CONTEXT, JUDGEMENT, DEDUCTION

GRETA CORAGLIA AND IVAN DI LIBERTI

Abstract. We introduce judgemental theories and their calculus as a general framework to present and study deductive systems. As an exemplification of their expressivity, we approach dependent type theory and natural deduction as special kinds of judgemental theories. Our analysis sheds light on both the topics, providing a new point of view. In the case of type theory, we provide an abstract definition of extensional type constructor featuring the usual formation, introduction, elimination and computation rules. For natural deduction we offer a deep analysis of structural rules, putting them into context. We finish the paper discussing the internal logic of a topos, a predicative topos, an elementary 2-topos et similia, and show how these can be organized in judgemental theories.

Keywords. Categorical logic, deductive systems, dependent type theory, natural deduction, topos, 2-categories.

MSC2020. 18A15; 18N45; 03F03; 03B38; 03G30.

Contents

Introduction 1
1. Judgemental theories 6
2. Judgement calculi 12
3. Plain dependent type theory 21
4. First-order logic 40
5. Ceci n’est pas un topos 51
6. Future developments 58
References 59

Everything that can be thought at all can be thought clearly.
Everything that can be said can be said clearly.

[Wir22, 4.116]

Introduction

General discussion. As the title advertises, this paper is concerned with the notions of context, judgement and deduction. These three notions belong to Logic, but different communities with different backgrounds and cultures have quite different perspectives on them. The purpose of this work is to present a mathematical and unified approach that accommodates these diverse takes on the topic of deduction.
In order to show how tightly this new framework captures the motions of deductive systems, we develop two applications of the theory we introduce, and since our choice is intended to pay tribute to two major cultures concerned with the topic, we look at examples from type theory \[MS84\] and from proof theory \[TS00\]. They each stand on different conceptual grounds, as it is exemplified by the two following rules.

\[
\text{(DTy)} \quad \frac{\Gamma \vdash a : A \quad \Gamma, A \vdash B \text{ Type}}{\Gamma \vdash B[a] \text{ Type}} \quad \text{(Cut)} \quad \frac{x; \Gamma \vdash \phi \quad x; \Gamma, \phi \vdash \psi}{x; \Gamma \vdash \psi}
\]

Despite their incredibly similar look, and the somehow parallel development of the theories in the same notational framework, there are some philosophical differences between the interpretation of the symbols above.

TT) In type theories, especially those inspired by the reflections of Martin-Löf, \(\Gamma \vdash B \text{ Type}\) is intuitively seen as a judgement. A judgement is an act of knowledge \[Mar96a, Mar87\] bound to a context (\(\Gamma\)) and pertinent to an object (\(B\)). For example, \(\Gamma \vdash B \text{ Type}\) could be read Given \(\Gamma\), \(B\) is a type. The ontological status of a context and an object is, in principle, very different. Also, and most notably, judgements can be of different kinds, claiming all sorts of possible things about their objects \[Mar96a\].

ND) In natural deduction, \(x; \Gamma \vdash \psi\) is intuitively seen as a consecution. A consecution is a relation between structured formulae \((x; \Gamma)\) and formulae \(\psi\) \[Kle67\]. For example, the sequent \(x; \Gamma \vdash \psi\) could be read The multiset of formulae \(\Gamma\), in the variables \(x\), entails \(\psi\). Besides the fact that structured formulae are multisets of formulae, there isn’t an ontological difference between the glyphs appearing on the left and right side of the entailment.

These differences, though admittedly subtle and not that easy to detect on a technical level, dictate a part of the experts’ intuition on the topics (see Section 2.1). Of course, one could argue that these different points of view are mostly philosophical, that the oversimplification commanded by the length of this introduction stresses on them in a somewhat artificial way, and that some variations are allowed, for example \[NvP08\] adopts what we would call a more type theoretic perspective on proof theory, and indeed it is always possible to adopt a judgemental perspective on consecutions. In particular, the deep connection between proof theory and type theory has of course been studied for a while, and its development falls under the paradigm that is mostly known as propositions-as-types \[Wad15\], and how to translate intuitionistic natural deduction into the language of types is beautifully described in \[Mar96b\]. This work aims at providing a new, perhaps more semantic, argument in the same unifying direction.

Rebooting some ideas from \[Jac99\], we conciliate the differences in a unified categorical framework that can highlight and clarify in a more precise way the meaning of all these apparently specific phenomena. Going back to the example of (DTy) and (Cut), we intuitively see how they both fit the same paradigm, in the sense that we could read both as instances of the following syntactic string of symbols

\[
(\Delta) \quad \bigtriangledown \vdash \blacksquare \quad \blacksquare \vdash \bigtriangledown
\]

which we usually parse as: by \(\triangle\), given \(\bigtriangledown \vdash \blacksquare\) and \(\blacksquare \vdash \bigtriangledown\) we deduce \(\bigtriangledown \vdash \blacksquare\). Our theory allows for a coherent expression of all such strings of symbols, and shows how a suitable choice of context either produces (DTy) or (Cut). As a
necessary biproduct of our effort, we get a theory that has both the advantage
of being very versatile, spanning much farther than dependent types and natural
deduction, and computationally meaningful in the sense that it has a built-in notion
of computation.

Our contribution. We introduce the notion of judgemental theory using the lan-
guage of category theory. Judgemental theories are philosophically inspired by
Martin-Löf’s reflections on the topic of judgement [Mar87, Mar96a], and techni-
cally grounded on recent developments in the categorical treatment of dependent
type theory [Awo18, Uem23]. We believe that our perspective is also very attuned
to type refinement systems as described in [MZ15].

\[ \Gamma \vdash H \quad \lambda H \vdash F \quad F \]

Usually, when looking at the premises of a rule, we are confronted with a list of
(nested) judgements as above, which then are transformed into another judgement
by the rule. The technical advantage of our notion is to allow natively for nested
judgements. That is, for us a nested family of judgements is actually a whole
judgement per se:

\[ \Gamma \vdash H.\lambda F \quad H.\lambda F \]

This flexibility allows for an algebraic treatment of extensional type constructors
(and of connectives), in a fashion that is somewhat inspired by Awodey’s natural
models. Judgment classifiers will be categories living over contexts, and functors
between them will regulate deduction rules. In the example below, which is the
formation rule for the Π-constructor in dependent type theory, the category \( U.\Delta U \)
classifies the nested judgement in the premise of the rule (on the right).

\[ \begin{array}{ccc}
\text{ctx} & \Pi & U \\
\downarrow & \downarrow & \downarrow \\
U.\Delta U & \rightarrow & U \\
\end{array} \]

\( \text{(IF)} \)

\[ \Gamma \vdash A \text{ Type} \quad \Gamma.A \vdash B \text{ Type} \]

\[ \Gamma \vdash \Pi A B \text{ Type} \]

We expand the original approach à la Jacobs, where some of these ideas were
evidently hinted at both in the treatment of propositional logic [Jac99, Chapter 2],
and in the treatment of type theories [Jac99, Chapter 10]. It also expands Awodey’s
natural models [Awo18], taking very seriously his algebraic presentation of some
constructors: see for example the discussion at page 9 and later Prop. 2.4 in loc.
cit.

It should be noted that Logical Frameworks [HHP93] of Plotkin et al. have a
similar purpose, and their system is based on AII-calculus. Logical Frameworks do
rely on the notion of judgement in a substantial way, as we do, but their approach
is somewhat much more syntactic. More recently [Uem23] has provided recipes to
transform Logical Frameworks into more categorical gadgets, based on a generaliza-
tion of Awodey’s natural models. One of the advantages of our approach is to avoid
the complexity of Logical Frameworks (and thus of Uemura’s recipe), substituting
it with a native categorical language.

In the next subsection we will discuss in detail all the achievements of this
structure on a technical level, though we end this very qualitative discussion with
a bird’s-eye view on a list of advantages of our system.

1. We provide an algebraic approach to the notion of rule, which is also suitable
   for an analysis of the proof theory associated to a deductive system.
In the case of type theory, this provides a clear definition of extensional type constructor, which was actually not available before, if not in a case-by-case form. We put into perspective the usual paradigm of rules (formation, introduction, elimination, $\beta$- and $\eta$-computation).

In a similar spirit, we provide an in depth analysis of what constitutes what in proof theory are called structural rules. We here see that branches in proof trees are cones in our framework.

We introduce the notion of policy for a judgemental theory, inspired by the classical Cut of the Gentzen calculus. Surprisingly, type dependency in dependent type theory is precisely a type theoretic form of Cut.

Our proofs are computationally meaningful. In a sense, this is due to the structural rigidity and the algebraicity of the framework. Each of our proofs needs to be as atomized and as transparent as possible, this will be particularly evident in our analysis of the proof theory generated by a dependent type theory with $\Pi$-types. The whole framework feels like a categorical proof assistant when doing proofs.

While this introduction seems to focus mainly on dependent type theories and natural deduction, the reader will notice that we have only chosen these two specific frameworks as an exemplum of the expressive power of this theory. Indeed we could have covered modal logic, infinitary logics and much more.

Structure and main results.

Section 1. After the definition of (pre)judgemental theory (Definition 1.0.1), we introduce judgemental theories (Definition 1.0.4), these are the mathematical gadgets which the whole paper is built on. In a nutshell, judgemental theories are pre-judgemental theories closed under a family of categorical constructions which will modulate the deductive power of our logical systems.

Section 2. Each judgemental theory has an associated judgement calculus, which is a graphical bookkeeping of the categorical properties of the judgemental theory. In Section 2.1 we go through a critical analysis of the calculi of a dependent type theory and of natural deduction, both to highlight their main features and to have an inspirational account on what a calculus should look like. Then, in Section 2.2, Section 2.3 and Section 2.4 we declare the dictionary to translate a judgemental theory into its judgement calculus.

Section 3. In this section we show how to recover dependent type theories in our framework. After a definition of (judgemental) dependent type theory (Definition 3.0.1), we discuss its relation with natural models à la Awodey (Theorem 3.1.2) and with comprehension categories à la Jacobs (Construction 3.2.1). The rest of the section is dedicated to showing that the judgement calculus of a dependent type theory recovers the usual calculus of a dependent type theory (dtt). In Section 3.3 we declare a dictionary to convert our notation in the standard notation of dtts. In Section 3.4 we recover context extension and type dependency, in Section 3.5 $\Pi$-types, in Section 3.6 $\text{Id}$-types. One can look at these subsections as a translation of the main results of [Awo18] in our language. Yet, our proofs are much more synthetic and computationally meaningful, as they need to be incredibly atomized. Section 3.7 introduces a general notion of extensional type constructor, featuring
the usual formation, introduction, elimination and computation rules. On a technical level, this is one of the most significant contributions of the paper, and of course the subsections 3.5 and 3.6, could be a corollary of this subsection. To clarify this, we recover unit types and Σ-types as a corollary of Section 3.7. Because a judgement calculus is essentially only a representation of the judgemental theory, our technology is much more than a model of dependent type theory, or a categorical semantics for calculi, it is intrinsically a very syntactic object.

Section 4. We then provide the correct judgemental infrastructure to sustain natural deduction for first order logic. Our exposition follows that in Section 3, meaning that we start from a (pre)judgemental theory (Definition 4.0.3), for readability reasons we pin-point a dictionary (Section 4.1), and discuss which rules it generates, depending on the axioms we put on the judgemental system. In Remark 4.0.5 and Remark 4.0.7 we translate logical structures which are traditionally coded via doctrines into the fibrational setting, meaning the treatment of connectives and that of weakening, respectively. In Section 4.3 we provide evidence for structural rules, then for connectives in Section 4.4. We add quantifiers in Definition 4.4.1 and show that we have rules regulating them in Section 4.6. One of our novelties emerges in Section 4.2, in Definition 4.3.1, where we give a glimpse of the introduction of monads as modalities in the context of judgemental theories, and in the treatment of the cut rule (see Section 4.7). The correspondence between proof trees and (co)cones in the theory is very telling (see Remark 4.3.3).

Section 5. This section frames the internal logic of a topos-like category in the language of dependent type theories. Our notion is perfectly suited to present the Mitchell-Bénabou language of a topos (Section 5.1). In Section 5.2 we introduce a notion of predicative (elementary) topos and show that it supports an essentially identically expressive internal logic, encoded by a dependent type theory. All infinitary pretopoi that we are aware of fall into our assumptions. Section 5.3 brings the previous discussion to the internal logic of an elementary 2-topos in the sense of Weber, and shows that its internal logic can also be organized via a dependent type theory. Our treatment perfects that of Weber handling size issues in a more precise way.

Acknowledgements. The authors are especially grateful to Nathanael Arkor, Jacopo Emmenegger and Francesco Dagnino for their comments and for their guidance through the literature. We are indebted to Pino Rosolini, Milly Maietti and Mike Shulman for inspiring discussions. Both authors are grateful to the anonymous referee for their comments, which improved the presentation of the paper.

For part of this work the first author was supported by the BRIO “Bias, Risk and Opacity in AI” PRIN project (n.2020SSKZ7R) and by the Departments of Excellence 2023-2027 initiative, awarded by the Italian Ministry of Education, Universities and Research (MIUR). Additionally, the first author would like to thank the University of Genova’s PhD programme for supporting them during a large part of this project. The second author was supported by the Swedish Research Council (SRC, Vetenskapsrådet) under Grant No. 2019-04545. The research has received funding from Knut and Alice Wallenbergs Foundation through the Foundation’s program for mathematics.
1. Judgemental theories

Definition 1.0.1 (Pre-judgemental theory). A pre-judgemental theory \((\text{ctx}, J, R, P)\) of \((\text{contexts}, \text{judgements}, \text{rules}, \text{policies})\) is specified by the following data:

1. \((\text{ctx})\) a category (with terminal object \(\odot\));
2. \((J)\) a set of functors \(f : F \rightarrow \text{ctx}\) over the category of contexts;
3. \((R)\) a set of functors \(\lambda : F \rightarrow G\);
4. \((P)\) a set of 2-dimensional cells filling (some) triangles induced by the rules (functors in \(R\)) and the judgements (functors in \(J\)), as in the diagrams below.

\[
\begin{array}{ccc}
\text{ctx} & \overset{\lambda}{\rightarrow} & G \\
\downarrow & & \downarrow \\
\text{ctx} & \overset{\lambda}{\rightarrow} & G
\end{array}
\]

Notation 1.0.2. Let us introduce a bit of terminology:

- contexts \(\Gamma, \Theta\) are objects of \(\text{ctx}\), morphisms \(\sigma : \Theta \rightarrow \Gamma\) are substitutions;
- the element \(g : G \rightarrow \text{ctx}\) of \(J\) is the classifier of the judgement \(G\). We will often blur the distinction between the classifier and its judgement. In general we use letters such as \(F, G, H\);
- objects \(G\) in \(\mathcal{G}\) are usually named after corresponding letter;
- a rule \(\lambda\) is an element of \(R\);
- a policy \(\lambda\) is an element of \(P\);

and for special judgemental theories that happen to have an established notation we declare a switch of notation in the appropriate section.

This is all the syntactic data needed to describe deduction: judgement classifiers prescribe the status of objects with respect to contexts; rules transform objects into other objects, with the context changing accordingly; and policies allow for the possibility that the context of the premise of the rule and that of the consequent are somehow naturally related, either covariantly (i.e. \(\lambda : f \Rightarrow g\lambda\)), contravariantly (i.e. \(\lambda : g\lambda \Rightarrow f\)), or constantly (i.e. when the triangle is strictly commutative) with respect to the direction of the rule. A bird’s eye view of this first definition and what it might have been can be found in Section 6.

Example 1.0.3 (Toy Martin-Löf type theory). In order to get acquainted with the definition, let us introduce the categorical syntax to present a toy type theory. Consider a category \(\text{ctx}\) of contexts and substitutions, \(U\) a category (universe) of types and \(\mathcal{U}\) a category (universe) of terms. For simplicity, we imagine that a term is always registered together with its type, so that objects of \(\mathcal{U}\) are of the form \((a, A)\) with \(A\) an object in \(U\). Define the pre-judgemental theory having \(J = \{u, ˙u\}\), \(R = \{\Sigma\}\), \(P = \{\text{Id} : u \circ \Sigma \Rightarrow ˙u\}\) as below.

\[
\begin{array}{ccc}
\text{ctx} & \overset{\Sigma}{\rightarrow} & \mathcal{U} \\
\downarrow & & \downarrow \\
\text{ctx} & \overset{\text{Id}}{\leftarrow} & \mathcal{U}
\end{array}
\]

Intuitively, \(\hat{u}\) classifies terms with their context, \(u\) does the same for types, \(\Sigma\) performs typing, meaning it is the second projection, and \(\text{Id}\) shows that such an operation preserves the context.
We know that this all looks very unorthodox. We will use this toy example to get a first small impression on how judgemental theories work, see 2.3.5, 2.4.1, and above all Section 3.

On the data expressed by a pre-judgemental theory we wish to impress some deductive power. This is achieved using some 2-categorical constructions and properties.

**Definition 1.0.4** (Judgemental theory). A judgemental theory $(\text{ctx}, J, R, P)$ is a pre-judgemental theory such that

1. $R$ and $P$ are closed under composition;
2. the judgements are precisely those rules whose codomain is $\text{ctx}$;
3. $R$ and $P$ are closed under finite limits (see Requirement 1.0.9, Requirement 1.0.10 and Requirement 1.0.11), $\sharp$-liftings (see Requirement 1.0.14) and whiskering (see Requirement 1.0.15).

The rest of this section is dedicated to clarifying the technical aspects of this definition. In the next section we will see that these properties influence the inference power of our logical systems. The more we put, the more we infer.

**Remark 1.0.5.** The condition (2) in Definition 1.0.4 is actually not needed, yet it is not harmful for the theory and it allows a cleaner axiomatization of (3), which otherwise would not look as pretty.

**Remark 1.0.6** (Infinitary judgemental theories). We could have allowed $\lambda$-small limits for $\lambda$ a (regular) cardinal, so that we are actually studying finitary judgemental theories. In the present work we stick to this choice.

**Remark 1.0.7** (Economical presentations of judgemental theories). In the majority of concrete instances, a judgemental theory is presented by a pre-judgemental theory $(\text{ctx}, J, R, P)$, in the sense that we close the data of judgements, rules and policies under finite limits and $\sharp$-liftings and whiskering. This produces the smallest judgemental theory containing $(\text{ctx}, J, R, P)$.

**Notation 1.0.8.** When a classifier $\mathcal{X}$ is obtained by iterated pullback of classifiers along rules, we try to use a notation that keeps in mind this special property of the classifier. Consider thus the diagram below.

![Diagram](image)

- We use the notation $\mathcal{F} \times \mathcal{G}$ when we pullback classifiers along classifiers.
- When we pullback a classifier along a rule, we use the notation $\mathcal{H}.\mathcal{F}$. We can make sense of this as if we put an additional bound on $\mathcal{F}$, and this is induced from $\mathcal{H}$ via $\lambda$. The reader will find more about this in 2.3.2.
When we iterate this procedure, for example as in the diagram, we use the notation $(\mathcal{H}, \lambda F)_{\gamma, \mathcal{V}}$. When $g = f$ we write it $\mathcal{H}_{\gamma, \mathcal{V}}$.

We are aware that this notation is not entirely economical, nor uniquely determined, but in the practical circumstances of this paper, it will be very useful.

**Requirement 1.0.9** (Pullbacks). $\mathcal{R}$ is closed under pullbacks in the sense that, given solid (black) spans and cones in $\mathcal{R}$ as below, we have that all the colored arrows belong to $\mathcal{R}$.

In this definition we see the advantage of including $\mathcal{J}$ in $\mathcal{R}$, otherwise we would have to specify another axiom for the case in which the span is made of judgments. This could have been done without major differences, but would lead to an incredible proliferation of diagrams.

**Requirement 1.0.10** (Equalizers). Similarly to the case of pullbacks, we require that the equalizer $\mathcal{E}$, together with its limiting maps, belongs to the rules.

**Requirement 1.0.11** (Powers). We also require that, for all rules $\mathcal{X} \rightarrow \mathcal{Y}$, we can form the finite powers below in $\mathcal{R}$ and that, as in Requirement 1.0.10 and Requirement 1.0.9, all the arrows induced by their universal properties by cones made of rules, are rules too.

Regarding our meta-theory, this only requires that the finite product of sets (or classes, or $\kappa$-sets for some inaccessible cardinal $\kappa$, depending on the meta-theory of choice) is again a set (or class, or $\kappa$-set).

Finally, we complete the discussion of Definition 1.0.4 by explaining what we mean by closure under $\sharp$-liftings – and what a $\sharp$-lifting is.

**Definition 1.0.12** ($\sharp$-lifting). Consider a functor $f: \mathcal{A} \rightarrow \mathcal{B}$ and a 2-cell $\alpha: c' \Rightarrow c: \mathcal{C} \rightarrow \mathcal{B}$, and compute the pullback of $c$ and $c'$ along $f$. A sharp lifting or $\sharp$-lifting of $\alpha$ along $f$ is a pair $(id \times_B f, \alpha \times_B f)$ of a functor and a natural transformation.
as below,

\[
\begin{array}{ccc}
  c \times_{B} f & \xrightarrow{f^*c} & A \\
  \text{id} \times_{B} f & \xrightarrow{\alpha \times_{B} f} & c' \times_{B} f \\
  & \xRightarrow{f^*c'} & \\
  c & \xrightarrow{c} & B \\
  \text{id} & \xRightarrow{c} & c' \\
\end{array}
\]

so that all “vertical” squares commute.

Remark 1.0.13. We could not find any precise instance of \(♯\)-liftings in the literature, our construction seems original. The most comparable results seems to be contained in [Gra66], see for example Thm. 2.10 in loc. cit.

