Irrelevance in Type Theory with a Heterogeneous Equality Judgement

Andreas Abel

Department of Computer Science
Ludwig-Maximilians-University Munich
andreas.abel@ifi.lmu.de

Abstract. Dependently typed programs contain an excessive amount of static terms which are necessary to please the type checker but irrelevant for computation. To obtain reasonable performance of not only the compiled program but also the type checker such static terms need to be erased as early as possible, preferably immediately after type checking. To this end, Pfenning's type theory with irrelevant quantification, that models a distinction between static and dynamic code, is extended to universes and large eliminations. Novel is a heterogeneously typed implementation of equality which allows the smooth construction of a universal Kripke model that proves normalization, consistency and decidability.

Keywords: dependent types, proof irrelevance, heterogeneously typed equality, algorithmic equality, logical relation, universal Kripke model.

1 Introduction and Related Work

Dependently typed programming languages such as Agda [9], Coq [13], and Epigram [15] allow the programmer to express in one language programs, their types, rich invariants, and even proofs of these invariants. Besides code executed at run-time, dependently typed programs contain much code needed only to pass the type checker, which is a the same time the verifier of the proofs woven into the program.

Program extraction takes type-checked terms and discards parts that are irrelevant for execution. Augustsson's dependently typed functional language Cayenne [6] erases *types* using a universe-based analysis. Coq's extraction procedure has been designed by Paulin-Mohring and Werner [21] and Letouzey [14] and discards not only types but also proofs. The erasure rests on Coq's universe-based separation between propositional (Prop) and computational parts (Set/Type). The rigid Prop/Set distinction has the drawback of code duplication: A structure which is sometimes used statically and sometimes dynamically needs to be coded twice, once in Prop and once in Set.

An alternative to the fixed Prop/Set-distinction is to let the usage context decide whether a term is a proof or a program. Besides whole-program analyses such as data flow, some type-based analyses have been put forward. One of them is Pfenning's modal type theory of *Intensionality, Extensionality, and Proof Irrelevance* [22] which introduces functions with irrelevant arguments that play the role of proofs. Not only can these arguments be erased during extraction, they can also be disregarded in type conversion tests during type checking. This relieves the user of unnecessary proof burden

(proving that two proofs are equal). Furthermore, proofs can not only be discarded during program extraction but directly after type checking, since they will never be looked at again during type checking subsequent definitions.

In principle, we have to distinguish "post mortem" program extraction, let us call it *external erasure*, and proof disposal during type checking, let us call it *internal erasure*. External erasure deals with closed expressions, programs, whereas internal erasure deals with open expressions that can have free variables. Such free variables might be assumed proofs of (possibly false) equations and block type casts, or (possibly false) proofs of well-foundedness and prevent recursive functions from unfolding indefinitely. For type checking to not go wrong or loop, those proofs can only be externally erased, thus, the Prop/Set distinction is not for internal erasure. In Pfenning's type theory, proofs can never block computations even in open expressions (other than computations on proofs), thus, internal erasure is sound.

Miquel's Implicit Calculus of Constructions (ICC) [17] goes further than Pfenning and considers also *parametric* arguments as irrelevant. These are arguments which are irrelevant for function execution but relevant during type conversion checking. Such arguments may only be erased in function application but not in the associated type instantiation. Barras and Bernardo [8] and Mishra-Linger and Sheard [19] have build decidable type systems on top of ICC, but both have not fully integrated inductive types and types defined by recursion (large eliminations). Barras and Bernardo, as Miquel, have inductive types only in the form of their impredicative encodings, Mishra-Linger [20] gives introduction and elimination principles for inductive types by example, but does not show normalization or consistency.

Our long-term goal is to equip Agda with internal and external erasure. To this end, a type theory for irrelevance is needed that supports user-defined data types and functions and types defined by pattern matching. Experiments with my prototype implementation MiniAgda [2] have revealed some issues when combining Miquel-style irrelevance with large eliminations (see Ex. 2 in Sec. 2). Since it is unclear whether these issues can be resolved, I have chosen to scale Pfenning's notion of proof irrelevance up to inductive types.

In this article, we start with the "extensionality and proof irrelevance" fragment of Pfenning's type theory in Reed's version [23, 24]. We extend it by a hierarchy of predicative universes, yielding *Irrelevant Intensional Type Theory* IITT (Sec. 2). Based on a heterogeneous algorithmic equality which compares two expressions, each in its own context at its own type (Sec. 3), we smoothly construct a Kripke model that is both sound and complete for IITT (Sec. 4). It allows us to prove soundness and completeness of algorithmic equality, normalization, subject reduction, consistency, and decidability of typing in one go (Sec. 5). The model is ready for data types, large eliminations, types with extensionality principles, and internal erasure (Sec. 6).

The novel technical contributions of this work are a heterogeneous formulation of equality in the specification of type theory, and the universal Kripke model that yields all interesting meta-theoretic results at once.

The Kripke model is inspired by previous work on normalization by evaluation [3]. There we have already observed that a heterogeneous treatment of algorithmic equality solves the problem of defining a Kripke logical relation that shows completeness of

algorithmic equality. Harper and Pfenning [12] hit the same problem, and their fix was to erase dependencies in types. In weak type theories like the logical framework erasure is possible, but it does not scale to large eliminations.

Related to our present treatment of IITT is Goguen's *Typed Operational Semantics* [11]. He proves meta-theoretic properties such as normalization, subject reduction, and confluence by a Kripke logical predicate of well-typed terms. However, his notion of equality is based on reduction and not a step-wise algorithm.

Awodey and Bauer [7] give a categorical treatment of proof irrelevance which is very similar to Pfenning and Reed's. However, they work in the setting of Extensional Type Theory with undecidable type checking, I could not directly use their results for this work.

Due to lack of space, proofs have been mostly omitted; more proofs are available in an extended version of this article on the author's home page.

2 Irrelevant Intensional Type Theory

In this section, we present *Irrelevant Intensional Type Theory* IITT which features two of Pfenning's function spaces [22], the ordinary "extensional" $(x : U) \to T$ and the proof irrelevant $(x \div U) \to T$. The main idea is that the argument of a $(x \div U) \to T$ function is counted as a proof and can neither be returned nor eliminated on, it can only be passed as argument to another proof irrelevant function or data constructor. Technically, this is realized by annotating variables as relevant, $x : U$, or irrelevant, $x \div U$, in the typing context, to restrict the use of irrelevant variables to use in irrelevant arguments.

