□Parametric Polymorphism for Agnostic Type Theories

Alexander Kuklev[1,2] ‹a@kuklev.com›
[1]Radboud University Nijmegen, Software Science
[2]JetBrains Research

Our starting point will be a type theory with a countable hierarchy of universes introduced by the following infinite family of rules:

$$\frac{}{\Gamma \vdash \mathtt{Type} : \mathtt{Type}^+} \qquad \frac{}{\Gamma \vdash \mathtt{Type}^+ : \mathtt{Type}^{++}} \qquad \frac{}{\Gamma \vdash \mathtt{Type}^{++} : \mathtt{Type}^{+++}} \qquad \ldots$$

Considered together, these rules introduce a countably infinite family of well-typed terms `Type`, `Type⁺`, `Type⁺⁺`, etc., but they have to be considered together as the type used in each rule is first introduced by the next rule.

Let us postulate the first universe `Type` to be ΠΣ-closed and add some basic types to taste:

$$\frac{}{\mathtt{Void} : \mathtt{Type}} \qquad \frac{}{\mathtt{Unit} : \mathtt{Type}} \qquad \frac{}{\mathtt{Bool} : \mathtt{Type}} \qquad \frac{}{\mathtt{Nat} : \mathtt{Type}}$$

$$\frac{\Gamma \vdash X : \mathtt{Type} \quad \Gamma, x : X \vdash Y(x) : \mathtt{Type}}{\Gamma \vdash (x : Y) \times Y(x) : \mathtt{Type}} \qquad \frac{\Gamma \vdash X : \mathtt{Type} \quad \Gamma, x : X \vdash Y(x) : \mathtt{Type}}{\Gamma \vdash \forall (x : Y)\ Y(x) : \mathtt{Type}}$$

(We will write `X → Y` for non-dependent Π-types `∀(_ : X) Y`.)

We want a type theory where all closed-form type former definitions `F(K : Type⁺) : Type`, can be lifted to the higher universes. The "closed-form" restriction is essential to avoid inconsistencies. So let's introduce the S4 necessity □-modality mapping types `T` to sets of their closed-form inhabitants `t : □T`. Naïvely, we can try the following rules:

$$\frac{\Box\Gamma \vdash x : X}{\Box\Gamma, \Delta \vdash x : \Box X}(\Box\mathtt{Intro}) \qquad \frac{\Gamma \vdash x : \Box X}{\Gamma \vdash x : X}(\Box\mathtt{Elim})$$

Here we say that an inhabitant definable in a context only containing closed-form expressions is a closed-form inhabitant, and that a closed-form inhabitant of `X` is an inhabitant of `X`.

Unfortunately, this definition is unsatisfactory in presence of dependent types. To proceed, we need to make our type theory `{0, ω}`-graded, that is we will allow marking some variables in contexts as opaque using zero subscripts above the colon. It will allow introducing parametric quantifiers `∀<x : X> T(x)` (note angle brackets instead of parens):

$$\frac{\Gamma \vdash X : \mathtt{Type} \quad \Gamma, x : X \vdash Y(x) : \mathtt{Type}}{\Gamma \vdash \forall \langle x : Y \rangle\ Y(x) : \mathtt{Type}} \qquad \frac{\Gamma \vdash X : \mathtt{Type} \quad \Gamma, x :^\circ X \vdash y : Y(X)}{\Gamma \vdash \{\ x :^\circ X \mapsto Y(X)\ \} : \forall \langle x : Y \rangle\ Y(x)}$$

But more importantly, it allows adjusting the rules for the □-modality to work well with dependent types. In the introduction rule we allow opaque variables, while in the elimination rule we state that a closed-form element can only depend on non-closed-form elements opaquely:

$$\frac{\Box\Gamma, \Delta^\circ \vdash x : X}{\Box\Gamma, \Delta^\circ, \Sigma \vdash x : \Box X}(\Box\mathtt{Intro}) \qquad \frac{\Gamma \vdash x : \Box X(t)}{\Gamma^\circ \vdash x : X(t)}(\Box\mathtt{Elim})$$

