Internal Strict Propositions Using Point-Free Equations

István Donkó
Eötvös Loránd University, Budapest, Hungary

Ambrus Kaposi
Eötvös Loránd University, Budapest, Hungary

Abstract

The setoid model of Martin-Löf's type theory bootstraps extensional features of type theory from intensional type theory equipped with a universe of definitionally proof irrelevant (strict) propositions. Extensional features include a Prop-valued identity type with a strong transport rule and function extensionality. We show that a setoid model supporting these features can be defined in intensional type theory without any of these features. The key component is a point-free notion of propositions. Our construction suggests that strict algebraic structures can be defined along the same lines in intensional type theory.

2012 ACM Subject Classification Theory of computation → Type theory

Keywords and phrases Martin-Löf's type theory, intensional type theory, function extensionality, setoid model, homotopy type theory

Digital Object Identifier 10.4230/LIPIcs.TYPES.2021.6

Supplementary Material *Software (Formalisation)*: `https://bitbucket.org/akaposi/prop` [12]
archived at `swh:1:dir:6648713cc70e9c6fa8a71cccaa31c1d91cfbc418`

Funding *István Donkó*: Supported by the ÚNKP-21-3 New National Excellence Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund.
Ambrus Kaposi: Supported by the Bolyai Fellowship of the Hungarian Academy of Sciences, Project no. BO/00659/19/3, and by the "Application Domain Specific Highly Reliable IT Solutions" project which has been implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the Thematic Excellence Programme TKP2020-NKA-06 (National Challenges Subprogramme) funding scheme.

Acknowledgements Thanks to Christian Sattler for discussions on the topics of this paper. Many thanks to the anonymous reviewers for their comments and suggestions.

1 Introduction

The setoid model of type theory pioneered by Hofmann [15] supports the following extensional features that are missing from intensional type theory: function extensionality, propositional extensionality (univalence for propositions [4]) and quotient inductive-inductive types [18]. If the setoid model is defined in an intensional metatheory and all equations of the model (such as the β rule) hold definitionally, then it constitutes a *model construction* (also called syntactic translation): any model of intensional type theory can be turned into its "setoidified" variant which supports the extensional features, thus bootstrapping the extensional features from intensional type theory. Hofmann's original model only justified some of the equations definitionally. Altenkirch showed that if the metatheory supports a sort TyP of definitionally proof irrelevant propositions in addition to the sort Ty of types, then there is a version of the setoid model where all equations are definitional [2]. After he presented this model construction at the Symposium on Logic in Computer Science in 1999 [2], Per Martin-Löf asked whether it is possible to remove the extra requirement of TyP. As far as we know, the question is still open.

© István Donkó and Ambrus Kaposi;
licensed under Creative Commons License CC-BY 4.0

In this setoid model a closed type is a setoid: a type together with an equivalence relation; a term is a function between the types which respects the relations. If the equivalence relation is proof relevant (Ty-valued), then terms have to additionally include components about respecting the reflexivity, symmetry and transitivity proofs, then when proving equalities of terms (such as the β law), one has to show that the corresponding new components are equal, which forces the introduction of new components, and so on. This problem is usually referred to as coherence problem, see [15, Section 5.3] for a discussion in the context of the setoid model, or [19] for a recent exposition of the general phenomenon. Altenkirch's solution [2] was to make the relation TyP-valued instead of Ty-valued: in this case, terms automatically respect reflexivity proofs as there is only one proof of reflexivity, up to definitional equality. We could avoid requiring TyP by using the internally definable universe of homotopy propositions hProp [23]. If the relation is hProp-valued, terms respect reflexivity proofs up to the internal identity type. However to show that the relation for Π types is in hProp, we need that hProp is closed under Π. To prove this, we need function extensionality, which defeats the purpose of the model bootstrapping function extensionality.

In this paper we show that in intensional type theory there is an alternative notion of proposition that is closed under Π. A type A is an hProp if any two elements of A are equal. We can also express this equation in a point-free way: the two functions "first" and "second" both having type $A \to A \to A$ are equal. We call this property $\mathsf{isPfProp}$ for point-free propositions.

$$\mathsf{isHProp}\,A \equiv (a\,a' : A) \to \mathsf{Id}_A\,a\,a' \qquad\qquad \mathsf{isPfProp}\,A \equiv \mathsf{Id}_{(A\to A\to A)}\,(\lambda a\,a'.a)\,(\lambda a\,a'.a')$$

In the presence of function extensionality, $\mathsf{isHProp}\,A$ and $\mathsf{isPfProp}\,A$ are equivalent. However, in intensional type theory without function extensionality, the latter is stronger. $\mathsf{isPfProp}$ classifies definitionally proof irrelevant types in the empty context: from canonicity it follows that if $\mathsf{isPfProp}\,A$ for a closed type A, then all elements of A are definitionally equal. For a type family $A : D \to \mathsf{Type}$ over a closed type we use a dependent variant of $\mathsf{isPfProp}$:

$$\mathsf{isPfPropd}\,A \equiv \mathsf{Id}_{((d:D)\to A\,d\to A\,d\to A\,d)}\,(\lambda d\,a\,a'.a)\,(\lambda d\,a\,a'.a')$$

In intensional type theory, unlike hProp, $\mathsf{isPfPropd}$ is closed under Π types. This essentially relies on the η rule for Π types. Using η for Σ and $\top$, we can prove that $\mathsf{isPfPropd}$ is closed under these type formers too. $\mathsf{isPfProp}$ only includes $\bot$ if it has a weak η rule saying that any two elements of $\bot$ are definitionally equal. This is usually not the case in intensional type theory where $\bot$ is defined as an inductive type.