Expression and context syntax. We distinguish between relevant ($t\ u$ or simply $t\ u$) and irrelevant application ($t \stackrel{\cdot}{} u$). Accordingly, we have relevant ($\lambda x : U.\, T$) and irrelevant abstraction ($\lambda x \div U.\, T$). Our choice of typed abstraction is not fundamental; a bidirectional type-checking algorithm [10] can reconstruct type and relevance annotations at abstractions and applications.

$\mathsf{Var}\ \ni x, y, X, Y$			
$\mathsf{Sort} \ni s$	$::= \mathsf{Set}_k\ \ (k \in \mathbb{N})$		universes
$\mathsf{Ann} \ni \star$	$::= \div \mid :$		annotation: irrelevant, relevant
$\mathsf{Exp} \ni t, u, T, U$	$::= s \mid (x \star U) \to T$		sort, (ir)relevant function type
	$\mid x \mid \lambda x \star U.\, t \mid t\ ^\star u$		lambda-calculus
$\mathsf{Cxt} \ni \Gamma, \Delta$	$::= \diamond \mid \Gamma.\, x \star T$		empty, (ir)relevant extension

Expressions are considered modulo α-equality, we write $t \equiv t'$ when we want to stress that t and t' identical (up to α).

Sorts. IITT is a pure type system (PTS) with infinite hierarchy of predicative universes $\mathsf{Set}_0 : \mathsf{Set}_1 : ....$ The universes are not cumulative. We have the PTS axioms $\mathsf{Axiom} = \{(\mathsf{Set}_i, \mathsf{Set}_{i+1}) \mid i \in \mathbb{N}\}$ and the rules $\mathsf{Rule} = \{(\mathsf{Set}_i, \mathsf{Set}_j, \mathsf{Set}_{\max(i,j)}) \mid i, j \in \mathbb{N}\}$. As customary, we will write the side condition $(s, s') \in \mathsf{Axiom}$ just as (s, s') and likewise $(s_1, s_2, s_3) \in \mathsf{Rule}$ just as (s_1, s_2, s_3). IITT is a full and functional PTS, which means

that for all s_1, s_2 there is exactly one s_3 such that (s_1, s_2, s_3). As a consequence, there is no subtyping, types are unique up to equality.

Substitutions σ are maps from variables to expressions. We require that the domain $\mathsf{dom}(\sigma) = \{x \mid \sigma(x) \neq x\}$ is finite. We write id for the identity substitution and $[u/x]$ for the singleton substitution σ with $\mathsf{dom}(\sigma) = \{x\}$ and $\sigma(x) = u$. Capture avoiding parallel substitution of σ in t is written as juxtaposition $t\sigma$.

Contexts Γ feature two kinds of bindings, relevant $(x : U)$ and irrelevant $(x \div U)$ ones. Only relevant variables are in scope in an expression. Resurrection $\Gamma^{\oplus}$ turns all irrelevant bindings $x \div T$ into relevant $x : T$ ones [22]. It is the tool to make irrelevant variables, also called proof variables, available in proofs. Extending context Γ by some bindings to context Δ is written $\Delta \leq \Gamma$.

Judgements of IITT.

$\vdash \Gamma$	context Γ is well-formed
$\vdash \Gamma = \vdash \Gamma'$	contexts Γ and Γ' are well-formed and equal
$\Gamma \vdash t : T$	in context Γ, expression t has type T
$\Gamma \vdash t : T = \Gamma' \vdash t' : T'$	typed expressions t and t' are equal

Derived judgements.

$\Gamma \vdash t \div T$	$\iff$	$\Gamma^{\oplus} \vdash t : T$
$\Gamma \vdash t \div T = \Gamma' \vdash t' \div T'$	$\iff$	$\Gamma \vdash t \div T$ and $\Gamma' \vdash t' \div T'$
$\Gamma \vdash t = t' \star T$	$\iff$	$\Gamma \vdash t \star T = \Gamma \vdash t' \star T$
$\Gamma \vdash T$	$\iff$	$\Gamma \vdash T : s$ for some s
$\Gamma \vdash T = \Gamma' \vdash T'$	$\iff$	$\Gamma \vdash T : s = \Gamma' \vdash T' : s'$ for some s, s'
$\Gamma \vdash T = T'$	$\iff$	$\vdash \Gamma$ and $T \equiv s \equiv T'$ or $\Gamma \vdash T = \Gamma \vdash T'$

Context well-formedness and typing $\vdash \Gamma$ and $\Gamma \vdash t : T$, extending Reed [23] to PTS style. Note that there is no variable rule for irrelevant bindings $(x \div U) \in \Gamma$.

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

$$\frac{\vdash \Gamma}{\Gamma \vdash s : s'}(s, s') \qquad \frac{\Gamma \vdash U : s_1 \qquad \Gamma.\, x \star U \vdash T : s_2}{\Gamma \vdash (x \star U) \to T : s_3}\,(s_1, s_2, s_3)$$

$$\frac{\vdash \Gamma \qquad (x : U) \in \Gamma}{\Gamma \vdash x : U} \qquad \frac{\Gamma.\, x \star U \vdash t : T}{\Gamma \vdash \lambda x \star U.\, t : (x \star U) \to T}$$

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

When we apply an irrelevant function $\Gamma \vdash t : (x \div U) \to T$ to u, the argument u is typed in the resurrected context $\Gamma^{\oplus} \vdash u : U$. This means that u is treated as a proof and the proof variables become available.

Parallel computation (β) and extensionality (η).

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

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

Equivalence rules.

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

$$\frac{\Gamma_1 \vdash t_1 : T_1 = \Gamma_2 \vdash t_2 : T_2 \qquad \Gamma_2 \vdash t_2 : T_2 = \Gamma_3 \vdash t_3 : T_3}{\Gamma_1 \vdash t_1 : T_1 = \Gamma_3 \vdash t_3 : T_3}$$

Compatibility rules.

$$\frac{\vdash \Gamma = \vdash \Gamma'}{\Gamma \vdash s : s' = \Gamma' \vdash s : s'}\ (s, s') \qquad \frac{(x{:}U) \in \Gamma \quad \Gamma \vdash U : s = \Gamma' \vdash U' : s' \quad (x{:}U') \in \Gamma'}{\Gamma \vdash x : U = \Gamma' \vdash x : U'}$$

$$\frac{\begin{array}{c}\Gamma \vdash U : s_1 = \Gamma' \vdash U' : s_1' \\ \Gamma.\, x\star U \vdash T : s_2 = \Gamma'.\, x\star U' \vdash T' : s_2'\end{array}}{\Gamma \vdash (x\star U) \to T : s_3 = \Gamma' \vdash (x\star U') \to T' : s_3'}$$

