Universe Subtyping in Martin-Löf Type Theory
Internship Report

Gabriel Scherer
under the supervision of Andreas Abel

Revised version: August 24, 2011

The pragmatic goal of this internship was to establish the theoretical soundness of a particular feature of the Agda proof assistant, the so-called *universe polymorphism*. This is part of a series of work by my supervisor Andreas Abel [ACD07] aiming to provide a coherent metatheory for the different features implemented or desired in this proof assistant. More precisely, we first considered universe *cumulativity*, a well-known feature for universe hierarchy, then universe *polymorphism*, which is a less explored area, and finally attempted to add *irrelevance* to the type system, to ease universe levels manipulations.

All those notions will be introduced in more detail in the first section. I will highlight the main goals and constraints of the desired theory, in comparison to related works. I will also have to discuss some of the technical choices and issues we have faced; those technical decisions have large consequences – often unforeseen – on the technical development and its structure.

The second section will describe the formal type system – or type systems, as we will consider variants – used throughout the report. To this *declarative* type system corresponds an *algorithmic* system that provides a concrete implementation and decision procedure. We will insist on the "close to the metal" aspect of these algorithms, that do not afford superfluous checks to ease the metatheoretic study.

Building the metatheory of a dependent type system is a lot of technical work, but most of the presentation is reusable for other systems having similar features. In particular, we have focused in this internship on proof techniques using *logical relations* [Gog00, VC02], a very important concept that I will try to expose in the third section, as independently of our specific details as possible.

We will see in the fourth section how logical relations give us the important meta-theoretic results, including subject reduction of the declarative system, and termination, soundness and completeness of the algorithmic system, providing a decidability result for the whole theory.

The fifth section will be dedicated to an exploration of the type system features considered in this work. First universe polymorphism, then the rougher parts, irrelevant function spaces and heterogeneous equality. They bring an additional layer of complexity that would have obscured the previous developments if presented simultaneously.

Space restrictions, proofs, related work and appendix Due to the quite severe space constraints imposed on this report, we have made unsatisfying compromises. When given the choice between sacrifying informal explanations and formal proofs, we have kept the explanations. All the lemmas we used in our formal development are given, so it is still possible to follow the construction; moreover, a few particularly relevant proof details are given. The detailed proofs are available in an unpublished article [AS11] containing the full proofs of the core metatheory

— without cumulativity or universe polymorphism. This article also contains a full, detailed discussion of related work, thanks to the vast field knowledge of the supervisor; we have kept the set of citations in the present document smaller, intentionally including only the references that we had some time to study during the internship.

Finally, as a last offering to the dragon of page number reduction, we have moved to the appendix both our informal discussion of the logical relation definition – which the reader knowledgeable in Kripke logical relation can easily skip – and our fifth section, describing the less well established parts of our theory, universe polymorphism, level irrelevance and heterogeneous equality. While maybe representing the most novel part of the work, they are expected to change considerably during the last weeks of the internship which, at the time of writing, is not finished.

1 Goals, challenges, related works and design space

Situation The type theory we work in is essentially Martin-Löf Type Theory, as present in the proof assistant Agda. The main distinguishing feature, in particular with respect to the Calculus of Constructions (CoC), is the use of a typed equality including η-equivalence. No part of the system is impredicative, in constrast to System F polymorphism or Coq's Prop universe; we only have a family of predicative universes Set_i. As for the CoC, it is presented without stratification between terms, types and sorts. Here are the classic syntax and the basic rules, of which we will consider some variants.

$$
\begin{array}{lll}
\mathsf{Var} \ni x, y, X, Y & & \\
\mathsf{Sort} \ni s & ::= \mathsf{Set}_k \quad (k \in \mathbb{N}) & \text{universes} \\
\mathsf{Exp} \ni t, u, T, U & ::= s \mid (x{:}U) \to T & \text{sort, function type} \\
& \quad \mid x \mid \lambda x{\star}U.\, t \mid t\ u & \text{lambda-calculus} \\
\mathsf{Cxt} \ni \Gamma, \Delta & ::= \diamond \mid \Gamma.\, x{:}T & \text{empty context, context extension}
\end{array}
$$

Context validity judgement $\Gamma \vdash$

$$
\frac{}{\diamond \vdash} \qquad \frac{\Gamma \vdash \quad \Gamma \vdash T : s}{\Gamma.\, x{:}T \vdash}
$$

Typing judgement $\Gamma \vdash t : T$

$$
\frac{\Gamma \vdash}{\Gamma \vdash \mathsf{Set}_i : \mathsf{Set}_{i+1}} \qquad \frac{\Gamma \vdash U : s \quad \Gamma.\, x{:}U \vdash T : s}{\Gamma \vdash (x{:}U) \to T : s}
$$

$$
\frac{\Gamma \vdash \quad (x{:}U) \in \Gamma}{\Gamma \vdash x : U} \qquad \frac{\Gamma \vdash U : s \quad \Gamma.\, x{:}U \vdash t : T}{\Gamma \vdash \lambda x{:}U.\, t : (x{:}U) \to T} \qquad \frac{\Gamma \vdash t : (x{:}U) \to T \quad \Gamma \vdash u : U}{\Gamma \vdash t\ u : T[u/x]}
$$

$$
\frac{\Gamma \vdash t : T \quad \Gamma \vdash T = T'}{\Gamma \vdash t : T'}
$$

The last rule is a *conversion* rule relying on a type equality judgement $\Gamma \vdash T = T'$, which is here left undefined. This the main variability point of the theory, and it concentrates all the metatheoretic difficulties, the typing rules being otherwise quite natural — and syntax-directed. We will discuss several possible designs for our equality judgement, and the vast differences in metatheory development they entail.

Here are some standard examples for the enjoyment of the reader — where the abbreviation $U \to T$ is understood as $(x : U) \to T$ with x not free in T; note that $\to$ is right associative.

$$\begin{array}{lll} \mathsf{id} & : & (X : \mathsf{Set}_0) \to X \to X \\ \mathsf{id} & := & \lambda X : \mathsf{Set}_0.\, \lambda x : X.\, x \\ \mathsf{Nat} & : & \mathsf{Set}_1 \\ \mathsf{Nat} & := & (X : \mathsf{Set}_0) \to (X \to X) \to X \to X \end{array}$$

Goal: close the gap between formal description and actual implementation We will produce an algorithmic system corresponding to this declarative system, suitable for a practical implementation in a proof assistant. This is inspired in particular by the typed algorithmic equality check of Harper and Pfenning [HP05], but we claim a practical improvement in the removal of a specific equality check that was known to be useless in practice, but necessary to establish the desired metatheoretic results.

Goal: an approach that scales to large elimination The typed equality algorithm of Harper and Pfenning crucially operates on *erased* version of the types, as type dependencies would make the transitivity of the algorithm problematic. This is acceptable in their Logical Framework as typed equality relies only on the *shape* of types, not on their term subparts. In contrast, we wish to allow *large eliminations*, where the shape of types may depend on a value, using for example the following operator constructing types of different shapes from boolean terms:

$$\begin{array}{lll} \mathsf{T} & : & \mathsf{Bool} \to \mathsf{Set}_0 \\ \mathsf{T}\ \mathsf{true} & := & \mathsf{Bool} \to \mathsf{Bool} \\ \mathsf{T}\ \mathsf{false} & := & \mathsf{Bool} \end{array}$$

We therefore have to abandon erasure, and design an algorithm directed by fully dependent types. The price to pay is a more difficult metatheoretic study. For example, completeness of the algorithmic equality is proved, as is customary since the seminal work of Coquand [Coq91], using a logical relation; but while Harper and Pfenning only need to define their logical relation on simple, erased types, we have to define a logical relation on fully dependent types, making the well-foundedness arguments more delicate.

While we are not concerned in this report with inductive datatypes and related powerful extensions to the type theory, we expect our development and metatheoretic results to scale to those settings.

Goal: universe cumulativity In counterpart to their reassuringly understandable model constructions, predicative theories are less comfortable to use than the impredicative ones. When doing type-level programming, one often wish to use type polymorphism, in the spirit of system F. But quantification on Set_0 is a poor palliative that quickly shows its limits. Suppose for example one wished to define the type $\mathsf{List}\ n\ T$ of lists of length n and element type T, using the previously given type of Church numerals, a binary product type $\mathsf{Prod} : \mathsf{Set}_0 \to \mathsf{Set}_0 \to \mathsf{Set}_0$ and a unit type $\mathsf{Unit} : \mathsf{Set}_0$:

$$\begin{array}{lll} \mathsf{List} & : & \mathsf{Nat} \to (X : \mathsf{Set}_0) \to \mathsf{Set}_0 \\ \mathsf{List}\ n\ X & := & n\ \mathsf{Set}_0\ (\mathsf{Prod}\ X)\ \mathsf{Unit} \end{array}$$

Unfortunately, this definition cannot work as Nat quantifies over types in Set_0, to which Set_0 itself doesn't belong. We can change the definition of Nat to quantify

on Set_1, making this last example work, but then we cannot use it at level Set_0 anymore.

Cumulativity helps with this problem, by making it possible to lift elements of an universe to all higher universes. For example, if $\Gamma \vdash t : \mathsf{Set}_0$ holds, we also want $\Gamma \vdash t : \mathsf{Set}_1$ to hold. This is done by a *cumulativity relation* $\preccurlyeq$ at the type level, which is a restricted form of subtyping on universes, conveniently extended to function types: for example we have $\Gamma \vdash \mathsf{Set}_0 \preccurlyeq \mathsf{Set}_1$ and $\Gamma \vdash \mathsf{Set}_1 \to \mathsf{Set}_1 \preccurlyeq \mathsf{Set}_0 \to \mathsf{Set}_2$.

Cumulativity was discussed in depth in Luo's thesis [Luo90], and implemented in the proof assistants Lego, Plastic and Coq. We have successfully ported Luo's definition of cumulativity to our setting, resulting in a typed cumulativity relation — in contrast to the untyped definition in the context of Extended Calculus of Constructions. Contrarily to most presentation after Luo's, our cumulativity relation is contravariant, rather than invariant, in the domain of the function types.

Interestingly, such a type system with cumulativity was described in Ulf Norell's PhD thesis [Nor07], which laid the ground for the Agda2 system. But the metatheory of this system has not been developped, and current versions of Agda do not support cumulativity, relying on a relaxation of the function type rule to get some of its flexibility:

$$\frac{\Gamma \vdash U : \mathsf{Set}_i \quad \Gamma.\, x{:}U \vdash T : \mathsf{Set}_j}{\Gamma \vdash (x{:}U) \to T : \mathsf{Set}_{\max(i,j)}}$$

Goal: universe polymorphism Once type cumulativity is enabled, it is sufficient for our former example to define Nat as taking a type in universe Set_0, and use it at both levels Set_0 and Set_1. But the problem may re-appear, say, at level Set_2.

The practical solution usually suggested is to define such polymorphic operators at a "sufficiently high level", say Set_8, which is enough for all but the most insane uses of type-level computations. While possibly sufficient in practice, this method is clearly unsatisfying, and various techniques have been suggested to handle universe levels in a less ad-hoc manner.

In particular, Harper and Pollack [HP91] have imitated the principle, well-known among mathematicians building predicative *proof* theories [Fef04], of *typical ambiguity*: one does not give the explicit universe level used, e.g., by writing Set instead of Set_i, and the system infers a suitable level by itself. This model scales better than picking a "large enough" level, as code changes that introduce new level constraints do not necessitate to update levels by hand, they are re-inferred accordingly. They also propose to control the instantiation of those constraints at a "definition" level, allowing ML-flavoured polymorphism, where different uses of the same definition can make independent level instantiations.

The proof assistant Coq implements such a *typical ambiguity* for its predicative hierarchy Type_i. Level constraints are collected are resolved at the module level. This is a satisfying solution, and it succeeds in hiding the predicative subtilities to the user, which for all practical purposes can live in the convenient $\mathsf{Type} : \mathsf{Type}$ world.

However, Courant [Cou02] noted some insufficiences of the system at the cross-module scale. In essence, if the internal level constraints are not visible in a module *interface*, it is not possible to check the typability of some code using a module without referring to the module *implementation*, breaking separate checking. He suggests making the universe level explicit again, as well as the inequality constraints between them, to be exposed in the module signature.

Agda exhibits explicit universe polymorphism: $(i : \mathsf{Level}) \to (X : \mathsf{Set}_i) \to X \to X$ is the type of the identity function polymorphic in its type universe level. It's still

predicative as such a term lives in a higher universe, Set_ω, on which Level doesn't quantify. This can however prove inconvenient when mixing parameters of unrelated types, as we need to pass several level parameters around and repeatedly use their least upper bound.

Other options are possible, for example a universe-lifting operation – the idea being that typability is preserved by lifting all the universe levels occuring in a term by a constant – as exposed in some predicative mathematical theories [Fef04], or informally suggested by Conor McBride [McB11]. It's unclear to us how much those solutions differ from quantifying everything over a "shifting base" level i and using levels Set_{i+k} for constant k instead of the previous Set_k; some experimentation with the different solutions would be helpful.

Our present goal is to perform the much awaited metatheoretic study of Agda's universe polymorphism, and evaluate the practical benefits of having added cumulativity. We have not yet reached a definitive conclusion, and more experiments are needed to evaluate the strengths and weaknesses of this point of the design space.

Goal: Universe level irrelevance Finally, we hope to build on previous work of Abel [Abe11] to enable some form of "irrelevance" for universe levels. The idea behind level irrelevance is to avoid asking the user to provide equality proofs between two similar types using different instantiations of the level variables.

Irrelevance is a ongoing research topic in dependently typed calculi. The Coq system has a distinct universe Prop for "propositions and their proofs" that allows external erasure: it is possible to extract/compile a closed Coq program into an other executable language, erasing the Prop fragments of the Coq program during the translation. This erasability also protects the rest of the system from the convenient impredicativity of the Prop universe.

Recently, more flexible approaches have been proposed, where we do not distinguish a separate erasable universe, but use instead an *irrelevant* function space, where the argument can be used as a proof, but not in a computationally relevant position; this allows to ignore terms applied in irrelevant positions during equality checking, providing *internal* erasure. This has been developed for the Logical Framework by Pfenning [Pfe01] and Reed [Ree03], extended to the Calculus of Constructions by Miquel [Miq01], Mishra-Linger and Sheard [MLS08], and Barras and Bernardo [BB08].

The naive idea is that we would like a "level-polymorphic" universe constructor $\mathsf{PSet} : (i \div \mathsf{Level}) \to \mathsf{Set}_{i+1}$ that is irrelevant in its level parameter i, so that polymorphic identity $\lambda i \div \mathsf{Level}.\, \lambda X : \mathsf{PSet}\ i.\, \lambda x : X.\, x$ would be level-irrelevant, and identity at different levels be considered equal. However, this cannot work as it would also imply equality between all equality levels, e.g., $\mathsf{PSet}\ 0$ and $\mathsf{PSet}\ 1$, collapsing the whole hierarchy to a single inconsistent universe. We need a finer notion of irrelevance, which we will investigate in the last section of this report.

