

Generalized Universe Hierarchies and First-Class Universe Levels

András Kovács @ ORCID

Eötvös Loránd University, Hungary

Abstract

In type theories, universe hierarchies are commonly used to increase the expressive power of the theory while avoiding inconsistencies arising from size issues. There are numerous ways to specify universe hierarchies, and theories may differ in details of cumulativity, choice of universe levels, specification of type formers and eliminators, and available internal operations on levels. In the current work, we aim to provide a framework which covers a large part of the design space. First, we develop syntax and semantics for cumulative universe hierarchies, where levels may come from any set equipped with a transitive well-founded ordering. In the semantics, we show that induction-recursion can be used to model transfinite hierarchies, and also support lifting operations on type codes which strictly preserve type formers. Then, we consider a setup where universe levels are first-class types and subject to arbitrary internal reasoning. This generalizes the bounded polymorphism features of Coq and at the same time the internal level computations in Agda.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases type theory, universes

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Funding The author was supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.3-VEKOP-16-2017-00002).

1 Introduction

Users of type theories often view universe levels as a bureaucratic detail, a necessary annoyance in service of boosting expressive power while retaining logical consistency. However, universe hierarchies are not going away any time soon in practical implementations of type theory. In recent developments of systems, we are getting more universes and more adjacent features:

- Agda recently added a limited cumulativity as an optional feature for universes [6], and the upcoming 2.6.2 version will extend the $\omega + 1$ universe hierarchy to $\omega * 2$.
- Coq added support for cumulative inductive types [22] and a form of bounded universe polymorphism [25].

At this point, there is a veritable zoo of universe features in existing implementations. We have perhaps even more design choices when considering the formal metatheory of type theories. Do type formers stay in the same universe, or take the \sqcup of universes of constituent types? Can eliminators target any universe, or do we instead use lifting operators to cross levels? What kind of universe polymorphism do we have, can we quantify over level bounds? Is there a type of levels, or are levels in a separate syntactic layer?

The aim of the current work is to develop semantics which covers as much as possible from the range of sensible universe features. This way, theorists and language implementors can grab a desired bag of features, and be able to show consistency of their system by a straightforward translation to one of the systems in this paper.

Contributions

1. In Section 3 we describe models of type theories where universe levels may come from any set with a well-founded transitive ordering relation. We specify models as categories



© András Kovács;
licensed under Creative Commons License CC-BY 4.0

42nd Conference on Very Important Topics (CVIT 2016).

Editors: John Q. Open and Joan R. Access; Article No. 23; pp. 23:1–23:17

Leibniz International Proceedings in Informatics



Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

equipped with level-indexed diagrams of families, as a variation on categories with families. Each morphism of levels is mapped to a lifting operation on terms and types. By varying the preservation properties of lifting operations, we can describe a range of stratification features, from two-level type theory to cumulative universes.

2. In Section 4 we use induction-recursion to model the mentioned theories. We model the strongest formulations for lifting and universes, namely cumulative universes with Russell-style type decoding.

3. In Section 5 we describe type theories with internal types for levels and level morphisms, and extend the previous inductive-recursive semantics to cover these as well. Here, we can additionally represent various universe polymorphism features and level computations.

We provide an Agda formalization of the contents of the paper. The formalization is not complete, as we skip proofs involving an excessive number of equality coercions (which are more suited to informal reasoning, using equality reflection), and instead focus on the key points.

2 Metatheory

We work in a Martin-Löf type theory which has the following features.

- Two universes named Set_0 and Set_1 , where Set_0 supports inductive-recursive types (IR) as specified by Dybjer and Setzer [9]. We may omit the universe indices if they can be inferred or if we work over arbitrary indices.
- Function extensionality and uniqueness of identity proofs (UIP). Additionally, we assume equality reflection in this paper, thus working in extensional type theory, to avoid noise from equality transports.
- We write function types as $(x : A) \rightarrow B$ with $\lambda x. t$ inhabitants. We may group multiple arguments with the same type, as in $(x y : A) \rightarrow B$. We have Σ -types as $(x : A) \times B$, with pairing as (t, u) . We have \top as the unit type with inhabitant tt , \perp as the empty type, and Bool with true and false inhabitants. Propositional identity is written as $t = u$ (coinciding with definitional equality).
- We occasionally use $\{x : A\} \rightarrow B$ for an Agda-like notation for function types with implicit arguments. We usually omit implicit applications but may explicitly write them as $t \{u\}$. We may omit implicit function types altogether if it is clear where certain variables are quantified.

3 Generalized Universe Hierarchies

In this section, we first describe notions of models for type theories with generalized universes, and discuss several variations of universes and lifting operations. Then, we pick a concrete variant (the strongest, in a sense) and construct a model for it in the metatheory.

For the basic structure of typing contexts and substitutions, let us review categories with families.

3.1 Categories with Families

► **Definition 1.** A *category with family* (cwf) [8] consists of the following data:

- A category with a terminal object. We denote the set of objects as $\text{Con} : \text{Set}$ and use capital Greek letters starting from Γ to refer to objects. The set of morphisms is $\text{Sub} : \text{Con} \rightarrow \text{Con} \rightarrow \text{Set}$, and we use σ, δ and so on to refer to morphisms. The terminal

object is \bullet with unique morphism $\epsilon : \text{Sub } \Gamma \bullet$. In initial models (that is, syntaxes) of type theories, objects correspond to typing contexts, morphisms to parallel substitutions and the terminal object to the empty context; this informs the naming scheme.

■ A *family structure*, containing $\text{Ty} : \text{Con} \rightarrow \text{Set}$ and $\text{Tm} : (\Gamma : \text{Con}) \rightarrow \text{Ty } \Gamma \rightarrow \text{Set}$, where Ty is a presheaf over the category of contexts and Tm is a presheaf over the category of elements of Ty . This means that both types (Ty) and terms (Tm) can be substituted, and substitution has functorial action. We use A, B, C to refer to types and t, u, v to refer to terms, and use $A[\sigma]$ and $t[\sigma]$ for substituting types and terms. Additionally, a family structure has *context comprehension* which consists of a context extension operation $- \triangleright - : (\Gamma : \text{Con}) \rightarrow \text{Ty } \Gamma \rightarrow \text{Con}$ together with an isomorphism $\text{Sub } \Gamma (\Delta \triangleright A) \simeq ((\sigma : \text{Sub } \Gamma \Delta) \times \text{Tm } \Gamma (A[\sigma]))$ which is natural in Γ .

From the comprehension structure, we recover the following notions:

- By going right-to-left along the isomorphism, we recover *substitution extension* $-, - : (\sigma : \text{Sub } \Gamma \Delta) \rightarrow \text{Tm } \Gamma (A[\sigma]) \rightarrow \text{Sub } \Gamma (\Delta \triangleright A)$. This means that starting from ϵ or the identity substitution id , we can iterate $-, -$ to build substitutions as lists of terms.
- By going left-to-right, and starting from $\text{id} : \text{Sub } (\Gamma \triangleright A) (\Gamma \triangleright A)$, we recover the *weakening substitution* $\mathbf{p} : \text{Sub } (\Gamma \triangleright A) \Gamma$ and the *zero variable* $\mathbf{q} : \text{Tm } (\Gamma \triangleright A) (A[\mathbf{p}])$.
- By weakening \mathbf{q} , we recover a notion of variables as De Bruijn indices. In general, the n -th De Bruijn index is defined as $\mathbf{q}[\mathbf{p}^n]$, where \mathbf{p}^n denotes n -fold composition.

There are other ways for presenting the basic categorical structure of models, which are nonetheless equivalent to cwfs, including natural models [2] and categories with attributes [4]. We use the cwf presentation for its immediately algebraic character and closeness to conventional explicit substitution syntax.

