n, n+m) \text{ type}$$

E2.2 Show that for arbitrary types $\Gamma \vdash \sigma$ type, $\Gamma \vdash \tau$ type, and $\Gamma, x{:}\,\sigma, y{:}\,\tau \vdash \rho$ type the following type corresponding to the axiom of choice is inhabited:

$$\Gamma \vdash (\Pi x{:}\,\sigma.\Sigma y{:}\,\tau.\rho) \Rightarrow (\Sigma f{:}\,\sigma \Rightarrow \tau.\Pi x{:}\,\sigma.\rho[(f\ x)/y]) \text{ type}$$

E2.3 By analogy to the type of natural numbers define the rules for a list type former which to any type $\sigma$ associates a type $\mathit{List}(\sigma)$ consisting of finite sequences of elements of $\sigma$. Hint: think of lists as inductively generated from the empty list by successive additions of elements of $\sigma$ ("cons"). Define a length function of type $\mathit{List}(\sigma) \to \mathbf{N}$ and define a type $\mathit{Vec}_\sigma(M)$ of lists of length $M$ for each $M : \mathbf{N}$ using lists, the identity type, and the $\Sigma$-type.

E2.4 Define a type of binary natural numbers with three constructors: *Zero*, $\mathit{Suc}_0$, and $\mathit{Suc}_1$ and define a conversion function from these binary representations to $\mathbf{N}$.

E2.5 Give the rules for a universe $U$ containing a code $\hat{\mathbf{0}}$ for the empty type $\mathbf{0}$ and a code $\hat{\mathbf{1}}$ for the unit type $\mathbf{1}$. Show that in a type theory which supports natural numbers, this universe, and the empty type itself the following type in the empty context is inhabited

$$\diamond \vdash \mathit{Id}_{\mathbf{N}}(0, \mathit{Suc}(0)) \to \mathbf{0} \text{ type}$$

corresponding to Peano's fourth axiom $0 \neq 1$. Hint: define using $\mathsf{R}^{\mathbf{N}}$ a function $f{:}\, \mathbf{N} \to U$ such that $\diamond \vdash f0 = \hat{\mathbf{1}} : U$ and $\diamond \vdash f(\mathit{Suc}(0)) = \hat{\mathbf{0}} : U$. Later on we will show by a semantic argument that the above type is not inhabited in the absence of a universe (Smith 1988).

E2.6* (Troelstra and van Dalen 1988) Show that in type theory without the empty type $\mathbf{0}$ such an empty type can be defined as $\mathit{Id}_{\mathbf{N}}(0, \mathit{Suc}(0))$. The elimination operator $\mathsf{R}^{\mathbf{0}}_\sigma$ must then be defined by induction on the structure of $\sigma$. Notice that in view of the semantic result anticipated in the previous exercise this definition hinges on the fact that there is no empty type in the first place.

E2.7* Show that for any type theory containing some or all of the type formers described above the rules Weak and Subst are admissible.

E2.8* (Weak $\Sigma$-types in the Calculus of Constructions) For $\vdash \sigma$ type and $x{:}\,\sigma \vdash T : \mathit{Prop}$ define

$$\exists x{:}\,\sigma.P \overset{\text{def}}{=} \forall c{:}\,\mathit{Prop}.(\forall x{:}\,\sigma.P \Rightarrow c) \Rightarrow c$$

where $X \Rightarrow Y$ abbreviates $\forall p{:}\,\mathit{Prf}(X).Y$. Define a pairing operation which to $M{:}\,\sigma$ and $N{:}\,\mathit{Prf}(P[M])$ associates an element $\exists\text{-I}(M,N) : \mathit{Prf}(\exists x{:}\,\sigma.P)$ and define in the case that $\sigma = \mathit{Prf}(S)$ for some $S{:}\,\mathit{Prop}$ a first projection $\mathrm{witness}_S : \mathit{Prf}(\exists x{:}\,\mathit{Prf}(S).T) \to \mathit{Prf}(S)$. Show that for $M$ and $N$ as above one has $\vdash \mathrm{witness}_S\, \exists\text{-I}(M,N) = M$.

E2.9 Give the rules for a universe closed under impredicative quantification using the "economic syntax" exemplified in rule U-Π-Ty

E2.10* Prove that whenever $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$ then $\Gamma \vdash \sigma = \tau$ type by induction on derivations. Find some type annotations in term formers which can safely be omitted without violating this property. Discuss the properties a type theory must have so that the type annotation in application can be omitted. In other words when can we replace $App_{[x:\sigma]\tau}(M,N)$ by $App(M,N)$ in the official syntax without violating uniqueness of types. See (Streicher 1991) for a thorough discussion of this point.

## 2.4 Context morphisms

Definition 2.11 Let $\Gamma$ and $\Delta \overset{\text{def}}{=} x_1{:}\,\sigma_1,\dots,x_n{:}\,\sigma_n$ be valid contexts. If $f \overset{\text{def}}{=} (M_1,\dots,M_n)$ is a sequence of $n$ pre-terms we write

$$\Gamma \vdash f \Rightarrow \Delta$$

and say that $f$ is a context morphism from $\Gamma$ to $\Delta$ if the following $n$ judgements hold:

$$\begin{array}{l}\Gamma \vdash M_1 : \sigma_1\\ \Gamma \vdash M_2 : \sigma_2[M_1/x_1]\\ \dots\\ \Gamma \vdash M_n : \sigma_n[M_1/x_1][M_2/x_2]\dots[M_{n-1}/x_{n-1}]\end{array}$$

Examples. For any context $\Gamma$ we have the empty context morphism () from $\Gamma$ to $\diamond$ and this is the only context morphism from $\Gamma$ to $\diamond$. If $\Gamma \equiv x_1{:}\,\sigma_1,\dots,x_n{:}\,\sigma_n$ is a context and $\Gamma \vdash \sigma$ type and $x$ is a fresh variable, then $(x_1,\dots x_n)$ forms a context morphism from $\Gamma, x{:}\,\sigma$ to $\Gamma$ which we denote by $\mathsf{p}(\Gamma,\sigma)$. A more concrete example is $(0, \mathit{Refl}_{\mathbf{N}}(0))$ which forms a context morphism from $\diamond$ to $n{:}\,\mathbf{N}, p{:}\,\mathit{Id}_{\mathbf{N}}(0,n)$ as $\diamond 0{:}\,\mathbf{N}$ and $\diamond \mathit{Refl}_{\mathbf{N}}(0) : (\mathit{Id}_{\mathbf{N}}(0,n))[0/n]$. The same sequence of terms also forms a context morphism from $\diamond$ to $n{:}\,\mathbf{N}, p{:}\,\mathit{Id}_{\mathbf{N}}(0,0)$ which shows that the "target context" $\Delta$ is not uniquely determined by $f$. For any term $\Gamma \vdash M : \sigma$ we can form a context morphism $\Gamma \vdash \overline{M} \Rightarrow \Gamma, x{:}\,\sigma$ where $\overline{M} \equiv (x_1,\dots,x_n,M)$ if $\Gamma \equiv x_1{:}\,\sigma_1,\dots,x_n{:}\,\sigma_n$. Finally, we have the identity context morphism $\Gamma \vdash \mathit{id}_\Gamma \Rightarrow \Gamma$ given by $\mathit{id}_\Gamma \equiv (x_1,\dots,x_n)$.

### 2.4.1 Generalised substitution

We denote syntactic identity up to renaming of variables by $\equiv$. If $\Gamma \vdash f \Rightarrow \Delta$ and $\tau$ is a pre-type we write $\tau[f/\Delta]$ for the simultaneous replacement of the $\Delta$-variables in $\tau$ by the corresponding terms in $f$, more precisely, if $\Delta \equiv x_1{:}\,\sigma_1,\dots,x_n{:}\,\sigma_n$ and $f \equiv (M_1,\dots,M_n)$ then

$$\tau[f/\Delta] \equiv \tau[M_1/x_1][M_2/x_2]\dots[M_n/x_n]$$

The attribute "simultaneous" means that the variables in $\Delta$ should be made disjoint from those in $\Gamma$ before performing the substitution. We define $-[f/\Delta]$ analogously for pre-terms, pre-contexts, and judgements $\mathcal{J}$ of the form $M : \sigma$, $\sigma$ type, $M = N : \sigma$, $\sigma = \tau$ type.

By induction on the length of $\Delta$ and using rules Weak and Subst we can then establish the following property:

Proposition 2.12 If $\Gamma \vdash f \Rightarrow \Delta$ and $\Delta, \Theta \vdash \mathcal{J}$ then $\Gamma, \Theta[f/\Delta] \vdash \mathcal{J}[f/\Delta]$.

One is only interested in the case where $\Theta \equiv \diamond$ as this subsumes the general case with $f$ replaced by the context morphism $\mathsf{q}(f, \Theta)$ from $\Gamma, \Theta[f/\Delta]$ to $\Delta, \Theta$ given by $\mathsf{q}(f, \Theta) \equiv (f, z_1, \ldots, z_k)$ for $\Theta \equiv z_1{:}\, \theta_1, \ldots, z_k{:}\, \theta_k$. However, in order to get the inductive argument through one needs the case of non-empty $\Theta$. When $\Theta \equiv \diamond$ and when no confusion can arise we write $\tau[f]$ for $\tau[f/\Delta]$ and similarly for terms, contexts, and judgements.

Notice the special case of the context morphism $\mathsf{p}(\Gamma, \sigma)$ defined above. If $\Gamma \vdash \mathcal{J}$ then $\Gamma, x{:}\, \sigma \vdash \mathcal{J}[\mathsf{p}(\Gamma, \sigma)]$, but $\mathcal{J} \equiv \mathcal{J}[\mathsf{p}(\Gamma, \sigma)]$ so the generalised substitution subsumes weakening. Similarly, for $\Gamma, x{:}\, \sigma \vdash \mathcal{J}$ and $\Gamma \vdash M : \sigma$ we have $\mathcal{J}[\overline{M}] \equiv \mathcal{J}[M/x]$ so ordinary substitution is subsumed, too.

The defined substitution operation allows us to establish the following derived typing rule for non-empty context morphisms:

$$\frac{\Gamma \vdash f \Rightarrow \Delta \qquad \vdash \Delta, x{:}\, \sigma \ \mathit{ctxt} \qquad \Gamma \vdash M : \sigma[f]}{\Gamma \vdash (f, M) \Rightarrow \Delta, x{:}\, \sigma} \quad \text{Mor-Cons}$$

This rule together with

$$\frac{\vdash \Gamma \ \mathit{ctxt}}{\Gamma \vdash () \Rightarrow \diamond} \quad \text{Mor-Empty}$$

generates all valid judgements of the form $\Gamma \vdash f \Rightarrow \Delta$.

If $\Gamma \vdash f \Rightarrow \Delta$ and $\Delta \vdash g \Rightarrow \Theta$ where $g \equiv (N_1, \ldots, N_k)$ we can form the list of terms $g \circ f \equiv (N_1[f], \ldots, N_k[f])$. In other words $g \circ f$ is obtained by simultaneously replaceming the $\Delta$-variables in $g$ by the corresponding terms in $f$. This list $g \circ f$ forms a context morphism from $\Gamma$ to $\Theta$ and $\circ$ as indicated is an associative operation. Although this can be seen directly with some intuition about substitutions we prefer to state it as a proposition together with some other properties which the reader is invited to prove by simultaneous induction on the length of $g$.

Proposition 2.13 Assume $\mathrm{B} \vdash e \Rightarrow \Gamma$, $\Gamma \vdash f \Rightarrow \Delta$, and $\Delta \vdash g \Rightarrow \Theta$. Furthermore let $\Theta \vdash \sigma$ type and $\Theta \vdash M : \sigma$. Then the following equations hold up to syntactic identity.

$$\begin{array}{l}\Gamma \vdash g \circ f \Rightarrow \Theta \\ \sigma[g \circ f] \equiv \sigma[g][f] \\ M[g \circ f] \equiv M[g][f] \\ (g \circ f) \circ e \equiv g \circ (f \circ e)\end{array}$$

Hint for the proof: define $g \circ f$ inductively by $() \circ f \overset{\text{def}}{=} ()$ and $(g, M) \circ f \overset{\text{def}}{=} (g \circ f, M[f])$ where $(g, M)$ denotes the list $g$ extended by $M$. Use the fact that $\tau[f] \equiv \tau$ if none of the $\Delta$-variables occurs in $\tau$ and that $\tau[N/x][f] \equiv \tau[f/\Delta][N[f]/x]$. □

### 2.4.2 Context morphisms and definitional equality

Let $f \equiv (M_1, \ldots, M_n)$ and $g \equiv (N_1, \ldots, N_n)$. If $\Gamma \vdash f \Rightarrow \Delta$ and $\Gamma \vdash g \Rightarrow \Delta$ then we write $\Gamma \vdash f = g \Rightarrow \Delta$ as an abbreviation for the $n$ judgements $\Gamma \vdash M_1 = N_1 : \sigma_1$, $\Gamma \vdash M_2 = N_2 : \sigma_2[M_1/x_1]$, $\ldots$ , $\Gamma \vdash M_n = N_n : \sigma_n[M_1/x_1] \ldots [M_{n-1}/x_{n-1}]$ if $\Delta \equiv x_1 \colon \sigma_1, \ldots, x_n \colon \sigma_n$. Notice that we could equivalently replace the second judgement by $\Gamma \vdash M_2 = N_2 : \sigma_2[N_1/x_1]$ in view of the first one and the congruence rules for definitional equality. If $\Gamma \vdash f = g \Rightarrow \Delta$ we say that $f$ and $g$ are definitionally equal context morphisms from $\Gamma$ to $\Delta$. By straightforward induction it is now possible to derive congruence rules for the defined operators on context morphisms, w.r.t. this definitional equality and we also have that if $\vdash \Gamma = \Gamma'$ *ctxt*, $\vdash \Delta = \Delta'$ *ctxt*, and $\Gamma \vdash f = g \Rightarrow \Delta$ then $\Gamma' \vdash f = g \Rightarrow \Delta'$.

## Exercises

E2.14 Show that the above-defined context morphisms $\mathsf{p}(\Gamma, \sigma)$, $\overline{M}$, and $\mathsf{q}(f, \Theta)$ have the following properties:

- If $\Gamma \vdash f \Rightarrow \Delta$ then $id_\Delta \circ f \equiv f \equiv f \circ id_\Gamma$.
- If $\Gamma \vdash M : \sigma$ then $\mathsf{p}(\Gamma, \sigma) \circ \overline{M} \equiv id_\Gamma$.
- If $\Gamma \vdash (f, M) \Rightarrow \Delta, x \colon \sigma$ then $\mathsf{p}(\Gamma, \sigma) \circ (f, M) \equiv f$ and $x[(f, M)] \equiv M$.
- If $\Gamma \vdash f \Rightarrow \Delta$ and $\vdash \Delta, x \colon \sigma$ *ctxt* then $\mathsf{p}(\Delta, \sigma) \circ \mathsf{q}(f, x \colon \sigma) \equiv f \circ \mathsf{p}(\Gamma, \sigma[f])$.
- If $\Gamma \vdash f \Rightarrow \Delta$ and $\Delta \vdash M : \sigma$ then $\overline{M} \circ f \equiv \mathsf{q}(f, x \colon \sigma) \circ \overline{M[f]}$.
- If $\Gamma \vdash \sigma$ type and $x$ fresh then $id_{\Gamma, x:\sigma} \equiv (\mathsf{p}(\Gamma, \sigma), x)$.

E2.15 Show that if $\Delta \vdash \Pi x \colon \sigma.\tau$ type and $\Gamma \vdash f \Rightarrow \Delta$ then $(\Pi x \colon \sigma.\tau)[f] \equiv \Pi x \colon \sigma[f].\tau[\mathsf{q}(f, x \colon \sigma)]$. Hint: you may assume that $x$ does not occur in $\Delta$ as types are identified up to renaming of bound variables.

## 3 Category-theoretic semantics of type theory

Now we develop an abstract notion of semantics for theories of dependent types of which most known interpretations of type theory form an instance.

The main purpose in defining such an abstract semantics is that it is easier to show that a mathematical structure forms an instance of the abstract framework rather than defining an interpretation function for it directly. This is achieved by essentially three properties of the abstract semantics:

- Substitution is a primitive operation rather than inductively defined.
- Variables are replaced by combinators for substitutions.
- Definitional equality is modelled by true (set-theoretic) equality.