A difficult technical choice: homogeneous or heterogeneous equality One crucial difficulty with dependent type systems is the *asymmetry* of the classic compositional rule for equality of function applications:

$$\frac{\Gamma \vdash f = f' : (x:U) \to T \qquad \Gamma \vdash u = u' : U}{\Gamma \vdash f u = f' u' : T[u/x]}$$

If both the functions and the arguments are equal, then the application of the functions to the arguments are equal. The problem is in the type of this term equality: as the return type T of the dependent function space $(x:U) \to T$ depends

on the argument x, we have to pick either $T[u/x]$ or $T[u'/x]$, and are forced to make an arbitrary asymmetric choice.

This makes the metatheory of such a definitional equality harder. For example, the *syntactic validity* property, according to which $\Gamma \vdash t = t' : T$ implies both $\Gamma \vdash t : T$ and $\Gamma \vdash t' : T$, is not immediate: we have $\Gamma \vdash f'u' : T[u'/x]$ but not directly $\Gamma \vdash f'u' : T[u/x]$. A very similar problem appears in the algorithmic equality, and is the reason for Harper and Pfenning to erase type dependencies, obliterating this difficulty.

Another possibility is to instead use an *heterogeneous* equality judgement of the form $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$. In heterogeneous equalities, both context and result types may vary, but the contexts have the same domain, and their types are pairwise definitionally equal. The application rule becomes the heavy but symmetrical

$$\frac{\Gamma \vdash f : (x{:}U) \to T = \Gamma' \vdash f' : (x{:}U') \to T' \qquad \Gamma \vdash u : U = \Gamma' \vdash u' : U'}{\Gamma \vdash fu : T[u/x] = \Gamma' : f'u' : T'[u'/x]}$$

With this heterogeneous presentation, the asymmetry is avoided and syntactic validity, as well as transitivity of a the corresponding heterogeneous algorithmic equality, are easy to obtain. There is no free lunch, however, as the use of heterogeneous equality raises difficulties later in the technical development, to establish the transitivity of the logical relation.

At the very beginning of this internship, we had an heterogeneous equality, inspired by previous work of Abel [Abe11], but when faced with this unexpected technical difficulty we had to retreat to the more familiar terrain of homogeneous equality. We believe to have overcome the previous difficulties with the asymmetries of homogeneous equality. Instead of using one logical relation to establish the metatheory of both the definitional and algorithmic systems, we use *two* different logical relations, the first one on top of definitional equality, which allows to prove the non-trivial transitivity – and therefore soundness – of the algorithmic system, and the second one on top of the algorithmic equality, finally proving its completeness.

Unfortunately, the introduction of level irrelevance may disturb this compromise. If we want to allow equality between types living at different levels, an heterogeneous equality becomes again useful, and perhaps even necessary. We have therefore investigated heterogeneous equality again, and we believe to have disarmed some of the previous technical difficulties.

There is no clear consensus yet on which approach is better. We will try to expose the main ideas of both developments, under the space constraints. The rigorous reader may skip the development of level irrelevance and heterogeneous equality, that are still in infancy and may yet yield unpleasant surprises; we claim that the rest of the metatheoretic work, in particular the combination of homogeneous equality and two logical relations to work on universe cumulativity, is solid.

Aside: we have attempted a mechanical formalization, in Coq, of the core technical argument — the logical relation to establish algorithmic soundness. This is ridden with technical difficulties; in particular, our paper proofs perform an universe construction by an inductive-recursive definition, which is not supported by Coq and difficult to encode conveniently. Despite building on previous work – Barras's Coq En Coq [BW97] – unfortunate technical choices on our part, e.g., in the representation of weakening, have made the development challenging and time consuming. We have not been able to mechanize a satisfying portion of the development, but still believe that computer-assisted proofs have a role to play in those increasingly treacherous technical arguments.

2 Definitional and Algorithmic equality

In this section, we present in more detail the definitional and algorithmic equalities, along with the cumulativity relation — and its algorithmic counterpart.

2.1 Definitional equality

Below are the rules of definitional equality. They are separated in four groups.

Computation (β) and extensionality (η)

$$\frac{\Gamma.\, x{:}U \vdash t : T \qquad \Gamma \vdash u : U}{\Gamma \vdash (\lambda x{:}U.\, t)\, u = t[u/x] : T[u/x]} \qquad \frac{\Gamma \vdash t : (x{:}U) \to T}{\Gamma \vdash t = \lambda x{:}U.\, t\, x : (x{:}U) \to T}$$

Those are the two most important rules of typed equality. β-equality carries the "computational meaning" of equalities, while η-equality express the observational nature of the system. Along with the λ-compatibility rule given later, it is equivalent to the following rule that highlights the extensional nature of function equality:

$$\frac{\Gamma.\, x{:}U \vdash t\, x = t'\, x : T}{\Gamma \vdash t = t' : (x{:}U) \to T}$$

Equivalence rules are not syntax-directed; in particular, transitivity is the hard rule to remove when moving from a definitional to an algorithmic presentation.

$$\frac{\Gamma \vdash t : T}{\Gamma \vdash t = t : T} \qquad \frac{\Gamma \vdash t = t' : T}{\Gamma \vdash t' = t : T} \qquad \frac{\Gamma \vdash t_1 = t_2 : T \qquad \Gamma \vdash t_2 = t_3 : T}{\Gamma \vdash t_1 = t_3 : T}$$

Compatibility rules are compositional rules expressing the equality of two terms as a combination of equality on their subterms. They are syntax-directed, so we will find somewhat similar-looking rules in the algorithmic presentation.

$$\frac{\vdash \Gamma}{\Gamma \vdash s = s : s + 1} \qquad \frac{\Gamma \vdash U = U' : s \qquad \Gamma.\, x{:}U \vdash T = T' : s}{\Gamma \vdash (x{:}U) \to T = (x{:}U') \to T' : s}$$

$$\frac{(x{:}U) \in \Gamma \qquad \vdash \Gamma}{\Gamma \vdash x = x : U} \qquad \frac{\Gamma \vdash U = U' : s \qquad \Gamma.\, x{:}U \vdash T : s \qquad \Gamma.\, x{:}U \vdash t = t' : T}{\Gamma \vdash \lambda x{:}U.\, t = \lambda x{:}U'.\, t' : (x{:}U) \to T}$$

$$\frac{\Gamma \vdash t = t' : (x{:}U) \to T \qquad \Gamma \vdash u = u' : U}{\Gamma \vdash t\, u = t'\, u' : T[u/x]}$$

Conversion rule

$$\frac{\Gamma \vdash t = t' : T \qquad \Gamma \vdash T = T'}{\Gamma \vdash t = t' : T'}$$

Note that the absence of the expected "easy syntactic lemmas" here is not only due to space constraints: there are not easy syntactic lemma. We only prove that a weakening result, by syntactic induction.

Definition 1 (Weakening) *We will note $\Gamma \leq \Delta$ when the valid context Γ is a weakening of the valid context Δ: all the bindings of Δ also belong to Γ.*

Lemma 2 (Weakening) *If $\Gamma \leq \Delta$ then*

- $\Delta \vdash t : T$ *implies* $\Gamma \vdash t : T$
- $\Delta \vdash t = t' : T$ *implies* $\Gamma \vdash t = t' : T$

Anything more is out of reach. In particular, we cannot prove syntactic validity ($\Gamma \vdash t = t' : T$ implies $\Gamma \vdash t' : T$), or even that $\Gamma \vdash t : T$ implies $\Gamma \vdash T : s$, due to the type dependency $T[u/x]$ in the application rules. Proving substitution (that $\Gamma, x : U \vdash t : T$ and $\Gamma \vdash u : U$ implies $\Gamma \vdash t[u/x] : T[u/x]$) would itself require a substantial syntactic effort. We do not attempt to pave our way through the syntactic subtleties, as those properties will follow as results of the logical relation construction, following a technique of Goguen [Gog00].

2.2 Algorithmic type checking and equality

Algorithmic type checking We remark that the type checking rule ($\Gamma \vdash t : T$) presented above (1, page 2) are almost algorithmic rules. Indeed, all rules are syntax-directed, except for the conversion rule. To obtain a type checking algorithm, one can notice that equality checking is only useful on applications and function type checking: those are the only rules where we compare the type of two subterms, which may or may not be equal. In all other case, the return type is a composition of the subterm types, so no additional conversion rule may help typability. We therefore turn the definitional checking rule into algorithmic by inserting an equality check $\mathrel{\widehat{\Longleftrightarrow}}$, yet to be defined, in those two rules:

$$\frac{\Gamma \vdash U \Rightarrow s \quad \Gamma, x : U \vdash T \Rightarrow s' \quad \Gamma \vdash s \mathrel{\widehat{\Longleftrightarrow}} s'}{\Gamma \vdash (x : U) \to T \Rightarrow s}$$

$$\frac{\Gamma \vdash t \Rightarrow (x : U) \to T \quad \Gamma \vdash u \Rightarrow U' \quad \Gamma \vdash U \mathrel{\widehat{\Longleftrightarrow}} U'}{\Gamma \vdash t\, u \Rightarrow T[u/x]}$$

One important remark is that the algorithmic equality check is only performed on types that are returned by the algorithmic typing algorithm, supposedly well-formed. This means that we can assume that algorithmic equality will only be called on well-typed terms, assumption which will play an important role in the design of a practical equality checking algorithm.

Weak-head normalization We first define weak-head normal forms (whnf), in particular neutral whnf, which are terms applied to a head variable – the only possible whnf after enough η-expansion – weak-head reduction $t \searrow a$ and active application $t \mathbin{@} u \searrow a$. We will write $\downarrow t$ for the a such that $t \searrow a$, if it exists.

$$\begin{array}{llll} \mathsf{Whnf} \ni a, b, f, A, B, F & ::= & s \mid (x : U) \to T \mid \lambda x : U.\, t \mid n & \text{whnf} \\ \mathsf{Wne} \ni n, N & ::= & x \mid n\, u & \text{neutral whnf} \end{array}$$

$$\frac{t \searrow f \qquad f \mathbin{@} u \searrow a}{t\, u \searrow a} \qquad \frac{}{a \searrow a} \qquad \frac{t[u/x] \searrow a}{(\lambda x : U.\, t) \mathbin{@} u \searrow a} \qquad \frac{}{n \mathbin{@} u \searrow n\, u}$$

The algorithmic equality alternates weak-head normalizations (to account for β-reductions), type-directed equality (η-expansions), and structural equality on neutral terms. The core idea is that once we have reduced equality to two neutral terms[1] $\Gamma \vdash x\, u_1\, u_2 ... \longleftrightarrow x\, u'_1\, u'_2 ...$, we can look into the context for the type of x, which must be of the form $T_1 \to T_2 \to ...$, and therefore deduce the types of u_i, u'_i, on which we can run type-directed equality $\Gamma \vdash u_i \mathrel{\widehat{\Longleftrightarrow}} u'_i : T_i$.

[1] if the head variable on each side of the equality is different, equality checking fails

Type equality $\Delta \vdash A \Longleftrightarrow A'$, for weak head normal forms, and $\Delta \vdash T \mathrel{\widehat{\Longleftrightarrow}} T'$, for arbitrary well-formed types, checks that two given types are equal in their respective contexts.

$$\frac{}{\Delta \vdash s \Longleftrightarrow s} \qquad \frac{\Delta \vdash N \longleftrightarrow N' : s}{\Delta \vdash N \Longleftrightarrow N'} \qquad \frac{\Delta \vdash \downarrow T \Longleftrightarrow \downarrow T'}{\Delta \vdash T \mathrel{\widehat{\Longleftrightarrow}} T'}$$

$$\frac{\Delta \vdash U \mathrel{\widehat{\Longleftrightarrow}} U' \qquad \Delta.\, x{:}U \vdash T \mathrel{\widehat{\Longleftrightarrow}} T'}{\Delta \vdash (x{:}U) \to T \Longleftrightarrow (x{:}U') \to T'}$$

Structural equality $\Delta \vdash n \longleftrightarrow n' : A$ and $\Delta \vdash n \mathrel{\widehat{\longleftrightarrow}} n' : T$ checks the neutral expressions n and n' for equality and at the same time infers their type, which is returned as output, allowing neutral applications to deduce the type of their last argument and call typed-directed equality in turn.

$$\frac{\Delta \vdash n \mathrel{\widehat{\longleftrightarrow}} n' : T}{\Delta \vdash n \longleftrightarrow n' : \downarrow T} \qquad \frac{(x{:}T) \in \Delta}{\Delta \vdash x \mathrel{\widehat{\longleftrightarrow}} x : T} \qquad \frac{\Delta \vdash n \longleftrightarrow n' : (x{:}U) \to T \qquad \Delta \vdash u \mathrel{\widehat{\Longleftrightarrow}} u' : U}{\Delta \vdash n\, u \mathrel{\widehat{\longleftrightarrow}} n'\, u' : T[u/x]}$$

Type-directed equality $\Delta \vdash t \Longleftrightarrow t' : A$ and $\Delta \vdash t \mathrel{\widehat{\Longleftrightarrow}} t' : T$ proceeds by inspecting the structure of the supplied type, performing η-expansions until we get to a base type, where we switch to structural equality again.

$$\frac{\Delta.\, x{:}U \vdash t\, x \mathrel{\widehat{\Longleftrightarrow}} t'\, x : T}{\Delta \vdash t \Longleftrightarrow t' : (x{:}U) \to T} \qquad \frac{\Delta \vdash T \mathrel{\widehat{\Longleftrightarrow}} T'}{\Delta \vdash T \Longleftrightarrow T' : s} \qquad \frac{\Delta \vdash \downarrow t \mathrel{\widehat{\longleftrightarrow}} \downarrow t' : T}{\Delta \vdash t \Longleftrightarrow t' : N} \qquad \frac{\Delta \vdash t \Longleftrightarrow t' : \downarrow T}{\Delta \vdash t \mathrel{\widehat{\Longleftrightarrow}} t' : T}$$

Note that, in the but-last rule, we do not check that the inferred type T of $\downarrow t$ is convertible to the ascribed type N. Since algorithmic equality is only invoked for well-typed t, we know that this must always be the case. Skipping this test is our improvement over Harper and Pfenning's algorithm.