► **Notation 1.** As De Bruijn indices are hard to read, we will mostly use nameful notation for binders. For example, assuming $\text{Nat} : \{\Gamma : \text{Con}\} \rightarrow \text{Ty } \Gamma$ and $\text{Id} : \{\Gamma : \text{Con}\} (A : \text{Ty } \Gamma) \rightarrow \text{Tm } \Gamma A \rightarrow \text{Tm } \Gamma A \rightarrow \text{Ty } \Gamma$, we may write $\bullet \triangleright (n : \text{Nat}) \triangleright (p : \text{Id Nat } n n)$ for a typing context, instead of using numbered variables or cwf combinators as in $\bullet \triangleright \text{Nat} \triangleright \text{Id Nat } \mathbf{q} \mathbf{q}$.

► **Notation 2.** In the following, we will denote families by (Ty, Tm) pairs and overload context extension $- \triangleright -$ for different families.

A family structure may be closed under certain *type formers*. For example, we may close a family over function types by assuming $\Pi : (A : \text{Ty } \Gamma) \rightarrow \text{Ty } (\Gamma \triangleright A) \rightarrow \text{Ty } \Gamma$ together with abstraction, application, $\beta\eta$ -rules, and equations for the action of substitution on type and term formers.

In the following, whenever we introduce a type or term former, we always assume that it is natural with respect to substitution, i.e. all type and term formers have a corresponding substitution rule.

3.2 Morphisms and Inclusions of Families

In the rest of the paper we make use of categories equipped with possibly multiple family structures, which serves as basis for specifying universe hierarchies. However, it is not very useful to simply have multiple copies of family structures together with their type formers. In that case, every constructor and eliminator of every type former stays in the same family, and there is no interaction between families, and the most we can do is to mix them together in typing contexts. In this subsection we describe several ways of crossing between families.

► **Definition 2.** A *family morphism* F between $(\text{Ty}_0, \text{Tm}_0)$ and $(\text{Ty}_1, \text{Tm}_1)$ families consists of natural transformations mapping types to types and terms to terms, which preserves context extensions up to context isomorphism, i.e. we have that $(\Gamma \triangleright F A) \simeq (\Gamma \triangleright A)$, where \simeq denotes existence of an invertible context morphism.

Family morphisms are restrictions of so-called *weak morphisms* [3] (or *pseudomorphisms* [15]) of cwfs: a weak morphism which has the identity action on the base category is exactly a family morphism.

► **Lemma 1.** Every family morphism has invertible action on terms, i.e. there is an $F^{-1} : \text{Tm } \Gamma (F A) \rightarrow \text{Tm } \Gamma A$.

Proof. From the preservation of context extension by F , we know that F is invertible on the zero De Bruijn index, so for $q : \text{Tm } (\Gamma \triangleright F A) (F(A[p]))$, we have $F^{-1}q : \text{Tm } (\Gamma \triangleright F A) (A[p])$, such that $F(F^{-1}q) = q$. If we have any $t : \text{Tm } \Gamma (F A)$, we also have $(F^{-1}q)[\text{id}, t] : \text{Tm } \Gamma A$, and this yields the inverse action F^{-1} which acts on all terms. ◀

► **Notation 3.** In the following, we will write $\text{Lift} : \text{Ty}_0 \Gamma \rightarrow \text{Ty}_1 \Gamma$ for the action of some morphism on types, $\uparrow : \text{Tm}_0 \Gamma A \rightarrow \text{Tm}_1 \Gamma (\text{Lift } A)$ for the action on terms, and \downarrow for the inverse action on terms. We will also call the action on types *type lifting* and the action on terms *term lifting*.

We may think about the relation between *modalities* and morphisms. The main difference is that morphisms impose no structural restrictions on variables and contexts. More concretely, every Lift is dependent right adjoint [3] to the identity functor on the base category, as we have $\text{Tm } (\text{Id } \Gamma) A \simeq \text{Tm } \Gamma (\text{Lift } A)$. Hence, every morphism can be viewed as a degenerate modality.

Assume family structures $(\text{Ty}_0, \text{Tm}_0)$ and $(\text{Ty}_1, \text{Tm}_1)$ and a morphism between them. This corresponds to a basic version of *two-level type theory* [1]. This theory can be interpreted as having $(\text{Ty}_1, \text{Tm}_1)$ as a metaprogramming layer which can generate object-level programs in the $(\text{Ty}_0, \text{Tm}_0)$ layer. Lifted types correspond to types of object-level expressions; for example, $\text{Bool}_0 : \text{Ty}_0 \Gamma$ is the object-level type of Booleans, while Lift Bool_0 is the meta-level type of Bool_0 -expressions, and $\text{Bool}_1 : \text{Ty}_1 \Gamma$ is the type of meta-level Booleans. It is possible to compute a Bool_0 from a Bool_1 . Given $b : \text{Tm}_1 \Gamma \text{Bool}_1$, we can construct $\downarrow (\text{if } b \text{ then } \uparrow \text{true}_0 \text{ else } \uparrow \text{false}_0) : \text{Tm}_0 \Gamma \text{Bool}_0$.

But there is no way to compute a Bool_1 from a Bool_0 : we can try to lift the input, but there is no elimination rule for Lift Bool_0 in Ty_1 . Conceptually, terms of Lift Bool_0 are *expressions*, so they are not necessarily true or false, they can also be variables or neutral expressions, so the usual Boolean elimination is not justified for them. In general, the setup ensures that we can map from positive types in Ty_1 to their counterparts in Ty_0 , but not the other way around, while negative types are preserved by Lift up to type isomorphism.

The rules of two-level type theory appear to model a form of generative two-stage compilation with dependent types. Comparing this system to e.g. BER MetaOCaml [16], we can relate Lift to `code`, \uparrow to the quasi-quotation operation $<->$, and \downarrow to escape \sim .

While the staging aspect could be subject of future research, we currently focus on “sizing” hierarchies instead of staging hierarchies. This means that we want to eliminate from any family to any other family which is connected by a morphism.

► **Definition 3.** A *family inclusion* is a family morphism which preserves all type and term formers. This assumes that every type former which is contained in the source family, is also contained in the target family.

174 Some examples for preservation equations for type and term formers:

$$\begin{aligned}
 175 \quad & \text{Lift } (\Pi (x : A) B) = \Pi (x : \text{Lift } A) (\text{Lift } (B[x \mapsto \downarrow x])) \\
 176 \quad & \uparrow (\lambda (x : A). t) = \lambda (x : \text{Lift } A). \uparrow (t[x \mapsto \downarrow x]) \\
 177 \quad & \text{Lift Bool}_0 = \text{Bool}_1 \\
 178 \quad & \uparrow \text{true}_0 = \text{true}_1
 \end{aligned}$$

180 In general, we can skip specifying preservation for \downarrow , since it follows from \uparrow preservation equations.

182 Assume an inclusion from $(\text{Ty}_0, \text{Tm}_0)$ to $(\text{Ty}_1, \text{Tm}_1)$. Now, we can eliminate from Bool_0 to Bool_1 . If we have some $b : \text{Tm}_0 \Gamma \text{Bool}_0$, we also have $\uparrow b : \text{Tm}_1 \Gamma (\text{Lift Bool}_0)$, hence $\uparrow b : \text{Tm}_1 \Gamma \text{Bool}_1$. Then, we can use Bool_1 elimination, as in if $\uparrow b$ then true_1 else $\text{false}_1 : \text{Tm}_1 \Gamma \text{Bool}_1$. The \uparrow computation ensures that the eliminator computes appropriately on canonical terms: if b is true_0 , we get $\uparrow \text{true}_0 = \text{true}_1$ as the if-then-else scrutinee.

187 A family inclusion corresponds to a *cumulative hierarchy* consisting of two families: every type former of the smaller family is included in the larger family, with the same elimination rules.