With the help of point-free propositions, we give a partial positive answer to Martin-Löf's question: in intensional type theory without a sort of propositions, we define the setoid model with $\bot$, $\top$, Π, Σ, types, a sort TyP closed under $\top$, Π, Σ and a TyP-valued identity type with function extensionality. Our answer is partial because $\bot$ is not in TyP, and the model does not support inductive types, or a universe of propositions. We also define an external version of this model as a model construction taking as input a model of intensional type theory, and outputting a model with extensionality principles. This latter construction only uses external point-free propositions which are the same as subobjects in category theory, but we still haven't encountered it in the literature.

Recently, there is a renewed interest in models of type theory with a sort TyP. Agda was extended with a universe of strict propositions [13], this was used to formalise fully featured variants of the setoid model [4, 3, 18], strict presheaf models were built using TyP [22], and the metatheory of type theories with TyP was studied [1, 11]. One difference between TyP and pfProp is that (as every sort) the former is static: it only includes types which are built into it. The latter is dynamic: any type is included for which all elements are definitionally equal. Another difference is that proof irrelevance holds definitionally for assumed elements of TyP, while we only know proof irrelevance up to propositional equality for members of pfProp.

More generally, in intensional type theory, point-free equations can be used to describe strict algebraic structures. One has to express the algebraic equations in a point-free way. For example, in a strict monoid with carrier M and binary operation $- \otimes -$, associativity is expressed as $\mathsf{Id}_{\mathsf{M}\to\mathsf{M}\to\mathsf{M}\to\mathsf{M}}\,\big(\lambda x\,y\,z.(x \otimes y) \otimes z\big)\,\big(\lambda x\,y\,z.x \otimes (y \otimes z)\big)$. Natural numbers with addition are not a strict monoid because addition is only weakly associative. An example of a strict monoid is the function space $A \to A$ for any type A with composition as the binary operation.

1.1 Structure of the paper

After describing related work, in Section 2 we explain our notation and the notion of model of type theory we use (category with families). In Section 3 we define point-free propositions and show that they are closed under $\top$, Σ and Π. In Section 4, we show that any model of type theory can be equipped with a sort of strict propositions. This can be seen as the external version of Section 3. We compare the internal and external notions of propositions in Section 5. Then we describe how point-free propositions can be applied to construct the setoid model. As a warmup, we define the model construction externally (Section 6). Then we turn to our main application of internal point-free propositions and define the setoid model internally to a model of intensional type theory (Section 7). In Section 8 we give more examples of strict algebraic structures. We conclude in Section 9.

Sections 3 and 7 were formalised in Agda [12], and can be understood without much intuition about categories with families.

1.2 Related work

Hofmann defined two versions of the setoid model in an intensional metatheory [15], one of them did not have dependent types, the other justified some computation rules (e.g. β rules for Σ types) only up to propositional equality, and not definitionally. Altenkirch [2] justified all the rules of type theory but relied on a definitionally proof-irrelevant universe of propositions. He sketched a normalisation proof for a type theory with such a universe. Coquand [9] defined a setoid model in intensional type theory which justifies a weak function space: there is no substitution rule for λ and no η rule. Palmgren [21] formalised a set-theoretic interpretation of extensional type theory in an intensional metatheory. He used setoids for encoding sets as well-founded trees quotiented by bisimulation, hence it can also be seen as a setoid model. Thus it is similar to our Construction 17 and it justifies more types including inductive types and a universe. It is not clear whether one can obtain a model construction analogous to Construction 15 from his interpretation.

Strict propositions were introduced in Agda and Coq in a way that is compatible with univalence [13]. Issues with rewriting-style normalisation for a type theory with strict propositions, a strict identity type and a strong transport were found by Abel and Coquand [1]. Normalisation for type theory with strict propositions but without such an identity type was proved by Coquand [11].

The setoid model as a model construction was described in [4] together with an Agda formalisation using strict propositions in Agda. This was extended with an inductive-recursive universe of setoids in [3].

In [4], a variant of the setoid model was described in which transport has a definitional computation rule. In the accompanying formalisation, a point-free equation was used to ensure this property: instead of $(a : A) \to \mathsf{coe}_A\,\mathsf{refl}\,a = a$, the equation $(\lambda a.\mathsf{coe}_A\,\mathsf{refl}\,a) = (\lambda a.a)$ was used. In his brilliant paper [16], Hugunin shows that function extensionality is not needed to define natural numbers (and inductive types) from W-types in intensional type theory. He constructs a predicate which selects the "canonical" elements in natural numbers defined by W-types. His construction has a similar "point-free" flavour and also essentially relies on η for function space.

