A general definition of dependent type theories

Andrej Bauer   Philipp G. Haselwarter
Peter LeFanu Lumsdaine

September 11, 2020

Abstract

We define a general class of dependent type theories, encompassing Martin-Löf's intuitionistic type theories and variants and extensions. The primary aim is pragmatic: to unify and organise their study, allowing results and constructions to be given in reasonable generality, rather than just for specific theories. Compared to other approaches, our definition stays closer to the direct or naïve reading of syntax, yielding the traditional presentations of specific theories as closely as possible.

Specifically, we give three main definitions: *raw type theories*, a minimal setup for discussing dependently typed derivability; *acceptable type theories*, including extra conditions ensuring well-behavedness; and *well-presented type theories*, generalising how in traditional presentations, the well-behavedness of a type theory is established step by step as the type theory is built up. Following these, we show that various fundamental fitness-for-purpose metatheorems hold in this generality.

Much of the present work has been formalised in the proof assistant Coq.

Contents

1 Introduction

1.1 Overview

We give a general definition of dependent type theories, encompassing for example Martin-Löf's intuitionistic type theories and many variants and extensions.

The primary aim is to give a setting for formal general versions of various constructions and results, which in the literature have been given for specific theories but are heuristically understood to hold for a wide class of theories: for instance, the conservativity theorem of (Hofmann, 1997), or the coherence theorems of (Hofmann, 1995; Lumsdaine and Warren, 2015).

This has been a sorely felt gap in the literature until quite recently; the present work is one of several recent approaches to filling it (Isaev, 2017; Uemura, 2019; Brunerie, 2020b).

A secondary aim is to stick very closely to an elementary understanding of syntax. Established general approaches — for instance, logical frameworks and categorical semantics — give, in examples, not the original syntax of the example theories, but an embedded or abstracted version, typically then connected to the original syntax by adequacy or initiality theorems. Our approach directly recovers quite conventional presentations of the example theories themselves.

As a corollary of this goal, we must confront the bureaucratic design decisions of syntax: the selection of structural rules, and so on. These are often swept under the rug in specific type theories as "routine"; to initiates they are indeed standard, but newcomers to the field often report finding this lack of detail difficult. We therefore elide nothing, and set out a precise choice of all such decisions, carefully chosen and proven to work well in reasonable generality, which we hope will be of value to readers.

In pursuit of the above aims, we offer not one main definition of type theory, but three, at increasing levels of refinement.

Firstly, we define *raw type theories*, as a conceptually minimal description of traditional presentations of type theories by symbols and rules, sufficient to define the derivability relation, but not yet incorporating any well-formedness constraints on the rules.

Secondly, we give sufficient conditions on a raw type theory to imply that derivability over it is well-behaved in various standard ways. Specifically, we isolate simple syntactic checks that suffice to imply core fitness-for-purpose properties, and package these into the notion of an *acceptable type theory*.

Thirdly, we analyse the well-founded nature of traditional presentations, involved in more elaborate constructions such as the categorical semantics, as well as (arguably) the intuitive assignment of meaning to a theory. This leads us to the notion of *well-presented type theories*, which we hope can serve as a full-fledged proposal fulfilling our primary aim.

1.2 Specifics

We aim, as far as possible, not to argue for any novel approach to setting up type theories, but simply to give a careful analysis of how type theories are traditionally presented, in order to lay out a generality in which such presentations can be situated. As such, the first few components of our definition are the expected ones.

We begin with an appropriate notion of *signature*, for untyped syntax with variable-binding, and develop the standard notions of "raw" *syntactic expressions* over such

signatures, including substitution, translation along signature morphisms, and so on.

With the syntax of types and terms properly set up, a type theory is traditionally presented by giving a collection of *rules*. Type theorists are very accustomed to reading these — but as anyone who has tried to explain type theory to a non-initiate knows, there is a lot to unpack here. The core of our definition is a detailed study of the situation: what is really going on when we write and read inference rules, check that they are meaningful, and interpret them as a presentation of type theory?

Take the formation rule for Π-types:

$$\frac{\Gamma \vdash A \text{ type} \qquad \Gamma, x{:}A \vdash B \text{ type}}{\Gamma \vdash \Pi(x{:}A) . B \text{ type}}$$

When pressed to explain this, most type theorists will say that the rule represents *inductive clauses* for constructing derivations, or *closure conditions* for the derivability predicate: given derivations of the judgements above the line, a derivation of the judgement below the line is constructed. In particular, if Γ, A and B are syntactically valid representations of a context and types, and the judgements

$$\Gamma \vdash A \text{ type,} \qquad \text{and} \qquad \Gamma, x{:}A \vdash B \text{ type}$$

are both derivable, then so is the judgement

$$\Gamma \vdash \Pi(x{:}A) . B \text{ type.}$$

This understanding of rules is sufficient for explaining the definition of a specific type theory, and defining derivability of judgements.

However, it is in general too permissive: to be well-behaved, type theories should not be given by arbitrary closure conditions, but only by those that can be specified syntactically by rules looking something like the traditional ones. In other words, we want to make explicit the idea of a rule as a syntactic entity in its own right, accompanied with a mathematically precise explanation of what makes it type-theoretically acceptable, and how it gives rise to a quantified family of closure conditions.

So to a first approximation, we say a rule consists of a collection of judgements — its *premises* — and another judgement, its *conclusion*. However, a subtlety lurks: what are Γ, A, and B in the above Π-formulation rule?

The symbol Γ is easy: we can dispense with it entirely. We prescribe (as type theorists often do, heuristically) that all rules should be valid over arbitrary contexts, and so since the arbitrary context is always present in the interpretation of a rule as a family of closure conditions, it never needs to be included in the syntactic specification of the rule. This precisely justifies a common "abuse of notation" in presenting type theories: the context Γ is omitted when writing down the rules, and one mentions apologetically somewhere that all rules should be understood as over an arbitrary ambient context.

Explaining A and B is more interesting. They are generally called "metavariables", and in the family of closure conditions they are indeed that — quantified variables of the meta-theory, ranging over syntactic entities. However, if the rule is to be considered as formal syntactic entity, A and B must themselves be part of that syntax. We therefore take the premises and conclusion of rules as formed over the ambient signature of the type theory *extended* with extra symbols A and B to represent the metavariables of the rule.

These considerations result in the notion of a *raw rule*, whose premises and conclusion are raw judgements over a signature extended with metavariables. A *raw type*

theory is then just a family of raw rules. It holds enough information to be used, but still permits arbitrariness that must be dispensed with. At the very least, the type and term expressions appearing in the rule ought to represent derivable types and terms, respectively. Thus in the next stage of our definition we ask that every rule be accompanied with derivations showing that the *presuppositions* hold, namely, that its type expressions are derivable types and that its terms have derivable types. Another condition that we impose on rules is *tightness*, which roughly requires that the metavariables symbols be properly typed by the premises, and for rules that build term or a type judgements, that they do so in the most general form.

Even though every rule of a raw type theory may be presuppositive and tight, the theory as a whole may be deficient, for example, if one of the symbols has no corresponding formation rule, or several of them. Overall we call a type theory *tight* when there is bijective correspondence between its symbols and formation rules, which are tight themselves. And to make sure equality is well behaved, we also require that for each symbol there is a suitable congruence rule ensuring that the symbol commutes with equality. When a raw type theory has all these features, we call it an *acceptable* type theory.

Because the derivations of presuppositions appeal to the very rules they certify, an unsettling possibility of circular reasoning arises. We resolve the matter in two ways. First, we add one last stage to the definition of type theories and ask that all the rules, as well as the premises within each rule, be ordered in a well-founded manner. Second, we show that for acceptable type theories whose contexts and premises are well-founded as finite sequences, circularities can always be avoided by passing to the *well-founded replacement* of the theory (Section 6.5). Apart from expelling the daemons of circularity, the well-founded order supplies a useful induction principle.

In the end, the definition of a general type theory has roughly five stages:

1. the *signature* (Definition 3.8) describes the arities of primitive type and term symbols that form the *raw syntax* (Definition 3.11),

2. *raw rules* (Definition 4.18) constitute a *raw type theory* (Definition 4.36),

3. the raw rules are verified to be *tight* and *presuppositive* (Definitions 5.1 and 5.6, and therefore *acceptable* (Definition 5.7),

4. the raw type theory is verified to consist of acceptable symbol rules (Definition 5.11) and equations, that it is *tight* and *congruous*, and therefore *acceptable* (Definition 5.12).

5. finally, an acceptable type theory may be *well-presented* (Definition 6.20) and hence *well-founded* (Definition 6.19), or we may pass to its *well-founded replacement* (Theorem 6.32).

We readily acknowledge that there are many alternative ways of setting up type theories, each serving a useful purpose. It is simply our desire to actually give *one* mathematically complete description of what type theor*ies* are in general.

Once the definition is complete, we should provide evidence of its scope and utility. We do so in Section 5 by proving fundamental meta-theorems, among which are:

1. Derivability of presuppositions, Theorem 5.15, stating that the presuppositions of a derivable judgement are themselves derivable.

2. Elimination of substitution, Theorem 5.22, stating that anything that can be derived using the substitution rules can also be derived without them.

3. Uniqueness of typing, Theorem 5.23, stating that a term has at most one type, up to judgmental equality.

4. An inversion principle, Theorem 5.27, that reconstructs the proof-relevant part of the derivation of a derivable judgement from the information given in the judgement.

Our definitions are set up to support a meta-theoretic analysis of type theories, but deviate from how type theories are presented in practice. First, one almost always encounters only finitary syntax in which contexts and premises are presented as finite sequences – we call these *sequential contexts and premises* and treat them in Sections 6.1 and 6.2. Second, theories are not constructed in five stages, but presented through rules that are manifestly acceptable and free of circularities, while symbols are introduced simultaneously with their formation rules. In Sections 6.3 and 6.4 we make these notions precise by defining *well-presented rules and theories*, and their realisations as raw type theories.

Having heard tales about minor but insidious mistakes in the literature on the metatheory of syntax, we decided to protect ourselves from them by formalising parts of our paper in the Coq proof assistant (Coq development team, 2020). An overview of the formalisation is given in Appendix A, including comments about the metamathematical foundations sufficient for carrying out our work. The formalisation allows us to claim a high level of confidence and omission of routine syntactic arguments. Nevertheless, we still strove to make the paper self-contained by following the established standards of informal rigour.

1.3 Disclaimers

Having said what this paper is about, it is worth saying a little about what it is *not*.

It is most certainly not intended as a prescriptive definition of what all dependent type theories should be. Many important type theories in the literature are not covered by our definition, and we do not mean to reject them. The aim of this work is simply pragmatic: to encompass *some large class* of theories of interest, in order to better organise and unify their study. We very much hope our approach may be extended to wider generalities.

We do not claim or aim to supersede other general approaches to studying type theories, such as those based on logical frameworks. Such approaches are well-developed, powerful for many applications, and sidestep some complications of the present approach. However, all such approaches (that we are aware of) work by using a somewhat modified syntax (e.g. embedded in a larger system) — the syntax they yield is not obviously the same as the syntax given by a "direct" or "naïve" reading of presentations of theories. They are typically accompanied by adequacy theorems, or similar, showing equivalence between the modified syntax and the naïve, for the specific type theory under consideration.

By contrast, we aim to directly study and generalise the naïve approach itself, which (to our knowledge) has not been done previously in such generality. Our motivations are therefore largely complementary to such approaches. A more detailed comparison is given in Section 7.

5

2 Preliminaries

We begin by setting up definitions and terminology of a general mathematical nature that we will use throughout.

2.1 Families

For several reasons, we work with *families* in places where classical treatments would use either *subsets* of, or *lists* from, a given set. While the term is standard (e.g. "the product of a family of rings"), we make rather more central use of it than is usual, so we establish some notations and terminology.

Definition 2.1. Given a set X, a **family** K **of elements of** X (or briefly, a **family on** X) consists of an index set $\operatorname{ind} K$ and a map $\operatorname{ev}_K : \operatorname{ind} K \to X$. We let $\operatorname{Fam} X$ denote the collection of all families on X, and use the **family comprehension** notation $\langle e_i \in X \mid i \in I \rangle$ for the family indexed by I that maps i to e_i. A family may be explicitly described by displaying the association of indices to values. For example, we may write $\langle 0\colon e_0, 1\colon e_1, 2\colon e_2 \rangle$ for the family $\langle e_i \mid i \in \{0, 1, 2\} \rangle$.

Example 2.2. Any *subset* $A \subseteq X$ can be viewed as a family $\langle i \in X \mid i \in A \rangle$, with $\operatorname{ind} A$ as A itself and ev_A the inclusion $A \hookrightarrow X$. Motivated by this, we will often speak of a family K as if it were a subset, writing $x \in K$ rather than $x \in \operatorname{ind} K$, and treating such x itself as an element of X rather than explicitly writing $\operatorname{ev}_K(x)$.

Example 2.3. Any *list* $\ell = [x_0, \ldots, x_n]$ of elements of X can be viewed as a family, with $\operatorname{ind} \ell = \{0, \ldots, n\}$ and $\operatorname{ev}_\ell(i) = x_i$, or equivalently $\ell = \langle 0\colon x_0, \ldots, n\colon x_n \rangle$. We will often use list notation to present concrete examples of families.

Working constructively, it is quite important to keep the distinction between families and subsets where classical treatments would confound them. For instance, a propositional theory is usually classically defined as a *set* of propositions; we would instead use a *family* of propositions. In a derivation over the theory, uses of axioms therefore end up "tagged" with elements of the index set of the theory, typically explaining how a certain proposition arises as an axiom (since the same proposition might occur as an instance of axiom schemes in multiple ways). These record constructive content which may be needed for, say, interpreting axioms according to a proof by cases over the axiom schemes of the theory.

Our use of families where most traditional treatments use lists — e.g. for specifying the argument types of a constructor — is less mathematically significant. It is partly to avoid baking in assumptions of finiteness or ordering where they are not required; but it is mostly motivated just by the formalisation, where families provide a more appropriate abstraction.

Definition 2.4. A **map of families** $f : K \to L$ between families K and L on X is a map $f : \operatorname{ind} K \to \operatorname{ind} L$ such that $\operatorname{ev}_L \circ f = \operatorname{ev}_K$.

We shall notate such a map as $\langle f(x) \rangle_{x \in \operatorname{ind} K}$. Indeed, the notation $\langle f(x) \rangle_{x \in A}$ works for *any* maps $f : A \to B$, as it is just an alternative way of writing λ-abstractions.

Families and their maps form a category $\operatorname{Fam} X$, which is precisely the slice category Set/X. A map $r : X \to Y$ yields a functorial action $r_* : \operatorname{Fam} X \to \operatorname{Fam} Y$ which takes $K \in \operatorname{Fam} X$ to the family $r_* K$ with $\operatorname{ind}(r_* K) = \operatorname{ind} K$ and $\operatorname{ev}_{r_* K} = r \circ \operatorname{ev}_K$. It is perhaps clearer to write down the action in terms of family comprehension: $r_* \langle e_i \mid i \in I \rangle = \langle r(e_i) \mid i \in I \rangle$.

Definition 2.5. Given a function $r : X \to Y$ and families K, L on X, Y respectively, a *map* $f : K \to L$ *over* r is a map $f : r_* K \to L$; equivalently, a map $f : \operatorname{ind} K \to \operatorname{ind} L$ forming a commutative square over r.

2.2   Closure systems

The general machinery of derivations as closure systems occurs throughout logic, and is independent of the specific syntax or judgements of the logical systems involved.

Definition 2.6. A *closure rule* (P, c) on a set X consists of a family P of elements in X, its *premises*, and a *conclusion* $c \in X$. A *closure system* $\mathcal{S}$ on a set X is a family of closure rules on X, where we respectively write $\operatorname{prems} R$ and $\operatorname{concl} R$ for the premises and the conclusion corresponding to a rule $R \in \mathcal{S}$. We write $\operatorname{Rule} X$ and $\operatorname{Clos} X$ for the collections of closure rules and closure systems on X, respectively.

As is tradition, we display a closure rule with premises $[p_1, \ldots, p_n]$ and conclusion c as

$$\frac{p_1 \quad \cdots \quad p_n}{c}$$

The constructions of closure rules and closure systems are evidently functorial in the ambient set. A map $f : X \to Y$ sends a rule $R \in \operatorname{Rule} X$ to the rule $f_* R \in \operatorname{Rule} Y$ with $\operatorname{prems}(f_* R) := \langle f(p) \mid p \in \operatorname{prems} R \rangle$ and $\operatorname{concl}(f_* R) := f(\operatorname{concl} R)$. Similarly, a closure system $\mathcal{S}$ on X is taken to the closure system $f_* \mathcal{S}$ on Y, defined by $f_* \mathcal{S} := \langle f_* r \mid r \in \mathcal{S} \rangle$.

Definition 2.7. A *simple map* $\mathcal{S} \to \mathcal{T}$ between closure systems $\mathcal{S}$ and $\mathcal{T}$ on X is just a map between them as families. More generally, a *simple map* $\bar{f} : \mathcal{S} \to \mathcal{T}$ *over* $f : X \to Y$ from $\mathcal{S} \in \operatorname{Clos} X$ to $\mathcal{T} \in \operatorname{Clos} Y$ is just a simple map $\bar{f} : f_* \mathcal{S} \to \mathcal{T}$, or equivalently a family map $\bar{f}$ over $f_* : \operatorname{Rule} X \to \operatorname{Rule} Y$.

A closure system yields a notion of derivation:

Definition 2.8. Given a closure system $\mathcal{S}$ on X, a family H of elements in X, and an element $c \in X$, the *derivations* $\operatorname{Der}_{\mathcal{S}}(H, c)$ *of c from hypotheses* H are inductively generated by:

1. for every $h \in H$, there is a corresponding derivation $\operatorname{hyp} h \in \operatorname{Der}_{\mathcal{S}}(H, h)$,

2. for every rule $R \in \mathcal{S}$ and a map $D \in \prod_{p \in \operatorname{prems} R} \operatorname{Der}_{\mathcal{S}}(H, p)$ there is a derivation $\operatorname{der}(R, D) \in \operatorname{Der}_{\mathcal{S}}(H, \operatorname{concl} R)$.

In the second clause above D is a dependent map, i.e., for each $p \in \operatorname{prems} R$ we have $D_p \in \operatorname{Der}_{\mathcal{S}}(H, p)$. We do not shy away from using products of families and dependent maps when the situation demands them.

The elements of $\operatorname{Der}_{\mathcal{S}}(H, c)$ may be seen as well-founded trees with edges and nodes suitably labelled from $X, \mathcal{S}$, and H. We take such inductively generated families of sets as primitive; their existence may be secured one way or another, depending on the ambient mathematical foundations. The essential feature of derivations, which we rely on, is the structural induction principle they provide.

It is easy to check that derivations are functorial in simple maps of closure systems, in a suitable sense:

Proposition 2.9. *A simple map $\bar{f} : \mathcal{S}_X \to \mathcal{S}_Y$ of closure systems over $f : X \to Y$ acts on derivations as $\bar{f}_* : \mathrm{Der}_{\mathcal{S}_X}(H, c) \to \mathrm{Der}_{\mathcal{S}_Y}(f_*H, f(c))$ for each H and c. The action is moreover functorial, in that $\mathrm{id}_* = \mathrm{id}$ and $(\bar{f} \circ \bar{g})_* = \bar{f}_* \circ \bar{g}_*$.* $\hfill\square$

Often, one wants a more general notion of map, sending each rule of the source system not necessarily to a single rule of the target system, but instead to a *derived* rule:

Definition 2.10. A ***derivation of a rule*** R over a closure system $\mathcal{S}$ is a derivation of $\mathrm{concl}\, R$ from $\mathrm{prems}\, R$ over $\mathcal{S}$. Given such a derivation, we call R a ***derived rule*** of $\mathcal{S}$, or say R is ***derivable over*** $\mathcal{S}$. A ***map of closure systems*** $\bar{f} : \mathcal{S} \to \mathcal{T}$ over $f : X \to Y$ is a function giving, for each rule R of C, a derivation of f_*R in $\mathcal{T}$.

To show that maps of closure systems preserve derivability, we need a *grafting* operation on derivations.

Lemma 2.11. *Given an ambient closure system $\mathcal{S}$, suppose D is a derivation of c from hypotheses H over $\mathcal{S}$, and for each $h \in H$, D_h is a derivation of h from H'. Then there is a derivation of c from H' over $\mathcal{S}$.*

Proof. The derivation of c from H' is constructed inductively from a derivation of c from H:

1. if c is derived as one of the hypotheses $h \in H$, then D_h derives c from H',

2. if $\mathrm{der}(R, D')$ derives c from H, then for each $p \in \mathrm{prems}\, R$ we inductively obtain a derivation D_p'' of p from H' from the corresponding derivation D_p' of p from H, and assemble these into the derivation $\mathrm{der}(R, D'')$ of c from H'. $\hfill\square$

Definition 2.12. A closure system map $\bar{f} : \mathcal{S} \to \mathcal{T}$ over $f : X \to Y$ ***acts on derivations***: if D is a derivation of c from H over $\mathcal{S}$, there is a derivation $\bar{f}_*D$ of $f(c)$ from f_*H over $\mathcal{T}$.

Proof. $\bar{f}_*D$ is defined by recursion on D. Wherever D uses a rule R of $\mathcal{S}$, with derivations D_h of the premises, $\bar{f}_*D$ uses the given derivation $\bar{f}(R)$ of f_*R, with the derivations $\bar{f}_*D_h$ grafted in at the hypotheses. $\hfill\square$

Categorically, grafting can be recognised as the multiplication operation of a monad structure on derivations, and our maps of closure systems can be seen as Kleisli maps for this monad (relative to simple maps). One can thus show that they form a category, that the action on derivations is functorial, and so on. We do not make this precise here, as it is not required for the present paper.

2.3 Well-founded orders

There will be several occasions when we shall have to prevent dependency cycles (between premises of a rule, or between rules of a type theory). For this purpose we review a notion of well-foundedness which accomplishes the task.

Definition 2.13. A ***strict partial order*** on a set A is an irreflexive and transitive relation $<$ on A. A subset $S \subseteq A$ is ***$<$-progressive*** when, for all $x \in A$,

$$(\forall y \in A \,.\, y < x \Rightarrow y \in S) \Rightarrow x \in S.$$

A ***well-founded order*** is a strict partial order $<$ in which a subset is the entire set as soon as it is $<$-progressive. For each $x \in A$, the ***initial segment*** $\downarrow i := \{y \in A \mid y < x\}$ is the set of elements preceding x with respect to the order.

In terms of an induction principle a strict partial order is well-founded when, for every predicate φ on A,

$$\forall x \in A \,.\, (\forall y \in A \,.\, y < x \Rightarrow \varphi(y)) \Rightarrow \varphi(x) \Rightarrow \forall x \in A \,.\, \varphi(x).$$

Classically there are many equivalent definitions of well-founded orders. Constructively, the situation is more complicated, cf. (Taylor, 1999, §2.5); this definition is one of the most standard, and the most suited to our purpose.

3 Raw syntax

In this section, we set out our treatment of raw syntax with binding, sometimes called "pre-syntax" to indicate that no typing information is present at this stage. There is nothing essentially novel — briefly, we use a standard modern treatment, closely inspired by that of (Fiore et al., 1999), but focus on concrete constructions rather than categorical characterisations. So we take raw expressions as inductively generated trees, and scope systems, developed below, for keeping track of variable scopes and binding. We spell out the details in order to have a self-contained presentation tailored to our requirements, and to set up terminology and notation we will use later.

3.1 Scope systems

We first address the question of how to treat variables and binding. Should we use terms with named variables up to α-equivalence, or de Bruijn indices, or reuse the binding structure of a framework language? The last option is appealing, as it dispenses with many cumbersome details, but we shall avoid it precisely because we want to confront the cumbersome details of type theory.

Rather than choosing a particular answer, we formulate and use a general structure for binding, abstracting away the implementation-specific details of several approaches, but retaining the common structure required for defining syntax.

Definition 3.1. A *scope system* consists of:

1. a collection of *scopes* $\mathcal{S}$;

2. for each scope γ, a set of *positions* $|\gamma|$;

3. an *empty scope* 0 with no positions, $|0| = \emptyset$;

4. a *singleton scope* 1 with a unique position, $|1| = 1$;

5. operations giving for all scopes γ and δ a *sum scope* $\gamma \oplus \delta$, and functions

$$|\gamma| \xrightarrow{\ \text{inl}\ } |\gamma \oplus \delta| \xleftarrow{\ \text{inr}\ } |\delta|$$

   exhibiting $|\gamma \oplus \delta|$ as a coproduct of $|\gamma|$ and $|\delta|$.

A scope may be seen as "a context, without the type expressions": in raw syntax, one cares about what variables are in scope, without yet caring about their types.

The singleton scope 1 is not needed for most of the development of general type theories — in the present paper, it is used only for sequential contexts (Section 6.1) and notions building on these. However, it is present in all examples of interest, so we include it in the general definition.

We will also use scopes to describe *binders*: if some primitive symbol S binds γ variables in its i-th argument, then for an instance of S in scope δ, the i-th argument will be an expression in scope $\delta \oplus \gamma$. Most traditional constructors bind finitely many variables; to facilitate this, we let $[n]$ denote the sum of $n \in \mathbb{N}$ copies of 1, which also provides alternative notations $[0]$ and $[1]$ for the empty and singleton scopes, respectively.

Example 3.2. *De Bruijn indices* and *de Bruijn levels* can be seen as scope systems, with $\mathbb{N}$ as the set of scopes, $|n| := \{0, 1, \ldots, n-1\}$, $0 := 0$, and $m \oplus n := m + n$.

The difference lies just in the choice of coproduct inclusions $\mathrm{inl} : |m| \to |m + n| \leftarrow |n| : \mathrm{inr}$. Setting $\mathrm{inl}(i) := i + n$, $\mathrm{inr}(j) = j$ gives de Bruijn *indices*, as going under a binder increments the variables outside it. Setting $\mathrm{inl}(i) := i$, $\mathrm{inr}(j) := j + n$ gives de Bruijn *levels*, as higher positions go to the innermost-bound variables. Over these scope systems, our syntax precisely recovers standard de Bruijn-style syntax, as in (de Bruijn, 1972) and subsequent work.