This construction is clearly less known than the others appearing in the previous sections, but it will be of fundamental importance in using judgemental theories, as it is actually quite closely related to the process of computing substitution. We detail its technical features in Section 1.1 and its consequences for the logic in Section 2.6, but for the moment the reader only needs to know that whenever \(f\) is a(n) (op)fibration, such a \(♯\)-lifting exists.

Requirement 1.0.14 (\(♯\)-lifting). Consider a policy \(λ\) as in the diagram below, and a rule \(R^*\) which is a fibration. Then, by Theorem 1.1.2, there is a pair \((R^*λ, R^*λ')\) as below. Closure for \(♯\)-liftings amounts to ask that \(R^*λ\) and \(R^*λ'\) belong to \(R\) and \(P\), respectively. Similarly, for \(R_*\) an opfibration, we get the op-diagram on the right. Notice that in both cases the square containing \(λ\) and \(R^*λ\) or \(R_*λ\) commutes strictly.

Requirement 1.0.15 (Whiskering). As it is quite frequent in 2-category theory, one might want to compose 1-cells with 2-cells. As our theory is quite heavily 2-dimensional, it only make sense that we ask that performing such an operation does not bring us out of our logic. We recall the general definition in the 2-category \(\text{Cat}\), as it is the one we are interested in now. Consider categories, functors, and natural transformations as below.

\[
\begin{array}{ccc}
  \mathcal{A} & \xrightarrow{F} & \mathcal{B} \\
  \mathcal{C} & \xrightarrow{G} & \mathcal{D} \\
\end{array}
\]
One can always define natural transformations $\alpha \ast F : GF \Rightarrow G'F$ and $H \ast \alpha : HG \Rightarrow HG'$ that point-wise act as

$$(\alpha \ast F)_A = \alpha_{F,A} \quad (H \ast \alpha)_B = H(\alpha_B).$$

Given classifiers $\mathcal{X}, \mathcal{Y}, \mathcal{Z}$, rules $\lambda, \lambda'$, $\gamma$ and a policy $\lambda^\#$, then, we say that the judgemental theory is closed under whiskering in the sense that the colored natural transformations are policies too.

\[
\begin{align*}
\mathcal{X} \xrightarrow{\lambda} \mathcal{Y} \xrightarrow{\gamma} \mathcal{Z} \quad &\quad \mathcal{X} \xrightarrow{\gamma \lambda} \mathcal{Z} \\
\mathcal{Z} \xrightarrow{\lambda} \mathcal{Y} \xrightarrow{\lambda'} \mathcal{X} \quad &\quad \mathcal{Z} \xrightarrow{\lambda \gamma} \mathcal{X}
\end{align*}
\]

Remark 1.0.16. We understand that up to this point the reader has been faced with many concepts and strange notations that they have no intuition for. Therefore, before we formally describe what it means to define a calculus based on the blocks that are our judgemental theories, we advise the reader to skip to Section 3.3.1 and see what it is that we are trying to achieve.

1.1. Notions of substitution.

Definition 1.1.1. A judgement classifier is (op)substitutional if it is an (op)fibration. A rule is (op)cartesian if it preserves (op)cartesian maps. A policy is (op)Frobenius with respect to a given judgement classifier if it has cartesian components.

By extension, we will say that a (pre)judgemental theory is (op)substitutional if all judgement classifiers are (op)substitutional, all rules are (op)cartesian, and all policies are (op)Frobenius.

Theorem 1.1.2 (Characterizing fibrations via $\sharp$-lifting). The following are equivalent for a functor $p : E \to B$:

1. $p$ is a fibration with a cleavage $S$;
2. each 2-cell $\alpha : c' \Rightarrow c : C \to B$ admits a terminal $\sharp$-lifting along $p$, meaning that provided another $(g, \beta)$ $\sharp$-lifting of $\alpha$ along $p$, we have a unique vertical $\beta$ such that $\beta = \alpha \times_B p \ast \beta$.

\[
\begin{align*}
\begin{array}{c}
c \times_B p \\
g \downarrow \beta
\end{array} &\Rightarrow \begin{array}{c}
E \\
\ast
\end{array} &\quad
\begin{array}{c}
c \times_B p \\
g \downarrow \beta
\end{array} &\Rightarrow \begin{array}{c}
E
\end{array}
\end{align*}
\]

Proof. If $p$ is a fibration with cleavage $S$, we can define the functor

$$id \times_B p : c \times_B p \to c' \times_B p, \quad (X, A) \mapsto (X, S(A, \alpha X))$$

reindexing $A$ along $\alpha X : c'X \to cX = pA$. All desired squares commute. Moreover, we have a natural transformation $\alpha \times_B p : p^\gamma c' \circ id \times_B p \Rightarrow p^\gamma c$ that on components is defined as follows

$$(\alpha \times_B p)(X, A) = s_{A, \alpha X}.$$

In particular, since each $s_{A, \alpha X}$ is cartesian we have that the pair $(id \times_B p, \alpha \times_B p)$ enjoys the desired universal property: the induced unique vertical arrows assemble into the necessary $\beta$. 

Conversely, let $p$ a functor and consider the trivial 2-cell $\text{dom} \Rightarrow \text{cod}$, then there exists a terminal sharp lifting as below,

therefore $\text{id} \times_B p$ maps a pair $(A, \sigma : \Theta \to pA)$ to a pair $(B, \sigma : \Theta \to pA)$ with $pB = \Theta$, and there is a morphism $(\alpha \times_B p)_{(A,\sigma)} : B \to A$

in $\mathcal{E}$ over $\sigma$. We denote $(\text{id} \times_B p)_{(A,\sigma)} = S(A,\sigma)$ and $(\alpha \times_B p)_{(A,\sigma)} = s_{A,\sigma}$. It is cartesian because any other map over $\sigma : \Theta \to pA$ is part of another $\sharp$-lifting $(g, \beta)$ of $\alpha$ along $p$ and since $(\text{id} \times_B p, \alpha \times_B p)$ is terminal with respect to this property, the unique induced $\beta$ produces a suitable unique vertical map into $S(A,\sigma)$.

**Notation 1.1.3** (Substitution). For the time being, and to avoid continuous explicit reference to a given cleavage for each fibration involved, we write $A[\sigma]$ for what was called $S(A,\sigma)$ up to this point.

See Section 2.6 for what these imply for judgemental theories, for the moment we only prove a couple of technical results.

**Lemma 1.1.4** ($\sharp$-lifting of cartesian functors). Consider a fibration $h$ and a 2-cell $\lambda^\sharp$ as follows, and apply the construction in Requirement 1.0.14.

If $\lambda$ preserves cartesian maps, then so does $h^\ast \lambda$.

**Proof.** Consider a morphism $a = (a_1, a_2) : (F', H') \to (F, H)$ in $\mathcal{F.H}$, meaning a pair $a_1 : F' \to F$ in $\mathcal{F}$ and $a_2 : H' \to H$ in $\mathcal{H}$ such that $f(a_1) = \sigma = h(a_2)$. One can check (see, for example, [Jac93, Proposition 2.6]) that this is cartesian with respect to $h \circ (f,h)$ if and only if both $a_1$ is $f$-cartesian and $a_2$ is $h$-cartesian. The latter is
equivalent to saying that $a_2$ is of the form $a_2 = \sigma \cdot H[\sigma] \to H$. Now consider that the functor $h^*\lambda$ acts as follows

$$(a_1, a_2): (F', H') \to (F, H) \quad \mapsto \quad (\lambda a_1, a_2^1): (\lambda F', H'[\lambda^x_{F'}]) \to (\lambda F, H[\lambda^x_{F}])$$

with $a_2^1$ the unique map induced by naturality of $\lambda^x$ at $h(a_2)$. Assume that $a$ is cartesian, then we end up having

$$H[\sigma][\lambda^x_{F'}] \xrightarrow{\lambda^x_{F'}} H[\sigma] \xrightarrow{\sigma} H[\lambda^x_{F}] \xrightarrow{\lambda^x_{\Phi}} H$$

therefore $a_2^1 = \sigma'$ is itself cartesian. Hence if $\lambda$ preserves cartesian maps, then so does $h^*\lambda$. □

**Remark 1.1.5** ($\sharp$-lifting is cartesian). The natural transformation $h^*\lambda^x$ has $h$-cartesian components.

**Proof.** This is actually trivial by definition of $h^*\lambda^x$: in fact, it acts as

$$(h^*\lambda^x)_{(F, H)} = \lambda^x_{\Phi}: H[\lambda^x] \to H.$$ □

## 2. Judgement Calculi

In the previous section we have introduced judgemental theories, very concrete mathematical objects for which we have presented a suggestive notation referencing some logical intuition. This section is devoted to grounding that intuition and showing that each (pre)judgemental theory is a categorical version of a proof assistant or, more technically, something that supports the categorical semantics for the specification of a type system. We will see how a judgemental theory automatically produces a deductive system via a process of translation. Actually, a judgemental theory is intrinsically a calculus of deduction in a very precise sense.

This section will describe a way to translate the data of a judgemental theory $(\text{ctx}, \mathcal{J}, \mathcal{R}, \mathcal{C})$ into a judgement calculus.

Of course, such a process of translation requires an almost formal definition of judgement calculus, which must be flexible enough to encode the usual calculi that are used in type theory and in proof theory. For the reasons expressed in the introduction, this is a non-trivial task.
2.1. **Prolegomena.** As we have hinted in the introduction, a very general definition of deductive system or *calculus* is much easier to describe than to actually define. Of course, one can make reference to [Res02], or to [MS84], or to [HHP93], or to several variations of this notion, but there is no unified take we find satisfying. In this subsection we go through a critical analysis of the deductive systems of a dependent type theory and of a proof theory to better motivate the choices of the next subsection.

2.1.1. *The deductive system of a DTT.* We consider the problem of defining the semantics of the underlying signature, judgements and rules defining a formal calculus of a dependent type theory based on Martin-Löf’s type theory. There are indeed several approaches in the literature, and the very notion of type theory is somehow (intentionally) fuzzy. We would go as far as to say that a complete agreement on the matter does not exist. Of course, this flexibility is part of the richness of this theory. The informality in the definition of rule and type constructor is one of the reasons for which the topic of (categorical) semantics for dependent type theory is both so popular and so useful in the theoretical research on dependent type theory. Most sources would probably agree that to declare (the calculus of) a dependent type theory means to specify three boxes of data.

**(S)** Syntax (contexts, types, terms): a theory of dependent types is -informally- a formal system dealing with *types* and *terms* in *context*. From a symbolic point of view, these are a bunch of glyphs that we use as atoms of our language.

\[ \Gamma A a : A \]

**(J)** Judgements (about contexts, types, terms): a judgement is a very simple sentence made up of symbols from the syntax, and whose intention is to somehow bound together pieces of atomic data. The most simple type theories of the sort we refer to present three possible (kinds of) judgements,

\[ \Gamma \vdash \Gamma \text{ctx} \quad \Gamma \vdash A \text{Type} \quad \Gamma \vdash a : A \]

which are informally interpreted as \( \Gamma \) is *a context*, \( A \) is *a type in context* \( \Gamma \), \( a \) is *a term of type* \( A \) in *context* \( \Gamma \).

**(R)** Rules (to declare new types, terms, and interact with the syntax): Finally, we should be able to interact with and declare a type. For those that are acquainted with a programming language, this need is completely evident. Indeed we might want to introduce a type which is constructed from other types.

\[ \text{(DTy)} \quad \frac{\Gamma \vdash a : A \quad \Gamma, A \vdash B \text{Type}}{\Gamma \vdash B[a] \text{ Type}} \]

Depending on the complexity of the theory, beyond a bunch of basic rules (like type and term dependency (Section 3.4)), we find type constructors. Type constructors are packages of rules, labelled by their feature, that allow to construct new types from old ones. Below we list the inescapable labels, for a constructor whose name is - say - \( \Phi \).

\( \Phi \) Some *formation rule(s)*, presenting the type. They specify under which circumstances we can assume it to exist (or, from the point of view of programming languages, we can form it). By circumstances, we usually refer to syntactic data.
Some introduction rule(s), producing the canonic terms of a such type. Given a set of syntactic data, they tell how to cook up a term of the new type.

Some elimination rule(s), specifying the interaction between a term of the new type and the terms of the types that contributed to the formation of the new type.

Additionally, based on the computational semantics associated to the theory one wishes to consider, one needs to describe how introduction and elimination interact with one another, meaning to provide suitable conversion rules.

These three packages of data reflect the necessities of a type theory: indeed, type theory emerged as a foundational framework, but from a cultural point of view its history is intertwined with that of programming languages. This deep interaction has shaped several aspects of type theory, we will see this especially in the declarative and interactive nature of the Rules box. Let us stress on the fact that, besides these informal distinctions, there is no formal definition of a type constructor, nor of a rule.

While the three boxes of Syntax, Judgements, and Rules are definitely there in any type theory, the list of rules, judgments, and the sort of symbols that inhabit them is subject to major choices. Even those that we have listed can be seen as somewhat arbitrary. Still, we believe that in any reasonable type theory the data above will be included. In most of the concrete instances, type theories are even richer than what we have listed above:

- morphisms of contexts are usually added to Syntax;
- definitional equality is usually added to Judgements;
- \(\beta\) - and \(\eta\)-computation are almost always added to Rules, and through definitional equality they determine how introduction and elimination interact with one another. Also, we will see that our type theory has context formation, which stands for a set of rules that form fresh contexts from existing types. Finally, if the syntax is enriched with morphisms of contexts, there might be rules regulating their interaction with judgements (that would be substitution).

A vast majority of computer scientists and type theorists would probably classify \(\beta\) - and \(\eta\)-computation as an essential feature of a type theory.

2.1.2. Natural deduction. As it was said in the introduction, natural deduction has already been shown to be fittingly translatable in the language of types [Mar96b], but we here describe its interpretation separately for multiple reasons:

(1) on one hand, natural deduction for first-order logic has had a greater fortune in being studied and employed in our schools and universities, and it is the one that we believe is understood best among most people, therefore

(2) we believe that its “familiarity” makes it easier for the reader to connect the categorical syntax for the intuition of what the rules should be, moreover

(3) such a familiarity allows us the freedom to describe more rules, and the practice of such an encoding is the main aim of this paper.

The following presentation is mostly inspired by [Res02, Def 2.18], but it is coherent with the treatment of [NvP08] and [TS00] too.

When specifying a natural deduction calculus we provide three boxes of data:
(S) Syntax (variables, formulae): a natural deduction calculus is -informally- a formal system dealing with variables, lists of formulae and formulae. From a symbolic point of view, these are glyphs that will be the atom of our calculus,

\[ x \quad \Gamma \quad \phi. \]

Often punctuation symbols as the semicolon ; are used too to combine the symbols.

(S) Sequents: a sequent is a very simple sentence made up of symbols from the syntax, and whose intention is to specify an entailment relation between the data on the left and the data on the right of the entailment symbol.

\[ x; \Gamma \vdash \phi. \]

For example, the sequent above could be read the list of formulae in \( \Gamma \) entails the formula \( \phi \), and they all have (at most) free variables in \( x \).

(R) Rules: in natural deduction rules are used to state atomic consequences, they transform a family of sequents into a (family of) sequent(s).

\[
\text{(Cut)} \quad \frac{x; \Gamma \vdash \phi \quad x; \Gamma, \phi \vdash \psi}{x; \Gamma \vdash \psi}
\]

Traditionally, there is a distinction between structural rules [Res02, 2.23] and other rules. Referring to [Res02, pag. 26], the structural rules influence what we can prove. The more structural rules you have, the more you will be able to prove. The other rules are more in the spirit of type constructors and they account for the behavior of the logical operators, like \( \land, \lor, \forall, \exists \).

In modal logics, they can account for the behavior of modal operators too.

It is pretty intuitive that we can treat a sequent as a form of judgement. This just amounts to a re-tuning of our intuition with respect to the way we are used to read sequents. On the other hand it is not entirely trivial to find a precise correspondence between the rules of proof theory and the constructors of type theory. For example, for the reason that there is not a precise definition of constructor, nor a classification of them.

2.1.3. Judgement calculi. Given the discussion above, our challenge is pretty clear: how to accommodate extensional type constructors, connectives, and deduction rules in a conceptually unified and technically coherent framework? Provide that we can see sequents as judgements, how do we formally deal with their manipulation from a semantic point of view? Let us dive into the definitions. For us, to declare a judgement calculus means to specify three boxes of data:

(S) Syntax (contexts and objects);

(J) Judgements (acts of knowledge bound to a context and pertaining to a (list of) object(s));

(R) Rules (transforming judgements into other judgements).

2.2. Syntax. Let \((\text{ctx}, \mathcal{F}, \mathcal{R}, \mathcal{E})\) be a (pre)judgmental theory. Then its corresponding judgement calculus has in its Syntax box letters for each context and each object in a judgement classifier.

\[
\begin{array}{c}
\Gamma \\
\in \text{ctx}
\end{array} \quad \begin{array}{c}
\mathcal{F} \quad \in \mathcal{F}
\end{array}
\]
2.3. Judgements. Judgemental calculi include two main kinds of judgement for each $F \in \mathcal{F}$.

- The first kind of judgement acknowledges a $\mathcal{F}$-empirical evidence and clarifies the status of an object. One can see it as a kind of Tarskian snow for our approach, meaning something that fulfills Tarski’s requirement for something to “characterize unambiguously the class of those words and expressions which are to be considered meaningful” [Tar56]. For $F \in f^{-1}(\Gamma)$, then we find in our set of judgement the writing

$$\Gamma \vdash F \mathcal{F}.$$  

This can be understood as Given $\Gamma$, $F$ exists or Given $\Gamma$, $G$ is green, or Given $\Gamma$, $M$ is made of marble. In the case of the same category $\mathcal{F}$ appearing as the domain of two different judgements, we might use $\Gamma \vdash F F(f)$.

- The second kind of judgement is an equality checker for the equality classified by the judgement. We will write

$$\Gamma \vdash F = F',$$

when $F, F' \in f^{-1}(\Gamma)$ and $F = F'$. This could be read as Given $\Gamma$, $F$ and $F'$ are indistinguishable by existence\(^1\), or Given $\Gamma$, $G$ and $G'$ are indistinguishable by green. Notice that the interpretation of the notion of equality is relative to the choice of the classifier. If we were to look at something classifying types in a type theory, the equality would (and will, in Section 3) be indistinguishability up to computations.

In the table below, we find on the left column the judgement and on the right its translation in terms of the judgemental theory.

| $\Gamma_1 \vdash F_1 \mathcal{F}$ | $\ldots$ | $\Gamma_n \vdash F_n \mathcal{F}$ | $(F_1 \ldots F_n) \in f^{-1}(\Gamma_1) \times \ldots \times f^{-1}(\Gamma_n)$ |
|---------------------------------|-----------|---------------------------------|------------------------------------------------------------------|
| $\Gamma \vdash F = F'$          | $F, F' \in f^{-1}(\Gamma)$ and $F = F'$ |

It might seem that such simple judgements do not guarantee much in terms of expressiveness. This is in fact far from the truth! Recall that a judgemental theory is closed under finite limits and several constructions, thus we obtain an incredible variety of complex judgements.

**Remark 2.3.1** (On notions of equality and the relationship between theory and meta-theory). Notice that all choices made here are to consider as “external”, in the sense that they constitute the building blocks of our calculus. They do not prevent from having, say, an identity judgement in a judgemental theory, see for example Section 3.6.

2.3.2 (Nested judgements: pullbacks). Let $(\text{ctx}, \mathcal{F}, \mathcal{R}, \mathcal{E})$ be a judgemental theory and consider a judgement of the form,

$$\Gamma \vdash H.\lambda F \mathcal{H.}\lambda \mathcal{F}.$$  

\(^1\)This could be actually read Are identical.
We will see that such a judgement classifies a nested family of judgements, depending on the data of \( \mathcal{H}, \mathcal{F} \) and \( \lambda \).

\[
\begin{array}{c}
\mathcal{H} \xrightarrow{\lambda} \mathcal{G} \xrightarrow{g} \text{ctx} \\
\mathcal{H} \xrightarrow{f} \mathcal{F} \\
\mathcal{H} \xrightarrow{\lambda} \mathcal{F} \xrightarrow{\ldots} g \lambda f \xrightarrow{\ldots} \mathcal{F}
\end{array}
\]

By inspecting the pullback diagram that defines \( \mathcal{H} \lambda \mathcal{F} \) we can see that judgements of this form are in bijection with pairs of judgements of the form,

\[
\Gamma \vdash H \quad g \lambda H \vdash F \quad \mathcal{F}.
\]

This means that we are entitled to see the line above, which is composed of two related but separate judgements (in possibly different contexts!), as a single one (in context \( \Gamma \)). We call judgements of this form \textit{nested}. Notice that, depending on what we want to express, we could say that the relation binding \( H \) to \( \Gamma \) (hence the judgement classifier with domain \( \mathcal{H} \) in the line above) is either \( g \lambda \) itself, therefore forcing both \( H \) and \( F \) to have the same context, or some other \( h \). Still, we chose to present the pullback in its most general form.

As a string of symbols, notice that a nested judgement is an informal judgement, in the sense that it is not well defined in our framework. Despite this, we will feel free to use notations as that above because they are a bit easier to parse from a human perspective. This means that for the rest of the paper we will write

\[
\begin{array}{c}
\Gamma \vdash H \quad \mathcal{H} \lambda \mathcal{F} \\
\mathcal{H} \lambda \mathcal{F} \xrightarrow{\ldots} g \lambda H \xrightarrow{\ldots} F \quad \mathcal{F}
\end{array}
\]

to intend that the synthetic judgment below is an \textit{alias} for the nested judgement above, which is defined in our context.

Our notation \((\mathcal{H} \lambda \mathcal{F})\) retains almost all the information needed to predict the kind of nested judgements we classify. This also explains why we write composed judgements the way we do: we think of the component in \( \mathcal{H} \) to be free, while that in \( \mathcal{F} \) is bounded via the map \( \lambda \).