2 Type theory

Our metatheory is extensional type theory and we use notations similar to Agda's. We write Type for the Russell universe (we don't write levels explicitly, but we work in a predicative setting), we write $\equiv$ for definitional equality, we write $(x : A) \to B$ for function space with $\lambda(x : A).t$ or $\lambda x.t$ for abstraction, juxtaposition for application, $(x : A) \times B$ for Σ types with a, b for pairing and $\pi_1\,\mathit{ab}$, $\pi_2\,\mathit{ab}$ for projections. We use the lower case Simonyi naming convention, e.g. ab is a name for a variable in $(x : A) \times B$. We use implicit arguments and implicit quantifications which we sometimes specify explicitly in subscripts. We write $\top$, tt for the unit type and its constructor. Function space, dependent products and unit have η laws. We write $\mathsf{Id}_A\,a\,a'$ or $a =_A a'$ or simply $a = a'$ for the identity type, it has constructor $\mathsf{refl} : \mathsf{Id}_A\,a\,a$ and eliminator $\mathsf{J} : \big(P : (a' : A) \to a = a' \to \mathsf{Type}\big) \to P\,a\,\mathsf{refl} \to (e : a = a') \to P\,a'\,e$ with definitional computation rules. We write $\mathsf{transp} : (P : A \to \mathsf{Type}) \to a = a' \to P\,a \to P\,a'$, $e \mathbin{\blacksquare} e' : a = a''$ for $e : a = a'$ and $e' : a' = a''$, $\mathsf{ap}\,f\,e : f\,a = f\,a'$ for $e : a = a'$, all defined via J. The empty type is denoted $\bot$ with eliminator $\mathsf{elim}_\bot$. We assume quotient inductive-inductive types (QIITs), that is, we have syntaxes for type theories (see paragraph after the next one).

In some places (e.g. in sections 3 and 7), we work internally to a model of intensional type theory, and use the same notations as for our metatheory. In these cases we specify precisely what features our model has and we only use those features, for example we don't use equality reflection. In such cases we use the phrase "external" to refer to the metatheory.

The notion of model of type theory we use is category with families (CwF, [8]). Using this presentation, type theory is a generalised algebraic theory and the syntax of a type theory is the initial algebra which is a QIIT. In extensional type theory, it is enough to assume the existence of a single QIIT to obtain syntaxes for all generalised algebraic theories [17]. We assume the existence of this QIIT (called the theory of QIIT signatures in [17]).

We give some intuition for the description of type theory as a CwF here. A gentler introduction is e.g. [5]. Figure 1 lists the components of a model of type theory with $\top$, Σ, Π, $\bot$ and Id types. A model of type theory consists of a category with families (CwF, left hand side of the figure), that is, a category of contexts and substitutions $(\mathsf{Con}, \dots, \mathsf{idr})$ with a terminal object (the empty context $\bullet$, the empty substitution ϵ, $\bullet\eta$), a presheaf of types $(\mathsf{Ty}, \dots, [\mathsf{id}])$ and a locally representable dependent presheaf of terms over types $(\mathsf{Tm}, \dots, \rhd\eta)$. Local representability is also called comprehension, and consists of the context extension operation $- \rhd -$ together with the natural isomorphism $\mathsf{Sub}\,\Delta\,(\Gamma \rhd A) \cong (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \times \mathsf{Tm}\,\Delta\,(A[\gamma])$ witnessed by $-, -, \dots, \rhd\eta$. Note that many operations have implicit arguments, for example $- \circ -$ takes Γ, Δ, Θ implicitly. Also, some equations only typecheck because of previous equations, for example, $[\mathsf{id}]$ for terms depends on $[\mathsf{id}]$ for types: the left hand side is in $\mathsf{Tm}\,\Gamma\,(A[\mathsf{id}])$, the right hand side is in $\mathsf{Tm}\,\Gamma\,A$. We don't write the transports because we work in extensional type theory.

Con	$: \mathsf{Set}$
Sub	$: \mathsf{Con} \to \mathsf{Con} \to \mathsf{Set}$
$- \circ -$	$: \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Sub}\,\Theta\,\Delta \to \mathsf{Sub}\,\Theta\,\Gamma$
ass	$: (\gamma \circ \delta) \circ \theta \equiv \gamma \circ (\delta \circ \theta)$
id	$: \mathsf{Sub}\,\Gamma\,\Gamma$
idl	$: \mathsf{id} \circ \gamma \equiv \gamma$
idr	$: \gamma \circ \mathsf{id} \equiv \gamma$
$\blacklozenge$	$: \mathsf{Con}$
ϵ	$: \mathsf{Sub}\,\Gamma\,\blacklozenge$
$\blacklozenge\eta$	$: (\sigma : \mathsf{Sub}\,\Gamma\,\blacklozenge) \to \sigma \equiv \epsilon$
Ty	$: \mathsf{Con} \to \mathsf{Set}$
$-[-]$	$: \mathsf{Ty}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{Ty}\,\Delta$
$[\circ]$	$: A[\gamma \circ \delta] \equiv A[\gamma][\delta]$
$[\mathsf{id}]$	$: A[\mathsf{id}] \equiv A$
Tm	$: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Set}$
$-[-]$	$: \mathsf{Tm}\,\Gamma\,A \to (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma])$
$[\circ]$	$: a[\gamma \circ \delta] \equiv a[\gamma][\delta]$
$[\mathsf{id}]$	$: a[\mathsf{id}] \equiv a$
$- \triangleright -$	$: (\Gamma : \mathsf{Con}) \to \mathsf{Ty}\,\Gamma \to \mathsf{Con}$
$-,-$	$: (\gamma : \mathsf{Sub}\,\Delta\,\Gamma) \to \mathsf{Tm}\,\Delta\,(A[\gamma]) \to \mathsf{Sub}\,\Delta\,(\Gamma \triangleright A)$
p_A	$: \mathsf{Sub}\,(\Gamma \triangleright A)\,\Gamma$
q_A	$: \mathsf{Tm}\,(\Gamma \triangleright A)\,(A[\mathsf{p}])$
$\triangleright\beta_1$	$: \mathsf{p} \circ (\gamma, a) \equiv \gamma$
$\triangleright\beta_2$	$: \mathsf{q}[\gamma, a] \equiv a$
$\triangleright\eta$	$: (\mathsf{p} \circ \gamma a, \mathsf{q}[\gamma a]) \equiv \gamma a$