Category-theoretic semantics is based on an abstraction from the combinators (like $\circ$, $\mathsf{p}$, $(-,-)$, $\mathsf{q}(-,-)$) and equations for context morphisms identified in the previous section. The key concept is the one of a category: a collection of objects (the contexts), and for any two objects a collection of morphisms (the context morphisms) together with an associative composition and identities. For lack of space we cannot give an introduction to categories here and need to presuppose some very basic notions, in particular categories, functors, natural transformations, isomorphisms, terminal objects, and the category of sets and functions. Reading the relevant parts of the first chapter of (Lambek and Scott 1985), for instance, should suffice to attain the required state of knowledge. If $\mathcal{C}$ is a category we write $|\mathcal{C}|$ or $\mathcal{C}$ for its collection of objects and $\mathcal{C}(A, B)$ for the collection of morphisms from $A$ to $B$. We also write $f : A \to B$ instead of $f \in \mathcal{C}(A, B)$. A final prerequisite: if $\phi$ is an informal proposition then we define the set $[\phi]$ by $[\phi] \equiv \{\star\}$ if $\phi$ is true and $[\phi] \equiv \emptyset$ if $\phi$ is false.

## 3.1 Categories with families

We choose the semantic framework of categories with families (CwFs) (Dybjer 1996) a variant of Cartmell's categories with attributes which have the advantage of being equationally defined, rather than using conditional equations. Furthermore, CwFs are closer to the syntax than categories with attributes and therefore—this is the hope of the author—should be easier to understand.

The definition of a CwF follows the structure of the judgements in type theory except that context morphisms and substitution are part of the structure rather than defined afterwards. Along with the explanation of CwFs we define two important instances: the term model $\mathcal{T}$ of the calculus of dependent types described in § 2. and the set-theoretic model $\mathcal{S}et$ as running examples.

If we include context morphisms the syntax contains four kinds of objects: contexts, context morphisms, types, and terms. Accordingly, for each of these we have a domain of interpretation in the model. More precisely, a CwF $\mathcal{C}$ contains

- a category $\mathcal{C}$ of semantic contexts and context morphisms
- for $\Gamma \in \mathcal{C}$ a collection $\mathit{Ty}_{\mathcal{C}}(\Gamma)$ of semantic types
- for $\Gamma \in \mathcal{C}$ and $\sigma \in \mathit{Ty}_{\mathcal{C}}(\Gamma)$ a collection $\mathit{Tm}_{\mathcal{C}}(\Gamma, \sigma)$ of semantic terms

Where appropriate we leave out the attribute "semantic" and write $\mathit{Tm}_{\mathcal{C}}(\sigma)$ instead of $\mathit{Tm}_{\mathcal{C}}(\Gamma, \sigma)$. We also omit the subscripts if they are clear from the context.

In the term model $\mathcal{T}$ the collection of contexts is the quotient by definitional equality of well-formed contexts, that is pre-contexts $\Gamma$ such that $\vdash \Gamma\ \mathit{ctxt}$. Two such contexts $\Gamma$ and $\Delta$ are identified if $\vdash \Gamma = \Delta\ \mathit{ctxt}$. We tend to denote equivalence classes by their representatives. A morphism from $\Gamma$ to $\Delta$ is an equivalence class with respect to definitional equality of syntactic context morphisms $\Gamma \vdash f \Rightarrow \Delta$. This is well-defined in view of the observations in §2.4.2. Composition and identities are given by the corresponding operations on syntactic context morphisms. $\mathit{Ty}_{\mathcal{T}}(\Gamma)$ is the set of pre-types $\sigma$ such that $\Gamma \vdash \sigma$ type again factored by definitional equality, that is $\sigma$ and $\tau$ are identified if $\Gamma \vdash \sigma = \tau$ type. Finally, $\mathit{Tm}(\Gamma, \sigma)$ is the set of pre-terms $M$ with $\Gamma \vdash M : \sigma$ factored by definitional equality.

The set-theoretic model $\mathcal{S}\mathit{et}$ has as category of contexts the category of sets and functions. An element of $\mathit{Ty}_{\mathcal{S}\mathit{et}}(\Gamma)$ is a family of sets $(\sigma_\gamma)_{\gamma \in \Gamma}$ indexed over $\Gamma$. An element of $\mathit{Tm}_{\mathcal{S}\mathit{et}}(\Gamma, \sigma)$ is an assignment of an element $M(\gamma)$ of $\sigma_\gamma$ for each $\gamma \in \Gamma$.

Next, we need constants and operations on these domains in order to interpret the rules of type theory. Moreover, substitution must be axiomatised in such a way that it corresponds to the defined syntactic substitution. The definition of CwFs only accounts for the structural rules common to all systems of dependent types as set out in § 2. The interpretation of the various type and term formers will be given afterwards as additional structure.

Semantic substitution is described by two operations for each context morphism, one for types and one for terms: if $f : \Gamma \to \Delta$ then there is a function $-\{f\} : \mathit{Ty}(\Delta) \to \mathit{Ty}(\Gamma)$ and for $\sigma \in \mathit{Ty}(\Delta)$ a function $-\{f\} : \mathit{Tm}(\Delta, \sigma) \to \mathit{Tm}(\Gamma, \sigma\{f\})$. These operations must be compatible with composition and identities in the following sense. If $\Gamma, \Delta, \Theta \in \mathcal{C}$, $f : \Gamma \to \Delta$, $g : \Delta \to \Theta$, $\sigma \in \mathit{Ty}(\Theta)$, and $M \in \mathit{Tm}(\Theta, \sigma)$ then the following equations are required to hold:

$$\begin{array}{rcccll}
\sigma\{id_\Theta\} & = & \sigma & \in & \mathit{Ty}(\Theta) & \text{(Ty-Id)} \\
\sigma\{g \circ f\} & = & \sigma\{g\}\{f\} & \in & \mathit{Ty}(\Gamma) & \text{(Ty-Comp)} \\
M\{id_\Theta\} & = & M & \in & \mathit{Tm}(\Theta, \sigma) & \text{(Tm-Id)} \\
M\{g \circ f\} & = & M\{g\}\{f\} & \in & \mathit{Tm}(\Gamma, \sigma\{g \circ f\}) & \text{(Tm-Comp)}
\end{array}$$

Notice that the former two equations are required for the two latter to "typecheck". Notice also, that substitution together with equations (Ty-Id) and

(Ty-Comp) makes $Ty$ a contravariant functor from $\mathcal{C}$ to $\mathcal{S}et$. The sets $Tm$ can also be organised into a functor, see §3.1.1 below.

In the term model substitution is the "generalised substitution" defined in §2.4.1. This means that we have $\sigma\{f\} \stackrel{\text{def}}{=} \sigma[f]$ and $M\{f\} \stackrel{\text{def}}{=} M[f]$. In the set-theoretic model substitution is given by pre-composition. If $f : \Delta \rightarrow \Gamma$ is a function and $(\sigma_\gamma)_{\gamma\in\Gamma}$ is a family of sets then $\sigma\{f\}$ is the family of sets given by $\sigma\{f\}_\delta \stackrel{\text{def}}{=} \sigma_{f(\delta)}$. Similarly, if $M \in Tm(\Gamma, \sigma)$ then $M\{f\}(\delta) \stackrel{\text{def}}{=} M(f(\delta))$. It is easy to see that the required equations hold.

Next we want to interpret the context formation rules. To model the empty context we require a terminal object $\top$ in the category $\mathcal{C}$. We usually write $\langle\rangle_\Gamma$ for the unique morphism from $\Gamma$ to $\top$. In the term model and in the set-theoretic model these terminal objects are the empty context and an arbitrary singleton set, respectively.

To interpret context extension we require for each $\Gamma \in \mathcal{C}$ and $\sigma \in Ty(\Gamma)$ a context $\Gamma.\sigma \in \mathcal{C}$ and a morphism $\mathsf{p}(\sigma) : \Gamma.\sigma \rightarrow \Gamma$. The context $\Gamma.\sigma$ is called the comprehension of $\sigma$ and $\mathsf{p}(\sigma)$ is called the projection associated to $\sigma$. In the term model $\Gamma.\sigma$ is the extended context $\Gamma, x{:}\sigma$ and $\mathsf{p}(\sigma)$ is the context morphism given by $\Gamma, x{:}\sigma \vdash \mathsf{p}(\Gamma, \sigma) \Rightarrow \Gamma$ as defined in §2.4. In the set-theoretic model $\Gamma.\sigma$ is the disjoint union of the $\sigma_\gamma$, i.e. the set $\{\, (\gamma, x) \mid \gamma \in \Gamma \wedge x \in \sigma_\gamma \,\}$. The function $\mathsf{p}(\sigma)$ then sends $(\gamma, x)$ to $\gamma$.

The morphism $\mathsf{p}(\sigma)$ can be seen as the first projection out of the generalised product $\Gamma.\sigma$. The second projection takes the form of an element $\mathsf{v}_\sigma \in Tm(\Gamma.\sigma, \sigma\{\mathsf{p}(\sigma)\})$ corresponding to the judgement $\Gamma, x{:}\sigma \vdash x : \sigma$.

In the term model this is the term $x$ in $\Gamma, x{:}\sigma \vdash x : \sigma$, whereas in the set-theoretic model we define it by the assignment $(\gamma, x) \mapsto x$. Note that, in this model, $\sigma\{\mathsf{p}(\sigma)\}_{(\gamma,x)} = \sigma_\gamma$.

According to the definition of syntactic context morphisms we need an operation which extends a semantic context morphism by a terms. If $f : \Gamma \rightarrow \Delta$, $\sigma \in Ty(\Delta)$, and $M \in Tm(\Gamma, \sigma\{f\})$ then there is a context morphism $\langle f, M\rangle_\sigma : \Gamma \rightarrow \Delta.\sigma$—the extension of $f$ by $M$—satisfying the following equations for $f : \Gamma \rightarrow \Delta$, $g : \mathrm{B} \rightarrow \Gamma$, $\sigma \in Ty(\Delta)$, $M \in Tm(\Gamma, \sigma\{f\})$.

$$\begin{array}{rclcll}
\mathsf{p}(\sigma) \circ \langle f, M\rangle_\sigma & = & f & : & \Gamma \rightarrow \Delta & \text{(Cons-L)} \\
\mathsf{v}_\sigma\{\langle f, M\rangle_\sigma\} & = & M & \in & Tm(\Gamma, \sigma\{f\}) & \text{(Cons-R)} \\
\langle f, M\rangle_\sigma \circ g & = & \langle f \circ g, M\{g\}\rangle_\sigma & : & \mathrm{B} \rightarrow \Delta.\sigma & \text{(Cons-Nat)} \\
\langle \mathsf{p}(\sigma), \mathsf{v}_\sigma\rangle_\sigma & = & id_{\Delta.\sigma} & : & \Delta.\sigma \rightarrow \Delta.\sigma & \text{(Cons-Id)}
\end{array}$$

In the term model the extension of $f$ by $M$ is $\Gamma \vdash (f, M) \Rightarrow \Delta, x{:}\sigma$, whereas in the set-theoretic model we have $\langle f, M\rangle_\sigma(\gamma{\in}\Gamma) \stackrel{\text{def}}{=} (f(\gamma), M(\gamma))$. We include the "type" information $\sigma$ in $\langle f, M\rangle_\sigma$ as it cannot be inferred from the "types" of $f$ and $M$.

This completes the definition of categories with families. Let us summarise that a CwF is a structure $(\mathcal{C}, Ty, Tm, -\{-\}, \top, \langle\rangle_-, -.-, \mathsf{p}, \mathsf{v}_-, \langle -, -\rangle_-)$ of sorts and operations subject to the requirements set out above. (The substitution $-\{-\}$ is understood to work for both types and terms.)

### 3.1.1 A more abstract definition

We give in this section an equivalent, but more abstract and more compact definition of CwF based on family-valued functors and a universal property. The idea of using family-valued functors is due to Peter Dybjer.

Definition 3.1 The category $\mathcal{F}am$ of families of sets has as objects pairs $B = (B^0, B^1)$ where $B^0$ is a set and $(B^1_b)_{b \in B^0}$ is a family of sets indexed over $B^0$. A morphism from $B$ to $C = (C^0, C^1)$ is a pair $(f^0, f^1)$ where $f^0 : B^0 \to C^0$ is a function and $f^1 = (f^1_b)_{b \in B^0}$ is a family of functions $f^1_b : B^1(b) \to C^1(f^0(b))$.

The carrier sets $Ty$ and $Tm$ of a CwF over category $\mathcal{C}$ can now be given more compactly as a single functor $\mathcal{F} : \mathcal{C}^{op} \to \mathcal{F}am$. Indeed, given $Ty$ and $Tm$ we obtain a functor functor $\mathcal{F}$ with object part

$$\mathcal{F}(\Gamma) = (Ty(\Gamma), (Tm(\Gamma, \sigma))_{\sigma \in Ty(\Gamma)})$$

The morphism part of $\mathcal{F}$ is induced by semantic substitution. Conversely, given a functor $\mathcal{F} : \mathcal{C}^{op} \to \mathcal{F}am$ we define $Ty(\Gamma) := F^0$ and $Tm(\Gamma, \sigma) = F^1(\sigma)$ where $\mathcal{F}(\Gamma) = (F^0, F^1)$. If $f : \Delta \to \Gamma$ then writing $\mathcal{F}(f) = (f^0, f^1)$ we have $f^0 : Ty(\Gamma) \to Ty(\Delta)$ and $f^1_\sigma : Tm(\Gamma, \sigma) \to Tm(\Delta, f^0(\sigma))$ giving us semantic substitution. The required equations follow from functoriality of $\mathcal{F}$.

Definition 3.2 Let $\mathcal{C}$ be a category and $\mathcal{F} = (Ty, Tm) : \mathcal{C}^{op} \to \mathcal{F}am$ be a functor. Furthermore, let $\Gamma$ be an object of $\mathcal{C}$ and $\sigma \in Ty(\Gamma)$. A comprehension of $\sigma$ is given by an object $\Gamma.\sigma$ of $\mathcal{C}$ together with two projections $\mathsf{p}(\sigma) : \Gamma.\sigma \to \Gamma$ and $\mathsf{v}_\sigma \in Tm(\Gamma.\sigma, \sigma\{\mathsf{p}(\sigma)\})$ such that for each $f : \Delta \to \Gamma$ and $M \in Tm(\sigma\{f\})$ there exists a unique morphism $\langle f, M \rangle_\sigma : \Delta \to \Gamma.\sigma$ satisfying $\mathsf{p}(\sigma) \circ \langle f, M \rangle_\sigma = f$ and $\mathsf{v}_\sigma\{\langle f, M \rangle_\sigma\} = M$.

Definition 3.3 (Dybjer) A category with families is given by the following data.

- a category $\mathcal{C}$ with terminal object,
- a functor $\mathcal{F} = (Ty, Tm) : \mathcal{C}^{op} \to \mathcal{F}am$,
- a comprehension for each $\Gamma \in \mathcal{C}$ and $\sigma \in Ty(\Gamma)$.

This definition is equivalent to the one in §3.1. The proof is left to the reader.

The fact that comprehensions enjoy a universal property and thus are unique up to isomorphism (see E3.12) means that up to a choice of representatives a CwF is fully determined by its underlying family-valued functor.

### 3.1.2 Terms and sections

Assume a CwF. If $M \in Tm(\Gamma, \sigma)$ then also $M \in Tm(\Gamma, \sigma\{id_\Gamma\})$ and thus

$$\overline{M} \overset{\text{def}}{=} \langle id_\Gamma, M\rangle_\sigma : \Gamma \to \Gamma.\sigma$$

By (Cons-L) we have $\mathsf{p}(\sigma) \circ \overline{M} = id_\Gamma$ thus $\overline{M}$ is a right inverse or a so-called section of $\mathsf{p}(\sigma)$. Conversely, if $f : \Gamma \to \Gamma.\sigma$ is a section of $\mathsf{p}(\sigma)$, that is, $\mathsf{p}(\sigma) \circ f = id_\Gamma$ then

$$\mathsf{v}_\sigma\{f\} \in Tm(\Gamma, \sigma\{\mathsf{p}(\sigma)\}\{f\}) = Tm(\Gamma, \sigma\{\mathsf{p}(\sigma) \circ f\}) = Tm(\Gamma, \sigma)$$

by (Ty-Comp), (Ty-Id), and assumption. These two operations establish a bijective correspondence between the collection $Sect(\mathsf{p}(\sigma))$ of sections of $\mathsf{p}(\sigma)$ and $Tm(\Gamma, \sigma)$, as $\mathsf{v}_\sigma\{\bar{M}\} = M$ by (Cons-L) and $\overline{\mathsf{v}_\sigma\{f\}} = \langle id_\Gamma, \mathsf{v}_\sigma\{f\}\rangle_\sigma = \langle \mathsf{p}(\sigma) \circ f, \mathsf{v}_\sigma\{f\}\rangle_\sigma = \langle \mathsf{p}(\sigma), \mathsf{v}_\sigma\rangle_\sigma \circ f = f$ by (Cons-Nat) and (Cons-Id).