190 ► **Definition 4.** A *strict family inclusion* between $(\text{Ty}_0, \text{Tm}_0)$ and $(\text{Ty}_1, \text{Tm}_1)$ is a family inclusion $(\text{Lift}, \uparrow, \downarrow)$ for which the following equations hold:

$$192 \quad (\Gamma \triangleright \text{Lift } A) = (\Gamma \triangleright A) \tag{1}$$

$$193 \quad \text{Tm}_1 \Gamma (\text{Lift } A) = \text{Tm}_0 \Gamma A \tag{2}$$

$$194 \quad \uparrow t = t \tag{3}$$

196 A strict inclusion corresponds to Sterling's *algebraic cumulativity* [21]. The additional equations are a matter of convenience: they allow us to omit term liftings in informal syntax¹. Most of the time we can also omit level annotations on term formers. For example, we have $\text{true}_0 : \text{Tm}_0 \Gamma \text{Bool}_0$, but also $\text{true}_0 : \text{Tm}_0 \Gamma (\text{Lift Bool}_0)$, hence $\text{true}_0 : \text{Tm}_0 \Gamma \text{Bool}_1$. Moreover, true_0 is definitionally equal to true_1 , since $\text{true}_0 = \uparrow \text{true}_0 = \text{true}_1$. Thus, using simply true is fine whenever the family is clear from context.

202 The definitional equality of true_0 and true_1 is important; without it canonicity would fail, since true_0 , false_0 , true_1 and false_1 would be four definitionally distinct inhabitants of Bool_1 . See Luo [17] for a discussion of related issues with cumulativity. It is not sufficient to specify a strict inclusion just by equations 1 and 2 in Definition 4, we need \uparrow together with equation 3 to identify term formers in different families. The other direction $\downarrow t = t$ is immediately derivable.

208 3.3 Level Structures

209 We would like to describe a range of setups with multiple families and morphisms between them. In this subsection we describe the indexing structures for such family diagrams. First, some preliminary definitions.

212 ► **Definition 5.** The *accessibility predicate* on relations is defined by the following inductive rules:

$$214 \quad \text{Acc} : \{A : \text{Set}\} \rightarrow (R : A \rightarrow A \rightarrow \text{Set}) \rightarrow A \rightarrow \text{Set}$$

$$215 \quad \text{acc} : \{a : A\} \rightarrow ((a' : A) \rightarrow R a' a \rightarrow \text{Acc } R a') \rightarrow \text{Acc } R a$$

¹ In a proof assistant, often we would still have to explicitly transport along the strict inclusion equations.

217 An inhabitant of $\text{Acc } R a$ proves that starting from $a : A$, all descending R -chains must be
 218 finite. This is ensured by the universal property of the inductive definition, as all inductive
 219 types must be well-founded.

220 ► **Lemma 2.** Accessibility is a mere proposition, i.e. all inhabitants of $\text{Acc } R a$ are equal [23,
 221 Lemma 10.3.4].

222 ► **Definition 6.** A relation $R : A \rightarrow A \rightarrow \text{Set}$ is *well-founded* if $(a : A) \rightarrow \text{Acc } R a$.

223 ► **Definition 7.** A *level structure* consists of the following components:

224 $\text{Lvl} \quad : \text{Set}_0$
 225 $- < - \quad : \text{Lvl} \rightarrow \text{Lvl} \rightarrow \text{Set}_0$
 226 $< \text{prop} : (p q : i < j) \rightarrow p = q$
 227 $- \circ - \quad : j < k \rightarrow i < j \rightarrow i < k$
 228 $< \text{wf} \quad : (i : \text{Lvl}) \rightarrow \text{Acc } < i$
 229

230 We overload Lvl to refer to a given level structure and also its underlying set. In short, a
 231 level structure is a set together with a transitive well-founded relation.

232 ► **Definition 8.** A *family diagram* over Lvl maps each $i : \text{Lvl}$ to a family structure $(\text{Ty}_i, \text{Tm}_i)$,
 233 and each $p : i < j$ to a family inclusion $(\text{Lift}_i^j p, \uparrow_i^j p, \downarrow_i^j p)$ between $(\text{Ty}_i, \text{Tm}_i)$ and $(\text{Ty}_j,$
 234 $\text{Tm}_j)$. Moreover, the mapping is functorial, so $\text{Lift}_i^k (p \circ q) A = \text{Lift}_j^k p (\text{Lift}_i^j q A)$, and similarly
 235 for $\uparrow_i^j p$ and $\downarrow_i^j p$. A *strict family diagram* is a family diagram where each inclusion is strict.

236 ► **Notation 4.** Sometimes we omit some of the i, j, p annotations from type and term liftings,
 237 if they are clear from context.

238 Our choice of level structures and diagrams is motivated by the following. First, we do not
 239 need identity morphisms in levels, because they would be mapped to trivial liftings, which are
 240 not interesting in our setting. Second, we do not need proof-relevant level morphisms, since
 241 any parallel pair of morphisms gives rise to isomorphic types. Concretely, given $p : i < j$ and
 242 $q : i < j$ such that $p \neq q$, we have $\text{Tm}_j \Gamma (\text{Lift } p A) \simeq \text{Tm}_i \Gamma A \simeq \text{Tm}_j \Gamma (\text{Lift } q A)$, and since
 243 $\text{Lift } p A$ and $\text{Lift } q A$ are in the same family, we can internally prove them isomorphic using
 244 function types and identity types. That said, every construction in this paper would still
 245 work with inverse categories as level structures.

246 3.4 Universes

247 At this point, we can talk about family diagrams, but no previously seen type former depends
 248 on levels in an interesting way. For example, Bool_i has the same canonical inhabitants as
 249 Bool_j , for any i and j . Universes introduce dependency on levels, by serving as classifiers for
 250 smaller families internally to larger families.

251 ► **Definition 9.** A family diagram supports *universe formation* if it supports the following:

252 $\text{U} \quad : (i j : \text{Lvl}) \rightarrow i < j \rightarrow \text{Ty}_j \Gamma$
 253 $\text{LiftU} : \text{Lift}_j^k p (\text{U } i j q) = \text{U } i k (p \circ q)$
 254

255 We also need a way to pin down universes as classifiers. We consider two variants.

256 ► **Definition 10.** A family diagram has *Coquand universes* if it has universe formation
 257 and additionally supports $\text{El} : \text{Tm}_j \Gamma (\text{U } i j p) \rightarrow \text{Ty}_i \Gamma$, and its inverse $\text{Code} : \text{Ty}_i \Gamma \rightarrow$
 258 $\text{Tm}_j \Gamma (\text{U } i j p)$.

► **Definition 11.** A family diagram has *Russell universes* if it has Coquand universes and additionally satisfies $\text{Tm}_j \Gamma (\text{U } i \ j \ p) = \text{Ty}_i \Gamma$ and $\text{El } t = t$.

The move from Coquand to Russell universes is fairly similar to the move from inclusions to strict inclusions. The Russell variant makes it possible to informally omit El and Code . Likewise, the $\text{El } t = t$ condition ensures appropriate naturality. If we only assumed $\text{Tm}_j \Gamma (\text{U } i \ j \ p) = \text{Ty}_i \Gamma$ but not Coquand universes, we would not be able to prove that a $t : \text{Tm}_j \Gamma (\text{U } i \ j \ p)$ substituted *as a term* is the same thing as t substituted *as a type*. Both would be written as $t[\sigma]$ in our notation, but they involve different $-[-]$ operations.

Unlike every other type or term former, there is no lifting computation rule for El and Code . Intuitively, the issue is that we would need to relate type lifting and term lifting, but while term lifting is invertible, type lifting is not. Lift sends a $\text{Ty}_i \Gamma$ to a $\text{Ty}_j \Gamma$, and $\text{Ty}_j \Gamma$ is not isomorphic to $\text{Ty}_i \Gamma$, because it contains more universes. So, for example, lifting $\text{Bool}_0 : \text{Ty}_0 \Gamma$ as a type to $\text{Ty}_1 \Gamma$ yields Bool_1 , but lifting Bool_0 as a term yields Bool_0 .