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

$$\frac{\begin{array}{c}\Gamma \vdash t : (x{:}U) \to T = \Gamma' \vdash t' : (x{:}U') \to T' \\ \Gamma \vdash u : U = \Gamma' \vdash u' : U'\end{array}}{\Gamma \vdash t\, u : T[u/x] = \Gamma' \vdash t'\, u' : T'[u'/x]} \qquad \frac{\begin{array}{c}\Gamma \vdash t : (x \div U) \to T = \Gamma' \vdash t' : (x \div U') \to T' \\ \Gamma^{\oplus} \vdash u : U \qquad \Gamma'^{\oplus} \vdash u' : U'\end{array}}{\Gamma \vdash t \div u : T[u/x] = \Gamma' \vdash t' \div u' : T'[u'/x]}$$

Conversion rule.

$$\frac{\Gamma_1 \vdash t_1 : T_1 = \Gamma_2 \vdash t_2 : T_2 \qquad \Gamma_2 \vdash T_2 = T_2'}{\Gamma_1 \vdash t_1 : T_1 = \Gamma_2 \vdash t_2 : T_2'}$$

Fig. 1. Rules of heterogeneous equality

Equality. Figure 1 presents the rules to construct the judgement $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$. The novelty is the heterogeneous typing: we do not locally enforce that equal terms must have equal types, but we will show it globally in Sec. 5. Note that in the compatibility rule for irrelevant application, the function arguments may be completely unrelated.

In heterogeneous judgements such as equality, we maintain the invariant that the two contexts Γ and Γ' have the same shape, i. e., bind the same variables with the same irrelevance status. Only the types bound to the variables maybe different in Γ and Γ'.

Context equality $\vdash \Gamma = \ \vdash \Gamma'$ is a partial equivalence relation (PER), i. e., a symmetric and transitive relation, given inductively by the following rules:

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

Typing and equality are closed under weakening. Typing enjoys the usual inversion properties. To show substitution we introduce judgements $\Delta \vdash \sigma : \Gamma$ for substitution typing and $\Delta \vdash \sigma : \Gamma = \Delta' \vdash \sigma : \Gamma'$ for substitution equality which are given inductively by the following rules:

$$\frac{\vdash \Delta}{\Delta \vdash \sigma : \diamond} \qquad \frac{\Delta \vdash \sigma : \Gamma \qquad \Gamma \vdash U \qquad \Delta \vdash \sigma(x) \star U\sigma}{\Delta \vdash \sigma : \Gamma.\, x{\star}U}$$

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

Lemma 1 (Substitution). *Substitution equality is a PER. Further:*

1. *If $\Delta \vdash \sigma : \Gamma$ and $\Gamma \vdash t : T$ then $\Delta \vdash t\sigma : T\sigma$.*
2. *If $\Delta \vdash \sigma : \Gamma = \Delta' \vdash \sigma' : \Gamma'$. and $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$ then $\Delta \vdash t\sigma : T\sigma = \Delta' \vdash t'\sigma' : T'\sigma'$.*

Example 1 (Algebraic structures). [1] In type theory, we can model an algebraic structure over a carrier set A by a record of operations and proofs that the operations have the relevant properties. Consider an extension of IITT by tuples and Leibniz equality:

$(x{\star}A) \times B$	$: \mathsf{Set}_{\max(i,j)}$	for $A : \mathsf{Set}_i$ and $x \star A \vdash B : \mathsf{Set}_j$
(a, b)	$: (x{\star}A) \times B$	for $a : A$ and $b : B[a/x]$
$\mathsf{let}\ (x, y) = p\ \mathsf{in}\ t : C$		for $p : (x{\star}A) \times B$ and $x \star A, y{:}B \vdash t : C$
$a \equiv b$	$: \mathsf{Set}_i$	for $A : \mathsf{Set}_i$ and $a, b : A$
refl	$: a \equiv a$	for $A : \mathsf{Set}_i$ and $a : A$
$\mathsf{sym}\ p$	$: b \equiv a$	for $p : a \equiv b$

In the presence of a unit type $1 : \mathsf{Set}_i$ with constructor $() : 1$, the class $\mathsf{SemiGrp}$ of semigroups over a fixed $A : \mathsf{Set}_0$ can be defined as

$$\begin{aligned}
&\mathsf{Assoc} \quad : (A \to A \to A) \to \mathsf{Set}_0 \\
&\mathsf{Assoc}\, m = (a, b, c : A) \to m\ (m\ a\ b)\ c \equiv m\ a\ (m\ b\ c) \\
&\\
&\mathsf{SemiGrp}\ :\ \mathsf{Set}_0 \\
&\mathsf{SemiGrp} = (m : A \to A \to A) \times (assoc \div \mathsf{Assoc}\ m) \times 1.
\end{aligned}$$

[1] Inspired by the 2010-09-23 message of Andrea Vezzosi on the Agda mailing list.

We have marked the component $assoc$ as irrelevant which means that two SemiGrp structures over A are already equal when they share the operation m; the shape of the associativity proofs might differ. For instance, consider the flip operator (in a slightly sugared definition):

$$\begin{aligned}
&\mathsf{flip} && : \mathsf{SemiGrp} \to \mathsf{SemiGrp}\\
&\mathsf{flip}\ (m, (assoc, u)) && = (\lambda a : A. \lambda b : A.\ m\, b\, a, (\mathsf{sym}\ assoc, ()))\\
&\mathsf{thm} && : (s : \mathsf{SemiGrp}) \to \mathsf{flip}\ (\mathsf{flip}\ s) \equiv s\\
&\mathsf{thm}\ s && = \mathsf{refl}
\end{aligned}$$

A proof thm that flip cancels itself is now trivial, since $\lambda a\, b.\, (\lambda a\, b.\, m\, b\, a)\, b\, a = m$ by $\beta\eta$-equality and the $assoc$-component is irrelevant. This saves us from constructing a proof of $\mathsf{sym}\ (\mathsf{sym}\ assoc) \equiv assoc$ and the type checker from validating it. While the saving is small for this small example, it illustrates the principle.

