Chapter 0

Prospectus

This introductory chapter is divided into two parts. It first discusses some generalities concerning logic, type theory and category theory, and describes some themes that will be developed in this book. It then continues with a description of the (standard) logic and type theory of ordinary sets, from the perspective of fibred category theory—typical of this book. This description focuses on the fundamental adjunctions that govern the various logical and type theoretic operations.

0.1 Logic, type theory, and fibred category theory

A logic is always a logic over a type theory. This statement sums up our approach to logic and type theory, and forms an appropriate starting point. It describes a type theory as a "theory of sorts", providing a domain of reasoning for a logic. Roughly, types are used to classify values, so that one can distinguish between zero as a natural number $0 : \mathbb{N}$ and zero as a real number $0 : \mathbb{R}$, and between addition $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ on natural numbers and addition $+ : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ on real numbers. In these examples we use atomic types $\mathbb{N}$ and $\mathbb{R}$ and composite types $\mathbb{N} \times \mathbb{N} \to \mathbb{N}$ and $\mathbb{R} \times \mathbb{R} \to \mathbb{R}$ obtained with the type constructors $\times$ for Cartesian product, and $\to$ for exponent (or function space). The relation ':' as in $0 : \mathbb{N}$, is the inhabitation relation of type theory. It expresses that 0 is of type $\mathbb{N}$, *i.e.* that 0 inhabits $\mathbb{N}$. It is like membership $\in$ in set theory, except that $\in$ is untyped, since everything is a set. But a string is something which does not inhabit the type of natural numbers. Hence we

shall have to deal with rules regulating inhabitation, like

$$\frac{}{0:\mathbb{N}} \qquad \text{and} \qquad \frac{n:\mathbb{N}}{\mathsf{succ}(n):\mathbb{N}}$$

The first rule is unconditional: it has no premises and simply expresses that the term 0 inhabits the type $\mathbb{N}$. The second rule tells that if we know that n inhabits $\mathbb{N}$, then we may conclude that $\mathsf{succ}(n)$ also inhabits $\mathbb{N}$, where $\mathsf{succ}(-)$ may be read as successor operation. In this way one can generate terms, like $\mathsf{succ}(\mathsf{succ}(0)):\mathbb{N}$ inhabiting the type $\mathbb{N}$.

In predicate logic one reasons about such terms in a type theory, like in

$$\forall x{:}\mathbb{N}.\, \exists y{:}\mathbb{N}.\, y > \mathsf{succ}(x).$$

This gives an example of a proposition. The fact that this expression is a proposition may also be seen as an inhabitation statement, so we can write

$$(\forall x{:}\mathbb{N}.\, \exists y{:}\mathbb{N}.\, y > \mathsf{succ}(x)) : \mathsf{Prop}$$

using a type Prop of propositions. In this particular proposition there are no free variables, but in predicate logic an arbitrary proposition $\varphi:\mathsf{Prop}$ may contain free variables. These variables range over types, like in:

$$x > 5 : \mathsf{Prop}, \text{ where } x{:}\mathbb{N} \qquad \text{or} \qquad x > 5 : \mathsf{Prop}, \text{ where } x{:}\mathbb{R}.$$

We usually write these free variables in a "context", which is a sequence of variable declarations. In the examples the sequence is a singleton, so we write

$$x{:}\mathbb{N} \vdash x > 5 : \mathsf{Prop} \qquad \text{and} \qquad x{:}\mathbb{R} \vdash x > 5 : \mathsf{Prop}.$$

The turnstile symbol $\vdash$ separates the context from the conclusion: we read the sequent $x{:}\mathbb{N} \vdash x > 5{:}\mathsf{Prop}$ as: in the context where the variable x is of type $\mathbb{N}$, the expression $x > 5$ is a proposition. Well-typedness is of importance, since if x is a string, then the expression $x > 5$ does not make sense (unless one has a different operation $>$ on strings, and one reads '5' as a string).

This explains what we mean with: a logic is always a logic over a type theory. Underlying a logic there is always a calculus of typed terms that one reasons about. But one may ask: what about single-sorted logic (*i.e.* single-typed, or untyped, logic) in which variables are thought of as ranging over a single domain, so that types do not really play a rôle? Then one still has a type theory, albeit a very primitive one with only one type (namely the type of the domain), and no type constructors. In such situations one often omits the (sole) type, since it has no rôle. But formally, it is there. And what about propositional logic? It is included as a border case: it can be seen as a degenerate predicate logic in which all predicates are closed (*i.e.* do not contain term variables), so one can see propositional logic as a logic over the empty type theory.

We distinguish three basic kinds of type theory:

- simple type theory (STT);
- dependent type theory (DTT);
- polymorphic type theory (PTT).

In *simple* type theory there are types built up from atomic types (like $\mathbb{N}$, $\mathbb{R}$ above) using type constructors like exponent $\to$, Cartesian product $\times$ or coproduct (disjoint union) $+$. Term variables $x{:}\sigma$ are used to build up terms, using atomic terms and introduction and elimination operations associated with the type constructors (like tuples and projections for products $\times$). Types in simple type theory may be seen as sets, and (closed) terms inhabiting types as elements of these sets. In *dependent* type theory, one allows a term variable $x{:}\sigma$ to occur in another type $\tau(x)$: Type. This increases the expressive power, for example because one can use in DTT the type $\mathsf{Matrix}(n, m)$ of $n \times m$ matrices (say over some fixed field), for $n{:}\mathbb{N}$ and $m{:}\mathbb{N}$ terms of type $\mathbb{N}$. If one thinks of types as sets, this type dependency is like having for each element $i \in I$ of a set I, another set $X(i)$. One usually writes $X_i = X(i)$ and sees $(X_i)_{i \in I}$ as an I-indexed family of sets. Thus, in dependent type theory one allows type-indexed-types, in analogy with set-indexed-sets. Finally, in *polymorphic* type theory, one may use additional type variables α to build up types. So type variables α may occur inside a type $\sigma(\alpha)$, like in the type $\mathsf{list}(\alpha)$ of lists of type α. This means that one has types, indexed by (or parametrised by) the universe Type of all types. In a set theoretic picture this involves a set $X_A = X(A)$ for each set A. One gets indexed collections $(X_A)_{A \in \mathbf{Sets}}$ of sets X_A.