**Example 2.3.3** (Lists). Let \( \mathcal{X} \) be a judgement classifier, and consider \( \mathcal{X}^n \), which is given by the (wide) pullback below,

\[
\begin{array}{c}
\mathcal{X} \xleftarrow{\pi_1} \mathcal{X} \\
\mathcal{X} \xleftarrow{\pi_n} \mathcal{X}
\end{array}
\]

then the judgement \( \Gamma \vdash X_1, \ldots, X_n \mathcal{X}^n \) can be interpreted as a list of judgements, as described below.

\[
\begin{array}{c}
\Gamma \vdash X_1, \ldots, X_n \mathcal{X}^n \\
\Gamma \vdash X_1 \mathcal{X} \quad \ldots \quad \Gamma \vdash X_n \mathcal{X}
\end{array}
\]
Notice that the fact that a judgemental theory is by definition closed under finite products implies that these judgements are always available.

**Example 2.3.4** (Composable arrows). Let $\mathcal{C}$ be a category and consider the following pullback as on the left. The resulting nested judgement, then, reads as on the right and classifies composable arrows in $\mathcal{C}$.

$$
\begin{array}{ccc}
C^2 \downarrow \text{dom} & \rightarrow & C \\
\downarrow \text{cod} & & \\
C^2 & \rightarrow & \mathcal{C}
\end{array}
$$

$$
\Gamma \vdash f, g \in C^2 \text{dom} \land \text{cod} \\
\Gamma \vdash f \in \mathcal{C}(\text{cod}) \\
\Gamma \vdash g \in \mathcal{C}(\text{dom})
$$

The middle ground is given by the actual *middle* object $f$ and $g$ share. Notice that, though the context (i.e. the object) is the same, their bounds to it are very different (that is, respectively cod and dom).

**Example 2.3.5** (Toy Martin-Löf type theory). In the judgemental theory generated by that in Example 1.0.3, we now have a way of coding, for example, pairs of types in the same context. This is achieved by the pullback $U \Delta U.$

$$
\begin{array}{ccc}
U \Delta U & \rightarrow & U \\
\downarrow \Delta U & & \\
\mathcal{U} & \rightarrow & \text{ctx}
\end{array}
$$

$$
\begin{array}{c}
\Gamma \vdash (A, A') \in U \Delta U \\
\Gamma \vdash A \in \mathcal{U} \\
\Gamma \vdash A' \in \mathcal{U}
\end{array}
$$

The examples above are not particularly interesting, though we believe they give an intuition of the expressive power of nested judgements. We hope Section 3 will be definitive proof.

**2.3.6** (Nested judgements: equalizers). Similarly to the previous case, equalizers classify nested judgements of the kind below.

$$
\begin{array}{ccc}
\mathcal{E}(\lambda, \lambda') & \rightarrow & F \\
\downarrow \lambda & & \downarrow \lambda
\end{array}
$$

$$
\begin{array}{c}
\Gamma \vdash F \in \mathcal{E}(\lambda, \lambda') \\
\Gamma \vdash \lambda F =_{\mathcal{G}} \lambda' F
\end{array}
$$

2.4. Rules. Let $(\text{ctx}, \mathcal{F}, \mathcal{G}, \mathcal{P})$ be a judgemental theory. Consider judgements and a rule as in the diagram below, for each rule $\lambda : \mathcal{F} \rightarrow \mathcal{G}$ and each judgement $\Gamma \vdash F \in \mathcal{F}$, we will write as follows (on the right).

$$
\begin{array}{ccc}
\mathcal{F} & \xrightarrow{\lambda} & \mathcal{G} \\
\downarrow f & & \downarrow g
\end{array}
$$

$$
\begin{array}{c}
\Gamma \vdash F \in \mathcal{F} \\
\Gamma \vdash g \lambda F \in \mathcal{G}
\end{array}
$$

From a technical level, this is just a compact way to organize the data of the functoriality of $\lambda$. Indeed it is true that $\lambda F \in g^{-1}(g \lambda F)$, so that $g \lambda F \vdash \lambda F \in \mathcal{G}$ is actually a judgement in our framework. This is the only kind of rule that we admit in our judgemental calculi, and in a sense all the rules are the same, there are no intrinsic labels like *structural, introduction, elimination*, and so on. Yet, similarly to the case of judgements, the closure under finite limits guarantees an incredible richness of rules, as we will see for the rest of the subsection.

**Example 2.4.1** (Toy Martin-Löf type theory). The rule $\Sigma$ from Example 1.0.3 now reads as follows.
Now recall that $u\Sigma = \dot{u}$, so the behavior of a policy is implied: see Section 3 for more on this. We follow the intuition provided for all the data of the judgemental theory in Example 1.0.3 and translate it in the usual type-theoretic strings of symbols. Then it reads as follows

\[
\Gamma \vdash a : A \\
\Gamma \vdash A : \text{Type}
\]

and depicts the typing rule. We thoroughly detail this process of translation in Section 3.3.

**Remark 2.4.2** (Rules with many outputs). The notion of nested judgement 2.3.2 and of our calculus as a whole have one additional very useful feature, and that is allowing for multiple consequents *simultaneously*. In fact, it is very common that one might want to write rules that deduce several judgements from the same (set of) judgement(s), but writing it organically is somewhat frowned upon, so that one usually encounters a proliferation of rules (for example two elimination rules in Section 3.6 and in Section 4.4). While we mostly follow the tradition with regard to this, the attentive reader will see that in fact they are always the product of the “break-down” of a single nested judgement. We make this explicit once in Remark 3.4.6.

**Remark 2.4.3** (Soundness and completeness). When one looks at this section as a whole, that is the process of producing a graphical/grammatical bookkeeping of the categorical properties of the judgemental theory, organized in the form of a collection of judgements and deductions, it is natural to raise the question which deductions are actually produced by a judgemental theory? There are two possible approaches to this question.

- The first approach is to refine our notion of calculus, and give a precise definition of what we mean by deductive system. Our presentation is not that far from a formalization. Then, one would say that the content of this section provides a kind of soundness/correctness theorem for the categorical syntax, and one could try to provide a completeness result that characterize all the possible judgements and deductions.
- The second approach is to claim that the question contains an implicit bias towards grammatical/syntactic representations of deductions, and that, in a sense, the categorical language already provides the grammar the reader is looking for, while the calculus in this section only represents a way to make it more digestible to the *grammarians*.

Both the approaches are valid, one maybe making a more political statement, and the other being more prone to the classical tradition. Because the author herself does not entirely agree on the path to follow, and because this paper already contains a lot of material, we choose not to invest more on this question in the present work. We will be greatly interested in developing this more along the line.

2.5. **Policies.** Recall that a policy is a 2-cell as follows

\[
\begin{array}{ccc}
\mathcal{F} & \xrightarrow{\lambda} & \mathcal{G} \\
\downarrow f & & \downarrow g \\
\text{ctx} & \xleftarrow{\lambda^!} & \text{ctx}
\end{array}
\]
with \( f, g \) judgement classifiers and \( \lambda \) a rule. This additional data contains that of the rule \( \lambda \), hence

\[
\lambda \vdash F \quad \frac{\Gamma \vdash F \quad \Theta \vdash \lambda F}{\Theta \vdash \lambda F \quad G}
\]

but it also establishes a relation between \( \Gamma = fF \) and \( \Theta = g\lambda F \), namely \( \lambda^F_\Theta : \Theta \rightarrow \Gamma \).

We wish our judgemental calculus to reflect this.

If \( f \) is a fibration by Requirement 1.0.14 we can lift \( \lambda^F \) along \( f \),

\[
\begin{array}{c}
\xymatrix{ F \ar[dr]^{g.f} \ar[r]^{f.f} & F \\
F \ar[r]_f & \text{ctx} \ar[u]_{\lambda^F} \ar[ru]_g & \\
& \ar[u]_{\lambda^F_\Theta} & \\
& & \ar[u]_\theta &}
\end{array}
\]

and get a pair \((f^*\lambda, f^*\lambda^F)\) such that on a pair of objects \( F, F' \) over the same context

\[
f^*\lambda : (F, F') \mapsto (\lambda F, F'[\lambda^F_\Theta]),
\]

hence we can use the universal property of pullbacks to precompose the policy “on top” with \((\text{id}, \text{id})\) to get the lax triangle on the right. This now reads as

\[
\begin{array}{c}
\xymatrix{ \Gamma \vdash F \quad \frac{\Gamma \vdash F \quad \Theta \vdash \lambda^F_\Theta}{\Theta \vdash F \quad \lambda^F_\Theta} \quad \frac{\lambda^F_\Theta : \Theta \rightarrow \Gamma}{\Theta \vdash \lambda^F_\Theta} }\end{array}
\]

and get a pair \((f^*\lambda, f^*\lambda^F)\) such that on a pair of objects \( F, F' \) over the same context

\[
f^*\lambda : (F, F') \mapsto (\lambda F, F'[\lambda^F_\Theta]),
\]

therefore we can use the universal property of pullbacks to precompose the policy “on top” with \((\text{id}, \text{id})\) to get the lax triangle on the right. This now reads as

\[
\begin{array}{c}
\xymatrix{ \Gamma \vdash F \quad \frac{\Gamma \vdash F \quad \Theta \vdash \lambda^F_\Theta}{\Theta \vdash F \quad \lambda^F_\Theta} \quad \frac{\lambda^F_\Theta : \Theta \rightarrow \Gamma}{\Theta \vdash \lambda^F_\Theta} }\end{array}
\]

where \( \lambda^F_\Theta : \Theta \rightarrow \Gamma \).

One could detail a similar argument for covariant policies: we do not do so here because in the present work we will only encounter contravariant ones. Still, the reader can easily see how covariant rules are strictly connected to comonads, and comonads have been proven of special interest in logic (see, for example, the treatment of equality in [DR21]), and this is why we have carried them through all definitions and technical proofs.

2.6. **On substitution.** Section 2.5 is a first instance of application of substitution-like properties in our setting, in it is worth noticing that the additional data of a policy can be only *externalized* in our setting when the target judgement classifier is a fibration. It seems worth it, then, to describe what information - from the point of view of judgemental theories - lies under the assumption that a functor is a fibration.

Recall from Theorem 1.1.2 that for a fibration \( p : \mathcal{E} \rightarrow \mathcal{B} \) there are \((p^*\text{id}, p^*\alpha)\) such that for any other \( \sharp \)-lifting \((g, \beta)\) there is a unique \( p \)-vertical \( \beta \) satisfying \( \beta = p^*\alpha \cdot \beta \).
Therefore it seems that the peculiarity of fibrations lies in the existence of a unique vertical \( \beta \). We break down its meaning in the following policy, resulting from whiskering \( \beta \) with \( p \circ \text{dom}.p \),

\[
\begin{array}{c}
B^2.\text{cod} \xrightarrow{\text{cod}.p} \mathcal{E} \\
\downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\
B^2.\text{dom} \xrightarrow{\text{dom}.p} \mathcal{E} \\
\end{array}
\]

and it is easy to see that \( p \circ \text{dom}.p \circ p^* \text{id} \) is a fibration, therefore we can apply the discussion in Section 2.5 and derive the following rule in our judgemental theory. With \( \sigma : \Theta \rightarrow pA \) and \( B(A, \sigma) = (\text{dom}.p \circ g)(A, \sigma) \), given that \( (A, \sigma)[\beta A, \sigma] = B(A, \sigma) \),

\[
\Theta \vdash (A, \sigma) B^2.\text{cod} \xrightarrow{g} B^2.\text{dom} \xrightarrow{(\text{podom}.p)^* \text{id}} B
\]

meaning that initiality translates to the fact that any substitution is derivable from the cartesian one.

2.7. Logics vs Theories. The next two sections will focus on modeling dependent type theories and natural deduction in our framework. To be precise, the data of a judgemental theory will be the same as a theory satisfying the specifics of an intended logic. In its current state, this is a limitation of our framework: we do not offer a modular way to specify a logic so that the theories in such a logic are precisely the judgemental theories of a certain shape, and we can only perform this presentation via a case by case analysis (which is precisely the content of the next two sections). That said though, this programme is not outside our general scope, and we briefly address this topic in the last paragraph of Section 6.

3. Plain dependent type theory

In this section we show what features must a judgemental theory have in order to support dependent type theory. We show it produces desired rules, and with respect to this provide some evidence of the computational power of judgemental theories. We then pin-point which rules one needs to add in order to gain typically desirable constructors, for example dependent products and identity. In doing so, we learn something about constructors in general and give a (unifying) definition of extensional type constructor.
**Definition 3.0.1** (Plain dependent type theory). A *plain dependent type theory* is a substitutional judgemental theory generated by the pre-judgemental theory described by the diagram below.

![Diagram](image)

To be precise, we mean that \( \mathcal{J} = \{ \hat{u}, u \} \) and that those are fibrations, \( \mathcal{R} = \{ \Sigma, \Delta \} \), \( \mathcal{P} \) contains the witness of the commutativity of the solid diagram, and both the unit and the counit \((\epsilon, \eta)\) of the adjunction \( \Sigma \dashv \Delta \). Finally, we require that \((\epsilon, \eta)\) are cartesian natural transformations. We call this \( pDTT \), for short.

We think of \( \mathcal{U} \) as classifying types, \( \hat{\mathcal{U}} \) as classifying terms, and the functor \( \Sigma \) as the one performing the typing. Its adjoint \( \Delta \) will interpret context extension. The choice of the greek letters \( \Sigma, \Delta \) is inspired by the notation classically used for polynomials, for example in \([GK13, p.7]\).

**Remark 3.0.2** (Notational caveats). As we mentioned in 1.0.8 our notation, while being very telling, sometimes hides pieces of data. For example one finds that \( \hat{\mathcal{U}}\Sigma\hat{\mathcal{U}} \cong \hat{\mathcal{U}} \times \hat{\mathcal{U}} \). This is an instance of the fact that, depending on the choice of maps along which one performs the pullbacks (and depending on the order in which one does so), one gets a classifier that is either nested, or it is not. In general, the *nesting degree* is subject to change. Such equations, though unpretty, will be interesting from the point of view of the theory. Each time something of this kind happens, we will state it explicitly.

### 3.1. From natural models to plain dtts

Recall that a natural model in the sense of \([Awo18]\) is the data of

1. a category \( \text{ctx} \) with terminal object;
2. an arrow \( p : \hat{U} \to U \) in the presheaf category \( \text{Psh(ctx)} \);
3. some *representability data*. This means that for all cospans as in the diagram below, we are given an object \( \Gamma.A \in C \), a morphism \( \delta_A : \Gamma.A \to \Gamma \) in \( \text{ctx} \) and an arrow \( q_A : \mathcal{K}(\Gamma.A) \to \hat{U} \), such that the square below is a pullback.

\[
\begin{array}{ccc}
\mathcal{K}(\Gamma.A) & \xrightarrow{q_A} & \hat{U} \\
\downarrow & & \downarrow \\
\mathcal{K}(\delta_A) & \xrightarrow{p} & \hat{U} \\
\downarrow & & \downarrow \\
\mathcal{K}\Gamma & \xrightarrow{A} & U
\end{array}
\]

**Remark 3.1.1** (Use of the Yoneda lemma). When working with natural models, the Yoneda lemma is heavily used and, in particular, for a presheaf \( X \) over \( \text{ctx} \) we tend to identify objects that are in a correspondence under the following (natural) bijection.

\[
X(\Gamma) \cong \text{NatTr}(\mathcal{K}\Gamma, X)
\]

When we want to avoid using such an abuse, we denote with \( x \) an element of \( X(\Gamma) \) and \( x^* \) its corresponding natural transformation and, conversely, for a natural
transformation \( y \) we call \( y^* \) its corresponding element. One of the advantages of dealing with judgemental theories is that such an ambiguity will be avoided entirely.

**Theorem 3.1.2.** A natural model is the same thing as a plain dependent type theory where the types and terms fibrations are discrete fibrations.

The greatest part of the theorem relies on the following result. Recall that with respect to a discrete fibration, all maps are cartesian, hence whatever unit and counit we supply, their component will be, too.

**Proposition 3.1.3.** Let \( p : \hat{U} \rightarrow U \) a morphism of presheaves over \( \text{ctx} \) and \( \Sigma_p \) its image through the Grothendieck biequivalence restricted to presheaves. The following are equivalent.

1. We are provided with some representability data for \( p \).
2. The functor \( \Sigma_p \) has a right adjoint \( \Delta_p \).

**Remark 3.1.4.** It is evident from the discussion between page 245 and 246 of [Awo18] that Awodey was aware of this result, but because he only sketches the proof of the proposition above, we provide it in full.

**Proof of 3.1.3.** First of all, let us briefly describe \( \Sigma_p : \hat{U} \rightarrow U \) in terms of \( p : \hat{U} \rightarrow U \), or at least how it acts on the objects. The category \( \hat{U} \) has for objects pairs \((\Gamma, a)\) where \( \Gamma \) is an object of \( \text{ctx} \) and \( a \in \hat{U}(\Gamma) \). Similarly, the category \( U \) has for objects pairs \((\Gamma, A)\) where \( \Gamma \) is an object of \( \text{ctx} \) and \( A \in U(\Gamma) \). The presheaf morphism \( p \) induces a function \( p_\Gamma : \hat{U}(\Gamma) \rightarrow U(\Gamma) \), therefore the (discrete) fibration morphism \( \Sigma_p \) it induces maps a pair \((\Gamma, a)\) to \((\Gamma, p_\Gamma(a))\). We denote it \( \Sigma_p \) in analogy with Definition 3.0.1.

(1 \( \Rightarrow \) 2) We will now construct the functor \( \Delta_p \) provided that \( p \) is representable.

\[
\begin{array}{c}
\hat{U} \\
\downarrow^u \\
\text{ctx} \\
\downarrow_u \\
U \\
\end{array}
\]

Consider an object \( A \in U \), recall that it corresponds by Remark 3.1.1 to an arrow \( A^* : \mathbb{k}uA \rightarrow U \). Then, we define \( \Delta_p : A \mapsto (q_A)^* \), where the latter is obtained by the representability condition at \( A^* \). On a substitution \( \sigma : \Theta \rightarrow \Gamma \) we take pullbacks as depicted below.
We now need to show that $\Sigma_p \dashv \Delta_p$. The easiest thing is to provide the unit and the counit.

($\epsilon$) We want to construct an arrow $\epsilon_A : \Sigma_p \Delta_p A \to A$. We define it to be the cartesian lifting of $\delta_A$ at $A$. Now we need to show that this is a natural transformation, but that follows from the universal property of cartesian lifts. In fact, for each $s : B \to A$, the composition $s \circ u^* \delta_B$ is the cartesian lifting of $\sigma \circ \delta_B$, $u^* \delta_A \circ \Sigma_p \Delta_p s$ that of $\delta_A \circ \Delta_p (\sigma)$, and $\delta_A \circ \Delta_p (\sigma) = \sigma \circ \delta_B$, therefore, by uniqueness (up to iso) of the cartesian lifting, $u^* \delta_A \circ \Sigma_p \Delta_p s = s \circ u^* \delta_B$ too.

($\eta$) We want to construct an arrow $a \to \Delta_p \Sigma_p a$. This is also obtained by cartesian lifting, that of $\gamma_a$ induced by the dotted arrow in the diagram below.

\[
\begin{array}{ccc}
\vdash \Gamma & \overset{a}{\longrightarrow} & A \\
\downarrow \rotatebox{90}{$\scriptstyle \epsilon$} & & \downarrow \\
\vdash \Gamma ((\Gamma, A)) & \overset{\eta_A}{\longrightarrow} & \hat{U} \\
\downarrow \rotatebox{90}{$\scriptstyle id$} & & \downarrow p \\
\vdash \Gamma (\delta_A) & \overset{\sigma}{\longrightarrow} & \hat{U} \\
\downarrow \rotatebox{90}{$\scriptstyle id$} & & \downarrow \\
\vdash \Gamma & \longrightarrow & A \\
\end{array}
\]

Naturality follows as for $\epsilon$.

Triangle identities of ($\epsilon, \eta$) lie above commutative diagrams, for $\Sigma_p$ and $\Delta_p$ respectively

\[
id_\Gamma = \delta_A \circ \gamma_a \quad \text{and} \quad id_{\Gamma, A} = \delta_{A\delta_A} \circ \gamma_{\sigma_A},
\]

therefore they are satisfied again by uniqueness of the cartesian lifting.

(2) $\Rightarrow$ (1) The diagram below describes the representability data.

\[
\begin{array}{ccc}
\vdash (u \Sigma_p \Delta_p A) & \overset{\Delta_p A}{\longrightarrow} & \hat{U} \\
\downarrow \rotatebox{90}{$\scriptstyle \phi$} & & \downarrow \\
\vdash (u(\epsilon_A)) & \overset{p}{\longrightarrow} & \hat{U} \\
\downarrow \rotatebox{90}{$\scriptstyle \epsilon$} & & \downarrow \\
\vdash \Gamma & \longrightarrow & A \\
\end{array}
\]

It is a pullback by the universal property of $\epsilon$: for each pair $(\sigma, b)$ such that $A \circ \sigma = p \circ b$, there is a map

\[
s : p \circ b = \Sigma_p (b) \to A
\]

induced by precomposition with $\sigma$. Therefore there must be a unique $\phi$ such that $\epsilon \circ \Sigma_p \phi = s$. Now $u(\Sigma_p \phi)$ gives the desired map into $u \Sigma_p \Delta_p A$.

\[\square\]

Proof of 3.1.2. There is a clear correspondence between couples $(\text{ctx}, p : \hat{U} \to U)$ and triangles as below, where $\hat{u}, u$ are discrete fibrations. The correspondence is given by the Grothendieck construction.
The additional axioms required on both ends are equivalent because of Proposition 3.1.3.