Example 2 (Large Eliminations). [2] The ICC* [8] or EPTS [19] irrelevant function type $(x \div A) \to B$ allows x to appear *relevantly* in B. This extra power raises some issues with large eliminations. Consider

$$\begin{aligned}
&\mathsf{T} && : \mathsf{Bool} \to \mathsf{Set}_0\\
&\mathsf{T}\ \mathsf{true} && = \mathsf{Bool} \to \mathsf{Bool}\\
&\mathsf{T}\ \mathsf{false} && = \mathsf{Bool}\\
&t && = \lambda F : (b \div \mathsf{Bool}) \to (\mathsf{T}\, b \to \mathsf{T}\, b) \to \mathsf{Set}_0.\\
& && \quad \lambda g : F\ \mathsf{false}\ (\lambda x : \mathsf{Bool}.\ x) \to \mathsf{Bool}.\\
& && \quad \lambda a : F\ \mathsf{true}\ (\lambda x : \mathsf{Bool} \to \mathsf{Bool}. \lambda y : \mathsf{Bool}.\ x\, y).\ g\, a.
\end{aligned}$$

The term t is well-typed in ICC$^* + \mathsf{T}$ because the domain type of g and the type of a are $\beta\eta$-equal after erasure $(-)^*$ of type annotations and irrelevant arguments:

$$\begin{aligned}
&(F\ \mathsf{false}\ (\lambda x : \mathsf{Bool}.\ x))^* = F\ (\lambda x x)\\
&\quad =_{\beta\eta} F\ (\lambda x \lambda y.\ x\, y) = (F\ \mathsf{true}\ (\lambda x : \mathsf{Bool} \to \mathsf{Bool}. \lambda y : \mathsf{Bool}.\ x\, y))^*
\end{aligned}$$

While a Curry view supports this, it is questionable whether identity functions at different types should be viewed as one. It is unclear how a type-directed equality algorithm (see Sec. 3) should proceed here; it needs to recognize that $x : \mathsf{Bool}$ is equal to $\lambda y : \mathsf{Bool}.\ x\, y : \mathsf{Bool} \to \mathsf{Bool}$. This situation is amplified by a unit type 1 with extensional equality. When we change $\mathsf{T}\ \mathsf{true}$ to 1 and the type of a to $F\ \mathsf{true}\ (\lambda x : 1.\ ())$ then t should still type-check, because $\lambda x.\ ()$ is the identity function on 1. However, η-equality for 1 cannot be checked without types, and a type-directed algorithm would end up checking $x : \mathsf{Bool}$ for equality with $() : 1$. This can never work, because by transitivity we would get that any two booleans are equal.

Summarizing, we may conclude that the type of F bears trouble and needs to be rejected. IITT does this because it forbids the irrelevant b in relevant positions such as $\mathsf{T}\, b$; ICC* lacks T altogether. Extensions of ICC* should at least make sure that b is never eliminated, such as in $\mathsf{T}\, b$. Technically, T would have to be put in a separate class of *recursive* functions, those that actually compute with their argument. We leave the interaction of the three different function types to future research.

[2] Inspired by discussions with Ulf Norell during the 11th Agda Implementor's Meeting.

3 Algorithmic Equality

The algorithm for checking equality in IITT is inspired by Harper and Pfenning [12]. Like theirs, it is type-directed, but in our case each term comes with its own type in its own typing context. The algorithm proceeds stepwise, by alternating weak head normalization and head symbol comparison. Weak head normal forms (whnfs) are given by the following grammar:

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

Weak head evaluation. $t \searrow a$ and active application $f \mathbin{@^\star} u \searrow a$ are given by the following rules.

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

Instead of writing the propositions $t \searrow a$ and $P[a]$ we will sometimes simply write $P[{\downarrow}t]$. Similarly, we might write $P[f \mathbin{@^\star} u]$ instead of $f \mathbin{@^\star} u \searrow a$ and $P[a]$. In rules, it is understood that the evaluation judgement is always an extra premise, never an extra conclusion.

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

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

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

Structural equality $\Delta \vdash n : A \longleftrightarrow \Delta' \vdash n' : A'$ and $\Delta \vdash n : T \mathrel{\widehat{\longleftrightarrow}} \Delta' \vdash n' : T'$ checks the neutral expressions n and n' for equality and at the same time infers their types, which are returned as output.

$$\frac{\Delta \vdash n : T \mathrel{\widehat{\longleftrightarrow}} \Delta' \vdash n' : T'}{\Delta \vdash n : {\downarrow}T \longleftrightarrow \Delta' \vdash n' : {\downarrow}T'} \qquad \frac{(x{:}T) \in \Delta \qquad (x{:}T') \in \Delta'}{\Delta \vdash x : T \mathrel{\widehat{\longleftrightarrow}} \Delta' \vdash x : T'}$$

$$\frac{\begin{array}{c}\Delta \vdash n : (x{:}U) \to T \longleftrightarrow \Delta' \vdash n' : (x{:}U') \to T' \\ \Delta \vdash u : U \mathrel{\widehat{\iff}} \Delta' \vdash u' : U'\end{array}}{\Delta \vdash n\, u : T[u/x] \mathrel{\widehat{\longleftrightarrow}} \Delta' \vdash n'\, u' : T'[u'/x]}$$

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

Note that the inferred types $T[u/x]$ and $T'[u'/x]$ in the last rule are a priori different, even if T is equal to T'. This motivates a heterogeneously-typed algorithmic equality.

Type-directed equality $\Delta \vdash t : A \iff \Delta' \vdash t' : A'$ and $\Delta \vdash t : T \mathrel{\widehat{\iff}} \Delta' \vdash t' : T'$ checks terms t and t' for equality and proceeds by the common structure of the supplied types, to account for η.

$$\frac{\Delta \vdash T \mathrel{\widehat{\iff}} \Delta' \vdash T'}{\Delta \vdash T : s \iff \Delta' \vdash T' : s'}$$

$$\frac{\Delta.\, x{\star}U \vdash t \mathbin{\star} x : T \mathrel{\widehat{\iff}} \Delta'.\, x{\star}U' \vdash t' \mathbin{\star} x : T'}{\Delta \vdash t : (x{\star}U) \to T \iff \Delta' \vdash t' : (x{\star}U') \to T'}$$

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

Note that in the but-last rule we do not check that the inferred type T of $\downarrow t$ equals 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 a conceptually important improvement over Harper and Pfenning [12].