These three type theories are thus distinguished by different forms of indexing of types: no indexing in simple type theory, indexing by term variables $x{:}\sigma$ in dependent type theory, and indexing by type variables $\alpha{:}\,\mathsf{Type}$ in polymorphic type theory. One can also combine dependent and polymorphic type theory, into more complicated type theories, for example, into what we call polymorphic dependent type theory (PDTT) or full higher order dependent type theory (FhoDTT).

What we have sketched in the beginning of this section is predicate logic over simple type theory. We shall call this simple predicate logic (SPL). An obvious extension is to consider predicate logic over dependent type theory, so that one can reason about terms in a dependent type theory. Another extension is logic over polymorphic type theory. This leads to dependent predicate logic (DPL) and to polymorphic predicate logic (PPL). If one sees a typed calculus as a (rudimentary) programming language, then these logics may be used as program logics to reason about programs written in simple, dependent, or polymorphic type theory. This describes logic as a "module" that one can

plug onto a type theory.

This book focuses on such structural aspects of logic and type theory. The language and techniques of category theory will be essential. For example, we talked about a logic *over* a type theory. Categorically this will correspond to one ("total") category, capturing the logic, being *fibred over* another ("base") category, capturing the type theory. Indeed, we shall make special use of tools from fibred category theory. This is a special part of category theory, stemming from the work of Grothendieck in algebraic geometry, in which (continuous) indexing of categories is studied. As we already mentioned, the various forms of type theoretic indexing distinguish varieties of type theory. And also, putting a logic on top of some type theory (in order to reason about it) will be described by putting a fibration on top of the categorical structure corresponding to the type theory. In this way we can put together complicated structures in a modular way.

Fibred category theory is ordinary category theory with respect to a base category. Also, one can say, it is ordinary category theory *over* a base category. Such a base category is like a universe. For example, several concepts in category theory are defined in terms of sets. One says that a category $\mathbb{C}$ has arbitrary products if for each *set* I and each I-indexed collection $(X_i)_{i\in I}$ of objects $X_i \in \mathbb{C}$ there is a product object $\prod_{i\in I} X_i \in \mathbb{C}$ together with projection morphisms $\pi_j\colon (\prod_{i\in I} X_i) \to X_j$, which are suitably universal. In category theory one is not very happy with this privileged position of sets and so the question arises: is there a way to make sense of such products with respect to an object I of a 'universe' or 'base category' $\mathbb{B}$, more general than the category **Sets** of sets and functions? This kind of generality is needed to interpret logical products $\forall x\colon \sigma.\, \varphi$ or type theoretic products $\Pi x\colon \sigma.\, \tau$ when the domain of quantification σ is not interpreted as a set (but as some ordered set, or algebra, for example).

Another example is local smallness. A category $\mathbb{C}$ is locally small if for each pair of objects $X, Y \in \mathbb{C}$ the morphisms $X \to Y$ in $\mathbb{C}$ form a *set* (as opposed to a proper class). That is, if one has homsets $\mathbb{C}(X, Y) \in \mathbf{Sets}$ as objects in the category of sets. Again the question arises whether there is a way of saying that $\mathbb{C}$ is locally small with respect to an arbitrary universe or base category $\mathbb{B}$ and not just with respect to **Sets**.

Fibred category theory provides answers to such questions. It tells what it means for a category $\mathbb{E}$ to be 'fibred over' a base category $\mathbb{B}$. In that case we write $\begin{smallmatrix}\mathbb{E}\\ \downarrow\\ \mathbb{B}\end{smallmatrix}$, where the arrow $\mathbb{E} \to \mathbb{B}$ is a functor which has a certain property that makes it into a fibration. And in such a situation one can answer the above questions: one can define quantification with respect to objects $I \in \mathbb{B}$ and say when one has appropriate hom-objects $\underline{\mathrm{Hom}}(X, Y) \in \mathbb{B}$ for $X, Y \in \mathbb{E}$.

The ways of doing this will be explained in this book. And for a category $\mathbb{C}$ there is always a 'family fibration' $\begin{smallmatrix}\mathrm{Fam}(\mathbb{C})\\ \downarrow\\ \mathbf{Sets}\end{smallmatrix}$ of set-indexed families in $\mathbb{C}$. The fibred notions of quantification and local smallness, specialised to this family fibration, are the ordinary notions described above. Thus, in the family fibration we have our standard universe (or base category) of sets.