$\top$	$: \mathsf{Ty}\,\Gamma$
$\top[]$	$: \top[\gamma] \equiv \top$
tt	$: \mathsf{Tm}\,\Gamma\,\top$
$\top\eta$	$: (t : \mathsf{Tm}\,\Gamma\,\top) \to t \equiv \mathsf{tt}$
Σ	$: (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$
$\Sigma[]$	$: (\Sigma\,A\,B)[\gamma] \equiv \Sigma\,(A[\gamma])\,(B[\gamma \circ \mathsf{p}, \mathsf{q}])$
$-,-$	$: (a : \mathsf{Tm}\,\Gamma\,A) \to \mathsf{Tm}\,\Gamma\,(B[\mathsf{id}, a]) \to \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B)$
π_1	$: \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B) \to \mathsf{Tm}\,\Gamma\,A$
π_2	$: (ab : \mathsf{Tm}\,\Gamma\,(\Sigma\,A\,B)) \to \mathsf{Tm}\,\Gamma\,(B[\mathsf{id}, \pi_1\,ab])$
$\Sigma\beta_1$	$: \pi_1\,(a, b) \equiv a$
$\Sigma\beta_1$	$: \pi_2\,(a, b) \equiv b$
$\Sigma\eta$	$: (\pi_1\,ab, \pi_2\,ab) \equiv ab$
$,[]$	$: (a, b)[\gamma] \equiv (a[\gamma], b[\gamma])$
Π	$: (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Ty}\,(\Gamma \triangleright A) \to \mathsf{Ty}\,\Gamma$
$\Pi[]$	$: (\Pi\,A\,B)[\gamma] \equiv \Pi\,(A[\gamma])\,(B[\gamma \circ \mathsf{p}, \mathsf{q}])$
lam	$: \mathsf{Tm}\,(\Gamma \triangleright A)\,B \to \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$
app	$: \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B) \to \mathsf{Tm}\,(\Gamma \triangleright A)\,B$
$\Pi\beta$	$: \mathsf{app}\,(\mathsf{lam}\,t) \equiv t$
$\Pi\eta$	$: \mathsf{lam}\,(\mathsf{app}\,t) \equiv t$
$\mathsf{lam}[]$	$: (\mathsf{lam}\,t)[\gamma] \equiv \mathsf{lam}\,(t[\gamma \circ \mathsf{p}, \mathsf{q}])$
$\bot$	$: \mathsf{Ty}\,\Gamma$
$\bot[]$	$: \bot[\gamma] \equiv \bot$
$\mathsf{elim}_\bot$	$: \mathsf{Tm}\,\Gamma\,\bot \to \mathsf{Tm}\,\Gamma\,A$
$\mathsf{elim}_\bot[]$	$: (\mathsf{elim}_\bot\,t)[\gamma] \equiv \mathsf{elim}_\bot\,(t[\gamma])$
Id_-	$: (A : \mathsf{Ty}\,\Gamma) \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Tm}\,\Gamma\,A \to \mathsf{Ty}\,\Gamma$
$\mathsf{Id}[]$	$: (\mathsf{Id}_A\,a\,a')[\gamma] \equiv \mathsf{Id}_{A[\gamma]}\,(a[\gamma])\,(a'[\gamma])$
refl	$: \mathsf{Tm}\,\Gamma\,(\mathsf{Id}_A\,a\,a)$
$\mathsf{refl}[]$	$: \mathsf{refl}[\gamma] \equiv \mathsf{refl}$
J	$: (P : \mathsf{Ty}\,(\Gamma \triangleright A \triangleright \mathsf{Id}_{A[\mathsf{p}]}\,(a[\mathsf{p}])\,\mathsf{q})) \to \mathsf{Tm}\,\Gamma\,(P[\mathsf{id}, a, \mathsf{refl}]) \to (e : \mathsf{Tm}\,\Gamma\,(\mathsf{Id}_A\,a\,a')) \to \mathsf{Tm}\,\Gamma\,(P[\mathsf{id}, a', e])$
$\mathsf{J}\beta$	$: \mathsf{J}\,P\,w\,\mathsf{refl} \equiv w$
$\mathsf{J}[]$	$: (\mathsf{J}\,P\,w\,e)[\gamma] \equiv \mathsf{J}\,(P[\gamma \circ \mathsf{p} \circ \mathsf{p}, \mathsf{q}[\mathsf{p}], \mathsf{q}])\,(w[\gamma])\,(e[\gamma])$

Figure 1 A model of type theory with $\top$, Σ, Π, $\bot$, Id. The left column is the definition of CwF, the right column contains the rules for the type formers, one after the other, in the same order.