3.2. **Plain dtts vs comprehension categories.** Another categorical approach to dependent type theories which is historically very meaningful was given by Jacobs in [Jac99]. This is the theory of comprehension categories and it is inherently presented in the form of a pre-judgemental theory as below.

Comprehension categories clearly realize some form of context extension, and that is given by display maps.

**Construction 3.2.1 (From pDTTs to comprehension categories).** Each plain dependent type theory produces a comprehension category as described by the steps below.

It is enough to follow the picture from left to right (and top to bottom) to see how a plain dependent type theory in our sense produces a display functor, which thus specifies a comprehension category.

Of course it is a legitimate question to ask whether every comprehension category can be realized via a plain dependent type theory. Turns out that the two are in fact equivalent, and to prove such a thing is the starting point of [CE24].

3.3. **Dictionary.** Dependent type theory has a well established notation, which we switch to in this subsection. The table below declares the dictionary between our framework and the classical notation.

Following the presentation in Section 2.1, it will need to take into account Syntax (but there is not much to say there), Judgements, and Rules. What we adopt here is a one-to-one rewriting of (some) components introduced in Section 2 in order to make the calculations we will see more transparent. Still, each string of symbols will simply represent its categorical backbone.
Dictionary for judgements. As we mentioned in Definition 3.0.1, we think of \( U \) as classifying types, \( \mathcal{U} \) as classifying terms, and of \( \Sigma \) as performing the typing. We make this clear with the choices in the translation that follow. (Sometimes we might omit the word \textit{Type} for brevity).

\[
\begin{array}{c}
\Gamma \vdash A U \\
\Gamma \vdash a \mathcal{U} \quad (\Gamma \vdash A U) \quad \Gamma \vdash \Sigma a =_U A
\end{array}
\quad \quad
\begin{array}{c}
\Gamma \vdash A Type \\
\Gamma \vdash a : A
\end{array}
\]

\[
\Gamma \vdash A =_U B \quad (\Gamma \vdash A U) \quad \Gamma \vdash B U
\]

\[
\begin{array}{c}
\Gamma \vdash a =_U b \quad (\Gamma \vdash A U) \quad \Gamma \vdash \Sigma a =_U A \quad \Gamma \vdash \Sigma b =_U A
\end{array}
\]

\[
\Gamma \vdash a =_U b : A
\]

Remark 3.3.1 (How many types to a term?). One might see our choice in the treatment of typing as profoundly Church-like, in the sense that to one term we only assign one type via the functor \( \Sigma \), and that is far from the practice. The generality of our definition, though, allows for some tweaks, so that if one wishes to have the possibility of assigning different types to the same term (say both \( 0 : N \) and \( 0 : Z \)) one can simply choose \( \mathcal{U} \) as a subcategory of two categories with, respectively, names for terms and for types (hence code the two above as \((0, N)\) and \((0, Z)\)), and make \( \Sigma \) act as a second projection.

Dictionary for rules. We also have a dictionary for rules, which we have (at least) two of. The first is implicitly used in Section 3.3.1, and it is the typing.

\[
\begin{array}{c}
\mathcal{U} \xrightarrow{\Sigma} U \\
\mathcal{U} \xrightarrow{\Sigma} \mathcal{U}
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash a : \Sigma a \quad (\Sigma) \\
\Gamma \vdash a \mathcal{U} \quad (\Sigma)
\end{array}
\]

The second is the policy \( \delta \) from Construction 3.2.1, which we here denote as follows.

\[
\begin{array}{c}
\mathcal{U} \xrightarrow{\delta} \mathcal{U} \\
\mathcal{U} \xrightarrow{\delta} \mathcal{U}
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash A \mathcal{U} \quad (\delta) \\
\Gamma \vdash q A : \Sigma \Delta A
\end{array}
\]

\[
\begin{array}{c}
\Gamma \vdash A Type \quad (\delta) \\
\Gamma \vdash a =_U b : A
\end{array}
\]

Context extension and type dependency. In this subsection we compute some rules that are automatically deduced by the finite-limit closure of a plain dependent type theory. As we will see, they correspond to some very well known rules in dependent type theories.

Context extension in a DTT, explicitly.

Notation 3.4.1. For readability reasons, and in order to highlight the correspondence between the logic and the categories without trivializing it, we denote \( A \sigma \) the result of the cartesian lifting of \( A \) along \( \sigma \) and \( A[\sigma] \) the substitution in the sense of the type theory.

All of the pieces appearing in the dictionary 3.3.2 surely do look familiar to the type-theorist reader, all but one, and that is \( \Sigma \Delta A \). In fact one might rightfully ask how to compute such an object.

Proposition 3.4.2 (On a formal emergence of substitution). Let \( A \) be on object in \( \mathcal{U} \). Then, \( \Sigma \Delta A = A \delta A \), in the sense of Section 2.5.
Proof. We know that there is an arrow $\epsilon_A : \Sigma \Delta A \to A$. By the discussion in Section 2.5, the thesis is equivalent to the fact that the cartesian lifting of $\delta_A$ along $u$ is precisely $\epsilon_A$. Recall, that $\delta_A$ is by definition $u(\epsilon_A)$, therefore it is a lifting. It is cartesian by assumption. \qed

Notice that this is as trivial as (and in fact it amounts to) proving that the process of computing weakening can be simulated in the syntax using substitution, provided that suitable substitution rules do in fact exist. We can re-read the rule hidden in the policy $\delta$ as follows.

\[
\begin{align*}
\Gamma &\vdash A \ U & \quad \text{(DTm)} \quad \Gamma &\vdash a : A, \Gamma.A \vdash b : B \\
\delta &\quad \quad \Gamma &\vdash B[a] \ Type & \\
\delta &\quad \quad \Gamma &\vdash a : A, \Gamma.A \vdash b : B \\
\delta &\quad \quad \Gamma &\vdash A \ Type \\
\delta &\quad \quad \Gamma &\vdash q_A : A\delta_A \\
\end{align*}
\]

Finally, we observe that the deductive rule on the right is a version of context extension in dependent type theory.

3.4.2. Type dependency in a DTT, explicitly. Similarly to the case of context extension, in a pDTT as in Definition 3.0.1 the most classical instances of type dependency emerge too. Let us produce the following two rules.

\[
\begin{align*}
\Gamma &\vdash a : A & \quad \Gamma &\vdash B[a] \ Type & \\
\delta &\quad \quad \Gamma &\vdash A \ Type \\
\delta &\quad \quad \Gamma &\vdash a : A, \Gamma.A \vdash b : B \\
\end{align*}
\]

In order to do so, we first need (nested) classifiers for the premises. More generally, with an iterated construction we will code composed judgements of the form below.

\[
\begin{align*}
\Gamma &\vdash a : A, \Gamma.A \vdash b : B \\
\end{align*}
\]

This is achieved as follows.

\[
\begin{align*}
\dot{U} \ U \Sigma \Delta \dot{U} &\longrightarrow U \Delta \dot{U} &\rightarrow U \\
\downarrow &\quad \dot{U} \ U \Sigma \Delta \dot{U} &\rightarrow U \ U \Delta \dot{U} &\rightarrow U \\
\downarrow &\quad \dot{U} \ U \Sigma \Delta \dot{U} &\rightarrow U \ U \Delta \dot{U} &\rightarrow U \\
\downarrow &\quad \dot{U} \ U \ U \Sigma \Delta \dot{U} &\rightarrow U \ U \ U \Delta \dot{U} &\rightarrow U \\
\downarrow &\quad \dot{U} \ U \ U \ U \Sigma \Delta \dot{U} &\rightarrow U \ U \ U \ U \Delta \dot{U} &\rightarrow U \\
\end{align*}
\]

For example, the fibration on $U \Delta \dot{U}$ classifies pairs $(A, B)$ of types such that $u \Sigma \Delta(A) = u(B)$. This is precisely the composed judgement $\Gamma \vdash A, \Gamma.A \vdash B$.

**Lemma 3.4.3** (Focus on $U \Delta \dot{U}$). In a plain dtt we have the following rules and policy.

\[
\begin{align*}
U \Delta \dot{U} &\longrightarrow U \Delta\dot{U} &\rightarrow U \\
\downarrow &\quad U \Delta \dot{U} &\rightarrow U \Delta \dot{U} &\rightarrow U \\
\downarrow &\quad U \Delta \dot{U} &\rightarrow U \Delta \dot{U} &\rightarrow U \\
\downarrow &\quad U \Delta \dot{U} &\rightarrow U \Delta \dot{U} &\rightarrow U \\
\end{align*}
\]

\[
\begin{align*}
(\Gamma \vdash A, \Gamma.A \vdash B) &\longrightarrow (\Gamma \vdash A) &\rightarrow (\Gamma.A \rightarrow \Gamma) \\
\end{align*}
\]
**Proof.** This is the first detailed instance of two judgement classifiers supported by the same category, since one could perform the two following compositions

$$
\begin{array}{ccc}
\mathcal{U} \Delta \mathcal{U} & \longrightarrow & \mathcal{U} \\
\downarrow \circ \delta & & \downarrow \circ \delta \\
\mathcal{U} & \longrightarrow & \text{ctx} \\
\downarrow \circ \delta & & \downarrow \circ \delta \\
\text{ctx} & \longleftarrow & \text{ctx}
\end{array}
$$

which are related as discussed in Section 3.4.3. Such a policy is the symptom of a shift in perspective: on the upper path, one travels along the pullback diagram above, therefore the context which one lands on is $\Gamma.A$; on the lower, one is concerned with the “original” context of $A$, therefore getting to $\Gamma$. They are related, as we have thoroughly discussed, by $\delta_A$. Notice that the lower path, being a composition of fibrations, is a fibration as well. □

Since the classifier in the lower part of the diagram in 3.4.3 will play an important role in a later discussion, we name it, $v : \mathcal{U} \Delta \mathcal{U} \rightarrow \text{ctx}$.

In [Awo18, Prop. 2.2] there is the construction of a presheaf $P(\mathcal{U})$, with $P$ a polynomial functor, classifying the same nested judgement as $\mathcal{U} \Delta \mathcal{U}$. The polynomial is defined as follows

$$P = P_p : \text{Psh}(\text{ctx}) \xrightarrow{\Upsilon^*} \text{Psh}(\text{ctx}) / \mathcal{U} \xrightarrow{\Pi_p} \text{Psh}(\text{ctx}) / \mathcal{U} \xrightarrow{\Sigma_U} \text{Psh}(\text{ctx})$$

meaning the pullback along the terminal presheaf morphism from $\Upsilon$, followed by the right adjoint to pullback along $p$, followed by composition with the terminal from $\mathcal{U}$. We apologize for the ambiguous notation ($\ circ \cup$, $\Pi$, $\Sigma$), but we promise this will only be used in the current section.

**Lemma 3.4.4** (Classifiers à la Awodey). One can show that the fibration $v : \mathcal{U} \Delta \mathcal{U} \rightarrow \text{ctx}$ is precisely the projection $\pi : \Upsilon \rightarrow P(\mathcal{U}) \rightarrow \text{ctx}$.

**Proof.** We sketch the identity fiber-wise. At each $\Gamma$, $(\mathcal{U} \Delta \mathcal{U})_\Gamma$ is comprised of pairs $(A, B)$ with $A, B$ in $\mathcal{U}$ such that $u(B) = u(\Sigma \Delta A)$ and $u(A) = \Gamma$. By Remark 3.1.1, such $A$ and $B$ correspond to $A^*$ and $B^*$ fitting in the following diagram,

$$
\begin{array}{ccc}
U & \xleftarrow{B^*} & \Upsilon^* \Gamma.A \\
\downarrow \circ \delta & & \downarrow \circ \delta \\
\Upsilon^* \Gamma & \longrightarrow & \Upsilon \\
\downarrow \circ \delta & & \downarrow \circ \delta \\
\Gamma & \longrightarrow & U
\end{array}
$$

with the central square being a pullback by representability of $p$. Using a result from [DT87], [Awo18, Prop. 2.2] shows that such diagrams are in a 1-to-1 correspondence with maps of the form $\Upsilon \Gamma \rightarrow P(\mathcal{U})$, hence with elements of $P(\mathcal{U})(\Gamma)$. □

We have shown that there is a very tight connection between our classifier and Awodey’s. We hope that, though almost tautological, this result can convince the reader about the advantages of our construction, as it makes it much easier to predict the correct pullback that constructs the desired classifier (this will be more and more evident in the following sections), while it might not be always easy to find
suitable (polynomial) functors to classify complex judgements. Also, we can avoid the complex machinery of polynomial functors (and, in this case, the conflicting notation).

In order to provide the rules (DTm) and (DTy) we build a map out of \( \hat{U} \Sigma \Delta U \) (and of \( \hat{U} \Sigma \Delta U \)), and all we have is \( \Sigma, \Delta, \eta, \epsilon \), finite limits closure, composition, substitution, whiskering, and \( \# \)-lifting. A few tries lead us to the following choice.

We call \( \eta' : \hat{u} \Rightarrow \hat{u} \Delta \Sigma \) the natural transformation induced by \( \eta \) via Requirement 1.0.15 and apply \( \# \)-lifting (Requirement 1.0.14) as on the left. Write \( \pi \) for projections.

When we compute each lifting, we see that the policy \( (u^* \eta') \) computes, starting from a pair \((a, b)\) some new term in context \( \Gamma \), while the policy \( (u^* \eta') \) matches to a pair \((a, B)\) a new type in context \( \Gamma \).

We give each a meaningful name, that is, extensively:

\[
(\hat{u}^* \text{id})(a, b) = (a, [a]),
\]

\[
(u^* \text{id})(a, B) = (a, B[a]).
\]

Notice that the typing is appropriate due to the action of the vertical \( \Sigma \).

We are now one step away from having (DTy) and (DTm), and in fact the distance between the policies \( u^* \text{id}, \hat{u}^* \text{id} \) and the desired rules is extremely subtle, and one could argue - though the author might disagree - a merely technical one: on the premise of, say, dependent typing, we now have the following nested judgement (which we write in our original notation for judgemental theories, so that we can make the difference evident)

\[
\Gamma.A \vdash (a, B) \quad \hat{U} \Sigma \Delta U (u \circ \hat{u} \Delta \Sigma . u)
\]

while we wish to have the pair stand over \( \Gamma \). That is achieved by \( v \) (that from Lemma 3.4.3),

\[
\Gamma \vdash (a, B) \quad \hat{U} \Sigma \Delta U (v),
\]

therefore we need to adjust the two policies accordingly. We can do that by regular 2-categorical manipulations attaching \( v \) (the composition of the colored arrows below) to the diagram above.
The policy on the right now is (DTy). One could repeat a similar argument for terms, which again have the correct typing because of the action of $\Sigma$ in the $\bullet$-lifting above.

**Remark 3.4.5** (Similarities between DTy and proof theoretic Cut). In the next section we highlight a remarkable connection between dependent typing and the cut rule from natural deduction: we redirect the reader to Remark 4.3.5 for more information.

3.4.3. **Substitution along display maps.** Of course there are (at least) two interesting natural transformations that we know of insisting on $\check{u}$. If $\eta$ is so interesting, one might wonder what repeating the process discussed in Section 3.4.2 with $\epsilon$ might bring. We have a hint about its outcome, and that is given by the $\delta$ from Construction 3.2.1, still we compute it precisely.

We call $\epsilon' : \check{u}\Delta\Sigma \Rightarrow \check{u}$. The construction detailed here, when explicitly computed, induces the two following rules involving $\delta_A : \Gamma.A \rightarrow \Gamma$,

\[
\check{u}^*id(a,a') = (a,a'\delta_A)
\]
\[
\check{u}^*id(a,A') = (a,A'\delta_A)
\]

meaning we can transport terms and types along arbitrary display maps, given that they insist on the same context.

\[
\Gamma \vdash a : A \quad \Gamma \vdash a' : A' \quad \Gamma.A \vdash a'\delta_A : A'\delta_A
\]

**Remark 3.4.6** (Rules for free). Since we now have rules involving the unit and rules involving the counit of an adjunction, we can exploit their relation to one another and show once again the computational power of judgemental theories. In particular, the (bases of the) constructions in Section 3.4.2 and Section 3.4.3 are related by the triangle identities:

\[
\check{u} \xrightarrow{\Sigma} u \xrightarrow{u} \text{ctx}
\]

\[
\check{u} \xrightarrow{\Sigma} u \xrightarrow{\Delta} \check{u} \xrightarrow{\check{u}} u \xrightarrow{u} \text{ctx}
\]

so that whiskering the two $\bullet$-liftings above to compute $\epsilon$ after $\eta$ yields the functor $\check{u}$. Then at each level we have the same relation. Therefore

\[
(A'\delta_A)[a] = A' \quad \text{and} \quad (a'\delta_A)[a] = a',
\]
or, explicitly, we have the following rule

\[
\begin{array}{c}
\frac{u \times \bar{u} \xrightarrow{\bar{a} \cdot \text{id}} \Sigma \Delta \bar{u} \xrightarrow{\bar{a} \cdot \text{id}} \bar{u} \times \bar{u}}{
\Gamma \vdash a : A \quad \Gamma \vdash a' : A'}
\end{array}
\]

which we did not know before. Such a rule is an instance of the discussion in Remark 2.4.2. Of course we cannot say the same for the opposite composition, but that is telling all in itself.

3.5. Dependent type theories with \(\Pi\)-types.

**Definition 3.5.1 (\(\Pi\)-types).** A plain dependent type theory with \(\Pi\)-types is a pDTT as in Definition 3.0.1 having two additional rules \(\Pi\), \(\lambda\) such that the diagram below is commutative and the upper square is a pullback.

\[
\begin{array}{ccc}
\Sigma \cdot (\Sigma \cdot u \cdot a) & \xrightarrow{\lambda} & \bar{u} \\
\downarrow & & \downarrow \\
\bar{u} & \xrightarrow{\pi} & \bar{u}
\end{array}
\]

Recall that \(v\) is that from Lemma 3.4.3. The rest of this subsection is devoted to showing that the proof theory generated by such a judgemental theory actually meets our intuition for having \(\Pi\)-types.

3.5.1. \(\text{\textit{A la Martin-Löf}}\). Having \(\Pi\)-types in the sense of [Mar75] means to implement the following rules,

\[
\begin{array}{l}
\text{(II')} \quad \Gamma \vdash A \quad \Gamma, A \vdash B \\
\quad \Gamma \vdash \Pi_A B \\
\quad \Gamma \vdash f : \Pi_A B \\
\quad \Gamma \vdash a : A \\
\quad \Gamma \vdash f(a) : B[a]
\end{array}
\]

\[
\begin{array}{l}
\text{(III)} \quad \Gamma \vdash A \quad \Gamma, A \vdash b : B \\
\quad \Gamma \vdash \lambda_A b : \Pi_A B \\
\quad \Gamma \vdash (\lambda_A b)(a) = b[a] : B[a]
\end{array}
\]

plus their congruence with definitional equality.

\[
\begin{array}{l}
\text{(II'')} \quad \Gamma \vdash A = A' \quad \Gamma, A \vdash B = B' \\
\quad \Gamma \vdash \Pi_A B = \Pi_A' B' \\
\quad \Gamma \vdash \lambda_A b = \lambda_{A'} b' : \Pi_A B \\
\quad \Gamma \vdash a = a' : A \\
\quad \Gamma \vdash f(a) = f'(a') : B[a]
\end{array}
\]

The first two rules are almost evident in the very definition of dependent type theory with \(\Pi\)-types, while the other rules will be derived by the limit closure of the class of judgements and rules.

(II') Type formation is precisely the rule (II) in the sense of Section 2.4 and Section 3.3.1, indeed \(\bar{u} \cdot \Sigma \Delta \bar{u}\) classifies precisely the premises of (II').

(III) Similarly, the introduction rule is precisely the rule (\(\lambda\)) in the sense of Section 2.4 and Section 3.3.2, where the commutativity of the diagram ensures the correct typing for the term.
In order to express the elimination rule, we first need to code its premise, that is the nested judgement

\[ \Gamma \vdash f : \Pi A B \quad \Gamma \vdash a : A. \]

Notice that, because of (II F), this is actually silent of two judgements, meaning it should read

\[ \Gamma \vdash A \quad \Gamma.A \vdash B \quad \Gamma \vdash f : \Pi A B \quad \Gamma \vdash a : A, \]

instead, so that this is really the judgement we need to give a classification of. One can check that \( \dot{\mathcal{U}}.\Sigma(\dot{\mathcal{U}}.\Delta\dot{\mathcal{U}}) \) classifies the first, second, and fourth judgement appearing above. Also, we know from Section 3.4.2 that \( \dot{\mathcal{U}}.\Sigma(\dot{\mathcal{U}}.\Delta\dot{\mathcal{U}}) \cong \dot{\mathcal{U}}.\Sigma\Delta\dot{\mathcal{U}}. \) This is an instance of Remark 3.0.2, and it just expresses the fact that, whenever we have a term \( a : A \), we really have its type in our code already.

To now introduce the term \( f \), we need to perform one more pullback. We attach the diagram above to that in Definition 3.5.1. We are entitled to do so because, by hypothesis, the square that \( \Pi \) and \( \lambda \) fit in has the correct map on its left. For brevity, and since it should not cause much trouble, for the remainder of the proof we call all “horizontal” projections \( \pi \), and all “vertical” ones \( \Sigma \).

\[
\begin{array}{ccc}
\dot{\mathcal{U}}.\Sigma\Delta\dot{\mathcal{U}} & \xrightarrow{\pi} & \dot{\mathcal{U}}.\Delta\dot{\mathcal{U}} \\
\downarrow \Sigma & & \downarrow \Sigma \\
\dot{\mathcal{U}} & \xrightarrow{\Delta} & \dot{\mathcal{U}} \\
\end{array}
\]

To express the classifier for the whole premise, then, is to compute the pullback against \( \Sigma \) of the composition of \( \Pi \) and \( \pi \) in the lower part of the diagram. Call \( \Pi' = \Pi \circ \pi \). The premise of (E) is then classified by \( (\dot{\mathcal{U}}.\Sigma\Delta\dot{\mathcal{U}})\Sigma.\Pi'\dot{\mathcal{U}} \). We can see how it all builds up in the following suggestive writing

\[
(\dot{\mathcal{U}}.\Sigma\Delta\dot{\mathcal{U}})\Sigma.\Pi'\dot{\mathcal{U}} \quad (a.(A.B)) \cdot f
\]

which is fibered over \( \Gamma \); though not all of its components are types or terms specifically in context \( \Gamma \), every judgement appearing in this nested one is built out of a construction performed entirely in \( \Gamma \).