Assuming Coquand or Russell universes and $p : i < j$, we can recover polymorphic functions, for example, we may have $\text{id} : \Pi(A : \text{U } i \ j \ p)(\text{Lift } p(\text{El } A) \rightarrow \text{Lift } p(\text{El } A))$ for the polymorphic identity function. Here, we quantify over terms of U , and since every type former stays on the same level (including Π), we have to Lift the types in the codomain to match the level of the domain. We can also recover large elimination, for example as in

$$(\lambda (b : \text{Bool}_j). \text{if } b \text{ then } \text{Code } \top_i \text{ else } \text{Code } \perp_i) : \text{Tm}_j \Gamma (\text{Bool}_j \rightarrow \text{U } i \ j \ p).$$

4 Semantics

In this section we give a model for a type theory with generalized universes. Let's make the notion of model concrete first.

► **Definition 12** (Notion of model for a type theory with generalized universes (TTGU)). Fix a Lvl structure. A model for TTGU consists of

1. A base category (Con, Sub) with a terminal object \bullet .
2. A strict family diagram $(\text{Ty}_i, \text{Tm}_i)$ over Lvl , supporting Russell universes, and each family structure is closed under the same basic type formers.

The choice of available basic type formers is up to personal taste, and it will not significantly affect the following model construction.

Both in families and universes we choose the stricter formulation, since if we give a model which proves the strict syntax consistent, we immediately get a model which proves the weak syntax consistent².

4.1 Inductive-Recursive Codes

The task is to interpret the Lvl -many universes of TTGU using a metatheoretic feature which is more compact and already well-studied. For this, we need to define a Lvl -indexed type of type codes. Since Lvl and $- < -$ can be arbitrary, we effectively need to define transfinite hierarchies of codes. We use an inductive-recursive (IR) [9] definition, for the following reasons.

² We always get *initial* and *terminal* models automatically, because of the algebraic character of the theories in this paper. We also get a *freely generated* strict model from a weak model, from the left adjoint of the functor which forgets the strictness equations. But none of these tricks can be used to automatically get a consistency proof.

First, IR is already supported in the proof assistant Agda, and it is very useful to be able to sketch out ideas in a machine-checked setting. It would be much harder to do the same when developing semantics in set theory.

Second, could we use weaker type-theoretic features, like super universes [19] or Mahlo universes [20]? These are sufficient to model transfinite hierarchies. However, it is not clear how to additionally support *recursive sub-universes*, which in our work corresponds to the type former preservation property of Lift. Thus, we give a custom IR definition which corresponds more directly to TTGU structure.

► **Definition 13** (IR codes). Assume $i : \text{Lvl}$ and $f : (j : \text{Lvl}) \rightarrow j < i \rightarrow \text{Set}_0$. We define U^{IR} and El^{IR} by simultaneous IR:

$$\begin{array}{ll}
 \text{U}^{\text{IR}} : \text{Set}_0 & \text{El}^{\text{IR}} : \text{U}^{\text{IR}} \rightarrow \text{Set}_0 \\
 \text{U}' : (j : \text{Lvl}) \rightarrow j < i \rightarrow \text{U}^{\text{IR}} & \text{El}^{\text{IR}} (\text{U}' j p) = f j p \\
 \Pi' : (A : \text{U}^{\text{IR}}) \rightarrow (\text{El}^{\text{IR}} A \rightarrow \text{U}^{\text{IR}}) \rightarrow \text{U}^{\text{IR}} & \text{El}^{\text{IR}} (\Pi' A B) = (a : \text{El}^{\text{IR}} A) \rightarrow \text{El}^{\text{IR}} (B a) \\
 \perp' : \text{U}^{\text{IR}} & \text{El}^{\text{IR}} \perp' = \perp \\
 \text{Bool}' : \text{U}^{\text{IR}} & \text{El}^{\text{IR}} \text{Bool}' = \text{Bool}
 \end{array}$$

We use the prime accents (') to disambiguate IR codes from type formers in TTGU or the metatheory. For basic type formers, we only include codes for function types, the empty type, and Bool. Other type formers are straightforward to add (and we do have more in the Agda formalization).

► **Notation 5.** We may write $\text{U}^{\text{IR}}_{i f}$ and $\text{El}^{\text{IR}}_{i f}$ in order to make parameters explicit.

$(\text{U}^{\text{IR}}, \text{El}^{\text{IR}})$ can be viewed as a *universe operator*: given semantics for an initial segment of Lvl (given by i and f), we create a new universe which is closed under basic type formers, and also closed under all sets in f by the way of U' . Most importantly, this operation can be transfinitely iterated. We first define universes for initial segments of Lvl, by induction on the accessibility of levels:

$$\begin{array}{l}
 \text{U}_{<} : (i : \text{Lvl}) \{p : \text{Acc} (- < -) i\} \rightarrow (j : \text{Lvl}) \rightarrow j < i \rightarrow \text{Set}_0 \\
 \text{U}_{<} i \{ \text{acc } f \} j p = \text{U}^{\text{IR}}_{j (\text{U}_{<} j \{ f j p \})}
 \end{array}$$

► **Definition 14** (Semantic universe). Since every level is accessible, we can define the full semantic hierarchy and its decoding function.

$$\begin{array}{ll}
 \text{U} : \text{Lvl} \rightarrow \text{Set}_0 & \text{El} : \{i : \text{Lvl}\} \rightarrow \text{U } i \rightarrow \text{Set}_0 \\
 \text{U } i = \text{U}^{\text{IR}}_{i (\text{U}_{<} i \{ < \text{wf } i \})} & \text{El } \{i\} = \text{El}^{\text{IR}}_{i (\text{U}_{<} i \{ < \text{wf } i \})}
 \end{array}$$

► **Lemma 3.** Assuming $p : i < j$, we have the computation rule $\text{U}_{<} j p = \text{U } j$. Proof: we may assume that any witness for $\text{Acc} (- < -) i$ is of the form $\text{acc } f$ for some f . Then the equation becomes $\text{U}^{\text{IR}}_{j (\text{U}_{<} j \{ f j p \})} = \text{U}^{\text{IR}}_{j (\text{U}_{<} j \{ < \text{wf } j \})}$, but by Lemma 2 the $f j p$ and $< \text{wf } j$ witnesses are equal. ◀

► **Definition 15** (Semantic Lift). We define by induction on U^{IR} a function with type $(p : i < j) \rightarrow (A : \text{U } i) \rightarrow (A' : \text{U } j) \times (\text{El } A' = \text{El } A)$. However, for the sake of clarity, we present this here as two (mutual) functions:

$$\begin{array}{l}
 \text{Lift} : (p : i < j) \rightarrow \text{U } i \rightarrow \text{U } j \\
 \text{ElLift} : (p : i < j) \rightarrow (A : \text{U } i) \rightarrow \text{El} (\text{Lift } A) = \text{El } A
 \end{array}$$

Let us look at Lift first:

$$\begin{aligned}
 \text{Lift } p (\mathbf{U}' k q) &= \mathbf{U}' k (p \circ q) \\
 \text{Lift } p (\Pi' A B) &= \Pi' (\text{Lift } p A) (\lambda a. \text{Lift } p (B a)) \\
 \text{Lift } p \perp' &= \perp' \\
 \text{Lift } p \text{Bool}' &= \text{Bool}'
 \end{aligned}$$

Above, the Π' definition is well-typed by $\text{ELift } p A$. For the proof of ELift , the only interesting case is \mathbf{U}' . Here, we need to show $\mathbf{U}_{<} j k (p \circ q) = \mathbf{U}_{<} i k q$, but by Lemma 3 both sides are $\mathbf{U} k$.