Now let us define the universe-shifting operator `( ⁺)` for all types. Its action on the types will be defined on a case-by-case basis for all type formers, i.e. coinductively. Types living inside the first universe `Type` are constructed without mentioning `Type`, so the universe-shifting operator does not affect them. Otherwise universe shifting is applied componentwise:

$$((x : Y) \times Y(x))^+ \mapsto (x : Y^+) \times (Y(x))^+$$

$$(\forall (x : Y)\ Y(x))^+ \mapsto \forall (x : Y^+) \times (Y(x))^+$$

Now we can finally introduce an infinite family of rules ensuring that closed-form type formers work in all universes above their original universe:

$$\frac{\Gamma \vdash K : \mathtt{Type}^{+} \quad \Gamma \vdash F : \Box(K \to \mathtt{Type})}{\Gamma \vdash F : K^{+} \to \mathtt{Type}^{+}}$$

$$\frac{\Gamma \vdash K : \mathtt{Type}^{++} \quad \Gamma \vdash F : \Box(K \to \mathtt{Type}^{+})}{\Gamma \vdash F : K^{+} \to \mathtt{Type}^{++}}$$

$$\frac{\Gamma \vdash K : \mathtt{Type}^{+++} \quad \Gamma \vdash F : \Box(K \to \mathtt{Type}^{++})}{\Gamma \vdash F : K^{+} \to \mathtt{Type}^{+++}}$$

$\ddots$

Closed-form type formers such as `List<T : Type> : Type` and `Endo<T : Type> := T → T` are now applicable to types in any universes. It also works for typeclasses[1] such as

```
data Monoid<this M : Type>(unit : M,
                           compose : M² → M,
                           ...axioms)

data Monad<this F : Type → Type>(    // Higher kinded typeclasses work too!
  unit<T>(x : T) : F<T>
  compose<X, Y>(x : F<X>, y : X → F<Y>) : F<Y>
  ...axioms)
```

(To deal with typeclasses more conveniently, let us introduce the following shorthand notation: given a typeclass `F : K → U`, let `∀<X : F> Y(X)` mean `∀<X : K> ∀(X : F<X>) Y(X)` where `X` can mean both the carrier `X : K` and the instance `X : F<T>`, disambiguated by the context.)

Let us now use parametric quantifiers for the cumulativity rules for terms of polymorphic types:

$$\frac{\Gamma \vdash K : \mathtt{Type}^{+} \quad \Gamma \vdash F : \Box(K \to \mathtt{Type}) \quad \Gamma \vdash c : \Box\forall\langle T : K\rangle\ F(T)}{\Gamma \vdash c : \forall\langle T : K^{+}\rangle\ F(T)} \ \ldots$$

Consider the following polymorphic function:

```
def id : ∀<T : Type> T → T
  x ↦ x
```

With the rule above, it additionally inhabits `Endo<T : Type⁺>`, `Endo<T : Type⁺⁺>`, etc.

Consider the canonical instance of the monoid typeclass and the Cayley Embedding construction:

```
object Endo<T> : Monoid<Endo<T>>
  unit = id<T>
  compose(f g : Endo<T>) = { x : T ↦ f(g(x)) }

def cayleyEmbedding : ∀<M : Monoid> MonoidMorphism<M, Endo(M)>
  ...
```

Definitions for small monoids apply to all monoids regardless of size through cumulativity, provided they are closed terms and only depend on carrier types parametrically.

The type theory we presented has a straightforward model in M. Shulman's set theory ZMC/𝕊. We interpret □-modality on types as relativization to small sets 𝕊, and since finite sets of formulas are reflected in 𝕊, relativized types contain closed terms. Reverse application of the reflection schema justifies polymorphism rules, while the Ackermann schema justifies cumulativity rules.

[1] Typeclasses are introduced as records with a marked (by `this`), possibly higher-kinded, typal parameter, but turn into a subtype of their marked parameter's type, e.g. $\mathtt{Bool}^{R}$ `<: Type`, so every `T : Bool`^{R} is both a type and an instance of $\mathtt{Bool}^{R}$`<T>`, which does not introduce ambiguities since types and families cannot have fields, while typeclass instances are records and consist from their fields. See `kotlin_typeclasses.pdf` for details.

