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 an unwieldy 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 version cumulativity as an optional feature for universes [5], and the upcoming 2.6.2 version will extend the $\omega + 1$ universe hierarchy to $\omega * 2$.
- Coq added support for cumulative inductive types [16], and added a form of bounded universe polymorphism [19].

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 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 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. 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 [7]. The Set_1 universe is not essential and we only use it as a convenience feature, in this paper and in the Agda formalization. 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 $(x : A) \to B$ with $\lambda\, x.\, t$ inhabitants, and Σ-types as $(x : A) \times B$, with pairing as $(t,\, u)$. We have $\top$ as the unit type with inhabitant tt, and $\bot$ as the empty type. Propositional identity is written as $t = u$ (coinciding with definitional equality).
- We occasionally use $\{x : A\} \to B$ 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) [6] consists of the following data:

- A category with a terminal object. We denote the set of objects as $\mathsf{Con} : \mathsf{Set}$ and use capital Greek letters starting from Γ to refer to objects. The set of morphisms is $\mathsf{Sub} : \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$, and we use σ, δ and so on to refer to morphisms. The terminal object is $\bullet$ with unique morphism $\epsilon : \mathsf{Sub}\,\Gamma\,\bullet$. 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 that we use here.
- A *family structure*, containing $\mathsf{Ty} : \mathsf{Con} \to \mathsf{Set}$ and $\mathsf{Tm} : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}$, 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 $- \triangleright - : (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$ together with an isomorphism $\mathsf{Sub}\,\Gamma\,(\Delta \triangleright A) \simeq ((\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \times \mathsf{Tm}\,\Gamma\,(A[\sigma]))$ 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* $-, - : (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Tm}\,\Gamma\,(A[\sigma])$. 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 $\mathsf{id} : \mathsf{Sub}\,(\Gamma \triangleright A)\,(\Gamma \triangleright A)$, we recover the *weakening substitution* $\mathsf{p} : \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$ and the *zero variable* $\mathsf{q} : \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}])$.
- By weakening q, we recover a notion of variables as De Bruijn indices. In general, the n-th De Bruijn index is defined as $\mathsf{q}[\mathsf{p}^n]$, where 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 $\mathsf{Nat} : \{\Gamma : \mathsf{Con}\} \to \mathsf{Ty}\,\Gamma$ and $\mathsf{Id} : \{\Gamma : \mathsf{Con}\}(A : \mathsf{Ty}\,\Gamma) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma$, we may write $\bullet \triangleright (n : \mathsf{Nat}) \triangleright (p : \mathsf{Id}\,\mathsf{Nat}\,n\,n)$ for a typing context, instead of using numbered variables or cwf combinators as in $\bullet \triangleright \mathsf{Nat} \triangleright \mathsf{Id}\,\mathsf{Nat}\,\mathsf{q}\,\mathsf{q}$.

▶ 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 $\Pi : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$ 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, Liftings 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 $(\mathsf{Tm}_0, \mathsf{Ty}_0)$ and $(\mathsf{Tm}_1, \mathsf{Ty}_1)$ 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* [10]) of cwfs: a weak morphism which has the identity action on the base category is exactly a family morphism. A plain family morphism is still not too interesting, so we add more features.

▶ **Definition 3.** A *lifting* is a family morphism which is bijective on terms. Note that we can drop the $\triangleright$-preservation condition from the specification of liftings, since it follows from the invertible action on terms.

▶ Notation 3. We write $\mathsf{Lift} : \mathsf{Ty}_0\,\Gamma \to \mathsf{Ty}_1\,\Gamma$ for the action of some lifting on types, $\uparrow$: $\mathsf{Tm}_0\,\Gamma\,A \to \mathsf{Tm}_1\,\Gamma\,(\mathsf{Lift}\,A)$ for the action on terms, and $\downarrow$ for the inverse action on terms.