Lemma 2 (Algorithmic equality is a Kripke PER). $\longleftrightarrow$, $\mathrel{\widehat{\longleftrightarrow}}$, $\iff$, *and* $\mathrel{\widehat{\iff}}$ *are symmetric and transitive and closed under weakening.*

Extending structural equality to irrelevance, we let

$$\frac{\Delta^{\oplus} \vdash n : A \longleftrightarrow \Delta^{\oplus} \vdash n : A \qquad \Delta'^{\oplus} \vdash n' : A' \longleftrightarrow \Delta'^{\oplus} \vdash n' : A'}{\Delta \vdash n \div A \longleftrightarrow \Delta' \vdash n' \div A'}$$

and analogously for $\Delta \vdash n \div T \mathrel{\widehat{\longleftrightarrow}} \Delta' \vdash n' \div T'$.

4 A Universal Kripke Model for IITT

In this section we build, based on algorithmic equality, a universal Kripke model of typed terms that is both sound and complete for IITT. Following Goguen [11] and previous work [3], we first define a semantic universe hierarchy $\mathcal{T}_i$ whose sole purpose is to provide a measure for defining a logical relation and proving some of its properties. The limit $\mathcal{T}_\omega$ corresponds to the proof-theoretic strength or ordinal of IITT.

4.1 An Induction Measure

We denote sets of expressions by $\mathcal{A}, \mathcal{B}$ and functions from expressions to sets of expressions by $\mathcal{F}$. Let $\widehat{\mathcal{A}} = \{t \mid \downarrow t \in \mathcal{A}\}$ denote the closure of $\mathcal{A}$ by weak head expansion. Dependent function space is defined as $\Pi\, \mathcal{A}\, \mathcal{F} = \{f \in \mathsf{Whnf} \mid \forall u \in \widehat{\mathcal{A}}.\, f \mathbin{@^\star} u \in \mathcal{F}(u)\}$.

By recursion on $i \in \mathbb{N}$ we define inductively sets $\mathcal{T}_i \subseteq \mathsf{Whnf} \times \mathcal{P}(\mathsf{Whnf})$ as follows [3, Sec. 5.1]:

$$\frac{}{(N, \mathsf{Wne}) \in \mathcal{T}_i} \qquad \frac{}{(\mathsf{Set}_j, |\mathcal{T}_j|) \in \mathcal{T}_i}\ (\mathsf{Set}_j, \mathsf{Set}_i) \in \mathsf{Axiom}$$

$$\frac{(U, \mathcal{A}) \in \widehat{\mathcal{T}_i} \qquad \forall u \in \widehat{\mathcal{A}}.\, (T[u/x], \mathcal{F}(u)) \in \widehat{\mathcal{T}_i}}{((x{\star}U) \to T, \Pi\, \mathcal{A}\, \mathcal{F}) \in \mathcal{T}_i}$$

Herein, $\widehat{\mathcal{T}_i} = \{(T, \mathcal{A}) \mid (\downarrow T, \mathcal{A}) \in \mathcal{T}_i\}$ and $|\mathcal{T}_j| = \{A \mid (A, \mathcal{A}) \in \mathcal{T}_j \text{ for some } \mathcal{A}\}$. The induction measure $A \in \mathsf{Set}_i$ shall now mean the minimum height of a derivation of $(A, \mathcal{A}) \in \mathcal{T}_i$ for some $\mathcal{A}$. Note that due to universe stratification, $A \in \mathsf{Set}_i$ is smaller than $\mathsf{Set}_i \in \mathsf{Set}_j$.

4.2 A Heterogeneously Typed Kripke Logical Relation

By induction on the maximum of the measures $A \in \mathsf{Set}_i$ and $A' \in \mathsf{Set}_{i'}$ we define two Kripke relations

$$\begin{array}{c} \Delta \vdash A : \mathsf{Set}_i \mathbin{\textcircled{\scriptsize S}} \Delta' \vdash A' : \mathsf{Set}_{i'} \\ \Delta \vdash a : A \mathbin{\textcircled{\scriptsize S}} \Delta' \vdash a' : A'. \end{array}$$

together with their respective closures $\widehat{\textcircled{\scriptsize S}}$ and the generalization to $\star$. The clauses are given in rule form.

$$\frac{\Delta \vdash N \iff \Delta' \vdash N' \qquad \Delta \vdash N : \mathsf{Set}_i = \Delta' \vdash N' : \mathsf{Set}_{i'}}{\Delta \vdash N : \mathsf{Set}_i \mathbin{\textcircled{\scriptsize S}} \Delta' \vdash N' : \mathsf{Set}_{i'}} \qquad \frac{\Delta \vdash n : \_ \mathrel{\widehat{\longleftrightarrow}} \Delta' \vdash n' : \_ \qquad \Delta \vdash n : N = \Delta' \vdash n' : N'}{\Delta \vdash n : N \mathbin{\textcircled{\scriptsize S}} \Delta' \vdash n' : N'}$$

$$\frac{\Delta \vdash \mathsf{Set}_i : \mathsf{Set}_{i+1} = \Delta' \vdash \mathsf{Set}_i : \mathsf{Set}_{i+1}}{\Delta \vdash \mathsf{Set}_i : \mathsf{Set}_{i+1} \mathbin{\textcircled{\scriptsize S}} \Delta' \vdash \mathsf{Set}_i : \mathsf{Set}_{i+1}}$$

$$\frac{\begin{array}{l} \Delta \vdash U : \mathsf{Set}_i \mathbin{\widehat{\textcircled{\scriptsize S}}} \Delta' \vdash U' : \mathsf{Set}_{i'} \\ \forall (\Gamma, \Gamma') \leq (\Delta, \Delta'),\ \Gamma \vdash u \star U \mathbin{\widehat{\textcircled{\scriptsize S}}} \Gamma' \vdash u' \star U' \implies \\ \qquad \Gamma \vdash T[u/x] : \mathsf{Set}_i \mathbin{\widehat{\textcircled{\scriptsize S}}} \Gamma' \vdash T'[u'/x] : \mathsf{Set}_{i'} \\ \Delta \vdash (x \star U) \to T : \mathsf{Set}_i = \Delta' \vdash (x \star U') \to T' : \mathsf{Set}_{i'} \end{array}}{\Delta \vdash (x \star U) \to T : \mathsf{Set}_i \mathbin{\textcircled{\scriptsize S}} \Delta' \vdash (x \star U') \to T' : \mathsf{Set}_{i'}}$$

$$\frac{\begin{array}{l} \forall (\Gamma, \Gamma') \leq (\Delta, \Delta'),\ \Gamma \vdash u \star U \mathbin{\widehat{\textcircled{\scriptsize S}}} \Gamma' \vdash u' \star U' \implies \\ \qquad \Gamma \vdash f\, ^\star u : T[u/x] \mathbin{\widehat{\textcircled{\scriptsize S}}} \Gamma' \vdash f'\, ^\star u' : T'[u'/x] \\ \Delta \vdash f : (x \star U) \to T = \Delta' \vdash f' : (x \star U') \to T' \end{array}}{\Delta \vdash f : (x \star U) \to T \mathbin{\textcircled{\scriptsize S}} \Delta' \vdash f' : (x \star U') \to T'}$$