Both the de Bruijn scope systems are *strict* in the sense that we have equalities $\gamma \oplus 0 = \gamma = 0 \oplus \gamma$ and $(\gamma \oplus \delta) \oplus \eta = \gamma \oplus (\delta \oplus \eta)$, whereas general scope systems provide only canonical isomorphisms. The equalities help reduce bureaucracy in several proofs, so we shall occasionally indulge in assuming that we work with a strict scope system. The doubtful reader may consult the formalisation, which makes no such assumptions.

Example 3.3. The *finite sets* system takes scopes to be finite sets, along with any choice of coproducts. It may be prudent to restrict to a small collection, say the hereditarily finite sets. Syntax over these scope systems gives a concrete implementation of categorical approaches such as (Fiore et al., 1999).

Example 3.4. Scope systems are not intrinsically linked to dependent type theories, but provide a useful discipline for syntax of other systems. For instance, in geometric logic, the infinitary disjunction is usually given with a side condition that the free variables of the disjunction must remain finite (Johnstone, 2002, D1.1.3(xi)). By using finite scopes, we can make the finiteness condition explicit from the start, and dispense with the side condition. This is similar in spirit to (Fiore et al., 1999) and more closely mirrors the categorical semantics. By contrast, the classical Hilbert-type logics $\mathcal{L}_{\alpha, \beta}$ of (Karp, 1964) allow genuinely infinite contexts and binders, which can be obtained by taking scopes to be ordinals $\delta \in \alpha$, with $|\delta| := \delta$.

Example 3.5. The traditional syntax using named variables, for both free and bound variables, is *not* an example of a scope system. In that approach, a fresh variable is not introduced by summing with a scope, but rather by a multivalued map allowing extension by *any* unused name.

Some implementations of syntax treat free and bound variables separately, for instance *locally nameless syntax* (McKinna and Pollack, 1993) uses concrete names for free variables but de Bruijn indices for bound variables. Scope systems as defined here do not subsume such syntax, but could be generalised to do so.

Categorically, a scope system can be viewed precisely as a category Scope with initial and terminal objects, binary coproducts, and a full and faithful functor into $(\mathrm{Set}, 0, +, 1)$ preserving this structure. The categorical structure is induced from Set: morphisms $\gamma \to \delta$ are taken as functions $|\gamma| \to |\delta|$ — we call these *renamings*. Two of these, $r : \gamma \to \delta$ and $r' : \gamma' \to \delta'$, give a sum map $r \oplus r' : \gamma \oplus \gamma' \to \delta \oplus \delta'$ arising from the universal property of coproducts.

For the remainder of the paper, we fix a scope system. To make concrete examples readable, we shall write them using the de Bruijn scopes from Example 3.2. They can be easily adapted to any other scope system.

3.2 Arities and signatures

While the arity of a simple algebraic operation is just the number of its arguments, the situation is complicated in type theory by the presence of *binders*. Each argument of a type-theoretic constructor may be a term or a type, and some of its variables may be bound by the constructor. We thus need a suitable notion of arity.

Definition 3.6. By *syntactic classes*, we mean the two formal symbols ty and tm, representing *types* and *terms* respectively. An *arity* α is a family of pairs (c, γ) where c is a syntactic class and γ is a scope. We call the indices of α *arguments* and write $\arg \alpha$ for the index set of α. Thus, each argument $i \in \arg \alpha$ has an associated syntactic class $\mathrm{cl}_\alpha\, i$ and a scope $\mathrm{bind}_\alpha\, i$, which we call the *binder* associated with the argument i, and we can write α as $\alpha = \langle (\mathrm{cl}_\alpha\, i, \mathrm{bind}_\alpha\, i) \mid i \in \arg \alpha \rangle$.

Example 3.7. In Martin-Löf type theory with the de Bruijn scope system, the constructor Π has arity $[(\mathsf{ty}, 0), (\mathsf{ty}, 1)]$. That is, the arity has two type arguments, and binds one variable in the second argument. A simpler example is the successor symbol in arithmetic whose arity is $[(\mathsf{tm}, 0)]$, because it takes one term argument and binds nothing. Still simpler, the arity of a constant symbol is the empty family.

Note that arities express only the basic syntactic information and do not specify the types of term arguments and bound variables, which will be encoded later by typing rules.

Definition 3.8. A *signature* Σ is a family of pairs of a syntactic class and an arity. We call the elements of its index set *symbols*. Thus each symbol $S \in \Sigma$ has an associated syntactic class c_S and an arity α_S. A *type symbol* is one whose associated syntactic class is ty, and a *term symbol* is one whose associated syntactic class is tm. The *arguments* $\arg S$ of S are the arguments of its arity α_S. Each argument $i \in \arg S$ has an associated syntactic class $\mathrm{cl}_S\, i$ and binder $\mathrm{bind}_S\, i$, as prescribed by the arity α_S.

Example 3.9. The following signature describes the usual constructors for dependent products:

$$
\begin{aligned}
\Pi &\mapsto (\mathsf{ty}, [(\mathsf{ty}, 0), (\mathsf{ty}, 1)]), \\
\lambda &\mapsto (\mathsf{tm}, [(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 1)]), \\
\mathsf{app} &\mapsto (\mathsf{tm}, [(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 0), (\mathsf{tm}, 0)]).
\end{aligned}
$$

Let us spell out the last line. The symbol app builds a term, because its syntactic class is tm, from four arguments. The first and the second arguments are types, with one variable bound in the second argument, while the third and the fourth arguments are terms. We thus expect an application term to be written as $\mathsf{app}(A, B, s, t)$, with one variable getting bound in B.

Definition 3.10. A *signature map* $F : \Sigma \to \Sigma'$ is a map of families between them, that is, a function from symbols of Σ to symbols of Σ', preserving the arities and syntactic classes. Signatures and maps between them form a category Sig.

3.3 Raw syntax

Once a signature Σ is given, we know how to build type and term expressions over it. We call this part of the setup "raw" syntax to emphasise its purely syntactic nature.

Definition 3.11. The *raw syntax* over Σ consists of the collections of *raw type expressions* $\mathrm{Expr}_\Sigma^{\mathsf{ty}}(\gamma)$ and *raw term expressions* $\mathrm{Expr}_\Sigma^{\mathsf{tm}}(\gamma)$, which are generated inductively for any scope γ as follows:

1. for every position $i \in |\gamma|$, there is a *variable* expression $\mathsf{var}_i \in \mathrm{Expr}_\Sigma^{\mathsf{tm}}(\gamma)$;

2. for every symbol $S \in \Sigma$ of syntactic class c_S, and a map

$$e \in \textstyle\prod_{i \in \arg S} \mathrm{Expr}_\Sigma^{\mathrm{cl}_S\, i}(\gamma \oplus \mathrm{bind}_S i),$$

there is an expression $S(e) \in \mathrm{Expr}_\Sigma^{\mathrm{c}_S}(\gamma)$, the *application* of S to arguments e.

Let us walk through the definition. The first clause states that the positions of γ play the role of available variable names, still without any typing information. The second clause explains how to build an expression with a symbol S: for each argument $i \in \arg S$, an expression e_i of a suitable syntactic class must be provided, where e_i may refer to variable names given by γ as well as the variables that are bound by S in the i-th argument. The expressions e_i are conveniently collected into a function e. When writing down concrete examples we write the arguments as tuples.

Example 3.12. The symbol Π has arity $[(\mathsf{ty}, 0), (\mathsf{ty}, 1)]$. So if A is a type expression with free variables amongst $\gamma \oplus 0$ (which is isomorphic to γ), and B is a type expression with free variables in $\gamma \oplus 1$ (which has an extra free variable available), then $\Pi(A, B)$ is a type expression with free variables in γ.

Definition 3.13. The *action of a renaming* $r : \gamma \to \delta$ on expressions is the map $r_* : \mathrm{Expr}_\Sigma^c(\gamma) \to \mathrm{Expr}_\Sigma^c(\delta)$, defined by structural recursion:

$$r_*(\mathsf{var}_i) := \mathsf{var}_{r(i)},$$
$$r_*(S(e)) := S(\langle (r \oplus \mathrm{id}_{\mathrm{bind}_S i})_*(e_i) \rangle_{i \in \arg S}).$$

Note how the definition uses the functorial action of $\oplus$ to extend the renaming r when it descends under the binders of a symbol.

Definition 3.14. The *action of a signature map* $F : \Sigma \to \Sigma'$ on expressions is the map $F_* : \mathrm{Expr}_\Sigma^c(\gamma) \to \mathrm{Expr}_{\Sigma'}^c(\gamma)$, defined by structural recursion:

$$F_*(\mathsf{var}_i) := \mathsf{var}_i,$$
$$F_*(S(e)) := F(S)(F_* \circ e).$$

Proposition 3.15. *The actions by renamings and signature maps commute with each other, and respect identities and composition. That is, they make expressions into a functor* $\mathrm{Expr} : \mathrm{Sig} \times \mathrm{Scope} \to \mathrm{Set}^{\{\mathsf{ty},\mathsf{tm}\}}$. $\qquad\square$

3.4 Substitution

We next spell out substitution as an operation on raw expressions, and note its basic properties.

Definition 3.16. A *raw substitution* $f : \gamma \to \delta$ over a signature Σ is a map $f : |\delta| \to \mathrm{Expr}^{\mathsf{tm}}_\Sigma(\gamma)$. The *extension* $f \oplus \eta : \gamma \oplus \eta \to \delta \oplus \eta$ *by a scope* η is the substitution

$$
\begin{aligned}
(f \oplus \eta)(\mathrm{inl}(i)) &:= \mathrm{inl}_*(f(i)) && \text{if } i \in |\delta|, \\
(f \oplus \eta)(\mathrm{inr}(j)) &:= \mathsf{var}_{\mathrm{inr}(j)} && \text{if } j \in |\eta|.
\end{aligned}
$$

The (contravariant) *action of f on an expression* $e \in \mathrm{Expr}^{\mathsf{c}}_\Sigma(\delta)$ gives the expression $f^* e \in \mathrm{Expr}^{\mathsf{c}}_\Sigma(\gamma)$, as follows:

$$
\begin{aligned}
f^*(\mathsf{var}_i) &:= f(i), \\
f^*(S(e)) &:= S(\langle (f \oplus \mathrm{bind}_S i)^* e_i \rangle_{i \in \arg S}).
\end{aligned}
$$

Example 3.17. The above definition, specialized to the de Bruijn scope systems of Example 3.2, precisely recovers the usual definition of substitution with de Bruijn indices or levels. Their explicit shift operators are abstracted away, in our setup, as renaming under coproduct inclusions.

Definition 3.18. Any renaming $r : \gamma \to \delta$ induces a substitution $\bar{r} : \delta \to \gamma$, with $\bar{r}(i) := \mathsf{var}_{r(i)}$. In particular, each scope δ has an *identity substitution* $\mathrm{id}_\delta : i \mapsto \mathsf{var}_i$. Substitutions $f : \gamma \to \delta$ and $g : \delta \to \theta$ may be *composed* to give a substitution $g \circ f : \gamma \to \theta$, defined by $(g \circ f)(k) := f^*(g(k))$.

We often write r instead of $\bar{r}$, a slight notational abuse grounded in the next proposition.

Proposition 3.19. *For all suitable renamings r, substitutions f, g, and expressions e:*

1. *Substitution generalises renaming:* $\bar{r}^* e = r_* e$.

2. *Identity substitution is trivial:* $\mathrm{id}^*_\delta e = e$.

3. *Substitution commutes with renaming:*

$$
r_*(f^* e) = (i \mapsto r_* f(i))^* e \qquad \text{and} \qquad f^*(r_* e) = (i \mapsto f(r(i)))^* e.
$$

4. *Substitution respects composition:* $f^*(g^* e) = (g \circ f)^*$.

5. *Composition of substitutions is unital and associative:*

$$
f = \mathrm{id} \circ f = f \circ \mathrm{id} \qquad \text{and} \qquad f \circ (g \circ h) = (f \circ g) \circ h.
$$

Proof. All direct by structural induction on expressions, as in the standard proofs for de Bruijn syntax. $\qquad\square$

The interaction of substitutions with signature maps is similarly straightforward: signature maps act functorially on raw substitutions, and the constructions of this section are natural with respect to the action. More precisely:

Proposition 3.20. *Given a signature map $F : \Sigma \to \Sigma'$, and a raw substitution $f : \delta \to \gamma$ over Σ, there is a raw substitution $F_* f : \delta \to \gamma$ over Σ' given by $(F_* f)(i) = F_*(f(i))$, and the action respects composition and identities in F. Moreover, given such F, for all suitable f and e, we have $F_*(f^* e) = (F_* f)^*(F_* e)$; similarly, for all suitable f, g, we have $F_*(f \circ g) = F_* f \circ F_* g$.* $\qquad\square$

3.5 Metavariable extensions and instantiations

As mentioned in the introduction, when we write down the *rules* of type theories, we will need to extend the ambient signature Σ by new symbols to represent the metavariables of the rule.

For instance, consider the constructor Π, with arity $[(\mathsf{ty}, 0), (\mathsf{ty}, 1)]$. In the rule for Π formation (Example 3.12), we shall use two new symbols A, B, corresponding to the arguments of Π in the premises of the rule. The classes and arities of these new symbols are given by their classes and binders as arguments of Π: they are both type symbols; the first one takes no arguments, and the second one takes one term argument.

Definition 3.21. The **simple arity** $\operatorname{simp} \gamma$ of a scope γ is the arity indexed by the positions $|\gamma|$, and whose arguments all have syntactic class tm with no binding, i.e., $\operatorname{simp} \gamma := \langle (\mathsf{tm}, 0) \mid i \in |\gamma| \rangle$.

Definition 3.22. The **metavariable extension** $\Sigma + \alpha$ **of** Σ **by arity** α is the signature indexed by $\operatorname{ind} \Sigma + \arg \alpha$, defined by

$$(\Sigma + \alpha)_{\iota_0(S)} := \Sigma_S \qquad \text{and} \qquad (\Sigma + \alpha)_{\iota_1(i)} := (\operatorname{cl}_\alpha i, \operatorname{simp}(\operatorname{bind}_\alpha i)).$$

We usually treat the injection of symbols $\Sigma \to \Sigma + \alpha$ as an inclusion, writing S instead of $\iota_0(S)$; and for each $i \in \arg \alpha$, we write meta_i for the corresponding new symbol $\iota_1(i)$ of $\Sigma + \alpha$, the **metavariable symbol** for i.

Example 3.23. The symbol λ has arity $[(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 1)]$. It thus gives rise to the metavariable extension $\Sigma + [(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 1)]$, which adjoins to Σ three metavariable symbols meta_0, meta_1, meta_2, which for readability we may rename to A, B, t, with classes and arities $(\mathsf{ty}, [])$, $(\mathsf{ty}, [(\mathsf{tm}, 0)])$, and $(\mathsf{tm}, [(\mathsf{tm}, 0)])$ respectively. That is, A and B are type symbols and t a term symbol, with the latter two each taking a term argument. These will then appear in the rule for λ, to denote the arguments of a generic instance of λ. We will often use more readable names for metavariable symbols, as here, without further comment.

Let γ be a scope and $\mathsf{meta}_i(e)$ a raw expression over signature $\Sigma + \alpha$ and γ. Then e is a map which assigns to each $j \in |\operatorname{bind}_\alpha i|$ an expression in $\operatorname{Expr}_{\Sigma+\alpha}^{\mathsf{tm}}(\gamma)$ because $\gamma \oplus 0 = \gamma$. Thus we may construe e as a raw substitution $e : \operatorname{bind}_\alpha i \to \gamma$. With this in mind, the following definition explains how metavariables are replaced with expressions.

Definition 3.24. Given a signature Σ, an arity α, and a scope γ, an **instantiation** $I \in \operatorname{Inst}_{\Sigma,\gamma}(\alpha)$ of α in scope γ is a family of expressions $I_i \in \operatorname{Expr}_\Sigma^{\operatorname{cl}_i \alpha}(\gamma \oplus \operatorname{bind}_i \alpha)$, for each $i \in \arg \alpha$. Such an instantiation acts on expressions $e \in \operatorname{Expr}_{\Sigma+\alpha}^{\mathsf{c}}(\delta)$ to give expressions $I_* e \in \operatorname{Expr}_\Sigma^{\mathsf{c}}(\gamma \oplus \delta)$, by replacing each occurrence of meta_i in e with a copy of I_i, with the arguments of meta_i recursively substituted for the corresponding variables in I_i:

$$I_*(\mathsf{meta}_i(e)) := (\gamma \oplus e)^* I_i,$$
$$I_*(\mathsf{var}_j) := \mathsf{var}_{\iota_1(j)},$$
$$I_*(S(e)) := S(\langle I_* e_j \rangle_{j \in \arg S}).$$

We call $I_* e$ the **instantiation of** e **with** I.

Example 3.25. Anticipating Example 4.19, the rule for function application will be written as follows, where for readability x stands for var_0:

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x\!:\!\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \qquad \vdash \mathsf{s} : \Pi(\mathsf{A}, \mathsf{B}(x)) \qquad \vdash \mathsf{t} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{s}, \mathsf{t}) : \mathsf{B}(\mathsf{t})}$$

All expressions are in the metavariable extension $\Sigma + \alpha_{\mathsf{app}}$, where Σ is some ambient signature including Π and app, and $\alpha_{\mathsf{app}} = [(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 0), (\mathsf{tm}, 0)]$ as in Example 3.9. The symbols A, B, s, t are the metavariable symbols of this extension. An instantiation of α_{app} in scope γ consists of expressions $A \in \mathrm{Expr}_\Sigma^{\mathsf{ty}}(\gamma)$, $B \in \mathrm{Expr}_\Sigma^{\mathsf{ty}}(\gamma \oplus 1)$, and $s, t \in \mathrm{Expr}_\Sigma^{\mathsf{tm}}(\gamma)$. Instantiating the conclusion with these expressions, over some context Γ, gives the judgement $\Gamma \vdash \mathsf{app}(A, B, s, t) : B[t/x]$.

Building on the above, instantiations also act on other objects built out of expressions, including substitutions and other instantiations; at the same time, being themselves syntactic objects, instantiations are acted upon by substitutions and signature maps; and all of these are suitably natural and functorial, as follows.

Definition 3.26. Given an instantiation $I \in \mathrm{Inst}_{\Sigma, \gamma}(\alpha)$:

1. The ***instantiation I acts on a substitution*** $f : \delta' \to \delta$ over $\Sigma + \alpha$ to give a substitution $I_* f : \gamma \oplus \delta' \to \gamma \oplus \delta$, defined by

$$\begin{aligned} (I_* f)(\mathrm{inl}\, i) &:= \mathrm{inl}\, i & (i \in \gamma), \\ (I_* f)(\mathrm{inr}\, j) &:= I_*(f j) & (j \in \delta). \end{aligned}$$

2. The ***instantiation I acts on an instantiation*** $J \in \mathrm{Inst}_{\Sigma + \alpha, \delta}(\beta)$ to give an instantiation $I_* J \in \mathrm{Inst}_{\Sigma, \gamma \oplus \delta}(\beta)$, defined by $(I_* J)_j := I_*(J_j)$.

3. A ***substitution*** $f : \delta \to \gamma$ over Σ ***acts on the instantiation*** I to give an instantiation $f^* I \in \mathrm{Inst}_{\Sigma, \delta}(\alpha)$, defined by $(f^* I)_i := (f \oplus \mathrm{bind}_\alpha i)^* I_i$.

4. The ***instantiation I is translated along a signature map*** $F : \Sigma \to \Sigma'$ to give an instantiation $F_* I \in \mathrm{Inst}_{\Sigma', \gamma}(\alpha)$, defined by $(F_* I)_i := F_*(I_i)$.

Proposition 3.27. *For all suitable instantiations I, J, signatures maps F, G, substitutions f, g, arities α, expressions e, and scopes γ, δ, θ:*

1. *Translation along signature maps is functorial:*

$$(G \circ F)_* I = G_*(F_* I) \qquad \text{and} \qquad \mathrm{id}_{\Sigma *} I = I.$$

2. *The actions of substitutions and instantiations on expressions and on each other are natural with respect to translation along signature maps:*

$$\begin{aligned} F_*(I_* e) &= (F_* I)_*((F + \alpha)_* e), \\ F_*(I_* f) &= (F_* I)_*((F + \alpha)_* f), \\ F_*(I_* J) &= (F_* I)_*((F + \alpha)_* J), \\ F_*(f^* I) &= (F_* f)^*(F_* I). \end{aligned}$$

3. *The action of substitutions on instantiations is functorial:*

$$(f \circ g)^* I = g^*(f^* I) \qquad \text{and} \qquad \mathrm{id}_\gamma{}^* I = I.$$

4. The action of instantiations is natural with respect to substitutions:

$$(f^*I)_*e = (f \oplus \delta)^*(I_*e),$$
$$I_*(g^*e) = (I_*g)^*(I_*e).$$

5. The action of instantiations is "associative" in the sense that

$$(I_*J)_*e = I_*(J_*((\iota_0 + \beta)_*e))$$

holds modulo the canonical associativity isomorphism between the scopes $(\gamma \oplus \delta) \oplus \theta$ and $\gamma \oplus (\delta \oplus \theta)$ of the left- and right-hand sides. $\square$

The above properties, while routine to prove, are a lot of boilerplate. They can be incorporated, if desired, into the statement that syntax forms a *scope-graded monad* on signatures in the sense of (Orchard et al., 2020), and that instantiations are certain Kleisli maps for this monad. As ever, however, we emphasise in this paper the elementary viewpoint, rather than categorical abstraction.

4 Raw type theories

Having described raw syntax, we proceed with the formulation of raw type theories. These hold enough information to prevent syntactic irregularities, and can be used to specify derivations and derivability, but are qualified as "raw" because they allow arbitrariness and abnormalities that are generally considered undesirable.

4.1 Raw contexts

Definition 4.1. A *raw context* Γ is a scope γ together with a family on $\mathrm{Expr}_\Sigma^{\mathrm{ty}}(\gamma)$ indexed by $|\gamma|$, i.e., a map $|\gamma| \to \mathrm{Expr}_\Sigma^{\mathrm{ty}}(\gamma)$. The positions of γ are also called the *variables* of Γ. We often write just Γ for γ, e.g., $i \in |\Gamma|$ instead of $i \in |\gamma|$, and $\mathrm{Expr}_\Sigma^{\mathrm{c}}(\Gamma)$ instead of $\mathrm{Expr}_\Sigma^{\mathrm{c}}(\gamma)$. We use a subscript for the application of a context Γ to a variable i, such that Γ_i is the type expression at index i.

This definition is somewhat non-traditional for dependent type theories, in a couple of ways. Contexts are more usually defined as lists, so their variables are ordered, and the type of each variable is assumed to depend only on earlier variables, i.e. $A_i \in \mathrm{Expr}_\Sigma^{\mathrm{ty}}(i)$. In our definition, the variables form an arbitrary scope, with no order assumed; and each type may *a priori* depend on any variables of the context.

One may describe the two approaches as *sequential* and *flat* contexts, respectively. The flat notion contains all the information needed when contexts are *used* in derivations; we view the sequentiality as extra information that may be provided later by a derivation of well-formedness of a context, cf. Section 6.1, but that is not needed at the raw level.

Example 4.2. With the de Bruijn index scope system, a raw context Γ of scope n may be written as $\langle A_i \in \mathrm{Expr}_\Sigma^{\mathrm{ty}}(n) \mid i \in |n| \rangle$. A more familiar way to display Γ is as list $[(n-1)\colon A_{n-1}, \ldots, 0\colon A_0]$, where each $A_i \in \mathrm{Expr}_\Sigma^{\mathrm{ty}}(n)$, but as raw contexts follow the flat approach, we should not think of this as imposing an order on the variables, and hence the list $[0\colon A_0, \ldots, (n-1)\colon A_{n-1}]$ denotes the same context Γ. Note, by the way, that at this stage contexts on de Bruijn indices are indistinguishable from contexts on de Bruijn levels. The difference becomes apparent once we consider context extension, and the scope coproduct inclusions come into play.

Definition 4.3. Let Γ be a raw context, δ be a scope, and $\Delta : |\delta| \to \mathrm{Expr}_\Sigma^{\mathsf{ty}}(|\Gamma| \oplus \delta)$ a family of expressions indexed by $|\delta|$. The ***context extension*** $\Gamma \,.\, \Delta$ is the raw context of scope $|\Gamma| \oplus \delta$, defined as

$$(\Gamma \,.\, \Delta)_{\mathrm{inl}\,j} := (\mathrm{inl}_* \circ \Gamma)_j \qquad \text{and} \qquad (\Gamma \,.\, \Delta)_{\mathrm{inr}\,k} := \Delta_k.$$

In other words, the extended raw context $\Gamma \,.\, \Delta$ is the map $[\mathrm{inl}_* \circ \Gamma, \Delta]$ induced by the universal property of the coproduct $||\Gamma| \oplus \delta|$.

Example 4.4. To continue Example 4.2, we can consider how context extension works for de Bruijn indices. Let $\Gamma = [2\colon A_2, 1\colon A_1, 0\colon A_0]$, where each $A_i \in \mathrm{Expr}_\Sigma^{\mathsf{ty}}(3)$, and $\Delta = [1\colon B_1, 0\colon B_0]$ with $B_j \in \mathrm{Expr}_\Sigma^{\mathsf{ty}}(3 + 2)$. The coproduct inclusions are $\mathrm{inl}\,i = i+2$ and $\mathrm{inr}\,j = j$. The context $\Gamma \,.\, \Delta$ has scope $3+2 = 5$, and is given by $[(i \mapsto i+2)_* \circ \Gamma, \Delta]$, which computes to $[4\colon \mathrm{inl}_* A_2, 3\colon \mathrm{inl}_* A_1, 2\colon \mathrm{inl}_* A_0, 1\colon B_1, 0\colon B_0]$. The effect is that the variables from Γ are renamed according to the scope of Δ, and the renaming acts on the associated type expressions accordingly, i.e., by shifting all variables by 2.