Transitivity requires soundness We can now expose the problem with transitivity in presence of type dependencies. We must show transitivity of the equality algorithms, otherwise it cannot possibly be complete with respect to definitional equality, which has a transitivity rule. We may want to show, for example, that $\Delta \vdash A_1 \Longleftrightarrow A_2$ and $\Delta \vdash A_2 \Longleftrightarrow A_3$ implies $\Delta \vdash A_1 \Longleftrightarrow A_3$. But consider two instances of the function type case:

$$\frac{\Delta \vdash U_1 \mathrel{\widehat{\Longleftrightarrow}} U_2 \qquad \Delta.\, x{:}U_1 \vdash T_1 \mathrel{\widehat{\Longleftrightarrow}} T_2}{\Delta \vdash (x{:}U_1) \to T_1 \Longleftrightarrow (x{:}U_2) \to T_2}$$

$$\frac{\Delta \vdash U_2 \mathrel{\widehat{\Longleftrightarrow}} U_3 \qquad \Delta.\, x{:}U_2 \vdash T_2 \mathrel{\widehat{\Longleftrightarrow}} T_3}{\Delta \vdash (x{:}U_2) \to T_2 \Longleftrightarrow (x{:}U_3) \to T_3}$$

To get transitivity in that case, we need to show that $\Delta \vdash U_1 \mathrel{\widehat{\Longleftrightarrow}} U_3$, which is a direct inductive hypothesis, and that $\Delta.\, x : U_1 \vdash T_1 \mathrel{\widehat{\Longleftrightarrow}} T_3$. That would be immediate if we had $\Delta.\, x{:}U_1$ instead of $\Delta.\, x{:}U_2$ as context of $\vdash T_2 \mathrel{\widehat{\Longleftrightarrow}} T_3$. Once we prove soundness, we can use $\Delta \vdash U_1 \mathrel{\widehat{\Longleftrightarrow}} U_2$ to deduce an equality between contexts $\Delta.\, x : U_1$ and $\Delta.\, x : U_2$, and obtain $\Delta.\, x : U_1 \vdash T_2 \mathrel{\widehat{\Longleftrightarrow}} T_3$ by conversion of equal contexts. So our proof of homogeneous algorithmic transitivity relies on algorithmic soundness, which itself needs significant metatheoretic results: subject reduction and function types injectivity.

2.3 Cumulativity

Definitional cumulativity The cumulativity relation is meant to model "inclusion between universes": $\Gamma \vdash \mathsf{Set}_i \preccurlyeq \mathsf{Set}_{i+1}$ meaning that all terms typed by Set_i are also accepted by Set_{i+1}. In the definitional system, it is an inductively defined relation building upon definitional equality:

Ordering rules

$$\frac{\Gamma \vdash T = T' : s}{\Gamma \vdash T \preccurlyeq T' : s} \qquad \frac{\Gamma \vdash T_1 \preccurlyeq T_2 : s \qquad \Gamma \vdash T_2 \preccurlyeq T_3 : s}{\Gamma \vdash T_1 \preccurlyeq T_3 : s}$$

Constructor rules

$$\frac{\vdash \Gamma}{\Gamma \vdash \mathsf{Set}_i \preccurlyeq \mathsf{Set}_{i+1} : \mathsf{Set}_{i+2}} \qquad \frac{\Gamma \vdash U \succcurlyeq U' : s \qquad \Gamma.\, x{:}U' \vdash T \preccurlyeq T' : s}{\Gamma \vdash (x{:}U) \to T \preccurlyeq (x{:}U') \to T' : s}$$

Conversion rule

$$\frac{\Gamma \vdash T \preccurlyeq T' : s \qquad \Gamma \vdash s \preccurlyeq s'}{\Gamma \vdash T \preccurlyeq T' : s'}$$

Notice that cumulativity is contravariant in its domain: $\Gamma \vdash U \succcurlyeq U' : s$ is defined to mean $\Gamma \vdash U' \preccurlyeq U : s$. Contrarily to the equality case, the T context is extended with U' instead of U; this is more general as $\Gamma \vdash U \succcurlyeq U' : s$ will allow the variable $x : U'$ to be considered of type U if necessary.

To use cumulativity in the type system, we add cumulativity conversion rules. As cumulativity includes equality thanks to the reflexivity rule, those rules subsume the corresponding conversion rules:

$$\frac{\Gamma \vdash t : T \quad \Gamma \vdash T \preccurlyeq T' : s}{\Gamma \vdash t : T'} \qquad \frac{\Gamma \vdash t = t' : T \quad \Gamma \vdash T \preccurlyeq T' : s}{\Gamma \vdash t = t' : T'}$$

Algorithmic cumulativity It helps to understand the structure of the definitional cumulativity proofs. A cumulativity proof first recursively explores function types and then, at the leaves (and according to the covariance or contravariance of the position), may change universe levels, or perform computations through equality; both an arbitrary number of times, thanks to the transitivity rule.

The algorithmic cumulativity check $\trianglelefteq$ exactly performs this traversal, relying on algorithmic equality to check computations at the leaves:

$$\frac{}{\Delta \vdash \mathsf{Set}_i \trianglelefteq \mathsf{Set}_j}\ i \leq j \qquad \frac{\Delta \vdash N \longleftrightarrow N' : s}{\Delta \vdash N \trianglelefteq N'}$$

$$\frac{\Delta \vdash U \trianglerighteq U' \qquad \Delta.\, x{:}U' \vdash T \trianglelefteq T'}{\Delta \vdash (x{:}U) \to T \trianglelefteq (x{:}U') \to T'} \qquad \frac{\Delta \vdash \downarrow T \trianglelefteq \downarrow T'}{\Delta \vdash T \mathrel{\widehat{\trianglelefteq}} T'}$$

It is interesting to compare algorithmic cumulativity with the algorithmic type equality: cumulativity is exactly a directed version of type equality. This helps to ensure that our design is natural rather than ad-hoc, and will provide a very direct antisymmetry proof once the equivalence of algorithmic and definitional equality is obtained.

3 Equivalence of definitional and algorithmic system: logical relations

We want to prove that definitional and algorithmic system are equivalent.

- *soundness*: algorithmic equality implies definitional equality. The algorithm is sound, it doesn't claim that two things are equal when they really aren't.
- *completeness*: definitional equality implies algorithmic equality. The algorithm is complete, it is able to conclude positively for all "really equal" terms; there are no complicated equalities that it may miss.

For the algorithmic system to be a practical replacement for explicit derivations, we also want termination:

- *decidability*: the checking algorithms terminate: they succeed or fail in finite time.

3.1 An informal introduction to Kripke logical relations

We will first informally present the rationale and requirements for logical relations, the main proof tool used to bring those results.

Due to space restrictions, this informal presentation has been relegated to appendix A, page 21. The reader who is not familiar with Kripke logical relations is strongly advised to read it first. The knowledgeable reader can go directly to the next section, defining the logical relation formally.

3.2 Formal presentation of a generic logical relation

Partial equivalence relations The relations discussed so far are generally transitive and symmetric, but not necessarily reflexive. For example the relation $\Gamma \vdash \_ = \_ : T$ is transitive and symmetric, but not reflexive as an ill-typed term t may not verify $\Gamma \vdash t = t : T$. We call them *partial* equivalence relations (P.E.R.), as they may be seen as equivalence relation defined only on a subset of the relation space — in this example the well-typed terms. We say that t is "defined" for the P.E.R. $\mathcal{R}$ if there exist t' such that $t \mathrel{\mathcal{R}} t'$. Then we have $t \mathrel{\mathcal{R}} t$ by transitivity from $t \mathrel{\mathcal{R}} t'$ and its symmetric $t' \mathrel{\mathcal{R}} t$: a P.E.R. is reflexive on its defined terms.

Semantic universe We first give the complete definition of our semantic universes: to account for the predicative hierarchy Set_i, we define instead of a single universe U a family of universes U_i recursively defined over $i \in \mathbb{N}$. This is an inductive-recursive definition. As informally justified in the appendix, the U_i are subsets of $\mathsf{Exp} \times \mathcal{P}(\mathsf{Exp})$, and they uniquely associate to a type code T its extension $\mathcal{A}$, a set of values. Here $|\mathsf{U}_i|$ is defined as $\{T \mid (T, \mathcal{A}) \in \mathsf{U}_i\}$.

$$\frac{}{(N, \mathsf{Wne}) \in \mathsf{U}_i} \qquad \frac{j < i}{(\mathsf{Set}_j, |\mathsf{U}_j|) \in \mathsf{U}_i} \qquad \frac{(U, \mathcal{A}) \in \widehat{\mathsf{U}_i} \quad \forall u \in \widehat{\mathcal{A}}.\, (T[u/x], \widehat{\mathcal{F}}(\downarrow u)) \in \widehat{\mathsf{U}_i}}{((x{:}U) \to T, \Pi\, \mathcal{A}\, \mathcal{F}) \in \mathsf{U}_i}$$

We have carefully defined the universe relation to approximate the cumulativity relation $\preccurlyeq$; in particular, neutrals live at all levels, and Set_j is present in all universes U_i for $i > j$.

Base relation We will build a generic logical relation $\mathcal{L}_{\Delta,T} \subseteq \mathsf{Exp} \times \mathsf{Exp}$ from any suitable base relation $\mathcal{B}_{\Delta,T}$. Both are parametrized by a context $\Delta \in \mathsf{Cxt}$ and a type $T \in \mathsf{Exp}$. For syntactic convenience, we will reuse the equality notation, by writing respectively $\Delta \vdash_{\mathcal{B}} t = t' : T$ and $\Delta \vdash_{\mathcal{L}} t = t' : T$.

We assume the following properties of the base relation $\mathcal{B}$.

Hypothesis 3 (P.E.R.) *$\mathcal{B}_{\Delta,T}$ is a partial equivalence relation.*

Hypothesis 4 (Context well-formedness) *$\mathcal{B}$ is only defined on well-formed contexts: $\Delta \vdash_{\mathcal{B}} t = t' : T$ implies $\Delta \vdash$.*

Hypothesis 5 (Variable definedness) *Well-typed variables are defined in $\mathcal{B}$: $\Delta.\, x : U \vdash_{\mathcal{B}} x = x : U$.*

Hypothesis 6 (Weakening) *If $\Delta \vdash_{\mathcal{B}} t = t' : T$ and $\Gamma \leq \Delta$, then $\Gamma \vdash_{\mathcal{B}} t = t' : T$.*

Hypothesis 7 (Type conversion) *If $\Delta \vdash_{\mathcal{B}} t = t' : T$ and $\Delta \vdash_{\mathcal{B}} T = T' : s$ then $\Delta \vdash_{\mathcal{B}} t = t' : T'$.*

The logical relation $\mathcal{L}$ We define the Kripke logical relation $\mathcal{L}_{\Delta,T}$ by induction over the membership of types to the semantic universe: for T and T' in $|\widehat{\mathsf{U}_i}|$ we define both $\Delta \vdash_{\mathcal{L}} T = T' : \mathsf{Set}_i$ and $\Delta \vdash_{\mathcal{L}} t = t' : T$.

We want to restrict our base relation $\vdash_{\mathcal{B}}$ to well-typed terms, to get syntactic validity later. We will write $\Delta \vdash_{\mathcal{B}} t :=: t' : T$ for the conjunction of $\Delta \vdash_{\mathcal{B}} t = t' : T$, $\Delta \vdash t : T$ and $\Delta \vdash t' : T$.

$$\frac{\Delta \vdash_{\mathcal{B}} N :=: N' : s}{\Delta \vdash_{\mathcal{L}} N = N' : s} \qquad \frac{\Delta \vdash_{\mathcal{B}} n :=: n' : N}{\Delta \vdash_{\mathcal{L}} n = n' : N} \qquad \frac{\vdash \Delta}{\Delta \vdash_{\mathcal{L}} s = s : s'}\ s' > s$$

$$\frac{\begin{array}{c}\Delta \vdash_{\mathcal{L}} U = U' : s \\ \forall \Gamma \leq \Delta,\ \Gamma \vdash_{\mathcal{L}} u = u' : U \implies \Gamma \vdash_{\mathcal{L}} T[u/x] = T'[u'/x] : s \\ \Delta \vdash_{\mathcal{B}} (x{:}U) \to T :=: (x{:}U') \to T' : s\end{array}}{\Delta \vdash_{\mathcal{L}} (x{:}U) \to T = (x{:}U') \to T' : s}$$

$$\frac{\begin{array}{c}\forall \Gamma \leq \Delta,\ \Gamma \vdash_{\mathcal{L}} u = u' : U \implies \Gamma \vdash_{\mathcal{L}} f \mathbin{@} u = f' \mathbin{@} u' : T[u/x] \\ \Delta \vdash_{\mathcal{B}} f :=: f' : (x{:}U) \to T\end{array}}{\Delta \vdash_{\mathcal{L}} f = f' : (x{:}U) \to T}$$

$$\frac{\begin{array}{ccccc} & & T \searrow A & \Delta \vdash_{\mathcal{B}} T = A : s & \\ t \searrow a & \Delta \vdash_{\mathcal{B}} t = a : A & \Delta \vdash_{\mathcal{B}} t' = a' : A & & t' \searrow a' \\ & & \Delta \vdash_{\mathcal{L}} a = a' : A & & \\ & & \Delta \vdash_{\mathcal{B}} t :=: t' : T & & \end{array}}{\Delta \vdash_{\mathcal{L}} t = t' : T}$$

Note that in the definition of $\Delta \vdash_{\mathcal{L}} t = t' : T$ for $(T, \mathcal{A}) \in \widehat{\mathsf{U}_i}$, we always have $t, t' \in \mathcal{A}$. This can be easily checked by induction, as the base case only relate neutrals.

Cumulativity The presence of cumulativity does not change much in the definition and property of the logical relation. We have several choices as to how to transpose cumulativity in a logical setting $\Delta \vdash_{\mathcal{L}} T \preccurlyeq T' : s$. We could parametrize over a "base cumulativity relation" with defined properties as we do for $\mathcal{B}$, we could choose to mirror the structure of the cumulativity check into a logical cumulativity relation, or we could use a very shallow definition, for which we would later prove that it indeed is related to cumulativity.

The first option is a bit heavy; we will detail the second and third option. In the second case, we would use a structured definition such as this one:

$$\frac{\vdash \Delta \quad i < j < k}{\Delta \vdash_{\mathcal{L}} \mathsf{Set}_i \preccurlyeq \mathsf{Set}_j : \mathsf{Set}_k} \qquad \frac{\begin{array}{c}\Delta \vdash_{\mathcal{L}} U \succcurlyeq U' : s \\ \forall \Gamma \leq \Delta,\ \Gamma \vdash_{\mathcal{L}} u = u' : U' \implies \Gamma \vdash_{\mathcal{L}} T[u/x] \preccurlyeq T'[u'/x] : s\end{array}}{\Delta \vdash_{\mathcal{L}} (x{:}U) \to T \preccurlyeq (x{:}U') \to T' : s}$$

In the third case, we would use a shallow definition such as this one:

$$\frac{\forall \Gamma \leq \Delta,\ \Gamma \vdash_{\mathcal{L}} t = t' : T \implies \Gamma \vdash_{\mathcal{L}} t = t' : T'}{\Delta \vdash_{\mathcal{L}} T \preccurlyeq T' : s}$$

The structured definition make it relatively easy to prove its correspondence with definitional and algorithmic cumulativity, as it follows the same structure. It asks for more work, however, to prove that the cumulativity conversion is still true at the logical level. The shallow relation makes cumulativity conversion trivial, but makes it harder to relate to definitional and algorithmic cumulativity, because *then* we would need to prove logical cumulativity conversion. We prove the same things in both case, but in different places.