$$\frac{t \searrow a \qquad \Delta \vdash t = a : \downarrow T \qquad \Delta' \vdash t' = a' : \downarrow T' \qquad t' \searrow a' \qquad \Delta \vdash a : \downarrow T \mathbin{\textcircled{\scriptsize S}} \Delta' \vdash a' : \downarrow T'}{\Delta \vdash t : T \mathbin{\widehat{\textcircled{\scriptsize S}}} \Delta' \vdash t' : T'}$$

$$\frac{\Delta^{\oplus} \vdash a : A \mathbin{\textcircled{\scriptsize S}} \Delta^{\oplus} \vdash a : A \qquad \Delta'^{\oplus} \vdash a' : A' \mathbin{\textcircled{\scriptsize S}} \Delta'^{\oplus} \vdash a' : A'}{\Delta \vdash a \div A \mathbin{\textcircled{\scriptsize S}} \Delta' \vdash a' \div A'}$$

$$\frac{\Delta^{\oplus} \vdash t : T \mathbin{\widehat{\textcircled{\scriptsize S}}} \Delta^{\oplus} \vdash t : T \qquad \Delta'^{\oplus} \vdash t' : T' \mathbin{\widehat{\textcircled{\scriptsize S}}} \Delta'^{\oplus} \vdash t' : T'}{\Delta \vdash t \div T \mathbin{\widehat{\textcircled{\scriptsize S}}} \Delta' \vdash t' \div T'}$$

It is immediate that the logical relation contains only well-typed terms, is symmetric, transitive, and closed under weakening.

Lemma 3 (Type and context conversion). *If* $\Delta \vdash t : T \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash t' : T'$ *and* $\Delta' \vdash T' : s' \mathbin{\widehat{\textcircled{S}}} \Delta'' \vdash T'' : s''$ *then* $\Delta \vdash t : T \mathbin{\widehat{\textcircled{S}}} \Delta'' \vdash t' : T''$.

Lemma 4 (Escape from the logical relation). *Let* $\Delta \vdash T : \mathsf{Set}_i \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash T' : \mathsf{Set}_{i'}$

1. $\Delta \vdash T \mathrel{\widehat{\Longleftrightarrow}} \Delta \vdash T'$.
2. *If* $\Delta \vdash t : T \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash t' : T'$ *then* $\Delta \vdash t : T \mathrel{\widehat{\Longleftrightarrow}} \Delta' \vdash t' : T'$.
3. *If* $\Delta \vdash n \star T \mathrel{\widehat{\longleftrightarrow}} \Delta' \vdash n' \star T'$ *and* $\Delta \vdash n \star T = \Delta \vdash n' \star T'$ *then* $\Delta \vdash n \star T \mathbin{\widehat{\textcircled{S}}} \Delta \vdash n' \star T'$.

4.3 Validity in the Model

Simultaneously and by induction on the length of Γ we define the PERs $\Vdash \Gamma = \Vdash \Gamma'$ and $\Delta \vdash \sigma : \Gamma \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash \sigma' : \Gamma'$ which presupposes the former. In rule notation this reads:

$$\frac{}{\Vdash \diamond = \Vdash \diamond} \qquad \frac{\Vdash \Gamma = \Vdash \Gamma' \qquad \Gamma \Vdash U = \Gamma' \Vdash U'}{\Vdash \Gamma.\, x{\star}U = \Vdash \Gamma'.\, x{\star}U'}$$

$$\frac{}{\Delta \vdash \sigma : \diamond \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash \sigma' : \diamond}$$

$$\frac{\Delta \vdash \sigma : \Gamma \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash \sigma' : \Gamma' \qquad \Delta \vdash \sigma(x) \star U\sigma \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash \sigma'(x) \star U'\sigma'}{\Delta \vdash \sigma : \Gamma.\, x{\star}U \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash \sigma' : \Gamma'.\, x{\star}U'}$$

Again at the same time, we define the following abbreviations, also given in rule notation:

$$\frac{}{\Gamma \Vdash s = \Gamma' \Vdash s} \qquad \frac{\Gamma \Vdash T : s = \Gamma' \Vdash T' : s'}{\Gamma \Vdash T = \Gamma' \Vdash T'}$$

$$\frac{\Vdash \Gamma = \Vdash \Gamma' \qquad \Gamma \Vdash T = \Gamma' \Vdash T' \qquad \forall \Delta \vdash \sigma : \Gamma \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash \sigma' : \Gamma' \implies \Delta \vdash t\sigma : T\sigma \mathbin{\widehat{\textcircled{S}}} \Delta' \vdash t'\sigma' : T'\sigma'}{\Gamma \Vdash t : T = \Gamma' \Vdash t' : T'}$$

Finally, let $\Gamma \Vdash t : T \iff \Gamma \Vdash t : T = \Gamma \Vdash t : T$ and $\Vdash \Gamma \iff \Vdash \Gamma = \Vdash \Gamma$.

Lemma 5 (Context satisfiable). *For the identity substitution* id *and* $\Vdash \Gamma = \Vdash \Gamma'$ *we have* $\Gamma \vdash \mathsf{id} : \Gamma \mathbin{\widehat{\textcircled{S}}} \Gamma' \vdash \mathsf{id} : \Gamma'$.

Theorem 1 (Completeness of IITT rules). *If* $\Gamma \Vdash t : T = \Gamma' \Vdash t' : T'$ *then* $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$ *and* $\Gamma \vdash T = \Gamma' \vdash T'$.

Theorem 2 (Fundamental theorem of logical relations).

1. *If* $\vdash \Gamma$ *then* $\Vdash \Gamma$.
2. *If* $\vdash \Gamma = \vdash \Gamma'$ *then* $\Vdash \Gamma = \Vdash \Gamma'$.
3. *If* $\Gamma \vdash t : T$ *then* $\Gamma \Vdash t : T$.
4. *If* $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$ *then* $\Gamma \Vdash t : T = \Gamma' \Vdash t' : T'$.

5 Meta-theoretic Consequences of the Model Construction

After doing hard work in the construction of a universal model, the rest of the meta-theory of IITT falls into our lap like a ripe fruit.