From now on, we will write all \( n \)-uples as above as traditional \( n \)-uples, since all pullbacks are subcategories of a product after all.

\[
(\dot{\mathcal{U}}.\Sigma\Delta\dot{\mathcal{U}})\Sigma.\Pi'\dot{\mathcal{U}}
\]
We now have two pullbacks insisting on the same cospan, then necessarily it is

\[(\dot{\mathcal{U}} \cdot \Sigma \Delta \dot{\mathcal{U}}) \Sigma \Pi' \dot{\mathcal{U}} \cong \dot{\mathcal{U}} \cdot \Sigma \Delta \dot{\mathcal{U}}.\]

This in not an instance of Remark 3.0.2, though, and the isomorphism above actually turns out to contain all the information needed to provide rules (E) and (β), and then some.

Clearly there is always a map going from right to left, just consider:

\[(a, b) \mapsto (A, B, \lambda_A b, a),\]

but Eq. (1) is adding three more pieces of information, meaning

(i) there is also a map going from left to right, (though we can always expand information, only this tells us we can compact it);

(ii) starting from the left, going right, and back left again, yields the identity;

(iii) starting from the right, going left, and back right again, yields the identity.

Of these, (i) will induce elimination and (iii) β-computation. The additional piece in (ii) will tell us something about what is generally called the η-rule, which is much more controversial. We will discuss it in detail in Section 3.5.3.

Call ζ and θ the inverse maps. A little calculation shows that they act as follows:

\[\theta : (a, b) \mapsto (A, B, \lambda_A b, a), \quad \zeta : (A, B, f, a) \mapsto (a, f_B),\]

where we write \(f_B\) for the term of type \(B\) in the second component of \(ζ\). Broadly speaking, \(θ\) computes introduction (this is evident by \(\lambda \circ \pi = \pi \circ \theta\)) and \(ζ\) elimination (both because of its typing and because we say so).

Before we can provide an explicit representation for the missing rules, we shall be able to account for writings \(b[a]\) and \(B[a]\). In order to do that, we need to use the diagram in Section 3.4.2. We paste it to the previous one as follows, calling

\[\gamma = \pi \circ \dot{u} \ast id \quad \text{and} \quad \dot{\gamma} = \pi \circ \dot{u} \ast id.\]

Notice that the map \(\Sigma : \dot{\mathcal{U}} \cdot \Sigma \Delta \dot{\mathcal{U}} \to \dot{\mathcal{U}} \cdot \Sigma \Delta \dot{\mathcal{U}}\) is precisely that appearing in Section 3.4.2 and Section 3.4.3, so that both “rectangles” insist on the same functor. All solid squares are pullbacks, the dashed one is only commutative.

The functor \(\dot{\gamma} \zeta\) is the elimination rule, because to each quadruple it matches a term of the correct type. We call \(\dot{\gamma}(a, f_B) = f_B[a] =: f(a)\).
Computation $\beta$ amounts to proving that if we apply introduction, followed by elimination, we kind of get to the point we started from. This is a rule with codomain as in 2.3.6, therefore we show that identity on $\dot{U}.\Sigma\Delta\dot{U}$ equalizes the following pair of arrows, 

$$
\begin{array}{c}
\dot{U}.\Sigma\Delta\dot{U} \\
\xrightarrow{\theta} \\
(\dot{U}.\Sigma\Delta\dot{U})\Sigma.\Pi\dot{U} \\
\xrightarrow{\zeta} \\
\dot{U}.\Sigma\Delta\dot{U} \\
\xrightarrow{id} \\
\dot{U}
\end{array}
$$

On the upper path is computed $(\lambda_A b)(a)$, on the lower we get $b[a]$. The two paths equalize trivially. The desired rule is then 

$$id : \dot{U}.\Sigma\Delta\dot{U} \to \dot{U}.\Sigma\Delta\dot{U} = E(\dot{\gamma}\dot{\zeta}\dot{\theta}, \dot{\gamma}).$$

3.5.2. Of congruence rules involving definitional equality. In our dictionary in 3.3 we decided that definitional equality of types and terms should be interpreted as judgemental equality according to $u$ and $\dot{u}$, respectively, hence as identity of objects in the respective “universe” categories. This guarantees that rules $(\Pi F=)$, $(\Pi I=)$, $(\Pi E=)$ are automatically verified. Rule $(\Pi I=)$, also known as the $\xi$-rule, in particular, is not verified by all models, especially those that are more computationally oriented, such as Kleene realizability or game semantics: we are indeed quite extensional in our spirit, but we believe this is more of a choice that we are making than a constraint of judgemental theories, and that it would be interesting to further develop the theory with different, weaker, but still finite-limit stable interpretations of judgemental equality.

3.5.3. Of $\eta$ and elimination. The $\eta$-rule accounts for the need to determine what happens in the case that one wants to apply elimination followed by introduction, and at first it looks exactly as the dual of $(\Pi \beta C)$. While it is clear that $\beta$ should prescribe equality of two terms, though, there is actually no agreement on the features $\eta$ should present, so that in the literature we find instances of the resulting computation of $\eta$ as being a conversion (i.e. consisting of a definitional equality), interpreted as an expansion, or a reduction (meaning a non-symmetric relation whose reflexive, symmetric, and transitive closure defines the conversion). The virtue of each process, and each of its 2-categorical delivery, is the topic of [See86].

In our framework, Eq. (1) tells us something about which $\eta$-rule we should be looking at, and in fact we have 

$$
\frac{\Gamma \vdash f : \Pi AB}{\Gamma \vdash f = \lambda A(f_B) : \Pi AB}
$$

which is precisely what $\theta \zeta = id$ says. This is only one of the possible expressions for $\eta$, and it differs from that presented in [Awo18, p.253], which much more swiftly agrees with the tradition of categories with families. This is because, in a sense, we think the notion of elimination presented there, and in Section 3.5.1 above, is not the correct one: it really is $\zeta$ performing the elimination, and it really is $f_B$ the term witnessing it. It is not in the computation through $\dot{\gamma}$ that a term of type $\Pi AB$ turns into a term involving $A, B$. This argument, together with the possibility of excluding the $\eta$ rule entirely, will be made much more clear in Section 3.7.

3.6. Dependent type theories with (extensional) Id-types. For identity types we need to be able to consider pairs of terms of the same type, this is why we begin by pulling back $\Sigma$ against itself. Call $\pi_1, \pi_2$ the corresponding projections and $\text{diag} : \dot{U} \to \dot{U} \times \dot{U}$ the unique map such that $\pi_1 \circ \text{diag} = id = \pi_2 \circ \text{diag}$. 

$$
\begin{array}{c}
\dot{U} \\
\xrightarrow{\theta} \\
(\dot{U} \times \dot{U}) \Sigma.\Pi\dot{U} \\
\xrightarrow{\zeta} \\
\dot{U} \\
\xrightarrow{id} \\
\dot{U}
\end{array}
$$
Definition 3.6.1 (Extensional Id-types). A plain dependent type theory with extensional Id-types is a pDTT as in Definition 3.0.1 having two additional rules $\text{id}, i$ such that the diagram below is commutative and the upper square is a pullback.

\[
\begin{array}{c}
\tilde{U} \\
\downarrow \text{diag} \\
\tilde{U} \times \tilde{U} \\
\downarrow \text{id} \\
\text{ctx}
\end{array}
\]

\[
\begin{array}{c}
\tilde{U} \\
\downarrow \Sigma \\
\tilde{U}
\end{array}
\]

Again, the rest of the subsection is dedicated to showing that the proof theory generated by such judgemental theory actually meets our intuition for having Id-types. Classically, having Id-types means to implement the following rules:

- **(IdF)**: \( \Gamma \vdash A \text{ Type} \quad \Gamma \vdash a : A \quad \Gamma \vdash b : A \quad \Gamma \vdash \text{id}_A(a, b) \text{ Type} \)
- **(IdI)**: \( \Gamma \vdash a : A \quad \Gamma \vdash i(a) : \text{id}_A(a, a) \)
- **(IdE)**: \( \Gamma \vdash c : \text{id}_A(a, b) \quad \Gamma \vdash a = b : A \)
- **(Idη)**: \( \Gamma \vdash c : \text{id}_A(a, b) \quad \Gamma \vdash c = i(a) : \text{id}_A(a, a) \)

As it was for Definition 3.5.1, the first two rules are evident in the very definition of dependent type theory with Id-types.

- **(IdF)** Type formation is precisely the rule \((\text{id})\) in the sense of Section 2.4 and Section 3.3.1. Clearly $\tilde{U} \times \tilde{U}$ classifies the premises of \((\text{idF})\).
- **(IdI)** Similarly, the introduction rule is the rule \((i)\) in the sense of Section 2.4 and Section 3.3.2, where the commutativity of the diagram forces the correct typing for the term.

For elimination and conversion we need to pin-point a classifier for judgements of the form

\[ \Gamma \vdash c : \text{id}_A(a, b), \]

but since the square is a pullback insisting on the cospan \((\text{id}, \Sigma)\), such a feat is achieved by the \((\text{upper-left}) \tilde{U}\). Then not only do $\text{id}, i$ compute the appropriate term and type (below on the left), but they also act as projections (below on the right).

\[
\begin{array}{c}
\tilde{U} \\
\downarrow \phi \\
\tilde{U} \times \tilde{U}
\end{array}
\]

\[
\begin{array}{c}
\tilde{U} \\
\downarrow \Sigma \\
\tilde{U}
\end{array}
\]

\[
\begin{array}{c}
\tilde{U} \\
\downarrow \text{diag} \\
\tilde{U}
\end{array}
\]

\[
\begin{array}{c}
\text{id}_A(a, a) \\
\downarrow \Sigma \\
\text{id}_A(a, b)
\end{array}
\]

The object classifying judgements of the form $\Gamma \vdash a = b : A$, instead, is the equalizer $E(\pi_1, \pi_2)$. By its universal property there must be a unique $\phi$ making the following
diagram commute.

\[
\begin{array}{ccc}
\mathcal{E}(\pi_1,\pi_2) & \xleftarrow{\phi} & \mathcal{U} \\
\downarrow{\epsilon} & & \downarrow{\text{diag}} \\
\mathcal{U} \times \mathcal{U} & \xrightarrow{id} & \mathcal{U}
\end{array}
\]

(\text{IdE}) The elimination rule, then, is \( \phi : \mathcal{U} \rightarrow \mathcal{E}(\pi_1,\pi_2) \).

(\text{Id}\eta) The computation rule is computed as

\[
\mathcal{U} \rightarrow \mathcal{E}(\pi,\psi \pi_1 e \phi) = \mathcal{E}(\pi,\pi) = \mathcal{U}
\]

therefore it is the map \( id : \mathcal{U} \rightarrow \mathcal{U} \).

There would be a notion of \( \beta \)-computation (in the sense of introduction followed by elimination) here, too, but it is not usually written because it is trivial once one has definitional equality. In fact, it takes the following form.

\[
\text{(Id}_{\beta 1}\text{)} \quad \frac{\Gamma \vdash a : A}{\Gamma \vdash a = a : A} \quad \text{(Id}_{\beta 2}\text{)} \quad \frac{\Gamma \vdash a : A}{\Gamma \vdash i(a) = i(a) : \text{Id}_A(a,a)}
\]

3.7. **A categorical definition of extensional type constructor.**

**Definition 3.7.1** (The extensional type constructor \( \Phi \)). A plain dependent type theory with extensional \( \Phi \)-types is a \textit{pDTT} as in Definition 3.0.1 having two additional rules \( \Phi, \Psi \) such that the diagram below is commutative and the upper square is a pullback.

\[
\begin{array}{ccc}
\mathcal{X} & \xrightarrow{\psi} & \mathcal{U} \\
\downarrow{\Lambda} & & \downarrow{\Sigma} \\
\mathcal{Y} & \xrightarrow{\phi} & \mathcal{U} \\
\downarrow{\text{ctx}} & & \\
& & \mathcal{U}
\end{array}
\]

**Remark 3.7.2.** Notice that the definition is implicitly assuming that \( \Lambda \) belongs to the closure of the generators under finite limits. Also, it is evident by the previous sections that \( \Pi \)-types and \( \text{Id} \)-types fall under this definition.

The rest of the subsection is devoted to showing that the proof theory generated by such judgemental theory actually meets our intuition for having extensional \( \Phi \)-types.

\[
\begin{array}{c}
(\Phi F) \quad \frac{\Gamma \vdash Y \ Y}{\Gamma \vdash \Phi Y \text{ Type}} \\
(\Phi I) \quad \frac{\Gamma \vdash X \ X}{\Gamma \vdash \Psi X : \Phi \Delta X}
\end{array}
\]

(\text{\Phi F}) Type formation is precisely the rule (\( \Phi \)) in the sense of Section 2.4 and Section 3.3.1.

(\text{\Phi I}) Similarly, the introduction rule is precisely the rule (\( \Psi \)) in the sense of Section 2.4 and Section 3.3.2, where the commutativity of the diagram forces the correct typing for the term.
Now, because we have requested that the square in Definition 3.7.1 is a pullback, we automatically get the dashed functors below.

\[
\begin{align*}
\mathcal{Y} \Sigma \Phi & \xrightarrow{\text{dashed functors}} \mathcal{U} \\
\mathcal{X} \xrightarrow{\psi} \mathcal{U} & \xrightarrow{\Lambda} \mathcal{Y} \xrightarrow{\phi} \mathcal{U} \\
\mathcal{Y} \xrightarrow{\phi} & \mathcal{U} \xrightarrow{\Sigma} \\
\mathcal{Y} \xrightarrow{\psi} & \mathcal{U} \xrightarrow{\Lambda} \mathcal{Y} \xrightarrow{\phi} \mathcal{U} \\
\mathcal{X} \xrightarrow{\psi} & \mathcal{U} \xrightarrow{\Lambda} \mathcal{Y} \xrightarrow{\phi} \mathcal{U}
\end{align*}
\]

\((\Phi E)\) The rule associated to functor \((-)(-)\) gives us the elimination rule on the right. Indeed the pullback category precisely classifies the premises of \((\Phi E)\).

\[
\begin{align*}
\Gamma & \vdash a : \Phi Y \\
\Gamma & \vdash \Phi Y(a) : \mathcal{X}
\end{align*}
\]

By essential uniqueness of pullbacks, the compositions \(((\mathcal{X} \rightarrow \mathcal{Y}) \circ (\Lambda \star \Psi)) \circ ((\mathcal{X} \rightarrow \mathcal{Y}))\) both amount to the identity of the respective object. This observation provided by universal property of the equalizer induces the arrows \(\eta\) and \(\beta\) in the diagram below.

\[
\begin{align*}
\mathcal{X} & \xrightarrow{id} \mathcal{X} \xrightarrow{\psi} \mathcal{Y} \xrightarrow{\Phi \Sigma \Phi \mathcal{U}} \mathcal{U} \\
\mathcal{Y} \Sigma \mathcal{Y} & \xrightarrow{id} \mathcal{Y} \Sigma \mathcal{Y} \xrightarrow{\Phi \Sigma \Phi \mathcal{U}} \mathcal{U}
\end{align*}
\]

\((\Phi \beta)\) The rule associated to the functor \(\beta\) is our \(\beta\)-computation. Indeed, if we write down the rule explicitly we get the following.

\[
\Gamma \vdash X \mathcal{X}
\]

\[
\Gamma \vdash \Psi X = \Psi((\Lambda X)\psi X) : \Phi \Lambda X
\]

\((\Phi \eta)\) The rule associated to the functor \(\eta\) is our \(\eta\)-computation. Indeed, if we write down the rule explicitly we get the following.

\[
\Gamma \vdash a : \Phi Y
\]

\[
\Gamma \vdash a = \Psi(\Phi \mathcal{Y}(a)) : \Phi Y
\]

Additionally, and as in the case of dependent products in Section 3.5.2, we have rules guaranteeing that definitional equality of terms and types is “preserved” through formation, introduction, and elimination. See thereof for a discussion on possible variations.

**Remark 3.7.3** (Weaker notions of type constructors). Our definition of type constructor is very modular: for example, if we request that the square in Definition 3.7.1 is a weak pullback (as opposed to a pullback) with a distinguished section, we can still construct the functors \((-)(-)\) and \(\Lambda \star \Psi\), and one of the two compositions still amounts to the identity. This ensures both elimination and \(\beta\)-computation, while we lose \(\eta\)-computation. This remark generalizes a similar analysis contained in [Aw18, Cor. 2.5].

We believe that Definition 3.7.1 is more proof of both the computational and the expressive power of judgemental theories. We now use the construction above to enrich a pDTT with units and dependent sums. We reverse engineer the theory
in order to provide the correct definition, and that will be all that we need because of the calculations above. By the end of this paper, we will have shown that Definition 3.7.1 captures dependent products, dependent sums, unit types, extensional identity types. In addition, the construction in [Awo18, §2.4] might suggest that it fits intensional identity, too, but we do not discuss this further here.

Remark 3.7.4 (Other type constructors). We are indeed aware that Definition 3.7.1 does not capture all type constructors used in both the theory and the practice of type theory, for example it does not allow for the description of (co)inductive types, but we believe that our categorical theory of judgement has been proved fruitful in coding syntactic data. Clearly finite limits will capture finite constructions, but 2-category theory is much more than finite, nor it is only about limits, therefore we trust that with some effort this work could be extended to different constructors.

3.8. Examples: unit types and Σ-types.

3.8.1. Dependent type theories with unit types. Our aim is to describe the premises of introduction and formation, and the relation they are in. Recall that the rules in question are

\[
\begin{align*}
\text{(uI)} & \quad \Gamma \vdash 1\,\text{Type} \\
\text{(uF)} & \quad \Gamma \vdash * : 1
\end{align*}
\]

so that both only take in input a context, and the premises are identical, hence our motivation to give the following definition.

Definition 3.8.1 (Unit-types). A plain dependent type theory with unit-types is a pDTT having two additional functors 1, * such that the diagram below is commutative and the upper square is a pullback.

\[
\begin{array}{ccc}
\text{ctx} & \xrightarrow{*} & \hat{U} \\
\downarrow \text{id} & & \downarrow \text{id} \\
\text{ctx} & \xrightarrow{1} & U \\
\end{array}
\]

We now show that the judgemental theory generated by diagrams in Definition 3.8.1 contains codes for formation, introduction, elimination, and computation of unit types. Introduction and formation in fact read as follows

\[
\begin{align*}
\text{(1)} & \quad \Gamma \vdash \text{id} \\
\text{(*)} & \quad \Gamma \vdash \text{id}
\end{align*}
\]

or, in our more familiar writing

\[
\begin{align*}
\text{(uI)} & \quad \Gamma \vdash \text{ctx} \\
\text{(uF)} & \quad \Gamma \vdash * : 1
\end{align*}
\]

moreover, the elimination rule is captured by the unique map \( \psi : \text{ctx}.1\hat{U} \rightarrow \text{ctx} \) and it translates to the syntactic writing on the left, while postcomposed with * it translates as the more familiar rule on the right

\[
\begin{align*}
\text{(ψ)} & \quad \Gamma \vdash t : 1 \\
\text{(*) ◦ ψ)} & \quad \Gamma \vdash * : 1
\end{align*}
\]
which is also denoted (uE). Finally, computation \( \beta \) and \( \eta \) can be decoded from the two following diagrams

\[
\begin{array}{ccc}
\text{ctx} & \xrightarrow{\beta} & \emptyset \\
\text{Eq} & \xrightarrow{id} & \text{ctx}
\end{array}
\quad
\begin{array}{ccc}
\text{ctx} & \xrightarrow{\ast} & \emptyset \\
\text{Eq} & \xrightarrow{u \circ \psi \phi} & \text{ctx}
\end{array}
\quad
\begin{array}{ccc}
\text{ctx}\hat{\Sigma} & \xrightarrow{\beta} & \emptyset \\
\text{Eq} & \xrightarrow{id} & \text{ctx}\hat{\Sigma}
\end{array}
\quad
\begin{array}{ccc}
\text{ctx}\hat{\Sigma} & \xrightarrow{1.\Sigma} & \emptyset \\
\text{Eq} & \xrightarrow{1.\Sigma \circ \psi \phi} & \text{ctx}\hat{\Sigma}
\end{array}
\]

with \( \phi \) the inverse to \( \psi \), which read, respectively, as follows.

\[
\begin{array}{c}
\frac{\Gamma \vdash \text{ctx}}{\Gamma \vdash \text{ctx}} \\
\frac{\Gamma \vdash \text{ctx} \rightarrow \ast \psi \phi}{\Gamma \vdash \text{ctx} \rightarrow \ast \psi \phi}
\end{array}
\quad
\begin{array}{c}
\frac{\Gamma \vdash t : 1}{\Gamma \vdash t = 1_\Gamma}
\end{array}
\]

3.8.2. **Dependent type theories with \( \Sigma \)-types.** We hope the reader will forgive us if to avoid confusion we adopt the unusual notation of \( \text{rbag} \) instead of \( \Sigma \). We then start to look at rules for formation and introduction, which for sum types are usually the following.

\[
\frac{\Gamma \vdash A \text{ Type} \quad \Gamma \vdash B \text{ Type}}{\Gamma \vdash \text{rbag}(A) \text{ Type}}
\quad
\frac{\Gamma \vdash A \text{ Type} \quad \Gamma \vdash B \text{ Type}}{\Gamma \vdash (a, b) : \text{rbag}(A)_B}
\]

In order to classify the premise of \((\text{rbag} F)\) we simply use \( \mathcal{U}, \Delta \mathcal{U} \) from Section 3.4.2. The premise of \((\text{rbag} I)\), instead, can be coded via the following nested judgement classifier

\[
\frac{(\mathcal{U}, \Delta \mathcal{U}) \Sigma. \gamma \mathcal{U} \xrightarrow{\mathcal{U}} \mathcal{U}}{(\mathcal{U}, \Delta \mathcal{U}) \Sigma. \mathcal{U} \xrightarrow{\mathcal{U}} \mathcal{U}}
\]

\[
\frac{(\mathcal{U}, \Delta \mathcal{U}) \Sigma. \mathcal{U} \xrightarrow{\mathcal{U}} \mathcal{U}}{(\mathcal{U}, \Delta \mathcal{U}) \Sigma. \mathcal{U} \xrightarrow{\mathcal{U}} \mathcal{U}}
\]

with \( \gamma = \pi \circ u^* \text{id} \) from Section 3.4.2. The desired rule \( \Lambda \), then, is the functor \((\mathcal{U}, \Delta \mathcal{U}) \Sigma. \gamma \mathcal{U} \rightarrow \mathcal{U}, \Delta \mathcal{U} \) appearing above.