There are many categorical notions arising naturally in logic and type theory (see the list below). And many arguments in category theory can be formulated conveniently using logic and type theory as "internal" language (sometimes called the "Mitchell-Bénabou" language, in the context of topos theory). These fields however, have different origins: category theory arose in the work of Eilenberg and Mac Lane in the 1940s within mathematics, and was in the beginning chiefly used in algebra and topology. Later it found applications in almost all areas of mathematics (and computer science as well, more recently). Type theory is also from this century, but came up earlier in foundational work by Russell in logic (to avoid paradoxes). Recently, type theory has become important in various (notably functional) programming languages, and in computer mathematics: many type theories have been used during the last two decades as a basis for so-called proof-assistants. These are special computer programs which assist in the verification of mathematical statements, expressed in the language of some (typed) logic. The use of types in these areas imposes certain restrictions on what can be expressed, but facilitates the detection of various errors. We think it is in a sense remarkable that two such fundamental fields (of category theory and of type theory)—with their apparent differences and different origins—are so closely related. This close relationship may be beneficial in the use and further development of both these fields.

We shall be especially interested in categorical phenomena arising within logic and type theory. Among these we mention the following.

(i) Every context of variable declarations (in type theory) or of premises (in logic) is an index. It is an index for a 'fibre' category which captures the logic or type theory that takes place within that context—with the declared variables, or under the assumptions. The importance of this categorical rôle of contexts is our motivation for paying more than usual attention to contexts in our formulations of type theory and logic.

(ii) Appropriately typed sequences of terms give rise to morphisms between contexts. This is the canonical way to produce a category from types and terms. These context morphisms induce substitution functors between fibre categories. The structural operations of weakening (adding a dummy assumption) and contraction (replacing two assumptions of the same kind by a single one) appear as special cases of these substitution functors: weakening

is substitution along a projection π, and contraction is substitution along a diagonal δ. These π and δ may be Cartesian projections and diagonals in simple and polymorphic type theories, or 'dependent' projections and diagonals in dependent type theory.

(iii) The basic operations of logic and type theory can be described as adjoints in category theory. Such operations standardly come with an introduction and an elimination operation, which are each other's inverses (via the so-called (β)- and (η)-conversions). Adjoint correspondences capture such situations. This may be familiar for the (simple) type theoretic constructors 1, $\times$, 0, $+$ and $\to$ (and for their propositional counterparts $\top$, $\wedge$, $\bot$, $\vee$ and $\supset$), since these are the operations of bicartesian closed categories (which can be described via standard adjunctions). But also existential $\exists x{:}\sigma.\,(-)$ and universal $\forall x{:}\sigma.\,(-)$ quantification in predicate logic over a type σ, dependent sum $\Sigma x{:}\sigma.\,(-)$ and product $\Pi x{:}\sigma.\,(-)$ in dependent type theory over a type σ, and polymorphic sum $\Sigma\alpha{:}\mathsf{Type}.\,(-)$ and product $\Pi\alpha{:}\mathsf{Type}.\,(-)$ in polymorphic type theory over the universe Type of types, are characterised as left and right adjoints, namely to the weakening functor which adds an extra dummy assumption $x{:}\sigma$, or $\alpha{:}\mathsf{Type}$. Moreover, equality $=_\sigma$ on a type σ is characterised as left adjoint to the contraction functor which replaces two variables $x, y{:}\sigma$ by a single one (by substituting x for y). By 'being characterised' we mean that the standard logical and type-theoretical rules for these operations are (equivalent to) the rules that come out by describing these operations as appropriate adjoints.

The most important adjunctions are:

$$\begin{aligned}
\text{existential } \exists,\ \text{sum } \Sigma &\ \dashv\ \text{weakening}\\
\text{weakening} &\ \dashv\ \text{universal } \forall,\ \text{product } \Pi\\
\text{equality} &\ \dashv\ \text{contraction}\\
\text{truth} &\ \dashv\ \text{comprehension (or 'subsets types')}\\
(\text{but also: equality} &\ \dashv\ \text{comprehension, via a different functor})\\
\text{quotients} &\ \dashv\ \text{equality.}
\end{aligned}$$

The first four of these adjoints were recognised by Lawvere (and the last two are identified in this book). Lawvere first described the quantifiers $\exists, \forall$ as left and right adjoints to arbitrary substitution functors. The above picture with separate adjoints to weakening and to contraction functors is a refinement, since, as we mentioned in (ii), weakening and contraction functors are special cases of substitution functors. (These operations of weakening and contraction can be suitably organised as a certain comonad; we shall define quantification and equality abstractly with respect to such comonads.)

(iv) As we mentioned above, the characteristic aspect of dependent type theory is that types may depend on types, in the sense that term variables inhabiting types may occur in other types. And the characteristic aspect of polymorphic type theory is that type variables may occur in types. Later we shall express this as: types may depend on kinds. These dependencies amount to certain forms of indexing. They are described categorically by fibred (or indexed) categories. Thus, if one knows the dependencies in a type theory, then one knows its underlying categorical structure. The additional type theoretic structure may be described via certain adjunctions, as in the previous point.

(v) Models of logics and type theories are (structure preserving) functors. From a specific system in logic or type theory one can syntactically build a so-called 'classifying' (fibred) category, using a term model—or generalised Lindenbaum-Tarski—construction. A model of this system is then a (fibred) functor with this classifying (fibred) category as domain, preserving appropriate structure. We shall make systematic use of this *functorial semantics*. It was introduced by Lawvere for single-typed simple type theories. And it extends to other logics and type theories, and thus gives a systematic description of models of (often complicated) logics and type theories.

(vi) If $\sigma = \sigma(\alpha)$ is a type (in polymorphic type theory) in which a free type variable α occurs, then, under reasonable assumptions about type formation, the operation $\tau \mapsto \sigma[\tau/\alpha]$ of substituting a type τ for α, is functorial. This functoriality is instrumental in describing the rules of (co-)inductively defined data types in terms of (co-)algebras of this functor. And the reasoning principles (or logic) associated with such data types can also be captured in terms of (co-)algebras (but for a different functor, obtained by lifting the original functor to the logical world of predicates and relations).