### 3.1.3 Weakening

Suppose that $f : \mathrm{B} \to \Gamma$ and $\sigma \in Ty(\Gamma)$. A context morphism $\mathsf{q}(f, \sigma) : \mathrm{B}.\sigma\{f\} \to \Gamma.\sigma$ called the weakening of $f$ by $\sigma$ is defined by

$$\mathsf{q}(f, \sigma) = \langle f \circ \mathsf{p}(\sigma\{f\}), \mathsf{v}_{\sigma\{f\}}\rangle_\sigma$$

In the term model $\mathsf{q}(f, \sigma)$ is the eponymous syntactic context morphism defined in §2.4; in the set-theoretic model we have $\mathsf{q}(f, \sigma)(\beta \in \mathrm{B}, x \in \sigma_{f(\beta)}) = (f(\beta), x)$.

A weakening map is a morphism of the form $\mathsf{p}(\sigma) : \Gamma.\sigma \to \Gamma$ or (inductively) a morphism of the form $\mathsf{q}(w, \tau)$ where $w$ is a weakening map. In the term model a weakening map takes the form of a projection from $\Gamma, x{:}\,\sigma, \Delta$ to $\Gamma, \Delta$.

We introduce the abbreviations $\sigma^+$ and $M^+$ for $\sigma\{w\}$ and $M\{w\}$ if $w$ is a weakening map which is clear from the context. Furthermore, if $f : \Gamma \to \Delta$ is any context morphism we may write $f^+$ for $\mathsf{q}(f, \sigma)$. For example, as demonstrated in E2.15 we have in the term model $\Pi x{:}\,\sigma.\tau\{f\} = \Pi x{:}\,\sigma\{f\}.\tau\{f^+\}$.

## Exercises

E3.4 Show that in a CwF the defined morphisms $\mathsf{q}(f, \sigma)$ from §3.1.3 do satisfy the coherence requirements $\mathsf{q}(id_\Gamma, \sigma) = id_{\Gamma.\sigma}$ and $\mathsf{q}(f \circ g, \sigma) = \mathsf{q}(f, \sigma) \circ \mathsf{q}(g, \sigma\{f\})$.

E3.5 Transform the following equations into the explicit notation; prove them, and explain their intuitive meaning. (1) $\mathsf{p}(\sigma)^+ \circ \overline{\mathsf{v}_\sigma} = \mathsf{p}(\sigma^+) \circ \overline{\mathsf{v}_\sigma} = id_{\Gamma.\sigma}$; (2) $f^+ \circ \overline{M\{f\}} = \overline{M} \circ f$; (3) $f = \langle \mathsf{p}(\sigma) \circ f, \mathsf{v}_\sigma\{f\}\rangle_\sigma$. Also expand the expression $\Gamma.\sigma.\tau$ when $\sigma, \tau \in Ty\Gamma$.

E3.6 Check that equations (Cons-L) ... (Cons-Id) hold in the set-theoretic model.

E3.7 This exercise will be taken up in later sections and will lead us up to Jan Smith's proof of the independence of Peano's fourth axiom from Martin-Löf's type theory without universes (Smith 1988). Let $\mathcal{P}$ be the poset of truth values $\{\mathbb{ff}, \mathbb{tt}\}$ where $\mathbb{ff} \leq \mathbb{tt}$ viewed as a category. Show that $\mathcal{P}$ has a terminal object, viz. $\mathbb{tt}$. Extend $\mathcal{P}$ to a CwF by putting $\mathit{Ty}_{\mathcal{P}}(\mathbb{tt}) = \mathit{Ty}_{\mathcal{P}}(\mathbb{ff}) = \{\mathbb{ff}, \mathbb{tt}\}$ and $\mathit{Tm}_{\mathcal{P}}(\Gamma, \sigma) = [\Gamma \leq \sigma]$. Hint: define comprehension by $\Gamma.\sigma \stackrel{\text{def}}{=} \Gamma \wedge \sigma$. An intuition for this model (or rather for the interpretation of the syntax in it) is to view $\mathbb{tt}$ as "potentially inhabited" and $\mathbb{ff}$ as "always empty".

## 3.2 Other notions of semantics

In the literature other notions of model have been offered which mostly are equivalent to CwFs. A key property of these models is that substitution on terms is a defined concept rather than a primitive. To understand how this works we need the notion of pullback in a category.

Definition 3.8 Let $\mathcal{C}$ be a category, $f : X \to Y$ and $g : Z \to Y$ morphisms with common codomain. A pullback of $g$ along $f$ is a pair of morphisms $p : P \to Y$ and $q : P \to Z$ such that $f \circ p = g \circ q$ and whenever $p' : P' \to X$ and $q' : P' \to Z$ are two morphisms with $f \circ p' = g \circ q'$ then there exists a unique morphism $h : P' \to P$ such that $p \circ h = p'$ and $q \circ h = q'$. [2]

$$\begin{tikzcd}
P' \arrow[dr, dotted, "h"] \arrow[drr, "q'"] \arrow[ddr, "p'"'] & & \\
& P \arrow[r, "q"] \arrow[d, "p"] & Z \arrow[d, "g"] \\
& X \arrow[r, "f"'] & Y
\end{tikzcd}$$

A quadruple $(p, q, f, g)$ of morphisms with $f \circ p = g \circ q$ (a commuting square) is called a pullback if $p$ and $q$ form a pullback of $g$ along $f$.

In the category $\mathcal{S}et$ the pullback of $g : Z \to Y$ along $g : X \to Y$ always exists and is (e.g.) given by $P \stackrel{\text{def}}{=} \{(x, z) \mid x \in X \wedge z \in Z \wedge f(x) = g(z)\}$. The two projections $p$, $q$ send a pair $(x, z) \in P$ to $x$ and $z$, respectively.

[2]This and the following diagrams were typeset using Paul Taylor's Latex diagram package the use of which is herewith gratefully acknowledged.

In $\mathcal{S}et$ we can view a morphism $g : Z \to Y$ as a family of sets indexed over $Y$, namely the family of sets $(g^{-1}(y))_{y\in Y}$ where $g^{-1}(y) \overset{\text{def}}{=} \{z \in Z \mid g(z) = y\}$. Conversely, if we are given a family of sets $(G_y)_{y\in Y}$ we can construct a function $g$ into $Y$ as the projection from the disjoint union

$$Z \overset{\text{def}}{=} \{(y,\gamma) \mid y \in Y \wedge \gamma \in G_y\}$$

to $Y$ which sends $(y, \gamma)$ to $y$. (This is precisely $\mathsf{p}((G_y)_{y\in Y})$ in the notation of §3.1.) Now a pullback of the thus defined $g : Z \to Y$ along $f : X \to Y$ is given by applying this construction to the composite family $(G_{f(x)})_{x\in X}$, more precisely, we can put

$$P \overset{\text{def}}{=} \{(x,\gamma) \mid x \in X \wedge \gamma \in G_{f(x)}\}$$

with projections $p(x, \gamma) = x$ and $q(x, \gamma) = (f(x), \gamma)$. Note that this is not equal to the canonical pullback of $g$ along $f$ which is $\{(x,(x',\gamma)) \mid x = x' \text{ and } \gamma \in G_{f(x')}\}$.

A more general correspondence between pullbacks and substitution arises in the framework of CwFs:

Proposition 3.9 Let $\mathcal{C}$ be a CwF, $f : \mathrm{B} \to \Gamma$, and $\sigma \in \mathit{Ty}(\Gamma)$. The following square is a pullback

$$\begin{tikzcd}
\mathrm{B}.\sigma\{f\} \arrow[r, "\mathsf{q}(f,\sigma)"] \arrow[d, "\mathsf{p}(\sigma\{f\})"'] & \Gamma.\sigma \arrow[d, "\mathsf{p}(\sigma)"] \\
\mathrm{B} \arrow[r, "f"'] & \Gamma
\end{tikzcd}$$

Proof. The diagram commutes by equation (Cons-L). To see that it is indeed a pullback assume $p' : \Theta \to \mathrm{B}$ and $q' : \Theta \to \Gamma.\sigma$ such that $f \circ p' = \mathsf{p}(\sigma) \circ q'$. We can decompose $q'$ as $q' = \langle \mathsf{p}(\sigma), \mathsf{v}_\sigma\rangle_\sigma \circ q' = \langle \mathsf{p}(\sigma) \circ q', \mathsf{v}_\sigma\{q'\}\rangle_\sigma = \langle f \circ p', N\rangle_\sigma$, where $N \overset{\text{def}}{=} \mathsf{v}_\sigma\{q'\}$. We have $N \in \mathit{Tm}(\sigma\{\mathsf{p}(\sigma)\}\{q'\}) = \sigma\{\mathsf{p}(\sigma) \circ q'\} = \sigma\{f \circ p'\} = \sigma\{f\}\{p'\}$. Therefore,

$$h \overset{\text{def}}{=} \langle p', N\rangle_{\sigma\{f\}} : \Theta \to \mathrm{B}.\sigma\{f\}$$

We have $\mathsf{p}(\sigma\{f\}) \circ h = q'$ by (Cons-L) and

$$\begin{array}{lll}
 & \mathsf{q}(f,\sigma) \circ h & \\
= & \langle f \circ \mathsf{p}(\sigma\{f\}), \mathsf{v}_{\sigma\{f\}}\rangle_\sigma \circ h & \text{by definition} \\
= & \langle f \circ \mathsf{p}(\sigma\{f\}) \circ h, \mathsf{v}_{\sigma\{f\}}\{h\}\rangle_\sigma & \text{by (Cons-Nat)} \\
= & \langle f \circ p', \mathsf{v}_{\sigma\{f\}}\{h\}\rangle_\sigma & \text{by (Cons-L)} \\
= & \langle f \circ p', \mathsf{v}_\sigma\{q'\}\rangle_\sigma & \text{by (Cons-R)} \\
= & \langle \mathsf{p}(\sigma) \circ q', \mathsf{v}_\sigma\{q'\}\rangle_\sigma & \text{by assumption} \\
= & q' & \text{by (Cons-Nat) and (Cons-Id)}
\end{array}$$

For uniqueness assume a morphism $h' : \Theta \to \mathrm{B}.\sigma\{f\}$ such that $\mathsf{p}(\sigma\{f\}) \circ h' = p'$ and $\mathsf{q}(f,\sigma) \circ h' = q'$. We must show $h' = h$. To see this, we expand $h'$ as $\langle \mathsf{p}(\sigma\{f\}) \circ h', \mathsf{v}_{\sigma\{f\}}\{h'\}\rangle_{\sigma\{f\}}$ using (Cons-Id), (Id-L), (Cons-Nat). By assumption we can rewrite this to $\langle p', M\rangle_{\sigma\{f\}}$ where

$$M \overset{\text{def}}{=} \mathsf{v}_{\sigma\{f\}}\{h'\} \in \mathit{Tm}(\sigma\{f \circ \mathsf{p}(\sigma\{f\}) \circ h'\}) = \mathit{Tm}(\sigma\{f \circ p'\}) = \mathit{Tm}(\sigma\{\mathsf{p}(\sigma) \circ q'\})$$

Now we have

$$\begin{array}{lll} & \mathsf{v}_\sigma\{q'\} & \\ = & \mathsf{v}_\sigma\{\mathsf{q}(f,\sigma) \circ h'\} & \text{by assumption} \\ = & \mathsf{v}_{\sigma\{f\}}\{h'\} & \text{by definition of } \mathsf{q} \text{ and (Cons-R)} \\ = & M & \end{array}$$

so we are done. □

The pullback property of substitution may be taken as primitive thereby making substitution on terms superfluous.

Definition 3.10 A category with attributes (Cartmell 1978; Moggi 1991; Pitts 1997) consists of

- A category $\mathcal{C}$ with terminal object $\top$.
- A functor $\mathit{Ty} : \mathcal{C}^{op} \to \mathcal{S}et$, i.e., a set $\mathit{Ty}(\Gamma)$ for each $\Gamma \in \mathcal{C}$ and a function $-\{f\} : \mathit{Ty}(\Gamma) \to \mathit{Ty}(\mathrm{B})$ for each $f : \mathrm{B} \to \Gamma$ such that (Ty-Id) and (Ty-Comp) from §3.1 hold.
- For each $\sigma \in \mathit{Ty}(\Gamma)$ an object $\Gamma.\sigma$ and a morphism $\mathsf{p}(\sigma) : \Gamma.\sigma \to \Gamma$.
- For each $f : \mathrm{B} \to \Gamma$ and $\sigma \in \mathit{Ty}(\Gamma)$ a pullback diagram

$$\begin{tikzcd} \mathrm{B}.\sigma\{f\} \arrow[r, "\mathsf{q}(f,\sigma)"] \arrow[d, "\mathsf{p}(\sigma\{f\})"'] & \Gamma.\sigma \arrow[d, "\mathsf{p}(\sigma)"] \\ \mathrm{B} \arrow[r, "f"'] & \Gamma \end{tikzcd}$$

  such that $\mathsf{q}(id_\Gamma, \sigma) = id_{\Gamma.\sigma}$ and $\mathsf{q}(f \circ g, \sigma) = \mathsf{q}(f,\sigma) \circ \mathsf{q}(g, \sigma\{f\})$.

It follows from Prop. 3.9 and E3.4 that every CwF is a category with attributes if we forget about the terms.

Conversely, given a category with attributes we can construct a CwF by putting $\mathit{Tm}(\Gamma,\sigma) \overset{\text{def}}{=} \mathit{Sect}(\mathsf{p}(\sigma))$ and for $M \in \mathit{Tm}(\Gamma,\sigma)$ defining $M\{f\}$ as the unique morphism with $\mathsf{p}(\sigma\{f\}) \circ M\{f\} = id_\mathrm{B}$ and $\mathsf{q}(f,\sigma) \circ M\{f\} = M \circ f$.

For $f : \mathrm{B} \to \Gamma, M \in \mathit{Tm}(\mathrm{B}, \sigma\{f\})$ we put $\langle f, M\rangle_\sigma \overset{\mathrm{def}}{=} \mathsf{q}(f,\sigma) \circ M$ and finally define $\mathsf{v}_\sigma$ as the unique morphism $\mathsf{v}_\sigma : \Gamma.\sigma \to \Gamma.\sigma.\sigma^+$ with $\mathsf{p}(\sigma^+) \circ \mathsf{v}_\sigma = \mathsf{p}(\sigma)^+ \circ \mathsf{v}_\sigma = id_{\Gamma.\sigma}$. We leave it as an exercise to verify that this defines indeed a CwF.

If one starts out with a category with attributes, constructs a CwF, and from this CwF again a category with attributes one ends up with the one to start off with. The other way round one gets back the original CwF with the set of terms $\mathit{Tm}(\Gamma, \sigma)$ replaced by the set of right inverses to $\mathsf{p}(\sigma)$ (cf. §3.1.2).

Notice that context extensions in categories with attributes are not unique up to isomorphism. For instance, if $\sigma \in \mathit{Ty}(\Gamma)$ nothing prevents us from defining $\Gamma.\sigma$ simply as $\Gamma$ and the corresponding morphisms $\mathsf{p}$ and $\mathsf{q}$ as identities.

There are various other notions of model all of which are essentially equivalent as far as interpretation of type theory in them is concerned. Locally-cartesian closed categories (Seely 1984) and categories with display maps (Taylor 1986; Lamarche 1987; Hyland and Pitts 1989) are less general than CwFs because semantic types are identified with their associated projections. Usually, in these models the conditions corresponding to (Ty-Comp) and (Ty-Id) only hold up to isomorphism, which makes the definition of the interpretation function more complicated. See (Hofmann 1995b; Curien 1993). Models based on fibrations (Jacobs 1991; Jacobs 1993; Ehrhard 1988) and indexed categories (Curien 1989; Obtułowicz 1989) are more general since they allow for morphisms between semantic types. These morphisms make it possible to describe certain type formers more elegantly, but do not have a direct counterpart in the syntax. Mention must also be made of contextual categories where semantic contexts carry an explicit tree structure corresponding to context extensions (see (Streicher 1991; Cartmell 1978) and E3.13).