The following properties of the logical relation will make reference to this logical cumulativity $\preccurlyeq_{\mathcal{L}}$; either definition is suitable in this case.

Properties of the logical relation By lack of space, we have not given the proofs of the following properties, but they are relatively simple, as the definition of the relation was precisely designed to provide those results. They are all proved by induction over the membership $T \in |\widehat{\mathsf{U}_i}|$ of the relating type in the semantic universe: this is the analoguous, for logical relations, of the syntactic induction performed on the more gentle inductive relations.

The first properties are mostly lifting of the base relation properties to the logical relation. As we have explained in our informal appendix, they are nonetheless not trivial. For example, weakening mandates the use of Kripke relations; type conversion also dictates the definition of $\mathcal{L}$ between function types.

Lemma 8 (Weakening) *If $\Gamma \leq \Delta$, then*

- $\Delta \vdash_{\mathcal{L}} T = T' : s$ *implies* $\Gamma \vdash_{\mathcal{L}} T = T' : s$
- $\Delta \vdash_{\mathcal{L}} t = t' : T$ *implies* $\Gamma \vdash_{\mathcal{L}} t = t' : T$
- $\Delta \vdash_{\mathcal{L}} T \preccurlyeq T' : s$ *implies* $\Gamma \vdash_{\mathcal{L}} T \preccurlyeq T' : s$

Lemma 9 (Type conversion)

- *If* $\Delta \vdash_{\mathcal{L}} t = t' : T$ *and* $\Delta \vdash_{\mathcal{L}} T = T' : s$ *then* $\Delta \vdash_{\mathcal{L}} t = t' : T'$.
- *If* $\Delta \vdash_{\mathcal{L}} t = t' : T$ *and* $\Delta \vdash_{\mathcal{L}} T \preccurlyeq T' : s$ *then* $\Delta \vdash_{\mathcal{L}} t = t' : T'$.

Lemma 10 (P.E.R.) *$\mathcal{L}$ is a partial equivalence relation. $\preccurlyeq_{\mathcal{L}}$ is transitive.*

While symmetry is very simple to prove, transitivity is made more delicate by the negative occurence of $\mathcal{L}$ at function types. We will detail the crucial case of transitivity at function types.

Proof We have

$$\frac{\begin{array}{c}\Delta \vdash_{\mathcal{L}} U_1 = U_2 : s \\ \forall \Gamma \leq \Delta,\ \Gamma \vdash_{\mathcal{L}} u = u' : U_1 \implies \Gamma \vdash_{\mathcal{L}} T_1[u/x] = T_2[u'/x] : s \\ \Delta \vdash_{\mathcal{B}} (x{:}U_1) \to T_1 = (x{:}U_2) \to T_2 : s\end{array}}{\Delta \vdash_{\mathcal{L}} (x{:}U_1) \to T_1 = (x{:}U_2) \to T_2 : s}$$

$$\frac{\begin{array}{c}\Delta \vdash_{\mathcal{L}} U_2 = U_3 : s \\ \forall \Gamma \leq \Delta,\ \Gamma \vdash_{\mathcal{L}} u = u' : U_2 \implies \Gamma \vdash_{\mathcal{L}} T_2[u/x] = T_3[u'/x] : s \\ \Delta \vdash_{\mathcal{B}} (x{:}U_2) \to T_2 = (x{:}U_3) \to T_3 : s\end{array}}{\Delta \vdash_{\mathcal{L}} (x{:}U_2) \to T_2 = (x{:}U_3) \to T_3 : s}$$

And wish to prove $\Delta \vdash_{\mathcal{L}} (x : U_1) \to T_1 = (x : U_3) \to T_3 : s$. We have $\Delta \vdash_{\mathcal{L}} U_1 = U_3 : s$ by inductive hypothesis, and $\Delta \vdash (x : U_1) \to T_1 \ \mathcal{B}\ (x : U_3) \to T_3 : s$ by transitivity of $\mathcal{B}$. Given $\Gamma \leq \Delta$ and $\Gamma \vdash_{\mathcal{L}} u = u' : U_1$, we have to prove that $\Gamma \vdash_{\mathcal{L}} T_1[u/x] = T_3[u'/x] : s$.

By induction hypothesis, $\mathcal{L}$ is a P.E.R. at type U_1, so in particlar from $\Gamma \vdash_{\mathcal{L}} u = u' : U_1$ we have $\Gamma \vdash_{\mathcal{L}} u = u : U_1$, from which we deduce $\Gamma \vdash_{\mathcal{L}} T_1[u/x] = T_2[u/x]$. We also have $\Gamma \vdash_{\mathcal{L}} U_1 = U_2$ by weakening hypothesis $\Delta \vdash_{\mathcal{L}} U_1 = U_2$, which allows to use type conversion to get $\Gamma \vdash_{\mathcal{L}} u = u' : U_2$. From this we deduce $\Gamma \vdash_{\mathcal{L}} T_2[u/x] = T_3[u'/x]$, which allows us to conclude by transitivity — as an inductive hypothesis.

□

Lemma 11 (Into the logical relation) *If $T \in \mathsf{U}_i$ and $\Delta \vdash_{\mathcal{B}} n = n' : T$ then $\Delta \vdash_{\mathcal{L}} n = n' : T$.*

This is true by definition of $\mathcal{L}$ when T is a neutral type. Otherwise, we proceed by induction on $T \in \mathsf{U}_i$. Note that we know that n, n' are approximate elements of T in U_i, as all types of the universe accept all neutrals — this is easily checked by induction.

3.3 Validity and substitutions

An important property of type systems is to behave well with respect to substitution; this is the property of *substitution* we have defined earlier. A similar property would be expected of the logical relation, in particular if we wish to prove that the β-reduction rule is admissible: $\Delta \vdash_{\mathcal{L}} (\lambda x{:}U.\, t)\ u = t[u/x] : T$.

Unfortunately, it is not at all easy to manipulate substitutions with the current definition of the logical relation. For example, it treats neutrals specifically, but the property of being neutral is not stable by substitution of the head variable.

The solution is to introduce a new relation, the *validity relation* $\Vdash_{\mathcal{L}}$, that is in some sense "closed over substitutions", as the logical relation was closed over applications.

Logically related substitutions To a relation $\Delta \vdash_{\mathcal{L}} t = t' : T$, we wish to apply *related* substitutions σ and σ', to get $\Delta \vdash_{\mathcal{L}} t\sigma = t'\sigma' : T\sigma$. We now define what it means for two substitutions to be related: they substitute related terms for a given variable. Additionally, as a substitution may remove some free variables and introduce new ones, the result context Δ is not necessarily identical to the origin context Γ; we write $\Delta \vdash_{\mathcal{L}} \sigma = \sigma' : \Gamma$ when σ, σ' are $\mathcal{L}$-related and translate terms from context Γ to Δ. This is reminiscent of the categorical view of substitutions as context morphisms. This relation between substitutions is inductively defined:

$$\frac{}{\Delta \vdash_{\mathcal{L}} \sigma = \sigma' : \diamond} \qquad \frac{\Delta \vdash_{\mathcal{L}} \sigma = \sigma' : \Gamma \quad \Delta \vdash_{\mathcal{L}} \sigma(x) = \sigma'(x) : U\sigma}{\Delta \vdash_{\mathcal{L}} \sigma = \sigma' : \Gamma.\, x{:}U}$$

The validity relation $\Vdash_{\mathcal{L}}$ We can now define the validity relation $\Gamma \Vdash_{\mathcal{L}} t = t' : T$. We also use a "context validity" relation $\Gamma \Vdash_{\mathcal{L}}$, and define $\Gamma \Vdash_{\mathcal{L}} T$ and $\Gamma \Vdash_{\mathcal{L}} t : T$ as syntactic sugar for $\exists s, \Gamma \Vdash_{\mathcal{L}} T = T : s$ and $\Gamma \Vdash_{\mathcal{L}} t = t : T$ respectively.

$$\frac{}{\diamond \Vdash_{\mathcal{L}}} \qquad \frac{\Gamma \Vdash_{\mathcal{L}} \qquad \Gamma \Vdash_{\mathcal{L}} U}{\Gamma.\, x{:}U \Vdash_{\mathcal{L}}} \qquad \frac{\Gamma \Vdash_{\mathcal{L}} \qquad \Gamma \Vdash_{\mathcal{L}} T : s \text{ (unless } T \text{ is a sort)} \quad \forall \Delta, \sigma, \sigma',\ \Delta \vdash_{\mathcal{L}} \sigma = \sigma' : \Gamma \implies \Delta \vdash_{\mathcal{L}} t\sigma = t'\sigma' : T\sigma}{\Gamma \Vdash_{\mathcal{L}} t = t' : T}$$

$$\frac{\Gamma \Vdash_{\mathcal{L}} \qquad \forall \Delta, \sigma, \sigma',\ \Delta \vdash_{\mathcal{L}} \sigma = \sigma' : \Gamma \implies \Delta \vdash_{\mathcal{L}} T\sigma \preccurlyeq T'\sigma' : s}{\Gamma \Vdash_{\mathcal{L}} T \preccurlyeq T' : s}$$

Lemma 12 (Substitution relation is a P.E.R.) *If* $\Gamma \Vdash_{\mathcal{L}}$ *then* $\Delta \vdash_{\mathcal{L}} \_ = \_ : \Gamma$ *is symmetric and transitive.* $\Delta \Vdash_{\mathcal{L}} \_ \preccurlyeq \_ : s$ *is transitive.*

Lemma 13 (Validity is a P.E.R.) $\Gamma \Vdash_{\mathcal{L}} \_ = \_ : T$ *is symmetric and transitive.*

Lemma 14 (Function types are injective (in the validity relation)) *If* $\Gamma \Vdash_{\mathcal{L}} (x{:}U) \to T = (x{:}U') \to T' : s$ *then* $\Gamma \Vdash_{\mathcal{L}} U = U' : s$ *and* $\Gamma.\, x{:}U \Vdash_{\mathcal{L}} T = T' : s$.

This is our first example of a meta-theoretic result that we easily obtain in the derived relation $\mathcal{L}$ and $\Vdash_{\mathcal{L}}$ – because it's embedded in $\mathcal{L}$'s definition – while we cannot prove it directly for the definitional equality. Once we particularize the generic $\mathcal{B}$ to definitional equality, and prove that $\mathcal{L}$ is not more restrictive than $\mathcal{B}$, we will have proved function type injectivity for definitional equality.

Lemma 15 (Context satisfiable) *The identity substitution is related to itself: if* $\Gamma \Vdash_{\mathcal{L}}$ *then* $\Gamma \vdash$ *and* $\Gamma \Vdash_{\mathcal{L}} \mathsf{id} = \mathsf{id} : \Gamma$.

The identity relation allows to close the loop between $\Vdash_{\mathcal{L}}$ and $\mathcal{B}$.

Lemma 16 (Completeness of the validity relation) *If* $\Gamma \Vdash_{\mathcal{L}} t = t' : T$ *then* $\Gamma \vdash_{\mathcal{B}} t = t' : T$ *and* $\Gamma \vdash_{\mathcal{B}} T = T : s$.

The fundamental theorem of the logical relation Now that we have built $\mathcal{L}$ and $\Vdash_{\mathcal{L}}$, we can show that they indeed have the desirable compositional properties of the definitional equality. We have already shown that the symmetry and transitivity rules are admissible, and we prove that $\Vdash_{\mathcal{L}}$, despite being built only from $\mathcal{B}$, admits the other equality rules.

Lemma 17 (Validity of β) *The following rule is admissible*

$$\frac{\Gamma.\, x{:}U \Vdash_{\mathcal{L}} t : T \qquad \Gamma \Vdash_{\mathcal{L}} u : U}{\Gamma \Vdash_{\mathcal{L}} (\lambda x{:}U.\, t)\, u = t[u/x] : T[u/x]}$$

The crux of the β-reduction proof is that if $\Delta \vdash_{\mathcal{L}} \sigma = \sigma' : \Gamma$ are related substitutions, then $\Gamma \Vdash_{\mathcal{L}} u : U$ implies that $\Delta \Vdash_{\mathcal{L}} (\sigma, u\sigma/x) = (\sigma', u\sigma'/x) : \Gamma.\, x{:}U$.

Lemma 18 (Validity of function equality and η)

$$\frac{\Gamma \Vdash_{\mathcal{L}} U = U' : s \qquad \Gamma.\, x{:}U \Vdash_{\mathcal{L}} T : s \qquad \Gamma.\, x{:}U \Vdash_{\mathcal{L}} t = t' : T}{\Gamma \Vdash_{\mathcal{L}} \lambda x{:}U.\, t = \lambda x{:}U'.\, t' : (x{:}U) \to T}$$

$$\frac{\Gamma \Vdash_{\mathcal{L}} t : (x{:}U) \to T}{\Gamma \Vdash_{\mathcal{L}} t = \lambda x{:}U.\, t\, x : (x{:}U) \to T}$$

The link between function equality and η is a *β-expansion*, which we have shown admissible.

Lemma 19 (Validity of function application)

$$\frac{\Gamma \Vdash_{\mathcal{L}} t = t' : (x{:}U) \to T \qquad \Gamma \Vdash_{\mathcal{L}} u = u' : U}{\Gamma \Vdash_{\mathcal{L}} t\, u = t'\, u' : T[u/x]}$$

The proof relies entirely, of course, on the definition of $\mathcal{L}$ at function types.

Lemma 20 (Validity of conversion)

$$\frac{\Gamma \Vdash_{\mathcal{L}} t = t' : T \qquad \Gamma \Vdash_{\mathcal{L}} T \preccurlyeq T' : s}{\Gamma \Vdash_{\mathcal{L}} t = t' : T'}$$

Theorem 21 (Fundamental Theorem of the Logical Relation) *Let $\mathcal{B}$ be a base relation satisfying the required properties (3.2, page 12), $\mathcal{L}$ and $\Vdash_{\mathcal{L}}$ defined as explained (3.2, page 12 and 3.3, page 14). Then the following completeness properties hold:*

- *If $\Gamma \vdash$ then $\Gamma \Vdash_{\mathcal{L}}$.*
- *If $\Gamma \vdash t : T$ then $\Gamma \Vdash_{\mathcal{L}} t : T$.*
- *If $\Gamma \vdash t = t' : T$ then $\Gamma \Vdash_{\mathcal{L}} t = t' : T$.*
- *If $\Gamma \vdash T \preccurlyeq T' : s$ then $\Gamma \Vdash_{\mathcal{L}} T \preccurlyeq T' : s$.*

This is proved using the previous validity lemmas: we do an syntactic induction on the definitional judgement, using the admissibility of each rule in the validity relation.