(vii) A logical framework is a type theory $\mathcal{T}$ which is expressive enough so that one can formulate other systems $\mathcal{S}$ of logic or of type theory inside $\mathcal{T}$. Categorically one may then describe (the term model of) $\mathcal{S}$ as an internal category in (the term model of) $\mathcal{T}$. We briefly discuss dependent type theory as a logical framework in Section 10.2, but we refer to [87] for this connection with internal categories.

This is not a book properly on logic or on type theory. Many logical and type theoretical calculi are described and some illustrations of their use are given, but there is nothing about specific proof-theoretic properties like cut-elimination, Church-Rosser or strong normalisation. Therefore, see [14]. The emphasis here lies on categorical semantics. This is understood as follows. Category theory provides means to say what a model of, say predicate logic, should look like. It gives a specification, or a hollow structure, which captures

the essentials. A proper model is something else, namely an instance of such a structure. We shall describe both these hollow structures, and some instances of these. (But we do not investigate the local structure or theories of the example models, like for example in [197] or in [13, Chapter 19].)

So what, then, is the advantage of knowing what the categorical structures are, corresponding to certain logics and type theories? Firstly, it enables us to easily and quickly recognise that certain mathematical structures are models of some logical or type theoretical calculus, without having to write out an interpretation in detail. The latter can be given for the 'hollow categorical structure', and need not be repeated for the particular instances. One only has to check that the particular structure is an instance of the general categorical structure. For example, knowing that a particular category (of domains, say) is Cartesian closed yields the information that we can interpret simple type theory. Secondly, once this is realised, we can turn things around, and start using our calculus (suitably incorporating the constants in a signature) to reason directly and conveniently about a (concrete or abstract categorical) model. This is the logician's view of the mathematician's use of language: when reasoning about a particular mathematical structure (say a group G), one formally adds the elements $a \in G$ as constants $\underline{a}$ to the language, and one uses the resulting "internal" language to reason directly about G. The same approach applies to more complex mathematical structures, like a fibred category of domains: one then needs a suitable type theoretic language to reason about such a complex (indexed) structure. The third advantage is that a clear (categorical) semantics provides a certain syntactic hygiene, and deepens the understanding of the various logical and type theoretical systems. For example, the principle that a (possibly new) operation in logic or type theory should correspond to an adjoint gives certain canonical introduction, elimination and conversion rules for the constructor. Fourthly, models can be used to obtain new results about one's logical or type theoretical system. Consistency, conservativity and independence results are often obtained in this manner. Finally, and maybe most importantly, models provide meaning to one's logical or type theoretical language, resulting in a better understanding of the syntax.

There are so many systems of logic and type theory because there are certain "production rules" which generate new systems from given ones.

(i) There are three basic type theories: simple type theory (STT), dependent type theory (DTT) and polymorphic type theory (PTT).

(ii) Given a certain type theory, one can construct a logic over this type theory with predicates $\varphi(\vec{x})\colon \mathsf{Prop}$ containing free variables $\vec{x}$ inhabiting types. This allows us to reason about (terms in) the given type theory.

(iii) Given a logic (over some type theory), one can construct a new type theory (extending the given one) by a propositions-as-types upgrade: one considers the propositions φ in the logic as types in the new type theory, and derivations in the logic as terms in the new type theory.

This modularity is reflected categorically in the following three points.

(i) There are three basic categorical structures: for STT (Cartesian closed categories), for DTT (what we call closed comprehension categories) and for PTT (certain fibred Cartesian closed categories).

(ii) Putting a logic on a type theory corresponds to putting a preorder fibration on top of the structure describing the type theory. For logic one uses preorder structures, since in logic one is interested in provability and not in explicit proofs (or proof-terms, as in type theory), which are described as non-trivial morphisms.

(iii) Under a propositions-as-types upgrade one replaces a preorder fibration by an ordinary fibration (with proper fibre categories), thus making room for proof-terms as proper morphisms.

(Both second points are not as unproblematic as they may seem, because one may have complicated type theories, say with two syntactic universes of types and of kinds, in which there are many ways of putting a logic on top of such a type theory: one may wish to reason about types, or about kinds, or about both in the same logic. Categorically, there are similarly different ways in which a preorder fibration can be imposed.)

By the very nature of its contents, this book is rather descriptive. It contains few theorems with deep mathematical content. The influence of computer science may be felt here, in which much emphasis is put on the description of various languages and formalisms.

Also, it is important to stress that this is not a book properly on fibred category theory. And it is not intended as such. It does contain the basic concepts and results from fibred category theory, but only as far as they are directly useful in logic or type theory (and not in topology, for example). Some of these basic results have not been published previously, but have been folklore for some time already. They have been discovered and rediscovered by various people, and the precise flow of ideas is hard to track in detail. What we present in this book is not a detailed historical account, and we therefore apologise in advance for any misrepresentation of history.

We sketch what we see as the main lines. In the development of fibred category and categorical logic one can distinguish an initial French period starting in the 1960s with Grothendieck's definition of a fibration (*i.e.* a fibred category), published in [107]. It was introduced in order to study descent. The

ensuing theory was further developed by Grothendieck and (among others) Giraud [100] and Bénabou. The latter's work is more logical and foundational in spirit than Grothendieck's (involving for example suitable fibred notions of local smallness and definability), and is thus closest to the current work. Many of the basic notions and results stem from this period.