Note that with raw contexts, we weaken types when extending a context. In approaches using sequential contexts with scoped syntax, weakening is instead performed when types are taken *out* of a context: that is, the variable rule (precisely stated) concludes $\Gamma \vdash \mathsf{var}_i : \iota_* \Gamma_i$, where $\iota : i \to n$ is the inclusion of an initial segment into the full context.

In concrete examples, we will write contexts in a more traditional style, as lists of variables names with their associated types:

$$x_1 {:} A_1, \ldots, x_n {:} A_n.$$

Like other syntactic objects, raw contexts are acted upon by signature maps and instantiations. The functoriality and commutation properties for these actions follow directly from the corresponding properties for expressions.

Definition 4.5. Given a signature map $F : \Sigma \to \Sigma'$ and a raw context Γ over Σ, the ***translation of*** Γ ***by*** F is the raw context $F_* \Gamma$ over Σ', with $|F_* \Gamma| := |\Gamma|$ and $(F_* \Gamma)_i := F_* \Gamma_i$.

Proposition 4.6. *The action of signature maps on raw contexts is functorial:*

$$F_*(G_* \Gamma) = (F \circ G)_* \Gamma \qquad \text{and} \qquad \mathrm{id}_{\Sigma *} \Gamma = \Gamma,$$

for all suitable F, G, Γ. $\qquad\qquad\square$

The action of instantiations is a little more subtle. Acting pointwise on the expressions of the context is not enough, since the instantiated expressions lie in a larger scope. We need to supply extra types for the extra scope, i.e., the scope of the instantiation must itself underlie a raw context.

Definition 4.7. An ***instantiation*** I ***in context*** Γ over Σ for arity α is an instantiation $I \in \mathrm{Inst}_{\Sigma, |\Gamma|}(\alpha)$. Then for any raw context Δ over $\Sigma + \alpha$, the ***context instantiation*** $(I, \Gamma)_* \Delta$ is the context over Σ with scope $|\Gamma| \oplus |\Delta|$ and with type expressions

$$
\begin{aligned}
(I, \Gamma)_* \Delta_{\mathrm{inl}(i)} &:= \mathrm{inl}^* \Gamma_i & (i \in \gamma), \\
(I, \Gamma)_* \Delta_{\mathrm{inr}(j)} &:= I_* \Delta_j & (j \in \delta).
\end{aligned}
$$

Briefly, $(I, \Gamma)_* \Delta$ is the context extension of Γ by the instantiations of the types of Δ.

Proposition 4.8. *Given instantiations*

$$I \in \mathrm{Inst}_{\Sigma,|\Gamma|}(\alpha) \text{ in context } \Gamma,$$
$$K \in \mathrm{Inst}_{\Sigma+\alpha,|\Delta|}(\beta) \text{ in context } \Delta, \text{ and}$$
$$\Theta \in \mathrm{Inst}_{(\Sigma+\alpha)+\beta,|\Theta|}(\gamma) \text{ in context } \Theta,$$

the equation

$$(I,\Gamma)_*((K,\Delta)_*\Theta) = (I_*K, (I,\Gamma)_*\Delta)_*\Theta$$

holds modulo the canonical isomorphism between the scopes $|\Gamma| \oplus (|\Delta| \oplus |\Theta|)$ *and* $(|\Gamma| \oplus |\Delta|) \oplus |\Theta|$ *of their right- and left-hand sides.* $\qquad\square$

4.2 Judgements

Our type theories have four primitive judgement forms, following Martin-Löf (Martin-Löf, 1984):

A type	"A is a type"
$t : A$	"term t has type A"
$A \equiv B$	"A and B are equal as types"
$s \equiv t : A$	"s and t are equal as terms of type A"

These are represented with symbols ty, tm, tyeq, and tmeq, respectively. For our needs we need to describe the judgement forms quite precisely. In fact, the following elaboration may seem a bit *too* precise, but we found it quite useful in the formalisation to make explicit all the concepts involved and distinctions between them.

Each judgement form has a family of **boundary slots** and possibly a **head slot**, where each slot has an associated syntactic class, as follows:

Form	Boundary	Head
ty	$[]$	ty
tm	$[\mathrm{ty}]$	tm
tyeq	$[\mathrm{ty}, \mathrm{ty}]$	
tmeq	$[\mathrm{tm}, \mathrm{tm}, \mathrm{ty}]$	

The table encodes the familiar constituent parts of the judgement forms:

1. "A type" has no boundary slots; the head slot, indicated by A, is a type.

2. "$t : A$" has one boundary type slot indicated by A, called the **underlying type**; the head, indicated by t, is a term.

3. "$A \equiv B$" has two type slots indicated by A and B, called the **left-hand side** and **right-hand side**; there is no head.

4. "$s \equiv t : A$" has two term slots indicated by s and t, and a type slot indicated by A, called the **left-hand side**, the **right-hand side** and the **underlying type**, respectively; there is no head.

The **slots** of a judgement form are the slots of its boundary, and the head, if present.

Definition 4.9. Given a raw context Γ and a judgement form ϕ, a **hypothetical judgement** of that form over Γ is a map J taking each slot of ϕ of syntactic class c to an element of $\mathrm{Expr}^c_\Sigma(\Gamma)$. We write $\mathrm{Judg}\,\Sigma$ for the set of all hypothetical judgements

over Σ. The types of Γ are the **hypotheses** and J is the **thesis** of the judgement. When there is no ambiguity, we will (following traditional usage) speak of a **judgment** to mean either a whole hypothetical judgement $\Gamma \vdash J$, or just a thesis J.

A hypothetical judgement is an **object judgement** if it is a term or a type judgement, and an **equality judgement** if it is a type or a term equality judgement.

Example 4.10. The boundary and the head slots of the judgement form tm are [ty] and tm, respectively. Thus a hypothetical judgement of this form over a raw context Γ is a map taking the slot in the boundary to a type expression $A \in \mathrm{Expr}_\Sigma^{\mathsf{ty}}(\Gamma)$ and the head slot to a term expression $t \in \mathrm{Expr}_\Sigma^{\mathsf{tm}}(\Gamma)$. This corresponds precisely to the information conveyed by a traditional hypothetical term judgement "$\Gamma \vdash t : A$".

In view of the preceding example we shall write a hypothetical judgement over Γ given by a map J in the traditional type-theoretic way

$$\Gamma \vdash J$$

where the elements of J are displayed in the corresponding slots.

Just like raw contexts, judgements are acted on by signature maps and instantiations.

Definition 4.11. Given a signature map $F : \Sigma \to \Sigma'$ and a judgement $\Gamma \vdash J$ over Σ, the translation $F_*(\Gamma \vdash J)$ is the judgement $F_*\Gamma \vdash F_*J$ over Σ' of the same form, where the thesis F_*J is J with F_* applied pointwise to each expression.

Proposition 4.12. *This action is functorial:* $F_*(G_*(\Gamma \vdash J)) = (FG)_*(\Gamma \vdash J)$, *and* $(\mathrm{id}_\Sigma)_*(\Gamma \vdash J) = (\Gamma \vdash J)$, *for all suitable* F, G, Γ, J. $\qquad \square$

Definition 4.13. Given a signature Σ, a hypothetical judgement $\Delta \vdash J$ over a metavariable extension $\Sigma + \alpha$, a raw context Γ over Σ, and an instantiation I of α in context Γ, the **judgement instantiation** $(I, \Gamma)_*(\Delta \vdash J)$ is the judgement $\Gamma \, . \, I_*\Delta \vdash I_*J$ over Σ, where the thesis I_*J is just J with I_* applied pointwise.

Proposition 4.14. *Given instantiations*

$$I \in \mathrm{Inst}_{\Sigma, |\Gamma|}(\alpha) \text{ in context } \Gamma \text{ and}$$
$$K \in \mathrm{Inst}_{\Sigma + \alpha, |\Delta|}(\beta) \text{ in context } \Delta,$$

and a judgement $\Theta \vdash J$ over $(\Sigma + \alpha) + \beta$, the equation

$$(I, \Gamma)_*((K, \Delta)_*(\Theta \vdash J)) = (I_*K, (I, \Gamma)_*\Delta)_*(\Theta \vdash J)$$

holds modulo the canonical associativity renaming between their scopes. $\qquad \square$

4.3 Boundaries

In many places, one wants to consider data amounting to a hypothetical judgement without a head expression (if it is of object form, and so should have a head). For instance, a *goal* or *obligation* in a proof assistant is specified by such data; or when adjoining a new well-formed rule to a type theory, before picking a fresh symbol for it (if it is an object rule), the conclusion is specified by such data.

These entities crop up frequently, and seem almost as fundamental as judgements, so deserve a name.

Definition 4.15. Given a raw context Γ and a judgement form ϕ, a ***hypothetical boundary*** of form ϕ over Γ is a map B taking each *boundary* slot of ϕ of syntactic class c to an element of $\mathrm{Expr}_\Sigma^c(\Gamma)$.

We display boundaries as judgements with a hole $\square$ where the head should stand, or with $\equiv^?$ in place of $\equiv$:

$$\square \;\mathsf{type} \qquad \square : A \qquad A \equiv^? B \qquad s \equiv^? t : A$$

Since equality judgements have no heads, there is no difference in data between an equality judgement and an equality boundary, but there is one of sense: the judgement $A \equiv B$ asserts an equality holds, whereas the boundary $A \equiv^? B$ is a goal to be established or postulated. Analogously, $\square \;\mathsf{type}$ and $\square : A$ can be read as goals, the former that a type be constructed, and the latter that A be inhabited.

The terminology about judgements, as well as many constructions, carry over to boundaries. In particular, the action of signature maps and instantiations on boundaries is defined just as in Definitions 4.11 and 4.13, and enjoys analogous properties to Propositions 4.12 and 4.14.

Finally, and crucially, boundaries can be completed to judgements. The data required depends on the form: completing an object boundary requires a head expression; completing an equality boundary, just a change of view.

Definition 4.16. Let B be a boundary in scope γ over Σ.

1. If B is of object form, then given an expression e of the class of B in scope γ, write $B[e]$ for the ***completion*** of B with head e, a judgement over γ.

2. If B is of equality form, then the completion of B is just B itself, viewed as a judgement.

Proposition 4.17. *Completion of boundaries is natural with respect to signature maps:* $F_*(B[e]) = (F_*B)[F_*e]$. $\qquad\square$

4.4 Raw rules

We now come to raw rules, syntactic entities that capture the notion of "templates" that are traditionally used to display the inference rules of a type theory. The raw rules include all the information needed in order to be *used*, for defining derivations and derivability of judgements — but they do not yet include the extra properties we typically check when considering rules, and which guarantee good properties of the resulting derivability predicates. We return to these later, in Section 5.1.

Definition 4.18. A ***raw rule*** R over a signature Σ consists of an arity α_R, together with a family of judgements over the extended signature $\Sigma + \alpha_R$, the ***premises*** of R, and one more judgement over $\Sigma + \alpha_R$, the ***conclusion*** of R. An ***object rule*** is one whose conclusion is an object judgement, otherwise it is an ***equality rule***.

Example 4.19. Following on from Example 3.9, the raw rule for function application has arity

$$\alpha_{\mathsf{app}} = [(\mathsf{ty}, 0), (\mathsf{ty}, 1), (\mathsf{tm}, 0), (\mathsf{tm}, 0)].$$

Writing A, B, s, t for the metavariable symbols of the extended signature $\Sigma + \alpha_{\mathsf{app}}$, the premises of the rule are the four-element family:

$$[\; \vdash \mathsf{A} \;\mathsf{type}, \quad [0 \colon \mathsf{A}] \vdash \mathsf{B}(\mathsf{var}_0) \;\mathsf{type}, \quad \vdash \mathsf{s} : \Pi(\mathsf{A}, \mathsf{B}(\mathsf{var}_0)), \quad \vdash \mathsf{t} : \mathsf{A} \;]$$

and its conclusion is the hypothetical judgement

$$\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(\mathsf{var}_0), \mathsf{s}, \mathsf{t}) : \mathsf{B}(\mathsf{t}).$$

Of course, the traditional type-theoretic way of displaying such a rule is

$$\frac{\vdash \mathsf{A} \ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x) \ \mathsf{type} \qquad \vdash \mathsf{s} : \Pi(\mathsf{A}, \mathsf{B}(x)) \qquad \vdash \mathsf{t} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{s}, \mathsf{t}) : \mathsf{B}(\mathsf{t})}$$

It may seem surprising that we have $\mathsf{B}(x)$ and $\mathsf{B}(\mathsf{t})$ rather than, say, B and $B[\mathsf{t}/x]$, since this style is usually apologised for as an abuse of notation. Here, it is precise and formal; B is a metavariable *symbol* in $\Sigma + \alpha_{\mathsf{app}}$, so it is applied to arguments when used in the syntax. Also note that the occurrences of x in the third premise and the conclusion are implicitly bound by Π and app, as can be discerned from their arities. When we instantiate the rule below with actual expressions A, B, s and t, then $\mathsf{B}(x)$ and $\mathsf{B}(\mathsf{t})$ will be translated into B and $B[t/x]$ respectively.

Definition 4.20. The functorial **action** of a signature map $F : \Sigma \to \Sigma'$ is the map F_* which takes a rule R over Σ to the rule $F_* R$ over Σ' whose arity is the arity α_R of R, and its premises and conclusion are those of R, all translated along the action of the induced map $F + \alpha_R : \Sigma + \alpha_R \to \Sigma' + \alpha_R$.

A raw rule should not itself be thought of as a closure rule (though formally it is one), but rather as a template specifying a whole family of closure rules.

Definition 4.21. Given a rule R, a raw context Γ, and an instantiation I of its arity α_R over Γ, all over a signature Σ, the **instantiation of R under I,** Γ, is the closure rule $(I, \Gamma)_* R$ on $\mathrm{Judg}\,\Sigma$ whose premises and conclusion are the instantiations of the corresponding judgements of R under I, Γ. The **closure system** $\mathrm{cl}\,R$ **associated to** R is the family $\langle (I, \Gamma)_* R \mid \Gamma \in \mathrm{Cxt}\,\Sigma, I \in \mathrm{Inst}_{\Sigma, \Gamma}(\alpha_R) \rangle$ of all such instantiations.

Example 4.22. Continuing Example 4.19, the raw rule R_{app} for application gives the closure system $\mathrm{cl}(R_{\mathsf{app}})$, containing for each raw context Γ (with scope γ) and expressions $A, B \in \mathrm{Expr}_\Sigma^{\mathsf{ty}}(\gamma)$, $s, t \in \mathrm{Expr}_\Sigma^{\mathsf{tm}}(\gamma)$ a closure rule

$$\frac{\Gamma \vdash A \ \mathsf{type} \qquad \Gamma, x{:}A \vdash B \ \mathsf{type} \qquad \Gamma \vdash s : \Pi(A, B) \qquad \Gamma \vdash t : A}{\Gamma \vdash \mathsf{app}(A, B, s, t) : B[t/x].}$$

This is visually similar to R_{app} itself, but not to be confused with it. The instantiation is a single closure rule, written over the ambient signature Σ; the original raw rule, written over $\Sigma + \alpha_{\mathsf{app}}$, is a template specifying the whole family of such closure rules. In the raw rule, $\mathsf{A}, \mathsf{B}, \mathsf{s}, \mathsf{t}$ are metavariable symbols from the extended signature $\Sigma + \alpha_{\mathsf{app}}$; in the instantiatiaion, the symbols A, B, s, t (note the difference in fonts) are the actual syntactic expressions the raw rule was instantiated with.

This construction of cl formalises the usual informal explanation that a single written rule is a shorthand for a scheme of closure conditions, with the quantification of the scheme inferred from the written rule.

Proposition 4.23. *The construction* cl *is* laxly natural *in signature maps, in that given* $F : \Sigma \to \Sigma'$ *and a rule R over Σ, there is an induced simple map of closure systems* $\mathrm{cl}\,R \to \mathrm{cl}\,F_* R$, *over* $F_* : \mathrm{Judg}\,\Sigma \to \mathrm{Judg}\,\Sigma'$.

Proof. For each instantiation $I \in \mathrm{Inst}_{\Sigma, |\Gamma|}(\alpha_R)$, we have $F_* I \in \mathrm{Inst}_{\Sigma', |\Gamma|}(\alpha_R)$ and $(F_* I, F_* \Gamma)_* R = F_*((I, \Gamma)_* R)$. $\qquad\square$

This is lax in the sense that the resulting map $\mathrm{cl}\, R \to \mathrm{cl}\, F_* R$ will not usually be an isomorphism: in general, not every instantiation of α_R over Σ' is of the form $F_* I$. This illustrates the need for considering raw rules formally, rather than just viewing a type theory as a collection of closure rules: when translating a type theory between signatures, we want not just the translations of the original closure rules, but all instantiations of the translated raw rules.

One might hope for cl to be similarly laxly natural under instantiations. However, this is not so straightforwardly true; we will return to this in Proposition 4.31, once the structural rules are introduced, and show a weaker form of naturality.

4.5 Structural rules

The rules used in derivations over a type theory will fall into two groups:

1. the ***structural rules***, governing generalities common to all type theories;

2. the ***specific rules*** of the particular type theory.

The structural rules over a signature Σ are a family of closure rules on $\mathrm{Judg}\,\Sigma$, which we now lay out. They are divided into four families:

- the variable rules,

- rules stating that equality is an equivalence relation,

- rules for conversion of terms and term equations between equal types, and

- rules for substitutions,

We have chosen the rules so that the development of the general setup requires no hard meta-theorems, as far as possible. In particular, we include the substitution rules into the formalism so that we can postpone proving elimination of substitution until Section 5.4. You might have expected to see congruence rules among the structural rules, but those we take care of separately in Section 4.6 because they depend on the specific rules.

The first three families of structural rules are straightforward.

Definition 4.24. For each raw context Γ over a signature Σ, and for each $i \in |\Gamma|$, the corresponding *variable rule* is the closure rule

$$\frac{\Gamma \vdash \Gamma_i\ \mathsf{type}}{\Gamma \vdash \mathsf{var}_i : \Gamma_i}$$

Taken together, the variable rules form a family indexed by such pairs (Γ, i).

While this had to be given directly as a family of closure rules, the next two groups of structural rules can be expressed as raw rules.

Definition 4.25. The *raw equivalence relation rules* are the following raw rules:

$$\frac{\vdash \mathsf{A}\ \mathsf{type}}{\vdash \mathsf{A} \equiv \mathsf{A}} \qquad\qquad \frac{\vdash \mathsf{A}\ \mathsf{type} \qquad \vdash \mathsf{B}\ \mathsf{type} \qquad \vdash \mathsf{A} \equiv \mathsf{B}}{\vdash \mathsf{B} \equiv \mathsf{A}}$$

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad \vdash \mathsf{B}\ \mathsf{type} \qquad \vdash \mathsf{C}\ \mathsf{type} \qquad \vdash \mathsf{A} \equiv \mathsf{B} \qquad \vdash \mathsf{B} \equiv \mathsf{C}}{\vdash \mathsf{A} \equiv \mathsf{C}}$$

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad \vdash \mathsf{s} : \mathsf{A}}{\vdash \mathsf{s} \equiv \mathsf{s} : \mathsf{A}} \qquad \frac{\vdash \mathsf{A}\ \mathsf{type} \qquad \vdash \mathsf{s} : \mathsf{A} \qquad \vdash \mathsf{t} : \mathsf{A} \qquad \vdash \mathsf{s} \equiv \mathsf{t} : \mathsf{A}}{\vdash \mathsf{t} \equiv \mathsf{s} : \mathsf{A}}$$

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad \vdash \mathsf{s} : \mathsf{A} \qquad \vdash \mathsf{t} : \mathsf{A} \qquad \vdash \mathsf{u} : \mathsf{A} \qquad \vdash \mathsf{s} \equiv \mathsf{t} : \mathsf{A} \qquad \vdash \mathsf{t} \equiv \mathsf{u} : \mathsf{A}}{\vdash \mathsf{s} \equiv \mathsf{u} : \mathsf{A}}$$

The *equivalence relation rules over* Σ is the sum of the closure systems associated to the above equivalence relation rules, over a given signature Σ.

We trust the reader to be able read off the arities of the metavariable symbols appearing in raw rules. For instance, from the use of A and s in the above term reflexivity rule we can tell that the rule has arity $[(\mathsf{ty}, 0), (\mathsf{tm}, 0)]$.

The conversion rules are written as raw rules, as well.

Definition 4.26. The *raw conversion rules* are the following raw rules:

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad \vdash \mathsf{B}\ \mathsf{type} \qquad \vdash \mathsf{s} : \mathsf{A} \qquad \vdash \mathsf{A} \equiv \mathsf{B}}{\vdash \mathsf{s} : \mathsf{B}}$$

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad \vdash \mathsf{B}\ \mathsf{type} \qquad \vdash \mathsf{s} : \mathsf{A} \qquad \vdash \mathsf{t} : \mathsf{A} \qquad \vdash \mathsf{s} \equiv \mathsf{t} : \mathsf{A} \qquad \vdash \mathsf{A} \equiv \mathsf{B}}{\vdash \mathsf{s} \equiv \mathsf{t} : \mathsf{B}}$$

Again, the *conversion rules over* Σ is the sum of the closure systems associated to the above conversion rules, over a signature Σ.

The remaining groups are the substitution and equality-substitution rules.

The substitution rule should formalize the notion that "well-typed" substitutions preserve derivability of judgements. Treatments taking simultaneous substitution as primitive usually say something like: a raw substitution $f : \Delta \to \Gamma$ is well-typed if $\Delta \vdash f(i) : f^*\Gamma_i$ for each $i \in |\Gamma|$. However, taking all these judgements as premises in the substitution rule is rather profligate: most substitutions in practice act non-trivially only on a small part of the context. For instance, a single-variable substitution may be represented as a raw substitution $\Gamma \to \Gamma.A$ acting trivially on Γ, so no checking should be required there. Indeed, in treatments taking single-variable substitution as primitive, only require checking of the substituted expression. To abstract this situation, we define the substitution rule as follows. Recall that a subset $X \subseteq Y$ is *complemented* when $X \cup (Y \setminus X) = Y$, a condition that is vacuously true in classical logic.

Definition 4.27. A *raw substitution* $f : \Delta \to \Gamma$ *acts trivially at* $i \in |\Gamma|$ when there is some (necessarily unique) $j \in |\Delta|$ such that $f(i) = \mathsf{var}_j$ and $\Delta_j = f^*\Gamma_i$. Given a complemented subset $K \subseteq |\Gamma|$ on which f acts trivially, the corresponding *substitution rule* is the closure rule

$$\frac{\Gamma \vdash J \qquad \text{for each } i \in |\Gamma| \setminus K : \quad \Delta \vdash f(i) : f^*\Gamma_i}{\Delta \vdash f^*J} \tag{4.1}$$

The substitution rules form a family of closure rules, indexed by $f : \Delta \to \Gamma$, K, and $\Gamma \vdash J$.

In the above definition, K is thought of as a set of positions at which f is *guaranteed* to act trivially, but it may also do so outside K, as there is no harm in checking positions at which f acts trivially.

The substitution rules are formulated carefully for another, more technical reason. In inductions over derivations (e.g. for Lemma 5.20), when a substitution descends under a binder, it gets extended to act trivially on the variables introduced by the binder. Within the inductive cases, we may not yet have enough information to conclude that the types of the bound variables are well-formed, but we can rely on the trivial action of the substitution. Keeping substitution rules flexible and economical in this way therefore keeps these inductive proofs much cleaner.

Along similar lines, we have rules stating that substitution of equal terms gives equal results.

Definition 4.28. Raw substitutions $f, g : \Delta \to \Gamma$ **act jointly trivially** at $i \in |\Gamma|$ when there is some (necessarily unique) $j \in |\Delta|$ such that $f(i) = g(i) = \mathsf{var}_j$ and $\Delta_j = f^*\Gamma_i = g^*\Gamma_i$. Given a complemented subset $K \subseteq |\Gamma|$ on which f and g act jointly trivially, the corresponding **equality-substitution rules** are the closure rules

$$
\frac{\Gamma \vdash A \text{ type} \\ \text{for each } i \in |\Gamma| \setminus K: \\ \Delta \vdash f(i) : f^*\Gamma_i \qquad \Delta \vdash g(i) : g^*\Gamma_i \qquad \Delta \vdash f(i) \equiv g(i) : f^*\Gamma_i}{\Delta \vdash f^*A \equiv g^*A}
$$

$$
\frac{\Gamma \vdash t : A \\ \text{for each } i \in |\Gamma| \setminus K: \\ \Delta \vdash f(i) : f^*\Gamma_i \qquad \Delta \vdash g(i) : g^*\Gamma_i \qquad \Delta \vdash f(i) \equiv g(i) : f^*\Gamma_i}{\Delta \vdash f^*t \equiv g^*t : f^*A}
$$

The equality-substitution rules form a family of closure rules, indexed by $f : \Delta \to \Gamma$, K, and either $\Gamma \vdash A$ type or $\Gamma \vdash t : A$.

Definition 4.29. The **structural rules over** Σ, denoted $\mathrm{Struct}\,\Sigma$, is the sum of the families of closure rules set out above: the variable, equivalence relation, conversion, substitution, and equality-substitution rules.

Proposition 4.30. *Given a signature map $F : \Sigma \to \Sigma'$, there is a simple map of closure systems $\mathrm{Struct}\,\Sigma \to \mathrm{Struct}\,\Sigma'$ over $F_* : \mathrm{Judg}\,\Sigma \to \mathrm{Judg}\,\Sigma'$.*

Proof. This is straightforward, amounting to checking that for each instance of a structural rule over Σ, F acts on the data to give an instance of the same structural rule over Σ', and the resulting closure condition is the translation along F_* of the original closure condition over Σ. $\square$

Before giving a similar statement about instantiations of structural rules, we must first tie up the loose end from above about instantiation of closure systems of raw rules.