For a good taxonomy of the different notions and various back-and-forth constructions see (Jacobs 1993).

## Exercises

E3.11 (Agnes Diller) Show that the requirement $\mathsf{q}(id_\Gamma, \sigma) = id_{\Gamma.\sigma}$ in Def. 3.10 is actually redundant.

E3.12* Prove that comprehensions are indeed unique up to isomorphism in the following sense: If $\sigma \in \mathit{Ty}(\Gamma)$ and $p : \Sigma \to \Gamma$ and $v \in \mathit{Tm}(\sigma\{f\})$ is a comprehension of $\sigma$ then there exists an isomorphism $f : \Gamma.\sigma \to \Sigma$ satisfying $p \circ f = \mathsf{p}(\sigma)$ and $v\{\sigma\} = \mathsf{v}_\sigma$.

Now give another proof of Prop. 3.9 by showing that $(\Theta, p', \mathsf{v}_\sigma\{q'\})$ is a comprehension of $\sigma\{f\}$.

E3.13* A contextual category (Cartmell 1978; Streicher 1991) is a category $\mathcal{C}$ with terminal object $\top$ and a tree structure on the objects given by a function $f$ on the objects of $\mathcal{C}$ such that $f(\top) = \top$ and $f$ ("father") is injective on $\mathcal{C} \setminus \{\top\}$ and for each $\Gamma \in \mathcal{C}$ there is a minimal $n$—the level of $\Gamma$—such that $f^n(\Gamma) = \top$. Moreover, the assignment $Ty(\Gamma) = \{\Delta \mid f(\Delta) = \Gamma\}$ extends to a category with attributes over $\mathcal{C}$. Spell this out without using the notion of a category with attributes and define a canonical construction of a contextual category out of a CwF. The objects of the contextual category are lists $(\sigma_1, \cdots, \sigma_k)$ where $\sigma_1 \in Ty(\top)$ and $\sigma_{i+1} \in Ty(\top.\sigma_1.\cdots.\sigma_i)$.

The advantage of contextual categories is that they rule out semantic contexts which do not arise from the empty context by successive applications of comprehension. This can be used to establish properties of semantic contexts by induction on the level. Examples of this can be found in (Streicher 1991).

E3.14* Assume a CwF $\mathcal{C}$. For each $\Gamma \in \mathcal{C}$ we define a category $\mathcal{D}(\Gamma)$ with objects the types over $\Gamma$ and in which a morphism from $\sigma$ to $\tau$ is a $\mathcal{C}$-morphism $f : \Gamma.\sigma \to \Gamma.\tau$ such that $\mathsf{p}(\tau) \circ f = \mathsf{p}(\sigma)$. Equivalently a morphism from $\sigma$ to $\tau$ can be defined as an element of $Tm(\Gamma.\sigma, \tau^+)$ (Why?). Explain why this defines indeed a category and find an extension of the assignment $\mathcal{D}$ to a contravariant functor from $\mathcal{C}$ to the category of categories.

Such a functor is called an indexed category and forms the heart of Curien and Ehrhard's notion of D-categories (Curien 1989; Ritter 1992).

## 3.3 Semantic type formers

In order to interpret a type theory in a CwF we must specify how the various type and term formers are to be interpreted. This results in certain requirements on a CwF which follow very closely the syntactic rules. We give a precise definition for closure under $\Pi, \Sigma, Id, \forall$ to convey the general pattern. The reader may then by herself define the conditions for the other type formers. Some are treated in the exercises.

Definition 3.15 A CwF supports $\Pi$-types if for any two types $\sigma \in Ty(\Gamma)$ and $\tau \in Ty(\Gamma.\sigma)$ there is a type $\Pi(\sigma,\tau) \in Ty(\Gamma)$ and for each $M \in Tm(\Gamma.\sigma, \tau)$ there is a term $\lambda_{\sigma,\tau}(M) \in Tm(\Gamma, \Pi(\sigma,\tau))$ and for each $M \in Tm(\Gamma, \Pi(\sigma,\tau))$ and $N \in Tm(\Gamma, \sigma)$ there is a term $App_{\sigma,\tau}(M,N) \in Tm(\Gamma, \tau\{\overline{M}\})$ such that (the appropriately typed universal closures of) the following equations hold:

$$\begin{array}{lcll}
App_{\sigma,\tau}(\lambda_{\sigma,\tau}(M), N) & = & M\{\overline{N}\} & \Pi\text{-C} \\
\Pi(\sigma,\tau)\{f\} & = & \Pi(\sigma\{f\}, \tau\{\mathsf{q}(f,\sigma)\}) \in Ty(\mathrm{B}) & \Pi\text{-S} \\
\lambda_{\sigma,\tau}(M)\{f\} & = & \lambda_{\sigma\{f\},\tau\{\mathsf{q}(f,\sigma)\}}(M\{\mathsf{q}(f,\sigma)\}) & \lambda\text{-S} \\
App_{\sigma,\tau}(M,N)\{f\} & = & App_{\sigma\{f\},\tau\{\mathsf{q}(f,\sigma)\}}(M\{f\}, N\{f\}) & App\text{-S}
\end{array}$$

The third equation typechecks by virtue of the second. The last equation makes implicit use of (Ty-Comp) and the fact that

$$\mathsf{q}(f,\sigma)\circ\overline{M\{f\}} = \overline{M}\circ f$$

which follows by rewriting both sides to $\langle f, M\{f\}\rangle_\sigma$.

We see that we stipulate exactly the same type and term formers as in the syntax together with an equation corresponding to Π-C. Congruence rules are not needed on the semantic level as everything preserves equality, but we need extra equations to specify that substitution commutes with the semantic type and term formers.

Since substitution is a primitive notion in the semantics we can obtain a more economic definition of dependent function spaces by restricting the *App*-combinator to variables and replacing it by an appropriate morphism.

Proposition 3.16 A CwF supports dependent function spaces iff there are operations $\Pi$ and $\lambda$ as in Def. 3.15 and for any two types $\sigma \in \mathit{Ty}(\Gamma)$ and $\tau \in \mathit{Ty}(\Gamma.\sigma)$ a morphism

$$\mathit{App}_{\sigma,\tau} : \Gamma.\sigma.\Pi(\sigma,\tau)^+ \to \Gamma.\sigma.\tau$$

such that

$$\mathsf{p}(\tau)\circ \mathit{App}_{\sigma,\tau} = \mathsf{p}(\Pi(\sigma,\tau)^+) \qquad\qquad \mathit{App}\text{-T}$$

and for every term $M \in \mathit{Tm}(\tau)$

$$\mathit{App}_{\sigma,\tau}\circ\overline{M\{\mathsf{p}(\sigma)\}} = \overline{M} \qquad\qquad \Pi\text{-C'}$$

and, finally, for every morphism $f : \mathrm{B}\to\Gamma$

$$\mathit{App}_{\sigma,\tau}\circ\mathsf{q}(\mathsf{q}(f,\sigma),\Pi(\sigma,\tau)\{\mathsf{q}(f,\sigma)\}) = \mathsf{q}(\mathsf{q}(f,\sigma),\tau)\circ \mathit{App}_{\sigma\{f\},\tau\{\mathsf{q}(f,\sigma)\}}$$

Proof. The first equation states that $\mathit{App}_{\sigma,\tau}$ leaves its first two arguments unchanged, and thus corresponds to a term of $\tau^+$. The second equation corresponds to Π-C, and the third one to *App*-S; stability under substitution. From the *App*-morphism we can define an application combinator as follows. Given $M \in \mathit{Tm}(\Pi(\sigma,\tau))$ and $N \in \mathit{Tm}(\sigma)$ then $\mathsf{v}_\tau\{\mathit{App}_{\sigma,\tau}\circ\langle N, M^+\rangle_{\Pi(\sigma,\tau)^+}\} \in \mathit{Tm}(\tau\{\mathsf{p}(\tau)\circ \mathit{App}_{\sigma,\tau}\circ\langle N, M^+\rangle_{\Pi(\sigma,\tau)^+}\} = \mathit{Tm}(\tau\{\overline{N}\})$ by Eqn. *App*-T and it follows from the other two equations that this term has the required properties. Conversely, suppose that a model supports dependent function spaces in the sense of Def. 3.15. Let $\Delta$ be $\Gamma.\sigma.\Pi(\sigma,\tau)^+$. We have $N \overset{\text{def}}{=} \mathsf{v}_\sigma^+ \in \mathit{Tm}(\Delta,\sigma^{++})$ and $M \overset{\text{def}}{=} \mathsf{v}_{\Pi(\sigma,\tau)^+} \in \mathit{Tm}(\Delta,\Pi(\sigma,\tau)^{++}) = \mathit{Tm}(\Delta,\Pi(\sigma^{++},\tau^{++}))$. Accordingly, we have

$$\mathit{App}_{\sigma,\tau}(M,N) \in \mathit{Tm}(\Delta,\tau^{++}\{N\}) = \mathit{Tm}(\Delta,\tau^+)$$

We define the application morphism as

$$\langle \mathsf{p}(\Pi(\sigma,\tau)^+, \mathit{App}_{\sigma,\tau}(M,N)\rangle_\tau$$

The verifications are left to the reader. □

This second definition of dependent function spaces allows for the following restriction by which $\lambda_{\sigma,\tau}(M)$ is required to be the unique element of $\mathit{Tm}(\Pi(\sigma,\tau))$ for which Π-C' holds.

Definition 3.17 A CwF supports Π-types in the strict sense if it supports them and whenever $M \in \mathit{Tm}(\Gamma.\sigma,\tau)$ and $U \in \mathit{Tm}(\Pi(\sigma,\tau))$ and $\mathit{App}_{\sigma,\tau} \circ \overline{U^+} = \overline{M}$ then $U = \lambda_{\sigma,\tau}(M)$.

The syntactic counterpart to these strict Π-types is an $\eta$-rule which allows one to conclude $\Gamma \vdash \lambda x{:}\,\sigma.(M\,x)^\tau = M : \Pi x{:}\,\sigma.\tau$ from $\Gamma \vdash M : \Pi x{:}\,\sigma.\tau$.

If Π-types are supported in the strict sense then Eqn. $\lambda$-S can be derived. Moreover, the operation $\Pi(\sigma,-)$ then becomes part of an adjunction, see, e.g. (Pitts 1997).

The term model of a type theory with Π-types supports Π-types with the obvious settings:

$$\Pi(\sigma,\tau) \overset{\text{def}}{=} \Pi x{:}\,\sigma.\tau$$

$$\lambda_{\sigma,\tau}(M) \overset{\text{def}}{=} \lambda x{:}\,\sigma.M^\tau$$

$$\mathit{App}_{\sigma,\tau}(M,N) \overset{\text{def}}{=} \mathit{App}_{[x:\sigma]\tau}(M,N)$$

The corresponding application morphism is the context morphism defined by $\Gamma, y{:}\,\sigma, z{:}\,\Pi x{:}\,\sigma.\tau \vdash (\vec{x}, \mathit{App}_{[x:\sigma]\tau}(z,y)) \Rightarrow \Gamma, x{:}\,\sigma, w{:}\,\tau$ where $\vec{x}$ are the variables in $\Gamma$. Notice that these Π-types are not strict unless we enrich our type theory with the abovementioned $\eta$-equation.

The set-theoretic model supports Π-types in the strict sense. If $(\sigma_\gamma)_{\gamma\in\Gamma} \in \mathit{Ty}(\Gamma)$ and $(\tau_{(\gamma,x)})_{(\gamma,x)} \in \Gamma.\sigma$ then we define $\Pi(\sigma,\tau)_\gamma \overset{\text{def}}{=} \Pi_{x\in\sigma_\gamma}\tau_{(\gamma,x)}$ and abstraction and application as their set-theoretic companions.

Definition 3.18 A CwF supports Σ-types if the following data are given:

- for any two types $\sigma \in \mathit{Ty}(\Gamma)$ and $\tau \in \mathit{Ty}(\Gamma.\sigma)$ there is a type $\Sigma(\sigma,\tau) \in \mathit{Ty}(\Gamma)$ such that

  $$\Sigma(\sigma,\tau)\{f\} = \Sigma(\sigma\{f\},\tau\{f^+\}) \qquad \Sigma\text{-S}$$

  whenever $f : \mathrm{B} \to \Gamma$.

- A morphism $\mathit{Pair}_{\sigma,\tau} : \Gamma.\sigma.\tau \to \Gamma.\Sigma(\sigma,\tau)$ such that $\mathsf{p}(\Sigma(\sigma,\tau)) \circ \mathit{Pair}_{\sigma,\tau} = \mathsf{p}(\sigma) \circ \mathsf{p}(\tau)$ and such that

  $$f^+ \circ \mathit{Pair}_{\sigma,\tau} = \mathit{Pair}_{\sigma\{f\},\tau\{f^+\}} \circ f^{++} \qquad \mathit{Pair}\text{-S}$$

  whenever $f : \mathrm{B} \to \Gamma$.

- For every type $\rho \in Ty(\Gamma.\Sigma(\sigma,\tau)$ and term $H \in Tm(\rho\{Pair_{\sigma,\tau}\})$ a term $\mathsf{R}^{\Sigma}_{\sigma,\tau,\rho}(H) \in Tm(\rho)$ such that

$$\mathsf{R}^{\Sigma}_{\sigma,\tau,\rho}(H)\{Pair_{\sigma,\tau}\} = H \qquad \Sigma\text{-C}$$

  and

$$\mathsf{R}^{\Sigma}_{\sigma,\tau,\rho}(H)\{f^+\} = \mathsf{R}^{\Sigma}_{\sigma\{f\},\tau\{f^+\},\rho\{f^+\}}(H\{f^{++}\}) \qquad \mathsf{R}^{\Sigma}\text{-S}$$

The term model supports $\Sigma$-types as follows: if $\Gamma, x{:}\,\sigma \vdash \tau$ type then $\Sigma(\sigma,\tau) \overset{\text{def}}{=} \Sigma x{:}\,\sigma.\tau$ and $Pair_{\sigma,\tau}$ is the context morphism

$$\Gamma, x{:}\,\sigma, y{:}\,\tau \vdash (\gamma, Pair_{[x:\sigma]\tau}(M,N)) \Rightarrow \Gamma, z{:}\,\Sigma x{:}\,\sigma.\tau$$

Finally, if $\Gamma, x{:}\,\sigma, y{:}\,\tau \vdash H : \rho[Pair_{[x:\sigma]\tau}(x,y)]$ then $\Gamma, z{:}\,\Sigma x{:}\,\sigma.\tau \vdash \mathsf{R}^{\Sigma}(H,z) : \rho$.

Also the set-theoretic model supports $\Sigma$-types as follows.

$$\Sigma(\sigma,\tau)_{\gamma\in\Gamma} \overset{\text{def}}{=} \{(x,y) \mid x \in \sigma_\gamma \wedge y \in \tau_{(\gamma,x)}\}$$

and

$$Pair_{\sigma,\tau}(\gamma,x,y) \overset{\text{def}}{=} (\gamma,(x,y))$$

This morphism $Pair$ is surjective and thus we can take $\Sigma$-C as the defining equation for $\mathsf{R}^{\Sigma}_{\sigma,\tau,\rho}$ which also ensures that $\mathsf{R}^{\Sigma}_{\sigma,\tau,\rho}(H)$ is the unique term with property $\Sigma$-C. Again, we say that $\Sigma$-types are supported in the strict sense if this is the case.

Definition 3.19 A CwF supports (intensional) identity types if for each type $\sigma \in Ty(\Gamma)$ the following data are given:

- a type $Id_\sigma \in Ty(\Gamma.\sigma.\sigma^+)$,
- a morphism $Refl_\sigma : \Gamma.\sigma \to I(\sigma)$ where $I(\sigma) = \Gamma.\sigma.\sigma^+.Id_\sigma$ such that $\mathsf{p}(Id_\sigma) \circ Refl_\sigma$ equals $\overline{\mathsf{v}_\sigma} : \Gamma.\sigma \to \Gamma.\sigma.\sigma^+$; the diagonal morphism,
- for each $\tau \in Ty(I(\sigma))$ a function $\mathsf{R}^{Id}_{\sigma,\tau} \in Tm(\tau\{Refl_\sigma\}) \to Tm(\tau)$

in such a way that these data are stable under substitution w.r.t. morphisms $f : \mathrm{B} \to \Gamma$ and such that whenever $\tau \in Ty(I(\sigma))$ and $H \in Tm(\tau\{Refl_\sigma\})$ then $\mathsf{R}^{Id}_{\sigma,\tau}(H)\{Refl_\sigma\} = H$.