We may think about the relation between *modalities* and liftings. The main difference is that we have 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 $\mathsf{Tm}\,(\mathsf{Id}\,\Gamma)\,A \simeq \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,A)$. Hence, every lifting can be viewed as a degenerate modality.

3.2.0.1 Two-level type theory

Assume family structures $(\mathsf{Ty}_0, \mathsf{Tm}_0)$ and $(\mathsf{Ty}_1, \mathsf{Tm}_1)$ and a lifting between them. This corresponds to a basic version of *two-level type theory* [1]. This theory can be interpreted as having $(\mathsf{Ty}_1, \mathsf{Tm}_1)$ as a metaprogramming layer which can generate object-level programs in the $(\mathsf{Ty}_0, \mathsf{Tm}_0)$ layer.

Lifted types correspond to types of object-level expressions; for example, $\mathsf{Bool}_0 : \mathsf{Ty}_0\,\Gamma$ is the object-level type of Booleans, while $\mathsf{Lift}\,\mathsf{Bool}_0$ is the meta-level type of Bool_0-expressions, and $\mathsf{Bool}_1 : \mathsf{Ty}_1\,\Gamma$ is the type of meta-level Booleans. It is possible to compute a $\mathsf{Lift}\,\mathsf{Bool}_0$ from a Bool_1. Given $b : \mathsf{Tm}_1\,\Gamma\,\mathsf{Bool}_1$, we can construct $\downarrow(\mathsf{if}\,b\,\mathsf{then}\uparrow\mathsf{true}_0\,\mathsf{else}\uparrow\mathsf{false}_0) : \mathsf{Tm}_0\,\Gamma\,\mathsf{Bool}_0$.

But there is no way to compute a Bool_1 from a $\mathsf{Lift}\,\mathsf{Bool}_0$, since $\mathsf{Lift}\,\mathsf{Bool}_0$ has no elimination rule in Ty_1. Conceptually, terms of $\mathsf{Lift}\,\mathsf{Bool}_0$ 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 eliminate 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.

Remarkably, the simple 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 [11], 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 4.** A *family inclusion* is a lifting 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.

Some examples for preservation equations for type and term formers:

$$\begin{aligned}
\mathsf{Lift}\,(\Pi\,(x : A)B) &= \Pi\,(x : \mathsf{Lift}\,A)(\mathsf{Lift}\,(B[x \mapsto \downarrow x])) \\
\uparrow(\lambda\,(x : A).\,t) &= \lambda\,(x : \mathsf{Lift}\,A).\uparrow(t[x \mapsto \downarrow x]) \\
\mathsf{Lift}\,\mathsf{Bool}_0 &= \mathsf{Bool}_1 \\
\uparrow \mathsf{true}_0 &= \mathsf{true}_1
\end{aligned}$$

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

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

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.

▶ **Definition 5.** A *strict family inclusion* between $(\mathsf{Tm}_0, \mathsf{Ty}_0)$ and $(\mathsf{Tm}_1, \mathsf{Ty}_1)$ is a family inclusion $(\mathsf{Lift}, \uparrow, \downarrow)$ for which the following equations hold:

$$(\Gamma \triangleright \mathsf{Lift}\,A) = (\Gamma \triangleright A) \tag{1}$$

$$\mathsf{Tm}_1\,\Gamma\,(\mathsf{Lift}\,A) = \mathsf{Tm}_0\,\Gamma\,A \tag{2}$$

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

A strict inclusion corresponds to Sterling's *algebraic cumulativity* [15]. The additional equations are a matter of convenience: they allow us to omit term liftings in informal syntax[1]. Most of the time we can also omit level annotations on term formers. For example, we have $\mathsf{true}_0 : \mathsf{Tm}_0\,\Gamma\,\mathsf{Bool}_0$, but also $\mathsf{true}_0 : \mathsf{Tm}_0\,\Gamma\,(\mathsf{Lift}\,\mathsf{Bool}_0)$, hence $\mathsf{true}_0 : \mathsf{Tm}_0\,\Gamma\,\mathsf{Bool}_1$. Moreover, true_0 is definitionally equal to true_1, since $\mathsf{true}_0 = \uparrow\mathsf{true}_0 = \mathsf{true}_1$. Thus, using simply true is fine whenever the family is clear from context.