$$
\begin{array}{ll}
,\circ & : (\gamma, a) \circ \delta \equiv (\gamma \circ \delta, a[\delta]) \\
\pi_1[] & : (\pi_1\, ab)[\gamma] \equiv \pi_1\,(ab[\gamma]) \\
\pi_2[] & : (\pi_2\, ab)[\gamma] \equiv \pi_2\,(ab[\gamma]) \\
- \times - & : \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \\
A \times B & :\equiv \Sigma\, A\,(B[\mathsf{p}]) \\
\mathsf{app}[] & : (\mathsf{app}\, t)[\gamma \circ \mathsf{p}, \mathsf{q}] \equiv \mathsf{app}\,(t[\gamma]) \\
- \$ - & : \mathsf{Tm}\,\Gamma\,(\Pi\, A\, B) \to (a : \mathsf{Tm}\,\Gamma\, A) \to \mathsf{Tm}\,\Gamma\,(B[\mathsf{id}, a]) \\
t \,\$\, a & :\equiv (\mathsf{app}\, t)[\mathsf{id}, a] \\
\$\beta & : \mathsf{lam}\, t \,\$\, a \equiv t[\mathsf{id}, a] \\
\$[] & : (t \,\$\, a)[\gamma] \equiv t[\gamma] \,\$\, a[\gamma] \\
- \Rightarrow - & : \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \to \mathsf{Ty}\,\Gamma \\
A \Rightarrow B & :\equiv \Pi\, A\,(B[\mathsf{p}])
\end{array}
$$

Figure 2 Provable equations and definable operations in a model of type theory with Σ, Π.

$$
\begin{array}{ll}
\mathsf{TyP} & : \mathsf{Con} \to \mathsf{Set} \\
-[-] & : \mathsf{TyP}\,\Gamma \to \mathsf{Sub}\,\Delta\,\Gamma \to \mathsf{TyP}\,\Delta \\
[\circ] & : A[\gamma \circ \delta] \equiv A[\gamma][\delta] \\
[\mathsf{id}] & : A[\mathsf{id}] \equiv A \\
\uparrow & : \mathsf{TyP}\,\Gamma \to \mathsf{Ty}\,\Gamma \\
\uparrow[] & : (\uparrow A)[\gamma] \equiv \uparrow(A[\gamma]) \\
\mathsf{irr} & : (u\, v : \mathsf{Tm}\,\Gamma\,(\uparrow A)) \to u \equiv v \\
\top\mathsf{P} & : \mathsf{TyP}\,\Gamma \\
\top\mathsf{P}[] & : \top\mathsf{P}[\gamma] \equiv \top\mathsf{P} \\
\mathsf{ttP} & : \mathsf{Tm}\,\Gamma\,\top\mathsf{P} \\
\Sigma\mathsf{P} & : (A : \mathsf{TyP}\,\Gamma) \to \mathsf{TyP}\,(\Gamma \rhd \uparrow A) \to \mathsf{TyP}\,\Gamma \\
\Sigma\mathsf{P}[] & : (\Sigma\mathsf{P}\, A\, B)[\gamma] \equiv \Sigma\mathsf{P}\,(A[\gamma])\,(B[\gamma \circ \mathsf{p}, \mathsf{q}]) \\
-,\mathsf{P}- & : (a : \mathsf{Tm}\,\Gamma\,(\uparrow A)) \to \mathsf{Tm}\,\Gamma\,(\uparrow B[\mathsf{id}, a]) \to \mathsf{Tm}\,\Gamma\,(\uparrow \Sigma\mathsf{P}\, A\, B) \\
\pi_1\mathsf{P} & : \mathsf{Tm}\,\Gamma\,(\uparrow \Sigma\mathsf{P}\, A\, B) \to \mathsf{Tm}\,\Gamma\, A \\
\pi_2\mathsf{P} & : (ab : \mathsf{Tm}\,\Gamma\,(\uparrow \Sigma\mathsf{P}\, A\, B)) \to \mathsf{Tm}\,\Gamma\,(\uparrow B[\mathsf{id}, \pi_1\, ab]) \\
\Pi\mathsf{P} & : (A : \mathsf{Ty}\,\Gamma) \to \mathsf{TyP}\,(\Gamma \rhd A) \to \mathsf{TyP}\,\Gamma \\
\Pi\mathsf{P}[] & : (\Pi\mathsf{P}\, A\, B)[\gamma] \equiv \Pi\mathsf{P}\,(A[\gamma])\,(B[\gamma \circ \mathsf{p}, \mathsf{q}]) \\
\mathsf{lamP} & : \mathsf{Tm}\,(\Gamma \rhd A)\,(\uparrow B) \to \mathsf{Tm}\,\Gamma\,(\uparrow \Pi\mathsf{P}\, A\, B) \\
\mathsf{appP} & : \mathsf{Tm}\,\Gamma\,(\uparrow \Pi\mathsf{P}\, A\, B) \to \mathsf{Tm}\,(\Gamma \rhd A)\,(\uparrow B)
\end{array}
$$

Figure 3 A model of type theory has a sort of proof-irrelevant propositions closed under ⊤, Σ, Π.