The term model of intensional type theory supports identity types with the following settings: if $\Gamma \vdash \sigma$ type then $Id_\sigma$ is the type $\Gamma, x{:}\,\sigma, y{:}\,\sigma \vdash Id_\sigma(x,y)$ type and $Refl_\sigma$ is the context morphism

$$\Gamma, x{:}\,\sigma \vdash (\gamma, x, x, Refl_\sigma(x)) \Rightarrow \Gamma, x{:}\,\sigma, y{:}\,\sigma, p{:}\,Id_\sigma(x,y)$$

If $\Gamma, x{:}\,\sigma, y{:}\,\sigma, p{:}\,\mathit{Id}_\sigma(x,y) \vdash \tau[x,y,p]$ type and $\Gamma, x{:}\,\sigma \vdash H : \tau[x,x,\mathit{Refl}_\sigma(x)]$ then

$$\Gamma, x{:}\,\sigma, y{:}\,\sigma, p{:}\,\mathit{Id}_\sigma(x,y) \vdash \mathsf{R}^{\mathit{Id}}([x{:}\,\sigma]H, x, y, p) : \tau[x,y,p]$$

which defines the required function.

The set-theoretic model supports identity types with $(\mathit{Id}_\sigma)_{(\gamma,x,y)} \stackrel{\text{def}}{=} [x = y]$ and $\mathit{Refl}_\sigma(\gamma, x) \stackrel{\text{def}}{=} (\gamma, x, x, \star)$. For $H \in \mathit{Tm}_{\mathcal{S}et}(\tau\{\mathit{Refl}_\sigma\})$ we put

$$\mathsf{R}^{\mathit{Id}}_{\sigma,\tau}(H)(\gamma, x, y, p) \stackrel{\text{def}}{=} H(x)$$

which is type correct because in the presence of $p \in \mathit{Id}_\sigma(x,y)$ we have $x = y$ and furthermore $p$ itself equals $\star$.

The semantic interpretation of natural numbers, unit types, empty types follows the pattern of the $\Sigma$-types and is left to the reader. We conclude the definition of semantic analogues to type formers by specifying the interpretation of a universe closed under impredicative quantification.

Definition 3.20 A CwF supports the Calculus of Constructions if it supports $\Pi$-types and the following data are given:

- a type $\mathit{Prop} \in \mathit{Ty}(\top)$. We write $\mathit{Prop}$ also for $\top.\mathit{Prop}$,
- A type $\mathit{Prf} \in \mathit{Ty}(\mathit{Prop})$,
- for $\sigma \in \mathit{Ty}(\Gamma)$ and $p : \Gamma.\sigma \to \mathit{Prop}$ a morphism $\forall_\sigma(p) : \Gamma \to \mathit{Prop}$,
- a term former $\lambda_{\sigma,p}(-)$ and a morphism $\mathit{App}_{\sigma,p}$ like in Prop 3.16 establishing that $\mathit{Prf}\{\forall_\sigma(p)\}$ is a dependent function space of $\mathit{Prf}\{p\}$ over $\sigma$

such that all these data are stable under substitution.

The term model is a model for the Calculus of Constructions with $\mathit{Prop}$ equal to the type of propositions $\mathit{Prop}$ in $\diamond \vdash \mathit{Prop}$ type and the type of proofs equal to $\mathit{Prf}(x)$ in $x{:}\,\mathit{Prop} \vdash \mathit{Prf}(x)$ type. A morphism from $\Gamma.\sigma$ to $\mathit{Prop}$ takes the form of a term $\Gamma, x{:}\,\sigma \vdash P : \mathit{Prop}$. Then $\Gamma \vdash \forall x{:}\,\sigma.P : \mathit{Prop}$ is a morphism from $\Gamma$ to $\mathit{Prop}$ with the required properties.

The set-theoretic model is a model for the Calculus of Constructions with the settings $\mathit{Prop} \stackrel{\text{def}}{=} \{\mathtt{ff}, \mathtt{tt}\}$ and $\mathit{Prf}_x \stackrel{\text{def}}{=} [x = \mathtt{tt}]$. If $p : \Gamma.\sigma \to \mathit{Prop}$ then we define $\forall_\sigma(p)(\gamma) \stackrel{\text{def}}{=} [\text{“}p(\gamma, x) = \mathtt{tt} \text{ for all } x \in \sigma_\gamma\text{”}]$.

Notice that in this model all elements of a type $\mathit{Prf}(A)$ are equal. By “Reynolds' paradox” (Reynolds and Plotkin 1988) this has to be so in any set-theoretic interpretation of the Calculus of Constructions. In §3.4 below we describe a model where this is not the case.

Notice that $\mathit{Prf}\{\forall_\sigma(p)\}$ is not equal to $\Pi(\sigma, \mathit{Prf}\{p\})$ but in bijective correspondence because if $p(\gamma, x) = \mathtt{tt}$ for each $x \in \sigma_\gamma$ then $\mathit{Prf}\{\forall_\sigma(p)\} = \{\star\}$, whereas $\Pi(\sigma, \mathit{Prf}\{p\})_\gamma$ is the set of functions from $\sigma_\gamma$ to $\{\star\}$ which is a singleton set, but not $\{\star\}$. However, this bijective correspondence enables us to define the required operations $\lambda_{\sigma,p}$ and $\mathit{App}_{\sigma,p}$.

## Exercises

E3.21 Show that the truth value model from E3.7 supports $\Pi$ and $\Sigma$-types each in the strict sense, as well as identity types with the settings $\Pi(\sigma,\tau) \overset{\text{def}}{=} \sigma \supset \tau$ and $\Sigma(\sigma,\tau) \overset{\text{def}}{=} \sigma \wedge \tau$ and $Id_\sigma \overset{\text{def}}{=} \mathrm{tt}$, where $\supset$ and $\wedge$ denote implication and conjunction of truth values.

E3.22* Show that a CwF supports $\Sigma$-types in the strict sense iff it supports them and each morphism $Pair_{\sigma,\tau}$ is an isomorphism.

E3.23 Spell out what it means that *Id* and *Refl* are stable under substitution.

E3.24 Explain how a CwF $\mathcal{C}$ supports unit types if (but not only if) there exists $\mathbf{1} \in Ty(\top)$ such that $\top.\mathbf{1}$ is a terminal object in $\mathcal{C}$ or equivalently $\top.\mathbf{1}$ is isomorphic to $\top$.

E3.25 Define what it means for a model to support natural numbers. Show that the set-theoretic model supports natural numbers by $\mathbf{N} \overset{\text{def}}{=} \omega$ and that the truth value model supports natural numbers by $\mathbf{N} \overset{\text{def}}{=} \mathrm{tt}$.

E3.26* Define what it means to support an empty type in the sense of §2.1.7 and show that the set-theoretic model supports it by $\mathbf{0} \overset{\text{def}}{=} \emptyset$ and that the truth value model supports it by $\mathbf{0} \overset{\text{def}}{=} \mathrm{ff}$ and that this interpretation is the only possibility. Conclude that if $M, N \in Tm(\Gamma,\sigma)$ in this model then the set $Tm(\Gamma.Id_\sigma\{\langle\overline{M}, N^+\rangle_{\sigma^+}\}, \mathbf{0})$ is empty.

E3.27 Show that the truth value model forms a model of the Calculus of Constructions with $Prop \overset{\text{def}}{=} \mathrm{tt}$, $Prf(\star) \overset{\text{def}}{=} \mathrm{tt}$, $\forall_\sigma(\star) \overset{\text{def}}{=} \star$.

E3.28* Define the semantics of a universe containing a code for the empty type and a code for the unit type. Show that the set-theoretic model supports such a universe, but that the truth value model does not.

E3.29* Let $\mathcal{P}oset$ be the category of posets and monotone functions. Extend $\mathcal{P}oset$ to a CwF by defining $Ty(\Gamma)$ for poset $\Gamma$ as the set of down-closed subsets (ideals) of $\Gamma$. If $\sigma \in Ty(\Gamma)$ then define $\Gamma.\sigma$ as $\sigma$ considered as a poset and let $\mathsf{p}(\sigma)$ be the inclusion from $\sigma$ to $\Gamma$. Finally, define $Tm(\Gamma,\sigma)$ as $[\Gamma = \sigma]$. For $\tau \in Ty(\Gamma.\sigma)$ let $\Pi'(\sigma,\tau) \in Ty(\Gamma)$ be $\{\gamma{\in}\Gamma \mid \forall x \in \sigma.x \leq \gamma \Rightarrow x \in \tau\}$. Show that $Tm(\Gamma, \Pi'(\sigma,\tau)) = Tm(\Gamma.\sigma,\tau)$, but that nevertheless $\Pi'$ does not determine dependent function spaces on $\mathcal{P}oset$ because it is not stable under substitution. In fact, the CwF $\mathcal{P}oset$ does not support dependent function spaces.

E3.30* (Extensional type theory) Say that a CwF with identity types supports them in the strict sense if whenever $\sigma \in Ty(\Gamma)$ and $\tau \in Ty(I(\sigma))$ (recall that $I(\sigma) \overset{\mathrm{def}}{=} \Gamma.\sigma.\sigma^+.Id_\sigma$) and $H \in Tm(\tau)$ then $H = \mathsf{R}^{Id}_{\sigma,\tau}(H\{Refl_\sigma\})$. Note that this is the case for the CwFs $\mathcal{S}et$ and $\mathcal{P}$ (WhyΓ). (1) Show that a CwF with identity types supports them in the strict sense iff the morphism $Refl_\sigma : \Gamma.\sigma \to I(\sigma)$ is an isomorphism (with inverse $Refl_\sigma^{-1} \overset{\mathrm{def}}{=} (\mathsf{p}(\sigma^+))^+$). (2) Extensional type theory is usually formulated as a type theory with identity types where rules Id-E and Id-C are replaced by the rules

$$\frac{\Gamma \vdash P : Id_\sigma(M, N)}{\Gamma \vdash M = N : \sigma}\ \text{Reflection} \qquad \frac{\Gamma \vdash P : Id_\sigma(M, N)}{\Gamma \vdash P = Refl_\sigma(M)}\ \text{Id-Can}$$

Show that the term model of extensional type theory supports identity types in the strict sense. (3) Show that rule Id-Can is independent of Reflection by considering the set-theoretic model with the setting $(Id_\sigma)_{(\gamma,x,y)} = \{\star, \star'\}$ if $x = y$ and $\emptyset$ otherwise. "Independence" is understood informally here; to establish that Id-Can cannot be derived from Reflection one needs the soundness theorem 3.35 below. (4) Show that if a model supports identity types in the strict sense then it supports Σ-types iff it supports them in the strict sense.

## 3.4 The $\omega$-set model

In this section we present another important CwF which provides a non-trivial interpretation of the Calculus of Constructions in which there is a proposition $\hat{\mathbf{N}} \in Tm(Prop_\top)$ such that $Tm(Prf(\hat{\mathbf{N}}))$ has more than one element.

An $\omega$-set $X$ consists of a set (denoted $|X|$ or $X$) and a relation $\Vdash_X \subseteq \omega \times |X|$ between natural numbers and elements of $X$ with the property that for each $x \in |X|$ there is an $n \in \omega$ such that $n \Vdash_X x$. If $n \Vdash_X x$ one says that $n$ realises $x$ or codes $x$ or is a code/realiser for $x$ so the condition on $\Vdash_X$ means that every $x \in X$ has a realiser.

A morphism from $\omega$-set $X$ to $Y$ is a function $f$ from $X$ to $Y$ with the property that there exists a partial recursive function $e$ (thought of as a natural number) such that whenever $n \Vdash_X x$ then $\{e\}(n)$—the application of $e$ to $n$—is defined and realises $f(x)$. The $\omega$-sets together with their morphisms form a category.

An example of an $\omega$-set is $\mathbf{N}$ where $|\mathbf{N}| = \omega$ and $n \Vdash_{\mathbf{N}} m$ iff $n = m$. The morphisms in the category of $\omega$-sets from $\mathbf{N}$ to $\mathbf{N}$ are exactly the total recursive functions. Another example is $\top$ with $|\top| = \{\star\}$ and $n \Vdash_\top \star$ for all $n \in \omega$. This $\omega$-set is readily seen to be a terminal object. If $X$ is a set then we form an $\omega$-set $\triangle X$ by $|\triangle X| = X$ and $n \Vdash_{\triangle X} x$ for all $n \in \omega$ and $x \in X$.

The category of $\omega$-sets can be extended to a CwF as follows. For $\omega$-set $\Gamma$ the set $Ty(\Gamma)$ consists of families of $\omega$-sets indexed over $\Gamma$. Thus, $\sigma \in Ty(\Gamma)$ means that $\sigma = (\sigma_\gamma)_{\gamma\in\Gamma}$ and each $\sigma_\gamma$ is an $\omega$-set. A term $M \in Tm(\Gamma, \sigma)$ is a function assigning to each $\gamma \in \Gamma$ an element $M(\gamma) \in \sigma_\gamma$ such that there exists a partial recursive function $e$ with the property that for each $\gamma \in \Gamma$ and $n \Vdash_\Gamma \gamma$ the computation $\{e\}(n)$ is defined and $\{e\}(n) \Vdash_{\sigma_\gamma} M(\gamma)$.

The $\omega$-set $\Gamma.\sigma$ has as underlying set the disjoint union $\Sigma_{\gamma\in\Gamma}\sigma(\gamma)$ and $n \Vdash_{\Gamma.\sigma} (\gamma, x)$ if $L(n) \Vdash_\Gamma \gamma$ and $R(n) \Vdash_{\sigma(\gamma)} x$ where $L$ and $R$ are the inverses to a bijection like $P(m, n) \stackrel{\text{def}}{=} 2^m(2n + 1)$ from $\omega \times \omega$ to $\omega$. The remaining components are left to the reader as an exercise.

The $\omega$-set model supports $\Pi$-types with the setting $\Pi(\sigma, \tau)_\gamma = \{M \in \Pi x{\in}\, \sigma_\gamma.\tau_{(\gamma,x)} \mid \exists e \text{ such that } T(e, M)\}$ where $T(e, M)$ means that $n \Vdash_{\sigma_\gamma} x$ implies $\{e\}(n) \Vdash M(x)$ for all $x \in \sigma_\gamma$ and $n \in \omega$. If $M \in \Pi(\sigma, \tau)_\gamma$ and $e \in \Gamma$ then $e \Vdash_{\Pi(\sigma,\tau)_\gamma} M$ if $T(e, M)$. Application and abstraction are defined as in the set-theoretic model. The required realisers are obtained from untyped application and abstraction ("smn-theorem").

All the other type formers we have considered and many more are also supported by the $\omega$-set model.

### 3.4.1 Modest sets

In an $\omega$-set $X$ two different elements can have the same realiser. An $\omega$-set where this is not so is called a modest set. In other words a modest set consists of a set $X$ and a partial surjective function $\Vdash_X$ from $\omega$ to $X$. More generally, $\sigma \in Ty(\Gamma)$ is modest if each $\sigma_\gamma$ is a modest set. It is easy to see that if $\tau \in Ty(\Gamma.\sigma)$ is modest then so is $\Pi(\sigma, \tau)$ for arbitrary $\sigma$. A modest set $X$ induces a symmetric, transitive relation $\sim_X$ (a partial equivalence relation, per) on the set $\omega$ of natural numbers by putting $m \sim_X n$ if there exists a (necessarily unique) $x \in X$ such that $m \Vdash_X x$ and $n \Vdash_X x$. Conversely, if $R \subseteq \omega\times\omega$ is any per we can define a modest set $Prf(R)$ having as underlying set the quotient of $\{n \mid nRn\}$ by $R$ (which restricted to this set is an equivalence relation). We have $m \Vdash_{Prf(R)} [n]_R$ if $mRn$. If $X$ is modest then $X$ is isomorphic to $Prf(\sim_X)$ in the category of $\omega$-sets and a per $R$ is equal to $\sim_{Prf(R)}$. This suggests to interpret the Calculus of Constructions by putting $Prop = \triangle(PER)$ where $PER$ is the set of symmetric and transitive relations on $\omega$. Notice that $Prop$ is not modest. The above-defined operation $Prf$ then yields a type $Prf \in Ty(Prop)$. If $p : \Gamma \to Prop$ then $p(\gamma)$ is a per for each $\gamma \in \Gamma$ and we have $Prf\{p\}_\gamma = Prf(p(\gamma))$.