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 [12] 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 5, we need $\uparrow$ together with equation 3 to identify term formers in different families. The other direction $\downarrow t = t$ is immediately derivable.

3.3 Level Structures

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.

▶ **Definition 6.** The *accessibility predicate* on relations is defined by the following inductive rules:

$$\mathsf{Acc} : \{A : \mathsf{Set}\}(R : A \to A \to \mathsf{Set}) \to A \to \mathsf{Set}$$

$$\mathsf{acc} : \{a : A\} \to ((a' : A) \to R\,a'\,a \to \mathsf{Acc}\,R\,a') \to \mathsf{Acc}\,R\,a$$

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

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

▶ **Lemma 1.** Accessibility is a mere proposition, i.e. all inhabitants of $\mathsf{Acc}\,R\,a$ are equal [18, Lemma 10.3.4].

▶ **Definition 7.** A relation $R : A \to A \to \mathsf{Set}$ is *well-founded* if $(a : A) \to \mathsf{Acc}\,R\,a$.

▶ **Definition 8.** A *level structure* consists of the following components:

$$\begin{aligned}
&\mathsf{Lvl} && : \mathsf{Set}_0\\
&- < - && : \mathsf{Lvl} \to \mathsf{Lvl} \to \mathsf{Set}_0\\
&< \mathsf{prop} && : (p\,q : i < j) \to p = q\\
&- \circ - && : j < k \to i < j \to i < k\\
&< \mathsf{wf} && : (i : \mathsf{Lvl}) \to \mathsf{Acc}\ <\ i
\end{aligned}$$

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

▶ **Definition 9.** A *family diagram* over Lvl maps each $i : \mathsf{Lvl}$ to a family structure $(\mathsf{Ty}_i,\ \mathsf{Tm}_i)$, and each $p : i < j$ to a family inclusion $(\mathsf{Lift}^j_i\,p,\ \uparrow^j_i\,p,\ \downarrow^j_i\,p)$ between $(\mathsf{Ty}_i,\ \mathsf{Tm}_i)$ and $(\mathsf{Ty}_j,\ \mathsf{Tm}_j)$. Moreover, the mapping is functorial, so $\mathsf{Lift}^k_i\,(p \circ q)\,A = \mathsf{Lift}^k_j\,p\,(\mathsf{Lift}^j_i\,q\,A)$, and similarly for $\uparrow^j_i\,p$ and $\downarrow^j_i\,p$. A *strict family diagram* is a family diagram where each inclusion is strict.

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

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

3.4 Universes

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

▶ **Definition 10.** A family diagram supports *universe formation* if it supports the following:

$$\begin{aligned}
&\mathsf{U} && : (i\,j : \mathsf{Lvl}) \to i < j \to \mathsf{Ty}_j\,\Gamma\\
&\mathsf{LiftU} && : \mathsf{Lift}^k_j\,p\,(\mathsf{U}\,i\,j\,q) = \mathsf{U}\,i\,k\,(p \circ q)
\end{aligned}$$

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

▶ **Definition 11.** A family diagram has *Coquand universes* if it has universe formation and additionally supports $\mathsf{El} : \mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p) \to \mathsf{Ty}_i\,\Gamma$, and its inverse $\mathsf{Code} : \mathsf{Ty}_i\,\Gamma \to \mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p)$.

▶ **Definition 12.** A family diagram has *Russell universes* if it has Coquand universes and additionally satisfies $\mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p) = \mathsf{Ty}_i\,\Gamma$ and $\mathsf{El}\,t = t$.