Proposition 4.31. *Let R be a raw rule over Σ, $R + \beta$ its translation to an extension $\Sigma + \beta$, and I an instantiation of β in some context Γ. Then there is a closure system map $\mathrm{cl}(R + \beta) \to \mathrm{cl}\,R + \mathrm{Struct}\,\Sigma$, over $(I, \Gamma)_* : \mathrm{Judg}\,(\Sigma + \beta) \to \mathrm{Judg}\,\Sigma$.*

Proof. We need to show that for each instantiation $K \in \operatorname{Inst}_{\Sigma+\beta,|\Delta|}(\alpha_R)$ in some context Δ, the closure condition $I_*((K,\Gamma)_* R)$ is derivable from $\operatorname{cl} R + \operatorname{Struct} \Sigma$.

Given such K and Γ, we can instantiate both under I to get an instantiation of R over Σ. We might hope that $(I_* K, I_* \Gamma)_* R = I_*((K,\Gamma)_*(R+\beta))$; by Proposition 4.14, we see that this does not strictly, but only up to an associativity renaming in each judgement.

The substitution structural rule comes to our rescue here. For each judgement J involved in R, with context Θ, the associativity renamings give substitutions between $I_*(K_* \Theta)$ and $(I_* K)_* \Theta$ acting trivially at every position, so the substitution rule lets us derive $I_*(K_* J)$ from $(I_* K)_* J$ (with no further premises), and vice versa.

The desired derivation of $I_*((K,\Gamma)_*(R+\beta))$ from $\operatorname{cl} R + \operatorname{Struct} \Sigma$ therefore consists of $(I_* K, I_* \Gamma)_* R$, together with an instance of the substitution rule after the conclusion and before each premise, implementing the associativity renamings. $\qquad\square$

Proposition 4.32. *Let* $I \in \operatorname{Inst}_{\Sigma,|\Gamma|}(\alpha)$ *be an instantiation in context* Γ. *Then there is a closure system map* $\operatorname{Struct}(\Sigma + \alpha) \to \operatorname{Struct} \Sigma$, *over* $(I,\Gamma)_* : \operatorname{Judg}(\Sigma + \alpha) \to \operatorname{Judg} \Sigma$.

Proof. For the structural rules given as raw rules, the required derivations are given by Proposition 4.31.

For the other structural rules, we start as in Proposition 4.30. Given an instance of a structural rule over $\Sigma + \alpha$, we instantiate its data under I to get an instance of the same structural rule over Σ. Wrapping this instance in associativity renamings, derived by the substitution rule as in Proposition 4.31, gives the required derivation of the instantiation of the original instance. $\qquad\square$

4.6 Congruence rules

Congruence rules, which state that judgemental equality commutes with type and term symbols, are peculiar enough to demand special attention.

They are present in almost all type theories, but rarely explicitly written out, and are often classified as structural rules. We reserve that term for the rules of the preceding section, which are independent of the specific theory under consideration. Congruence rules, by contrast, depend on the specific rules of a theory; for instance, the congruence rule for Π is determined by the formation rule for Π.

In this section we define how any object rule determines an associated congruence rule. We first set up an auxiliary definition, associating equality judgements to object judgements.

Definition 4.33. For signature maps $\ell, r : \Sigma \to \Sigma'$ and an object judgement J over Σ, we define the equality judgment $(\ell, r)^* J$ over Σ' by

$$
\begin{aligned}
(\ell, r)^*(\Gamma \vdash A \ \mathsf{type}) &:= (\ell_* \Gamma \vdash \ell_* A \equiv r_* A), \\
(\ell, r)^*(\Gamma \vdash t : A) &:= (\ell_* \Gamma \vdash \ell_* t \equiv r_* t : \ell_* A).
\end{aligned}
$$

Definition 4.34. Suppose R is a raw object rule over a signature Σ, with premises $\langle P_i \mid i \in I \rangle$ and conclusion C. Let ϕ_i be the judgement form of P_i, and take $I_{\mathrm{ob}} := \{i \in I \mid \phi_i \in \{\mathsf{ty}, \mathsf{tm}\}\}$, the set of object premises of R. The **associated congruence rule** $R_\equiv$ is a raw rule with arity $\alpha_{R_\equiv} := \alpha_R + \alpha_R$, defined as follows, where $\ell, r :$

$\Sigma + \alpha_R \to \Sigma + \alpha_{R_\equiv}$ are signature maps

$$\ell(\iota_0(S)) := \iota_0(S), \qquad\qquad r(\iota_0(S)) := \iota_0(S),$$
$$\ell(\iota_1(M)) := \iota_1(\iota_0(M)), \qquad\qquad r(\iota_1(M)) := \iota_1(\iota_1(M)):$$

1. The premises of $R_\equiv$ are indexed by the set $I + I + I_{\mathrm{ob}}$, and are given by:

   (a) the $\iota_0(i)$-th premise is $\ell_* P_i$,

   (b) the $\iota_1(j)$-th premise is $r_* P_j$,

   (c) the $\iota_2(k)$-th premise is the equality $(\ell, r)^* P_k$, cf. Definition 4.33.

2. The conclusion of $R_\equiv$ is $(\ell, r)^* C$.

Example 4.35. Definition 4.34 works as expected. For example, the congruence rule associated with the usual product formation rule

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}}{\vdash \Pi(\mathsf{A}, \mathsf{B}(x))\ \mathsf{type}}$$

comes out to be

$$\frac{\begin{array}{cc} \vdash \mathsf{A}'\ \mathsf{type} & x{:}\mathsf{A}' \vdash \mathsf{B}'(x)\ \mathsf{type} \\ \vdash \mathsf{A}''\ \mathsf{type} & x{:}\mathsf{A}'' \vdash \mathsf{B}''(x)\ \mathsf{type} \\ \vdash \mathsf{A}' \equiv \mathsf{A}'' & x{:}\mathsf{A}' \vdash \mathsf{B}'(x) \equiv \mathsf{B}''(x) \end{array}}{\vdash \Pi(\mathsf{A}', \mathsf{B}''(x)) \equiv \Pi(\mathsf{A}'', \mathsf{B}''(x))}$$

4.7 Raw type theories

After a considerable amount of preparation, we are finally in position to formulate what a rudimentary general type theory is.

Definition 4.36. A *raw type theory* T over a signature Σ is a family of raw rules over Σ.

Definition 4.37. The *associated closure system* of a raw type theory T over Σ is the closure system $\mathrm{cl}\, T := \mathrm{Struct}\,\Sigma + \coprod_{R \in T} \mathrm{cl}\, R$ on $\mathrm{Judg}\,\Sigma$; that is, it consists of the structural rules for Σ, and the closure rules generated by the instantiations of the rules of T. A *derivation in* T is a derivation over the closure system $\mathrm{cl}\, T$, in the sense of Definition 2.8.

Note that we have not included the congruence rules into the closure system associated with a raw type theory. Instead, the presence of congruence rules will be required separately as a well-behavedness condition in Section 5.2. Derivability and admissibility of rules may now be defined as follows.

Definition 4.38. Let T be a raw type theory over Σ, and R a raw rule over Σ.

1. R is *derivable* from T if its conclusion is derivable from its premises, over $T + \alpha_R$.

2. R is *admissible* for T if for every instance $(I, \Gamma)_* R$, its conclusion is derivable if its premises are derivable, all over T.

We record the basic category-theoretic structure of raw type theories.

Definition 4.39. Given a signature map $F : \Sigma \to \Sigma'$, and raw type theories T, T' over Σ and Σ' respectively, a **simple map** $\bar{F} : T \to T'$ **over** F is a family map $T \to T'$ over $F_* : \mathrm{RawRule}\,\Sigma \to \mathrm{RawRule}\,\Sigma'$. Such $\bar{F}$ is thus a map giving for each rule R of T a rule $\bar{F}(R)$ of T', whose premises and conclusion are the translations along F of those of R. There are evident identity simple maps, and composites over composites of signature maps, forming a category over the category of signatures.

Furthermore, a signature map $F : \Sigma \to \Sigma'$ **acts on a raw type theory** T over Σ, to give a raw type theory $F_*(T)$ over Σ', which consists of the translations F_*R of the rules R of T. As with family maps, maps $T \to T'$ over F correspond precisely to maps $F_*T \to T'$ over $\mathrm{id}_{\Sigma'}$. In the case of the inclusion to a metavariable extension $\iota_0 : \Sigma \to \Sigma + \alpha$, we write $T + \alpha$ for the translation $\iota_{0*}T$ of T to $\Sigma + \alpha$.

Proposition 4.40. *The construction* cl *is functorial in simple maps: a simple map of raw type theories* $\bar{F} : T \to T'$ *over* $F : \Sigma \to \Sigma'$ *induces a map* $\bar{F}_* : \mathrm{cl}\,T \to \mathrm{cl}\,T'$ *over* $F_* : \mathrm{Judg}\,\Sigma \to \mathrm{Judg}\,\Sigma'$, *and hence provides a translation of any derivation* $D \in \mathrm{Der}_T(H, (\Gamma \vdash J))$ *to a derivation* $\bar{F}_*D \in \mathrm{Der}_{T'}(F_*H, (F_*\Gamma \vdash F_*J))$.

Proof. Direct from the functoriality and naturality properties of structural rules (Proposition 4.30) and of the closure systems associated to raw rules (Proposition 4.23). $\square$

Corollary 4.41. *A signature map* $F : \Sigma \to \Sigma'$ *acts on* $D \in \mathrm{Der}_T(H, (\Gamma \vdash J))$ *to give a derivation* $F_*D \in \mathrm{Der}_{F_*T}(F_*H, F_*(\Gamma \vdash J))$, *functorially so.*

Proof. By Proposition 4.40, using the canonical simple map $T \to F_*T$ over F. $\square$

We use the previously corollary quite frequently to translate a derivation over a raw type theory to its extension. We mostly leave such applications implicit, as they are easily detected.

Instantiations also preserve derivability, but this is a significantly more involved construction — more so than one might expect — bringing together many earlier constructions and lemmas, and relying in particular on almost all the properties of Proposition 3.27.

Proposition 4.42. *Given a raw type theory* T *over* Σ, *an instantiation* $I \in \mathrm{Inst}_{\Sigma,\Gamma}(\alpha)$ *induces a closure system map* $(I, \Gamma)_* : \mathrm{cl}\,T + \alpha \to \mathrm{cl}\,T$ *over* $(I, \Gamma)_* : \mathrm{Judg}\,\Sigma + \alpha \to \mathrm{Judg}\,\Sigma$, *where* $T + \alpha$ *is the translation of* T *by the inclusion* $\Sigma \to \Sigma + \alpha$.

Proof. Again, direct from similar properties of structural rules (Proposition 4.32) and closure systems of raw rules (Proposition 4.31). $\square$

Corollary 4.43. *Let* T *be a raw type theory over* Σ. *An instantiation* $I \in \mathrm{Inst}_{\Sigma,\Gamma}(\alpha)$ *acts on a derivation* $D \in \mathrm{Der}_{T+\alpha}(H, (\Delta \vdash J))$ *to give the* **instantiation** $(I, \Gamma)_*D \in \mathrm{Der}_T((I, \Gamma)_*H, I_*(\Delta \vdash J))$.

Note that the hypotheses H and the judgement $\Delta \vdash J$ in the statement reside in the translation $T + \alpha$ by the inclusion $\Sigma \to \Sigma + \alpha$.

4.8 Summary

Raw type theories give a conceptually minimal way to make precise what is meant by traditional specifications of type theories, and a similarly minimal amount of data from which to define *derivability* on judgements.

As the name suggests, raw type theories are not a finished product. Type theories in nature almost always satisfy further well-formedness properties, and are rejected by audiences if they do not. In the next two sections, we will discuss these well-formedness properties.

In some ways, raw type theories may therefore be viewed as an unnatural or undesirable notion. However, most of the well-behavedness properties — or rather, the conditions on rules implying well-behavedness — themselves involve checking derivability of certain judgements. So raw type theories, as the minimal data for defining derivability, give a natural intermediate stage on the way to our main definition of "reasonable" type theories.

5 Well-behavedness properties

In this section we identify easily-checked syntactic properties of the rules specifying a type theory, and prove basic fitness-for-purpose meta-theorems, which together articulate the rules-of-thumb that researchers habitually use to verify that some collection of inference rules defines a "reasonable" type theory.

5.1 Acceptable rules

Not all raw rules are deemed reasonable from a type-theoretic point of view. But what standard of "reasonable" are we aiming to delineate? Essentially, the same as for the axioms of a theory in first-order logic: the axioms must be well-formed enough to be given some meaning, although that meaning may be "false", "wrong", or otherwise unexpected.

Consider for instance the following possible modifications of the rule for app, all written as raw rules:

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \qquad \vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(x)) \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})} \qquad (5.1)$$

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \qquad \vdash \mathsf{f} : \mathsf{A} \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})} \qquad (5.2)$$

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \qquad \vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(x)) \qquad \vdash \mathsf{a} : \Pi(\mathsf{A}, \mathsf{B}(x))}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})} \qquad (5.3)$$

The first is the usual rule for app, and should certainly be considered acceptable.

The second asks the argument f to be of type A. This is "wrong" under the usual reading of Π and app, but not entirely meaningless: one can introduce app with this typing rule, and obtain a well-behaved (if bizarre) type theory. So this should be accepted as a type-theoretic rule.

The third asks the argument a to be of type $\Pi(\mathsf{A}, \mathsf{B}(x))$. This is "not even wrong": the conclusion purports to introduce a term of type $\mathsf{B}(\mathsf{a})$, but that is not a well-formed type, since B expects an argument of type A, so a is not suitable (at least in the absence of other rules implying that $\vdash \mathsf{A} \equiv \Pi(\mathsf{A}, \mathsf{B}(x))$). This will therefore *not* be an acceptable rule.

Another unacceptable rule would be:

$$\frac{x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type} \qquad \vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(x)) \qquad \vdash \mathsf{a} : \mathsf{A} \qquad \vdash \mathsf{a} : \Pi(\mathsf{A}, \mathsf{B}(x))}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})} \qquad (5.4)$$

(with $\vdash \mathsf{A}\ \mathsf{type}$ above)

This is again clearly nonsense: it introduces a twice, with two different types.

There are rules which are not uncommon in practice, but which we will not accept directly, such as:

$$\frac{\vdash \mathsf{f} : \Pi(\mathsf{A}, \mathsf{B}(x)) \qquad \vdash \mathsf{a} : \mathsf{A}}{\vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), \mathsf{f}, \mathsf{a}) : \mathsf{B}(\mathsf{a})} \qquad (5.5)$$

While the rule is completely reasonable, making sense of it is rather subtle: checking, for instance, that $\mathsf{B}(\mathsf{a})$ in the conclusion is well-formed requires applying some kind of inversion principle, to the type $\Pi(\mathsf{A}, \mathsf{B}(x))$ from the premises. Whether such an inversion principle is available depends on the particularities of the type theory under consideration. In general, we want acceptable rules to be more straightforwardly and robustly well-behaved, so we expect that every metavariable used by the rule is explicitly introduced by some (unique) premise.

Finally, some rules have variant forms given by moving simple premises into the context of the conclusion. For example, the rule for application is sometimes given as

$$\frac{\vdash \mathsf{A}\ \mathsf{type} \qquad x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}}{x{:}\mathsf{A}, y{:}\Pi(\mathsf{A}, \mathsf{B}(x)) \vdash \mathsf{app}(\mathsf{A}, \mathsf{B}(x), y, x) : \mathsf{B}(x)} \qquad (5.6)$$

This variant has been called the *hypothetical* form, in contrast to the *universal* form (5.1). With substitution included as a structural rule, the two forms are equivalent: each is derivable from the other. In the absence of a substitution rule, they are not equivalent; the hypothetical form is too weak. We have also heard it argued that the universal form should be seen as conceptually prior. So both forms are arguably reasonable; but the universal form (5.1), with empty conclusion context, has the clearer claim, and no generality is lost by restricting to such forms.

Summarising the above discussion, there are several simple syntactic criteria commonly used as rules-of-thumb to determine "reasonability" of rules. We now formally define these criteria, and collect them into a definition of *acceptability* of rules.

Definition 5.1. Suppose R is a raw rule with arity α_R over a signature Σ. We say that R is *tight* when there exists a bijection β between the arguments of α_R and the object premises of R, such that for each argument i of α_R,

1. the context of the premise $\beta(i)$ has the scope $\mathrm{bind}_{\alpha_R} i$;

2. the judgement form of the premise $\beta(i)$ is $\mathrm{cl}_{\alpha_R} i$;

3. the head expression of the premise $\beta(i)$ is $\mathsf{meta}_i(\langle \mathsf{var}_j \rangle_{j \in \mathrm{bind}_{\alpha_R} i})$.

Note that the bijection β is unique, if it exists. The definition of tightness is admittedly a bit technical, but it captures a well-formedness condition of rules which is familiar but infrequently discussed explicitly. Namely, a rule is tight if its object premises provide the "typing" of its metavariable symbols.

Tightness alone does not suffice to make a rule reasonable, e.g., the rule (5.3) is tight but still broken because the type expression $\mathsf{B}(\mathsf{a})$ is senseless. We need another condition which ensures that the type and term expressions appearing in the rule make sense.

Definition 5.2. To each judgement $\Gamma \vdash J$, we associate the family of ***presuppositions*** $\mathrm{Presup}\,(\Gamma \vdash J)$, defined as the judgements formed over Γ by placing the boundary slots of J in the head position as follows:

$$\mathrm{Presup}\,(\Gamma \vdash A \text{ type}) := [\,],$$
$$\mathrm{Presup}\,(\Gamma \vdash s : A) := [\Gamma \vdash A \text{ type}],$$
$$\mathrm{Presup}\,(\Gamma \vdash A \equiv B) := [\Gamma \vdash A \text{ type}, \Gamma \vdash B \text{ type}],$$
$$\mathrm{Presup}\,(\Gamma \vdash s \equiv t : A) := [\Gamma \vdash A \text{ type}, \Gamma \vdash s : A, \Gamma \vdash t : A].$$

We shall need to know later on that presuppositions are natural with respect to the action of signature maps, instantiations, and raw substitutions.

Proposition 5.3. *Let $\Gamma \vdash J$ be a judgement over Σ, and $F : \Sigma \to \Sigma'$ a signature map. Then* $\mathrm{Presup}\,(F_*\Gamma \vdash F_*J) = F_*(\mathrm{Presup}\,(\Gamma \vdash J))$.

Proof. This is clear, for instance the presupposition of $F_*\Gamma \vdash F_*s : F_*A$ is $F_*\Gamma \vdash F_*A$ type, which is precisely what we get when F acts on $\Gamma \vdash A$ type, the presupposition of $\Gamma \vdash s : A$. $\qquad\square$

The reasoning that established the analogous statements about the actions of instantiations and raw substitutions is similarly easy.

Proposition 5.4. *Let $\Gamma \vdash J$ be a judgement over a metavariable extension $\Sigma + \alpha$ and $I \in \mathrm{Inst}_{\Sigma,\gamma}(\alpha)$ an instantiation. Then* $\mathrm{Presup}\,(I_*\Gamma \vdash I_*J) = I_*(\mathrm{Presup}\,(\Gamma \vdash J))$.
$$\square$$

Proposition 5.5. *Let $\Gamma \vdash J$ be a judgement and $f : \Delta \to \Gamma$ a raw substitution. Every presupposition of $\Delta \vdash f^*J$ has the form $\Delta \vdash f^*J'$, where $\Gamma \vdash J'$ is a presupposition of $\Gamma \vdash J'$.* $\qquad\square$

There is a weaker and a stronger condition that we can impose on a rule with regards to the presuppositions of its conclusion.

Definition 5.6. Let T be a raw type theory over a signature Σ and R a raw rule over Σ:

1. a raw rule R is ***weakly presuppositive over*** T when every presupposition of the conclusion of R is derivable in T (translated from Σ to $\Sigma + \alpha_R$) from the premises of R and the presuppositions of the premises of R,

2. a raw rule R is ***presuppositive over*** T when all presuppositions of the conclusion and of the premises of R are derivable in T (translated from Σ to $\Sigma + \alpha_R$) from the premises of R.

As far as derivability is concerned, weakly presuppositive rules are good enough, for a rule cannot be applied unless its premises have already been derived, in which case their presuppositions will be derivable as well — which is the gist of the proof of Theorem 5.15. However, if we were to give a meaning to a raw rule on its own, we would be hard-pressed to explain what the premises are about, unless their presuppositions were derivable as well, hence we take the stronger variant as the standard one.

Definition 5.7. A raw rule R is ***acceptable*** for a raw type theory T if it is tight, presuppositive over T, and has empty conclusion context.

Example 5.8.

1. The above rules (5.1), (5.2), (5.3), and (5.6) are tight.

2. The above rules (5.1), (5.2), (5.4), and (5.6) are presuppositive.

3. The rule

$$\frac{}{\vdash A\ \mathsf{type}}$$

   which allows us to infer that every type expression is a type, is not tight.

4. If S is a type symbol with the empty arity, the rule

$$\frac{}{\vdash S()\ \mathsf{type}}$$

   is presuppositive and tight.

5. Symmetry of type equality comes in two versions:

$$\frac{\vdash A \equiv B}{\vdash B \equiv A} \qquad \frac{\vdash A\ \mathsf{type} \qquad \vdash B\ \mathsf{type} \qquad \vdash A \equiv B}{\vdash B \equiv A}$$

   The left-hand one is not tight and is presuppositive, and the right-hand one is tight and presuppositive.

Proposition 5.9. *The congruence rule associated to an acceptable object rule is acceptable.*

Proof. Let R be a tight and presuppositive raw object rule over a raw type theory T with premises $\langle \Gamma_i \vdash J_i \mid i \in I \rangle$. There exists a bijection β_R between object premises of R and the arguments of α_R.

Definition 4.34 lays out the associated congruence rule $R_\equiv$. Its arity is $\alpha_{R_\equiv} = \alpha_R + \alpha_R$ and its premises are indexed by $I + I + I_{\mathrm{ob}}$, where I_{ob} is the set of object premises of R. The bijection $\beta_{R_\equiv}$ witnessing tightness of $R_\equiv$ is given by

$$\beta_{R_\equiv}(\iota_0(i)) := \iota_0(\beta_R(i)) \qquad \text{and} \qquad \beta_{R_\equiv}(\iota_1(j)) := \iota_1(\beta_R(j)).$$

Let us verify that the properties for tightness of $R_\equiv$ required in Definition 5.1 follow directly from the tightness of R. For any $\iota_0(i) \in \arg \alpha_{R_\equiv}$:

(1) The context of the premise $\beta_{R_\equiv}(\iota_0(i)) = \iota_0(\beta_R(i))$ is $\ell_* \Gamma_i$. The signature map $\ell : \Sigma + \alpha_R \to \Sigma + \alpha_{R_\equiv}$ does not change the underlying scope of Γ_i, and thus $\ell_* \Gamma_i$ has the same underlying scope as Γ_i, which equals $\mathrm{bind}_{\alpha_R} i$ because R is tight. Furthermore, $\mathrm{bind}_{\alpha_R} i = \mathrm{bind}_{\alpha_{R_\equiv}}(\iota_0(i))$, as required.

(2) The premise $\beta_{R_\equiv}(\iota_0(i))$ has judgement form $\mathrm{cl}_{\alpha_{R_\equiv}} \iota_0(i)$ by the analogous reasoning.

(3) The head of the premise $\beta_{R_\equiv}(\iota_i(i))$ is $\ell_* e$ where $e = \mathsf{meta}_i(\langle \mathsf{var}_j \rangle_{j \in \mathrm{bind}_{\alpha_R} i})$ by tightness of R. We need to show that $\ell_* e = \mathsf{meta}_{\iota_0(i)}(\langle \mathsf{var}_j \rangle_{j \in \mathrm{bind}_{\alpha_{R_\equiv}} \iota_0(i)})$, but this equation holds by the definitions of ℓ and of $\alpha_{R_\equiv}$.

The case of $\iota_1(j) \in \arg \alpha_{R_\equiv}$ is symmetric.

We also need to show that all presuppositions of the premises and the conclusion of $R_\equiv$ are derivable in $T_{R_\equiv}$ from the premises of $R_\equiv$, where $T_{R_\equiv} = \iota_{0*} \circ T$ is the translation of T along $\iota_0 : \Sigma \to \Sigma + \alpha_{R_\equiv}$.

Consider the premise $\Gamma_{\iota_0(i)} \vdash J_{\iota_0(i)}$ at index $\iota_0(i)$ for some $i \in I$. By Proposition 5.3, a presupposition of this premise is a presupposition $P = (\Gamma_i \vdash J')$ of the corresponding premise in R, translated along the signature map ℓ. By presuppositivity of R, the judgement P is derivable from $T_R = \iota_{0*} \circ T$, the translation of T along $\iota_0 : \Sigma \to \Sigma + \alpha_R$. By Corollary 4.41, we can translate such a derivation of P along ℓ_*, yielding a derivation in $T' = \ell_* \circ T_R$, where T' is T_R translated along ℓ. But $T' = \ell_* \circ T_R = \ell_* \circ \iota_{0*} \circ T = \iota_{0*} T = T_{R_\equiv}$, so we obtain a derivation in the correct theory.

The case of a premise indexed by $\iota_1(j)$ with $j \in I$ is similar, but the last step requires translation along the signature map $r = id_{\Sigma + \alpha_R} + \iota_1$ instead, mapping the metavariable symbols of α_R to the right-hand side metavariables of $R_\equiv$.

A premise P indexed by $\iota_2(k)$ is an equality associated to the k-th object premise of R. The presuppositions of P are derived by the corresponding object premises $\iota_0(k)$ and $\iota_1(k)$, and in the case of a term equation, the presupposition of the left-hand side.