► **Lemma 4.** Properties of Lift:

1. Lift preserves all basic type formers; this is immediate from the definition.
2. Lift is functorial, i.e. $\text{Lift } (p \circ q) A = \text{Lift } p (\text{Lift } q A)$. This follows by induction on A , and we make use of the irrelevance of $- < -$ in the \mathbf{U}' case. ◀

4.2 IR Model of TTGU

We give a model of TTGU in this section.

► **Notation 6.** To avoid name clashing between components of the model and metatheoretic definitions, we use **bold** font to refer to TTGU components.

► **Definition 16** (Base category). The base category is simply the category of sets and functions in Set_0 , i.e. $\mathbf{Con} = \text{Set}_0$, $\mathbf{Sub} \Gamma \Delta = \Gamma \rightarrow \Delta$, and the terminal object is \top .

► **Definition 17** (Family diagram). We map $i : \text{Lvl}$ to a family structure as follows.

$$\mathbf{T}_i \Gamma = \Gamma \rightarrow \mathbf{U} i \quad \mathbf{Tm}_i \Gamma A = (\gamma : \Gamma) \rightarrow \text{El} (A \gamma)$$

Type and term substitution are given by composition with some function $\sigma : \Gamma \rightarrow \Delta$. Comprehension structure is given by $\Gamma \triangleright A = (\gamma : \Gamma) \times \text{El} (A \gamma)$. Type lifting along $p : i < j$ is as follows:

$$\begin{aligned}
 \mathbf{Lift}_i^j p : \mathbf{T}_i \Gamma &\rightarrow \mathbf{T}_j \Gamma \\
 \mathbf{Lift}_i^j p A &= \lambda \gamma. \mathbf{Lift}_i^j p (A \gamma)
 \end{aligned}$$

Now, two of the strict inclusion equations follow from ELift , namely $(\Gamma \triangleright \mathbf{Lift}_i^j p A) = (\Gamma \triangleright A)$ and $\mathbf{Tm}_j \Gamma (\mathbf{Lift}_i^j p A) = \mathbf{Tm}_i \Gamma A$. Thus, we can just define term lifting as $\uparrow_i^j p t = t$ and $\downarrow_i^j p t = t$. Basic type formers are as follows.

$$\mathbf{\Pi} A B = \lambda \gamma. \Pi' (A \gamma) (\lambda \alpha. B (\gamma, \alpha)) \quad \perp_i = \lambda \gamma. \perp' \quad \mathbf{Bool}_i = \lambda \gamma. \text{Bool}'$$

$\mathbf{Lift}_i^j p$ preserves type formers by Lemma 4. We define basic term formers and eliminators using metatheoretic features, e.g. $\mathbf{true}_i = \lambda \gamma. \text{true}$ and $(\lambda_i x. t) = \lambda \gamma \alpha. t (\gamma, \alpha)$. Note that since semantic term formers are just external constructors, they do not depend on levels, so e.g. \mathbf{true}_i is the same at all i . This implies that $\uparrow_i^j p$ preserves term formers as well, so $(\mathbf{Lift}_i^j p, \uparrow_i^j p, \downarrow_i^j p)$ is a strict family inclusion.

We define universes as $\mathbf{U} i j p = \lambda \gamma. \mathbf{U}'_i j p$. With this, $\mathbf{Lift}_j^k p (\mathbf{U} i j q) = \mathbf{U} i k (p \circ q)$ follows by the definition of semantic Lift. The Russell universe equation $\mathbf{Tm}_j \Gamma (\mathbf{U} i j p) = \mathbf{T}_i \Gamma$ follows from Lemma 3, so we can define \mathbf{El} and \mathbf{Code} as identity functions.

► **Theorem 1** (Consistency of TTGU). *There is no closed syntactic term of \perp_i for any i .*

Proof. Assuming a syntactic $t : \mathbf{Tm}_i \bullet \perp_i$, we can interpret it in the previously given model, which yields an inhabitant of the metatheoretic \perp , hence a contradiction. ◀

5 First-Class Universe Levels

In the following, we specify and model type theories where levels and their morphisms are represented by internal types.

However, it would be awkward to pick a particular structure for levels, and specify a type theory which internalizes that structure; for example internalizing levels as natural numbers. We do not want to repeat the specification and semantics for each choice of level structure; instead, we aim to have a more generic solution.

1. We first give a specification of *type theory with dependent levels*, or TTDL, where levels and level morphisms may depend on typing contexts. Here, liftings, universes and type formers are specified, but the internal structure of levels is not yet pinned down.
2. We show that we can extend TTDL with *level reflection* rules, which identify levels with particular internal types, thereby getting *type theories with first-class levels*, or TTFL. This decreases the amount of work that we have to do, in order to get semantics for different level setups. We only need to pick an external level structure such that it can be also represented using TTDL type formers.

► **Definition 18.** A model of TTDL consists of the following.

1. A base category $(\mathbf{Con}, \mathbf{Sub})$ with terminal object \bullet .
2. A “dependent” level structure on the base category:

$$\begin{aligned} \mathbf{Lvl} & : \mathbf{Con} \rightarrow \mathbf{Set} \\ - < - & : \{\Gamma : \mathbf{Con}\} \rightarrow \mathbf{Lvl} \Gamma \rightarrow \mathbf{Lvl} \Gamma \rightarrow \mathbf{Set} \\ < \mathbf{prop} & : (p q : i < j) \rightarrow p = q \\ - \circ - & : j < k \rightarrow i < j \rightarrow i < k \end{aligned}$$

Additionally, \mathbf{Lvl} and $- < -$ are natural in the base category, so they support substitution operations. *Remark:* at this point, we do not require well-foundedness for $- < -$, as it has no bearing on basic lifting and universe rules, and well-foundedness will be usually internally provable when we add level reflection rules.

3. A “bootstrapping” assumption on levels. This can be any collection of levels and morphisms. We pick the assumption that $l_0, l_1 : \mathbf{Lvl} \Gamma$ exist together with $l_{01} : l_0 < l_1$. We need to assume at least some levels, because otherwise the TTDL syntax is trivial; it is not even possible to write down a non-empty context or a closed type former. $l_0 < l_1$ allows large eliminations on type formers so it provides a fair amount of expressive power.
4. A family structure:

$$\begin{aligned} \mathbf{Ty} & : (\Gamma : \mathbf{Con}) \rightarrow \mathbf{Lvl} \Gamma \rightarrow \mathbf{Set} \\ \mathbf{Tm} & : (\Gamma : \mathbf{Con}) \{i : \mathbf{Lvl} \Gamma\} \rightarrow \mathbf{Ty} \Gamma i \rightarrow \mathbf{Set} \\ - \triangleright - & : (\Gamma : \mathbf{Con}) \{i : \mathbf{Lvl} \Gamma\} \rightarrow \mathbf{Ty} \Gamma i \rightarrow \mathbf{Con} \end{aligned}$$

We have type and term substitution, which depends on level substitution. For instance, we have

$$-[-] : \mathbf{Ty} \Delta i \rightarrow (\sigma : \mathbf{Sub} \Gamma \Delta) \rightarrow \mathbf{Ty} \Gamma (i[\sigma])$$

We also have a comprehension isomorphism $\mathbf{Sub} \Gamma (\Delta \triangleright A) \simeq (\sigma : \mathbf{Sub} \Gamma \Delta) \times \mathbf{Tm} \Gamma (A[\sigma])$, which is natural in Γ .

427 5. A lifting structure with

$$\begin{aligned}
 428 \quad \text{Lift} &: \{\Gamma : \text{Con}\} \{i j : \text{Lvl } \Gamma\} \rightarrow i < j \rightarrow \text{Ty } \Gamma i \rightarrow \text{Ty } \Gamma j \\
 429 \quad \uparrow &: \{\Gamma : \text{Con}\} \{i j : \text{Lvl } \Gamma\} (p : i < j) \rightarrow \text{Tm } \Gamma A \rightarrow \text{Tm } \Gamma (\text{Lift } p A) \\
 430
 \end{aligned}$$

431 Such that

- 432 a. Lift preserves all basic type formers and has functorial action on $p \circ q$.
- 433 b. \uparrow has an inverse \downarrow , preserves all basic term formers and has functorial action on $p \circ q$.
- 434 c. $(\Gamma \triangleright A) = (\Gamma \triangleright \text{Lift } p A)$, and $\text{Tm } \Gamma A = \text{Tm } \Gamma (\text{Lift } p A)$ and $\uparrow t = t$.