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 $\mathsf{El}\,t = t$ condition ensures appropriate naturality. If we only assumed $\mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p) = \mathsf{Ty}_i\,\Gamma$ but not Coquand universes, we would not be able to prove that a $t : \mathsf{Tm}_j\,\Gamma\,(\mathsf{U}\,i\,j\,p)$ 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 $\mathsf{Ty}_i\,\Gamma$ to a $\mathsf{Ty}_j\,\Gamma$, and a larger Ty is not isomorphic a smaller one, because it contains more universes. So, for example, lifting $\mathsf{Bool}_0 : \mathsf{Ty}_0\,\Gamma$ as a type to $\mathsf{Ty}_1\,\Gamma$ 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 $id : \Pi(A : \mathsf{U}\,i\,j\,p)(\mathsf{Lift}\,p\,(\mathsf{El}\,A) \to \mathsf{Lift}\,p\,(\mathsf{El}\,A))$ 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

$$(\lambda\,(b : \mathsf{Bool}_j).\,\mathsf{if}\,b\,\mathsf{then}\,\mathsf{Code}\,\top_i\,\mathsf{else}\,\mathsf{Code}\,\bot_i) : \mathsf{Tm}_j\,\Gamma\,(\mathsf{Bool}_j \to \mathsf{U}\,i\,j\,p).$$

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 13** (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 (Ty_i, 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 consistent[2].

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) [7] 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 [13] or Mahlo universes [14]? 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 14** (IR codes)**.** Assume $i : \mathsf{Lvl}$ and $f : (j : \mathsf{Lvl}) \to j < i \to \mathsf{Set}_0$. We define U^{IR} and $\mathsf{El}^{\mathsf{IR}}$ by simultaneous IR. For basic type formers, below 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).

$$\begin{array}{lll} \mathsf{U}^{\mathsf{IR}} & : \mathsf{Set}_0 & \mathsf{El}^{\mathsf{IR}} : \mathsf{U}^{\mathsf{IR}} \to \mathsf{Set}_0 \\ \mathsf{U}' & : (j : \mathsf{Lvl}) \to j < i \to \mathsf{U}^{\mathsf{IR}} & \mathsf{El}^{\mathsf{IR}}\,(\mathsf{U}'\,j\,p) \quad = f\,j\,p \\ \Pi' & : (A : \mathsf{U}^{\mathsf{IR}}) \to (\mathsf{El}^{\mathsf{IR}}\,A \to \mathsf{U}^{\mathsf{IR}}) \to \mathsf{U}^{\mathsf{IR}} & \mathsf{El}^{\mathsf{IR}}\,(\Pi'\,A\,B) = (a : \mathsf{El}^{\mathsf{IR}}\,A) \to \mathsf{El}^{\mathsf{IR}}\,(B\,a) \\ \bot' & : \mathsf{U}^{\mathsf{IR}} & \mathsf{El}^{\mathsf{IR}}\,\bot' \quad = \bot \\ \mathsf{Bool}' & : \mathsf{U}^{\mathsf{IR}} & \mathsf{El}^{\mathsf{IR}}\,\mathsf{Bool}' \quad = \mathsf{Bool} \end{array}$$

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

(U^{IR}, $\mathsf{El}^{\mathsf{IR}}$) 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:

$$\mathsf{U}_< : (i : \mathsf{Lvl})\{p : \mathsf{Acc}\ <\ i\} \to (j : \mathsf{Lvl}) \to j < i \to \mathsf{Set}_0$$
$$\mathsf{U}_<\,i\,\{\mathsf{acc}\,f\}\,j\,p = {\mathsf{U}^{\mathsf{IR}}}_{j\,(\mathsf{U}_<\,j\,\{f\,j\,p\})}$$

▶ **Definition 15** (Semantic universe)**.** Since every level is accessible, we can define the full semantic hierarchy and its decoding function.