**Definition 3.8.2.** A plain dependent type theory with \( \text{rbag} \)-types is a pDTT as in Definition 3.0.1 having two additional rules \( \text{pair} \) such that the diagram below is commutative and the upper square is a pullback.

\[
\frac{(\mathcal{U}, \Delta \mathcal{U}) \Sigma. \gamma \mathcal{U} \xrightarrow{\mathcal{U}} \mathcal{U}}{(\mathcal{U}, \Delta \mathcal{U}) \Sigma. \mathcal{U} \xrightarrow{\mathcal{U}} \mathcal{U}}
\]

A pDTT with \( \text{rbag} \)-types immediately has formation and introduction (with \( \langle a, b \rangle = \text{pair}(a, b) \)) and, as follows from the content of Definition 3.7.1, the three (admittedly hard to look at) rules below. We write \( \Lambda \) for \( \Sigma.(u.\alpha \Delta) \circ \Sigma.\gamma \).
\[ \frac{\Gamma \vdash c : \prod_{A, B} (\prod_{\Delta \Sigma} \Sigma) \gamma \Delta \Sigma}{\Gamma \vdash \text{pair}(\prod_{\Delta \Sigma} \Sigma) \gamma \Delta \Sigma} \]

If we break down the job of the classifier \((\Delta \Sigma \Delta \Sigma) \Sigma \gamma \Delta \Sigma\) we recover the familiar following ones.

\[ \frac{\Gamma \vdash c : \prod_{A, B} (\prod_{\Delta \Sigma} \Sigma) \gamma \Delta \Sigma}{\Gamma \vdash \text{pair}(X) = \text{pair}(\prod_{\Delta \Sigma} \Sigma) \gamma \Delta \Sigma} \]

4. First-order logic

In this section we design the judgemental theory that performs the calculus of natural deduction. As for Section 3, we introduce the basic judgements and rules and show how they generate the desired structure, then we add more rules to perform additional computations. Though we follow the path of the well-known fibrational approach to first order logic, we spend some time in re-developing it in the context of judgemental theories: this is meant to present the benefits of the judgemental approach, to compare the resulting structure with that of dependent types, and to give a pedagogical example of how one might want to implement a judgemental theory starting from notions which are known to be fibrational in nature.

Remark 4.0.1 (Why we do not start from dependent type theory). As we discussed in Section 2.1.2, one could very well follow [Mar75] and use Section 3 as a starting point for this analysis by simply restricting it to the proof-irrelevant case. This is what is really happening in Definition 4.0.3 - and an explicit construction is actually provided in Remark 4.1.3 - but we choose to recover the whole theory from scratch for two reasons: on one hand, we hope that it makes the present work accessible to the non-(type theorist), or to someone who is more familiar with traditional first-order logic; on the other we aim to more swiftly align to the tradition of doctrines [Law70, Pit83, Mak93, MR13].

Again, as explained in 2.1.2, our distinction is mathematically artificial, and we will remark that throughout our discussion, see for example 4.3.5.

Remark 4.0.2 (Why we do not do Gentzen’s sequent calculus). On the other hand, we could have chosen to present first-order logic in the formalism of sequent calculus in [Gen35]. Though our framework allows us for it – and in fact many of the categorical constructions in the following section do so, already, starting from Definition 4.0.3 – we have chosen to take the perspective of natural deduction because on one hand we believe that, being closer to how logic is used makes it easier to follow what each categorical operation is doing and, secondly, dealing with
connective and quantifiers with pairs of introduction/elimination rules, as opposed to right/left introduction rules, helps to keep the connection with dependent types (4.0.1) in the back of the reader’s mind.

**Definition 4.0.3** (Natural deduction theory). A natural deduction theory is a substitutional (1.1.1) judgemental theory \((\text{ctx}, \mathcal{J}, \mathcal{R}, \mathcal{P})\) such that

- \(\text{ctx}\) is \(\text{Fin}\), the category of finite sets;
- \(\mathcal{J}\) can be presented by one judgement classifier \(p : \mathcal{P} \to \text{ctx}\), which is a faithful fibration, has fibered products and implication, and has fibered initial objects.

We think of \(\text{ctx}\) as the category of variables and terms and of \(\mathcal{P}\) as the category of well-formed formulae fibered over variables. We call this NDT for short.

**Remark 4.0.4** (On cardinality). We can define \(\lambda\)-ary theories but we would need to close judgemental theories under \(\lambda\)-small limits, and we would have to replace \(\text{Fin}\) with the category of \(\lambda\)-small sets.

**Remark 4.0.5** (From doctrines to classifiers). Let \(P : \text{ctx}^{\text{op}} \to \text{Pos}\) be a doctrine, intended in the most non-committal sense. Consider \(\square : \mathcal{P}^I \to \mathcal{P}\) any operational property/structure on \(P\), e.g.:

- having (finite) fibered meets \(\land : \mathcal{P}^I \to \mathcal{P}\);
- having (finite) fibered joins \(\lor : \mathcal{P}^I \to \mathcal{P}\);
- having a negation operator \(\neg : \mathcal{P} \to \mathcal{P}\).

then, by the Grothendieck construction, we obtain some corresponding diagram of fibrations,

\[
\begin{array}{ccc}
\mathcal{P}^I & \xrightarrow{\square} & \mathcal{P} \\
\downarrow & & \downarrow \\
\mathcal{P}^I & \xrightarrow{p} & \mathcal{P} \\
\downarrow & & \downarrow \\
\text{ctx} & & \text{ctx}
\end{array}
\]

This produces a pre-judgemental theory obtained by \(\mathcal{P}\), together with all its structural operators. For example, if \(P\) in an Heyting algebra fiber-wise, we have operators \(\perp, \top, \land, \lor, \Rightarrow\) of the proper arities on \(\mathcal{P}\).

**Remark 4.0.6** (The arrow category). Consider a NDT. Because it is closed under finite powers (Requirement 1.0.11) we can compute

\[
\mathcal{P}^2 \longrightarrow \mathcal{P}
\]

and we will show how the operations defined on \(\mathcal{P}\) lift to (a suitable subcategory of) \(\mathcal{P}^2\) thanks to the closure under finite limits. We will come back to this in Construction 4.1.2.

**Remark 4.0.7** (Weakening). Since \(p\) has fibered products we can compute the following nested judgement (on the left)
with an adjunction $\text{diag} \dashv \times$ whose counit computes projections. These are well known in the literature and perform what is usually called weakening:

$$x \times y \vdash x\ \text{ctx}.$$  

We can now define a span $(p^2, p^\times)$ out of $P^2$ (as a category, not as the fibration $P \times P$) with $p^\times : (\phi, \psi) \mapsto (\phi[\text{pr}_1] \land \psi[\text{pr}_2])$ the product of the respective cartesian lifts of $\phi, \psi$ along $\text{pr}_1, \text{pr}_2$,

$$\phi[\text{pr}_1] \land \psi[\text{pr}_2]$$  

and such a span makes the diagram involving $P^\times$ commute, therefore we have a unique rule

$$w : P^2 \to P^\times$$  

over $\text{ctx}^2$. If $p(\phi, \psi) = (x, y)$ we might denote $w(\phi, \psi) = w_y \phi \land w_x \psi$.

**Definition 4.0.8 (NDT with weakening).** A NDT is said to have weakening if for each $y \in \text{ctx}$, $\times \times y$ is in $\mathcal{J}$.

In this section we will show that, in fact, a NDT produces the calculus of natural deduction.

4.1. **Dictionary.** As we did in Section 3, we declare a local dictionary, both to make the paper more comprehensible and to account for classical notation.

**Notation 4.1.1 (Stratified contexts).** Notice that already in Remark 4.0.7 we follow the intuition and use $x, y, \ldots$ to name objects of $\text{ctx}$. In fact, we here want to give a way to present judgements that are traditionally of the form

$$x; \Gamma \vdash \psi$$  

so that they read as having two contexts: the free variables in the formulae and the formulae in the premise of the sequent. In fact, we will “stack up” two fibrations so that the objects living on top ($\Gamma \Rightarrow \psi$) are both fibered on those in the middle ($\Gamma$) and those on the bottom ($\times$). We hope to make it all clearer in the table that will follow.

**Construction 4.1.2 (Entailment).** We wish to represent entailment between two formulae in the same context. In order to do that we pick in $P^2$ (see Remark 4.0.6)
all objects belonging to the same fiber. Call $I : \text{ctx} \to \text{ctx}^2$ the functor mapping $\Gamma \mapsto \text{id}_\Gamma$ and compute the following (dashed) limit.

\[
\begin{array}{c}
P^2 I. p^2 \text{ctx} \\
\downarrow \downarrow \downarrow \downarrow \downarrow \\
\downarrow \downarrow \downarrow \downarrow \downarrow \\
\downarrow \downarrow \downarrow \downarrow \downarrow \\
P \text{ctx} \\
\text{dom} \quad \text{cod} \quad \text{dom} \quad \text{cod} \\
\downarrow \downarrow \downarrow \downarrow \\
\text{ctx} \\
\end{array}
\]

Both the pullback and all universal arrows belong to the judgemental theory. The classifier we are interested in is the composition $P^2 I. p^2 \text{ctx} \to \text{ctx}$, and will simply denote it with $e : \mathcal{E} \to \text{ctx}$.

**Remark 4.1.3** (Natural deduction as a type theory). One can check that $\mathcal{E}$ and $P$ fit into a plain dependent type theory as follows

\[
\begin{array}{c}
\mathcal{E} \quad \Sigma \quad P \\
\downarrow \downarrow \downarrow \\
\text{ctx} \\
\end{array}
\]

with $\Sigma : (!, \phi, \psi) \mapsto \psi$ and $\Delta : \phi \mapsto (\text{id}, \phi, \phi)$. They clearly form an adjoint pair, with $\Sigma$ cartesian. The counit is the identity, while the unit at each entailment is the entailment itself, hence both have cartesian components.

When no connectives nor quantifiers are involved, then, one can see the case for first order logic as a particular instance of dependent type theory with faithful type fibration.

**Remark 4.1.4.** The $\Gamma$ appearing in Notation 4.1.1 indicates a finite set of formulae in context $x$. We can see it as a product in $\mathcal{P}$ and, when we want to do so, we will write $\phi_\Gamma$.

We are finally ready to declare our local dictionary according to the notation of Section 2.3, and that is the following.

| $x \vdash e \mathcal{E}$ | $x \vdash (\text{dom} \circ I. p^2)(e) =_p \phi_\Gamma$ | $x \vdash (\text{cod} \circ I. p^2)(e) =_p \psi$ |
| $x \vdash e \mathcal{E}$ | $x \vdash (\text{dom} \circ I. p^2)(e) =_p \phi_\Gamma \land \phi$ | $x \vdash (\text{cod} \circ I. p^2)(e) =_p \psi$ |
| $x \vdash \Gamma \vdash \psi$ | $x ; \Gamma \vdash \psi$ |

**Remark 4.1.5.** Consider that in the case that ($x ; \perp \vdash \phi \mathcal{P}$ and $x \vdash \phi \to \psi \mathcal{E}$) then $x \vdash \phi \Rightarrow \psi \mathcal{P}$, therefore our framework accounts for the classical correspondence for all $x, \phi, \psi$,

\[
x \vdash \phi \Rightarrow \psi \quad \text{iff} \quad x ; \phi \vdash \psi.
\]

**Remark 4.1.6.** Since each $p$-fiber is thin, there is at most one $e \in \mathcal{E}$ between each pair of objects $(\phi, \psi) \in \mathcal{P} \times \mathcal{P}$.

**Notation 4.1.7.** In order to make our calculations more readable, we pin-point a specific notation for cod, dom in the case that they follow the inclusion of $\mathcal{E}$ into $\mathcal{P}^2$. We creatively write $c$ and $d$, respectively.
4.2. From properties to rules. Before we begin our analysis of rules of natural deduction, we show how certain properties lift from $\mathcal{P}$ (the category) to $\mathcal{E}$ (the judgement classifier). These will be instrumental in building up rules from $p$. In a sense, this subsection shows how to turn internal properties of $p$ into external rules about $p$, which is precisely what we did for contexts in Example 1.0.3.

Remark 4.2.1 (The domain-codomain policy). Since we will frequently use either $c$ or $d$ to select the consequent or the antecedent of a sequent, it will be useful to have a way to relate the two. The proof of Lemma 4.2.2 is clear evidence in this sense. There is a trivial policy

$$
\begin{array}{c}
\mathcal{P}^2 \\
\downarrow \text{id} \\
\downarrow \text{dom} \\
\downarrow \alpha \\
\downarrow \text{cod} \\
\downarrow \mathcal{P}
\end{array}
$$

where $\alpha_{\psi,\phi} = (\psi \to \phi)$. Note that if useful we might bravely invert the direction of id. We call $\alpha$, too, the obvious whiskering $d \Rightarrow c$.

Lemma 4.2.2 (A special instance of cut). The relation captured by $e : \mathcal{E} \to \text{ctx}$ is transitive in the sense that the rule below is in the NDT.

$$(T) \quad \frac{x; \psi \vdash \phi}{x; \psi \vdash \chi} \quad \frac{x; \phi \vdash \chi}{x; \phi \vdash \chi}$$

Proof. Consider the following $z$-lifting of the triangle in Remark 4.2.1 along $d$.

A little computation shows that the upper triangle reads as on the right, producing the desired rule

$$t := d.d \circ d*\text{id} : \mathcal{E}d.c\mathcal{E} \to \mathcal{E}.$$ 

Lemma 4.2.3 (Preservation through product as a rule). Interaction of arrows and products in $\mathcal{P}$ (the category) lifts to $\mathcal{E}$ (the judgement classifier) in the sense that the rule below is in the NDT.

$$(F) \quad \frac{x; \psi \vdash \phi}{x; \psi \Lambda \chi \vdash \phi \Lambda \chi} \quad \frac{x; \perp \vdash \chi}{x; \psi \Lambda \chi \vdash \phi \Lambda \chi}$$

Proof. It is coded by a functor $f : \mathcal{E} \times \mathcal{P} \to \mathcal{E}$ which to pairs $(\psi \to \phi, \chi)$ over some $x$ assigns the unique map $\psi \Lambda \chi \to \phi \Lambda \chi$ defined via the universal property of the product in the fiber over $x$. 

$\square$
4.3. **Formal structural rules.** Here we show that an NDT generates the following formal structural rules.

\[
\begin{align*}
\text{(H)} & \quad \frac{x; \Gamma, \phi \vdash \phi}{x; \Delta, \Gamma \vdash \phi} \\
\text{(Sw)} & \quad \frac{x; \Gamma, \psi, \psi \vdash \phi}{x; \Gamma, \phi \vdash \phi} \\
\text{(C)} & \quad \frac{x; \Gamma, \psi \vdash \phi}{x; \Gamma \vdash \psi} \\
\text{(W)} & \quad \frac{x; \Gamma \vdash \phi}{x; \Delta, \Gamma \vdash \phi}
\end{align*}
\]

We break the discussion into three parts.

4.3.1. **Hypothesis and the simple fibration.** Clearly for each pair \((\Gamma, \phi)\) over the same context, we have that \(\text{pr}_2 : \phi \wedge \phi \rightarrow \phi\), therefore \(\mathcal{E}\) classifies \(x; \Gamma, \phi \vdash \phi\).

\(\text{(H)}\) The Hypothesis rule, then, is coded into the existence of \(e\) itself.

We would be content with this already, but it is worth noticing that the association performing the projection \(\text{pr}_2\)

\[(\Gamma, \phi) \mapsto (\phi \wedge \phi \rightarrow \phi)\]

can be described functorially, and it contains some profound information. Such functor, in fact, provides an insight into possible developments of the present work, plus it (almost) allows for a presentation of the *simple fibration* from [Jac99], which has a meaningful logical interpretation: it constitutes the “least informative” type theory one can observe over a category with finite products. Therefore we say a little more about that.

**Definition 4.3.1** (The simple fibration). Define on the category \(\mathcal{E}\) the monad comprised of the following data:

- the functor \(S : \mathcal{E} \rightarrow \mathcal{E}\) acting as follows
  \[
  \psi \rightarrow \phi \quad \mapsto \quad \psi \wedge \phi \rightarrow \psi \rightarrow \phi;
  \]
- the 2-cell \(\eta : \text{Id} \Rightarrow S\) defined via the universal property of products;
- the 2-cell \(\mu : S \circ S \Rightarrow S\) acting as \((\text{pr}_1, \text{id})\).

**Remark 4.3.2.** \((S, \eta, \mu)\) is idempotent. This is because \(\mu\) acts as follows

\[
\begin{array}{cccc}
\psi \wedge \phi & \rightarrow & \psi & \rightarrow & \phi \\
\downarrow & & \downarrow & & \downarrow \\
\psi \wedge \phi & \rightarrow & \psi & \rightarrow & \phi
\end{array}
\]

and \((\psi \wedge \phi) \wedge \phi = \psi \wedge \phi\) because \(p\) is thin and its products are fibered, and in fact the forgetful functor from algebras over \(S\) into \(\mathcal{E}\) is fully faithful. All \(S\)-algebras are free.

The Kleisli category of \(S\) is equivalent to (the total category) of what in [Jac99] is called the *simple fibration* associated to \(p\). That is \(\mathcal{P} : s(P) \rightarrow \mathcal{P}\) where \(s(P)\) has for objects pairs \((\phi, \phi')\) in the same \(p\)-fiber and maps \(a = (a_1, a_2) : (\phi, \phi') \rightarrow (\psi, \psi')\) such that \(a_1 : \phi \rightarrow \psi\), \(a_2 : \phi \wedge \phi' \rightarrow \psi\), and \(p(a_1) = p(a_2)\). The functor \(\mathcal{P}\) acts as the first projection. If we call \(1 : \text{ctx} \rightarrow \mathcal{P}\) the (fibered) terminal object functor, one checks that \(\mathcal{P}1 \cong p\). Moreover, the functor

\[
q : (\phi, \phi') \mapsto (\phi \wedge \phi' \rightarrow \phi)
\]

induces a comprehension category (as in Construction 3.2.1) \(s(P) \rightarrow \mathcal{P}^\rightarrow\). The type theory associated to such a functor is (that equivalent to) untyped lambda calculus.
The functor $S$ induces the following rule.

$$
\begin{array}{ccc}
\mathcal{E} & \xrightarrow{S} & \mathcal{E} \\
\downarrow_{c} & & \downarrow_{c} \\
\mathcal{P} & \xrightarrow{\vdash} & \mathcal{P} \\
\downarrow_{p} & & \\
\text{ctx} & & \\
\end{array}
\quad \text{Dictionary in Section 4.1}
\quad \begin{array}{c}
x; \Gamma \vdash \phi \\
\hline
x \vdash \phi_{\Gamma} \\
\hline
x \vdash S(\phi_{\Gamma} \rightarrow \phi) \mathcal{E}
\end{array}
\quad \text{Definition 4.3.1}
\quad \begin{array}{c}
x \vdash \phi_{\Gamma} \land \phi \rightarrow \phi \mathcal{E} \\
\hline
x; \Gamma, \phi \vdash \phi
\end{array}
\quad \text{Dictionary in Section 4.1}

4.3.2. Swap and Contraction: fibered products everywhere.

(Sw) The Swap rule holds because the fibered product is symmetric and this too is expressed via a commutative triangle: consider the following composition

$$
\mathcal{P} \times \mathcal{P} \xrightarrow{s} \mathcal{P} \times \mathcal{P} \xrightarrow{\land} \mathcal{P}
$$

where the map $s$ computes the permutation. The desired rule is computed as the (iso)morphism

$$
s.(d.\land) : (\mathcal{P} \times \mathcal{P}).\mathcal{E} \rightarrow (\mathcal{P} \times \mathcal{P}).\mathcal{E}.
$$

(C) Contraction is supported by the following dashed map

$$
(\mathcal{P} \times \mathcal{P})d.(\land \circ \text{id} \times \Delta)\mathcal{E} \xrightarrow{d} \mathcal{E}
$$

$$
\xrightarrow{id \times \Delta} \mathcal{P} \times \mathcal{P} \xrightarrow{id \times \text{pr}_{1}} \mathcal{P} \times \mathcal{P}
$$

$$
\xrightarrow{\land} \mathcal{P}
$$

where we write $\land$ for the obvious product $\mathcal{P} \times \mathcal{P} \times \mathcal{P} \rightarrow \mathcal{P}$. On the bottom we have the triangle on the left commuting, and $\text{id} \times \Delta$ equalizing $\land$ and $\text{id} \times \text{pr}_{1} \circ \land$. The two nested judgements on the top classify, respectively, the antecedent and the consequent of (C), and the dashed map exists by the universal property of the "smaller" pullback.