Normalization and subject reduction. An immediate consequence of the model construction is that each term has a weak head normal form and that typing and equality is preserved by weak head normalization.

Theorem 3 (Normalization and subject reduction). *If* $\Gamma \vdash t : T$ *then* $t \searrow a$ *and* $\Gamma \vdash t = a : T$.

Correctness of algorithmic equality. Algorithmic equality is correct, i. e., sound, complete, and terminating. Together, this entails decidability of equality in IITT. Algorithmic equality is built into the model at every step, thus, completeness is immediate:

Theorem 4 (Completeness of algorithmic equality). *If* $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$ *then* $\Gamma \vdash t : T \mathrel{\widehat{\Longleftrightarrow}} \Gamma' \vdash t' : T'$.

Termination of algorithmic equality is a consequence of full normalization, which we have not defined explicitly, but which is implicit in the model.

Theorem 5 (Termination of algorithmic equality). *If* $\Delta \vdash t : T$ *and* $\Delta' \vdash t' : T'$ *then the query* $\Delta \vdash t : T \mathrel{\widehat{\Longleftrightarrow}} \Delta' \vdash t' : T'$ *terminates.*

Soundness of the equality algorithm is a consequence of subject reduction.

Theorem 6 (Soundness of algorithmic equality). *Let* $\Delta \vdash t : T$ *and* $\Delta' \vdash t' : T'$ *and* $\Delta \vdash T = \Delta' \vdash T'$. *If* $\Delta \vdash t : T \mathrel{\widehat{\Longleftrightarrow}} \Delta' \vdash t' : T'$ *then* $\Delta \vdash t : T = \Delta' \vdash t' : T'$.

Homogeneity. Although we defined IITT-equality heterogeneously, we can now show that the heterogeneity was superficial, i. e., in fact do equal terms have equal types. This was already implicit in the formulation of the equality algorithm which only compares terms at types of the same shape. By rather than building homogeneity into the definition of equality, we obtain it as a global result.

Theorem 7 (Homogeneity). *If* $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$ *then* $\vdash \Gamma = \ \vdash \Gamma'$ *and* $\Gamma \vdash T = \Gamma' \vdash T'$.

Consistency. Importantly, not every type is inhabited in IITT, thus, it can be used as a logic. A prerequisite is that types can be distinguished, which follows immediately from completeness of algorithmic equality.

Theorem 8 (Consistency). $X : \mathsf{Set}_0 \not\vdash t : X$.

Decidability. To round off, we show that typing in IITT is decidable. Type checking algorithms such as bidirectional checking [10] rely on injectivity of function type constructors, which is built into the definition of Ⓢ:

Theorem 9 (Function type injectivity). *If* $\Gamma \vdash (x \star U) \to T : s = \Gamma' \vdash (x \star U') \to T' : s'$ *then* $\Gamma \vdash U : s = \Gamma' \vdash U' : s'$ *and* $\Gamma . x \star U \vdash T : s = \Gamma' . x \star U' \vdash T' : s'$.

Theorem 10 (Decidability of IITT). *Equality* $\Gamma \vdash t : T = \Gamma' \vdash t' : T'$ *and typing* $\Gamma \vdash t : T$ *are decidable.*

6 Extensions

Data types and recursion. The semantics of IITT is ready to cope with inductive data types like the natural numbers and the associated recursion principles. Recursion into types, aka known as large elimination, is also accounted for since we have universes and a semantics which does not erase dependencies (unlike Pfenning's model [22]).

Types with extensionality principles. The purpose of having a typed equality algorithm is to handle η-laws that are not connected to the shape of the expression (like η-contraction for functions) but to the shape of the type only. Typically these are types T with at most one inhabitant, i. e., the empty type, the unit type, singleton types or propositions.[3] For such T we have the η-law

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

which can only be checked in the presence of type T. Realizing such η-laws gives additional "proof" irrelevance which is not covered by Pfenning's irrelevant quantification $(x \dot{:} U) \to T$.

Internal erasure. Terms $u \div U$ in irrelevant position are only there to please the type checker, they are ignored during equality checking. This can be inferred from the substitution principle: If $\Gamma.\, x \dot{:} U \vdash T$ and $\Gamma \vdash u, u' \div U$, then $\Gamma \vdash T[u/x] = T[u'/x]$; the type T has the same shape regardless of u, u'. Hence, terms like u serve the sole purpose to prove some proposition and could be replaced by a dummy $\bullet$ immediately after type-checking. This is an optimization which in the first place saves memory, but if expressions are written to interface files and reloaded later, it also saves disk space and execution time of saving and loaded. First experiments with an implementation of internal erasure in Agda [9] shows that savings are huge, like in formalizing category theory and algebra which uses structures with embedded proofs (see Example 1).

Internal erasure can be realized by making $\Gamma \vdash t \div T$ a judgement (as opposed to just a notation for $\Gamma^{\oplus} \vdash t : T$) and adding the rule

$$\frac{\Gamma \vdash t \div T}{\Gamma \vdash \bullet \div T}.$$

The rule states that if there is already a proof t of T, then $\bullet$ is a new proof of T. This preserves provability while erasing the proof terms. Conservativity of this rule can be proven as in joint work of the author with Coquand and Pagano [4].

Proof modality. Pfenning [22] suggests a modality $\triangle$ formed by the rule

$$\frac{\Gamma \vdash t \div T}{\Gamma \vdash t : \triangle T}.$$

which for instance allows the definition of the subset type $\{x : U \mid T\, x\}$ as $\Sigma x : U.\, \triangle(T\, x)$. Such a modality has been present in Nuprl as *Squash* type [20] and it is also known as the type of proofs of (proposition) T [4, 8]. Using the extensions of Example 1, we can encode it as $\triangle T = (\_ \dot{:} T) \times 1$.

[3] Some care is necessary for the type of Leibniz equality [1, 25].

7 Conclusions

We have extended Pfenning's notion of irrelevance to a type theory IITT with universes that accommodates types defined by recursion. A heterogeneous algorithmic equality judgement has allowed a smooth construction of a universal Kripke model that is both sound and complete for IITT, yielding normalization, consistency and decidability of IITT. Inspired by a heterogeneously typed semantics, we have formulated the declarative system with a heterogeneous equality judgement as well and demonstrated that this also yields a sound specification of type theory.