$$\mathsf{U} : \mathsf{Lvl} \to \mathsf{Set}_0 \qquad \mathsf{El} : \{i : \mathsf{Lvl}\} \to \mathsf{U}\,i \to \mathsf{Set}_0$$
$$\mathsf{U}\,i = {\mathsf{U}^{\mathsf{IR}}}_{i\,(\mathsf{U}_<\,i\,\{<\mathsf{wf}\,i\})} \qquad \mathsf{El}\,\{i\} = {\mathsf{El}^{\mathsf{IR}}}_{i\,(\mathsf{U}_<\,i\,\{<\mathsf{wf}\,i\})}$$

▶ **Lemma 2.** Assuming $p : i < j$, we have the computation rule $\mathsf{U}_<\,j\,p = \mathsf{U}\,j$. Proof: we may assume that any witness for $\mathsf{Acc}\ <\ i$ is of the form $\mathsf{acc}\,f$ for some f. Then the equation becomes ${\mathsf{U}^{\mathsf{IR}}}_{j\,(\mathsf{U}_<\,j\,\{f\,j\,p\})} = {\mathsf{U}^{\mathsf{IR}}}_{j\,(\mathsf{U}_<\,j\,\{<\mathsf{wf}\,j\})}$, but by Lemma 1 the $f\,j\,p$ and $<\mathsf{wf}\,j$ witnesses are equal. ◀

▶ **Definition 16** (Semantic Lift)**.** We define by induction on U^{IR} a function with type $(p : i < j)(A : \mathsf{U}\,i) \to (A' : \mathsf{U}\,j) \times (\mathsf{El}\,A' = \mathsf{El}\,A)$. However, for the sake of clarity, we present this here as two (mutual) functions:

$$\mathsf{Lift} \quad : (p : i < j) \to \mathsf{U}\,i \to \mathsf{U}\,j$$
$$\mathsf{ElLift} : (p : i < j)(A : \mathsf{U}\,i) \to \mathsf{El}\,(\mathsf{Lift}\,A) = \mathsf{El}\,A$$

Let us look at Lift first:

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

Above, the Π' definition is well-typed by $\mathsf{ElLift}\,p\,A$. For the proof of ElLift, the only interesting case is U'. Here, we need show $\mathsf{U}_<\,j\,k\,(p \circ q) = \mathsf{U}_<\,i\,k\,q$, but by Lemma 2 both sides are $\mathsf{U}\,k$. ◀

▶ **Lemma 3.** Properties of Lift:

1. Lift preserves all basic type formers; this is immediate from the definition.
2. Lift is functorial, i.e. $\mathsf{Lift}\,(p \circ q)\,A = \mathsf{Lift}\,p\,(\mathsf{Lift}\,q\,A)$. This follows by induction on A, and we make use of the irrelevance of $- < -$ in the 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 17** (Base category)**.** This is simply the category of sets and functions in Set_0, i.e. $\mathbf{Con} = \mathsf{Set}_0$, $\mathbf{Sub}\,\Gamma\,\Delta = \Gamma \to \Delta$, and the terminal object is $\top$.

▶ **Definition 18** (Family diagram)**.** We map $i : \mathsf{Lvl}$ to a family structure as follows.

$$\mathbf{Ty}_i\,\Gamma = \Gamma \to \mathsf{U}\,i \qquad \mathbf{Tm}_i\,\Gamma\,A = (\gamma : \Gamma) \to \mathsf{El}\,(A\,\gamma)$$

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

$$\begin{aligned}
&\mathbf{Lift}^j_i p : \mathbf{Ty}_i\,\Gamma \to \mathbf{Ty}_j\,\Gamma\\
&\mathbf{Lift}^j_i p\,A = \lambda\,\gamma.\,\mathsf{Lift}^j_i p\,(A\,\gamma)
\end{aligned}$$

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

$$\mathbf{\Pi}\,A\,B = \lambda\,\gamma.\,\Pi'\,(A\,\Gamma)\,(\lambda\,\alpha.\,B\,(\gamma, \alpha)) \qquad \bot_i = \lambda\,\gamma.\,\bot' \qquad \mathbf{Bool}_i = \lambda\,\gamma.\,\mathsf{Bool}'$$