In the late 1960s Lawvere first applied indexed categories in the study of logic. Especially, he described quantification and equality in terms of adjoints to substitution functors, and showed that also comprehension involves an adjunction. This may be seen as the start of categorical logic (explicitly, in his influential "Perugia Lecture Notes" and also in [192, 193]). At about the same time, the notion of elementary topos was formulated, by Lawvere and Tierney. This resulted in renewed attention for indexed (and internal) categories, to study phenomena over (and inside) toposes. See for example [173, 169] and the references there.

Then, in the 1980s there is the start of a type theoretic boom, in which indexed and fibred categories are used in the semantics of polymorphic and dependent type theories, see the basic papers [306, 307, 148] and the series of PhD theses [45, 330, 75, 185, 318, 252, 260, 7, 154, 89, 217, 86, 60, 289, 125, 4, 198, 133]. This book collects much material from this third phase. Explicitly, the connection between simple type theory and Cartesian closed categories was first established by Lawvere and Lambek. Later, dependent type theory was related to locally Cartesian closed categories by Seely, and to the more general "display map categories" by Taylor. The relation between polymorphic type theory and certain fibred (or indexed, or internal) Cartesian closed categories is due to Seely, Lamarche and Moggi. Finally, more complicated systems combining polymorphic and dependent systems (like the calculus of constructions) were described categorically by Hyland, Pitts, Streicher, Ehrhard, Curien, Pavlović, Jacobs and Dybjer. This led to the (surprising) discovery of complete internal categories by Moggi and Hyland (and to the subsequent development of 'synthetic' domain theory in abstract universes).

Interestingly, fibred categories are becoming more and more important in various other areas of (theoretical) computer science, precisely because the aspects of indexing and substitution (also called renaming, or relabelling) are so fundamental. Among these areas we mention (without pretension to be in any sense complete): database theory [295, 151, 9], rewriting [12], automata theory [175, 10], abstract environments [279], data flow networks [310], constraint programming [219], concurrency theory [345, 131], program analysis [230, 25], abstract domain theory [146] and specification [152, 327, 48, 159].

Many topics in the field of categorical logic and type theory are not discussed in this book. Sometimes because the available material is too recent (and unsettled), sometimes because the topic deviates too much from the main line,

but mostly simply because of lack of space. Among these topics we mention (with a few references): inductively and co-inductively defined types in dependent type theory [70, 71], categorical combinators [63, 290, 116], categorical normalisation proofs [147, 238, 5], fixed points [16], rewriting and 2-categorical structure [308, 278], modal logic [93], μ-calculi [313], synthetic domain theory [144, 331, 264], a fibred Giraud theorem [229], a fibred adjoint functor theorem [47, 246], descent theory [168] (especially with its links to Beth definability [208]), fibrations in bi-categories [315, 317], 2-fibrations [127], and the theory of stacks [100].

The choice has been made to present details of interpretation functions for simple type theory in full detail in Chapter 2, together with the equivalent functorial interpretation. In later chapters interpretations will occur mostly in the more convenient functorial form. For detailed information about interpretation functions in polymorphic and (higher order) dependent type theories we refer to [319, 61]. As we proceed we will be increasingly blurring the distinction between certain type theories and certain fibred categories, thus decreasing the need for explicit interpretations

0.2 The logic and type theory of sets

We shall now try to make the fibred perspective more concrete by describing the (familiar) logic and type theory of ordinary sets in fibred form. Therefore we shall use the fibrations of predicates over sets and of families of sets over sets, without assuming knowledge of what precisely constitutes a fibration. In a well-known situation we thus describe some of the structures that will be investigated in more abstract form in the course of this book. We shall write **Sets** for the category of (small) sets and ordinary functions between them.

Predicates on sets can be organised in a category, that will be called **Pred**, as follows.

objects pairs (I, X) where $X \subseteq I$ is a subset of a set I; in this situation we consider X as a predicate on a type I, and write $X(i)$ for $i \in X$ to emphasise that an element $i \in I$ may be understood as a free variable in X. When I is clear from the context, we sometimes write X for the object $(X \subseteq I)$.

morphisms $(I, X) \to (J, Y)$ are functions $u\colon I \to J$ between the underlying sets satisfying

$$X(i) \text{ implies } Y(u(i)), \quad \text{for each } i \in I.$$

Diagrammatically, this condition on such a function

$u\colon I \to J$ amounts to the existence of a necessarily unique (dashed) map

$$\begin{array}{ccc} X & \dashrightarrow & Y \\ \downarrow & & \downarrow \\ I & \xrightarrow{\;u\;} & J \end{array}$$

indicating that u restricts appropriately.

There is an obvious forgetful functor $\mathbf{Pred} \to \mathbf{Sets}$ sending a predicate to its underlying set (or type): $(I, X) \mapsto I$. This functor is a "fibration". And although it plays a crucial rôle in this situation, we do not give it a name, but simply write it vertically as $\begin{smallmatrix}\mathbf{Pred}\\ \downarrow \\ \mathbf{Sets}\end{smallmatrix}$ to emphasise that it describes predicates as living over sets.