It may be surprising that we obtain definitional equality and cumulativity from any base relation $\mathcal{B}$, but the hypotheses on $\mathcal{B}$ are in fact quite constraining; it can only be used for a relation that is "similar enough" to definitional equality. Fortunately, algorithmic equality can be persuaded to fit these restrictions; the Fundamental Theorem will then provides a completeness proof.

Finally, we may also extend the fundamental theorem to well-typed substitutions. Let $\Delta \vdash_{\mathcal{B}} \sigma = \sigma' : \Gamma$ and $\Gamma \Vdash_{\mathcal{L}} \sigma = \sigma' : \Gamma'$ be defined as:

$$\frac{\vdash \Delta}{\Delta \vdash_{\mathcal{B}} \sigma = \sigma' : \diamond} \qquad \frac{\Delta \vdash_{\mathcal{B}} \sigma = \sigma' : \Gamma \quad \Gamma \vdash U \quad \Delta \vdash_{\mathcal{B}} \sigma(x) = \sigma'(x) : U\sigma}{\Delta \vdash_{\mathcal{B}} \sigma = \sigma' : \Gamma.\, x{:}U}$$

$$\frac{\begin{array}{c}\Gamma \Vdash_{\mathcal{L}} \quad \Gamma' \Vdash_{\mathcal{L}} \\ \forall \Delta \vdash_{\mathcal{L}} \rho = \rho' : \Gamma,\ \Delta \vdash_{\mathcal{L}} \sigma\rho = \sigma'\rho' : \Gamma'\end{array}}{\Gamma \Vdash_{\mathcal{L}} \sigma = \sigma' : \Gamma'}$$

Corollary 22 (Fundamental lemma for substitutions) *If $\Gamma \vdash_{\mathcal{B}} \sigma = \sigma' : \Gamma'$ then $\Gamma \Vdash_{\mathcal{L}} \sigma = \sigma' : \Gamma'$.*

4 The eagerly awaited metatheoretic results

The Fundamental Lemma of the previous section relates a base relation $\mathcal{B}$, the validity relation $\Vdash_{\mathcal{L}}$ and the definitional typing and equality rules of our system. We will now exploit it to get the desired meta-theoretic results.

4.1 Metatheory of the definitional rules

We first establish results that are only concerned with the definitional system, and not related to algorithmic equality. For example, we have shown that $\Vdash_{\mathcal{L}}$ admits function type injectivity, and transferring this to definitional equality provide an important meta-theoretic result.

Definition 23 *Let $\mathcal{B}_{\Delta,T}$ be the following relation: $\Delta \vdash_{\mathcal{B}} t = t' : T$ if and only iff $\Delta \vdash t : T$ and $\Delta \vdash t' : T$ and $\Delta \vdash t = t' : T$.*

$\mathcal{B}_{\Delta,T}$ satisfies the required properties of the base relation described in (3.2, page 12), and $\Delta \vdash T \preccurlyeq T' : s$ implies $\Delta \vdash_{\mathcal{L}} T \preccurlyeq T' : s$.

We will note $\vdash_{\mathcal{S}}$ the corresponding logical relation (S for "soundness"), and $\Vdash_{\mathcal{S}}$ the corresponding validity relation.

The proof of the statements of this section are not given. The reader should not be confused by the apparent similarity with the previous section, where the proof where non-trivial but skipped for reasons of space constraints. Here the proofs are *short and easy*; it's generally only a matter of using the fundamental lemma to go from the definitional system to the validity relation, applying the corresponding and already-proved result there, and then using the identity substitution to return to the definitional world.

Syntactic validity is a direct consequence of the fundamental lemma, as we have already shown it to be true for the validity relation.

Corollary 24 (Syntactic validity)

- *If $\Gamma \vdash t : T$ then $\Gamma \vdash T$.*
- *If $\Gamma \vdash t = t' : T$ then $\Gamma \vdash t, t' : T$.*
- *If $\Gamma \vdash T \preccurlyeq T' : s$ then $\Gamma \vdash T, T' : s$.*

Function type injectivity This is a direct corollary of validity of function type injectivity (14, page 15).

Corollary 25 *If $\Gamma \vdash (x : U) \to T = (x : U') \to T' : s$ then $\Gamma \vdash U = U' : s$ and $\Gamma . x : U \vdash T = T' : s$.*

Substitution The second metatheoretic result is substitution. Let us write $\Delta \vdash \sigma = \sigma' : T$ for the relation on substitution determined by the $\mathcal{B}$ defined above. Substitution could have been proved syntactically, but this require to prove tricky interdependent syntactic lemmas; following the technique of Goguen [Gog00], we get it from the logical relation – more precisely, the fundamental lemma for substitutions – to avoid those subtleties.

Theorem 26 (substitution)

- *If $\Gamma \vdash \sigma = \sigma' : \Gamma'$ and $\Gamma' \vdash t : T$ then $\Gamma \vdash t\sigma = t\sigma' : T\sigma$.*
- *If $\Gamma \vdash \sigma = \sigma' : \Gamma'$ and $\Gamma' \vdash t = t' : T$ then $\Gamma \vdash t\sigma = t'\sigma' : T\sigma$.*
- *If $\Gamma \vdash \sigma = \sigma' : \Gamma'$ and $\Gamma' \vdash T \preccurlyeq T' : s$ then $\Gamma \vdash T\sigma \preccurlyeq T'\sigma' : s$.*

Inversion A condition for the decidability of type checking is the ability to invert typing derivations.

Lemma 27 (Inversion)

- *If $\Gamma \vdash x : T$ then $(x : U) \in \Gamma$ for some U with $\Gamma \vdash U \preccurlyeq T$.*
- *If $\Gamma \vdash \lambda x : U . t : T$ then $\Gamma . x : U \vdash t : T'$ for some T' with $\Gamma \vdash (x : U) \to T' \preccurlyeq T$.*
- *If $\Gamma \vdash t\, u : T$ then $\Gamma \vdash t : (x : U) \to T'$ and $\Gamma \vdash u : U$ for some U, T' with $\Gamma \vdash T'[u/x] \preccurlyeq T$.*
- *If $\Gamma \vdash \mathsf{Set}_i : T$ then $\Gamma \vdash \mathsf{Set}_{i+1} \preccurlyeq T$*
- *If $\Gamma \vdash (x : U) \to T' : T$ then, for some s, $\Gamma \vdash U : s$, $\Gamma . x : U \vdash T' : s$, and $\Gamma \vdash s \preccurlyeq T$.*

This proof is not a direct consequence of the logical relation, but depends on substitution. It is proved by induction on the derivations.

Context cumulativity We write $\Gamma \preccurlyeq \Gamma' \vdash$ if Γ, Γ' have the same domain and each types of Γ is more general than the corresponding type in Γ':

$$\frac{}{\diamond \preccurlyeq \diamond \vdash} \qquad \frac{\Gamma \preccurlyeq \Gamma' \vdash \qquad \Gamma \vdash U \preccurlyeq U'}{\Gamma.\, x{:}U \preccurlyeq \Gamma.\, '{:}xU'}$$

Context cumulativity conversion shows that going to a more general context preserves typability.

Theorem 28 (Context cumulativity conversion) *Let* $\Gamma \preccurlyeq \Gamma' \vdash$

- *If* $\Gamma' \vdash t : T$ *then* $\Gamma \vdash t : T$
- *If* $\Gamma' \vdash t = t' : T$ *then* $\Gamma \vdash t = t' : T$
- *If* $\Gamma' \vdash T \preccurlyeq T' : s$ *then* $\Gamma \vdash T \preccurlyeq T' : s$

It is proved using the substitution theorem (26, page 17), by relying on the fact that $\Gamma \preccurlyeq \Gamma' \vdash$ implies $\Gamma \vdash \mathsf{id} = \mathsf{id} : \Gamma'$.

Normalization and subject reduction The logical relation $\mathcal{L}$ was defined on weak-head normal forms of the universe U_i, and extended by $\mathcal{L}$ to the terms that reduce to those normal forms. By proving the Fundamental Lemma, we also proved, without noticing, that all the terms typed by the definitional system are in this situation. The normalization and subject reduction result falls out naturally.

Theorem 29 (Normalization and subject reduction) *If* $\Gamma \vdash t : T$ *then* $t \searrow a$ *and* $\Gamma \vdash t = a : T$.

Proof By the fundamental theorem we get $\Gamma \Vdash_{\mathcal{L}} t = t : T$ so $\Gamma \vdash_{\mathcal{L}} t = t : T$ using the identity substitution. The result directly follows from the definition of $\mathcal{L}$ on non-whnf:

$$\frac{\begin{array}{ccccc} & & T \searrow A & \Delta \vdash T = A & \\ t \searrow a & \Delta \vdash_{\mathcal{B}} t = a : A & & \Delta \vdash_{\mathcal{B}} t' = a' : A & t' \searrow a' \\ & & \Delta \vdash_{\mathcal{L}} a = a' : A & & \\ & & \Delta \vdash_{\mathcal{B}} t = t' : T & & \end{array}}{\Delta \vdash_{\mathcal{L}} t = t' : T}$$

□

Consistency Importantly, not every type is inhabited, thus, our system can be used as a logic. A prerequisite is that types can be distinguished, which follows immediately from the syntax-directed definition of the logical relation.

Lemma 30 (Type constructor discrimination) *Neutral types, sorts and function types are mutually unequal.*

- $\Gamma \vdash N \neq s$.
- $\Gamma \vdash N \neq (x \star U) \to T$.
- $\Gamma \vdash s = s'$ *implies* $s \equiv s'$.
- $\Gamma \vdash s \neq (x \star U) \to T$.

Lemma 31 (Distinct type constructors are incomparable) *Extending the previous result, we show that neutral types, sort and function types are mutually non-cumulative. We will write* $\Gamma \vdash T \not\preccurlyeq\!\not\succcurlyeq T'$ *to say that both* $\Gamma \vdash T \preccurlyeq T' : s$ *and* $\Gamma \vdash T \succcurlyeq T' : s$ *are false.*

- $\Gamma \vdash N \not\preccurlyeq\!\not\succcurlyeq s$.

- $\Gamma \vdash N \not\simeq (x \star U) \to T$.
- $\Gamma \vdash s \not\simeq (x \star U) \to T$.

From normalization and those type constructor discrimination results we can show that not every type is inhabited.

Theorem 32 (Consistency) $X : \mathsf{Set}_0 \not\vdash t : X$.

Proof Let $\Gamma = (X : \mathsf{Set}_0)$. Assuming $\Gamma \vdash t : X$, we have $\Gamma \vdash a : X$ for the whnf a of t. We invert on the typing of a. By Lemma 31, X cannot be equal – even modulo cumulativity – to a function type or sort, thus, a can neither be a λ nor a function type nor a sort, it can only be neutral. The only variable X must be in the head of a, but since X is not of function type, it cannot be applied. Thus, $a \equiv X$ and $\Gamma \vdash X : X$, implying $\Gamma \vdash \mathsf{Set}_0 \preccurlyeq X$ by inversion (27). This is in contradiction to Lemma 31, which states $\Gamma \vdash \mathsf{Set}_0 \not\simeq X$. □

4.2 Metatheory of the algorithmic equality

Using the result just proved for the definitional system, we will establish metatheoretic properties of the algorithmic equality, culminating in its soundness. From there, we will be able to use a second logical relation to get completeness.

Termination We have shown that well-typed terms are in the model. In particular, they are weak-head reducible so the algorithmic check terminates — on well-typed terms.

Theorem 33 (Termination of algorithmic equality) *If $\Delta \vdash t, t' : T$ then the query $\Delta \vdash t \mathrel{\widehat{\Longleftrightarrow}} t' : T$ terminates. If $\Delta \vdash T, T' : s$ then $\Delta \vdash T \mathrel{\widehat{\Longleftrightarrow}} T$ and $\Delta \vdash T \mathrel{\widehat{\trianglelefteq}} T'$ terminate.*

Soundness Soundness of the equality algorithm is a consequence of subject reduction. As we want to express soundness for typed terms, we define the relations $(:\Longleftrightarrow:)$, $(:\longleftrightarrow:)$, $(:\trianglelefteq:)$, etc. as the restriction of the relation on well-typed term: $\Gamma \vdash A :\trianglelefteq: A'$ if both $\Gamma \vdash A \trianglelefteq A'$ and $\Gamma \vdash A, A' : s$, etc.

Theorem 34 (Soundness of algorithmic equality and cumulativity)

- *If $\Delta \vdash t :\mathrel{\widehat{\Longleftrightarrow}}: t' : T$ then $\Delta \vdash t = t' : T$.*
- *If $\Delta \vdash n, n' : T$ and $\Delta \vdash n \mathrel{\widehat{\longleftrightarrow}} n' : U$ then $\Delta \vdash n = n' : U$ and $\Delta \vdash U \preccurlyeq T$.*
- *If $\Delta \vdash T :\mathrel{\widehat{\trianglelefteq}}: T'$ then $\Delta \vdash T \preccurlyeq T' : s$.*

Symmetry and transitivity

Lemma 35 (Type and context subsumption in algorithmic judgements) *Let $\vdash \Delta \succcurlyeq \Delta'$.*

- *If $\Delta \vdash A :\Longleftrightarrow: A' : s$ then $\Delta' \vdash A \Longleftrightarrow A' : s$.*
- *If $\Delta \vdash n :\longleftrightarrow: n' : A$ then $\Delta' \vdash n \longleftrightarrow n' : A'$ for some A' with $\Delta \vdash A \succcurlyeq A'$.*
- *If $\Delta \vdash t :\Longleftrightarrow: t' : A$ and $\Delta \vdash A \preccurlyeq A'$ then $\Delta' \vdash t \Longleftrightarrow t' : A'$.*
- *If $\Delta \vdash A :\trianglelefteq: A'$ then $\Delta' \vdash A \trianglelefteq A'$*

Lemma 36 (Algorithmic equality is transitive) *Let $\vdash \Delta = \Delta'$. In the following, let the terms submitted to algorithmic equality be well-typed.*

- *If $\Delta \vdash n_1 \mathrel{\widehat{\longleftrightarrow}} n_2 : T$ and $\Delta' \vdash n_2 \mathrel{\widehat{\longleftrightarrow}} n_3 : T'$ then $\Delta \vdash n_1 \mathrel{\widehat{\longleftrightarrow}} n_3 : T$ and $\Delta \vdash T = T'$.*

- *If* $\Delta \vdash t_1 \mathrel{\widehat{\Longleftrightarrow}} t_2 : T$ *and* $\Delta' \vdash t_2 \mathrel{\widehat{\Longleftrightarrow}} t_3 : T'$ *and* $\Delta \vdash T = T'$ *then* $\Delta \vdash t_1 \mathrel{\widehat{\Longleftrightarrow}} t_3 : T$.
- *If* $\Delta \vdash T_1 \mathrel{\widehat{\Longleftrightarrow}} T_2 : s$ *and* $\Delta' \vdash T_2 \mathrel{\widehat{\Longleftrightarrow}} T_3 : s$ *then* $\Delta \vdash T_1 \mathrel{\widehat{\Longleftrightarrow}} T_3 : s$
- *If* $\vdash \Delta \succcurlyeq \Delta'$ *and* $\Delta \vdash T_1 \mathrel{\widehat{\trianglelefteq}} T_2$ *and* $\Delta' \vdash T_2 \mathrel{\widehat{\trianglelefteq}} T_3$, *then* $\Delta' \vdash T_1 \mathrel{\widehat{\trianglelefteq}} T_3$.