A presupposition of the conclusion is derivable by appeal to the rule R itself for left and right hand side of the equation. In case $R_\equiv$ is a term equation, the type judgement arising as presupposition of the conclusion of R is derivable in T_R by presuppositivity of R, and can be translated along ℓ_* in the same way that we treated the left-hand copies of the premises. □

Proposition 5.10. *The raw structural rules, i.e., the equivalence relation rules and the conversion rules are acceptable for any type theory.*

Proof. Tightness is obvious. Presuppositivity is obvious for all but the conclusion of the equality conversion rule $\vdash s \equiv t : B$, which immediately follows from the ordinary conversion rule for term judgements. □

5.2 Acceptable type theories

It may happen that a raw type theory is flawed, even though each of its rules is acceptable. For instance, we might simply forget to state a rule governing one of the symbols, or provide two contradicting rules for the same symbol. Thus we also need a notion of acceptability of a raw type theory.

Definition 5.11. Suppose Σ is a signature and $S \in \Sigma$ has arity α_S. The **generic application of** S is the expression

$$\widehat{S} := S(\langle \mathsf{meta}_i(\langle \mathsf{var}_j \rangle_{j \in \mathrm{bind}_{\alpha_S} i}) \rangle_{i \in \mathrm{arg}\, \alpha_S}).$$

We say that an inference rule R is a **symbol rule for** S when its arity is α_S, the judgement form of the conclusion is the syntactic class of S, and its head is $\widehat{S}$.

Definition 5.12. A raw type theory T over Σ is:

1. **tight** if its rules are tight and there is a bijection β from the index set of Σ to the object rules of T such that, for every symbol S of Σ, $\beta(S)$ is a symbol rule for S;

2. **presuppositive** if all of its rules are presuppositive over T;

3. **substitutive** if all its rules have empty conclusion context; and

4. **congruous** if for every object rule of T the associated congruence rule (cf. Definition 4.34) is a rule of T.

A raw type theory is ***acceptable*** if it enjoys all of these properties.

The definition omits a common criterion for being "reasonable", namely there being a well-founded order that prevents cyclic references between parts of the theory. We address well-foundedness separately in Section 6, and provide a couple of examples showing how cyclic references may appear in an acceptable type theory.

Example 5.13. Let Q be a quantifier-like type symbol which takes a type and a term, and binds one variable in the term, with the raw rule

$$\frac{\vdash \mathsf{A} \text{ type} \qquad \mathsf{var}_0 : \mathsf{Q}(\mathsf{A}, \mathsf{t}(\mathsf{var}_0)) \vdash \mathsf{t}(\mathsf{var}_0) : \mathsf{A}}{\vdash \mathsf{Q}(\mathsf{A}, \mathsf{t}(\mathsf{var}_0)) \text{ type}}$$

The context in the second premise is *not* cyclic because Q binds var_0, but the premise itself is cyclic because the term metavariable t is introduced in a context that mentions it, and the rule is only presuppositive thanks to itself. Even so, the rule can still be used to derive judgements. For example, for any $\vdash t : A$ we can form the type $\mathsf{Q}(A, t)$. It is not clear what one would do with such rules, but we have no reason to banish them outright.

Example 5.14. The second example of cyclic references is a Tarski-style universe that contains itself, formulated as follows. Let u be a term constant and El a type symbol taking one term argument, with the raw rules

$$\frac{}{\vdash \mathsf{u} : \mathsf{El}(\mathsf{u})} \qquad\qquad \frac{\vdash \mathsf{a} : \mathsf{El}(\mathsf{u})}{\vdash \mathsf{El}(\mathsf{a}) \text{ type}}$$

Think of u as the code of the universe $\mathsf{El}(\mathsf{u})$ that contains itself, and El as the constructor taking codes to types. The rules themselves are not cyclic, and the type theory comprising them and the associated congruence rules is acceptable. However, in order to derive $\vdash \mathsf{El}(\mathsf{u})$ type, which is a presupposition for both rules, we need both rules. In this case the cycles can be broken easily enough: introduce a type constant $\vdash \mathsf{U}$ type and the equation $\vdash \mathsf{U} \equiv \mathsf{El}(\mathsf{u})$, then use U in place of $\mathsf{El}(\mathsf{u})$ in the above rules. In Section 6.5 we shall provide a general method for removing cyclic dependencies between rules by introduction of new symbols.

5.3 Derivability of presuppositions

Our first meta-theorem is a fairly easy one, giving a property that is always desired but not often explicitly discussed.

Theorem 5.15 (Presuppositions theorem)**.** *Let T be a raw type theory with all rules weakly presuppositive. If a judgement is derivable over T, then so are all its presuppositions.*

Proof. We proceed by induction on derivations D over T.

If D ends with a variable rule (Definition 4.24), then the only presupposition appears directly as the premise of the rule, so we may re-use its subderivation.

If D ends with a substitution rule (Definition 4.27), then its conclusion must be $\Delta \vdash f^*J$ for some substitution $f : \Delta \to \Gamma$ and judgement $\Gamma \vdash J$. By Proposition 5.5, each presupposition of the conclusion is $\Delta \vdash f^*J'$ for some presupposition $\Gamma \vdash J'$ of $\Gamma \vdash J$. But $\Gamma \vdash J$ is a premise of the last rule of D, so by induction we have a

derivation D' of $\Gamma \vdash J'$. So we can apply the substitution rule with $\Gamma \vdash J'$ (derived by D') and the same substitution f (with its premises derived as in D) to get the desired derivation of $\Delta \vdash f^*J'$.

Similarly, if D ends with an equality substitution rule (Definition 4.28), substituting an pair $f, g : \Delta \to \Gamma$ into a judgement $\Gamma \vdash J$, each presupposition of the conclusion can be derived by either a substitution (along f or g individually) or an equality substitution (along f, g) into some presupposition of $\Gamma \vdash J$.

The equivalence and conversion rules (Definitions 4.25 and 4.26) are presuppositive by Proposition 5.10, so we treat them together with the specific raw rules of T.

If D ends with an instance $(I, \Gamma)_*R$ of a raw rule R (either specific or structural), then its conclusion is of the form $(I, \Gamma)_*\Delta \vdash I_*J$, where $\Delta \vdash J$ is the conclusion of R. Now Proposition 5.4 tells us that each presupposition of the conclusion is an instantiation $(I, \Gamma)_*\Delta \vdash I_*J'$ of some presupposition $\Delta \vdash J'$ of $\Delta \vdash J$. Since R is weakly presuppositive, $\Delta \vdash J'$ is derivable from the premises of R plus their presuppositions. So by Corollary 4.43, $(I, \Gamma)_*\Delta \vdash I_*J'$ is derivable from the premises of $(I, \Gamma)_*R$ plus their presuppositions, which in turn are derivable by induction. $\square$

5.4 Elimination of substitution

In this section we show that over an acceptable type theory, the substitution rules (Definitions 4.27 and 4.28) can be eliminated: anything derivable with them is derivable without. At least, this will hold over a strict scope system; for a general scope system, it can be almost eliminated but not quite entirely.

Definition 5.16. An instance of the substitution rule (Definition 4.27) is a ***trivial renaming***, or just ***trivial***, if its substitution $f : \Delta \to \Gamma$ corresponds to a renaming on underlying scopes of the form $\mathrm{inl}^{-1} : |\Gamma| = |\Delta| \oplus 0 \to |\Delta|$, acting trivially at all positions.

Typically these arise with $\Gamma = (I, \Delta)_*[\,]$, an instantiation of the empty context; a trivial renaming is therefore of the form

$$\frac{(I, \Delta)_*[\,] \vdash J}{\Delta \vdash (\mathrm{inl}^{-1})_*J.}$$

In a *strict* scope system, trivial renamings are identities and hence redundant.

To avoid ambiguity with variance, we will in this section distinguish more carefully than usual between a renaming function $r : |\Gamma| \to |\Delta|$ and its associated substitution $\bar{r} : \Delta \to \Gamma$.

Definition 5.17. Call a derivation over a raw type theory T ***substitution-free*** if it uses only trivial instances of the substitution rule, and does not use the equality substitution rule. Equivalently, it uses just the variable rule, equality rules, conversion rules, trivial renamings, and the specific rules of T.

The core of this section, Lemma 5.20, will be that substitution is admissible for substitution-free derivations; this can be seen as defining an action of substitution on such derivations. We first need an analogous action of renaming, paralleling how substitution on expressions needed renaming to be defined first.

Lemma 5.18 (Admissibility of renaming)**.** *Let T be a substitutive type theory, with signature Σ. Let Γ and Γ' be contexts over Σ, and $r : |\Gamma| \to |\Gamma'|$ a renaming acting*

trivially at all positions in the sense of Definition 4.27, i.e. such that $\Gamma'_{r(i)} = r_*\Gamma_i$ for all $i \in \Gamma$. Then given a substitution-free derivation D of $\Gamma \vdash J$ in T, there is a substitution-free derivation r_*D of $\Gamma' \vdash r_*J$.
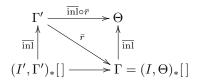
Proof. For this proof, we say a renaming **respects types** when it acts trivially at all positions; and say a derivation D with conclusion $\Gamma \vdash J$ is **renameable** if we have an operation giving, for every Γ' and renaming $r : |\Gamma| \to |\Gamma'|$ respecting types, a derivation r_*D of $\Gamma' \vdash r_*J$.

We show by induction that every derivation is renameable. Call the derivation under consideration D, and suppose given in each case suitable Γ', r.

If D concludes with a variable rule, giving $\Gamma \vdash \mathsf{var}_i : \Gamma_i$, then by induction, we can rename the derivation of the premise $\Gamma \vdash \Gamma_i$ type to a derivation of $\Gamma' \vdash \Gamma'_{r(i)}$ type, and then apply the variable rule to derive $\Gamma' \vdash \mathsf{var}_{r(i)} : \Gamma'_{r(i)}$, which is the desired judgement since r respects types.

If D concludes with a trivial renaming $s : \Gamma \to \Delta$, then by induction, the derivation of the premise is renameable; so renaming it along $r \circ s$, we are done.

Otherwise, D concludes with an instantiation $(I, \Theta)_*R$, where $I \in \mathrm{Inst}_{\Sigma,\Theta}(\alpha_R)$ is an instantiation, and R is either an equality rule, a conversion rule, or a specific rule of T. In each cases the conclusion of R is of the form $\vdash J'$, with empty context; so Γ is exactly $(I, \Theta)_*[]$, and J is has the form I_*J'. So now to derive $\Gamma' \vdash r_*(I_*J')$, we will apply the same raw rule R with the instantiation $I' := (r \circ \mathrm{inl})_*I$. Computing with renamings and instantiations according to (Proposition 3.27) shows that the conclusion of $(I', \Gamma')_*R$ is not quite $\Gamma' \vdash r_*J$, but is the same modulo a trivial renaming, according to the following commutative square:

$$
\begin{array}{ccc}
\Gamma' & \xrightarrow{\ \overline{\mathrm{inl}} \circ \bar{r}\ } & \Theta \\
\overline{\mathrm{inl}} \uparrow & \quad \nearrow \bar{r} \quad & \uparrow \overline{\mathrm{inl}} \\
(I', \Gamma')_*[] & \longrightarrow & \Gamma = (I, \Theta)_*[]
\end{array}
$$

We can therefore conclude the derivation of r_*D by the rule $(I', \Gamma')_*R$ followed by a trivial renaming.

It remains to derive the premises of $(I', \Gamma')_*R$. Each such premise is linked to a corresponding premise of $(I, \Theta)_*R$ by a renaming repecting types — specifically, a context extension of $r \circ \mathrm{inl}$. But by induction, we have renameable derivations of the premises of $(I, \Theta)_*R$; so we are done. $\qquad\square$

It is worthwhile to record a special case of admissibility of renaming.

Corollary 5.19 (Admissibility of weakening). *If a substitutive raw type theory derives $\Gamma \vdash J$ substitution-free, then it also derives $\Delta \vdash w_*J$ substitution-free for any* **weakening** *$w : \Gamma \to \Delta$, i.e., an injective variable renaming such that $\Delta_{w(i)} = w_*\Gamma_i$ for all $i \in |\Gamma|$.* $\qquad\square$

We can now give the action of substitution on derivations.

Lemma 5.20 (Admissibility of substitution). *Let T be a substitutive raw type theory over signature Σ. Let $f : \Delta \to \Gamma$ be a raw substitution over Σ, and $K \subseteq |\Gamma|$ a complemented subset such that:*

1. *f acts trivially at each $i \in K$ in the sense of Definition 4.27, i.e., for some $j \in \Delta$, $f(i) = \mathsf{var}_j$ and $\Delta_j = f^*\Gamma_i$*

2. *for each $i \in |\Gamma| \setminus K$, T derives $\Delta \vdash f(i) : f^*\Gamma_i$ without substitutions.*

Given a substitution-free derivation D of $\Gamma \vdash J$ over T, there is a substitution-free derivation f^*D of $\Delta \vdash f^*J$.

Before proceeding with the proof, we take a moment to comment on the condition the lemma assumes on f. It matches the condition used in the premises of the substitution rule, skipping type-checking on a set of indices K on which f acts trivially, and requiring derivability of $\Delta \vdash f(i) : f^*\Gamma_i$ only for $i \in |\Gamma| \setminus K$. An alternative, maybe more conventional condition would be to require $\Delta \vdash f(i) : f^*\Gamma_i$ for all $i \in |\Gamma|$.

There are a couple of reasons to weaken the condition as we do, thus strengthening the statement of the lemma. The superficial one is its applicability to raw substitutions that potentially contain ill-formed type expressions. The more essential one is that strengthened formulation is needed to keep the proof structurally inductive, allowing us to descend under a premise with a non-empty context, without needing to check that in the process the domain of f^* is extended with well-formed types. Even if they are in fact well formed, we cannot show this by appealing to an induction hypothesis, because the derivations involved are not structural subderivations of the one we are recursing over. What happens instead is that verification of well-formedness of types in contexts is deferred until their variables are accessed, at which point the variable rule provides the desired structural subderivations. This phenomenon seems to be a genuine consequence of spelling out the proof for a *general* class of type theories. For any specific type theory, only certain concrete type-schemes will occur in contexts of premises of rules; and these specific type-schemes are always designed by their authors in such a way that they can be shown well-formed individually, so that the inductive arguments do not break.

Proof of Lemma 5.20. Within this proof, all derivations are assumed substitution-free, and a (substitution-free) derivation D of a judgement $\Gamma \vdash J$ is called **substitutable** when, for all Δ, f and K satisfying the condition of the lemma, we have a (substitution-free) derivation of $\Delta \vdash f^*J$. We prove by induction that every derivation D is substitutable. Much of the proof parallels that of Lemma 5.18.

Suppose D concludes with a variable rule showing $\Gamma \vdash \mathsf{var}_i : \Gamma_i$, and $f : \Delta \to \Gamma$ is a suitable substitution, acting trivially on $K \subseteq |\Gamma|$. When $i \in K$, we work just as in Lemma 5.18: given a suitable substitution into the conclusion, we inductively substitute the premise derivation along the same substitution, and then conclude with the variable rule. Otherwise, for $i \in |\Gamma| \setminus K$, we use the derivation of $\Delta \vdash f(i) : f^*\Gamma_i$ given by assumption.

Next, if D concludes with a trivial renaming $\overline{\mathsf{inl}^{-1}} : \Gamma \to \Gamma'$, to conclude $\Gamma \vdash J$, then suppose $f : \Delta \to \Gamma$ is a substitution acting trivially on K and with derivations of $\Delta \vdash f(i) : f^*\Gamma_i$ for $i \in |\Gamma| \setminus K$. Then the substitution $\overline{\mathsf{inl}^{-1}} \circ f : \Delta \to \Gamma'$ acts trivially on $\mathsf{inl}(K) \subseteq |\Gamma'|$, and the same derivations witness that $\Delta \vdash f(\mathsf{inl}^{-1} i) : f^*\Gamma'_i$ for $i \in |\Gamma'| \setminus \mathsf{inl}(K)$. So by induction, we can substitute the derivation of the premise along $\overline{\mathsf{inl}^{-1}} \circ f$ to derive $\Delta \vdash f^*J$ as required.

Otherwise, D must conclude with an instantiation $(I, \Theta)_* R$, for some instantiation $I \in \mathrm{Inst}_{\Sigma, \Theta}(\alpha_R)$ and R a flat rule (structural or specific) with empty-context conclusion $\vdash J$. So, suppose given a suitable substitution $f : \Delta \to (I, \Theta)_*[]$, acting trivially on $K \subseteq |(I, \Theta)_*[]| = \Theta \oplus 0$; we need to derive $\Delta \vdash f^*I_*J$.

Just as in Lemma 5.18, we substitute I along $\overline{\mathsf{inl}} \circ f : \Delta \to \Theta$ to get another instantiation I' of α_R over Δ, such that the conclusion of $(I', \Delta)_* R$ is just a trivial renaming away from $\Delta \vdash f^*I_*J$. So it remains just to derive the premises of $(I', \Delta)_* R$.

Again as in Lemma 5.18, by induction we have substitutable derivations of all premises of $(I, \Theta)_* R$. So it suffices to give, for each premise $(I', \Delta)_* \Psi \vdash I'_* J'$ of $(I', \Delta)_* R$, a substitution $g : (I', \Delta)_* \Psi \to (I, \Theta)_* \Psi$ to the corresponding premise of $(I, \Theta)_* R$, with $g^* I_* J' = I'_* J'$, and with g satisfying the conditions of the lemma. (Recall that $(I, \Theta)_* \Psi$ is the context extension of Θ by the instantiations of types from Ψ, with positions $|\Theta| \oplus |\Psi|$, and $(I', \Delta)_*$ similarly.) We define:

$$g := (\overline{\mathrm{inl}} \circ f) \oplus |\Psi| : (I', \Delta)_* \to (I, \Theta)_* \Psi.$$

Now $g^* I_* J' = I'_* J'$ follows directly from Proposition 3.27 (which we will continue to use without further comment), the definitions of I' and g, and the following commuting diagram.

$$
\begin{array}{ccc}
(I', \Delta)_* \Psi & \xrightarrow{\ g\ } & (I, \Theta)_* \Psi \\
{\scriptstyle \overline{\mathrm{inl}}}\big\downarrow & & \big\downarrow{\scriptstyle \overline{\mathrm{inl}}} \\
\Delta & \xrightarrow{\ \overline{\mathrm{inl}} \circ f\ } & \Theta \\
{\scriptstyle \overline{\mathrm{inl}}}\big\uparrow & {\scriptstyle f}\nearrow & \big\uparrow{\scriptstyle \overline{\mathrm{inl}}} \\
(I', \Delta)_* [] & \xrightarrow{\hspace{2em}} & (I, \Theta)_* []
\end{array}
$$

Next, g clearly acts trivially on $\mathrm{inr}_* |\Psi| \subseteq |(I, \Theta)_* \Psi|$. It also acts trivially on the subset of $\mathrm{inl}_* |\Theta|$ corresponding to the given $K \subseteq |\Theta| \oplus 0$ on which f acts trivially.

Taking the union of these as the trivial set for g, it remains to show that for i in the subset of $\mathrm{inl}_* |\Theta|$ corresponding to $|\Theta| \oplus 0 \setminus K$, we have $(I', \Delta)_* \Psi \vdash g(i) : g^*((I, \Theta)_* \Psi)_i$. But this judgement is just the renaming of $\Delta \vdash f(j) : f^*((I, \Theta)_* \emptyset)_i$ along the evident map $|\Delta| \oplus 0 \to |\Delta| \oplus |\Psi|$. So using Lemma 5.18 to rename the derivation of $\Delta \vdash f(j) : f^*((I, \Theta)_* [])_i$ supplied with f, we are done. $\qquad\square$

Next, we show that substitution respects judgemental equality of raw substitutions. For this, we introduce a handy notation: for raw substitutions $f, g : \Gamma' \to \Gamma$ and an object judgement J, with head expression e and boundary B, we write $(f \equiv g)^* J$ for the equality judgement asserting that $f^* e$ and $g^* e$ are equal over the boundary $f^* B$. Thus $\Gamma' \vdash (f \equiv g)^*(A \ \mathsf{type})$ stands for $\Gamma' \vdash f^* A \equiv g^* A$ and $\Gamma' \vdash (f \equiv g)^*(e : A)$ stands for $\Gamma' \vdash f^* e \equiv g^* e : f^* A$.

Lemma 5.21 (Admissibility of equality substitution). *Let T be a substitutive and congruous raw type theory over Σ. Let $f, g : \Gamma' \to \Gamma$ be raw substitutions over Σ, and $K \subseteq |\Gamma|$ a complemented subset such that:*

1. *for each $i \in K$ there exists some (necessarily unique) $j \in |\Gamma'|$ such that $f(i) = g(i) = \mathsf{var}_j$, and $\Gamma'_j = f^* \Gamma_i$ or $\Gamma'_j = g^* \Gamma_i$,*

2. *for each $i \in |\Gamma| \setminus K$, T derives $\Gamma' \vdash f(i) : f^* \Gamma_i$ and $\Gamma' \vdash g(i) : g^* \Gamma_i$ and $\Gamma' \vdash f(i) \equiv g(i) : f^* \Gamma_i$ without substitutions.*

If T derives $\Gamma \vdash J$ without substitutions, then T derives $\Gamma' \vdash f^* J$, $\Gamma' \vdash g^* J$, and (if J is an object judgement) $\Gamma' \vdash (f \equiv g)^* J$, still without substitutions.

The assumption on f and g is perhaps a little surprising, especially the last "or" in case (1). Another peculiarity is the fact that we include $\Gamma' \vdash f^* J$ and $\Gamma' \vdash g^* J$ in the conclusion rather than obtaining them by elimination of substitution. The point is that the induction arguments need to work when we pass into extended contexts of

premises, whereby types of the form f^*A *or* g^*A are introduced; so we cannot assume either f or g satisfying the conditions of Theorem 5.22 individually, but need to give a condition on them together that is preserved. And since this condition is too weak for applying elimination of substitution to f or g, we carry the conclusion of that along as well.

Proof of Lemma 5.21. The proof proceeds by induction on the derivation D of $\Gamma \vdash J$. The details are closely analogous to elimination of substitution, so we spell out fewer.

Consider the case when D ends with a variable rule

$$\frac{\Gamma \vdash \Gamma_i \text{ type}}{\Gamma \vdash \mathsf{var}_i : \Gamma_i}$$

If case (2) applies for i, we are done immediately. If case (1) applies, we obtain derivations of $\Gamma' \vdash f_*\Gamma_i$ type, $\Gamma' \vdash g_*\Gamma_i$ type, and $\Gamma' \vdash f_*\Gamma_i \equiv g_*\Gamma_i$ by induction hypothesis. If $\Gamma'_j = f_*\Gamma_i$ then $\Gamma' \vdash \mathsf{var}_j : f_*\Gamma_i$ follows by the variable rule and $\Gamma' \vdash \mathsf{var}_j : g_*\Gamma_i$ from it by conversion. And of course, $\Gamma' \vdash \mathsf{var}_j \equiv \mathsf{var}_j : \Gamma'_j$ is derivable by reflexivity. If $\Gamma'_j = g_*\Gamma_i$, the situation is symmetric.

Otherwise, D concludes with an instantiation I_*R where $I \in \mathrm{Inst}_{\Sigma,\Gamma}(\alpha_R)$ and R is either an equality rule, a conversion rule, or a specific rule of T. The conclusion of I_*R has the form $\Gamma \vdash I_*J'$. We need to derive $\Gamma' \vdash f^*(I_*J')$, $\Gamma' \vdash g^*(I_*J')$, and if R is an object rule then also $\Gamma' \vdash (f \equiv g)^*(I_*J')$.

Let us first verify that the raw substitutions $f' := f \oplus |\Delta| : \Gamma' . (f^*I)_*\Delta \to \Gamma . I_*\Delta$ and $g' := g \oplus |\Delta| : \Gamma' . (f^*I)_*\Delta \to \Gamma . I_*\Delta$ satisfy the conditions (1) and (2) of the lemma for the complemented subset $K' \subseteq |\Gamma . I_*\Delta| = |\Gamma| + |\Delta|$, given by $K' = \{\iota_0(i) \mid i \in K\} \cup \{\iota_1(k) \mid k \in |\Delta|\}$:

1. For $\iota_0(i) \in K'$, there is $j \in \Gamma'$ such that $f(i) = g(i) = \mathsf{var}_j$ and either $\Gamma'_j = f^*\Gamma_i$ or $\Gamma'_j = g^*\Gamma_i$. Now f' and g' satisfy condition (1) because $f'(\iota_0(i)) = \mathsf{var}_{\iota_0(j)} = g'(\iota_0(i))$ and $(\Gamma' . (f^*I)\Delta_*)_{\iota_0(j)} = \iota_{0*}\Gamma'_j$, which is equal either to $\iota_{0*}(f^*\Gamma_i) = f'^*(\Gamma . I_*\Delta)$ or to $\iota_{0*}(g^*\Gamma_i) = g'^*(\Gamma . I_*\Delta)$, as the case may be.

2. For $\iota_1(k) \in K'$, condition (1) is satisfied by f' and g': it is clear that $f'(\iota_1(k)) = \mathsf{var}_{\iota_1(k)} = g'(\iota_1(k))$, while $f'^*(\Gamma . I_*\Delta)_{\iota_1(k)} = f'^*(I_*\Delta_k) = (f^*I)_*\Delta_k = (\Gamma' . (f^*I)_*\Delta)_{\iota_1(k)}$ holds, where we used Proposition 3.27 in the second step.

3. For $\iota_0(j) \in (|\Gamma|+|\Delta|) \setminus K'$, f' and g' satisfy (2) because the desired judgements

$$\Gamma' . (f^*I)_*\Delta \vdash f'(\iota_0(j)) : f'^*(\Gamma . I_*\Delta)_{\iota_0(j)}$$
$$\Gamma' . (f^*I)_*\Delta \vdash g'(\iota_0(j)) : g'^*(\Gamma . I_*\Delta)_{\iota_0(j)}$$
$$\Gamma' . (f^*I)_*\Delta \vdash f'(\iota_0(j)) \equiv g'(\iota_0(j)) : f'^*(\Gamma . I_*\Delta)_{\iota_0(j)}$$

are respectively equal to

$$\Gamma' . (f^*I)_*\Delta \vdash \iota_{0*}(f(j)) : \iota_{0*}(f^*\Gamma_j)$$
$$\Gamma' . (f^*I)_*\Delta \vdash \iota_{0*}(g(j)) : \iota_{0*}(g^*\Gamma_j)$$
$$\Gamma' . (f^*I)_*\Delta \vdash \iota_{0*}(f(j)) \equiv \iota_{0*}(g(j)) : \iota_{0*}(f^*\Gamma_j)$$

and these are derivable by Lemma 5.18 applied to the renaming ι_0 and the assumptions (2) for f and g.