$\mathbf{Lift}^j_i p$ preserves type formers by Lemma 3. We define basic term formers and eliminators using metatheoretic features, e.g. $\mathbf{true}_i = \lambda\,\gamma.\,\mathsf{true}$ and $(\mathbf{\lambda}_i\,x.\,t) = \lambda\,\gamma\,\alpha.\,t\,(\gamma, \alpha)$. 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 $\uparrow^j_i p$ preserves term formers as well, so $(\mathbf{Lift}^j_i p, \uparrow^j_i p, \downarrow^j_i p)$ is a strict family inclusion.

We define universes as $\mathbf{U}\,i\,j\,p = \lambda\,\gamma.\,\mathsf{U}'_i\,j\,p$. With this, $\mathbf{Lift}^k_j\,p\,(\mathbf{U}\,i\,j\,q) = \mathbf{U}\,i\,k\,(p \circ q)$ follows by the definition of semantic Lift. The Russell universe equation $\mathbf{Tm}_j\,\Gamma\,(\mathbf{U}\,i\,j\,p) = \mathbf{Ty}_i\,\Gamma$ follows from Lemma 2, so we can define $\mathbf{El}$ and $\mathbf{Code}$ as identity functions.

▶ **Theorem 1** (Consistency of TTGU)**.** *There is no closed syntactic term of $\bot_i$ for any i.*

Proof. Assuming a syntactic $t : \mathsf{Tm}_i \bullet \bot_i$, we can interpret it in the previously given model, which yields an inhabitant of the metatheoretic $\bot$, 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 morphism 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 19.** A model of TTDL consists of the following.

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

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

   Additionally, 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 $l_0, l_1 : \mathsf{Lvl}\,\Gamma$ 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:

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

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

$$-[-] : \mathsf{Ty}\,\Delta\,i \to (\sigma : \mathsf{Sub}\,\Gamma\,\Delta) \to \mathsf{Ty}\,\Gamma\,(i[\sigma])$$

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

5. A lifting structure with

$$\mathsf{Lift} : \{\Gamma : \mathsf{Con}\}\{i\,j : \mathsf{Lvl}\,\Gamma\} \to i < j \to \mathsf{Ty}\,\Gamma\,i \to \mathsf{Ty}\,\Gamma\,j$$
$$\uparrow \;\; : \{\Gamma : \mathsf{Con}\}\{i\,j : \mathsf{Lvl}\,\Gamma\}(p : i < j) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,p\,A)$$

   Such that

   a. Lift preserves all basic type formers and has functorial action on $p \circ q$.

   b. $\uparrow$ has an inverse $\downarrow$, preserves all basic term formers and has functorial action on $p \circ q$.

   c. $(\Gamma \triangleright A) = (\Gamma \triangleright \mathsf{Lift}\,p\,A)$, and $\mathsf{Tm}\,\Gamma\,A = \mathsf{Tm}\,\Gamma\,(\mathsf{Lift}\,p\,A)$ and $\uparrow t = t$.

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

6. A universe structure

$$\mathsf{U} : \{\Gamma : \mathsf{Con}\}(i\,j : \mathsf{Lvl}\,\Gamma) \to i < j \to \mathsf{Ty}\,\Gamma\,j \qquad \mathsf{El}\text{:}\ \mathsf{Tm}\,\Gamma\,(\mathsf{U}\,i\,j\,p) \to \mathsf{Ty}\,\Gamma\,i$$

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

7. Basic type formers.

▶ **Definition 20** (IR model of TTDL)**.** Assume an external Lvl structure such that it supports $l_0, l_1 : \mathsf{Lvl}$ and $l_{01} : l_0 < l_1$ (the bootstrapping assumption). We again use the IR universe constructions from Section 4.1, instantiated to the assumed Lvl structure. We describe components of the model in order. Again, we write components of the model in **bold** font.