Theorem 37 (Algorithmic equality is a P.E.R.) *The type, structural and type-directed equality judgements are symmetric and transitive on well-typed expressions.*

4.3 Completeness of algorithmic equality

Now that we have soundness of the algorithmic equality, we build a second logical relation to get completeness. We need to define a new base relation $\mathcal{B}$; to avoid confusion with the previous sections, we will note © the logical relation, and $\Delta \Vdash^{\mathsf{c}} t = t' : T$ the validity relation.

Due to the presence of three interleaved judgements, we cannot just "use algorithmic equality as base relation": which one? We will first use a base relation using a very restricted form of type-directed equality, build the corresponding logical relation, and in a second moment determine how the other algorithmic judgements are related to the logical relation; this will be our *escape lemma*, going from the logical relation to the algorithmic equality.

Definition 38 *Let* $\Delta \vdash_{\mathcal{B}} \_ = \_ : \_$ *be defined as:* $\Delta \vdash N :\Longleftrightarrow: N' : s$ *on neutral types,* $\Delta \vdash n :\Longleftrightarrow: n' : N$ *on neutral terms at a neutral type, and* $\Delta \vdash t :=: t' : T$ *otherwise.*

$\mathcal{B}$ *satisfies all the required properties of a base relation, and* $\Delta \vdash T \mathrel{\widehat{\trianglelefteq}} T' : s$ *implies* $\Delta \vdash_{\mathcal{L}} T \preccurlyeq T' : s$.

Let $\Delta \vdash_{\mathcal{C}} t = t' : T$ *be the corresponding logical relation (C for completeness), and* $\Delta \Vdash_{\mathcal{C}} t = t' : T$ *the validity relation.*

Remark that $\mathcal{B}$ is stronger than :=: by soundness of algorithmic equality, as :$\Longleftrightarrow$: implies definitional equality.

We now explain the relation between $\mathcal{C}$ and the three algorithmic judgements.

Lemma 39 (Escape from the logical relation) *Let* $\Delta \vdash_{\mathcal{C}} T = T' : s$

- $\Delta \vdash T \mathrel{\widehat{\Longleftrightarrow}} T'$.
- *If* $\Delta \vdash_{\mathcal{C}} t = t' : T$ *then* $\Delta \vdash t \mathrel{\widehat{\Longleftrightarrow}} t' : T$.
- *If* $\Delta \vdash n \mathrel{\widehat{\longleftrightarrow}} n' : T$ *and* $\Delta \vdash n = n' : T$ *then* $\Delta \vdash_{\mathcal{C}} n = n' : T$.

This is the non-trivial proof that allows completeness. We want to show that despite the very restricted use of algorithmic equality in the definition of the logical relation, that is only at base types, this relation coincides with algorithmic equality at all types. This works because the structure of the logical relation mirrors the algorithmic equality judgements.

Theorem 40 (Completeness of algorithmic equality and cumulativity) *If* $\Gamma \vdash t = t' : T$ *then* $\Gamma \vdash t \mathrel{\widehat{\Longleftrightarrow}} t' : T$. *If* $\Gamma \vdash T \preccurlyeq T' : s$ *then* $\Gamma \vdash T \mathrel{\widehat{\trianglelefteq}} T'$.

Proof for $\Gamma \vdash t = t' : T$ Since $\Gamma \vdash_{\mathcal{C}} \mathsf{id} = \mathsf{id} : \Gamma$, we have $\Gamma \vdash_{\mathcal{C}} t = t' : T$ by the Fundamental Theorem, and conclude with the escape lemma. □

We have shown that algorithmic equality is sound, complete and terminating. This means that our definitional type system is decidable.

Theorem 41 (Decidability) $\Gamma \vdash t : T$, $\Gamma \vdash t = t' : T$ *and* $\Gamma \vdash T \preccurlyeq T' : s$ *are decidable.*

The following results are not central to our development, but they strengthen our confindence in the cumulativity relation:

Theorem 42 (Antisymmetry of cumulativity) *If* $\Gamma \vdash T \preccurlyeq T' : s$ *and* $\Gamma \vdash T \succcurlyeq T' : s$*, then* $\Gamma \vdash T = T' : s$*.*

Theorem 43 (Principal types) *If* $\Gamma \vdash t : T$*, then there exists a* T' *such that* $\Gamma \vdash \downarrow t \overset{\frown}{\longleftrightarrow} \downarrow t : T'$ *and* $\Gamma \vdash T' \preccurlyeq T : s$*.*

5 Universe polymorphism, level irrelevance and heterogeneous equality

For more informal discussion of the more advanced feature considered, the reader is invited to read the appendix B, page 24.

Conclusion

In this report we have demonstrated proof techniques using two logicals relations to get the metatheory of a definitional system for dependent types with typed, extensional equality. We also proved its decidability by equivalence with a practical algorithmic type system, in a way that we believe would scale to large eliminations. Finally, we have incorporated a cumulativity relation inspired by Luo's ECC in our typed equality, and explored the addition of universe polymorphism and level irrelevance. We have also discussed the choice of homogeneous or heterogeneous presentations of equality, trying to expose the strength and weaknesses of both.

Before the end of the internship, we hope to be able to gain more assurance into the universe polymorphism mechanisms, and find a way to expose shape irrelevance that is not overly complex; while the core idea is rather simple, the technical presentation is currently very sophisticated. In particular, while we are seduced by the symmetry of the heterogeneous definitions, they are lesser known and have proved suprisingly tricky to handle. We hope that more practice with homogeneous and heterogeneous presentations will help us understand the compromise to make; it is possible that some middle way, e.g., adding a context conversion rule to a homogeneous system, could combine the advantages of the two approaches.

Finally, one question we still feel is unresolved, probability by lack of experience and intuition with thoose tools, is "how semantic" our use of logical relation really is. While we focused in this document on its justification as a pure "proof technique" (closure by application, closure by substitution), there are very strong similarities between our construction of a logical relation from an semantic universe and explicit model-construction semantic approaches to the consistency of type systems. It is unclear to us if we could formally claim that our logical relation is, in some sense, a model construction. If not, we would like to get a better understanding of the distance between the two notions.

Appendices

A An informal introduction to Kripke logical relations

The distance between definitional and algorithmic equality Perhaps surprisingly, non-syntax-directed rules such as transitivity are actually not the main

difficulty in an equivalence proof between definitional and algorithmic system. One may imagine pushing transitivity to the leaves of the derivation, using repeated β-conversion instead of a single β-reduction step, η-expansion to canonical forms, etc.

The major difficulty is the impedance mismatch between the *compositionality* of the definitional rules and the non-compositionality of the β-reductions, transpiring through the algorithmic equality. This is problematic in at least two situations.

The first instance of this problem is in the behavior of terms with respect to reducibility. Knowing all the β-reductions of the terms t and u do not tell us much about the β-reduction behavior of their application $t\,u$. In particular, t and u may be strongly normalizing, and $t\,u$ diverge. This is the well-known difficulty in termination proofs for typed lambda-calculi, and it justifies *reducibility candidates* methods, which enrich the termination condition with other requirements, such as termination of all applications of a function to normalizing terms.

The second manifestation can be directly seen in the algorithmic equality rule: none of them behaves well with respect to algorithmic equality. Consider:

$$\frac{\Delta \vdash n \longleftrightarrow n' : (x{:}U) \to T \qquad \Delta \vdash u \mathrel{\widehat{\Longleftrightarrow}} u' : U}{\Delta \vdash n\,u \mathrel{\widehat{\longleftrightarrow}} n'\,u' : T[u/x]}$$

As a different algorithmic check is applied on the operands of the application, it is not direct to check admissibility of the application compatibility rule. For none of $\mathcal{R} \in \{\mathrel{\widehat{\Longleftrightarrow}}, \longleftrightarrow\}$ we have that $\Gamma \vdash t\ \mathcal{R}\ t'$ and $\Gamma \vdash u\ \mathcal{R}\ u'$ implies $\Gamma \vdash t\,u\ \mathcal{R}\ t'\,u'$.

Logical relation(s) The solution in both cases is to define a *logical relation* that enrich a desired relation $\mathcal{R}$ into a relation $\mathcal{R}'$ which is "closed by application": application of $\mathcal{R}'$-related functions to $\mathcal{R}'$-related arguments yield $\mathcal{R}'$-related result, that is to say, roughly:

$$\frac{\Delta \vdash t\ \mathcal{R}\ t' : (x{:}U) \to T \qquad \forall (u, u'),\ \Delta \vdash u\ \mathcal{R}'\ u' : U \implies \Delta \vdash t\,u\ \mathcal{R}'\ t'\,u' : T[u/x]}{\Delta \vdash t\ \mathcal{R}'\ t' : (x{:}U) \to T}$$

For $t\mathcal{R}t'$ defined as "t and t' are equal and weak-head-normalizing", this yields a solution to our first problem. For $\mathcal{R}$ defined as algorithmic equality, we have a solution to the second problem.

On terms whose type N is neutral – all normalized types that are not function types –, we define $\mathcal{R}'$ to coincide with $\mathcal{R}$.

$$\frac{\Delta \vdash t\ \mathcal{R}\ t' : N}{\Delta \vdash t\ \mathcal{R}'\ t' : N}$$

Such a logical relation $\mathcal{R}'$ is more restrictive than $\mathcal{R}$. What we want to ensure in general is that it is actually equal to $\mathcal{R}$, as a way to prove that $\mathcal{R}$ indeed behaves well with respect to application. When $\mathcal{R}$ is based on definitional equality, what we need to do is to show that each of the definitional rules are admissible for $\mathcal{R}'$, for example that it is transitive, and closed under β-reduction; that will give us subject reduction, and algorithmic soundness as a derived result. When $\mathcal{R}$ is algorithmic equality, we will also prove that $\mathcal{R}'$ admits the definitional equality rule, and that will give us completeness of the algorithmic equality.

We first attempted to combine the two ideas in a single relation, proving weak-head normalization – so algorithmic soundness – and algorithmic completeness at the same time. But, as demonstrated earlier (2.2, page 9), we cannot prove algorithmic transitivity without soundness, so we also failed to prove transitivity of the hybrid relation. We have to build two separate logical relations, but in practice

the technical developments are very similar and don't need to be repeated; in this report we have tried to present the logical relation construction generically, to be later instantiated in those two different cases.

Kripke logical relations The logical relations are characterized by an occurence of the relation being defined in a negative position, on the left of an implication. This may make reasoning more difficult, and in particular breaks weakening: if $\Delta \vdash t\ \mathcal{R}'\ t' : (x:U) \to T$ and Γ is an extension of Δ, we do not necessarily have $\Gamma \vdash t\ \mathcal{R}'\ t' : (x:U) \to T$ with the rule given above. Indeed, we need to prove that $\Gamma \vdash u\ \mathcal{R}'\ u' : U$ implies $\Gamma \vdash t\,u\ \mathcal{R}'\ t'\ u'$, but we only know that $\Delta \vdash u\ \mathcal{R}'\ u' : U$ implies ..., and we cannot turn a Γ hypothesis into a stronger Δ hypothesis.

The solution is to instead use *Kripke* logical relations, where this negative occurence of the relation is closed over weakening:

$$\frac{\begin{array}{c}\Delta \vdash t\ \mathcal{R}\ t' : (x:U) \to T \\ \forall \Gamma \leq \Delta, \forall (u,u'),\quad \Gamma \vdash u\ \mathcal{R}'\ u' : U \implies \Gamma \vdash t\,u\ \mathcal{R}'\ t'\ u' : T[u/x]\end{array}}{\Delta \vdash t\ \mathcal{R}'\ t' : (x:U) \to T}$$

Type computation, dependencies, and universe construction Logical relations are more complex when defined on dependent type systems with type-level computation. The example rule given above is directed by the presence of a function type $(x:U) \to T$; in a simply-typed setting, such type-definition are direct, but with type-level computation – non-necessarily dependent, e.g., F_ω – this requires normalization at the type level.

Furthermore, type dependencies blur the simple term/type separation, and the logical relation will also need to be defined on types classified by sorts, to account for the behavior of their term subparts. Roughly:

$$\frac{\begin{array}{c}\Delta \vdash (x:U) \to T\ \mathcal{R}\ (x:U') \to T' : s \qquad \Delta \vdash U\ \mathcal{R}'\ U' : s \\ \forall (u,u'),\quad \Delta \vdash u\ \mathcal{R}'\ u' : U \implies \Delta \vdash T[u/x]\ \mathcal{R}'\ T'[u'/x] : s\end{array}}{\Delta \vdash (x:U) \to T\ \mathcal{R}'\ (x:U') \to T' : s}$$

Finally, we can see that it is not obvious that such a $\mathcal{R}'$ relation is well-defined. For example, the relation $\Delta \vdash \_\ \mathcal{R}'\ \_ : (x:U) \to T$ is defined in terms of $\Delta \vdash \_\ \mathcal{R}'\ \_ : U$, that is at a structurally smaller type, but also $\Delta \vdash \_\ \mathcal{R}'\ \_ : T[u/x]$ where $T[u/x]$ is not a subterm of $(x:U) \to T$. Morally, one could say that u being an element of type U, it is smaller – live in a lower universe – than T and U, and T and $T[u/x]$ should have "the same size". But it's not even clear that $\Delta \vdash u : U$, we only know that $\Delta \vdash u\ \mathcal{R}'\ u' : U$.

To justify the well-foundedness of the logical relation, we will first build a *semantic universe* U. This is a subset of the terms whose construction ensures that all its elements T give rise to a well-founded logical relation, ie., that $\Delta \vdash \_\ \mathcal{R}'\ \_ : T$ is defined for well-formed $\Delta \vdash$. In particular, as we define our logical relations on type in weak-head normal form: we will define a universe $\mathsf{U} \subseteq \mathsf{Whnf}$, containing only normal forms, and manipulate types whose whnf is in U.

We would like to have $(x:U) \to T$ in U if and only if "$(x:U) \to T$ classifies a well-defined logical relation", that is if $U \in \mathsf{U}$ and, for all elements u of type U, $T[u/x] \in \mathsf{U}$. But it is unclear what "u is of type U" means here, we are trying to build a semantic universe independent of any typing context Δ. The solution is to embed an *approximation* of the type relation "u is of type U" directly in U, by defining instead $\mathsf{U} \subseteq \mathsf{Whnf} \times \mathcal{P}(\mathsf{Whnf})$, with $(U, \mathcal{A}) \in \mathsf{U}$ meaning "U is a type in the universe, and $\mathcal{A}$ is a context-independent approximation of its elements".