435 Above we mention basic type formers, although we haven't yet specified those. The way
 436 this should be understood, is that any basic type former introduced from now on should
 437 come equipped with preservation equations for lifting. This is similar to how we mandate
 438 that any introduced type former must be natural with respect to substitution.

439 6. A universe structure

$$\begin{aligned}
 440 \quad \text{U} &: \{\Gamma : \text{Con}\} (i j : \text{Lvl } \Gamma) \rightarrow i < j \rightarrow \text{Ty } \Gamma j \quad \text{El} : \text{Tm } \Gamma (\text{U } i j p) \rightarrow \text{Ty } \Gamma i \\
 441
 \end{aligned}$$

442 Such that $\text{Lift } p (\text{U } i j q) = \text{U } i k (p \circ q)$, El has inverse Code, $\text{Tm } \Gamma (\text{U } i j p) = \text{Ty } \Gamma i$ and
 443 $\text{El } t = t$.

444 7. Basic type formers.

445 ► **Definition 19** (IR model of TTDL). Assume an external Lvl structure such that it supports
 446 $l_0, l_1 : \text{Lvl}$ and $l_{01} : l_0 < l_1$ (the bootstrapping assumption). We again use the IR universe
 447 constructions from Section 4.1, instantiated to the assumed Lvl structure. We describe
 448 components of the model in order. Again, we write components of the model in **bold** font.

- 449 1. The base category remains unchanged from the TTGU model.
- 450 2. For the level structure, we define $\text{Lvl } \Gamma = \Gamma \rightarrow \text{Lvl}$ and $i < j = (\gamma : \Gamma) \rightarrow i \gamma < j \gamma$.
 451 Substitution for internal levels and morphisms is given by function composition with
 452 $\sigma : \Gamma \rightarrow \Delta$. Internal composition and $< \text{prop}$ follow from the external counterparts.
- 453 3. The internal bootstrapping assumption is modeled with the external counterpart.
- 454 4. We define $\text{Ty } \Gamma i = (\gamma : \Gamma) \rightarrow \text{U } (i \gamma)$ and $\text{Tm } \Gamma A = (\gamma : \Gamma) \rightarrow \text{El } (A \gamma)$. Substitution is
 455 again function composition, and we have $\Gamma \triangleright A = (\gamma : \Gamma) \times \text{El } (A \gamma)$.
- 456 5. Type lifting is given by $\text{Lift } p A = \lambda \gamma. \text{Lift } (p \gamma) (A \gamma)$. Similarly as in the TTGU model,
 457 $\text{Tm } \Gamma A = \text{Tm } \Gamma (\text{Lift } p A)$ and $\Gamma \triangleright A = \Gamma \triangleright (\text{Lift } p A)$ follow from the ElLift equality, and
 458 term lifting is the identity function.
- 459 6. We define $\text{U } i j p = \lambda \gamma. \text{U}' (i \gamma) (p \gamma)$. Again, we have $\text{Tm } \Gamma (\text{U } i j p) = \text{Ty } \Gamma i$ by Lemma 3,
 460 and **El** and **Code** are identity functions.
- 461 7. Basic type formers are interpreted using U^{IR} codes. Preservation of type and term formers
 462 by lifting follows by the definition of Lift and El.

463 To summarize, the only interesting change compared to the TTGU model is that levels
 464 and level morphisms gain potential dependency on contexts. However, in the IR model, this
 465 is simply the addition of an extra semantic function parameter.

5.1 Level Reflection

► **Definition 20** (Level reflection rules). Assume that we have definitions for internal levels in the syntax of TTDL, i.e. all of the following are defined:

$$\begin{aligned}
& \text{Lvl}^I : \text{Ty } \Gamma \, l_0 \\
& l_0^I, l_1^I : \text{Tm } \Gamma \, \text{Lvl}^I \\
& - <^I - : \text{Tm } \Gamma \, \text{Lvl}^I \rightarrow \text{Tm } \Gamma \, \text{Lvl}^I \rightarrow \text{Ty } \Gamma \, l_0 \\
& l_{01}^I : \text{Tm } \Gamma \, (l_0^I <^I l_1^I)
\end{aligned}$$

A *reflection rule* for the above consists of

1. $\text{mk}_{\text{Lvl}} : \text{Tm } \Gamma \, \text{Lvl}^I \rightarrow \text{Lvl } \Gamma$ with its inverse un_{Lvl} , such that $\text{mk}_{\text{Lvl}} l_0^I = l_0$ and $\text{mk}_{\text{Lvl}} l_1^I = l_1$.
2. $\text{mk}_{<} : \text{Tm } \Gamma \, (i <^I j) \rightarrow \text{mk}_{\text{Lvl}} i < \text{mk}_{\text{Lvl}} j$ with its inverse $\text{un}_{<}$.

For any definition of internal levels, we may extend the specification of TTDL with the corresponding reflection rule, thereby getting an algebraic signature for a type theory with first-class levels (TTFL). We can easily get a TTFL with an IR model in the following way. First, pick an external Lvl structure which a) satisfies the bootstrapping assumption b) has sets of levels and morphisms which can be represented with syntactic TTDL types.

For example, if $\text{Lvl} = (\text{Nat}, - < -)$, with $l_0 = 0$ and $l_1 = 1$, and TTDL supports natural numbers, then we can define Lvl^I as the internal Nat_{l_0} , and define $- <^I -$ as the usual ordering of numbers, using TTDL type formers and large elimination (which is available from $l_0 < l_1$). Then it follows that the model in Definition 19, instantiated to the current level structure, satisfies level reflection. The model even supports the stricter $\text{Tm } \Gamma \, \text{Nat}_{l_0} = \text{Lvl } \Gamma$ equation, but in general it is easier to set up models if only an isomorphism is required.

5.2 Universe Features in TTFL

We describe some of the features expressible in TTFL.

Bounded universe polymorphism is realized by quantifying over levels and morphisms with the usual Π types. For example, if levels strictly correspond to internal natural numbers, we may have

$$\begin{aligned}
& \text{idUpTo3} : \Pi(l : \text{Nat}_3)(p : \text{Lift}_0^3(l <^I 3))(A : \text{U } l \, 3(\text{mk}_{<} p)) \rightarrow \text{Lift}(\text{mk}_{<} p) A \rightarrow \text{Lift}(\text{mk}_{<} p) A \\
& \text{idUpTo3} = \lambda l p A a. a
\end{aligned}$$

Here, we make sure that all types are on the same level, by appropriate lifting. We assume that internal levels are in Nat_0 , but we can bind an $l : \text{Nat}_3$, because by cumulativity l is also a term of Nat_0 . Likewise, the p variable is a term of $\text{Lift}_0^3(l <^I 3)$ and $l <^I 3$ as well.

Transfinite hierarchies are naturally supported. For example, Lvl can be identified with Maybe Nat_0 , where **Nothing** defines ω and **Just** n is a finite level. Then, by the definition of morphisms, we have $< \omega : \Pi(n : \text{Nat}_0) \rightarrow \text{Just } n <^I \omega$. We can use this to quantify over finite levels, as in the following type:

$$\Pi(n : \text{Nat}_\omega)(A : \text{U } n \, \omega(\text{mk}_{<} (< \omega n))) \rightarrow \text{Lift}(\text{mk}_{<} (< \omega n)) A \rightarrow \text{Lift}(\text{mk}_{<} (< \omega n)) A$$

This type is in $\text{Ty } \Gamma \, \omega$, but it is not in any universe, since ω is the greatest level.