If $p : \Gamma.\sigma \to Prop$ then we define $\forall_\sigma(p) : \Gamma \to Prop$ by $(\forall_\sigma(p))(\gamma) \stackrel{\text{def}}{=} \sim_{\Pi(\sigma,Prf\{p\})_\gamma}$. The verifications are tedious but essentially straightforward.

For $mRn$ iff $m = n$ we have that $Prf(R)$ is isomorphic to the $\omega$-set $\mathbf{N}$ which means that we can interpret a type theory in which the impredicative universe $Prop$ contains a code for the natural numbers. $Prop$ also contains codes for other reasonable data types and indeed the $\omega$-sets furnish a model for the CID mentioned in §2.2.3. Beeson (1985) and Allen (1987) show how to model type theory with non-impredicative universes (no $\forall$) entirely within the pers.

## Exercises

E3.31 Let $X$ be any set. Show that every morphism in $\omega$-set from $\triangle X$ to $\mathbf{N}$ is constant and that for arbitrary $\omega$-set $Y$ every set-theoretic function from $Y$ to $X$ is a morphism in $\omega$-set from $Y$ to $\triangle X$.

E3.32 Explain how the set-theoretic model supports lists, cf. E2.3.

E3.33 Explain how the $\omega$-set model supports identity types in the strict sense.

E3.34* For any $\omega$-set $X$ a per $\mathcal{I}(X)$ is defined as the transitive closure of the relation which relates two numbers if they realise a common element of $X$. Define a morphism $\eta_X : X \to \mathit{Prf}(\mathcal{I}(X))$ which sends $x \in X$ to the equivalence class of a realiser of $x$. Show that if $Y$ is a modest set and $f : X \to Y$ then there exists a unique morphism $\hat{f} : \mathit{Prf}(\mathcal{I}(X)) \to Y$ such that $f = \hat{f} \circ \eta_X$. This means that the modest sets form a reflective sub-category of the $\omega$-sets. One can define impredicative quantification in the $\omega$-set model by $\forall_\sigma(M)_{\gamma\in\Gamma} \overset{\text{def}}{=} \mathcal{I}(\Pi x{:}\,\sigma.\mathit{Prf}(M)_\gamma)$. In (Ehrhard 1989) such a reflection is made part of the definition of a model for the Calculus of Constructions.

## 3.5 Interpretation of the syntax

We have already implicitly spoken about interpretation of type theory in CwFs by appealing to the informal analogy between components of CwFs and type-theoretic entities. In this section we make this precise by defining an interpretation function and establishing a soundness property.

Assume for the rest of this section a fixed type theory and a CwF $\mathcal{C}$ supporting the type and term formers present in this theory. For simplicity we restrict our attention to $\Pi$-types.

We define a partial interpretation function $[\![-]\!]$ which maps:

- pre-contexts to objects of $\mathcal{C}$
- pairs $\Gamma\,;\sigma$, where $\Gamma$ is a pre-context and $\sigma$ is a pre-type, to families in $\mathit{Ty}([\![\Gamma]\!])$
- pairs $\Gamma\,;M$, where $\Gamma$ is a pre-context and $M$ is a pre-term to elements of $\mathit{Tm}(\sigma)$ for some $\sigma \in \mathit{Ty}([\![\Gamma]\!])$.

The definition is by induction on the lengths of the involved pre-terms, -types, and -contexts. We show below in Thm. 3.35 that the semantic function is defined on all contexts, types, and terms.

The semantic clauses are the following, where we adopt the convention that an expression containing an undefined subexpression is itself undefined. We also adopt the convention that expressions which do not "typecheck", like $\Gamma.\sigma$ if $\sigma \notin \mathit{Ty}(\Gamma)$, are undefined.

- $[\![\diamond]\!] = \top$
- $[\![\Gamma, x{:}\,\sigma]\!] = [\![\Gamma]\!].[\![\Gamma\,;\sigma]\!]$ if $x$ not in $\Gamma$, undefined otherwise.
- $[\![\Gamma\,; \Pi x{:}\,\sigma.\tau]\!] = \Pi([\![\Gamma\,;\sigma]\!]\,,\,[\![\Gamma, x{:}\,\sigma\ ;\ \tau]\!])$
- $[\![\Gamma, x{:}\,\sigma\,; x]\!] = \mathsf{v}_{[\![\Gamma\,;\,\sigma]\!]}$
- $[\![\Gamma, x{:}\,\sigma, \Delta, y{:}\,\tau\,; x]\!] = [\![\Gamma, x{:}\,\sigma, \Delta\,; x]\!]\{\mathsf{p}([\![\Gamma, x{:}\,\sigma, \Delta\,;\tau]\!])\}$
- $[\![\Gamma\,; \mathit{App}_{[x:\sigma]\tau}(M, N)]\!] = \mathit{App}_{[\![\Gamma\,;\,\sigma]\!],[\![\Gamma,x:\sigma\,;\,\tau]\!]} \circ \langle\overline{[\![\Gamma\,;M]\!]}, [\![\Gamma\,;N]\!]^{+}\rangle_{[\![\Gamma\,;\,\Pi x:\sigma.\tau]\!]^{+}}$
- $[\![\Gamma\,; \lambda x{:}\,\sigma.M^{\tau}]\!] = \lambda_{[\![\Gamma\,;\,\sigma]\!],[\![\Gamma,x:\sigma\,;\,\tau]\!]}([\![\Gamma, x{:}\,\sigma\ ;\ M]\!])$

Notice that the semantics of variables is defined by induction on the depth of their declaration as an appropriate weakening of a $\mathsf{v}_{\_}$-expression.

Theorem 3.35 The interpretation function enjoys the following soundness properties

- If $\Gamma \vdash$ then $[\![\Gamma]\!]$ is an object of $\mathcal{C}$.
- If $\Gamma \vdash \sigma$ then $[\![\Gamma\,;\sigma]\!]$ is an element of $\mathit{Ty}([\![\Gamma]\!])$.
- If $\Gamma \vdash M : \sigma$ then $[\![\Gamma\,;M]\!]$ is an element of $\mathit{Tm}([\![\Gamma\,;\sigma]\!])$.
- If $\vdash \Gamma = \Delta\ \mathit{ctxt}$ then $[\![\Gamma]\!] = [\![\Delta]\!]$.
- If $\Gamma \vdash \sigma = \tau$ type then $[\![\Gamma\,;\sigma]\!] = [\![\Gamma\,;\tau]\!]$.
- If $\Gamma \vdash M = N : \sigma$ then $[\![\Gamma\,;M]\!] = [\![\Gamma\,;N]\!]$.

The proof of this theorem, although essentially straightforward, presents surprising technical difficulties. The idea is to first establish a substitution lemma which relates syntactic substitution (and weakening) and semantic substitution ($-\{-\}$). This is necessary because the equations governing semantic type formers such as Π-C are formulated w.r.t. $-\{-\}$, whereas their syntactic counterparts (here Π-C) refer to syntactic substitution which is defined by structural induction. Due to type dependency one needs to account for substitution and weakening in the middle and not merely at the end of a context.

We need some notation first. For pre-contexts $\Gamma, \Delta$ and pre-type $\rho$ we define the expression $\mathbf{P}(\Gamma; \rho; \Delta)$ inductively by

$$\begin{aligned}&\mathbf{P}(\Gamma; \rho; \diamond) = \mathsf{p}([\![\Gamma\,; \rho]\!])\\ &\mathbf{P}(\Gamma; \rho; \Delta, x{:}\,\sigma) = \mathsf{q}(\mathbf{P}(\Gamma; \rho; \Delta), [\![\Gamma, \Delta\,; \sigma]\!])\end{aligned}$$

Now let $\Gamma, \Delta, \rho$ be as before and $M$ be a pre-term. We define the expression inductively by the following clauses.

$$\begin{aligned}&\mathbf{T}(\Gamma; \rho; \diamond; M) = \overline{[\![\Gamma; M]\!]}\\ &\mathbf{T}(\Gamma; \rho; \Delta, x{:}\,\sigma; M) = \mathsf{q}(\mathbf{T}(\Gamma; \rho; \Delta), [\![\Gamma, z{:}\,\rho, \Delta\,; \sigma]\!]) \qquad z \text{ fresh}\end{aligned}$$

The idea is that $\mathbf{P}(\Gamma; \rho; \Delta)$ is a morphism from $[\![\Gamma, z{:}\,\rho, \Delta]\!]$ to $[\![\Gamma, \Delta]\!]$ projecting out the $\rho$-part. Similarly, $\mathbf{T}(\Gamma; \rho; \Delta; M)$ is intended to go from $[\![\Gamma, \Delta[M/z]]\!]$ to $[\![\Gamma, z{:}\,\rho, \Delta]\!]$ yielding $[\![\Gamma\,; M]\!]$ at the $z{:}\,\rho$ position and variables otherwise. But that this is really the case has to be proved simultaneously with the weakening and substitution lemmas.

For possibly undefined expressions $s, t$ we write $s \simeq t$ to mean that if either side is defined then so is the other and both agree (Kleene equality).

Lemma 3.36 (Weakening) Let $\Gamma, \Delta$ be pre-contexts, $\rho, \sigma$ be pre-types, $N$ be a pre-term and $z$ be a fresh variable. Let $X \in \{\sigma, N\}$. The expression $\mathbf{P}(\Gamma; \rho; \Delta)$ is defined iff $[\![\Gamma, z{:}\,\rho, \Delta]\!]$ and $[\![\Gamma, \Delta]\!]$ are defined and in this case is a morphism from the former to the latter. If $[\![\Gamma, \Delta\,; X]\!]$ is defined then

$$[\![\Gamma, z{:}\,\rho, \Delta\,; X]\!] \simeq [\![\Gamma, \Delta\,; X]\!]\{\mathbf{P}(\Gamma; \rho; \Delta)\}$$

Lemma 3.37 (Substitution) Let $\Gamma, \Delta$ be pre-contexts, $\rho, \sigma$ be pre-types, $M, N$ be pre-terms, and $z$ a fresh variable. Let $X \in \{\sigma, N\}$ and suppose that $[\![\Gamma\,; M]\!]$ is defined.

The expression $\mathbf{T}(\Gamma; \rho; \Delta; M)$ is defined iff $[\![\Gamma, \Delta[M/z]]\!]$ and $[\![\Gamma, z{:}\,\rho, \Delta]\!]$ are both defined and in this case is a morphism from the former to the latter. If $[\![\Gamma, z{:}\,\rho, \Delta\,; X]\!]$ is defined then

$$[\![\Gamma, \Delta[M/z]\,; X]\!] \simeq [\![\Gamma, z{:}\,\rho, \Delta]\!]\{\mathbf{T}(\Gamma; \rho; \Delta; M)\}$$

The proofs of both lemmas proceed by induction on the lenghts of the involved pre-terms,-types, and -contexts. The most difficult case arises when the term $N$ is a variable. One must then make a case distinction on whether it is declared in $\Gamma$ or in $\Delta$ and perform some equational reasoning. The other cases follow by applying stability under substitution of the participating semantic type and term formers such as $\Pi$-S. If the type $\sigma$ or the term $N$ is a binder like $\Pi x{:}\,\phi.\psi$ the inductive hypothesis must be used with $\Delta$ extended by $z{:}\,\phi$. In order to prove the correct typing of the $\mathbf{P}$ and $\mathbf{T}$ morphisms one uses the second part of each lemma with shortened $\Delta$.

After this introduction we leave the proofs themselves to the reader as following the calculations seems to require as much effort as doing them by oneself.

Given the above two lemmas the proof of the soundness theorem becomes a straightforward induction on derivations. For the same reason as before this proof will not be reproduced here; the reader is instead encouraged to carry out at least a few cases of the inductive argument by himself and in case of serious difficulty consult Streicher's monograph (1991) where the verification of the corresponding interpretation in contextual categories is spelt out in a very detailed fashion.

The method of partially interpreting pre-syntax first and proving definedness by induction on derivations was invented by Streicher (1991) and is used in order to ensure that the interpretation does not depend on the particular derivation chosen. Notice that derivations of a given judgement are not unique as instances of the conversion rules Ty-Conv and Tm-Conv are not recorded in judgements. Alternatively, one could define the interpretation by induction on derivations and then use the device of pre-syntax to establish coherence of the semantics with respect to the conversion rules, but this would result in a more complicated proof. For a particular model this has been carried out in (Palmgren and Stoltenberg-Hansen 1990).

## Exercises

E3.38 Extend the interpretation function to $\Sigma$-types

E3.39* Formulate a completeness theorem for the semantics and use the term model to prove it. Deduce from the term model that if $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$ then $\Gamma \vdash \sigma = \tau$ type.

E3.40 Explain why the interpretation function cannot be defined in the same way as we did in case application is not typed, i.e., with $App_{[x:\sigma]\tau}(M, N)$ replaced by $(M\,N)$. Also explain why it would not matter to leave out the type annotation $\tau$ in $\lambda x{:}\,\sigma.M^\tau$ and similar situations.

E3.41 Try to define the interpretation function on triples $\Gamma\,; M\,; \sigma$ of typed terms in context rather than on the pairs $\Gamma\,; M$. What goes wrong$\Gamma$

E3.42* Extend the interpretation function to syntactic context morphisms and state a general substitution lemma for context morphisms (Pitts 1997). It seems difficult to prove this general substitution lemma directly without using Lemmas 3.36 & 3.37.

E3.43 Deduce from E3.26 and the soundness theorem that the following type is not inhabited in a type theory without universes.

$$\diamond \vdash \mathit{Id}_{\mathbf{N}}(0, \mathit{Suc}(0)) \to \mathbf{0} \text{ type}$$

E3.44 Use the $\omega$-set model to derive that in the Calculus of Constructions with natural numbers there does not exist a term $\diamond \vdash M : \mathit{Prop} \to \mathbf{N}$ and propositions $\diamond \vdash P : \mathit{Prop}$ and $\diamond \vdash Q : \mathit{Prop}$ such that $\diamond \vdash \mathit{App}(M, P) = 0 : \mathbf{N}$ and $\diamond \vdash \mathit{App}(M, Q) = \mathit{Suc}(0) : \mathbf{N}$

E3.45* Show that the interpretation of existential quantification $\exists x{:}\,\sigma.P$ as defined in E2.8 in the set-theoretic model is $\mathtt{tt}$ if $P(x)$ is $\mathtt{tt}$ for some $x \in [\![\sigma]\!]$ and $\mathtt{ff}$ otherwise. Conclude that the following extension of the Calculus of Constructions by a choice function is consistent in the sense that neither the empty type (if it exists) nor the type $\mathit{Prf}(\forall c{:}\,\mathit{Prop}.c)$ is inhabited in the empty context:

$$\frac{\Gamma \vdash H : \mathit{Prf}(\exists x{:}\,\sigma.P)}{\Gamma \vdash \mathit{Choice}(H) : \sigma}$$

It is known that such choice function becomes inconsistent if in addition one imposes the equation

$$\vdash \mathit{Choice}(\exists\text{-I}(M, N)) = M : \sigma$$

for all $M{:}\,\sigma$ and $N{:}\,\mathit{Prf}(P[M])$, see (Coquand 1990).

## 4 Extended example: presheaf models

In this section we encounter a family of interpretations of type theory which generalises the set-theoretic model in that types are interpreted as variable sets (presheaves) or families of such. There are various applications of such interpretations, see for example (Asperti and Martini 1992) and (Altenkirch, Hofmann, and Streicher 1996) where they are used to define models of the polymorphic lambda calculus in which type quantification is interpreted as cartesian product. We use a presheaf interpretation here to show that the Logical Framework in the sense of §2.2.2 forms a conservative extension of ordinary type theory. This result appears here for the first time; a more detailed version will be published elsewhere.

Preliminaries. Let $\mathcal{K}$ be a (small) category. A presheaf over $\mathcal{K}$ is a contravariant functor from $\mathcal{K}$ to the category $\mathcal{S}et$ of sets and functions. We denote by $\hat{\mathcal{K}}$ the functor category $\mathcal{S}et^{\mathcal{K}^{op}}$ of presheaves over $\mathcal{K}$. We denote presheaf application by subscripting. That is, if $F \in \hat{\mathcal{K}}$ and $u \in \mathcal{K}(I, J)$ then

$F_u : F_J \to F_I$. If $\mu \in \hat{\mathcal{K}}(F,G)$ then $\mu_I : F_I \to G_I$. We may think of $\mathcal{K}$ as a category of stages or worlds and of a presheaf as a set varying with these stages. The Yoneda embedding $\mathbf{y} : \mathcal{K} \to \hat{\mathcal{K}}$ sending $I$ to $\mathcal{K}(-,I)$ is a full and faithful embedding of the stages into $\hat{\mathcal{K}}$. See (Barr and Wells 1990) for more information on presheaves.