Integrating irrelevance and data types in dependent type theory does not seem without challenges. We have succeeded to treat Pfenning's notion of irrelevance, but our proof does not scale directly to *parametric* function types, a stronger notion of irrelevant function types called implicit quantification by Miquel [18].[4] Two more type theories build on Miquel's calculus [17], Barras and Bernardo's ICC* [8] and Mishra-Linger and Sheard's *Erasure Pure Type Systems* (EPTS) [19], but none has offered a satisfying account of large eliminations yet. Miquel's model [16] features data types only as impredicative encodings. For irrelevant, parametric, and recursive functions to coexist it seems like three different function types are necessary, e. g., in the style of Pfenning's *irrelevance, extensionality and intensionality*. We would like to solve this puzzle in future work, not least to implement high-performance languages with dependent types.

Acknowledgments. Thanks to Bruno Barras, Bruno Bernardo, Thierry Coquand, Dan Doel, Hugo Herbelin, Conor McBride, and Ulf Norell for discussions on irrelevance in type theory. Thanks to Jason Reed, Conor McBride, and the anonymous referees for suggesting improvements on the text. Work on a previous paper has been carried out while being invited researcher at PPS, Paris, in the INRIA πr^2 team headed by Pierre-Louis Curien and Hugo Herbelin. I have benefited from presentations at JFLA 2010, the Agda Implementer's Meeting in March 2010 and the DTP workshop in 2010.

References

1. Andreas Abel. Extensional normalization in the logical framework with proof irrelevant equality. In Olivier Danvy, editor, *2009 Wksh. on Normalization by Evaluation*, 2009.
2. Andreas Abel. MiniAgda: Integrating sized and dependent types. In Ana Bove, Ekaterina Komendantskaya, and Milad Niqui, editors, *Wksh. on Partiality And Recursion in Interactive Theorem Provers (PAR 2010)*, 2010.
3. Andreas Abel, Thierry Coquand, and Peter Dybjer. Verifying a semantic $\beta\eta$-conversion test for Martin-Löf type theory. In Philippe Audebaud and Christine Paulin-Mohring, editors, *Proc. of the 9th Int. Conf. on Mathematics of Program Construction, MPC 2008*, volume 5133 of *Lect. Notes in Comput. Sci.*, pages 29–56. Springer, 2008.
4. Andreas Abel, Thierry Coquand, and Miguel Pagano. A modular type-checking algorithm for type theory with singleton types and proof irrelevance. In P.-L. Curien, editor, *Proc. of the 9th Int. Conf. on Typed Lambda Calculi and Applications, TLCA 2009*, volume 5608 of *Lect. Notes in Comput. Sci.*, pages 5–19. Springer, 2009.

[4] A function argument is parametric if it is irrelevant for computing the function result while the type of the result may depend on it. In Pfenning's notion, the argument must also be irrelevant in the type.

5. Roberto M. Amadio, editor. *Proc. of the 11th Int. Conf. on Foundations of Software Science and Computational Structures, FOSSACS 2008*, volume 4962 of *Lect. Notes in Comput. Sci.* Springer, 2008.
6. Lennart Augustsson. Cayenne - a language with dependent types. In *Proc. of the 3rd ACM SIGPLAN Int. Conf. on Functional Programming (ICFP '98)*, volume 34 of *SIGPLAN Notices*, pages 239–250. ACM Press, 1999.
7. Steven Awodey and Andrej Bauer. Propositions as [Types]. *J. Log. Comput.*, 14(4):447–471, 2004.
8. Bruno Barras and Bruno Bernardo. The implicit calculus of constructions as a programming language with dependent types. In Amadio [5], pages 365–379.
9. Ana Bove, Peter Dybjer, and Ulf Norell. A brief overview of Agda - a functional language with dependent types. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *Theorem Proving in Higher Order Logics, TPHOLs 2009*, volume 5674 of *Lect. Notes in Comput. Sci.*, pages 73–78. Springer, 2009.
10. Thierry Coquand. An algorithm for type-checking dependent types. In *Proc. of the 3rd Int. Conf. on Mathematics of Program Construction, MPC '95*, volume 26 of *Sci. Comput. Program.*, pages 167–177. Elsevier, May 1996.
11. Healfdene Goguen. *A Typed Operational Semantics for Type Theory*. PhD thesis, University of Edinburgh, August 1994. Available as LFCS Report ECS-LFCS-94-304.
12. 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.
13. INRIA. *The Coq Proof Assistant Reference Manual*. INRIA, version 8.2 edition, 2008. http://coq.inria.fr/.
14. Pierre Letouzey. A new extraction for Coq. In Herman Geuvers and Freek Wiedijk, editors, *TYPES*, volume 2646 of *Lect. Notes in Comput. Sci.*, pages 200–219. Springer, 2002.
15. Conor McBride and James McKinna. The view from the left. *J. Func. Program.*, 2004.
16. Alexandre Miquel. A model for impredicative type systems, universes, intersection types and subtyping. In *Proc. of the 15th IEEE Symp. on Logic in Computer Science (LICS 2000)*, pages 18–29, 2000.
17. Alexandre Miquel. The implicit calculus of constructions. In Samson Abramsky, editor, *Proc. of the 5th Int. Conf. on Typed Lambda Calculi and Applications, TLCA 2001*, volume 2044 of *Lect. Notes in Comput. Sci.*, pages 344–359. Springer, 2001.
18. Alexandre Miquel. *Le Calcul des Constructions implicite: syntaxe et sémantique.* PhD thesis, Université Paris 7, December 2001.
19. Nathan Mishra-Linger and Tim Sheard. Erasure and polymorphism in pure type systems. In Amadio [5], pages 350–364.
20. Richard Nathan Mishra-Linger. *Irrelevance, Polymorphism, and Erasure in Type Theory*. PhD thesis, Portland State University, 2008.
21. Christine Paulin-Mohring and Benjamin Werner. Synthesis of ML programs in the system Coq. *J. Symb. Comput.*, 15(5/6):607–640, 1993.
22. Frank Pfenning. Intensionality, extensionality, and proof irrelevance in modal type theory. In *LICS 2001: IEEE Symposium on Logic in Computer Science*, June 2001.
23. Jason Reed. Proof irrelevance and strict definitions in a logical framework, 2002. Senior Thesis, published as Carnegie-Mellon University technical report CMU-CS-02-153.
24. Jason Reed. Extending higher-order unification to support proof irrelevance. In David A. Basin and Burkhart Wolff, editors, *Theorem Proving in Higher Order Logics, TPHOLs 2003*, volume 2758 of *Lect. Notes in Comput. Sci.*, pages 238–252. Springer, 2003.
25. Benjamin Werner. On the strength of proof-irrelevant type theories. *Logical Meth. in Comput. Sci.*, 4, 2008.