For a specific set I, the "fibre" category $\mathbf{Pred}_I$ is defined as the subcategory of $\mathbf{Pred}$ of predicates $(X \subseteq I)$ on I and of morphisms that are mapped to the identity function on I. This category $\mathbf{Pred}_I$ may be identified with the poset category $\langle P(I), \subseteq \rangle$ of subsets of I, ordered by inclusion. For a function $u\colon I \to J$ there is "substitution" functor $u^*\colon P(J) \to P(I)$ in the reverse direction, by

$$\big(Y \subseteq J\big) \longmapsto \big(\{i \mid u(i) \in Y\} \subseteq I\big).$$

Clearly we have $Y \subseteq Y' \;\Rightarrow\; u^*(Y) \subseteq u^*(Y')$, so that u^* is indeed a functor. Two special cases of substitution are weakening and contraction. Weakening is substitution along a Cartesian projection $\pi\colon I \times J \to I$. It consists of a functor

$$P(I) \xrightarrow{\pi^*} P(I \times J) \qquad \text{sending} \qquad X \mapsto \{(i,j) \mid i \in X \text{ and } j \in J\}$$

by adding a dummy variable $j \in J$ to a predicate X. Contraction is substitution along a Cartesian diagonal $\delta\colon I \to I \times I$. It is a functor

$$P(I \times I) \xrightarrow{\delta^*} P(I) \qquad \text{given by} \qquad Y \mapsto \{i \in I \mid (i,i) \in Y\}.$$

It replaces two variables of type I by a single variable.

Each fibre category $P(I)$ is a Boolean algebra, with the usual set theoretic operations of intersection $\cap$, top element $(I \subseteq I)$, union $\cup$, bottom element $(\emptyset \subseteq I)$, and complement $I \backslash (-)$. These operations correspond to the propositional connectives $\wedge, \top, \vee, \bot, \neg$ in (Boolean) logic. They are preserved by substitution functors u^* between fibre categories.

The categorical description of the quantifiers $\exists, \forall$ is less standard (than the propositional structure of subsets). These quantifiers are given by operations between the fibres—and not inside the fibres, like the propositional

connectives—since they bind free variables in predicates (and thus change the underlying types). They turn out to be adjoints to weakening, as expressed by the fundamental formula:

$$\exists \dashv \pi^* \dashv \forall.$$

In more detail, we define for a predicate $Y \subseteq I \times J$,

$$\begin{aligned} \exists(Y) &= \{i \in I \mid \exists j \in J.\, (i,j) \in Y\} \\ \forall(Y) &= \{i \in I \mid \forall j \in J.\, (i,j) \in Y\}. \end{aligned}$$

These assignments $Y \mapsto \exists(Y)$ and $Y \mapsto \forall(Y)$ are functorial $P(I \times J) \rightrightarrows P(I)$. And they are left and right adjoints to the above weakening functor $\pi^*\colon P(I) \to P(I \times J)$ because there are the following basic adjoint correspondences.

$$\frac{Y \subseteq \pi^*(X) \quad \text{over } I \times J}{\exists(Y) \subseteq X \quad \text{over } I} \qquad \text{and} \qquad \frac{\pi^*(X) \subseteq Y \quad \text{over } I \times J}{X \subseteq \forall(Y) \quad \text{over } I}$$

(Where the double line means: if and only if.)

For a set (or type) I, equality $i = i'$ for elements $i, i' \in I$ forms a predicate on $I \times I$. Such equality can also be captured categorically, namely as left adjoint to the contraction functor $\delta^*\colon P(I \times I) \to P(I)$. One defines for a predicate $X \subseteq I$ the predicate $\mathrm{Eq}(X)$ on $I \times I$ by

$$\mathrm{Eq}(X) = \{(i,i') \in I \times I \mid i = i' \text{ and } i \in X\}.$$

Then there are adjoint correspondences

$$\frac{\mathrm{Eq}(X) \subseteq Y \quad \text{over } I \times I}{X \subseteq \delta^*(Y) \quad \text{over } I}$$

Notice that the predicate $\mathrm{Eq}(X)$ is equality on I for the special case where X is the top element I. See also Exercise 0.2.2 below for a description of a right adjoint to contraction, in terms of inequality.

The operations of predicate logic can thus be identified as certain structure in this fibration $\begin{smallmatrix}\mathbf{Pred}\\ \downarrow \\ \mathbf{Sets}\end{smallmatrix}$, namely as structure in and between its fibres. Moreover, it is a property of the fibration that this logical structure exists, since it can be characterised in a universal way—via adjoints—and is thus given uniquely up-to-isomorphism. The same holds for the other logical and type theoretical operations that we identify below.

Comprehension is the assignment of a set to a predicate, or, as we shall say more generally later on, of a type to a predicate. This assignment takes a predicate to the set of elements for which the predicate holds. It also has a universal property. Therefore we first need the "truth" functor $1\colon \mathbf{Sets} \to$

Pred, which assigns to a set I the truth predicate $1(I) = (I \subseteq I)$ on I; it is the terminal object in the fibre over I. Comprehension (or subset types, as we shall also say) is then given by a functor $\{-\}\colon \mathbf{Pred} \to \mathbf{Sets}$, namely

$$\{(Y \subseteq J)\} = \{j \in J \mid Y(j)\} = Y.$$

Hence $\{-\}\colon \mathbf{Pred} \to \mathbf{Sets}$ is simply $(Y \subseteq J) \mapsto Y$. It is right adjoint to the truth functor $1\colon \mathbf{Sets} \to \mathbf{Pred}$ since there is a bijective correspondence between functions u and v in a situation:

$$\frac{1(I) \xrightarrow{u} (Y \subseteq J) \quad \text{in } \mathbf{Pred}}{I \xrightarrow[v]{} \{(Y \subseteq J)\} \quad \text{in } \mathbf{Sets}}$$

In essence this correspondence tells us that $Y(j)$ holds if and only if $j \in \{(Y \subseteq J)\}$.