## 4.1 Presheaves as a CwF

Our aim is to construct a CwF which has $\hat{\mathcal{K}}$ as the category of contexts. This category has a terminal object given by $\top_I = \{\star\}$. We define types, substitution, terms, and comprehension in order. From now on we will use the generic letters $\Gamma, f, \sigma, \ldots$ to range over presheaves, natural transformations, families of presheaves, etc. if these arise in the context of a CwF.

If $\Gamma$ is a presheaf then we form its category of elements $\int(\Gamma)$ with objects $(I,\gamma)$ where $I \in \mathcal{K}$ and $\gamma \in \Gamma_I$. A morphism from $(J,\gamma')$ to $(I,\gamma)$ is a $\mathcal{K}$-morphism $u \in \mathcal{K}(J,I)$ such that $\Gamma_u(\gamma) = \gamma'$. In other words, if $u \in \mathcal{K}(J,I)$ and $\gamma \in \Gamma_I$ then $u$ is an $\int(\Gamma)$-morphism from $(J,\Gamma_u(\gamma))$ to $(I,\gamma)$. If appropriate we may also write $(u,\gamma)$ for this morphism. Composition is inherited from $\mathcal{K}$.

For presheaf $\Gamma \in \hat{\mathcal{K}}$ we define the set $\mathit{Ty}(\Gamma)$ to consist of the presheaves over $\int(\Gamma)$. If $\sigma \in \mathit{Ty}(\Gamma)$ and $(I,\gamma) \in \int(\Gamma)$ then we write $\sigma_I(\gamma)$ rather than $\sigma_{(I,\gamma)}$ for $\sigma$ at argument $(I,\gamma)$ and similarly for morphisms. That $\sigma$ is a presheaf means that if $u \in \mathcal{K}(J,I)$ and $\gamma \in \Gamma_I$ and $x \in \sigma_I(\gamma)$ then $\sigma_u(\gamma)(x) \in \sigma_J(\Gamma_u(\gamma))$ and this action is compatible with composition and identities in $\mathcal{K}$.

Now suppose that $f : \Delta \to \Gamma$ is a natural transformation. We define a functor $\int(f) : \int(\Delta) \to \int(\Gamma)$ by $\int(f)(I,\delta) = (I, f_I(\delta))$ and for $u \in \mathcal{K}(J,I)$, $\delta \in \Delta_I$ we define $\int(f)(u,\delta) = (u, f_I(\delta))$. Now if $\sigma \in \mathit{Ty}(\Gamma)$ is a family of presheaves then we define $\sigma\{f\} \in \mathit{Ty}(\Delta)$ as the composition $\sigma \circ \int(f)$. So, we have $\sigma\{f\}_I(\delta) = \sigma_I(f_I(\delta))$. It is clear that this has the required functoriality properties.

Next we define terms. If $\sigma \in \mathit{Ty}(\Gamma)$ then an element $M$ of $\mathit{Tm}(\sigma)$ assigns to each stage $I$ and element $\gamma \in \Gamma_I$ an element $M_I(\gamma) \in \sigma_I(\gamma)$ in such a way that if $u \in \mathcal{K}(J,I)$ then $\sigma_u(\gamma)(M_I(\gamma)) = M_J(\Gamma_u(\gamma))$. If $M \in \mathit{Tm}(\sigma)$ and $f : \Delta \to \Gamma$ then $M\{f\} \in \mathit{Tm}(\sigma\{f\})$ is given by composition as $M\{f\}_I(\delta) = M_I(f_I(\delta))$.

If $\sigma \in \mathit{Ty}(\Gamma)$ then the presheaf $\Gamma.\sigma$ is defined by $(\Gamma.\sigma)_I = \{(\gamma,x) \mid \gamma \in \Gamma_I,\ x \in \sigma_I(\gamma)\}$. If $u \in \mathcal{K}(J,I)$ and $(\gamma,x) \in (\Gamma.\sigma)_I$ then $(\Gamma.\sigma)_u(\gamma,x) = (\Gamma_u(\gamma), \sigma_u(\gamma)(x))$. The projection $\mathsf{p}(\sigma)$ is defined by $\mathsf{p}(\sigma)_I(\gamma,x) = \gamma$; the variable $\mathsf{v}_\sigma \in \mathit{Tm}(\sigma\{\mathsf{p}(\sigma)\})$ is defined by $(\mathsf{v}_\sigma)_I(\gamma,x) = x$. Finally, if $f : \Delta \to \Gamma$ and $M \in \mathit{Tm}(\sigma\{f\})$ then $(\langle f, M\rangle_\sigma)_I(\delta) = (f_I(\delta), M_I(\delta))$. The verifications are straightforward expansions of the definitions. We have thus established that presheaves over $\mathcal{K}$ furnish a CwF.

## 4.2 Type formers in $\hat{\mathcal{K}}$

The presheaf model supports most of the type formers the set-theoretic model supports, often by the very same constructions. The major difference is the interpretation of Π-types which is carried out in a way similar to the treatment of implication in Kripke models, and of course similar to the definition of exponentiation in functor categories.. Suppose that $\sigma \in \mathit{Ty}(\Gamma)$ and $\tau \in \mathit{Ty}(\Gamma.\sigma)$. Following the convention from §2.2.2 we will write $(\sigma)\tau$ rather than $\Pi(\sigma,\tau)$ for the dependent function space of $\sigma$ and $\tau$. It is defined as follows. If $I \in \mathcal{K}$ and $\gamma \in \Gamma_I$ then an element $f$ of $(\sigma)\tau_I(\gamma)$ associates to $J \in \mathcal{K}$ and $w : J \to I$ and $x \in \sigma_J(\Gamma_w(\gamma))$ an element $f(w,x) \in \tau_J(\Gamma_w(\gamma), \sigma_w(\gamma)(x))$. Notice that $w \in \int(\Gamma)((J, \Gamma_w(\gamma))\,,\,(I,\gamma))$ and thus $\sigma_w(\gamma)(x) \in \sigma_I(\Gamma_w(\gamma))$. Moreover the assignment $f$ must be natural in the sense that if in addition $w' : J' \to J$ then

$$f(w \circ w', \sigma_{w'}(\Gamma_w(\gamma))(x)) = \tau_{w'}(\Gamma_w(\gamma), \sigma_w(\gamma)(x))(f(w,x))$$

The generalisation to "later stages" $J$ is necessary to make $(\sigma)\tau$ a presheaf and not merely an assignment of sets to stages. Indeed, if $v : I' \to I$ and $f \in (\sigma)\tau_I(\gamma)$ then we can define $(\sigma)\tau_v(\Gamma_v(\gamma))(f) \in (\sigma)\tau_I'(\Gamma_v(\gamma))$ by

$$(\sigma)\tau_v(\Gamma_v(\gamma))(f)(w\colon J \to I', x{\in}\,\sigma_J(\Gamma_w(\Gamma_v(\gamma)))) \overset{\text{def}}{=} f(v \circ w, x)$$

which is valid because $\Gamma_w(\Gamma_v(\gamma)) = \Gamma_{v\circ w}(\gamma)$.

If $M \in \mathit{Tm}(\Gamma.\sigma, \tau)$ then $\lambda_{\sigma,\tau}(M) \in \mathit{Tm}((\sigma)\tau)$ is given by

$$\lambda(M)_I(\gamma{\in}\,\Gamma_I\;,\; J{\in}\,\mathcal{K}, w : J \to I, x{\in}\,\sigma_J(\Gamma_w(\gamma))) \overset{\text{def}}{=} M_J(\Gamma_w(\gamma), x)$$

The application morphism $\mathit{App}_{\sigma,\tau} : \Gamma.\sigma.(\sigma)\tau^+ \to \Gamma.\sigma.\tau$ maps at stage $I$ elements $\gamma \in \Gamma_I$ and $x \in \sigma_I(\gamma)$ and $f \in (\sigma)\tau_I(\gamma)$ to $f(id_I, x) \in \tau_I(\gamma, x)$. It is routine to check that this defines dependent function spaces in the strict sense.

The dependent sum $\Sigma(\sigma,\tau)$ of the presheaves above is given by stage-wise set-theoretic dependent sum. This means that

$$\Sigma(\sigma,\tau)_I(\gamma) = \{(x,y) \mid x{\in}\,\sigma_I(\gamma) \wedge y{\in}\,\tau_I(\gamma,x)\}$$

We omit the associated term formers. We remark without proof that $\hat{\mathcal{K}}$ supports natural numbers and other inductive types; it also supports universes if the ambient set theory w.r.t. which the presheaves are formed supports them.

## 4.3 Conservativity of the logical framework

Let $\mathcal{T}$ be a theory of dependent types like one of those set out in §2. The precise nature of $\mathcal{T}$ is unimportant.

Furthermore, we let $\mathcal{T}_{LF}$ denote the Logical Framework presentation of $\mathcal{T}$, i.e., a dependent type theory with $\Pi$-types and one universe *Set* together with constants and equations corresponding to the type and term formers in $\mathcal{T}$. We have a conversion map $\mathbf{i}$ translating terms, types, and judgements in $\mathcal{T}$ into ones in $\mathcal{T}_{LF}$ in such a way that judgements are preserved. More precisely, if $\Gamma \vdash \mathcal{J}$ in $\mathcal{T}$ then $\mathbf{i}(\Gamma) \vdash \mathbf{i}(\mathcal{J})$ in $\mathcal{T}_{LF}$. For distinction, we write $\vdash_{\mathcal{T}}$ and $\vdash_{\mathcal{T}_{LF}}$ for the judgement relations in the two type theories. For example, in $\mathcal{T}$ we might have

$$x{:}\,\mathbf{N} \vdash_{\mathcal{T}} \Pi y{:}\,\mathbf{N}.\mathit{Id}(x,y) \to \mathit{Id}(y,x) \text{ type}$$

Applying the translation yields the following judgement in $\mathcal{T}_{LF}$:

$$x{:}\,\mathit{El}(\hat{\mathbf{N}}) \vdash_{\mathcal{T}_{LF}} \mathit{El}(\hat{\Pi} y{:}\,\mathit{El}(\hat{\mathbf{N}}).\hat{\mathit{Id}}(x,y)\hat{\to}\hat{\mathit{Id}}(y,x)) \text{ type}$$

In other words, $\mathbf{i}(\mathbf{N}) = \mathit{El}(\hat{\mathbf{N}})$, etc. If we omit the *El*-operator and the $\hat{-}$-decorations then the translated judgement looks exactly like the one to start with. We will do so in the sequel and omit the coercion $\mathbf{i}$.

Due to the presence of type variables $\mathcal{T}_{LF}$ is a proper extension of $\mathcal{T}$. Judgements like $\vdash \mathit{Set}$ type or $F{:}\,\mathbf{N} \to \mathit{Set} \vdash F(0)$ are not in the image of $\mathbf{i}$. A natural question to ask is whether $\mathcal{T}_{LF}$ is conservative over $\mathcal{T}$. There is more than one way to extend the notion of conservativity from logic to theories of dependent types. We will here use the simplest one and prove the following theorem:

Theorem 4.1 If $\Gamma \vdash_{\mathcal{T}} \sigma$ type and $\Gamma \vdash_{\mathcal{T}_{LF}} M : \sigma$ for some term $M$ then there exists a term $M'$ such that $\Gamma \vdash_{\mathcal{T}} M' : \sigma$.

Notice that $M$ itself need not be a legal $\mathcal{T}$-term, it could for instance contain a subterm like $([X{:}\,\mathit{Set}]\mathit{Suc}(0))(\mathbf{N})$ which is equal, but not identical to $\mathit{Suc}(0)$.

Our strategy for proving this theorem consists of exhibiting a model of $\mathcal{T}_{LF}$ with the property that the interpretation of $\mathcal{T}$ in it (notice that such a model also models $\mathcal{T}$) is full. That is to say, if $\mathit{Tm}([\![\Gamma\,;\sigma]\!]) \neq \emptyset$ for some $\Gamma \vdash_{\mathcal{T}} \sigma$ type then there exists $M$ with $\Gamma \vdash_{\mathcal{T}} M : \sigma$. Such a model is furnished by the presheaf model $\hat{\mathcal{T}}$ where $\mathcal{T}$ is the (category of contexts of the) term model of $\mathcal{T}$. The fullness of the interpretation of $\mathcal{T}$ in $\hat{\mathcal{T}}$ is essentially a consequence of the Yoneda lemma and will be proved at the end of this section.

Let us first show how $\hat{\mathcal{T}}$ models the Logical Framework. We have already demonstrated that $\hat{\mathcal{T}}$ (like every presheaf model) supports dependent function spaces. We interpret $\mathit{Set} \in \mathit{Ty}_{\hat{\mathcal{T}}}(\top) \cong \hat{\mathcal{T}}$ as the presheaf $\mathit{Ty}_{\mathcal{T}}$ which to a context $\Gamma \in \mathcal{T}$ associates the $\mathcal{T}$-types in context $\Gamma$ quotiented by definitional equality. The interpretation of $\mathit{El} \in \mathit{Ty}_{\hat{\mathcal{T}}}(\top.\mathit{Set}) \equiv \widehat{\textstyle\int(\mathit{Set})}$ is defined as the presheaf $\mathit{Tm}_{\mathcal{T}}$ which to $\Gamma \in |\mathcal{T}|$ and $\sigma \in \mathit{Ty}_{\mathcal{T}}(\Gamma) = \mathit{Set}_{\Gamma}$ associates the set $\mathit{Tm}_{\mathcal{T}}(\Gamma,\sigma)$ of terms of type $\sigma$ in context $\Gamma$ factored by definitional equality. This extends to a presheaf by term substitution.

For the demonstration that $\hat{\mathcal{T}}$ models $\mathcal{T}_{LF}$ it remains to show that *Set* contains codes for all the type and term formers present in $\mathcal{T}$. We will deal with this task by way of example and assume that $\mathcal{T}$ contains $\Pi$-types and the (ad hoc) operator $\mathbf{L}$ from §2.2.2.

We deal with the $\mathbf{L}$-operator first. In order to simplify the notation and in view of the soundness and completeness of the semantics we will use the syntax of type theory with named variables to denote entities in $\hat{\mathcal{T}}$. Thus we require an element $\hat{\mathbf{L}} \in \mathit{Tm}(\top, (\mathit{Set})\mathit{Set})$ and an element $\hat{\mathbf{l}} \in \mathit{Tm}(\top, (\sigma\colon \mathit{Set}, m\colon \mathit{El}(\sigma))\mathit{El}(\mathbf{L}(\sigma)))$. Since the dependent function spaces in $\hat{\mathcal{T}}$ are strict the first task is equivalent to exhibiting an element, for simplicity also denoted $\hat{\mathbf{L}}$, of $\mathit{Tm}_{\hat{\mathcal{T}}}(\sigma\colon \mathit{Set}\,,\ \mathit{Set})$ which we describe explicitly by

$$\hat{\mathbf{L}}_\Gamma(\sigma \in \mathit{Ty}_{\mathcal{T}}(\Gamma)) \overset{\text{def}}{=} \mathbf{L}(\sigma)$$

Recall that $\mathit{Set}_\Gamma = \mathit{Ty}_{\mathcal{T}}(\Gamma)$. The naturality of this assignment amounts to checking that for $f : \mathrm{B} \to \Gamma$ and $\sigma \in \mathit{Ty}_{\mathcal{T}}(\Gamma)$ we have $\mathbf{L}(\sigma)[f] = \mathbf{L}(\sigma[f])$ which is immediate from the properties of syntactic substitution.

Similarly, the second task is equivalent to finding a term

$$\hat{\mathbf{l}} \in \mathit{Tm}_{\hat{\mathcal{T}}}(\sigma\colon \mathit{Set}, M\colon \mathit{El}(\sigma)\,,\ \mathit{El}(\hat{\mathbf{L}}(\sigma)))$$

Again, we define it explicitly by

$$\hat{\mathbf{l}}_\Gamma(\sigma\in \mathit{Ty}_{\mathcal{T}}(\Gamma), M\in \mathit{Tm}_{\mathcal{T}}(\Gamma,\sigma)) = \mathbf{l}(M) \in \mathit{Tm}_{\mathcal{T}}(\Gamma,\sigma)$$

Recall here that $\mathit{El}_\Gamma(\sigma) = \mathit{Tm}_{\mathcal{T}}(\Gamma,\sigma)$. Naturality is again a consequence of stability under substitution. Notice that up to the necessary abstraction e.g. from $\mathit{Tm}(\sigma\colon \mathit{Set}\,,\ \mathit{Set})$ to $\mathit{Tm}((\mathit{Set})\mathit{Set})$ the required constants in $\hat{\mathcal{K}}$ are given exactly by their counterparts in $\mathcal{T}$.