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

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

5.1 Level Reflection

▶ **Definition 21** (Level reflection rules)**.** Assume that we have definitions for internal levels in the syntax of TTDL, i.e. all of the following are defined:

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

A *reflection rule* for the above consists of

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

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 $\mathsf{Lvl} = (\mathsf{Nat}, -<-)$, 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 20, instantiated to the current level structure, satisfies level reflection. The model even supports the stricter $\mathsf{Tm}\,\Gamma\,\mathsf{Nat}_{l_0} = \mathsf{Lvl}\,\Gamma$ 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 internal levels strictly correspond to natural numbers, we may have

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

Here, we make sure that all types in the Π are on the same level. We assume that internal levels are in Nat_0, but we can bind an $l : \mathsf{Nat}_3$, because by cumulativity l is also a term of Nat_0.

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

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

This type is in $\mathsf{Ty}\,\Gamma\,\omega$, 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 $\mathsf{Nat} \to \mathsf{Lvl}$ 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 derivable[3].

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

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

Any model of TTFL can support the above rules: we can define $- \leq -$ and coerce by indexed induction-recursion [8], where we define coercion along $\mathsf{U}\leq$ by type lifting, and coercion along $\Pi\leq$ by backwards-forwards coercion.

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 [17], 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 $\Pi(i : \mathsf{Lift}\,?\,\mathsf{Lvl}^I)(A : \mathsf{U}\,(\mathsf{mk}_{\mathsf{Lvl}}\,i)\,?\,?) \to \mathsf{Lift}\,?\,A \to \mathsf{Lift}\,?\,A$. The issue is in $\mathsf{U}\,(\mathsf{mk}_{\mathsf{Lvl}}\,i)\,?\,?$, where we 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 internally type finite level polymorphism; this is what Agda 2.6.2 does.

[3] A level structure which is trichotonomous and supports *extensionality*, i.e. $(\forall i.\,(i < j) \iff (i < k)) \to j = k$, is a *type-theoretic ordinal*. Assuming excluded middle, type-theoretic ordinals are equivalent to classical ordinals [18, Section 10.3].

6 Related Work

Universes: Martin-Löf (U in U), Girard, Martin-Löf (countable predicative), CoC, Coq, Luo ECC. Universe poly in Agda (?), in Coq, cumulativity, Sterling, Coquand normalization (?)

Transfinite: super, higher-order, Mahlo, IR, indexed IR

Harper univ checking?

7 Conclusion and Future Work

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

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

This trade-off between convenience and formal minimalism is similar to the situation with inductive types. Formally, W-types and identity types are easier to handle than general 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 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) [10, 15] can be adapted to the theories in this paper. It seems that the main task in doing so is to define proof-relevant logical predicates over U^{IR}.

Second, we only focus on using universes as sized-based classifiers for types. Stratification features are also present in two-level type theory [1], modal type theories [9] or as h-levels in homotopy type theory [18]. 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.

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 research.

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 Mannaa, 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. Agda developers. Agda documentation, 2021. URL: https://agda.readthedocs.io/en/v2.6.1.3/.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. 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.
14. 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.
15. Jonathan Sterling. Algebraic type theory and universe hierarchies. *CoRR*, abs/1902.08848, 2019. URL: http://arxiv.org/abs/1902.08848, arXiv:1902.08848.

16 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.

17 Amin Timany and Matthieu Sozeau. Cumulative inductive types in coq. *LIPIcs: Leibniz International Proceedings in Informatics*, 2018.

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

19 Beta Ziliani and Matthieu Sozeau. A unification algorithm for coq featuring universe polymorphism and overloading. In Kathleen Fisher and John H. Reppy, editors, *Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming, ICFP 2015, Vancouver, BC, Canada, September 1-3, 2015*, pages 179–191. ACM, 2015. URL: https://doi.org/10.1145/2784731.2784751, doi:10.1145/2784731.2784751.