We similarly check that the raw substitutions $f'' := f \oplus |\Delta| : \Gamma'.(g^*I)_*\Delta \to \Gamma.I_*\Delta$ and $g'' := g \oplus |\Delta| : \Gamma' . (g^*I)_*\Delta \to \Gamma . I_*\Delta$ satisfy the conditions of the lemma with the same set K', too. The verification is similar to the case of f' and g' above, and at this point the disjunction in (1) lets us exchange the role of g and f.

We may now derive $\Gamma' \vdash f^*(I_*J')$ by the closure rule $(f^*I)_*R$, as it has the correct conclusion by Proposition 3.27. To see that its premises are derivable, we verify that for any premise $\Delta \vdash J''$ of R, the instantiation by f^*I, namely

$$\Gamma' . (f^*I)_*\Delta \vdash (f^*I)_*J'' \tag{5.7}$$

is derivable. The corresponding premise of I_*R, which is

$$\Gamma . I_*\Delta \vdash I_*J'', \tag{5.8}$$

is derivable by assumption, and so we obtain (5.7) by the induction hypothesis for (5.8) applied to f' and g'.

By a similar argument $\Gamma' \vdash g^*(I_*J')$ is derivable by the closure rule $(g^*I)_*R$, we only need to use f'' and g'' instead of f' and g' to derive the premise

$$\Gamma' . (g^*I)_*\Delta \vdash (g^*I)_*J''. \tag{5.9}$$

It remains to be checked that $\Gamma' \vdash (f \equiv g)^*(I_*J')$ is derivable when R is an object rule. Because T is congruous, the congruence rule C associated with R is a specific rule of T. Let $\ell, r : \Sigma + \alpha_R \to \Sigma + \alpha_C$ be the signature maps from Definition 4.34 and $I' := f^*I + g^*I \in \mathrm{Inst}_{\Sigma,\Gamma'}(\alpha_R + \alpha_R)$. Note that $I'_* \circ \ell_* = f^*I$ and $I'_* \circ r_* = g^*I$. The instantiation I'_*C is a closure rule whose conclusion is precisely $\Gamma' \vdash (f \equiv g)^*(I_*J')$, so we only have to establish that its premises are derivable, of which there are three kinds:

1. For each premise $\Delta \vdash J''$ of R, there is a corresponding premise $I'_*(\ell_*(\Delta \vdash J''))$, which is equal to (5.7). We have already seen that it is derivable.

2. For each premise $\Delta \vdash J''$ of R, there is a corresponding premise $I'_*(r_*(\Delta \vdash J''))$, which is equal to (5.9). Its derivability has been established, too.

3. For each object premise $\Delta \vdash J''$ of R, there is a corresponding premise, namely the associated equality judgement (Definition 4.33) instantiated by I'. A short calculation relying on Proposition 3.27 shows that the judgement is

$$\Gamma' . (f^*I)_*\Delta \vdash (f' \equiv g')^*J'',$$

   which is one of the consequences of the induction hypothesis for (5.8) applied to f' and g'. $\qquad \square$

We can now put these together into the main theorem of this section.

Theorem 5.22 (Elimination of substitution). *Let T be a substitutive and congruous raw type theory; then every derivable judgement over T has a substitution-free derivation.*

Proof. Work by induction over the original derivation. At substitution rules, apply Lemma 5.20; and at equality substitution rules, Lemma 5.21. $\qquad \square$

5.5 Uniqueness of typing

Whether it is desirable for a term to have many types depends on one's motivations, but certainly in our setting, where the terms record detailed information about premises, we should expect a term to have at most one type, which we prove here.

Theorem 5.23. *If a tight, substitutive raw type theory T derives $\Gamma \vdash A$* type*, $\Gamma \vdash B$* type*, $\Gamma \vdash t : A$ and $\Gamma \vdash t : B$ then it also derives $\Gamma \vdash A \equiv B$.*

Proof. By Theorem 5.22 it suffices to prove the claim for substitution-free derivations. Suppose we have derivations D_A, D_B, D_1 and D_2:

$$\frac{D_A}{\Gamma \vdash A \text{ type}} \qquad \frac{D_B}{\Gamma \vdash B \text{ type}} \qquad \frac{D_1}{\Gamma \vdash t : A} \qquad \frac{D_2}{\Gamma \vdash t : B}$$

The proof proceeds by a double induction on the derivations D_1 and D_2.

Consider the case where D_1 ends with a conversion:

$$\frac{\dfrac{D_{1,A'}}{\Gamma \vdash A' \text{ type}} \quad \dfrac{D_{1,A}}{\Gamma \vdash A \text{ type}} \quad \dfrac{D_{1,t}}{\Gamma \vdash t : A'} \quad \dfrac{D_{1,\text{eq}}}{\Gamma \vdash A' \equiv A}}{\Gamma \vdash t : A}$$

We apply the induction hypothesis to $D_{1,t}$ and D_2 to derive $\Gamma \vdash A' \equiv B$. The desired $\Gamma \vdash A \equiv B$ now follows from $D_{1,\text{eq}}$ by symmetry and transitivity of equality. The case where D_2 ends with a conversion is symmetric, except that it does not require the use of symmetry.

Consider the case where D_1 ends with a variable rule:

$$\frac{\dfrac{D_1'}{\Gamma \vdash \Gamma_j \text{ type}}}{\Gamma \vdash \mathsf{var}_j : \Gamma_j}$$

Because T is tight D_2 must end with a variable or a conversion rule. We have already dealt with the latter one. If D_2 ends with a variable rule, then $A = \Gamma_j = B$, and we may conclude $\Gamma \vdash A \equiv B$ by reflexivity.

In the remaining case D_1 and D_2 both end with instantiations of specific rules of T. Let β be the map which takes each symbol $S \in \Sigma$ to the corresponding symbol rule in T. There is a unique symbol $S \in \Sigma$, such that D_1 and D_2 both end with instantiations of $\beta(S)$:

$$\frac{D_1}{\Gamma \vdash I_* S(\langle \mathsf{meta}_i \rangle_{i \in \arg S}) : I_* C} \qquad \frac{D_2}{\Gamma \vdash J_* S(\langle \mathsf{meta}_j \rangle_{j \in \arg S}) : J_* C}$$

Of course, $I_* C$ is just A and $J_* C$ is B, and both heads are equal to t, from which it follows that

$$S(\langle I_* \mathsf{meta}_i \rangle_{i \in \arg S}) = S(\langle J_* \mathsf{meta}_i \rangle_{i \in \arg S}),$$

and so I and J are equal because they match on every $i \in \arg S$. Thus $A = I_* C = J_* C = B$, and we may derive $\Gamma \vdash A \equiv B$ by reflexivity. $\qquad \square$

We record a more economical version of uniqueness of typing, which one can afford in reasonable situations.

Corollary 5.24. *If an acceptable type theory derives $\Gamma \vdash e : A$ and $\Gamma \vdash e : B$ then it also derives $\Gamma \vdash A \equiv B$.*

Proof. Apply Theorem 5.15 to $\Gamma \vdash e : A$ and $\Gamma \vdash e : B$ to obtain $\Gamma \vdash A$ type and $\Gamma \vdash B$ type, and conclude by Theorem 5.23. $\square$

Acceptability also easily gives us uniqueness of typing for term equalities.

Corollary 5.25. *If an acceptable type theory derives $\Gamma \vdash s \equiv t : A$ and $\Gamma \vdash s \equiv t : B$ then it also derives $\Gamma \vdash A \equiv B$.*

Proof. Again, apply Theorem 5.15 to get $\Gamma \vdash A$ type and $\Gamma \vdash B$ type, and conclude by Theorem 5.23. $\square$

5.6 An inversion principle

Given the fact that a judgement $\Gamma \vdash J$ is derivable, to what extent can a derivation of it be constructed just from the information given in the judgement? We show in this section that, for sufficiently well-behaved type theories, one can read off the proof-relevant part of a derivation from the head of J. The proof-irrelevant parts are the applications of conversion rules, and subderivations of equalities. The former may be arranged to always appear just once after variable and symbol rules, while the latter must be dealt with on a case-by-case basis, as a particular type theory may or may not possess an algorithm that checks derivability of equalities.

When we attempt to reconstruct a derivation from a judgement, the first obstacle we face is what types should be given to the subterms appearing in a judgement. For the variables the answer is clear, while for symbol expressions it is natural to use the types dictated by the corresponding rules, as follows.

Definition 5.26. Let T be a tight raw type theory over Σ and β the assignment of rules to the symbols of Σ. Thus for each term symbol $S \in \Sigma$, the conclusion of $\beta(S)$ takes the form $\vdash \widehat{S} : A_S$ for some $A_S \in \mathrm{Expr}^{\mathrm{ty}}_{\Sigma + \alpha_R}(0)$. Given a term expression $t \in \mathrm{Expr}^{\mathrm{tm}}_{\Sigma}(\Gamma)$, its **natural type** $\tau_\Gamma(t) \in \mathrm{Expr}^{\mathrm{ty}}_{\Sigma}(\Gamma)$ is defined by

$$\tau_\Gamma(\mathsf{var}_i) := \Gamma_i \qquad \text{and} \qquad \tau_\Gamma(S(e)) := e_* A_S,$$

where we used $e \in \prod_{i \in \mathrm{arg}\, S} \mathrm{Expr}^{\mathrm{cls}\, i}_{\Sigma}(\gamma \oplus \mathrm{bind}_S i)$ as an instantiation, so that $e_* A_S$ is the expression in which each $\mathsf{meta}_i(e')$ is replaced by $(e')^* e_i$.

To put it more simply, the natural type of $S(e)$ is the type one obtains by applying the symbol rule for S to the premises determined by e.

Theorem 5.27 (Inversion principle). *Let T be an acceptable type theory over Σ.*

1. *If T derives $\Gamma \vdash \mathsf{var}_i : A$ then it does so by an application of a variable rule, followed by a conversion:*

$$\frac{\dfrac{D'}{\Gamma \vdash \mathsf{var}_i : \Gamma_i} \qquad \dfrac{D''}{\Gamma \vdash \Gamma_i \equiv A}}{\Gamma \vdash \mathsf{var}_i : A}$$

2. *If T derives $\Gamma \vdash S(e) : A$ then it does so by an application of the symbol rule for S, followed by a conversion:*

$$\frac{\dfrac{D'}{\Gamma \vdash S(e) : \tau_\Gamma(S(e))} \qquad \dfrac{D''}{\Gamma \vdash \tau_\Gamma(S(e)) \equiv A}}{\Gamma \vdash S(e) : A}$$

3. *If T derives $\Gamma \vdash S(e)$* type *then it does so by an application of the symbol rule for S.*

Proof. Let T be an acceptable type theory over Σ, and β the assignment of symbol rules to the symbols of Σ. To establish the first two claims, we proceed by induction on a substitution-free derivation D, which exists by Theorem 5.22.

If D ends with a variable rule,

$$\frac{\dfrac{D}{\Gamma \vdash \Gamma_i \text{ type}}}{\Gamma \vdash \mathsf{var}_i : \Gamma_i}$$

then we obtain the desired derivation by attaching a dummy conversion rule:

$$\frac{\dfrac{\dfrac{D}{\Gamma \vdash \Gamma_i \text{ type}}}{\Gamma \vdash \mathsf{var}_i : \Gamma_i} \qquad \dfrac{\dfrac{D}{\Gamma \vdash \Gamma_i \text{ type}}}{\Gamma \vdash \Gamma_i \equiv \Gamma_i}}{\Gamma \vdash \mathsf{var}_i : \Gamma_i}$$

Otherwise, D ends with an application of the conversion rule

$$\frac{\dfrac{D'}{\Gamma \vdash \mathsf{var}_i : B} \qquad \dfrac{D''}{\Gamma \vdash B \equiv A}}{\Gamma \vdash \mathsf{var}_i : A}$$

We apply the induction hypothesis to D' to obtain a derivation of the form

$$\frac{\dfrac{D^*}{\Gamma \vdash \mathsf{var}_i : \Gamma_i} \qquad \dfrac{D^{**}}{\Gamma \vdash \Gamma_i \equiv B}}{\Gamma \vdash \mathsf{var}_i : B}$$

Using the transitivity rule, we combine D^{**} and D'' into a derivation of $\Gamma \vdash \Gamma_i \equiv A$, which can then be used together with D^* to get the desired form of derivation.

If D ends with an application of the symbol rule $\beta(S)$,

$$\frac{D'}{\Gamma \vdash S(e) : \tau_\Gamma(S(e))},$$

then by Theorem 5.15 there is a derivation D'' of the presupposition $\Gamma \vdash \tau_\Gamma(S(e))$ type. We apply reflexivity to D'' to obtain $\Gamma \vdash \tau_\Gamma(S(e)) \equiv \tau_\Gamma(S(e))$, and then conversion to get the desired derivation. Otherwise, D ends with an application of a conversion rule, in which case we proceed as in the variable case.

The third claim is trivial, because $\beta(S)$ is the only rule which can be instantiated to have the conclusion $\Gamma \vdash S(e)$ type, apart from substitution rules, which we have dispensed with. $\qquad \square$

The above theorem may be applied repeatedly to obtain a canonical form of the proof-relevant part of a derivation. The missing subderivations of equalities must be provided by other means. Also notice that it is easy enough to avoid insertion of unnecessary appeals to conversion rules along reflexivity.

A useful consequence of Theorem 5.27 is the fact that a type of a term may be calculated directly from the term (and the symbol rules), as long as it has one.

Corollary 5.28. *In an acceptable type theory, a typeable term has its natural type.*

Proof. Whenever $\Gamma \vdash e : A$ is derivable, then so is $\Gamma \vdash e : \tau_\Gamma(e)$ because its derivation appears as a subderivation in the statement of Theorem 5.27. $\qquad\square$

6 Well-founded presentations

So far, our type theories have omitted one typical characteristic occurring in practice: the *ordering* of the presentation of the theory. This ordering appears, implicitly or explicitly, at three levels:

1. The *positions of a context* usually form a finite sequence, and each type depends only on the preceding part of the context.

2. The *premises of each rule* typically follow some well-founded order, usually simply a finite sequence, and the boundary of each premise depends only on the earlier ones.

3. The *rules of the theory* are themselves well-founded, and each rule depends only on the earlier rules. This order is quite often infinite, in for instance theories with hierarchies of universes, and need not be total, as seen in the example below.

At each of these three levels, the "depends only on" holds in two senses:

1. *Raw expressions*: each type expression of a context uses only the preceding variables; in a rule, the expressions of each premise boundary only use previously-introduced metavariables; and in a theory, the raw premises and boundary of a rule only use previously-introduced symbols of the theory.

2. *Derivations for presuppositivity*: each type expression in a context is a derivable type over just the preceding part of the context; each premise of a rule can be checked well-formed using just the preceding premises; and so can each rule using just the earlier rules.

Example 6.1. The Π-formation rule uses no symbols of the signature in its premises or boundary, and relies only on structural rules for its well-formedness. The rules for λ-abstraction and function application both use Π in their raw expressions, and depend on the Π-formation rule for their reasonability, but not on any other earlier symbols or rules. And the β-reduction rule in turn depends on all three of these.

Similarly, there is a natural order within the rules for Σ-types. On the other hand, neither of the Σ- or Π-type groups naturally precedes the other, and it would be unnatural to force them into a total order.

Traditionally, well-foundedness is treated in two different ways, depending on the levels. Since the class of all contexts of a theory is formally defined — contexts are "user-definable" — their well-foundedness must be explicitly mandated somehow; and so it is, usually by the context judgement $\vdash \Gamma$ cxt.

Rules and theories by contrast are not "user-definable": each development usually presents a single theory, or a few, with a specific collection of rules. The ordering on these can therefore be left entirely unstated, but it is almost always clearly present. The writer ensures it when setting up the theory; the reader follows it when understanding the theory, and convincing themself of its reasonableness; and it is respected in later proofs and constructions.

It would be jarring, for instance, and often logically impossible, to give the semantics of β-reduction before that of λ-abstraction. On the other hand, it would be unsurprising if a writer introduced the rules for Π-types before those for Σ-types, but then gave semantics with Σ-types first. Overall, the implicit partial order on rules is always respected, but no particular total extension of it is.

Defining what it means for a presentation to be ordered is a little subtler than one might expect; we work up to it gradually, considering contexts first, then rules, and finally theories.

6.1 Sequential contexts

In our setting, with scoped syntax, and with contexts as maps from positions to types (we henceforth refer to these as *flat* contexts), traditional sequential contexts may be recovered in various ways. They are all straightforwardly equivalent — indeed, a sufficiently informal statement of the traditional definition could be read as any of them — but explicitly comparing them provides a useful warmup for the less straightforward cases with rules and type theories later.

Recall that $[n]$ denotes the sum of $n \in \mathbb{N}$ copies of $[1]$. When working with sequential contexts, we will identify the positions of $[n]$ with $\{0, \ldots, n-1\}$, and denote the evident "subscope inclusion" maps by $w_{i,j} : [i] \to [j]$.

Definition 6.2 (Sequential context I). A **raw sequential context** over a signature Σ is a list $\Gamma = [\Gamma_0, \ldots, \Gamma_{n-1}]$, where $\Gamma_i \in \mathrm{Expr}_{\Sigma}^{\mathsf{ty}}([i])$, for each $i \in [n]$. We write $\Gamma_{<i}$ for the initial segment $[\Gamma_0, \ldots, \Gamma_{i-1}]$. The **flattening** of a raw sequential context is the raw flat context of scope $[n]$ whose i-th type is the weakening $w_{i,n_*}(\Gamma_i)$. We typically leave flattening implicit, writing Γ both for a sequential context and its flattening.

Given a signature Σ and a raw type theory T over it, a raw sequential context Γ over Σ is **well-formed over** T if for each $i \in \Gamma$, the judgement $\Gamma_{<i} \vdash \Gamma_i$ type is derivable.

Alternately, we can define sequentiality as a property of flat contexts:

Definition 6.3 (Sequential context II). A raw flat context Γ of scope $[n]$ is **sequential** if for each $i \in [n]$, all variables var_j occurring in Γ_i have $j < i$. Thus each Γ_i is uniquely of the form $w_{i,n_*}(\overline{\Gamma_i})$, from which we define the initial segments $\Gamma_{<i}$ as sequential raw contexts of scope $[i]$.

A sequential context Γ over Σ is **well-formed over** T if for each $i \in \Gamma$, the judgement $\Gamma_{<i} \vdash \Gamma_i$ type is derivable.

Finally, we can define well-formed flat contexts via the traditional derivation rules, without reference to raw sequential contexts.

Definition 6.4 (Sequential context III). The property $\vdash \Gamma$ cxt, read as "Γ is **sequentially well-formed**" over a given theory, is the inductive predicate on flat contexts de-

fined by the following closure conditions, the latter for all suitable Γ, A:

$$\frac{}{\vdash [\,] \; \mathsf{cxt}} \qquad\qquad \frac{\vdash \Gamma \; \mathsf{cxt} \qquad \Gamma \vdash A \; \mathsf{type}}{\vdash \Gamma \,.\, A \; \mathsf{cxt}}$$

Each of the above definitions moreover has two possible readings: *proof-relevant*, where by derivability of a judgement $\Gamma \vdash A$ type we mean that a specific derivation is given, and *proof-irrelevant*, where we merely mean that some derivation exists. We take the proof-relevant reading in all cases.

Proposition 6.5. *Definitions 6.2, 6.3 and 6.4 are all equivalent, as predicates on flat contexts, in both their proof-relevant and -irrelevant forms.*

Proof. Essentially straightforward, given the fact, already mentioned in Definition 6.3, that the variable-occurrence constraint there precisely characterises the images of the weakenings $w_{i,n_*} : \mathrm{Expr}^{\mathsf{ty}}_{\Sigma}([i]) \hookrightarrow \mathrm{Expr}^{\mathsf{ty}}_{\Sigma}([n])$. $\qquad\square$

Of these definitions, Definition 6.3 is the simplest to state, especially if one sweeps under the rug the inverse-weakening required for defining initial segments. However, when spelling out details carefully, this inverse-weakening is tedious to keep track of. When we bump these definitions up to sequential rules or type theories, therefore, we will focus on approaches based on Definitions 6.2 and 6.4.

6.2 Sequential rules

Next, we wish to define sequential *rules*, in which premises form a finite sequence, and each refers only to the previous ones. Analogously to Definition 6.3, the easiest version to state is to start from an ordinary raw rule, and add desirable properties, together with restrictions on how earlier parts can be used in later parts:

Definition 6.6 (Sequential rule, provisional)**.** Let R be a tight raw rule over a signature Σ, with premises indexed by $[n]$, and β the bijection witnessing its tightness. We say that R is ***sequential*** if for all $i \in [n]$ and $j \in \alpha_R$, if meta_j appears in the i-th premise, then $\beta(j) < i$.

Moreover, say that R is *(sequentially) well-formed* over a raw type theory T, also over Σ, when for all $i \in [n]$, the presuppositions of the i-th premise of R can be derived from the premises indexed by $[i]$.

This definition is adequate, but is in several regards somewhat unsatisfying:

1. We have said here, as in Definition 6.3, that the premises are formed over the extension by all the metavariables, and their presuppositions derived from all the premises, but use only the preceding ones. When applying this condition, one typically wants to consider them as formed, or derived, over the extension by just the preceding initial segment. So rather than restricting them back there, and having to keep track of such restriction, it is simpler to say from the start, as in Definition 6.2, that they are formed, or derived, over those initial segments.

2. "Tightness" gives a redundancy of data in two ways. Firstly, the heads of all object-judgement premises are redundant: each head must be the corresponding metavariable, applied to all variables of its scope. Besides this, the arity itself is determined by the indexing family of the rule together with the scopes and forms of the premises.

These issues can be remedied by defining sequential presentations inductively, analogously to Definition 6.4, and adding each premise not as a full judgement but just its *boundary*, whose head, if any, will be filled in automatically to ensure tightness by construction.

Definition 6.7. Given a signature Σ and a raw type theory T over it, we define inductively the ***sequential premise-families*** P with arities α_P, and simultaneously their ***flattenings*** as families of judgements over $\Sigma + \alpha_P$, as follows.

1. The ***empty sequential premise-family*** $\langle\,\rangle$ has empty arity $\alpha_{\langle\,\rangle}$, and its flattening is the empty family.

2. Let P be a sequential premise-family, with arity α_P and flattening F. Let $\Delta \vdash B$ be a boundary over $\Sigma + \alpha_P$, such that all presuppositions of $\Delta \vdash B$ are derivable over $(T + F) + \alpha_P$ (i.e. the translation of $T + F$ from Σ to $\Sigma + \alpha_P$), and Δ is a well-formed sequential context over the same theory.

   Then there is an ***extension sequential premise-family*** $P; (\Delta \vdash B)$.

   If $\Delta \vdash B$ is an object boundary of class c, then the associated arity $\alpha_{P;(\Delta\vdash B)}$ is $\alpha_P + \langle (c, |\Delta|) \rangle$; or if $\Delta \vdash B$ is an equality boundary, $\alpha_{P;(\Delta\vdash B)}$ is just α_P.

   The flattening of $P; (\Delta \vdash B)$ is $\langle \iota_*(\Gamma_i \vdash J_i) \mid i \in I \rangle + \langle (\iota_*\Delta) \vdash J \rangle$, where ι is the inclusion $\alpha_P \to \alpha_{P;(\Delta\vdash B)}$, and J is ι_*B with the head, if any, filled by the expression $\mathsf{meta}_\star(\langle \mathsf{var}_i \rangle_{i \in \delta})$, where $\star$ is the new argument adjoined to the arity.

Notationally, we will not distinguish the flattening from the sequential premise-family itself.

Definition 6.8. A ***sequential rule*** $P \implies J$ over Σ and T is a sequential premise-family P, together with a judgement $\Gamma \vdash J$ over $\Sigma + \alpha_P$, the ***conclusion***, whose presuppositions are derivable over $T + P$ (with T translated to the metavariable extension). A sequential rule has an evident flattening as a raw rule.

Again, we do not notate the flattening explicitly. Note that as in the definition of raw rules the conclusion J has an empty context.

The addition of a boundary in the extension step of Definition 6.7 is precisely analogous to the traditional context extension rule, as in Definition 6.4. There, the extension is *specified* just by a type A, but its *effect* is to add a term-of-type judgement $\mathsf{var}_i : A$, where the term is automatically determined to be a (fresh) variable, rather than specified as input to the extension.

Reading Definition 6.7 with an eye towards computer-formalisation, one may note it can be formalized in several ways: as an inductive-recursive definition of a set with functions to arities and families of judgements; as an inductive family of sets indexed over pairs of an arity and a family of judgements; or an $\mathbb{N}$-indexed sequence of sets together with functions to arities and families, by induction on $n \in \mathbb{N}$, the length of the family. These are all equivalent, by standard generalities about inductive definitions.

Proposition 6.9. *The flattening of a sequential rule with empty conclusion context is acceptable.*

Proof. Tightness is immediate, by construction of the arity of the premise-family and the heads of its object-judgement premises. Presuppositivity is similarly by construction, from the well-formedness conditions in the definitions of sequential rules and premise-families, with the latter inductively translated along metavariable extensions as the premise-family is built up. □

As we defined premise-families using just boundaries rather than complete judgements, similarly when we define well-founded type theories we will specify them using sequential rules whose conclusions have no heads. We will also (for substitutivity) restrict attention to empty conclusion contexts.