In essence, this is also the case for the $\Pi$-type, but due to the binding behaviour we encounter a slight complication. We wish to construct an element of

$$\mathit{Tm}_{\hat{\mathcal{T}}}(\top, (\sigma\colon \mathit{Set}, \tau\colon (\mathit{El}(\sigma))\mathit{Set})\mathit{Set})$$

By "uncurrying" this amounts to constructing a term $\hat{\Pi}$ of

$$\mathit{Tm}_{\hat{\mathcal{T}}}(\sigma\colon \mathit{Set}, \tau\colon (\mathit{El}(\sigma))\mathit{Set}\,,\ \mathit{Set})$$

At stage $\Gamma$ the arguments to $\hat{\Pi}$ are a type $\sigma \in \mathit{Ty}_{\mathcal{T}}(\Gamma)$ and an element of $(\mathit{El}(\sigma))\mathit{Set}_\Gamma$. Call the latter set $X$ temporarily. An element $\tau$ of $X$ associates by definition of dependent function spaces in $\hat{\mathcal{K}}$ to $f : \mathrm{B} \to \Gamma$ and $M \in \mathit{Tm}_{\mathcal{T}}(\mathrm{B}, \sigma\{f\})$ a type $\tau(\mathrm{B}, f, M) \in \mathit{Ty}(\mathrm{B})$. This assignment is natural in the sense that for $f' : \mathrm{B}' \to \mathrm{B}$ we have $\tau(\mathrm{B}', f \circ f', M\{f'\}) = \tau(\mathrm{B}, f, M)\{f'\} \in \mathit{Ty}(\mathrm{B}')$. But this means that the whole of $\tau$ can be reconstructed from its particular instance $\tau_0 \overset{\text{def}}{=} \tau(\Gamma.\sigma, \mathsf{p}(\sigma), \mathsf{v}_\sigma) \in \mathit{Ty}(\Gamma.\sigma)$. Indeed, for arbitrary

$f : \mathrm{B} \to \Gamma$ and $M \in Ty(\sigma\{f\}$ we have that $\tau(\mathrm{B}, f, M) = \tau_0\{\langle f, M\rangle_\sigma\}$ by naturality and equations (Cons-L) and (Cons-R). By (Cons-Id) we also get the converse and have thus established a bijective correspondence between the sets $(El(\sigma))Set_\Gamma$ and $Ty_{\mathcal{T}}(\Gamma.\sigma = Set_{\Gamma.\sigma}$. Therefore, the arguments at stage $\Gamma$ to the term $\hat{\Pi}$ which we aim to construct amount to a type $\sigma \in Ty(\Gamma)$ and a type $\tau_0 \in Ty(\Gamma.\sigma)$. We define the result as the syntactic dependent function space $\Pi(\sigma, \tau_0)$ in $\mathcal{T}$. Summing up, we have defined

$$\Pi_\Gamma(\sigma \in Set_\Gamma, \tau \in (El(\sigma))Set) = \Pi_{\mathcal{T}}(\sigma, \tau(\Gamma.\sigma, \mathsf{p}(\sigma), \mathsf{v}_\sigma))$$

So up to the bijection between the function space $(El(\sigma))Set_\Gamma$ and the set of types in $\Gamma.\sigma$ the $\Pi$-constant in $\mathcal{T}_{LF}$ has again been obtained directly from its syntactic companion. The same goes for the other constants $\hat{\lambda}$ and $\hat{App}$ whose definition we omit. In the case of $\hat{\lambda}$ we face again an argument of the dependent function space type $(x\colon El(\sigma))El(\tau)$ which at stage $\Gamma$ is isomorphic to $Tm_{\mathcal{T}}(\Gamma.\sigma\,,\,\tau)$ by an analysis similar to the one which led to the characterisation of $(El(\sigma))Set$ before.

One can more generally characterise the dependent function spaces of the form $(El(-))F$ for arbitrary presheaf $F$ as certain "shifts" of $F$. This allows for a systematic translation of arbitrary type and term formers possibly binding variables from $\mathcal{T}$ to $\mathcal{T}_{LF}$. The general strategy should have become clear from the example of $\Pi$. The important point is that in $\hat{\mathcal{K}}$ the function space $(El(\sigma))Set$ is so strongly confined by the naturality condition that it only contains functions induced by a syntactic type with free variable of type $\sigma$.

Proof of Thm. 4.1 Suppose that $\Gamma \vdash_{\mathcal{T}} \sigma$ type and that $\Gamma \vdash_{\mathcal{T}_{LF}} M : \sigma$. By induction on derivations we find that the interpretations of $\Gamma$ and $\sigma$ in $\hat{\mathcal{T}}$ have the following properties:

$$[\![\Gamma]\!]_\Delta \cong \mathcal{T}(\Delta, \Gamma)$$

and

$$[\![\Gamma\,;\sigma]\!]_\Delta(f \in \mathcal{T}(\Delta, \Gamma)) = \sigma[f] \in Set_\Delta$$

Thus, in particular, we have $[\![\Gamma\,;\sigma]\!]_\Gamma(id_\Gamma) = \sigma$. Therefore, the interpretation of $M$ in $\hat{\mathcal{T}}$ yields $[\![\Gamma\,;M]\!]_\Gamma(id_\Gamma) \in Tm_{\mathcal{T}}(\Gamma, \sigma)$; thus $\Gamma \vdash_{\mathcal{T}} M' : \sigma$ for any representative $M'$ in the class $[\![\Gamma\,;M]\!]$. The theorem is proved. $\square$

One may consider a Logical Framework which does not only support $\Pi$-types, but several other type formers like $\Sigma$-types, e.g. for modularisation and natural numbers, e.g. to define syntax. As long as these type formers are supported by $\hat{\mathcal{T}}$ ($\Sigma$ and $\mathbf{N}$ are) the conservativity theorem continues to hold by the same proof. Using a dependent version of the glueing construction (Crole 1993) it is possible to obtain the stronger property that the term $M'$ in Thm. 4.1 is $\mathcal{T}_{LF}$-equal to $M$.

We also remark that we have not used any particular properties of the term model in the construction of the presheaf model so that it can be formed out of any CwF and thus gives a canonical way to lift a model of some type theory to a model for the presentation of this type theory in the Logical Framework.

## 5 Other applications of semantic methods

We give some directions for further reading on the subject of semantical methods in the study of theories of dependent types. Independence results are the subject of (Streicher 1992) and (Hofmann and Streicher 1994). Semantical methods in order to derive syntactic properties of type theories like strong normalisation and thus decidability of type checking have been used in (Hyland and Ong 1993; Altenkirch 1994; Goguen 1995). In (Moggi 1991) and (Harper, Mitchell, and Moggi 1990) categories with attributes are used to give an account of higher-order modules in functional programming. There is an intriguing connection with Paulin's work on program extraction in type theory (Paulin-Mohring 1989) and the "deliverables" approach to program development (Burstall and McKinna 1993). In each of these works a type is modelled as a type or a set together with a predicate or a dependent type and terms are modelled as terms which preserve these predicates. A similar interpretation has been used in (Hofmann 1995a) where a translation of a type theory with a quotient type former into ordinary type theory and other applications of syntactic models are described.

Connections between category-theoretic semantics and abstract machines have been noticed in (Curien 1986) and (Ehrhard 1988) and were subsequently exploited and applied in (Ritter 1992) where an evaluator for the Calculus of Constructions is derived from its category-theoretic semantics.

Last, but not least we mention the use of domain-theoretic interpretations of type theory in order to establish the consistency of general recursion and fixpoint combinators with dependent types (Palmgren and Stoltenberg-Hansen 1990). In a similar direction goes (Reus 1995) where an interpretation of type theory using synthetic domain theory has been employed to establish the consistency of a very powerful dependent type theory incorporating higher-order logic, general recursion, and impredicativity.

## References

Allen, S. (1987). A non-type-theoretic account of Martin-Löf's types. In Symposium on Logic in Computer Science.

Altenkirch, T. (1994). Proving strong normalization of CC by modifying realizability semantics. In H. Barendregt and T. Nipkow (Eds.), Types for Proofs and Programs, Springer LNCS Vol. 806, pp. 3–18.

Altenkirch, T., M. Hofmann, and T. Streicher (1996). Reduction-free normalisation for a polymorphic system. In Proc. of the 11th IEEE Symp. on Logic in Comp. Sci. (LICS), New Brunswick, New Jersey.

Asperti, A. and S. Martini (1992). Categorical models of polymorphism. Information and Computation 99, 1–79.

Barr, M. and C. Wells (1990). Category Theory for Computing Science. International Series in Computer Science. Prentice Hall.

Beeson, M. (1985). Foundations of Constructive Mathematics. Springer.

Burstall, R. and J. McKinna (1993). Deliverables: An approach to program semantics in constructions. In Proc. MFCS '93, Springer LNCS, Vol. 711. Also as LFCS technical report ECS-LFCS-91-133.

Cartmell, J. (1978). Generalized Algebraic Theories and Contextual Categories. Ph. D. thesis, Univ. Oxford.

Constable, R. et al. (1986). Implementing Mathematics with the Nuprl Development System. Prentice-Hall.

Coquand, T. (1990). Metamathematical investigations of a calculus of constructions. In P. Odifreddi (Ed.), Logic and Computer Science, pp. 91–118. Academic Press Ltd.

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

Coquand, T. and C. Paulin-Mohring (1989). Inductively defined types. In LNCS 389. Springer.

Crole, R. (1993). Categories for Types. Cambridge University Press.

Curien, P.-L. (1986). Categorical Combinators, Sequential Algorithms and Functional Programming. Pitman.

Curien, P.-L. (1989). Alpha-conversion, conditions on variables and categorical logic. Studia Logica 3, 318–360.

Curien, P.-L. (1993). Substitution up to isomorphism. Fundamenta Informaticae 19, 51–86.

Dowek, G. et al. (1991, Dec.). The COQ proof assistant user's guide, V5.6. Rapport technique 134, INRIA Rocquencourt.

Dybjer, P. (1996). Internal type theory. In Proc. BRA TYPES workshop, Torino, June 1995, Springer LNCS. To appear.

Ehrhard, T. (1988). Une sémantique catégorique des types dépendants. Application au Calcul des Constructions. Ph. D. thesis, Univ. Paris VII.

Ehrhard, T. (1989). Dictoses. In Proc. Conf. Category Theory and Computer Science, Manchester, UK, pp. 213–223. Springer LNCS Vol. 389.

Goguen, H. (1995). Typed operational semantics. In Proc. TLCA '95, Edinburgh, Springer LNCS Vol. 902.

Harper, R., F. Honsell, and G. Plotkin (1993, January). A framework for defining logics. Journal of the ACM 40(1), 143–184.

Harper, R. and J. C. Mitchell (1993). On the type structure of Standard ML. ACM Trans. Programming Lang. and Systems 15(2), 211–252.

Harper, R., J. C. Mitchell, and E. Moggi (1990). Higher-order modules and the phase distinction. In Conference record of the 17th ACM Symposium on Principles of Programming Languages (POPL), San Francisco, CA USA, pp. 341–354.

Hofmann, M. (1995a). Extensional Concepts in Intensional Type Theory. Ph. D. thesis, Univ. of Edinburgh.

Hofmann, M. (1995b). On the interpretation of type theory in locally cartesian closed categories. In J. Tiuryn and L. Pacholski (Eds.), Proc. CSL '94, Kazimierz, Poland, Springer LNCS, Vol. 933, pp. 427–442.

Hofmann, M. (1996). Conservativity of equality reflection over intensional type theory. In Proc. BRA TYPES workshop, Torino, June 1995, Springer LNCS. To appear.

Hofmann, M. and T. Streicher (1994). A groupoid model refutes uniqueness of identity proofs. In Proceedings of the 9th Symposium on Logic in Computer Science (LICS), Paris.

Huet, G. (1990). A uniform approach to type theory. In Logical Foundations of Functional Programming. Addison-Wesley.

Hyland, J. M. E. and C.-H. L. Ong (1993). Modified realisability toposes and strong normalisation proofs. In J. F. Groote and M. Bezem (Eds.), Typed Lambda Calculi and Applications, Springer LNCS, Vol. 664.

Hyland, M. and A. Pitts (1989). The Theory of Constructions: Categorical Semantics and Topos-Theoretic Models. In Categories in Computer Science and Logic. AMS.

Jacobs, B. (1991). Categorical Type Theory. Ph. D. thesis, University of Nijmegen.

Jacobs, B. (1993). Comprehension categories and the semantics of type theory. Theoretical Computer Science 107, 169–207.

Lamarche, F. (1987). A simple model for the theory of constructions. In J. W. Gray and A. Scedrov (Eds.), Proc. of AMS Research Conf., Boulder, Colorado, pp. 137–199. AMS.

Lambek, J. and P. Scott (1985). Introduction to Higher-Order Categorical Logic. Cambridge University Press.

Luo, Z. (1991). Program specification and data refinement in type theory. In S. Abramsky and T. S. E. Maibaum (Eds.), Proc. TAPSOFT '91, Springer LNCS, Vol. 493, pp. 142–168.

Luo, Z. (1994). Computation and Reasoning. Oxford University Press.

Luo, Z. and R. Pollack (1992). LEGO Proof Development System: User's Manual. Technical Report ECS-LFCS-92-211, University of Edinburgh.

Magnusson, L. and B. Nordström (1994). The ALF proof editor and its proof engine. In H. Barendregt and T. Nipkow (Eds.), Types for Proofs and Programs, Springer LNCS Vol. 806, pp. 213–237. Springer-Verlag.

Martin-Löf, P. (1975). An intuitionistic theory of types: Predicative part. In H. E. Rose and J. C. Sheperdson (Eds.), Logic Colloquium 1973, pp. 73–118. North-Holland.

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

Martin-Löf, P. (1984). Intuitionistic Type Theory. Bibliopolis·Napoli.

Moggi, E. (1991). A category-theoretic account of program modules. Math. Struct. in Comp. Sci. 1(1), 103–139.

Nordström, B., K. Petersson, and J. M. Smith (1990). Programming in Martin-Löf's Type Theory, An Introduction. Clarendon Press, Oxford.

Obtułowicz, A. (1989). Categorical and algebraic aspects of Martin-Löf type theory. Studia Logica 3, 299–317.

Palmgren, E. and V. Stoltenberg-Hansen (1990). Domain interpretation of Martin-Löf's partial type theory. Ann. of Pure and Appl. Logic 48, 135–196.

Paulin-Mohring, C. (1989). Extracting $F_\omega$'s programs from proofs in the calculus of constructions. In Principles of Programming Languages (POPL), pp. 1–17. ACM.

Pitts, A. (1997). Categorical logic. In Handbook of Logic in Computer Science (Vol. VI). Oxford University Press. To appear.

Reus, B. (1995). Program verification in Synthetic Domain Theory. Ph. D. thesis, LMU, München.

Reynolds, J. C. and G. D. Plotkin (1988, May). On functors expressible in the polymorphic typed lambda calculus. Technical Report ECS-LFCS-88-53, University of Edinburgh.

Ritter, E. (1992). Categorical Abstract Machines for Higher-Order Typed Lambda Calculi. Ph. D. thesis, University of Cambridge.

Seely, R. A. G. (1984). Locally cartesian closed categories and type theory. Mathematical Proceedings of the Cambridge Philosophical Society 95, 33–48.

Smith, J. (1988). The independence of Peano's fourth axiom from Martin-Löf's type theory without universes. Journal of Symbolic Logic 53(3).

Streicher, T. (1991). Semantics of Type Theory. Birkhäuser.

Streicher, T. (1992). Dependence and independence results for (impredicative) calculi of dependent types. Math. Struct. Comp. Sci. 2, 29–54.

Streicher, T. (1993). Semantical Investigations into Intensional Type Theory. Habilitationsschrift, LMU München.

Taylor, P. (1986). Recursive Domains, Indexed Category Theory, and Polymorphism. Ph. D. thesis, University of Cambridge.

Troelstra, A. and D. van Dalen (1988). Constructivism in Mathematics, An Introduction, Volume I. North-Holland.