Variables are represented using typed De Bruijn indices. The zero De Bruijn index is $\mathsf{q} : \mathsf{Tm}\,(\Gamma \rhd A)\,(A[\mathsf{p}])$, the one index is given by $\mathsf{q}[\mathsf{p}] : \mathsf{Tm}\,(\Gamma \rhd A \rhd B)\,(A[\mathsf{p}][\mathsf{p}])$, two is $\mathsf{q}[\mathsf{p}][\mathsf{p}] : \mathsf{Tm}\,(\Gamma \rhd A \rhd B \rhd C)\,(A[\mathsf{p}][\mathsf{p}][\mathsf{p}])$, and so on. Some provable equations and definable operations are listed in Figure 2.

The right hand side of Figure 1 lists rules for $\top$, Σ, Π, $\bot$ and Id types, in this order. The first three type formers have η laws, the latter two don't (they are instances of inductive types). Every operation comes with substitution laws (e.g. $\mathsf{lam}[]$), some of them are not listed as they are provable (see Figure 2). Non-dependent special cases of Π and Σ are also listed there.

Figure 3 lists the operations and equations for a model having a sort of definitionally proof-irrelevant propositions TyP which is closed under $\top$, Σ and Π. Terms of propositional types are expressed with the help of lifting $\Uparrow$ which converts a TyP into a Ty. Because of irr, there is no need to state equations for terms of lifted types, all equations hold.

Two important properties of models that we sometimes assume are canonicity [10] and normalisation [6, 10]. Canonicity for $\bot$ says that there is no $\mathsf{Tm}\,\diamond\,\bot$. Canonicity for Id says that for any $t : \mathsf{Tm}\,\diamond\,(\mathsf{Id}_A\,a\,a')$, we have $a \equiv a'$ and $t \equiv \mathsf{refl}$. Normalisation says that there is a function from terms to normal forms $\mathsf{norm} : \mathsf{Tm}\,\Gamma\,A \to \mathsf{Nf}\,\Gamma\,A$ such that all terms are equal to their normalised versions ($\ulcorner - \urcorner$ is the inclusion from Nf to Tm): for all $a : \mathsf{Tm}\,\Gamma\,A$, $\ulcorner \mathsf{norm}\,a \urcorner = a$. Normal forms for the theory of Figure 1 are defined mutually with variables and neutral terms by the following three inductive types.

$x ::= \mathsf{q} \mid x[\mathsf{p}]$	variables
$n ::= x \mid \pi_1\,n \mid \pi_2\,n \mid n\,\$\,v \mid \mathsf{elim}_\bot\,n \mid \mathsf{J}\,A\,v\,n$	neutral terms
$v := n^* \mid \mathsf{tt} \mid (v, v) \mid \mathsf{lam}\,v \mid \mathsf{refl}$	normal forms

These should be understood as typed rules and there is a restriction (n^*) that only neutral terms at base types are included in normal forms. Base types are $\bot$ and Id in our case.

Sometimes we just talk about intensional type theory when we don't want to specify precisely what type formers we have in a model.

3 Point-free propositions internally

In this section we show that (the dependent variant of) point-free propositions is closed under $\top$, Σ and Π. We work internally to a model of type theory with a universe Type closed under type formers Id, $\top$, Σ, Π. This section was formalised in Agda [12].

The η rule for $\top$ says that for any two $t, t' : \top$ we have $t \equiv t'$, so we also have that $(\lambda(t\,t' : \top).t) \equiv (\lambda t\,t'.t')$, hence $\mathsf{refl} : \big((\lambda t\,t'.t) = (\lambda t\,t'.t')\big) \equiv \mathsf{isPfProp}\,\top$.

As a warmup for Σ, we prove closure under non-dependent products.

▶ **Proposition 1.** *If* $\mathsf{isPfProp}\,A$ *and* $\mathsf{isPfProp}\,B$*, then* $\mathsf{isPfProp}\,(A \times B)$*.*

Proof. We assumed $p_A : \mathsf{isPfProp}\,A \equiv \big((\lambda(a\,a' : A).a) = (\lambda a\,a'.a')\big)$ and $p_B : \mathsf{isPfProp}\,B \equiv \big((\lambda(b\,b' : B).b) = (\lambda b\,b'.b')\big)$ and we want to obtain that $A \times B$ is a point-free proposition.

$$\begin{aligned} p_{A\times B} : \mathsf{isPfProp}\,(A \times B) &\equiv \big((\lambda ab\,ab'.ab) = (\lambda ab\,ab'.ab')\big) \equiv \\ &\big((\lambda ab\,ab'.(\pi_1\,ab, \pi_2\,ab)) = (\lambda ab\,ab'.(\pi_1\,ab', \pi_2\,ab'))\big) \end{aligned}$$

When rewriting the type of $p_{A\times B}$, we applied the η rule for products which says that $ab \equiv (\pi_1\, ab, \pi_2\, ab)$ for any $ab : A \times B$. Then we prove the equality in two steps: first we use p_A to show that $\pi_1\, ab = \pi_1\, ab'$ while we keep the $\pi_2\, ab$ component constant

$$\begin{aligned}
&p^1_{A\times B} : \big(\lambda ab\, ab'.(\pi_1\, ab, \pi_2\, ab)\big) = \big(\lambda ab\, ab'.(\pi_1\, ab', \pi_2\, ab)\big)\\
&p^1_{A\times B} :\equiv \mathsf{ap}\,\big(\lambda z.\lambda ab\, ab'.(z\,(\pi_1\, ab)\,(\pi_1\, ab'), \pi_2\, ab)\big)\, p_A,
\end{aligned}$$

then we use p_B to show that $\pi_2\, ab = \pi_2\, ab'$ while we keep the $\pi_1\, ab'$ components constant. In the middle we have the function returning the mixed pair $(\pi_1\, ab', \pi_2\, ab)$.