1 Unary parametricity

We have achieved that `id := { x ↦ x }` inhabits `Endo<T>` in all universes, but we can also extend our type theory so we can show that `id` is the only closed-form inhabitant of `∀<T> Endo<T>` up to equivalence. The □-modality together with $(\ ^{d})$ operator from Displayed Type Theory allows □-internal parametric reasoning. As opposed to type theories with non-modal internal parametricity, this approach does not contradict LEM (law of excluded middle) maintaining the underlying type theory constructively agnostic.

In 1941, Alonzo Church noticed that natural numbers can be represented as polymorphic functions of the type `∀<T> (T → T) → T → T`. All other inductive types also have Church encodings, and the type of `id : ∀<T> (T → T)` is the Church encoding for the `Unit` type. To establish that `id` is its unique closed-form inhabitant, it is enough to postulate that closed-form inhabitants of Church encoded inductive datatypes are exhausted by Church encodings.

Every inductive type `I : U` comes with a dual typeclass $\texttt{I}^{R}\texttt{<T : U>}$. For natural numbers:

```
data ℕᴿ<this T : U>(base : T
                    next : T → T)
```

Instances of $\texttt{I}^{R}$ define structural recursion for the type `I`. Church encodings $\texttt{x}^{c}$ `: ∀<T : I`^{R}`> → T` evaluate instances of $\texttt{I}^{R}$ for `x`:

```
def 0ᶜ = { T :° U, T : ℕᴿ<T> ↦ T.base }
def 1ᶜ = { T :° U, T : ℕᴿ<T> ↦ T.next T.base }
def 2ᶜ = { T :° U, T : ℕᴿ<T> ↦ T.next (T.next T.base) }
...
```

Both the original and the Church-encoded inductive type form an instance of the typeclass $\texttt{I}^{R}$:

```
object ℕ : ℕᴿ
  def base = 0
  def next = (1 +)

object ℕᶜ : ℕᴿ
  def base = 0ᶜ
  def next = (1 +)ᶜ
```

To postulate that the instance `ℕ` is the initial model, we need to introduce the induction rule (that is, the dependent elimination rule) for `ℕ`. Ensuring that closed-form inhabitants of `ℕᶜ` are exhausted by Church encodings of `ℕ` elements is essentially the same rule, but for the type `□ℕᶜ` instead of `ℕ`. To formulate both rules uniformly for all inductive types, let us apply the $(\ ^{d})$ operator introduced in Displayed Type Theory to the typeclass $\texttt{I}^{R}$:

```
data ℕᴿᵈ<T : U>(M : ℕᴿ<T>)<this Ts : T → U>(base : T
                                            next : T → T)
```

Now we can formulate induction and unary parametricity:

```
I-ind(n : I) : ∀<M : Iᴿᵈ I> → (M n)
```

```
I-par(n : □Iᶜ) : ∀(R : Iᴿᵈ Iᶜ) → (R n)
```

We can use `Unit-par` to derive that `id` is the unique closed-form inhabitant of `∀<T> (T → T)`:

```
data Unitᴿ<this T>(point : T)  // Dual of the `Unit` type are pointed types

object Helper : Unitᴿᵈ<Unitᶜ> { id : Unitᶜ ↦ (id ≃ { x ↦ x }) }
  def point = refl

theorem ∀(id : □∀<T : U> T → T) id ≃ { x ↦ x }
  id ↦ Unit-par id Helper
```

Ideas we present in "Reedy Types and Dependent Type Families"[2] indicate a direction beyond unary parametricity towards `S`-shaped parametricity for any effectively presentable Reedy category `S`. In addition to well-known applications of binary (Wadler's "Theorems for Free") and n-ary parametricity, it enables extending polymorphism and cumulativity beyond `upwards-only` allowing to generalize from `U` to presheaf universes `J → U` and `(J → U⁺) → U`.