4.3.3. Weakening and Cut: more transitivity. We will see that to provide both Weakening and Cut it is sufficient to apply (T) from Lemma 4.2.2 to appropriate triples. Let us start with (W), first: notice that, as it happened in Section 3 and is evident from Section 4.1, the consequent in (W) is actually silent of (at least) one judgement, that is $x; \bot \vdash \psi$. The procedure we follow for (W) is that of

$$
\begin{array}{c}
\hline
x; \Gamma \vdash \phi \\
x; \Gamma, \psi \vdash \phi \land \psi
\end{array}
\quad (F)
\quad \hline
\begin{array}{c}
x; \Gamma \vdash \phi \lor \psi
\end{array}
\quad (T)

$$

$$
\begin{array}{c}
x \vdash \phi_{\Gamma} \\
x \vdash \phi_{\Gamma} \land \phi \rightarrow \phi \mathcal{E}
\end{array}
\quad \begin{array}{c}
x; \Gamma, \phi \vdash \phi
\end{array}
\quad \begin{array}{c}
x \vdash \phi_{\Gamma} \land \phi \rightarrow \phi \mathcal{E}
\end{array}
\quad \begin{array}{c}
x; \Gamma, \phi \vdash \phi
\end{array}
$$
therefore we need to pre-process the premise of \( t \) in order to apply it to triples of the form \((φT ∧ ψ, φ ∧ ψ, φ)\). This is achieved via the following diagram

\[
\begin{array}{c}
\mathcal{E} \times \mathcal{P} \\
\downarrow f \quad \downarrow e \times \text{id} \quad \downarrow q_1 \quad \downarrow \gamma_1 \\
\mathcal{E} \times \mathcal{P} \\
\downarrow t \quad \downarrow \varepsilon \quad \downarrow \text{id} \quad \downarrow d \quad \downarrow \varepsilon \\
\mathcal{E} \quad \mathcal{E} \quad \varepsilon \quad \text{id} \quad \varepsilon \quad \mathcal{P}
\end{array}
\]

with \( q_1 \) the map \((φ, ψ) \mapsto (φ ∧ ψ → φ)\) acting on pairs in the same \( p \)-fiber. Notice how this is related to \( q \) in Definition 4.3.1.

**Remark 4.3.3** (Cones and branches). Here branches in the tree of a deduction correspond to cones over limit diagrams. We could make this statement more precise, but we hope the following discussion speaks for itself.

(W) Weakening is computed by the dashed arrow above followed by \( t \) from Lemma 4.2.2.

For (Cut) we again apply Lemma 4.2.2, this time to the triple \((φT, φT ∧ φ, ψ)\), that is we will build the diagram corresponding to the following composing rules.

\[
\begin{array}{c}
\frac{x; Γ ⊨ φ}{x; Γ ⊨ φ} \\
\frac{x; Γ ⊨ φ, x; Γ, φ ⊨ ψ}{x; Γ ⊨ φ ∧ ψ} \\
\frac{x; Γ ⊨ φ}{x; Γ ⊨ ψ}
\end{array}
\]

Therefore we want a map from \( \mathcal{E}_{d.d}S\mathcal{E} \), classifying the premise of Cut, into \( \mathcal{E}_{d.c}\mathcal{E} \) so that we then can apply (T). That is achieved as follows.

\[
\begin{array}{c}
\mathcal{E}_{d.d}S\mathcal{E} \\
\downarrow d.S.d \\
\mathcal{E} \\
\downarrow (id, d) \\
\mathcal{E} \times \mathcal{P} \\
\downarrow f \\
\mathcal{E} \\
\downarrow \varepsilon \\
\mathcal{P}
\end{array}
\]

(Cut) Cut is computed by the dashed arrow above followed by \( t \) from Lemma 4.2.2.

**Remark 4.3.4** (Cut is a policy). While perhaps not evident, the Cut rule is in fact a policy in the sense of Definition 1.0.1: it just pre-processes data going into the policy (T) from Lemma 4.2.2.

\[
\begin{array}{c}
\mathcal{E}_{d.d}S\mathcal{E} \quad \mathcal{E}_{d.c}\mathcal{E} \quad \mathcal{E}_{d.d}\mathcal{E}
\end{array}
\]

**Remark 4.3.5** (Relationship between DTy and Cut). As we have seen, at the very core of Cut sits the policy (T) from Lemma 4.2.2. The reader will notice the incredible similarity between the process that constructs (T) and the process that constructs (DTy).
Not only do the diagrams in Lemma 4.2.2 and in Section 3.4.3 look very similar, but even their ingredients have affine logical meaning. Indeed, in both cases the hypothesis of the policy is a nested judgement where a modality appears: in the case of natural deduction, this is the monad \( S \), in the case of dependent type theory it is the monad \( \Delta \Sigma \). Of course, some delicate differences appear too\(^2\). This kind of thoughts could lead to a general notion of cut, a special family of policies, but we leave such a task for a possible future work.

4.4. Formal rules for connectives. We here show that the moment we ask that connectives are closed under \( p \)-fibers, with \( p \) a NDT, we automatically get the expected rules. Since Definition 4.0.3 already contains the requirement that \( p \) has fibered products, we here show how to provide in a NDT rules for \( \land \), and need to ask nothing more of it. If the reader inspects the constructions below, they will see that such a procedure could be repeated for NDTs having \( p \) additionally equipped with \( \lor \), \( \neg \).

The ones which are usually required for \( \land \) are the following.

\[
\frac{x; \Gamma \vdash \phi}{x; \Gamma \vdash \phi \land \psi} \quad \frac{x; \Gamma \vdash \phi \land \psi}{x; \Gamma \vdash \phi} \quad \frac{x; \Gamma \vdash \phi \land \psi}{x; \Gamma \vdash \psi}
\]

(\( \land I \)) Introduction is represented by the functor \( \text{conj} : \mathcal{E}d.d \mathcal{E} \to \mathcal{E} \) induced by the universal property of the fibered product.

(\( \land E_1 \)) In order to represent its domain, we compute the equalizer

\[
\mathcal{E}(d.d, \text{conj}) \longrightarrow \mathcal{E}d.d \mathcal{E} \xrightarrow{\text{conj}} \mathcal{E}
\]

where by \( d.d \) (sadly ambiguous, in this case) we wish to express the first projection of the pullback. The desired rule is then induced by the first product projection and it assumes the following form.

\[
\mathcal{E}(d.d, \text{conj}) \to \mathcal{E} \to \mathcal{E}
\]

(\( \land E_2 \)) Dually, we compose the equalizer with the functor induced by the second projection.

**Definition 4.4.1** (Heyting and Boolean NDTs). A NDT is said to

- be **Heyting** if we have operators \( \bot, \top, \land, \lor, \Rightarrow \) of the proper arities on \( p \);
- be **Boolean** if it is Heyting and, being \( \neg := (-) \Rightarrow \bot \), the morphism of fibrations \( -\neg \) is equivalent to id.

4.5. Substitution. While in Section 3 we thought of morphisms of contexts as substitutions, in the setting of proof theory we regard them as terms. When we write a map

\[ y \to x \]

we see it as a list of terms and denote it as such:

\[ [t/x] : y \to x. \]

\(^2\)For example the \textit{height} at which one performs the action of the monad, that is \( \check{U} \) and \( U \) are manipulated over contexts while both instances of \( \mathcal{E} \) are bounded to formulae.
In particular, if \( x = x_1 \times \cdots \times x_k \), each term \( t_i = \text{pr}_i \circ t \) is a term built up from \( y \) and in context \( x_i \), with \( i = 1, \ldots, k \). Then we can identify \( \text{ctx}_x \) with the classifier collecting all terms in context \( x \).

All of this belongs to the intuition and in fact there is nothing more to \( \text{ctx} \) than what described in Definition 4.0.3, but it is with this perspective that we now look at how substitution behaves in NDTs. Recall from Section 2.5 that substitutionality allows us to compute rules and policies as the following

\[
\begin{align*}
\text{ctx}^2.\text{cod}P & \xrightarrow{\text{cod},p} P \\
\text{ctx}^2.\text{dom}P & \xrightarrow{p,\alpha} \text{ctx}^2 \xrightarrow{\alpha,\text{dom}} \text{ctx}
\end{align*}
\]

and here we have only blindly expanded the information contained in the diagram on the left by following the discussion in 2.5.

4.6. Formal rules for quantifiers. Finally, we wish to give an account of quantifiers, hence we introduce more structure on \( p \) and on the judgemental theory it generates.

**Definition 4.6.1** (First order NDTs). A NDT is said to

- be intuitionistic first order if it has weakening (Remark 4.0.7), is Heyting and \( w \) from Remark 4.0.7 has left and right adjoints,
  
  \[ \exists \dashv w \dashv \forall, \]

  where \( \exists, \forall \) are morphisms of fibrations and belong to \( \mathcal{J} \); we call such theories IcFOTs, for short;
- is classical first order if it is intuitionistic first order and also Boolean; we call these cFOTs for short.

We believe that the request of being morphisms of fibrations (i.e. preserve cartesian squares) is related to the more traditional properties required for \( \forall, \exists \), namely Frobenius reciprocity and Beck-Chevalley.

**Remark 4.6.2.** Consider a (intuitionistic) first order theory in the traditional sense. Then it induces a (I)cFOT: (the fibration associated to) the hyperdoctrine of Lindenbaum-Tarski algebras of well-formed formulae, as for example in [MR13].

We only provide explicit representation of the rules involving \( \forall \) in the IcFOT, \( \exists \) could be worked out in a similar fashion. First of all, notice that the pair of adjoint functors \( w \dashv \forall \) induces (via the hom-set isomorphism) the following rule (on the left)

\[
(FA) \quad \frac{x \times y \vdash w(\phi, \psi) \leq \chi P}{(x, y) \vdash (\phi, \psi) \leq \forall x P \times P} \quad \frac{x \times y \vdash w_y \phi \land w_x \psi \leq \chi P}{x \vdash \phi \leq \forall y \chi P \quad y \vdash \psi \leq \forall x \chi P}
\]

which, if we denote \( \forall_y = \text{pr}_1 \circ \forall \) and \( \forall_x = \text{pr}_2 \circ \forall \), amounts to the rule on the right. The two rules we need to produce are the following.
\[(\forall I)\quad \frac{x \times y; \; w_y \Gamma \vdash \phi}{x; \; \Gamma \vdash \forall y \phi}\]

\[(\forall E)\quad \frac{x; \; \Gamma \vdash \forall y \phi}{x; \; \Gamma \vdash \phi[t/y]}\]

Notice that we included the writing \(w_y \Gamma\) (with \(w_y\) of the kind described in Remark 4.0.7) to express the desired dependency, since in this case we wish to say that there is no \(y\) free in \(\Gamma\). Also, writing \(\phi[t/y]\) is a bit improper in the sense that, since \(p(\phi) = p(\phi_1) \times y = x \times y\), each substitution in \(\phi\) should have codomain \(x \times y\).

It is clear what happens here, but we will go into detail when the time comes.

We begin with Introduction. It does actually pretty much read as the fact that \(\forall\) is right adjoint to \(w\) “at” the triple \(((\phi_1, \phi_2), \phi)\), but if we wish to write a rule in the sense of Definition 1.0.1, we shall start computing the premise, which we do via the following pullback

\[
\begin{array}{ccc}
P(Pr_1 \times) & \longrightarrow & P \\
\downarrow & & \downarrow \times \\
\text{ctx}^2 & \longrightarrow & \text{ctx}
\end{array}
\]

which classifies pairs \((x \vdash \Gamma, x \times y \vdash \phi)\). But now we exploit the fact that

\[w_y \Gamma \leq \phi \quad \text{iff} \quad w_y \Gamma \land \phi = w_y \Gamma\]

so we ask of the equalizer of the maps

\[
\begin{array}{ccc}
P(Pr_1 \times) & \longrightarrow & P^x \\
\downarrow & \downarrow \times & \downarrow \times \\
\text{ctx}^2 & \longrightarrow & \text{ctx}
\end{array}
\]

with the top one computing \(w_y \Gamma \land \phi\) and the bottom one \(w_y \Gamma\). We denote \(\mathcal{E}(p^x, p^x(Pr_1, Pr_1))\) with \(\mathcal{A}\).

\[(\forall I)\] The introduction rule is the functor \(\mathcal{A} \rightarrow \mathcal{A}\) which follows from the hom-set isomorphism discussed above. Its inverse implies that actually it is the following.

\[
\begin{array}{ccc}
x \times y; \; w_y \Gamma \vdash \phi & \longrightarrow & x; \; \Gamma \vdash \forall y \phi
\end{array}
\]

With Elimination, we (implicitly) use the isomorphism above and write \([t/y]\) for \(([x/x], [t/y]) : x \rightarrow x \times y\) exploiting \(\text{ctx}_{/x \times y} \cong \text{ctx}_{/x} \times \text{ctx}_{/y}\). Using substitution again as in Section 2.5, we get

\[
\begin{array}{ccc}
x \times y; \; w_y \Gamma \vdash \phi & \longrightarrow & x; (w_y \Gamma)[t/y] \vdash \phi[t/y]
\end{array}
\]

But recall that \(w_y \Gamma = \phi_1[pr_1]\), and since

\[\text{[pr_1][t/y]} : x \rightarrow x \times y \rightarrow x\]

is \(\text{id} = [x/x]\), given that also \(p\) is faithful, we automatically get \((w_y \Gamma)[t/y] = \Gamma\) concluding the proof.
4.7. **Cut elimination.** In pointing out necessary features of a judgemental analogue of natural deduction, we see that no instance of Cut is (explicitly) mentioned and, instead, in Section 4.3 Cut is shown to *automatically* be in the IcFOT generated by \( p : \mathcal{P} \rightarrow \text{ctx} \). We regard this as an instance of what in sequent calculus is called “cut elimination” (and is shown to be quite hard to prove [Gen64]), or of “normalization” in natural deduction (which, in turn, follows almost instantly from admissibility).

In a very precise sense, such rule is a tool that we already have encoded in the theory the moment we require that it satisfies some properties that we deem fundamental. In fact, curiously, the main reason it works is the existence of the domain-codomain policy (Remark 4.2.1) and *not* (only) composition of arrows. More on this peculiarity was discussed in Remark 4.3.5.

5. **Ceci n’est pas un topos**

The definitions developed in this work allow for a discussion about the internal logic of a topos, intended in the most unbiased sense. Indeed this section will touch on several variations of the concept:

- elementary topoi à la Lawvere-Tierney [Law71];
- pretopoi and predicative approaches in the spirit of Maietti [Mai05];
- 2-topoi à la Weber [Web07] and cosmoi à la Street [Str74, Str80].

We will see that all these notions of *topos* support a plain dependent type theory in the sense of Section 3. Such a dtt recovers, among other things, the Mitchell-Bénabou language of the topos and nicely interacts with its Kripke-Joyal semantics. Most importantly, though, our treatment frames the main feature of a topos-like category in a clear way. The discussion is set in such a way that at each step the level of conceptual complexity gets higher and higher. The discussion about predicative foundations, in particular, contains a key point of view to understand our treatment of 2-topoi, which is an improved version of [Web07].

5.1. **Elementary topoi.**

5.1.1 (A bit of history). The internal logic of a topos has been discussed by several authors. After [MLM94], this collective humus has been crystallized in the Mitchell-Bénabou language and its *tautological* interpretation, the Kripke-Joyal semantics. These attributions are somewhat symbolic. For what concerns the Mitchell-Bénabou language, the best historical account is given, to our knowledge, by Johnstone [Joh77]. After Mitchell’s original contribution [Mit72], Johnstone refers to the unfindable [Cos72] for Bénabou’s contribution, but the paper is actually authored by Coste. [Osi75a] and others were definitely part of the intellectual debate on the topic. For what concerns the Kripke-Joyal semantics the situation is much more cloudy, Osius [Osi75b] tells us that the original ideas from Joyal were never published, while a footprint of Joyal’s contribution to the topic only emerges (in French) in [BJ81]. These ideas were later conveyed in several texts with slight variations, like [LS88] and [Bor94]. Both in the case of the language and its semantics, we will refer to the presentation in [MLM94, VI, Sec. 5 and 6] which is in a sense the most informal and essential. Our main objective is to demonstrate that our formalism can reboot the core ideas behind the Mitchell-Bénabou language. We will not discuss in detail Kripke-Joyal semantics, even though the connection could be drawn, as exemplified by the recent [AGH21].
**Definition 5.1.2** (The dtt of an elementary topos). For an an elementary topos $\mathcal{E}$, we can construct a dependent type theory in the sense of Definition 3.0.1 as follows.

![Diagram]

The map $\Sigma_\top$ is induced (via precomposition) by the map $\top : 1 \to \Omega$ which picks the top-element of $\Omega$. $\Delta_\top$ is given by pullback, and of course the whole discussion fits perfectly with Theorem 3.1.2, with the technical advantage that the presheaves in this case are internally represented by objects in the topos, thus there is no need to use the Yoneda embedding.

**Remark 5.1.3** (Comprehension category, display maps, monomorphisms). Of course, at this point the whole content of Section 3 applies, and thus we can load a whole judgement calculus for this dependent type theory. For example, the rule

\[
\frac{\Gamma \vdash \phi : \mathcal{E}/\Omega}{\Delta \phi \vdash \Delta \phi : \mathcal{E}/1}
\]

is telling us that to each proposition $\phi : X \to \Omega$, corresponds an object $\Delta \phi$, which is precisely the object supporting the subobject of $X$ classified by $\phi$. Similarly, following Construction 3.2.1, we obtain a representation of the internal logic of the topos in terms of a comprehension category,

\[
\text{disp} : \mathcal{E}/\Omega \to \mathcal{E}^2.
\]

Such correspondence maps a formula $\phi$ to the dashed colored arrow in the construction above. It follows that the correspondence maps a proposition to its zero locus, i.e. the monomorphism whose characteristic function is precisely $\phi$. Of course, this idea is not novel and it dates back to Taylor’s PhD thesis or his more recent [Tay99].

**Remark 5.1.4** (Mitchell-Bénabou reloaded). Following [MLM94, VI, Sec. 5] we see that there is a canonical dictionary between our judgements classified by $\mathcal{E}/\Omega$ and *formulae*, i.e. terms of type $\Omega$ in the sense of [MLM94, pag. 299, right after the bulleted list]. Moreover, and somewhat most importantly, display maps construct subobjects as zero locus of formulae, as explained in [MLM94, pag. 300, right after the bulleted list].

\[
\begin{array}{ccc}
\Gamma \vdash \phi : \mathcal{E}/\Omega & \quad & \phi(x) \\
X \vdash \Delta \phi : \mathcal{E}/\Omega & \quad & \{x | \phi(x)\}
\end{array}
\]

Notice the difference between $\Delta \phi$ and $\text{disp}_\phi$: even though they might seem to be similar things, the first one gives us the support of the subobject, while the second
one gives us the subobject itself.

\[
\{ x | \phi(x) \} \xrightarrow{\Delta \phi} 1
\]

\[
\begin{array}{c}
\text{disp}_x \\
downarrow \\
X \\
\phi \\
\downarrow \\
\Omega
\end{array}
\]

As a result of this discussion, one can use the judgement calculus produced by this dependent type theory to simulate the internal logic of the topos, and the result will be consistent with the Mitchell-Bénabou language of the topos.

Let us give a few examples. Notice that we chose topos as a very strong theory, but in fact Lemma 5.1.5, Lemma 5.1.6 show the modularity of our approach, in a fashion very much affine to \cite{Mai05}.

**Lemma 5.1.5.** The pDTT induced by a topos \( E \) has unit types in the sense of Definition 3.8.1.

*Proof.* It suffices to show that we have functors \(*\) and \(1\) making the following diagram commute and the square a pullback.

\[
\begin{array}{ccc}
\mathcal{E} & \xrightarrow{\ast} & \mathcal{E}/\mathcal{T} \\
\downarrow & & \downarrow \Sigma \\
\mathcal{E} & \rightarrow & \mathcal{E}/\Omega
\end{array}
\]

Let us denote \( !_X \) the unique map from \( X \) to the terminal – for the moment, elsewhere we have and we will use \( X \) both for the object and the map to 1. One can easily check that defining \( \ast : X \mapsto !_X \), and in the obvious way on morphisms, and \( 1 : X \mapsto \top \circ !_X \), and in the obvious way on morphisms, does the job. \( \square \)

**Lemma 5.1.6.** The pDTT induced by a topos \( E \) has extensional identity types in the sense of Definition 3.6.1.

This can be proved in similarly as in Lemma 5.1.5, using equalizers. We take a bit of care in proving the following, instead.

**Lemma 5.1.7.** The pDTT induced by a topos \( E \) has dependent product types in the sense of Definition 3.5.1.

*Proof.* It suffices to show that we have functors \( \lambda \) and \( \Pi \) making the following diagram commute and the square a pullback.

\[
\begin{array}{ccc}
\mathcal{E}/\Omega \Delta \mathcal{E}/\mathcal{T} & \xrightarrow{\lambda} & \mathcal{E}/\mathcal{T} \\
\downarrow & & \downarrow \Sigma \\
\mathcal{E}/\Omega \Delta \mathcal{E}/\Omega & \xrightarrow{\Pi} & \mathcal{E}/\Omega
\end{array}
\]
Let us first compute the two categories
\[ \mathcal{E}_{/\Omega} \cdot \Delta \mathcal{E}_{/\Omega} \quad \text{and} \quad \mathcal{E}_{/\Omega} \cdot \Delta \mathcal{E}_{/\top} . \]
Following the construction in Section 3.4.2, we can see that they respectively have objects
\[ (\phi, \psi) \quad \text{and} \quad (\phi, \{x|\phi(x)\}) \]
with \( \phi, \psi \) as below.