$$\begin{aligned}
&p^2_{A\times B} : \big(\lambda ab\, ab'.(\pi_1\, ab', \pi_2\, ab)\big) = \big(\lambda ab\, ab'.(\pi_1\, ab', \pi_2\, ab')\big)\\
&p^2_{A\times B} :\equiv \mathsf{ap}\,\big(\lambda z.\lambda ab\, ab'.(\pi_1\, ab', z\,(\pi_2\, ab)\,(\pi_2\, ab'))\big)\, p_B
\end{aligned}$$

We obtain the desired equality via transitivity:

$$p_{A\times B} :\equiv p^1_{A\times B} \centerdot p^2_{A\times B} \qquad ◀$$

To show closure of point-free propositions under Σ types, we have $A : \mathsf{Type}$, $B : A \to \mathsf{Type}$, $\mathsf{isPfProp}\, A$, but assuming $(a : A) \to \mathsf{isPfProp}\,(B\, a)$ is not enough. We express that B is a family of propositions using a dependent version of $\mathsf{isPfProp}$:

$$\begin{aligned}
&\mathsf{isPfPropd} : (A \to \mathsf{Type}) \to \mathsf{Type}\\
&\mathsf{isPfPropd}\, B :\equiv \big(\lambda(a : A)(b\, b' : B\, a).b\big) = (\lambda a\, b\, b'.b')
\end{aligned}$$

The non-dependent version is a special case when there is an element of the indexing type $a_0 : A$, because given $B : \mathsf{Type}$ and $p_B : \mathsf{isPfPropd}\,(\lambda(a : A).B)$, we have $\mathsf{ap}\,(\lambda z.z\, a_0)\, p_B : \mathsf{isPfProp}\, B$.

We show the dependent version of closure under Σ types.

▶ **Proposition 2.** *Given $A : D \to \mathsf{Type}$ and $B : \Sigma\, D\, A \to \mathsf{Type}$, if $\mathsf{isPfPropd}\, A$ and $\mathsf{isPfPropd}\, B$, then $\mathsf{isPfPropd}\,(\lambda d.\Sigma\,(A\, d)\,(\lambda a.B\,(d, a)))$.*

Proof. We have $p_A : \mathsf{isPfPropd}\, A \equiv (\lambda d\, a\, a'.a) = (\lambda d\, a\, a'.a')$ and $p_B : \mathsf{isPfPropd}\, B \equiv (\lambda da\, b\, b'.b) = (\lambda da\, b\, b'.b')$, our goal is to obtain

$$p_{\Sigma AB} : (\lambda d\, ab\, ab'.ab) = (\lambda d\, ab\, ab'.ab') \equiv (\lambda d\, ab\, ab'.(\pi_1\, ab, \pi_2\, ab)) = (\lambda d\, ab\, ab'.(\pi_1\, ab', \pi_2\, ab')).$$

We want to prove this in two steps as for non-dependent products, but because B depends on A, the middle pair $(\pi_1\, ab', \pi_2\, ab)$ is not well-typed. We replace the second component $\pi_2\, ab : B\,(d, \pi_1\, ab)$ with

$$\mathsf{transp}_{\lambda a.B\,(d,a)}\Big(\mathsf{ap}\,\big(\lambda z.z\, d\,(\pi_1\, ab)\,(\pi_1\, ab')\big)\, p_A\Big)\,(\pi_2\, ab) \;:\; B\,(d, \pi_1\, ab'),$$

and we will use a more general version of this second component defined as

$$f\, ab\, ab'\, e := \mathsf{transp}_{\lambda a.B\,(d,a)}\Big(\mathsf{ap}\,\big(\lambda z.z\, d\,(\pi_1\, ab)\,(\pi_1\, ab')\big)\, e\;\Big)\,(\pi_2\, ab) \;:\; B\,(d, h\, d\,(\pi_1\, ab)\,(\pi_1\, ab'))$$

for any d, ab, ab', h and $e : (\lambda d\, a\, a'.a) = h$. Now the first step has type

$$p^1_{\Sigma AB} : (\lambda d\, ab\, ab'.ab) = \big(\lambda d\, ab\, ab'.(\pi_1\, ab', f\, ab\, ab'\, p_A)\big)$$

and we prove it by induction on p_A using J:

$$p^1_{\Sigma AB} :\equiv \mathsf{J}\,\Big(\lambda h\, e.(\lambda d\, ab\, ab'.ab) = \big(\lambda d\, ab\, ab'.(h\, d\,(\pi_1\, ab)\,(\pi_1\, ab'), f\, ab\, ab'\, e)\big)\Big)\,\mathsf{refl}\, p_A$$

In the next step we simply use ap on p_B and we conclude by transitivity:

$$p^2_{\Sigma AB} : \big(\lambda d\,ab\,ab'.(\pi_1\,ab', f\,ab\,ab'\,p_A)\big) = (\lambda d\,ab\,ab'.ab')$$
$$p^2_{\Sigma AB} :\equiv \mathsf{ap}\,\Big(\lambda z.\lambda d\,ab\,ab'.(\pi_1\,ab', z\,(d, \pi_1\,ab')\,(f\,ab\,ab'\,p_A)\,(\pi_2\,ab'))\Big)\,p_B$$
$$p_{\Sigma AB} :\equiv p^1_{\Sigma AB} \mathbin{\blacksquare} p^2_{\Sigma AB} \qquad \blacktriangleleft$$

▶ **Corollary 3.** *For A : Type and $B : A \to \mathsf{Type}$, if $\mathsf{isPfProp}\,A$ and $\mathsf{isPfPropd}\,B$, then $\mathsf{isPfProp}\,(\Sigma\,A\,B)$.*

Finally, we show closure of $\mathsf{isPfPropd}$ under dependent function space.