Definition 6.10. A *sequential rule-boundary* $P \implies B$ over Σ and T is a sequential premise-family P, together with a boundary $\Gamma \vdash B$ over $\Sigma + \alpha_P$ with empty context, whose presuppositions are derivable over $T + P$.

Rule-boundaries can of course be completed to rules, by filling in a head if required.

Definition 6.11. The *realisation* of a sequential rule-boundary $R = (P \implies B)$ as a sequential rule (or, via flattening, a raw rule) is defined according to the form of B:

1. If B is an object boundary of class c, then given a symbol $\mathsf{S} \in \Sigma$ with arity α_P and class c, the *realisation $R[\mathsf{S}]$ of R with* S is the sequential rule

$$P \implies B[\widehat{S}]$$

   given by completing B with the generic application of S.

2. If B is an equality, no further input is required: the *realisation of* R is just $P \implies B$ with B viewed as a judgement.

This gives, by construction:

Proposition 6.12. *The realisation of an object rule-boundary for* S *yields a symbol rule for* S. $\qquad\square$

Example 6.13. The sequential rule-boundary

$$(\vdash \mathsf{A}\ \mathsf{type}); (x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}) \implies (\vdash \square\ \mathsf{type})$$

realised with the symbol Π gives the sequential rule

$$(\vdash \mathsf{A}\ \mathsf{type}); (x{:}\mathsf{A} \vdash \mathsf{B}(x)\ \mathsf{type}) \implies (\vdash \Pi(\mathsf{A}, \mathsf{B}(x))\ \mathsf{type})$$

whose flattening is the usual formation rule for dependent products, as in Example 4.35. With Σ instead of Π, it gives the formation rule for dependent sums.

6.3 Well-presented rules

Sequential rules and rule-boundaries give a satisfactory treatment covering most example theories, and sufficing for many purposes, including implementation in proof assistants. For instance, the Andromeda proof assistant (Andromedans, 2020; Bauer et al., 2019) implements a variant of sequential rules and rule boundaries in the trusted nucleus.

Here we consider the generalisation from finite sequences to arbitrary well-founded orders, partly to encompass infinitary rules, but mainly as a warm-up for well-founded theories.

Definitions 6.14, 6.15 and 6.18 given below are rather long and pedantic, so we give first a guiding overview. We follow the pattern first seen in Definition 6.2, where the components of the definitions must be stratified into several stages, with each stage making use of functions defined on earlier stages.

1. At the first stage, we can specify just the *shape* of the family of premises. This consists of a well-ordered set $(P, <)$, to index the premises, along with for each $i \in P$, the judgement form φ_i and scope γ_i for the i-th premise.

   Form this data we can compute the arity of the rule, α_P, and more generally the arity $\alpha_{P_{<i}}$, specifying what metavariables may occur in the i-th premise.

2. At the second stage, with the arities $\alpha_{P_{<i}}$ available, we can specify the *raw syntax* of the premises. The i-th premise P_i is given by a boundary $\Gamma_i \vdash B_i$ of form φ_i, scope γ_i, and written over $\Sigma + \alpha_{P_{<i}}$.

   From these, filling in heads of object premises as required for tightness, we can construct the flattening of P as family of judgements over $\Sigma + \alpha_P$, and more generally the flattening of $P_{<i}$ over $\Sigma + \alpha_{P_{<i}}$.

3. At the third stage, with the flattenings available, we can now specify the *well-formedness* conditions. Derivations of presuppositions of P_i should be given over the ambient theory T, translated up to $\Sigma + \alpha_{P_{<i}}$, and with (the flattenings of) preceding premises $P_{<i}$ available as hypotheses.

4. We are now done with the hard part. Having specified the premises, the conclusion is given as in the sequential case, as a well-formed boundary B over $\Sigma + \alpha_P$, whose head (if B is of object form) will later be filled in to yield a symbol rule.

While the above explanation sounds plausible, it sweeps several technical subtleties under the rug. Most importantly, since each premise is specified over its own signature $\Sigma + \alpha_{P_{<i}}$, we need to handle the translations between these extensions, and up to the overall extension $\Sigma + \alpha_P$. Spelling out all details in full, we have the following definitions.

Definition 6.14. A *well-founded premises-shape* (I, S) is given by a well-founded set $(I, <)$, and a family $S = \langle (\varphi_i, \gamma_i) \rangle_{i \in I}$, where φ_i is a judgement form and γ_i a scope. Given these, we define:

1. The arity α_S of S is the subfamily $\langle (\varphi_i, \gamma_i) \rangle_{\{i \in I \mid \varphi_i \in \{\mathsf{ty}, \mathsf{tm}\}\}}$ of the object forms of S.

2. For each $i \in I$, the initial segment $S_{<i} := \langle (\varphi_j, \gamma_j) \rangle_{j < i}$ is itself a well-founded premises-shape indexed by the initial segment $\downarrow i \subseteq I$, and hence it also has an associated arity $\alpha_{S_{<i}}$.

3. For each $i < j \in I$, there are evident family maps $\alpha_{S_{<i}} \rightarrow \alpha_{S_{<j}}$ and hence $S_{<i} \rightarrow S_{<j}$, satisfying evident composition conditions with each other and with the subfamily inclusions $S_{<j} \rightarrow S$.

Definition 6.15. Given a signature Σ and a well-founded premises-shape (I, S) as in Definition 6.14, a *well-founded premise-family* P is given by a family $B = \langle B_i \rangle_{i \in I}$ where B_i is a boundary of form φ_i in scope γ_i, and over $\Sigma + \alpha_{S_{<i}}$. Given these, we define:

1. The *flattening* $P^\flat$ is the family of judgements $\langle P_i \rangle_{i \in I}$ over $\Sigma + \alpha_S$, where P_i is the boundary B_i translated along the inclusion $\Sigma + \alpha_{S_{<i}} \rightarrow \Sigma + \alpha_S$, and when $\varphi_i \in \{\mathsf{ty}, \mathsf{tm}\}$ completed with the head expression $\mathsf{meta}_i(\langle \mathsf{var}_j \rangle_{j \in \gamma_i})$.

2. For each $i \in I$, the initial segment $B_{<i}$ yields a well-founded premise family $P_{<i}$ with respect to the well-founded premises-shape $S_{<i}$ indexed by the initial segment $\downarrow i$. Thus it has its own flattening $P^\flat_{<i}$.

48

3. For each $j < i$, the judgement P_j as a member of the flattening $P^\flat_{<i}$ translated along the signature inclusion $\Sigma + \alpha_{S_{<i}} \to \Sigma + \alpha_S$ yields the same flattening P_j, but as a member of $P^\flat$.

   This exhibits the translation of $P^\flat_{<i}$ along the inclusion $\Sigma + \alpha_{S_{<i}} \to \Sigma + \alpha_S$ as a subfamily of $P^\flat$.

4. Similarly, for all $k < j < i$, the judgement P_k as a member of the flattening $P^\flat_{<j}$ translated along the signature inclusion $\Sigma + \alpha_{S_{<j}} \to \Sigma + \alpha_{S_{<i}}$ yields the same flattening P_k, but as a member of $P^\flat_{<i}$.

   This exhibits the translation of $P^\flat_{<j}$ along the inclusion $\Sigma + \alpha_{S_{<j}} \to \Sigma + \alpha_{S_{<i}}$ as a subfamily of $P^\flat_{<i}$.

Definition 6.16. A well-founded premise-family P as in Definition 6.15 is *well-formed over T* if for each $i \in I$, there are derivations of all presuppositions of B_i from hypotheses $P^\flat_{<i}$, in the translation of T to $\Sigma + \alpha_{S_{<i}}$.

   A ***well-presented premise-family*** is a well-formed well-founded premise-family P. Its arity α_P is the associated arity α_S of the underlying premises-shape S.

When no confusion can occur, we will write the flattening of a well-founded premise-family P just as P, rather than $P^\flat$.

Definition 6.17. A ***well-presented rule-boundary*** $P \implies B$ over Σ, T consists of a well-presented premise-family P together with a boundary with empty context $\vdash B$ over $\Sigma + \alpha_P$, the ***conclusion boundary***, such that all presuppositions of B derivable from P in the translation of T to $\Sigma + \alpha_P$. The ***arity*** of such a rule-boundary is the arity α_P of its premise-family.

Definition 6.18. The ***realisation*** of a well-presented rule-boundary $R = (P \implies B)$ as a raw rule is defined according to the form of B.

1. If B is an object boundary of class c, then given a symbol $\mathsf{S} \in \Sigma$ of arity α_P and class c, the ***realisation $R[\mathsf{S}]$ of R with S*** has premises the flattening of P, and conclusion $B[\widehat{S}]$.

2. If B is an equality boundary, no extra input is required: the ***realisation of R*** has premises the flattening of P, and conclusion just B viewed as an equality judgement.

6.4 Well-presented type theories

Finally, we reach well-foundedness for type theories. Once again, a by now familiar pattern emerges. It is fairly straightforward to define well-foundedness as an after-market property of acceptable type theories, but a better definition is obtained by putting in a little more work.

   We start with the simpler version.

Definition 6.19. Let $T = \langle R_i \rangle_{i \in I}$ be an acceptable type theory over a signature Σ, and let $\beta : |\Sigma| \to I$ the bijection from symbols to their rules. Then T is ***well-founded*** when all its rules are well-founded, and the index set I has a well-founded order $<$, such that:

1. If $\mathsf{S} \in \Sigma$ appears in R_i then $\beta(\mathsf{S}) < i$.

2. Each R_j has derivations of presuppositions that only refer to symbols S with $\beta(\mathsf{S}) < j$ and rules R_i with $i < j$.

For the more refined version, we follow a similar pattern to what we saw for well-presented rules in Section 6.3, with the definition stratified into three stages:

1. First, the *shape*: a well-founded order (to index the rules), and the premises-shapes and judgement forms of all rules. This suffices to compute the signature of the theory, and of its initial segments.

2. Next, the *raw* part: for each rule of the theory, a well-founded premise-family, and (raw) conclusion boundaries, of the shapes and forms specified in the first stage, and over the signature of the appropriate initial segment. These suffice to compute the flattening of the theory as a raw type theory, and of its initial segments.

3. Finally, the *derivations* showing well-formedness of each rule over the preceding initial segment.

Having previously given Definitions 6.14, 6.15 and 6.16 in rather excruciating detail, we proceed here slightly more concisely, trusting the reader to be able to fill in the elided details along the lines spelled out in those definitions.

Definition 6.20.

1. A ***well-founded type theory shape*** T consists of a well-founded order $(I, <)$, together with for each $i \in I$, a well-founded premises-shape S_i and judgement form φ_i (seen as the premises shape and conclusion form of the ith rule).

   From these, we can define the total signature Σ_T of T: its symbols are just $\{\mathsf{S}_i \in I \mid \varphi_i \in \{\mathsf{ty}, \mathsf{tm}\}\}$, with S_i having arity α_{S_i} and class φ_i. Similarly, we get signatures for initial segments $\Sigma_{T_{<i}}$, and signature maps between these, and from these to Σ_T.

2. A ***well-founded raw type theory*** T consists of a well-founded type theory shape as above (which we also call T), together with for each $i \in I$, a well-founded premise-family P_i of shape S_i and a boundary B_i of form φ_i over the signature $\Sigma_{T_{<i}}$.

   From these, we can define the flattening $T^\flat$ of T as a raw type theory over Σ_T. Its rules consist of the realisations of all rule-boundaries $P_i \implies B_i$, using (when i is of object form) the symbol S_i, together with the associated congruence rules of the object rules thus added. Similarly, we obtain the flattening $T^\flat_{<i}$ of each initial segments of T, as a raw type theory over $\Sigma_{T_{<i}}$.

3. A ***well-presented type theory*** T consists of a well-founded raw type theory T as above that is additionally ***well-formed***, in that it is equipped with, for each i, derivations exhibiting $P_i \implies B_i$ as a well-formed rule-boundary over $T^\flat_{<i}$.

As with well-presented rules, we will not notate flattening, when there is no ambiguity.

Proposition 6.21. *The flattening of a well-presented type theory T is acceptable and well-founded.*

Proof. Well-foundedness, tightness, substitutivity, and congruousness are immediate by construction. Presuppositivity is almost as direct, requiring just translation of well-formedness of rules from the signatures $\Sigma_{T_{<i}}$ and theories $T_{<i}$ up to the full signature Σ and theory T. $\qquad\square$

6.5 The well-founded replacement

Definition 5.12 of acceptable type theories allows cyclic references of three kinds: between types of a context, premises of a rule, or rules of a type theory. We shall not concern ourselves with the former two, since type theories occurring in practice all avoid them by using sequential contexts and sequential rules from Sections 6.1 and 6.2. We address the latter one, to vindicate our design choices from earlier sections, to demonstrate that our setup supports non-trivial meta-theoretic methods, and to give an interesting new construction that likely has further applications.

For the remainder of this section, all contexts, raw rules, and rule-boundaries are presumed to be sequential. Also, it will be convenient to speak of a raw theory T without explicitly displaying its underlying signature. When we need to refer to the signature, we do so by writing Σ_T.

In Example 5.14 an acceptable type theory was rectified to a well-founded one by the introduction of a new symbol and an equation. When one looks at other specific examples the same strategy works, possibly with the introduction of several symbols and equations. In order to present a general method we first lay some category-theoretic groundwork. We save the adventure of spiralling into the depths of category theory for another day, and instead establish just enough structure to keep the syntactic constructions organized.

Definition 6.22. A ***raw syntax map*** $f : \Sigma \to \Sigma'$ is given by a family of expressions $f_S \in \mathrm{Expr}^{c_S}_{\Sigma'+\mathrm{arg}\,S}(0)$, one for each $S \in \Sigma$. Such a map acts on $e \in \mathrm{Expr}^c_\Sigma(\gamma)$ to give $f_* e \in \mathrm{Expr}^c_{\Sigma'}(\gamma)$ by

$$f_*(\mathsf{var}_i) := \mathsf{var}_i \qquad \text{and} \qquad f_*(S(e)) := (f_* \circ e)_* f_S. \tag{6.1}$$

The ***metavariable extension*** $f + \alpha : \Sigma + \alpha \to \Sigma' + \alpha$ by arity α is the raw syntax map defined by

$$(f + \alpha)_S := f_S \qquad \text{and} \qquad (f + \alpha)_{\mathsf{meta}_i} := \mathsf{meta}_i(\langle \mathsf{var}_j \rangle_{j \in \mathrm{bind}_\alpha\, i})$$

In words, a raw syntax map $\Sigma \to \Sigma'$ interprets each symbol in Σ as a suitable compound expression over Σ', the interpretation extends compositionally to all expressions over Σ, and metavariable extensions act on such a map by extending it trivially.

Let us unravel the second clause in (6.1), as it is a bit terse. Given a symbol $S \in \Sigma$ and its arguments $e \in \prod_{i \in \mathrm{arg}\,S} \mathrm{Expr}^{\mathrm{cl}_S\, i}_\Sigma(\gamma \oplus \mathrm{bind}_S\, i)$, the composition $f_* \circ e$ takes each $i \in \mathrm{arg}\,S$ to $f_* e_i \in \mathrm{Expr}^{\mathrm{cl}_S\, i}_{\Sigma'}(\gamma \oplus \mathrm{bind}_S\, i)$ – it is an instantiation of arity $\mathrm{arg}\,S$, which thus acts on f_S to yield an expression $(f_* \circ e)_* f_S \in \mathrm{Expr}^{c_S}_{\Sigma'}(\gamma)$, where we took into account that $\gamma \oplus 0 = \gamma$.

The action of a raw syntax map evidently extends from expressions to context, judgements, and boundaries, and thanks to the metavariable extensions also to raw rules and rule-boundaries.

Proposition 6.23. *Signatures and raw syntax maps form a category:*

- *The identity morphism* $\mathrm{id}_\Sigma : \Sigma \to \Sigma$ *takes* $S \in \Sigma$ *to the generic application* $\widehat{S}$.

- *The composition of* $f : \Sigma \to \Sigma'$ *and* $g : \Sigma' \to \Sigma''$ *is the map* $g \circ f : \Sigma \to \Sigma''$ *that interprets each* $S \in \Sigma$ *as* $(g \circ f)_S := g_* f_S$.

Raw syntax map actions are functorial.

Proof. A straightforward application of the basic properties of instantiations. □

Given raw theories T and T', a raw syntax map $f : \Sigma_T \to \Sigma'_T$ between the underlying signatures may be entirely unrelated to T and T'. Requiring it to map derivable judgements in T to derivable judgements in T' helps, but ignores the fact that raw theories are families of raw rules, not derivable judgements. Here is a better definition.

Definition 6.24. A *raw theory map* $f : T \to T'$ is a raw syntax map $f : \Sigma_T \to \Sigma'_T$ on the underlying signatures which maps each specific rule R of T to a derivation of $f_* R$ in T'.

Proposition 6.25. *Raw theories and raw theory maps form a category* RawTh. *A raw theory map* $f : T \to T'$ *acts functorially on a derivation* D *of* $\Gamma \vdash J$ *in* T *to give a derivation* $f_* D$ *of* $f_* \Gamma \vdash f_* J$ *in* T'.

Proof. Let us first describe the action of f on a derivation D of $\Gamma \vdash J$. We proceed by recursion on the structure of D.

Structural rules are mapped to the corresponding structural rules, e.g., if D concludes with a variable rule $\Gamma \vdash \mathsf{var}_i : \Gamma_i$ then $f_* D$ concludes with the variable rule $f_* \Gamma \vdash \mathsf{var}_i : f_* \Gamma_i$, and similarly for other structural rules.

Consider the case where D ends with an instantiation $I_* R$ of a specific rule R of T, whose premises are $\langle \Gamma_k \vdash J_k \rangle_{k \in K}$. Suppose f takes R to the derivation D_R of $f_* R$ in T'. First we recursively map each derivation D_k of the k-th premise $I_* \Gamma_k \vdash I_* J_k$ to a derivation $f_* D_k$ of $f_* (I_* \Gamma_k \vdash I_* J_k)$ in T', which is the same as $(f_* I)_* \Gamma_k \vdash (f_* I)_* J_k$, because acting by I and then by f is the same as acting by the instantiation $f_* I$. We then take $f_* D$ to be the derivation $(f_* I)_* R_D$ with the derivations $f_* D_k$ of its hypotheses grafted onto it, as in Definition 2.10.

It is straightforward to verify that the action on derivations so constructed satisfies functoriality, $(g \circ f)_* D = g_* (f_* D)$.

The categorical structure of RawTh is inherited from the structure of raw syntax maps. Additionally, given composable raw theory maps f and g, we let their composition $g \circ f$ take a specific rule R to the derivation $g_* D_R$, where D_R is the derivation of $f_* R$ provided by f. $\square$

It may happen that a raw theory map $f : T \to T'$ maps an uninhabited type to an inhabited one, or an underivable equality to a derivable one. Let us make precise the sense in which such a map fails to be conservative, and at the same time generalise inhabitation of types to general completion of boundaries in the presence of premises.

Definition 6.26. Given a raw theory T, and an object rule-boundary $P \implies B$ over Σ_T whose syntactic class is c_B, say that $e \in \mathrm{Expr}^{c_B}_{\Sigma_T + \alpha_P}(0)$ *realises* the rule-boundary when $P \implies B[e]$ is derivable in T.

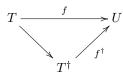**Example 6.27.** Inhabitation of a closed type A corresponds to realisation of the rule-boundary $\langle \rangle \implies \square : A$. We can also express more general inhabitation tasks, for instance $(\vdash A\ \mathsf{type}) \implies \square\ \mathsf{type}$ asks for a construction of a type from a type parameter A, and $(\vdash A\ \mathsf{type}); ([\mathsf{var}_0 : A] \vdash B(\mathsf{var}_0)\ \mathsf{type}) \implies \square\ \mathsf{type}$ for a Π-like higher type constructor.

Definition 6.28. A raw theory map $f : T \to T'$ is *conservative* when:

1. f reflects equations: if T' derives the equational rule $f_* R$ then T derives the equational rule R, and

2. f reflects realisers: there is a map r such that if e realises the object rule-boundary $f_* (P \implies B)$ in T' then $r(R, e)$ realises $P \implies B$ in T.
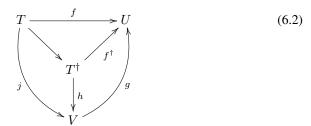
In the definition, we asked for a map r to witness reflection of realisers in order to avoid spurious applications of the axiom of choice.

Lemma 6.29. *Every raw theory map $f : T \to U$ factors through a conservative map $f^\dagger : T^\dagger \to U$*

$$
\begin{array}{ccc}
T & \xrightarrow{\quad f \quad} & U \\
 & \searrow \qquad \nearrow f^\dagger & \\
 & T^\dagger &
\end{array}
$$

in a weakly universal fashion.

In the lemma, by weak universality we mean that whenever f factors through a conservative map $g : V \to U$, there is a map $h : T^\dagger \to V$, not necessarily unique, such that the following diagram commutes:

$$
\begin{array}{ccc}
T & \xrightarrow{\quad f \quad} & U \\
\end{array}
\qquad (6.2)
$$

with maps j, h, g, $f^\dagger$ through $T^\dagger$ and V.

Proof of Lemma 6.29. We shall construct $T^\dagger$ by adjoining new symbols to T, so that whenever e realises $f_* R$ in U, the corresponding new symbol realises R in T. However, the new symbols generate new rule-boundaries, so the process needs to be iterated, and we have to adjoin equations as well. The construction of $f^\dagger : T^\dagger \to U$ thus proceeds in an inductive fashion, as follows.

Initially the signature $\Sigma_{T^\dagger}$ is just Σ_T, the specific rules of $T^\dagger$ are those of T, and $f^\dagger$ acts like f. We inductively extend $\Sigma_{T^\dagger}$ with new symbols, $T^\dagger$ with new specific rules, and $f^\dagger$ with new values, as follows:

1. If $P \Longrightarrow B$ is an object rule-boundary in $T^\dagger$ of arity α and e realises $f_*^\dagger P \Longrightarrow f_*^\dagger B$ in U, then we extend $\Sigma_{T^\dagger}$ with a ***new symbol*** $\mathsf{c}_{(P \Longrightarrow B, e)}$ of arity α, and $T^\dagger$ with the associated symbol rule $P \Longrightarrow B[\widehat{\mathsf{c}}_{(P \Longrightarrow B, e)}]$. We extend $f^\dagger$ by letting it map $\mathsf{c}_{(P \Longrightarrow B, e)}$ to e.

2. If R is an equational rule in $T^\dagger$ such that $f_*^\dagger R$ is derivable in U, we extend $T^\dagger$ with R as a specific rule.

Because we assumed all contexts and premise-families to be sequential, the inductive definition is complete after countably many repetitions of the above process. Alternatively, $T^\dagger$ could be constructed as a suitable colimit.

The map f obviously factors as $f = f^\dagger \circ i$ where $i : T \to T^\dagger$ is induced by the inclusion $\Sigma_T \to \Sigma_{T^\dagger}$.

The map $f^\dagger$ is conservative by construction. It obviously reflects equations, while an object rule-boundary R in $T^\dagger$, such that e realises $f_*^\dagger R$ in U, is realised by $\mathsf{c}_{(R,e)}$.

It remains to be shown that the factorization is weakly universal. Consider a factorization $f = g \circ j$ through a conservative map $g : V \to U$, as in (6.2). There exists a map r, not necessarily unique, that witnesses conservativity of g. The desired factor $h : T^\dagger \to V$ is defined inductively: it acts like j on symbols of Σ_T, and takes $\mathsf{c}_{(R,e)}$ to $r(h_* R, e)$. $\qquad \square$

Corollary 6.30. *A raw theory T has a weakly universal conservative map $t : T^{\mathrm{wf}} \to T$ where T^{wf} is well-founded.*

Proof. We take $T^{\mathrm{wf}} = \langle\rangle^\dagger$ and $t = o^\dagger$, as in Lemma 6.29, where $o : \langle\rangle \to T$ is the unique map from the empty theory $\langle\rangle$. Weak universality is immediate, and well-foundedness of $\langle\rangle^\dagger$ is witnessed by the inductive nature of its construction. $\qquad\square$

Definition 6.31. The map $t : T^{\mathrm{wf}} \to T$ from Corollary 6.30 is called the ***well-founded replacement of*** T.

Here, finally is the main theorem of this section.

Theorem 6.32. *If T is acceptable, the map $t : T^{\mathrm{wf}} \to T$ has a section.*

The theorem establishes a form of equivalence of T^{wf} and T, because any conservative map $r : U \to V$ with a section $s : V \to U$ is an isomorphism up to judgmental equality. Indeed, $r \circ s = \mathrm{id}_V$ because s is a section of r, while $s \circ r$ is identity up to judgemental equality: given any derivable rule $P \implies A$ type in U, the rule $r_* P \implies r_*(s_*(r_* A)) \equiv r_* A$ is derivable in V by reflexivity, and hence $P \implies s_*(r_* A) \equiv A$ in U by conservativity of r. An analogous argument works for term judgements.

Proof of Theorem 6.32. The theory T^{wf} has symbols of the form $\mathsf{c}_{(P \implies B, e)}$, where B is a closed boundary, but in the proof we will have to deal with boundaries that refer to variables.

For this purpose, define the *(variable-to-metavariable) promotion* of an expression $e \in \mathrm{Expr}_\Sigma^{\mathrm{c}}(\gamma)$ to be the expression $\widetilde{e} \in \mathrm{Expr}_{\Sigma + \mathrm{simp}\,\gamma}^{\mathrm{c}}(0)$, cf. Definition 3.21, which is e with the variables replaced by metavariables,

$$\widetilde{\mathsf{var}_i} := \mathsf{meta}_i, \qquad \text{and} \qquad \widetilde{S(e)} := S(\langle \widetilde{e_i} \rangle_{i \in \mathrm{arg}\,S}).$$

The associated *demotion* is the instantiation $D_\gamma \in \mathrm{Inst}_{\Sigma, \gamma}(\mathrm{simp}\,\gamma)$ which takes the metavariables back to variables, $D_\gamma(\mathsf{meta}_i) = \mathsf{var}_i$. Thus we have $e = D_{\gamma_*}\widetilde{e}$.