Induction on levels and level morphisms. In Agda 2.6.1, there is an internal type of finite levels, and while construction rules and some built-in operations on levels are exposed, there is no general elimination rule on levels. Thus, there is a $\text{Nat} \rightarrow \text{Lvl}$ conversion

function but it has no inverse. In contrast, TTFL supports arbitrary elimination on levels and morphisms.

Type formers returning in least upper bounds of levels. It is common in type theories to allow type formers to have parameter types in different universe levels, say i and j , and return in level $i \sqcup j$. In TTFL, whenever levels are *trichotomous*, meaning that the ordering and equality of levels is internally decidable, $i \sqcup j$ can be defined as the greater of i and j , and the “heterogeneous” type formers are derivable³.

Coercive cumulative subtyping. TTFL as specified does not directly support cumulative subtyping. However, it is compatible with coercive subtyping. Consider the following rules:

$$\begin{aligned}
 & - \leq - : \text{Ty } \Gamma \ i \rightarrow \text{Ty } \Gamma \ j \rightarrow \text{Set} \\
 & \text{coerce} : A \leq B \rightarrow \text{Tm } \Gamma \ A \rightarrow \text{Tm } \Gamma \ B \\
 & \leq \text{refl} : A \leq A \\
 & \text{U} \leq : i < i' \rightarrow \text{U } i \ j \ p \leq \text{U } i' \ k \ q \\
 & \Pi \leq : (p : A' \leq A) \rightarrow ((a' : \text{Tm } \Gamma \ A') \rightarrow B[x \mapsto \text{coerce } p \ a']) \leq B'[x \mapsto a']) \\
 & \rightarrow \Pi(x : A) B \leq \Pi(x : A') B'
 \end{aligned}$$

Any model of TTFL can support the above rules: we can define $- \leq -$ and coerce by indexed induction-recursion [10], where we define coercion along $\text{U} \leq$ by type lifting, and coercion along $\Pi \leq$ by backwards-forwards coercion. It is possible to extend the subtyping relation with rules for other basic type formers.

Note that Π is contravariant in the domain. This is easily supported with our IR semantics, unlike in the set-theoretic model of cumulativity for Coq [22], where function domains are invariant.

5.3 Effects of Choice of Level Structure

TTFL features clearly vary depending on level structures. We make some basic observations.

- We did not mandate that the level of Lvl^I is the least level, i.e. that $l_0 < i$ for every $i \neq l_0$. If this holds, then it is possible to have level polymorphism at every level: at l_0 we can just bind a Lvl^I , and at every other level, we can lift Lvl^I to that level. However, levels are not necessarily totally ordered, and l_0 does not have to be the least. This means that universe polymorphism is prohibited in levels which are not connected to l_0 .
- If levels are given by a limit ordinal, then every TTFL type is contained in a universe. If levels form a successor ordinal, then this is not the case. For example, Agda 2.6.1 has $\omega + 1$ levels (externally), where Set_ω is the topmost universe, but Set_ω has no type.
- While it is possible to quantify over all levels (using plain Π types), it is not possible to have level polymorphism over all levels. We may try to type an identity function for all levels, as $\Pi(i : \text{Lift } ? \text{Lvl}^I)(A : \text{U } (\text{mk}_{\text{Lvl}} i) ??) \rightarrow \text{Lift } ? A \rightarrow \text{Lift } ? A$. The issue is in $\text{U } (\text{mk}_{\text{Lvl}} i) ??$, where we would have to find a level which is larger than *every* level. The solution to this issue is to simply add more levels. For example, for polymorphism over finite levels, we may pick $\omega + \omega$ as the first limit ordinal which can internalize finite level polymorphism; this is what Agda 2.6.2 does.

³ A level structure which is trichotomous and supports *extensionality*, i.e. $(\forall i. (i < j) \iff (i < k)) \rightarrow j = k$, is a *type-theoretic ordinal*. Assuming excluded middle, type-theoretic ordinals are equivalent to classical ordinals [23, Section 10.3].

550 6 Related Work

551 Predicative hierarchies originate from Russell’s ramified type theories [24]. In the more
 552 modern formulations of type theory, Martin-Löf proposed a countable predicative hierarchy
 553 [18], as a way to remedy the inconsistency of the previous version of the theory (which assumed
 554 type-in-type). Harper and Pollack described universe inference with level assignments and
 555 also a form of level polymorphism [14]. Sterling [21] gave an algebraic specification much
 556 like ours for a type theory with countable cumulative universes, and proved canonicity for it.

557 There have been proposals for strengthening universes with various closure principles
 558 and universe operators. Palmgren’s super universes and higher-order universes [19] and
 559 Setzer’s Mahlo universes [20] are examples for this. These are sufficient to model transfinite
 560 hierarchies, but as we noted in Section 4.1, we do not know how to model strict inclusions
 561 with them. Variants of induction-recursion [9, 10, 11] are particularly flexible and powerful
 562 extensions to universes.

563 It is worth to summarize here the universe features in the current type theory implement-
 564 ations.

565 **Agda 2.6.1** has $\omega + 1$ -many non-cumulative predicative universes as Set_i , with optional
 566 cumulative subtyping only for universes [6]. It also has an internal type $\text{Level} : \text{Set}_0$ for
 567 finite levels (hence, excluding ω), which supports constructors and some built-in operations,
 568 but no general elimination rule. There is also a countable parallel hierarchy Prop_i for strict
 569 propositions [12]. Agda 2.6.2 will extend the Set_i hierarchy to $\omega * 2$.

570 **Coq 8.13** has ω -many cumulative predicative universes with cumulative subtyping for
 571 all type formers [22]. It supports bounded universe polymorphism, but it has no internal
 572 type for levels, and universe polymorphic definitions are not internally typeable. It also has
 573 an impredicative Prop universe and optionally impredicative bottom Set universe. Version
 574 8.13 added experimental support for a parallel countable cumulative hierarchy for strict
 575 propositions.

576 **Lean 3.3** has countable non-cumulative predicative Type_i universes with universe poly-
 577 morphism, and no internal type of levels [7]. It also has strict impredicative Prop .

578 **Idris 1** has countable cumulative predicative universes with cumulative subtyping only
 579 for universes, typical-ambiguity-style level inference and no universe polymorphism [5].

580 Of the above features, what TTFL does not support is a) impredicativity b) the interaction
 581 of Prop and Type universes, i.e. the restrictions on Prop elimination.

582 7 Conclusion and Future Work

583 In the current work, we developed a framework for modeling a variety of universe features in
 584 type theories. At this point, we may ask the question: if IR is sufficient to model every feature,
 585 why not simply add IR support in a practical implementation, and drop the menagerie of
 586 universe features?

587 The answer is that IR provides is a *deep embedding* of universe features, which is usually
 588 less convenient to use than *native* features. For example, both Coq and Agda have powerful
 589 automatic solving for filling out implicit universe levels. We also do not have to invoke El
 590 or the $\text{U}_<$ computation rule explicitly, and in Coq we can use implicit syntax for subtyping
 591 instead of explicit coercions.

592 This trade-off between convenience and formal minimalism is similar to the situation
 593 with inductive types. Formally, W-types and identity types are easier to handle than general
 594 inductive families, but the latter are far more convenient to actually use. Ideally, we would

like to justify complicated convenience features by reduction to minimal features. With the current paper, we hope to have made progress in this manner.

7.1 Future Work

Several related topics are not discussed in this paper and could be subject to future work.

First, besides consistency, we are often interested in *canonicity*, *normalization* or other metatheoretical properties. The current work focuses on consistency and leaves other properties to future work. We did keep canonicity in mind when specifying the systems in this paper. Hopefully the usual proof method of gluing (in other words, proof-relevant logical predicates) [15, 21] can be adapted to the theories in this paper.