The vertical map on the left hand side of the square computes the diagonal of the pullback square above, meaning it acts as \((\phi, \{x|\phi(x)\}) \mapsto (\phi, \phi \circ \text{disp}_\phi).\)

To provide suitable \( \Pi, \lambda \) we of course look at right adjoints to pullback functors. The fact that they reasonably model dependent products has been widely discussed from the publication of [See84], with distinguished treatments in [CZ21], where an explicit construction is given, and in [Mai05], where it is better framed in the context of the different properties of a topos and their logical counterpart.

One can always show that for a given \( \phi : X \to \Omega \) (and, in fact, for any \( f : X \to Y \)), we have the following equivalence and adjunction,

\[ \mathcal{E}/X \cong (\mathcal{E}_{/\Omega})_{/\phi} \quad \begin{array}{c}
\phi^* \downarrow \\
\phi_\Lambda
\end{array} \]

see for example [MLM94, IV.7]. Given a pair \((\phi, \psi)\) in \( \mathcal{E}_{/\Omega} \cdot \Delta \mathcal{E}_{/\Omega} \), then, it is natural to compute \( \text{disp}_\psi \),

\[ \begin{array}{ccc}
1 & \triangleleft & \{x, \phi(x)|\psi(x)\} \\
\uparrow & \searrow & \downarrow \text{disp}_\psi \\
\Omega & \leftarrow & \{x|\phi(x)\} \\
\downarrow & \swarrow & \downarrow \Delta \phi & \downarrow \top \\
X & \phi & \Omega
\end{array} \]

and define \( \Pi(\phi, \psi) = \Pi_{\phi}(\text{disp}_\phi \circ \text{disp}_\psi) \). As for \( \lambda \), we put \( \lambda(\phi, \{x|\phi(x)\}) = \{x|\phi(x)\} \).

The square involving \( \Pi, \lambda \) commutes because \( \{x, \phi(x)|\phi(x)\} = \{x|\phi(x)\} \) hence the composition of displays is mapped to the trivial triangle \( \text{disp}_\phi : \phi \circ \text{disp}_\psi \to \phi \).

The universal property of \( \Pi_{\phi} \) is what guarantees that the domain of \( \Pi(\phi, \psi) \) is, in fact \( \rho[\Pi(\phi, \psi)(\rho)] \). From this remark, one can immediately show that the desired square is a pullback. \( \square \)

This is nothing new, but we believe it provides a different perspective on the internal logic of a topos (or any category, really). It ends up being really close to the following intuition.

We can then conclude that describing the internal dependent type theory of a category means to capture the type-theoretic properties of the codomain fibration, while describing the internal many-sorted logic of a category – considering the sorts as types – means
to capture the properties of the subobject fibration together with
the one-dimensional structure of the category under consideration.

[Maï05]

In a sense, our work is about extending this process to more than just the codomain
fibration.

5.2. Predicative topoi. Under the name of predicative mathematics goes a gra-
dient of foundations that, at its extreme, rejects the assumption of function spaces
and powersets. In this sense, the category of sets we are used to work with, and on
which the whole program of ETCS [Law64, LM05] is built on, is inherently impred-
icative. As Awodey pointed out in his talk at the CT2021 in Genova [Awo21], this
bit of impredicativity is the trade off for a very algebraic approach to set theory, so
that its main features can be encoded in few axioms, as those in the definition of
elementary topos. Yet, for a sufficiently topos-like predicative foundation, we can
still reason in a way that is very similar to the case of an elementary topos, and
provide a dtt whose judgement calculus is the internal logic of the predicative topos.

Definition 5.2.1 (Virtual object). A presheaf \( P : \mathcal{C}^{\text{op}} \to \text{Set} \) is virtually repres-
entable, or more simply a virtual object if it preserves all limits that exist. A
subobject \( P \to \mathcal{Y} \cdot c \) of a representable that is a virtual object is called a virtual
subobject of \( c \).

Remark 5.2.2 (Freyd dust). Virtual objects will play a crucial role in our definition
of predicative topoi. Before we give it, though, we feel the need to put a bit of
context around our virtual objects. While the name itself, and in a sense the
intuition that we have on them, is somewhat original, the general idea has been
known to category theorists since forever. If we ignore the solution set condition in
the Adjoint Functor Theorem, then the Yoneda embedding yields an equivalence
of categories

\[ \mathcal{Y} : \mathcal{C} \to \text{Cont}(\mathcal{C}^{\text{op}}, \text{Set}) . \]

Thus, virtual objects are a kind of Freyd dust covering the image of the Yoneda
embedding. These presheaves have almost indistinguishable properties with respect
to a representable, and – up to a size issue – they are just the image of the Yoneda
embedding. This intuition sits at the core of the very recent [Bra21], and was
already used from a technical point of view in [MP89, 6.4].

Definition 5.2.3 (Predicative topos). A predicative topos \( \mathcal{C} \) is a category with
finite limits that

- is virtually cartesian closed, i.e. \( \mathcal{C}(\cdot \times b, c) \) is a virtual object for all \( b, c \);
- has specification, i.e. virtual subobjects are representable;
- has virtual subobject classifier, i.e. the subobject doctrine \( \text{Sub} : \mathcal{C}^{\text{op}} \to \text{Set} \)
is a virtual object.

Remark 5.2.4 (Descent, Descent, Descent). This definition captures a key feature
of Grothendieck topoi. Indeed, if one inspects the reason for which a Grothendieck
topos has a subobject classifier, one discovers that the exactness properties of the
category force the subobject functor to be continuous, thus descent implies that \( \text{Sub} \)
is a virtual object. Because descent is the defining feature of infinitary pretopoi,
their subobject doctrine is a virtual object too. It follows that an infinitary pretopos
with specification is a predicative topos too. If we want these exactness property to
be witnessed by an object in the category (i.e. if we want \( \text{Sub} \) to be representable)
we trade its existence with predicativity. This very geometric point of view is implicitly claiming that some form of descent is the key feature of a topos, which is impredicatively forced in the definition of elementary topos via its subobject classifier. Let us isolate the main observation of this remark in the corollary below.

**Corollary 5.2.5.** An infinitary pretopos with specification is a predicative topos.

**Definition 5.2.6** (The dtt of a predicative topos). Let $\mathcal{C}$ be a predicative topos. Consider the following pullback diagram in the category of prestacks over $\mathcal{C}$,

$$
\begin{array}{ccc}
P & \rightarrow & 1 \\
\downarrow & & \downarrow \\
\Gamma \phi & \rightarrow & \text{Sub}
\end{array}
$$

Because all the prestacks involved in the cospan are virtual objects, and virtual objects are trivially closed under limits, $P$ is virtual. Since $\top$ is a mono, and monos are pullback stable, $P$ is a virtual subobject, and thus it is represented by assumption via some object $\Gamma.\phi \in \mathcal{C}$. It follows as in the proof of Theorem 3.1.2, that in the diagram below involving the subobject fibration, the functor $\Sigma\top$ has a right adjoint, which thus provides a plain dtt in our sense.

$$
\begin{array}{ccc}
\mathcal{C}/_{\top} & \rightarrow & \text{Sub} \\
\downarrow & & \downarrow \\
\mathcal{C} & \leftarrow & \Delta
\end{array}
$$

5.3. **Elementary 2-topoi.** Elementary 2-topoi were introduced by Weber in [Web07], with Yoneda structures and cosmoi [Str80] in mind. The analogy with elementary topos is exemplified by the prototypical example of elementary 2-topos.

**Example 5.3.1** (The 2-topos of categories). Consider the 2-category $\text{Cat}$, with some flexibility on size. To be more precise, $\text{cat}$ is the 2-category of (essentially) small categories, $\text{Cat}$ is the 2-category of locally small, but possibly large categories, $\text{CAT}$ is the 2-category of locally large categories. Then

$$
\begin{array}{ccc}
\text{Elts}(\phi) & \rightarrow & \text{Set} \\
\downarrow & & \downarrow \\
\mathcal{C} & \rightarrow & \text{Set}
\end{array}
$$

there is an equivalence of categories - established by the Grothendieck construction - between discrete fibrations over $\mathcal{C}$ and copresheaves $\phi$ as in the diagram above. This is telling us that $\text{Cat}$ has a classifier of discrete fibrations, given by the copresheaf construction. So, in a 2-topos, discrete opfibrations play the analog of monomorphisms, and their associated prestack is representable.

$$\text{Fib}_{\text{disc}}(\mathcal{C}) \simeq \text{Cat}(\mathcal{C}, \text{Set}).$$
**Definition 5.3.2** (Elementary 2-topos, very similar to [Web07, Def. 4.10]). An elementary 2-topos is a cartesian closed 2-category with finite limits and a classifier of discrete fibrations.

**Remark 5.3.3** (Not exactly Weber). Weber’s original definition allows for a more humble notion of classifier, indeed it can be a classifier of some discrete fibrations. Also, it is based on the notion of opfibration, but this choice does not lead to any conceptual difference in our treatment. Among the examples, he gives $\text{Set}_{\lambda \rightarrow \lambda} \to \text{Set}$, as the classifier of the fibrations with $\lambda$-small fibers. Our definition, which is in some sense more ambitious but also closer to that of elementary topos rules out all our desired examples.

- $\text{cat}$ has finite limits and is cartesian closed, but it does not have a classifier.
- $\text{Cat}$ has finite limits and a subobject classifier, but is not cartesian closed.
- $\text{CAT}$ does not have the classifier, again.

This is probably the reason behind Weber’s flexibility, indeed the classifiers of $\lambda$-small fibrations are by many considered a sufficiently expressive alternative that successfully eludes size issues. We do not see it that way. Here we see that we have a problem that is very similar to the predicativity case.

Luckily, there is a very consistent way to fix Weber’s definition of elementary 2-topos in such a way that all the listed desiderata are indeed examples. The situation is very similar to that of predicativity. Indeed, the prestack of fibrations

$$P : \text{cat}^{\text{op}} \to \text{Cat}$$

is a virtual object (because it is classified by hom-ing into $\text{Set}$), despite not being representable. This is witnessing the fact that $\text{cat}$ has a 2-dimensional version of descent, and indeed it is a 2-topos in the sense of Street [Str82]. Of course, on a technical level, it just follows from the fact that such a prestack is almost representable, and thus of course it is a virtual object.

**Definition 5.3.4** (The fibration of discrete opfibrations). Let $\mathcal{K}$ be a 2-category, and consider the prestack mapping an object $k$ to the category of discrete fibrations over $k$,

$$k \mapsto \text{Fib}_{dsc}(k).$$

Via the Grothendieck construction, this prestack comes with an associated fibration, for which we will use the same name. Moreover, because the identity of $k$ is always a discrete fibration, we can construct the following commutative triangle.

$$
\begin{array}{ccc}
\mathcal{K}/1 & \xrightarrow{\Sigma_T} & \text{Fib}_{dsc} \\
\downarrow & & \downarrow \\
\mathcal{K} & \xrightarrow{} & \\
\end{array}
$$

**Definition 5.3.5** (Similar to [Web07, Def. 4.10]). An elementary 2-topos is a 2-category $\mathcal{K}$ that

1. has finite 2-limits,
2. is cartesian closed,
3. the prestack of discrete fibrations is a 2-virtual object, i.e. it preserves all 2-limits,
4. $\Sigma_T$ above has a right adjoint.
Remark 5.3.6 (The dtt of an elementary 2-topos). As in the case of Definition 5.1.2 and Definition 5.2.6 the existence of the right adjoint for \( \Sigma^\top \) provides us with a dtt in the sense of Theorem 3.1.2, expressing the internal logic of the elementary 2-topos.

Example 5.3.7. Now, let us show that \( \text{cat} \) is an elementary 2-topos in our sense. Given the discussion above, it is enough to verify the condition (4) in the definition above. In the spirit of Theorem 3.1.2, this follows from the observation that if \( \mathcal{C} \) is a small category, the category of elements of a copresheaf \( \text{Elts}(\phi) \) is always small, and thus we can construct the right adjoint \( \Delta^\top \).

6. Future developments

There are two kinds of future developments for this project. To begin with, the new language that we propose allows us to compare, analyze, and design deductive systems.

One one hand, as we have specified in the introduction to this paper, we here only see a couple of possible applications of the framework of judgemental theories, but their versatility suggests many more are possible, for example to modal or linear logic. A taste of the first is already contained in [CE24]. Moreover, as any other calculus, questions of compactness and normalization arise. We believe trying to answer them would lead to interesting insights into both the logic and the category theory.

On the other hand, in Remark 4.3.5 a well-known link between the cut rule and substitution of terms in expressed in our framework. There we suggested many common features of these cut-like phenomena, and study their intrinsic properties. Moreover, it feels like our treatment of substitution might intercept some concepts in [MS21], where a calculus of substitution is introduced by means of composition of certain dinatural transformations. This is a relation that we wish to investigate in future work.

In a different direction, the general theory of judgemental theories shows some possible tweaks and adjustments that may lead to a crisper and sharper presentation.

Firstly, the attentive reader might have noticed that the choice of fixing a given category for contexts is a mere formality, and it actually makes the definitions less smooth that we wished, see for example the discussion pertaining Definition 1.0.4: if anything, this work has convinced us that the notion of context in a logical theory is simply a relative one. We believe that this line of thought and work should be explored further. Nevertheless, we decided to keep the exposition closer to classical presentations as not to make an already cryptic theory appear even more strenuous to follow.

Finally, a recent work by the second author and Osmond [DLO22] shows that 2-categories with finite bilimits can be used to specify many fragments of first-order logic, in such a way that their functorial semantics recovers precisely their theories. We believe there is a possible unification of the theory of judgemental theories (of a certain shape) with the theory introduced in [DLO22], but we shall defer such speculations to future work.
References

[AGH21] Steve Awodey, Nicola Gambino, and Sina Hazratpour. Kripke-Joyal forcing for type theory and uniform fibrations. arXiv preprint arXiv:2110.14576, 2021.

[Awo18] Steve Awodey. Natural models of homotopy type theory. Mathematical Structures in Computer Science, 28(2):241–286, 2018.

[Awo21] Steve Awodey. Univalence in ∞-topoi. https://www.youtube.com/watch?v=wDFDhu1NQhY, 2021.

[BJ81] André Boileau and André Joyal. La logique des topos. The Journal of Symbolic Logic, 46(1):6–16, 1981.

[Bor94] Francis Borceux. Handbook of Categorical Algebra, volume 1 of Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1994.

[Bra21] Martin Brandenburg. Large limit sketches and topological space objects. 2021.

[CE24] Greta Coraglia and Jacopo Emmenegger. A 2-categorical analysis of context comprehension. arXiv preprint arXiv:2403.03085, 2024.

[Cos72] Michel Coste. Langage interne d’un topos. Seminaire Bénabou, Université Paris-Nord, 1972.

[CZ21] Olivia Caramello and Riccardo Zanfa. On the dependent product in toposes. Mathematical Logic Quarterly, 67(3):282–294, 2021.

[DLO22] Ivan Di Liberti and Axel Osmond. Bi-accessible and bipresentable 2-categories. arXiv preprint arXiv:2203.07046, 2022.

[DR21] Francesco Dagnino and Giuseppe Rosolini. Doctrines, modalities and comonads. 2021.

[DT87] Roy Dyckhoff and Walter Tholen. Exponentiable morphisms, partial products and pullback complements. Journal of Pure and Applied Algebra, 49(1):103–116, 1987.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. ii. Mathematische Zeitschrift, 39:405–431, 1935.

[Gen64] Gerhard Gentzen. Investigations into logical deduction. American Philosophical Quarterly, 1(4):288–306, 1964.

[GK13] Nicola Gambino and Joachim Kock. Polynomial functors and polynomial monads. Mathematical Proceedings of the Cambridge Philosophical Society, 154(1):153–192, 2013.

[Gen67] John W. Gray. Fibred and cofibred categories. In S. Eilenberg, D. K. Harrison, S. Mac Lane, and H. Röhrl, editors, Proceedings of the Conference on Categorical Algebra, pages 21–83, Berlin, Heidelberg, 1966. Springer Berlin Heidelberg.

[HLP93] Robert Harper, Furio Honsell, and Gordon Plotkin. A framework for defining logics. Journal of the ACM (JACM), 40(1):143–184, 1993.

[Jac93] Bart Jacobs. Comprehension categories and the semantics of type dependency. Theoretical Computer Science, 107(2):169–207, 1993.

[Jac99] Bart Jacobs. Categorical logic and type theory. Elsevier, 1999.

[Joh77] Peter T. Johnstone. Topos theory, volume 10 of London Mathematical Society Monographs, 1977.

[Kle67] Stephen Cole Kleene. Mathematical Logic. John Wiley & Sons, 1967.

[Law64] F. William Lawvere. An elementary theory of the category of sets. Proceedings of the National academy of Sciences of the United States of America, 52(6):1506, 1964.

[Law70] F. William Lawvere. Equality in hyperdoctrines and comprehension schema as an adjoint functor. Proceedings of the American Mathematical Society, pages 1–14, 1970.

[Law71] F. William Lawvere. Quantifiers as sheaves. In Proc Intern. Congress of Math., pages 1506–1511. Gauthier-Villars, 1971.

[LM05] F. William Lawvere and Colin McLarty. An elementary theory of the category of sets (long version) with commentary. Reprints in Theory and Applications of Categories, 11:1–35, 2005.

[LS88] Joachim Lambek and Philip J. Scott. Introduction to higher-order categorical logic, volume 7. Cambridge University Press, 1988.

[Mai05] Maria E. Maietti. Modular correspondence between dependent type theories and categories including pretopoi and topoi. Mathematical Structures in Computer Science, 15(6):1089–1149, 2005.

[Mak93] Michael Makkai. The fibrational formulation of intuitionistic predicate logic I: completeness according to Gödel, Kripke, and Läuchli, part 2. Notre Dame J. Formal Log., 34:471–498, 1993.
[Mar75] Per Martin-Löf. An intuitionistic theory of types: Predicative part. In H.E. Rose and J.C. Shepherdson, editors, Logic Colloquium ’73, volume 80 of Studies in Logic and the Foundations of Mathematics, pages 73–118. Elsevier, 1975.

[Mar87] Per Martin-Löf. Truth of a proposition, evidence of a judgement, validity of a proof. Synthese, pages 407–420, 1987.

[Mar96a] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. Nordic Journal of philosophical logic, 1(1):11–60, 1996.

[Mar96b] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. Nordic Journal of Philosophical Logic, 1(1):11–60, 1996.

[Mit72] William Mitchell. Boolean topos and the theory of sets. Journal of Pure and Applied Algebra, 2(3):261–274, 1972.

[MLM94] Saunders Mac Lane and Ieke Moerdijk. Sheaves in Geometry and Logic. Springer New York, 1994.

[MP89] Michael Makkai and Robert Paré. Accessible Categories: The Foundations of Categorical Model Theory. American Mathematical Society, 1989.

[MRS13] Maria E. Maietti and Giuseppe Rosolini. Quotient completion for the foundation of constructive mathematics. Logica Universalis, 7(3):371–402, 2013.

[MS84] Per Martin-Löf and Giovanni Sambin. Intuitionistic type theory, volume 9. Bibliopolis Naples, 1984.

[MS21] Guy McCusker and Alessio Santamaria. Composing dinatural transformations: Towards a calculus of substitution. Journal of Pure and Applied Algebra, 225(10):106689, 2021.

[MZ15] Paul-André Melliès and Noam Zeilberger. Functors are type refinement systems. In Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL ’15, page 3–16, New York, NY, USA, 2015. Association for Computing Machinery.

[NvP08] Sara Negri and Jan von Plato. Structural proof theory. Cambridge university press, 2008.

[Osi75a] Gerhard Osius. Logical and set theoretical tools in elementary topoi. In Model Theory and Topoi, pages 297–346. Springer, 1975.

[Osi75b] Gerhard Osius. A note on Kripke-Joyal semantics for the internal language of topos. In Model theory and topos, pages 349–354. Springer, 1975.

[Pit83] Andrew M. Pitts. An application of open maps to categorical logic. Journal of Pure and Applied Algebra, 29:313–326, 1983.

[Res02] Greg Restall. An introduction to substructural logics. Routledge, 2002.

[See84] Robert A. G. Seely. Locally cartesian closed categories and type theory. In Mathematical proceedings of the Cambridge philosophical society, volume 95, pages 33–48. Cambridge University Press, 1984.

[See86] Robert A. G. Seely. Modelling Computations: a 2-categorical Framework. The College, 1986.

[Str74] Ross Street. Elementary cosmoi I. In Category Seminar, pages 134–180. Springer, 1974.

[Str80] Ross Street. Cosmoi of internal categories. Transactions of the American Mathematical Society, 258(2):271–318, 1980.

[Str82] Ross Street. Characterization of bicategories of stacks. In Category Theory, pages 282–291. Springer, 1982.

[Tar56] Alfred Tarski. The concept of truth in formalized languages. Logic, semantics, metamathematics, 2(152-278):7, 1956.

[Tay99] Paul Taylor. Practical Foundations of Mathematics. Number v. 59 in Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1999.

[TS00] Anne S. Troelstra and Helmut Schwichtenberg. Basic proof theory. Number 43. Cambridge University Press, 2000.

[Uem23] Taichi Uemura. A general framework for the semantics of type theory. Mathematical Structures in Computer Science, 33(3):134–179, 2023.

[Wad15] Philip Wadler. Propositions as types. Communications of the ACM, 58(12):75–84, 2015.

[Web07] Mark Weber. Yoneda structures from 2-toposes. Applied Categorical Structures, 15(3):259–323, 2007.

[Wit22] Ludwig Wittgenstein. Tractatus logico-philosophicus. London: Routledge, 1981, 1922.