We first give some notation and then produce a satisfying inductive definition for an universe. For $\mathcal{A} \in \mathcal{P}(\mathsf{Whnf})$ we will write $\widehat{\mathcal{A}} := \{t \mid \downarrow t \in \mathcal{A}\}$. Similarly,

$\widehat{\mathsf{U}} := \{(T, \mathcal{A}) \mid (\downarrow T, \mathcal{A}) \in \mathsf{U}\}$. Finally, for $\mathcal{A} \in \mathcal{P}(\mathsf{Whnf})$ and $\mathcal{F} \in \mathsf{Whnf} \to \mathcal{P}(\mathsf{Whnf})$, a family of sets, we define the "dependent function space"

$$\Pi\, \mathcal{A}\, \mathcal{F} := \{f \in \mathsf{Whnf} \mid \forall u \in \widehat{\mathcal{A}},\ f\ u \in\ \widehat{\mathcal{F}}(\downarrow u)\}$$

We arrive at the following inductive definition, where Wne is the syntactic class of neutral normal forms defined in 2.2, page 8.

$$\frac{}{(N, \mathsf{Wne}) \in \mathsf{U}}\ N \in \mathsf{Wne} \qquad \frac{(U, \mathcal{A}) \in \widehat{\mathsf{U}} \quad \forall u \in \widehat{\mathcal{A}}, (T[u/x], \widehat{\mathcal{F}}(\downarrow u)) \in \widehat{\mathsf{U}}}{((x : U) \to T, \Pi\, \mathcal{A}\, \mathcal{F}) \in \mathsf{U}}$$

Again, this seemingly obscure definition is mostly an approximation of types and their terms independently of any context: without a context, we don't know what variables – that is, neutrals – mean, and we choose to allow them to mean *anything*: any neutral is considered an approximate type, and has any neutral as approximate element. This gross approximation is enough to justify the well-foundedness of a logical relation, as the definition of the relation $\mathcal{R}'$ on neutral types is immediate: it coincides with the original relation $\mathcal{R}$, and thus will not pose any foundedness problem.

Note that it is unclear whether this approximation indeed contains all well-typed terms, or if some of them are left out of the definition. This may be problematic as we only define our logical relation on this universe: there could be some legitimate term that is not in the logical relation because our universe approximation is not precise enough. But that will be ensured in due time: when we will prove that the logical relation $\mathcal{R}'$ is indeed equal to $\mathcal{R}$ – e.g., definitional or algorithmic equality – we will in particular have shown that all $\mathcal{R}$-related terms are in the universe approximation.

The formal presentation of the logical relation is in section 3.2, page 11.

B Level polymorphism, irrelevance and heterogeneous equality

In this section, we will quickly expose the more advanced type system features considered. We will only highlight the main ideas, both because of the space constraints and because the theory is still in flux.

We will however try to expose clearly and rigourously the problem encountered with heterogeneous equality, and a proposed solution to overcome them. We believe this may be relevant and applicable to other work on different type system features.

B.1 Level polymorphism

The changes needed to integrate level polymorphism are relatively modest, if we use the same function arrow $(x : U) \to T$ to represent terms parametrized over a level: this allows to reuse all the metatheory of functions, instead of introducing a new constructor. We want to be able to write something like $(i : \mathsf{Level}) \to (X : \mathsf{Set}\ i) \to X \to X$. This type cannont consistently live in any of the Set_i, so we need a higher universe Set_ω – on which level-polymorphism does *not* quantify.

Definitional system We introduce a new type Level, to classify universe levels, equipped of a $0 : \mathsf{Level}$ constant and a $(\_ + 1) : \mathsf{Level} \to \mathsf{Level}$ operation.

We remove the universe hierarchy Set_i, introducing instead a constant constructor Set. The type of Set is relatively suspect, as it is self-referential:

$$\mathsf{Set} : (i : \mathsf{Level}) \to \mathsf{Set}(i + 1)$$

This problem, also noted by Brown [Bro06] seems not to be an issue in practice; if it proved problematic, we could instead introduce a syntactic family $\mathsf{Set}[t]$ indexed by expressions (of type Level), and integrate the typing of the family as a rule:

$$\frac{\Delta \vdash t : \mathsf{Level}}{\Delta \vdash \mathsf{Set}[t] : \mathsf{Set}[t+1]}$$

Finally, we introduce a constant Set_ω, the universe in which live the level-polymorphic terms. Note that Set_ω is a constant which is distinct from all applications of the Set constructor — in particular, there is no "level ω". We have the following typing and cumulativity rules:

$$\frac{\Delta \vdash}{\Delta \vdash \mathsf{Level} : \mathsf{Set}_\omega} \qquad \frac{\Delta \vdash i : \mathsf{Level}}{\Delta \vdash \mathsf{Set}\ 0 \preccurlyeq \mathsf{Set}\ i} \qquad \frac{\Delta \vdash i : \mathsf{Level}}{\Delta \vdash \mathsf{Set}\ i \preccurlyeq \mathsf{Set}_\omega}$$

$$\frac{\Delta \vdash i : \mathsf{Level}}{\Delta \vdash \mathsf{Set}\ i \preccurlyeq \mathsf{Set}\ (i+1)} \qquad \frac{\Delta \vdash i_1 = i_2 : \mathsf{Level}}{\Delta \vdash \mathsf{Set}\ i_1 = \mathsf{Set}\ i_2}$$

The fact that $\mathsf{Level} : \mathsf{Set}_\omega$ forces all level-polymorphic functions to live in universe Set_ω, as it must be. Note that we do *not* have $\mathsf{Level} \preccurlyeq \mathsf{Set}_\omega$: the intuition is that elements of Level are not types, so do not need to be included in Set_ω. Finally, let's give an example of well-typed level-polymorphic definition:

$$\mathsf{Id} := \lambda i : \mathsf{Level}.\, \lambda X : \mathsf{Set}\ i.\, \lambda x : X.\, x$$

Implicit level/universes coercion The Set constructor is really a wrapper from levels to universes; this is reminiscent of a more explicit presentation of the predicative universe hierarchy where we distinguish type *codes* $c : \mathsf{Set}_i$ and the corresponding *type* $\mathsf{El}(c)$.

We have considered removing the Set constructor and quantifying directly on universes, $\mathsf{Id} := \lambda s : \mathsf{Set}_\omega.\, \lambda X : s.\, \lambda x : X.\, x$, but then the successor operation $\_ + 1$ needs to have type $\mathsf{Set}_\omega \to \mathsf{Set}_\omega$, and can also be applied on types, despite there are not universes.

It would also be possible to distinguish a Sets constant inhabited only by the universes, with rules $\mathsf{Set}_i : \mathsf{Sets}$ and $\mathsf{Sets} \preccurlyeq \mathsf{Set}_\omega$. But that would break the principal types, as we would have $\mathsf{Set}_i : \mathsf{Set}_{i+1}$ and $\mathsf{Set}_i : \mathsf{Sets}$, whith none more general than the other.

We have therefore choosed to stay safe by keeping the explicit level/universe separation.

Algorithmic system and logical relation The algorithmic rules also need to be adapted for universe polymorphism. The changes are minimal and natural, and won't be detailed in this repart.

On the semantic side, we add a two new universes U_ω and $\mathsf{U}_{\mathsf{poly}}$ to the hierarchy.

$$\mathsf{U}_\omega := \bigcup_{i \in \mathbb{N}} \mathsf{U}_i \qquad \mathsf{U}_{\mathsf{poly}} := \mathsf{U}_\omega \cup \{(\mathsf{Set}_\omega, |\mathsf{U}_\omega|), (\mathsf{Level}, \mathbb{N} \cup \mathsf{Wne})\}$$

The logical relation is defined by induction over the membership to the universe; for $T, T' \in |\mathsf{U}_i|$, $\Delta \vdash_{\mathcal{L}} T = T' : \mathsf{Set}_i$ and $\Delta \vdash_{\mathcal{L}} t = t' : T$ are defined. We extend it to relations $\Delta \vdash_{\mathcal{L}} T = T' : \mathsf{Set}_\omega$ when $T, T' \in \mathsf{U}_\omega$.

$$\frac{\Delta \vdash}{\Delta \vdash_{\mathcal{L}} \mathsf{Level} = \mathsf{Level} : \mathsf{Set}_\omega} \qquad \frac{\Delta \vdash_{\mathcal{L}} i_1 = i_2 : \mathsf{Level}}{\Delta \vdash_{\mathcal{L}} \mathsf{Set}\ i_1 = \mathsf{Set}\ i_2 : \mathsf{Set}_\omega}$$

A type that cannot be typed Adding this supplementary universe Set_ω immediately prompts the question: "and after that?" We could imagine a level $\mathsf{Set}_{\omega+1}$, or quantifying on all $\mathsf{Set}_{\omega+k}$ at $\mathsf{Set}_{\omega*2}$... "and after that?" While proof theorists have investigated ordinal hierarchies of dyzzying strength, we think Set_ω is a reasonable first step for a practical programming language.

An important difference with the previous theory is that Set_ω is a type that cannot itself be typed. This means you cannot abstract over a variable x of type Set_ω, as this would require $\Gamma \vdash \mathsf{Set}_\omega : s$.

If this difference proved too perturbing to the user, we could always add "turtles all the way down" by grafting an infinite universe hierarchy $\mathsf{Set}_{\omega+i}$, just to regain that property that all universes are typeable by a successor universe.

B.2 Irrelevance

The work on irrelevance builds on previous work by Abel [Abe11], which integrates an irrelevant function space in the present type system, using an older presentation of the metatheoretic techniques presented here. The yet unpublished article with Abel [AS11] reinstates those results in the current version of the theory — without cumulativity, but this is an orthogonal aspect.

Complete irrelevance As presented in the introduction (1, page 5), the irrelevant function space, written $(x \dot{:} U) \to T$, represent functions that do not use the argument, except in irrelevant positions. The arguments of irrelevant functions $\lambda x \dot{:} U.\, t$ are marked as irrelevant in the context $\Gamma.\, x \dot{:} U$; they may not be used as usual variables as the variable rule require $(x : U) \in \Gamma$, but become usable in the argument of irrelevant functions, which are type-checked in a "resurrected context" $\Gamma^{\dot{:}}$ which turns all irrelevant $x \dot{:} U$ into usable $x : U$ again.

$$\frac{\Gamma.\, x \dot{:} U \vdash t : T}{\Gamma \vdash \lambda x \dot{:} U.\, t : (x \dot{:} U) \to T} \qquad \frac{\Gamma \vdash t : (x \dot{:} U) \to T \quad \Gamma^{\dot{:}} \vdash u : U}{\Gamma \vdash t \mathbin{\dot{:}} u : T[u/x]}$$

To check equality between two applications $f \mathbin{\dot{:}} u$, $f' \mathbin{\dot{:}} u'$ to irrelevant functions, we only need to check equality of functions, and well-typedness – and not equality – of the arguments. At the definitional, algorithmic and logical relation levels, where $\Gamma \vdash_{\mathcal{L}} t : T$ means $\Gamma \vdash_{\mathcal{L}} t = t : T$, and is a "logical well-typedness check":

$$\frac{\Gamma \vdash t = t' : (x \dot{:} U) \to T \quad \Gamma^{\dot{:}} \vdash u : U \quad \Gamma^{\dot{:}} \vdash u' : U}{\Gamma \vdash t \mathbin{\dot{:}} u = t' \mathbin{\dot{:}} u' : T[u/x]} \qquad \frac{\Delta \vdash n \mathrel{\widehat{\longleftrightarrow}} n' : (x \dot{:} U) \to T}{\Delta \vdash n \mathbin{\dot{:}} u \mathrel{\widehat{\longleftrightarrow}} n' \mathbin{\dot{:}} u' : T[u/x]}$$

$$\frac{\forall \Gamma \leq \Delta,\ \Gamma \vdash_{\mathcal{L}} u : U \wedge \Gamma \vdash_{\mathcal{L}} u' : U \implies \Gamma \vdash_{\mathcal{L}} t \mathbin{\dot{:}} u = t' \mathbin{\dot{:}} u' : T[u/x]}{\Delta \vdash_{\mathcal{L}} t = t' : (x \dot{:} U) \to T}$$

Shape irrelevance We would like levels to be irrelevant to allow for more equalities, e.g., $\vdash \mathsf{Id}\ 0 = \mathsf{Id}\ (0+1)$, but also to make already-true equalities less tedious to prove by removing the level-equality proof obligations. The previous notion of irrelevance is unfortunately too strong for universe levels, as making our set constructor completely irrelevant, $\mathsf{Set} : (i \dot{:} \mathsf{Level}) \to \mathsf{Set}\ (i+1)$, would collapse all universes, e.g., $\diamond \vdash \mathsf{Set}\ 1 = \mathsf{Set}\ 2$.

The solution we currently explore, very speculatively, would be to add a difference irrelevant marker, *shape irrelevance* $\ddot{:}$, that woud allow equalities between terms using different universes *in their types*, but not between different universes *as terms*.

One way to do this is to have two equality modes, $\Gamma \vdash t = t' : T$ for the usual, or *strict*, equality, and $\Gamma \vdash t \doteq t' : T$ for *shape equality*. In strict equality, the $\ddot{:}$

irrelevance marker is basically ignored, and in shape equality it is transformed into the irrelevant $\div$ marker.

But with this design, the homogeneous equality used so far raises new problems. We have already discussed its inherent asymmetry, manifest for example in the function equality rule — we use $\star$ as "relevance marker variables" to quantify over $\{:, \dddot{:}, \div\}$

$$\frac{\Gamma \vdash U \doteq U' : s \quad \Gamma.\, x\star U \vdash t = t' : T}{\Gamma \vdash \lambda x\star U.\, t = \lambda x\star U'.\, t' : (x\star U) \to T}$$

The arbitrary choice of U over U' in the subcheck of $t = t'$ makes syntactic validity, specifically proving that $\Gamma.\, x\star U \vdash t' : T$, difficult. But the situation is even worse with shape-irrelevance, consider for example $\lambda X : \mathsf{Set}\, i.\, t = \lambda X : \mathsf{Set}\, j.\, t'$; shape equality $\mathsf{Set}\, i \doteq \mathsf{Set}\, j$ will hold even for unrelated i, j, and then $\Gamma.\, X : \mathsf{Set}_j \vdash t' : T$ may *not* hold, breaking syntactic validity.

This issue has encouraged us to consider heterogeneous equality again, which we will shortly present in the next subsection.