Quotient sets can also be described using the fibration of predicates over sets. We first form the category **Rel** of (binary) relations on sets by pullback:

$$\begin{array}{ccc} \mathbf{Rel} & \longrightarrow & \mathbf{Pred} \\ \downarrow & \lrcorner & \downarrow \\ \mathbf{Sets} & \xrightarrow[I \mapsto I \times I]{} & \mathbf{Sets} \end{array}$$

Via this pullback we restrict ourselves to predicates with underlying sets of the form $I \times I$. Explicitly, the category **Rel** has

objects	pairs (I, R) where $R \subseteq I \times I$ is a (binary) relation on $I \in \mathbf{Sets}$.
morphisms	$(I, R) \to (J, S)$ are functions $u\colon I \to J$ between the underlying sets with the property $$R(i, i') \text{ implies } S(u(i), u(i')), \quad \text{for all } i, i' \in I.$$

The functor $\mathbf{Rel} \to \mathbf{Sets}$ in the diagram is then $(I, R) \mapsto I$. It will turn out to be a fibration by construction. The abovementioned equality predicate yields an equality functor $\mathrm{Eq}\colon \mathbf{Sets} \to \mathbf{Rel}$, namely

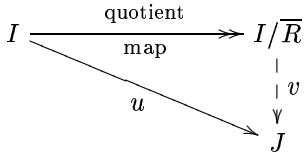
$$J \mapsto \mathrm{Eq}(J) = \{(j, j) \mid j \in J\}.$$

Quotients in set theory can then be described in terms of a left adjoint $\mathcal{Q}$ to this equality functor Eq: a relation $R \subseteq I \times I$ is mapped to the quotient set $I/\overline{R}$, where $\overline{R} \subseteq I \times I$ is the least equivalence relation containing R. Indeed

there is an adjoint correspondence between functions v and u in:

$$\frac{\mathcal{Q}(I,R) = I/\overline{R} \xrightarrow{\ v\ } J \quad \text{in } \mathbf{Sets}}{R \xrightarrow[u]{} \mathrm{Eq}(J) \quad \text{in } \mathbf{Rel}}$$

This correspondence can be reformulated as: for each functon $u\colon I \to J$ with $u(i) = u(i')$ for all $i, i' \in I$ for which $R(i,i')$ holds, there is a unique function $v\colon I/\overline{R} \dashrightarrow J$ in a commuting triangle



Finally we mention that predicates over sets give us higher order logic. There is a distinguished set $2 = \{0,1\}$ of propositions, with special predicate $(\{1\} \subseteq 2)$ for truth: for every predicate $(X \subseteq I)$ on a set I, there is a unique function $\mathrm{char}(X \subseteq I)\colon I \to 2$ with

$$(X \subseteq I) = \mathrm{char}(X \subseteq I)^*(\{1\} \subseteq 2).$$

This existence of "characteristic morphisms" is what makes the category of sets a topos. It allows us to quantify via this set 2 over propositions.

This completes our first glance at the fibred structure of the logic of sets. In the remainder of this section we sketch some of the type theoretic structure of sets, again in terms of a fibration, namely in terms of the "family" fibration $\begin{smallmatrix}\mathrm{Fam}(\mathbf{Sets})\\ \downarrow\\ \mathbf{Sets}\end{smallmatrix}$ of set-indexed-sets. It captures the dependent type theory (with type-indexed-types) of sets.

The category $\mathrm{Fam}(\mathbf{Sets})$ of families of sets has

objects pairs (I, X) consisting of an index set I and a family $X = (X_i)_{i \in I}$ of I-indexed sets X_i.

morphisms $(I,X) \to (J,Y)$ are pairs (u,f) consisting of functions

$$I \xrightarrow{\ u\ } J \quad \text{and} \quad f = \Big(X_i \xrightarrow{\ f_i\ } Y_{u(i)}\Big)_{i \in I}$$

There is a projection functor $\mathrm{Fam}(\mathbf{Sets}) \to \mathbf{Sets}$ sending an indexed family to its underlying set index set: $(I,X) \mapsto I$. It will turn out to be a fibration. Essentially this will mean that there are (appropriate) substitution or reindexing functors: for a function $u\colon I \to J$ between index sets, we can map a

family $Y = (Y_j)_{j \in J}$ over J to a family over I via:

$$(Y_j)_{j \in J} \longmapsto (Y_{u(i)})_{i \in I}.$$

We shall write u^* for this operation. It extends to a functor between "fibre" categories: for an arbitrary set K, let $\mathrm{Fam}(\mathbf{Sets})_K$ be the "fibre" subcategory of $\mathrm{Fam}(\mathbf{Sets})$ of those families (K, X) with K as index set, and with morphisms (id_K, f) with the identity on K as underlying function. Then $u: I \to J$ yields a substitution functor $u^*: \mathrm{Fam}(\mathbf{Sets})_J \to \mathrm{Fam}(\mathbf{Sets})_I$.

Notice that there is an inclusion functor $\mathbf{Pred} \hookrightarrow \mathrm{Fam}(\mathbf{Sets})$ of predicates into families, since every predicate $(X \subseteq I)$ yields an I-indexed family $(X_i)_{i \in I}$ with

$$X_i = \begin{cases} \{*\} & \text{if } i \in X \\ \emptyset & \text{otherwise.} \end{cases}$$

It is not hard to see that this yields a full and faithful functor $\mathbf{Pred} \hookrightarrow \mathrm{Fam}(\mathbf{Sets})$, which commutes with substitution. It is a 'morphism of fibrations'.