Second, we only focus on using universes as size-based classifiers for types. Stratification features are also present in two-level type theory [1], modal type theories [13] or as h-levels in homotopy type theory [23]. It would be interesting to port universe features in this paper to two-level type theory, as they would hopefully model a form of stage polymorphism in multi-stage compilation. We could try representing Prop universes in TTFL as well. This is closely related to h-level based stratification.

Third, we do not discuss implementation strategies and ergonomics of universe features. Which universe hierarchies support good proof automation? What kind of impact do first-class levels have on elaboration algorithms? Hopefully the current work can aid answering these questions, by at least giving a way to quickly check if some features are logically consistent.

Lastly, we do not handle impredicative universes. The main reason for this is that we do not know the consistency of having IR and impredicative function space together in the same universe, and modeling impredicativity seems to require this assumption in the metatheory. This could be investigated as well in future work.

References

- 1 Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications. *ArXiv e-prints*, may 2019. URL: <http://arxiv.org/abs/1705.03307>.
- 2 Steve Awodey. Natural models of homotopy type theory. *Math. Struct. Comput. Sci.*, 28(2):241–286, 2018. URL: <https://doi.org/10.1017/S0960129516000268>, doi:10.1017/S0960129516000268.
- 3 Lars Birkedal, Ranald Clouston, Bassel Manna, Rasmus Ejlers Møgelberg, Andrew M. Pitts, and Bas Spitters. Modal dependent type theory and dependent right adjoints. *Math. Struct. Comput. Sci.*, 30(2):118–138, 2020. URL: <https://doi.org/10.1017/S0960129519000197>, doi:10.1017/S0960129519000197.
- 4 John Cartmell. *Generalised algebraic theories and contextual categories*. PhD thesis, Oxford University, 1978.
- 5 The Idris Community. Documentation for the idris language, 2021. URL: <http://docs.idris-lang.org/en/latest/index.html>.
- 6 Agda developers. Agda documentation, 2021. URL: <https://agda.readthedocs.io/en/v2.6.1.3/>.
- 7 Lean developers. Lean reference manual, version 3.3, 2021. URL: https://leanprover.github.io/reference/lean_reference.pdf.
- 8 Peter Dybjer. Internal type theory. In Stefano Berardi and Mario Coppo, editors, *Types for Proofs and Programs, International Workshop TYPES'95, Torino, Italy, June 5-8, 1995, Selected Papers*, volume 1158 of *Lecture Notes in Computer Science*, pages 120–134. Springer, 1995. URL: https://doi.org/10.1007/3-540-61780-9_66, doi:10.1007/3-540-61780-9_66.

- 9 Peter Dybjer and Anton Setzer. A finite axiomatization of inductive-recursive definitions. In Jean-Yves Girard, editor, *Typed Lambda Calculi and Applications, 4th International Conference, TLCA'99, L'Aquila, Italy, April 7-9, 1999, Proceedings*, volume 1581 of *Lecture Notes in Computer Science*, pages 129–146. Springer, 1999. URL: https://doi.org/10.1007/3-540-48959-2_11, doi:10.1007/3-540-48959-2_11.
- 10 Peter Dybjer and Anton Setzer. Indexed induction-recursion. *J. Log. Algebraic Methods Program.*, 66(1):1–49, 2006. URL: <https://doi.org/10.1016/j.jlap.2005.07.001>, doi:10.1016/j.jlap.2005.07.001.
- 11 Neil Ghani, Lorenzo Malatesta, and Fredrik Nordvall Forsberg. Positive inductive-recursive definitions. *Log. Methods Comput. Sci.*, 11(1), 2015. URL: [https://doi.org/10.2168/LMCS-11\(1:13\)2015](https://doi.org/10.2168/LMCS-11(1:13)2015), doi:10.2168/LMCS-11(1:13)2015.
- 12 Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without K. *Proc. ACM Program. Lang.*, 3(POPL):3:1–3:28, 2019. URL: <https://doi.org/10.1145/3290316>, doi:10.1145/3290316.
- 13 Daniel Gratzer, G. A. Kavvos, Andreas Nuyts, and Lars Birkedal. Multimodal dependent type theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 492–506. ACM, 2020. URL: <https://doi.org/10.1145/3373718.3394736>, doi:10.1145/3373718.3394736.
- 14 Robert Harper and Robert Pollack. Type checking with universes. *Theor. Comput. Sci.*, 89(1):107–136, 1991. URL: [https://doi.org/10.1016/0304-3975\(90\)90108-T](https://doi.org/10.1016/0304-3975(90)90108-T), doi:10.1016/0304-3975(90)90108-T.
- 15 Ambrus Kaposi, Simon Huber, and Christian Sattler. Gluing for type theory. In Herman Geuvers, editor, *4th International Conference on Formal Structures for Computation and Deduction, FSCD 2019, June 24-30, 2019, Dortmund, Germany*, volume 131 of *LIPIcs*, pages 25:1–25:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. URL: <https://doi.org/10.4230/LIPIcs.FSCD.2019.25>, doi:10.4230/LIPIcs.FSCD.2019.25.
- 16 Oleg Kiselyov. The design and implementation of BER metaocaml - system description. In Michael Codish and Eijiro Sumii, editors, *Functional and Logic Programming - 12th International Symposium, FLOPS 2014, Kanazawa, Japan, June 4-6, 2014. Proceedings*, volume 8475 of *Lecture Notes in Computer Science*, pages 86–102. Springer, 2014. URL: https://doi.org/10.1007/978-3-319-07151-0_6, doi:10.1007/978-3-319-07151-0_6.
- 17 Zhaohui Luo. Notes on universes in type theory. *Lecture notes for a talk at Institute for Advanced Study, Princeton* (URL: <http://www.cs.rhul.ac.uk/home/zhaohui/universes.pdf>), page 16, 2012.
- 18 Per Martin-Löf. An intuitionistic theory of types: predicative part. In H.E. Rose and J.C. Shepherdson, editors, *Logic Colloquium '73, Proceedings of the Logic Colloquium*, volume 80 of *Studies in Logic and the Foundations of Mathematics*, pages 73–118. North-Holland, 1975.
- 19 Erik Palmgren. On universes in type theory. In *Twenty-five years of constructive type theory*, volume 36 of *Oxford Logic Guides*, page 191 – 204. Oxford University Press, 1998.
- 20 Anton Setzer. Extending martin-löf type theory by one mahlo-universe. *Arch. Math. Log.*, 39(3):155–181, 2000. URL: <https://doi.org/10.1007/s001530050140>, doi:10.1007/s001530050140.
- 21 Jonathan Sterling. Algebraic type theory and universe hierarchies. *CoRR*, abs/1902.08848, 2019. URL: <http://arxiv.org/abs/1902.08848>, arXiv:1902.08848.
- 22 Amin Timany and Matthieu Sozeau. Cumulative inductive types in coq. In Hélène Kirchner, editor, *3rd International Conference on Formal Structures for Computation and Deduction, FSCD 2018, July 9-12, 2018, Oxford, UK*, volume 108 of *LIPIcs*, pages 29:1–29:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018. URL: <https://doi.org/10.4230/LIPIcs.FSCD.2018.29>, doi:10.4230/LIPIcs.FSCD.2018.29.
- 23 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. <https://homotopytypetheory.org/book>, Institute for Advanced Study, 2013.

- 694 24 A. N. Whitehead and B. Russell. Principia mathematica. *Revue de Métaphysique et de Morale*,
695 19(2):19–19, 1911.
- 696 25 Beta Ziliani and Matthieu Sozeau. A unification algorithm for coq featuring universe poly-
697 morphism and overloading. In Kathleen Fisher and John H. Reppy, editors, *Proceedings*
698 *of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP*
699 *2015, Vancouver, BC, Canada, September 1-3, 2015*, pages 179–191. ACM, 2015. URL:
700 <https://doi.org/10.1145/2784731.2784751>, doi:10.1145/2784731.2784751.