B.3 Heterogeneous equality

A solution to avoid the asymmetry problems of the equality check $\Gamma \vdash t = t' : T$ is to move to *heterogeneous equality*, where contexts and types may differ on both sides of the equal sign: $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$. We preserve the invariant that Γ, Γ' have the same domains, and the intuition in that the type in each, as well as T, T', are pairwise equal upto definitional equality. For example, the function equality rule becomes:

$$\frac{\Gamma \vdash U : s = \Gamma' \vdash U' : s' \qquad \Gamma.\, x\star U \vdash t : T = \Gamma'.\, x\star U' \vdash t' : T'}{\Gamma \vdash \lambda x\star U.\, t : (x\star U) \to T = \Gamma' \vdash \lambda x\star U'.\, t' : (x\star U') \to T'}$$

While syntactically heavier, this rule has the advantage of preserving syntactic validity: while in the homogeneous setting it is a non-trivial result to prove that $\Gamma.\, x\star U \vdash t' : T$ holds, it is there immediate that we have $\Gamma'.\, x\star U' \vdash t' : T'$. A result that is achieved by the semantic method of logical relations in our previous development becomes trivial, strengthening the syntactic virtues of our definitional system.

Algorithmic equality can easily, if heavily, be adapted to the heterogeneous setting. A particularly compelling aspect of heterogeneous algorithmic equality is that, as the asymmetry has been removed, there is nothing in the way of transitivity anymore; we can easily prove that algorithmic equality is a P.E.R. However, this does not seem to be enough to reduce the formal development to one single logical relation instead of two; context conversion for algorithmic equality still seems problematic in absence of soundness, even for heterogeneous equality.

We can make the intuition that context are pairwise equal more precise, by defining a *context equality* relation $\Delta \vdash = \Delta' \vdash$, reminiscent of the $\Gamma \preccurlyeq \Gamma' \vdash$ relation defined previously (4.1, page 18):

$$\frac{}{\diamond \vdash = \diamond \vdash} \qquad \frac{\Gamma \vdash = \Gamma' \vdash \quad \Gamma \vdash U : s = \Gamma' \vdash U' : s'}{\Gamma.\, x\star U \vdash = \Gamma'.\, x\star U' \vdash}$$

It is then immediate than $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$ implies $\Gamma \vdash = \Gamma' \vdash$.

The logical transitivity failure All is well until we consider the logical relation:

$$\frac{\begin{array}{c}\Delta \vdash_{\mathcal{B}} t : (x\star U) \to T = \Delta' \vdash_{\mathcal{B}} t' : (x\star U') \to T' \\ \Delta \vdash_{\mathcal{L}} U : s = \Delta' \vdash_{\mathcal{L}} U' : s' \\ \forall (\Gamma, \Gamma') \leq (\Delta, \Delta'), \\ \quad \Gamma \vdash_{\mathcal{L}} u \star U = \Gamma' \vdash_{\mathcal{L}} u' \star U' \implies \Gamma \vdash_{\mathcal{L}} t \,^\star u : T[u/x] = \Gamma' \vdash_{\mathcal{L}} t' \,^\star u' : T'[u'/x]\end{array}}{\Delta \vdash_{\mathcal{L}} t : (x\star U) \to T = \Delta' \vdash_{\mathcal{L}} t' : (x\star U') \to T'}$$

The main problem with this definition is not the awkwardness of the $\vdash_{\mathcal{L}}$ syntax in an heterogeneous setting, but that transitivity seems to fail. Indeed, suppose we have:

$$\begin{array}{c}\Delta_1 \vdash_{\mathcal{L}} U_1 : s_1 = \Delta_2 \vdash_{\mathcal{L}} U_2 : s_2 \\ \forall (\Gamma_1, \Gamma_2) \leq (\Delta_1, \Delta_2), \\ \Gamma_1 \vdash_{\mathcal{L}} u \star U_1 = \Gamma_2 \vdash_{\mathcal{L}} u' \star U_2 \implies \Gamma_1 \vdash_{\mathcal{L}} t_1 \,^\star u : T_1[u/x] = \Gamma_2 \vdash_{\mathcal{L}} t_2 \,^\star u : T_2[u'/x] \\ \Delta_2 \vdash_{\mathcal{L}} U_2 : s_2 = \Delta_3 \vdash_{\mathcal{L}} U_3 : s_3 \\ \forall (\Gamma_2, \Gamma_3) \leq (\Delta_2, \Delta_3), \\ \Gamma_2 \vdash_{\mathcal{L}} u \star U_2 = \Gamma_3 \vdash_{\mathcal{L}} u' \star U_3 \implies \Gamma_2 \vdash_{\mathcal{L}} t_2 \,^\star u_2 : T_2[u/x] = \Gamma_3 \vdash_{\mathcal{L}} t_3 \,^\star u_3 : T_3[u'/x]\end{array}$$

and want to prove

$$\begin{array}{c}\Delta_1 \vdash_{\mathcal{L}} U_1 : s_1 = \Delta_3 \vdash_{\mathcal{L}} U_3 : s_3 \\ \forall (\Gamma_1, \Gamma_3) \leq (\Delta_1, \Delta_3), \\ \Gamma_1 \vdash_{\mathcal{L}} u \star U_1 = \Gamma_3 \vdash_{\mathcal{L}} u' \star U_3 \implies \Gamma_1 \vdash_{\mathcal{L}} t_1 \,^\star u : T_1[u/x] = \Gamma_3 \vdash_{\mathcal{L}} t_3 \,^\star u' : T_3[u'/x]\end{array}$$

Proving $U_1 = U_3$ still works inductively. But then, given context (Γ_1, Γ_3) we have a (u, u') pair at (U_1, U_3). What we want to do is, as in the homogeneous proof (3.2, page 14), to use type conversion to get (u, u) at (U_1, U_2) and (u, u') at (U_2, U_3), before using transitivity at T. But to do this, we shall provide some context $\Gamma_2 \leq \Delta_2$. There is no immediate choice for this Γ_2, and because of that no immediate proof of transitivity. We didn't have this problem in the homogeneous case, because all implications were quantified over the same context $\Gamma \leq \Delta$.

If we cannot prove transitivity of the logical relation, then we don't have a fundamental lemma, as it relies on the fact that all definitional rules are admissible, in particular transitivity.

Context conversion Our intuition of why the transitivity of the logical relation *should* hold is that if Δ_2 and Δ_3, really have pairwise (shape-)equal types, then we should be able to build a suitable $\Gamma_2 \leq \Delta_2$ by "applying to Γ_3 the changes between Δ_3 and Δ_2".

This intuition can be made precise by defining a *context conversion* operation $\{\Delta \Rightarrow \Delta'\}$ for equal contexts $\Delta \vdash = \Delta' \vdash$, that would transform a well-typed term in context Δ to a corresponding well-typed term in context Δ'. This can be done by a simple *directed* reading of the already-existing compatibility rules for heterogeneous equality, $\Gamma \vdash t : T \Rightarrow \Gamma' \vdash t' : T'$, for example:

$$\frac{(x : U) \in \Gamma \quad (x : U') \in \Gamma'}{\Gamma \vdash x : U \Rightarrow \Gamma' \vdash x : U'}$$

$$\frac{\Gamma \vdash U : s \Rightarrow \Gamma' \vdash U' : s' \qquad \Gamma.\, x\star U \vdash t : T \Rightarrow \Gamma'.\, x\star U' \vdash t' : T'}{\Gamma \vdash \lambda x\star U.\, t : (x\star U) \to T \Rightarrow \Gamma' \vdash \lambda x\star U'.\, t' : (x\star U') \to T'}$$

Given $\Gamma \vdash = \Gamma' \vdash$ and a well-typed t ($\Gamma \vdash t : T$ for some T), $\Gamma \vdash t : T \Rightarrow \Gamma \vdash t' : T'$ always holds. As the compositionality rules are syntax-directed, it uniquely defines a t'. We write $t\{\Gamma \Rightarrow \Gamma'\}$ for such a t'.

Lemma 44 *Context conversion also applies on the type component: if* $\Gamma \vdash = \Gamma' \vdash$ *and* $\Gamma \vdash t : T$*, then* $\Gamma \vdash t\{\Gamma \Rightarrow \Gamma'\} : T\{\Gamma \Rightarrow \Gamma'\}$.

We can then extend $\{\Gamma \Rightarrow \Gamma'\}$ on contexts $\Delta \leq \Gamma$:

- $\Gamma\{\Gamma \Rightarrow \Gamma'\} := \Gamma'$
- If $\Delta \leq \Gamma$, then $(\Delta.\, x \star U)\{\Gamma \Rightarrow \Gamma'\} := \Delta\{\Gamma \Rightarrow \Gamma'\}.\, x \star (U\{\Gamma \Rightarrow \Gamma'\})$

Lemma 45 (Context conversion is compatible with weakening) *If* $\Delta \leq \Gamma$*, then* $\Delta\{\Gamma \Rightarrow \Gamma'\} \leq \Gamma'$.

Logical transitivity with context conversion The previous definition of context conversion was done in the definitional setting. Using the very similar properties of type conversion at the logical relation level, we are able to transpose the definition to logical context relation $\Gamma \vdash_{\mathcal{L}} = \Gamma' \vdash_{\mathcal{L}}$. We will write $\{\Gamma \Rightarrow \Gamma'\}$ as well for this new operation, as it is always clear from the context which is applied.

The $\{\Gamma \Rightarrow \Gamma'\}$ operation is used in the heterogeneous version of the type conversion lemma — lemma which was used in the homogeneous proof of transitivity of the logical relation.

Lemma 46 (Type conversion) *If* $\Delta_1 \vdash_{\mathcal{L}} t_1 : T_1 = \Delta_2 \vdash_{\mathcal{L}} t_2 : T_2$ *and* $\Delta_2 \vdash_{\mathcal{L}} T_2 : s_2 = \Delta_3 \vdash_{\mathcal{L}} T_3 : s_3$ *then* $\Delta_1 \vdash_{\mathcal{L}} t_1 : T_1 = \Delta_3 \vdash_{\mathcal{L}} t_2\{\Delta_2 \Rightarrow \Delta_3\} : T_3$.

We can now proceed with the proof of transitivity. The core idea is that we had $\Delta_1 \vdash = \Delta_2 \vdash = \Delta_3 \vdash$, plus $(\Gamma_1, \Gamma_3) \leq (\Delta_1, \Delta_3)$, but we missed a $\Gamma_2 \leq \Delta_2$ to use the hypothesis implications. We can now build it, using context conversion, as either $\Gamma_1\{\Delta_1 \Rightarrow \Delta_2\}$ or $\Gamma_3\{\Delta_3 \Rightarrow \Delta_2\}$; both choices make the proof proceed.

References

[Abe11] Andreas Abel. Irrelevance in type theory with a heterogeneous equality judgement. In Martin Hofmann, editor, *Foundations of Software Science and Computational Structures, 14th International Conference, FOSSACS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26 - April 3, 2011. Proceedings*, volume 6604 of *Lecture Notes in Computer Science*, pages 57–71. Springer-Verlag, 2011.

[ACD07] Andreas Abel, Thierry Coquand, and Peter Dybjer. Normalization by evaluation for Martin-Löf Type Theory with typed equality judgements. In *22nd IEEE Symposium on Logic in Computer Science (LICS 2007), 10-12 July 2007, Wroclaw, Poland, Proceedings*, pages 3–12. IEEE Computer Society Press, 2007.

[Ama08] Roberto M. Amadio, editor. *Foundations of Software Science and Computational Structures, 11th International Conference, FOSSACS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29 - April 6, 2008. Proceedings*, volume 4962 of *Lecture Notes in Computer Science*. Springer-Verlag, 2008.

[AS11] Andreas Abel and Gabriel Scherer. Irrelevance in type theory with a heterogeneous equality judgement. Unpublished, 2011.

[BB08] Bruno Barras and Bruno Bernardo. The implicit calculus of constructions as a programming language with dependent types. In Amadio [Ama08], pages 365–379.

[Bro06] Daniel Brown. Exploring universe polymorphism in omega. Master's thesis, 2006.

[BW97] Bruno Barras and Benjamin Werner. Coq in coq. Technical report, 1997.

[Coq91] Thierry Coquand. An algorithm for testing conversion in type theory. In G. Huet and G. Plotkin, editors, *Logical Frameworks*, pages 255–279. Cambridge University Press, 1991.

[Cou02] Judicaël Courant. Explicit universes for the calculus of constructions. In *Proceedings of the 15th International Conference on Theorem Proving in Higher Order Logics*, pages 115–130, 2002.

[Fef04] Solomon Feferman. Typical ambiguity: Trying to have your cake and eat it too. In *One Hundred Years Of Russell's Paradox: Mathematics, Logic, Philosophy*, pages 135–151. Walter De Gruyter Inc., 2004.

[Gog00] Healfdene Goguen. A Kripke-style model for the admissibility of structural rules. In Paul Callaghan, Zhaohui Luo, James McKinna, and Robert Pollack, editors, *Types for Proofs and Programs, International Workshop, TYPES 2000, Durham, UK, December 8-12, 2000, Selected Papers*, volume 2277 of *Lecture Notes in Computer Science*, pages 112–124. Springer-Verlag, 2000.

[HP91] Robert Harper and Robert Pollack. Type checking with universes. *Theoretical Computer Science*, 89:107–136, 1991. doi:10.1016/0304-3975(90)90108-T.

[HP05] Robert Harper and Frank Pfenning. On equivalence and canonical forms in the LF type theory. *ACM Transactions on Computational Logic*, 6(1):61–101, 2005. doi:http://doi.acm.org/10.1145/1042038.1042041.

[Luo90] Zhaohui Luo. *ECC: An Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, 1990. Available from: http://www.lfcs.inf.ed.ac.uk/reports/90/ECS-LFCS-90-118/.

[McB11] Conor McBride. Crude but effective stratification, 2011. Available from: http://www.e-pig.org/epilogue/?p=857.

[Miq01] Alexandre Miquel. The implicit calculus of constructions. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001, Krakow, Poland, May 2-5, 2001, Proceedings*, volume 2044 of *Lecture Notes in Computer Science*, pages 344–359. Springer-Verlag, 2001.

[MLS08] Nathan Mishra-Linger and Tim Sheard. Erasure and polymorphism in pure type systems. In Amadio [Ama08], pages 350–364.

[Nor07] Ulf Norell. *Towards a Practical Programming Language Based on Dependent Type Theory*. PhD thesis, Göteborg, Sweden, September 2007.

[Pfe01] Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *16th IEEE Symposium on Logic in Computer Science (LICS 2001), 16-19 June 2001, Boston University, USA, Proceedings*. IEEE Computer Society Press, 2001.

[Ree03] Jason Reed. Extending higher-order unification to support proof irrelevance. In David A. Basin and Burkhart Wolff, editors, *Theorem Proving in Higher Order Logics, 16th International Conference, TPHOLs 2003, Rom, Italy, September 8-12, 2003, Proceedings*, volume 2758 of *Lecture Notes in Computer Science*, pages 238–252. Springer-Verlag, 2003.

[VC02] Joseph C. Vanderwaart and Karl Crary. A simplified account of the metatheory of Linear LF. In *Third International Workshop on Logical Frameworks and Metalanguages (LFM 2002), FLoC'02 affiliated workshop, Copenhagen, Denmark*, 2002. An extended version appeared as CMU Technical Report CMU-CS-01-154.