2 Classical reasoning and functional logic programming

In the companion paper[3] we argue that it is also possible to introduce a modality dual to $\Box$, namely the S4-possibility modality mapping each type `T` to the spectrum `◇T` of its formal inhabitants, i.e., inhabitants that can "non-constructively shown to exist" using choice operator (as in Lean4) and double negation elimination as its special case. This modality allows classical (non-constructive) reasoning within `◇`-fragment without compromising computational properties of the underlying type theory such as canonicity, normalization and decidability of type checking, as well as its compatibility with univalence. These desirable properties remain intact even if impermeability of the `◇`-fragment is violated by computational Markov Principle (CMP) that allows evaluating Turing-complete computations given a closed-form classical proof of their non-divergence:

$$\frac{\texttt{c : Computation<T>} \quad \texttt{nonDivergence : □◇(c ≠ ⊥)}}{\texttt{eval(c, nonDivergence) : T}}$$

We can easily show that the type `M = W(T : U) ◇T` of iterative classical sets satisfies all axioms of the classical set theory ZFC, and the reasoning under `◇`-modality satisfies Hilbert axioms of classical logic, thus our proposed type theory contains a model of ZFC.

3 Relation to Shulman's strong Feferman set theory `ZMC/`$\mathbb{S}$

For any fixed finite collection of type formers `Fₙ : (Kₙ → U) → U` where `Kₙ : U⁺` induction recursion allows constructing a universe `V : U` containing codes for all types `T : U` that can be generated using type formers `Fₙ`. Let us assume that all our universes `Type : Type⁺ : Type⁺⁺ : Type⁺⁺⁺ : ⋯` contain all such inductive-recursive types and additionally that there is an impredicative universe `Prop : Type` of propositions.

Resulting type theory provides the same level of comfort regarding formalization of category theory as the strong Feferman set theory `ZMC/`$\mathbb{S}$ introduced by M. Shulman.[4] Moreover, it is, presumably, a conservative extension thereof.

There are two ways to interpret `ZFC`-sets into the type theory:

- The type of small iterative classical sets `M := W(T : Type) ◇T` is a model of ZFC.
- Its impredicative encoding inside `Prop`: `M' ≜ W(P : Prop) ◇P` is also a model of ZFC.

Let us translate the language of `ZMC/`$\mathbb{S}$ into the type theory interpreting $\mathbb{S}$-bounded quantification with `M` and unbounded quantification with `M'`. So far, we have manually checked that this translation satisfies all axioms of `ZMC/`$\mathbb{S}$. At the same time, it seems possible to exhibit a construction that yields `ZMC/`$\mathbb{S}$-valued models for finite fragments of this type theory built in such a way that the type-theoretic translation of the set-theoretic formula `φ` is only inhabited if the formula `φ` holds in the model.

Conservativity implies equiconsistency, so it should be possible to adapt the consistency-dependent canonicity proof for $\texttt{CC}_{\texttt{obs}}$ by L. Pujet and N. Tabareau[5] to show desirable computational properties claimed above.

[2] https://akuklev.github.io/reedy.pdf
[3] https://akuklev.github.io/verse
[4] https://arxiv.org/pdf/0810.1279
[5] https://dl.acm.org/doi/10.1145/3498693

4 Relation to Stratified Type Theory

In 2024, J. Chan and S. Weirich devised a stratified type theory `StraTT`,[6] a logically consistent type theory that allows speaking of the all-encompassing universe `Type : Type` by stratifying typing judgments. This approach parallels New Foundations, a non-well-founded set theory developed by W. V. Quine in 1937, the only successful foundational theory able to speak of all-encompassing self-containing universal objects, which was recently shown to be consistent using Lean proof assistant.[7] Stratified type theory can probably be extended to admit the ω-category of all ω-categories, and used to explore the boundaries of what can be said about such an object. We think it is crucial to understand the relation between `StraTT` and our approach.

[6] https://arxiv.org/pdf/2309.12164
[7] https://arxiv.org/abs/1503.01406