Our aim is to describe the dependent coproduct $\coprod$ and product $\prod$ of families of sets as adjoints to weakening functors, in analogy with the situation for existential $\exists$ and universal $\forall$ quantification in the logic of sets. But in this situation of families of sets we have weakening functors π^* induced not by Cartesian projections $\pi: I \times J \to I$, but by "dependent" projections $\pi: \{I \mid X\} \to I$, with domain $\{I \mid X\}$ given by the disjoint union:

$$\{I \mid X\} = \{(i, x) \mid i \in I \text{ and } x \in X_i\}$$

which generalises the Cartesian product. The weakening functor π^* associated with this dependent projection $\pi: \{I \mid X\} \to I$ sends a family $Y = (Y_i)_{i \in I}$ over I to a family $\pi^*(Y)$ over $\{I \mid X\}$ by vacuously adding an extra index x, as in:

$$\pi^*(Y) = (Y_i)_{(i \in I, x \in X_i)}.$$

(As we shall see later, the projection $\pi: \{I \mid X\} \to I$ arises in a canonical way, since the assignment $(I, X) \mapsto \{I \mid X\}$ yields a functor $\mathrm{Fam}(\mathbf{Sets}) \to \mathbf{Sets}$, which is right adjoint to the terminal object functor $1: \mathbf{Sets} \to \mathrm{Fam}(\mathbf{Sets})$, sending a set J to the J-indexed collection $(\{*\})_{j \in J}$ of singletons. The counit of this adjunction has the projection π as underlying map. Thus, the operation $(I, X) \mapsto \{I \mid X\}$ is like comprehension for predicates, as described above.)

The claim is that the dependent coproduct $\coprod$ and product $\prod$ for set-indexed sets are left and right adjoints to the weakening functor π^*. Therefore we have

to define coproduct $\coprod$ and product $\prod$ as functors

$$\mathrm{Fam}(\mathbf{Sets})_{\{I \mid X\}} \underset{\prod}{\overset{\coprod}{\leftarrow \pi^* \rightarrow}} \mathrm{Fam}(\mathbf{Sets})_I$$

$$\{I \mid X\} \xrightarrow[\pi]{} I$$

acting on an $\{I \mid X\}$-indexed family $Z = (Z_{(i,x)})_{i \in I, x \in X_i}$ and producing an I-indexed family. These functors are given by

$$\begin{aligned} \textstyle\coprod(Z)_i &= \{(x, z) \mid x \in X_i \text{ and } z \in Z_{(i,x)}\} \\ \textstyle\prod(Z)_i &= \{\varphi \colon X_i \to \textstyle\bigcup_{x \in X_i} Z_{(i,x)} \mid \forall x \in X_i . \varphi(x) \in Z_{(i,x)}\}. \end{aligned}$$

We then get the fundamental relation

$$\coprod \dashv \pi^* \dashv \prod$$

since there are bijective adjoint correspondences between families of functions f and g in:

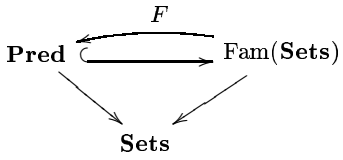
$$\frac{Z \xrightarrow{f} \pi^*(Y) \quad \text{over } \{I \mid X\}}{\coprod(Z) \xrightarrow[g]{} Y \quad \text{over } I} \qquad \text{and} \qquad \frac{\pi^*(Y) \xrightarrow{f} Z \quad \text{over } \{I \mid X\}}{Y \xrightarrow[g]{} \prod(Z) \quad \text{over } I}$$

Also in this situation, there are adjoints to contraction functors δ^* (induced by *dependent* diagonals), given by equality and inequality. But we do not further pursue this matter, and conclude our introduction at this point. What we have sketched is that families of sets behave like dependent types, and that subsets behave like predicates, yielding a logic over (dependent) type theory. We have shown that the basic operations of this logic and of this type theory can be described by adjunctions, in a fibred setting. In the course of this book we shall (among many other things) be more precise about what it means to have such a logic over a type theory and we shall axiomatise all of the structure found above, and identify it in many other situations.

Finally, the next few exercises may help the reader to become more familiar with the structure described above.

Exercises

0.2.1. Define a left adjoint $F\colon \mathrm{Fam}(\mathbf{Sets}) \to \mathbf{Pred}$ to the inclusion functor



such that: (1) F makes the triangle commute (so it does not change the index set), and (2) F commutes with substitution.

0.2.2. Define for a subset $X \subseteq I$ the relation $\mathrm{nEq}(X) \subseteq I \times I$ by

$$\mathrm{nEq}(X) = \{(i, i') \mid i \neq i' \text{ or } i \in X\}$$

and show that the assignment $X \mapsto \mathrm{nEq}(X)$ is right adjoint to contraction $\delta^*\colon P(I \times I) \to P(I)$. Notice that $\mathrm{nEq}(X)$ at the bottom element $X = \emptyset$ is inequality on I.

0.2.3. Show that the equality functor $\mathrm{Eq}\colon \mathbf{Sets} \to \mathbf{Rel}$ also has a right adjoint.

0.2.4. Check that the operation $(I, X) \mapsto \{I \mid X\}$ yields a functor $\mathrm{Fam}(\mathbf{Sets}) \to \mathbf{Sets}$, and show that it is right adjoint to the terminal object functor $\mathbf{Sets} \to \mathrm{Fam}(\mathbf{Sets})$, mapping a set J to the family of singletons $(\{*\})_{j \in J}$. Describe the unit and counit of the adjunction explicitly.