One level up, given a premise-family P and a context Γ over $\Sigma + \alpha_P$, the *promotion* of Γ is the extension $P; \widetilde{\Gamma}$ of P in which the variables of Γ are promoted to metavariables of suitable types,

$$(P; \widetilde{[\,]}) := P \qquad \text{and} \qquad (P; \widetilde{\Gamma . A}) := (P; \widetilde{\Gamma}); (\vdash \widetilde{A}\ \mathsf{type}).$$

We begin the construction of a section of t by defining a map d which maps sequential rules and rule-boundaries from T to T^{wf} by replacing compound expressions e with suitable symbols $\mathsf{c}_{(R, e)}$ from T^{wf}. When acting on contexts and judgements, d takes a sequential rule-family P from T as an additional parameter, in which case we write d_P.

The map d recurses over a premise-family in T to give a premise-family in T^{wf}:

$$d(\langle\rangle) := \langle\rangle \qquad \text{and} \qquad d(P; (\Gamma \vdash J)) := (d(P); d_P(\Gamma \vdash J)).$$

Similarly, it takes a sequential context Γ over $T + P$ to one over $T^{\mathrm{wf}} + d(P)$:

$$d_P([\,]) := [\,],$$
$$d_P(\Gamma . A) := d_P(\Gamma) . \left(D_{|\Gamma|_*}\widehat{\mathsf{c}}_{((d(P); \widetilde{d_P(\Gamma)}) \implies \square\ \mathsf{type}), \widetilde{A})}\right).$$

54

In the second clause, d_P recurses into Γ and extends it with the rather intimidating

$$D_{|\Gamma|_*}\widehat{\mathsf{c}}_{((d(P);\widetilde{d_P(\Gamma)})\Longrightarrow\square\ \mathsf{type}),\widetilde{A})},$$

which is just the generic application of the c-symbol for the promoted $\widetilde{A}$, demoted back to Γ. Note that $(d(P);\widetilde{d_P(\Gamma)}) = d(P;\widetilde{\Gamma})$ and hence $t_*(d_P(\Gamma\mathrel{.}A)) = \Gamma\mathrel{.}A$.

It remains to explain how d_P maps a judgement $\Gamma\vdash J$ over $T+P$ to one over $T^{\mathrm{wf}}+d(P)$. Here too we use the same method of demoting a generic application of a symbol for a promoted expression:

$$\begin{aligned}
d_P(\Gamma\vdash A\ \mathsf{type}) &\coloneqq (\Gamma'\vdash A'\ \mathsf{type}),\\
d_P(\Gamma\vdash t:A) &\coloneqq (\Gamma'\vdash t':A'),\\
d_P(\Gamma\vdash A\equiv B) &\coloneqq (\Gamma'\vdash A'\equiv B'),\\
d_P(\Gamma\vdash s\equiv t:A) &\coloneqq (\Gamma'\vdash s'\equiv t':A'),
\end{aligned}$$

where

$$\begin{aligned}
\Gamma' &\coloneqq d_P(\Gamma),\\
A' &\coloneqq D_{|\Gamma|_*}\widehat{\mathsf{c}}_{((d(P);\widetilde{\Gamma'}\Longrightarrow\square\ \mathsf{type}),\widetilde{A})},\\
B' &\coloneqq D_{|\Gamma|_*}\widehat{\mathsf{c}}_{((d(P);\widetilde{\Gamma'}\Longrightarrow\square\ \mathsf{type}),\widetilde{B})},\\
t' &\coloneqq D_{|\Gamma|_*}\widehat{\mathsf{c}}_{((d(P);\widetilde{\Gamma'}\Longrightarrow\square:\widetilde{A'}),\widetilde{t})},\\
s' &\coloneqq D_{|\Gamma|_*}\widehat{\mathsf{c}}_{((d(P);\widetilde{\Gamma'}\Longrightarrow\square:\widetilde{A'}),\widetilde{s})}.
\end{aligned}$$

By having d map $\square$ to $\square$ the above clauses also provide the action of d on boundaries. Finally, let d map a sequential rule $P\Longrightarrow J$ in T to the sequential rule $d(P)\Longrightarrow J'$ where $d_P(\vdash J) = (\vdash J')$, and similarly for rule-boundaries.

We have arranged d in such a way that $t_*(d(R)) = R$ for any sequential rule R over T. Moreover, if R is derivable in T, then $d(R)$ is derivable in T^{wf}, by an appeal to suitable symbol rules in T^{wf}. For instance, d maps the rule $P\Longrightarrow A$ type to the rule $d(P)\Longrightarrow\widehat{\mathsf{c}}_{((d(P)\Longrightarrow\square\ \mathsf{type}),A)}$ type. If the former is derivable in T then the latter is a symbol rule of T^{wf}.

At last, let us define the section s of t. Consider first a type symbol $S\in\Sigma_T$. Because T is acceptable, it has a unique symbol rule $P\Longrightarrow\widehat{S}$ type. When we map it with d we get

$$d(P)\Longrightarrow\widehat{\mathsf{c}}_{((d(P)\Longrightarrow\square\ \mathsf{type}),\widehat{S})}\ \mathsf{type},$$

which is a symbol rule in T^{wf}. We may therefore take $s_S \coloneqq \widehat{\mathsf{c}}_{((d(P)\Longrightarrow\square\ \mathsf{type}),\widehat{S})}$.

A term symbol $S\in\Sigma_T$ is dealt with analogously. Its symbol rule takes the form $P\Longrightarrow\widehat{S}:A$, which is mapped by d to

$$d(P)\Longrightarrow\widehat{\mathsf{c}}_{((d(P)\Longrightarrow\square:A'),\widehat{S})}:A',$$

where $A' = \widehat{\mathsf{c}}_{((d(P)\Longrightarrow\square\ \mathsf{type}),A)}$, Again, this is a symbol rule in T^{wf}, so we may define $s_S \coloneqq \widehat{\mathsf{c}}_{((d(P)\Longrightarrow\square:A'),\widehat{S})}.$ $\qquad\square$

Example 6.33. We revisit Example 5.14, the type theory expressing type-in-type in a cyclic fashion as $\vdash \mathsf{u}:\mathsf{El}(\mathsf{u})$. Earlier we pointed out that the theory can be made well-founded by using the defined type constant $\vdash\mathsf{U}\equiv\mathsf{El}(\mathsf{u})$. The well-founded replacement works much the same way, except that it is replete with many more defined symbols. The analogue of U appears already at the first stage of the construction.

Indeed, the rule-boundary $\langle\,\rangle \Longrightarrow \Box$ type is realised by $\mathsf{El}(\mathsf{u})$, hence the well-founded replacement contains the type constant $U = \mathsf{c}_{(\langle\,\rangle \Longrightarrow \Box\ \mathsf{type},\mathsf{El}(\mathsf{u}))}$. We also have the new symbols

$$El = \mathsf{c}_{((\vdash \mathsf{a}:U) \Longrightarrow \Box\ \mathsf{type},\widehat{\mathsf{El}})} \qquad \text{and} \qquad u = \mathsf{c}_{(\Longrightarrow \Box:U,\widehat{\mathsf{u}})},$$

and these suffice to express the type equation $\vdash U \equiv El(u)$, which is a specific rule of the well-founded replacement because it is mapped to a valid equation in the original type theory.

7 Discussion and related work

We set out to give a detailed and general mathematical definition of dependent type theories, accomplished by an analysis of their traditional accounts. Having completed the task, let us take stock of what has been accomplished.

We calibrated abstraction at the level that keeps a connection with concrete syntax, but also clearly identifies the category-theoretic structure underlying the abstract syntax. As such, our work may serve as a theoretical grounding and a guideline for practical implementations of type theories on one hand, and on the other as a ladder to be climbed and discarded by those who wish to ascend to higher, more abstract viewpoints of dependent type theories.

There is not much to remark on our treatment of raw syntax, as the topic has been studied before and is well understood, except to remark that scope systems have served us well as a general approach to scoping and binding of variables.

More interesting are the subsequent stages of our definition. Giving the definition in generality forces us to isolate and precisely define various notions that are traditionally treated only informally, and often only passed on in folklore rather than in writing. For instance, we articulate precisely the distinction between the syntactic specification of an inference rule, and the scheme of closure conditions that it begets — a distinction which, once seen, was implicitly present all along, but which is not generally appreciated or consciously articulated.

We initially considered raw theories as just a stepping stone towards the definition of well-presented type theories, but have come to feel that they are of significant intrinsic interest. Their simplicity makes them easy to work with, and even though they permit deviations from the orthodox teachings, they boast with a surprisingly rich collection of meta-theorems.

The well-behavedness properties of raw rules and raw type theories, such as presuppositivity, tightness, and congruity, took some effort to define and explain, but quickly paid off. On a technical level, they allowed us to fine-tune the requirements that enable the various meta-theorems. More importantly, once we incorporated them into our type-theoretic vocabulary they streamlined communication and invigorated the mind where there used to be just an uneasy adherence to heuristic techniques.

We hope that our selection of meta-theorems is illustrative enough to inspire further generally applicable meta-theorems, and comprehensive enough to relieve future designers of type theories from having to redo the work. We have intentionally restrained any category-theoretic analysis of the landscape we explore, but it will be visible in the background to many readers, demanding future exploration of the categorical structure of the syntactic notions, both for its own sake and to connect with a general categorical semantics. This, of course, we hope to return to in future work.

How widely does our definition of type theories cast its net? It takes little effort to enumerate many examples of interest, such as variants of Martin-Löf type theory (Martin-Löf, 1972), in both its intensional and extensional incarnations, homotopy type theory (Univalent Foundations Program, 2013), simple type theory (Church, 1932), some presentations of the higher-order logic of toposes (Lambek and Scott, 1986), etc. But it is equally easy to list counter-examples: System F (Girard, 1972; Reynolds, 1974) and related type systems that directly quantify over all types, pure type systems (Barendregt, 1992), cubical type theories (Cohen et al., 2015), cohesive type theories (Schreiber and Shulman, 2014), etc. Can such a diverse collection of formalisms be unified under the umbrella of an even more general definition of type theories? Doubtlessly, our work can be pushed and stretched in various directions, and we hope it will. But we also state again that we do not intend our definitions to be definitive or prescriptive, nor consider our methods to be superior to others. After all, type theory is an open-ended idea.

Several years ago Vladimir Voevodsky's relentless inquiries into the precise mathematical underpinnings of type theories motivated us to undertake the study of general type theories. We were hardly alone to do so. Voevodsky himself initially worked to develop the framework of B-systems and C-systems (Voevodsky, 2014, 2016), but these will remain tragically unfinished. There is by now a spectrum of various approaches to the meta-theory of type theories, which cannot be justly reviewed in the remaining space. We only mention a selection of contemporaneous developments and their relation to our work.

Logical framework approaches   When discussing and presenting this work, we have often been asked why we bothered, when logical frameworks (LF) (Harper et al., 1993; Pfenning, 2001) already give a satisfactory definition of type theories.

The main answer is that most work with logical frameworks does not give a general definition in the same sense that we are looking for. It sets up a framework within which many type theories may be defined, but that is not quite the same thing. Indeed, at the point when we were first embarking on the present project, no definition in a generality close to our aims had been proposed in the LF literature (to our knowledge), nor could we see how to do so using LF methods. Since then, Uemura has succeeded in giving a very clean general definition using the LF; we discuss that work in detail below.

A significant secondary motivation for the present approach, though, was to directly recover the standard "naïve" presentation of particular type theories, in specific examples. This problem is, by design, something that LF-based approaches bypass entirely. One may argue — as some have — that this desire is misguided: that since LF-based approaches are so much cleaner, naïve syntax should be discarded as obsolete, and LF-embedded presentations of theories preferred as primitive. We however find that view somewhat unsatisfactory, for several reasons.

Firstly, even if we *should* be always reading type theories as LF presentations, we *have not* been. At least within the literature on constructive type theory in the tradition of Martin-Löf, most work still uses the naïve reading, including the work of Martin Hofmann (Hofmann, 1997) and others (Martin-Löf, 1972; Streicher, 1991; Univalent Foundations Program, 2013). Or rather, most of the literature stays silent about the issue, but where the intended reading is made clear, it tends to be the naïve one. Secondly, most work using LF approaches explicitly comments on the setup, and often gives or cites adequacy theorems. This seems to suggest that writers agree that the correctness of LF presentations of type theories rests, in part, on their connection to

the naïve presentations. Finally, it seems very difficult to adopt a position of *completely* discarding the naïve readings, and reading all presentations of type theories always as shorthands for their LF embedding. This is because the framework is itself a type theory, whose presentation must sooner or later be given the naïve reading, rather than as embedded in a further framework — it cannot be "turtles all the way".

We therefore believe it is important to have both the naïve and LF-based definitions of syntax defined and developed in as wide a generality as possible (as well as other approaches, such as those of categorical logic). The naïve approach is most natural and conceptually basic. The LF approach is cleaner and simpler to set up, and easier to analyse and apply for some purposes. Both should be available, and connected by an adequacy theorem, not just in special cases but in generality.

Uemura's general type theories   Another general definition of type theories has recently been given by Taichi Uemura. Indeed, (Uemura, 2019) provides two definitions, one semantic and one syntactic, and shows their correspondence with a general initiality theorem.

In terms of generality, Uemura's definition essentially subsumes ours and indeed generalises much further, encompassing type theories with different judgement forms, while still retaining enough structure to allow proofs of important type-theoretic metatheorems. It therefore quite satisfactorily solves one of the major goals we set out to solve with the present project.

On inspection, however, Uemura's approach is sufficiently different that it is complementary with our approach, rather than subsuming it. His syntactic definition is given via a particularly ingenious use of a logical framework; as with other LF-based approaches, this keeps the setup very clean, but means that in specific examples, it does not so closely recover the standard naïve reading of the theory in question.

It does not, therefore, address our secondary goal of taking seriously the naïve reading of syntax, and directly recovering it in examples. We therefore hope that it should be possible in the future to connect our syntactic definition with Uemura's by means of a generalised adequacy theorem, and show that for theories with Martin-Löf's original four judgment forms, the two approaches are equivalent.

Other general definitions of dependent type theories   Independently of our work, Guillaume Brunerie has proposed a general syntactic definition of dependent type theories (Brunerie, 2020a), and is formalising it in Agda (Brunerie, 2020b). His approach is very similar to ours, which we see as a welcome convergence of ideas.

Valery Isaev has also proposed a definition of dependent type theories, in (Isaev, 2017). His approach is semantic, avoiding syntax with binding, and defining dependent type theories as certain essentially algebraic theories extending the theory of categories with families. The generality of his definition seems to be roughly similar to ours, but a precise comparison seems slightly subtle to state, and is beyond the scope of the present paper.

Meta-theory of type theory in type theory   When one takes type theory seriously as a foundation of mathematics, it is natural and even imperative, to develop the metatheory of type theories in type theory itself. Whereas in the LF approach the expressivity of the ambient formalism is curbed to ensure an adequacy theorem, here we gladly trade adequacy for working in a full-fledged dependent type theory.

Such a project has been undertaken by Thorsten Altenkirch and Ambrus Kaposi (Altenkirch and Kaposi, 2016). Broadly speaking, a *specific* object type theory is constructed in one fell swoop as a quotient-inductive-inductive type (QIIT) that incorporates all judgement forms, the structural and the specific rules. The inductive character of the definition automatically provides the correct notion of derivation, while the ambient type theory guarantees that only derivable judgements can be constructed — the "raw" stage is completely side-stepped. The quotienting capabilities favourably relate the ambient propositional equality with the object-level judgmental equality. From a semantic point of view, the construction is the type-theoretic analogue of an initial-model construction. The ingenuity of the definition allows one to prove many meta-theorems quite effortlessly, especially with the aid of a proof assistant.

Our bottom-up approach can add little to the setup in terms of abstraction, but can possibly provide useful clues on how to pass from the case-by-case presentations of object type theories to a single type whose inhabitants are (presentations of) general type theories. For instance, the type of well-presented type theories would have to improve on our staged definitions by joining them into a single mutually recursive inductive definition that would incorporate the above QIIT construction of a single object-level theory, suitably adapted, as the realisation of a well-presented theory.

8   References

Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 18–29, 2016.

Andromedans. Andromeda proof assistant. http://www.andromeda-prover.org, 2020.

Henk Barendregt. Lambda Calculi with Types. In Samson Abramsky, Dov Gabbay, and Thomas Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Clarendon Press, 1992.

Andrej Bauer, Jason Gross, Peter LeFanu Lumsdaine, Michael Shulman, Matthieu Sozeau, and Bas Spitters. The HoTT library: a formalization of homotopy type theory in Coq. In *CPP 2017: Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, pages 164–172, January 2017.

Andrej Bauer, Philipp G. Haselwarter, and Anja Petković. A generic proof assistant. In *Foundations and Applications of Univalent Mathematics – Herrsching (Germany)*, December 2019.

Guillaume Brunerie. A general class of dependent type theories. Seminar for foundations of mathematics and theoretical computer science, Faculty of mathematics and physics, University of Ljubljana, March 2020a.

Guillaume Brunerie. A formalization of general type theories in Agda. https://github.com/guillaumebrunerie/general-type-theories, 2020b.

Alonzo Church. A set of postulates for the foundation of logic. *The Annals of Mathematics*, 33(2):346–366, 1932.

Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: a constructive interpretation of the univalence axiom. In *21st International Conference on Types for Proofs and Programs*, Tallinn, Estonia, May 2015.

Coq development team. Coq. http://coq.inria.fr/, 2020.

N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indagationes Mathematicae (Proceedings)*, 75(5):381–392, 1972.

Marcelo Fiore, Gordon Plotkin, and Daniele Turi. Abstract syntax and variable binding (extended abstract). In *14th Symposium on Logic in Computer Science*, pages 193–202. IEEE Computer Society, 1999.

Jean-Yves Girard. *Interprétation Fonctionelle et Élimination des Coupures de l'arithmétique d'ordre Supérieur*. PhD thesis, Université Paris VII, France, 1972.

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

Martin Hofmann. On the interpretation of type theory in locally Cartesian closed categories. In *Computer science logic (Kazimierz, 1994)*, volume 933 of *Lecture Notes in Comput. Sci.*, pages 427–441. Springer, Berlin, 1995. doi: 10.1007/BFb0022273.

Martin Hofmann. Syntax and Semantics of Dependent Types. In Andrew M. Pitts and Peter Dybjer, editors, *Semantics and Logics of Computation*, pages 79–130. Cambridge University Press, 1997.

Valery Isaev. Algebraic presentations of dependent type theories, March 2017. arXiv:1602.08504.

Peter T. Johnstone. *Sketches of an elephant: a topos theory compendium. Vol. 2*, volume 44 of *Oxford Logic Guides*. Oxford University Press, 2002.

Carol R. Karp. *Languages with expressions of infinite length*. North–Holland Publishing Co., Amsterdam, 1964.

Joachim Lambek and Philip J. Scott. *Introduction to higher order categorical logic*, volume 7 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1986.

Peter LeFanu Lumsdaine and Michael A. Warren. The local universes model: an overlooked coherence construction for dependent type theories. *ACM Trans. Comput. Log.*, 16(3):Art. 23, 31, 2015. ISSN 1529-3785. doi: 10.1145/2754931. arXiv:1411.1736.

Peter LeFanu Lumsdaine, Andrej Bauer, and Philipp G. Haselwarter. A formalisation of general type theories in Coq. https://github.com/peterlefanulumsdaine/general-type-theories/tree/arXiv, 2020.

Per Martin-Löf. An intuitionstic theory of types. *Technical Report, University of Stockholm*, 1972.

Per Martin-Löf. *Intuitionistic type theory*, volume 1 of *Studies in Proof Theory. Lecture Notes*. Bibliopolis, Naples, 1984.

James McKinna and Robert Pollack. Pure type systems formalized. In *Typed Lambda Calculi and Applications*, volume 664 of *Lecture Notes in Computer Science*, 1993.

Dominic Orchard, Philip Wadler, and Harley Eades. Unifying graded and parameterised monads. *Electronic Proceedings in Theoretical Computer Science*, 317, May 2020.

Frank Pfenning. Logical frameworks. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning (in 2 volumes)*, pages 1063–1147. Elsevier and MIT Press, 2001.

John C. Reynolds. Towards a Theory of Type Structure. In *Colloque Sur La Programmation, Paris, France*, volume 19 of *Lecture Notes in Computer Science*, pages 408–425. Springer Verlag, 1974.

Urs Schreiber and Michael Shulman. Quantum Gauge Field Theory in Cohesive Homotopy Type Theory. *Electronic Proceedings in Theoretical Computer Science*, 158: 109–126, july 2014.

Thomas Streicher. *Semantics of type theory*. Progress in Theoretical Computer Science. Birkhäuser Boston Inc., 1991.

Paul Taylor. *Practical foundations of mathematics*, volume 59 of *Cambridge Studies in Advanced Mathematics*. Cambridge University Press, 1999.

Taichi Uemura. A general framework for the semantics of type theory, November 2019. arXiv:1904.04097.

The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

Vladimir Voevodsky. B-systems, October 2014. arXiv:1410.5389.

Vladimir Voevodsky. Subsystems and regular quotients of C-systems, March 2016. arXiv:1406.7413.

A   Formalisation in Coq

We have partially formalised our work in the Coq proof assistant (Coq development team, 2020) on top of the HoTT library (Bauer et al., 2017). The formalisation is publicly available at (Lumsdaine et al., 2020), wherein further instructions are given on how to compile and use the formalisation. The formalisation will continue to evolve in future; the description here refers to the version tagged as `arXiv`.

The formalisation broadly follows the structure of the paper. Table 1 lists selected major definitions and theorems from the paper, along with the names of the corresponding items in the formalisation, if any. Almost all material of Sections 2, 3 and 4 has been formalised, as has some but not all of Section 5. Versions of the main definitions of Section 6 are also formalised, but at time of writing, their treatment in the formalisation has non-trivial differences from the definitions here; such items are marked with an asterisk.

Throughout the paper we worked rigorously but informally, and without discussing which mathematical foundation might be sufficient to carry out the constructions and proofs. On this topic we may consult the formalisation.

Paper	Formalisation
Family (Definition 2.1)	Auxiliary.Family.family
Closure rule (Definition 2.6)	Auxiliary.Closure.rule
Closure system (Definition 2.6)	Auxiliary.Closure.system
Derivation (Definition 2.8)	Auxiliary.Closure.derivation
Scope system (Definition 3.1)	Syntax.ScopeSystem.system
De Bruijn scope system (Example 3.2)	Examples.ScopeSystemExamples.DeBruijn
Syntactic class (Definition 3.6)	Syntax.SyntacticClass.class
Arity (Definition 3.6)	Syntax.Arity.arity
Signature (Definition 3.8)	Syntax.Signature.signature
Signature map (Definition 3.10)	Syntax.Signature.map
Raw expressions (Definition 3.11)	Syntax.Expression.expression
Raw substitution (Definition 3.16)	Syntax.Substitution.raw_substitution
Metavariable extension (Definition 3.22)	Syntax.Metavariable.extend
Instantiation of syntax (Definition 3.24)	Syntax.Metavariable.instantiate_expression
Raw context (Definition 4.1)	Typing.Context.raw_context
Raw rule (Definition 4.18)	Typing.RawRule.raw_rule
Instantiation of derivations (Corollary 4.43)	Typing.RawTypeTheory.instantiate_derivation
Associated closure system (Definition 4.21)	Typing.RawRule.closure_system
Structural rules (Definition 4.29)	Typing.StructuralRule.structural_rule
Congruence rule (Definition 4.34)	Typing.RawRule.raw_congruence_rule
Raw type theory (Definition 4.36)	Typing.RawTypeTheory.raw_type_theory
Acceptable rule (Definition 5.7)	(not formalised)
Acceptable type theory (Definition 5.12)	Metatheorem.Acceptability.acceptable
Presuppositions theorem (Theorem 5.15)	Metatheorem.Presuppositions.presupposition
Admissibility of renaming (Lemma 5.18)	Metatheorem.Elimination.rename_derivation
Admissibility of substitution (Lemma 5.20)	Metatheorem.Elimination.substitute_derivation
Admissibility of equality substitution (Lemma 5.21)	Metatheorem.Elimination.substitute_equal_derivation
Elimination of substitution (Theorem 5.22)	Metatheorem.Elimination.elimination
Uniqueness of typing (Theorem 5.23)	(not formalised)
Inversion principle (Theorem 5.27)	(not formalised)
Sequential context (Definition 6.4)	ContextVariants.wf_context_derivation[*]
Sequential rule (Definition 6.8)	(not formalised)
Well-presented rule (Definition 6.18)	(not formalised)
Well-presented type theory (Definition 6.20)	Presented.TypeTheory.type_theory[*]
Well-founded replacement (Theorem 6.32)	(not formalised)

Table 1: The correspondence between the paper and the formalisation (Lumsdaine et al., 2020). Items marked with $(*)$ differ non-trivially from their counterparts in the paper.

Our formalisation is built on top of a homotopy type theory library with an eye towards future formalisation of the categorical semantics of type theories, but is so far agnostic with respect to commitments such as the Univalence axiom or the Uniqueness of identity proofs. The only axiom that we use is function extensionality. In other words, the code can be read in plain Coq.

The formalisation confirms that our development is constructive, there are no uses of excluded middle or the axiom of choice.

It is a bit harder to tell how many universes we have used, because Coq relieves the user from explicit handling of universes. Two seem to be enough, one to serve as a base and another to work with families over the base. The base universe can be very small, say consisting of the decidable finite types, if we limit attention to finitary syntax only.

We rely in many places on the ability to perform inductive constructions and carry out proofs by induction, and so we require some meta-theoretic support for these. Of course, there is no shortage of induction in Coq, and even a fairly weak set theory will have the capability to construct the necessary inductive structures, whereas the higher-order logic of toposes would have to be extended with W-types. Alternatively, we could restrict to finitary syntax, contexts and rules throughout to allow Gödelization of syntax and reliance on induction supplied by arithmetic.