▶ **Proposition 4.** *Given $A : D \to \mathsf{Type}$, $B : \Sigma\,D\,A \to \mathsf{Type}$, if $\mathsf{isPfPropd}\,B$, then $\mathsf{isPfPropd}\,(\lambda d.(a : A\,d) \to B\,(d, a))$.*

Proof. Using $p_B : (\lambda da\,b\,b'.b) = (\lambda da\,b\,b'.b')$, we define

$$p_{\Pi AB} : (\lambda d\,f\,f'.f) = (\lambda d\,f\,f'.f') \equiv (\lambda d\,f\,f'\,a.f\,a) = (\lambda d\,f\,f'\,a.f'\,a)$$
$$p_{\Pi AB} :\equiv \mathsf{ap}\,(\lambda z\,d\,f\,f'\,a.z\,(d, a)\,(f\,a)\,(f'\,a))\,p_B. \qquad \blacktriangleleft$$

▶ **Corollary 5.** *For A : Type and $B : A \to \mathsf{Type}$, if $\mathsf{isPfPropd}\,B$, then $\mathsf{isPfProp}\,((a : A) \to B\,a)$.*

4 Point-free propositions externally

In this section we show that any model of type theory with $\top$, Σ, Π types has a sort TyP closed under the same type formers. This can be seen as an externalisation of the previous section.

Recall that a model of type theory (a CwF, see Section 2) has a sort of strict propositions if there is a presheaf TyP together with a "lifting" natural transformation $\uparrow$ into Ty, and terms of a lifted type are equal.

First we define a predicate on types expressing externally that the type is a point-free proposition.

▶ **Definition 6.** *For a type $A : \mathsf{Ty}\,\Gamma$ in any CwF, let $\mathsf{isExtPfProp}\,A :\equiv (\mathsf{q}_A[\mathsf{p}_{A[\mathsf{p}]}] \equiv \mathsf{q}_{A[\mathsf{p}]})$.*

That is, in the context $\Gamma \rhd A \rhd A[\mathsf{p}]$, the terms $\mathsf{q}[\mathsf{p}]$ and q (1 and 0 De Bruijn indices, both having type $A[\mathsf{p}][\mathsf{p}]$) are definitionally equal. We call this the external variant of pfProp because it is clear that it is equivalent to saying $\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])) \equiv \mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})$ which is the external statement of $\lambda x\,y.x = \lambda x\,y.y$. In the next section, we will relate the external and internal variants formally.

Elements of a type which $\mathsf{isExtPfProp}$ are equal in any context.

▶ **Proposition 7.** *For a type A, $\mathsf{isExtPfProp}\,A$ is equivalent to*

$u \equiv v$ for all $\gamma : \mathsf{Sub}\,\Delta\,\Gamma$ and $u, v : \mathsf{Tm}\,\Delta\,(A[\gamma])$.

Proof. Left to right: we have $\mathsf{q}[\mathsf{p}][\gamma, u, v] \equiv \mathsf{q}[\gamma, u, v]$, hence $u \equiv v$. Right to left: we choose $u :\equiv \mathsf{q}[\mathsf{p}]$, $v :\equiv \mathsf{q}$. ◀

In category theory, external point-free propositions over Γ are called subobjects of Γ.

▶ **Proposition 8.** *For an $A : \mathsf{Ty}\,\Gamma$, $\mathsf{isExtPfProp}\,A$ is equivalent to the morphism $\mathsf{p}_A : \mathsf{Sub}\,(\Gamma \rhd A)\,\Gamma$ being a monomorphism.*

Proof. Left to right: given $\mathsf{p}_A \circ (\gamma, a) \equiv \mathsf{p}_A \circ (\gamma', a')$, we need to show $(\gamma, a) \equiv (\gamma', a')$. Using the assumption we have $\gamma \equiv \mathsf{p}_A \circ (\gamma, a) \equiv \mathsf{p}_A \circ (\gamma', a') \equiv \gamma'$, hence a and a' are both in $\mathsf{Tm}\,\Delta\,(A[\gamma])$. We get $a \equiv a'$ from Proposition 7.

Right to left: given two terms $a, a' : \mathsf{Tm}\,\Gamma\,(A[\gamma])$, we have $\mathsf{p}_A \circ (\gamma, a) \equiv \gamma \equiv \mathsf{p}_A \circ (\gamma, a')$, hence by assumption $(\gamma, a) \equiv (\gamma, a')$ and applying $\mathsf{q}[-]$ to both sides we obtain $a \equiv a'$. ◀

▶ **Construction 9.** *Every CwF with $\top$, Σ, Π can be equipped with a sort of strict propositions closed under the same type formers.*

Construction. We have to define all components in Figure 3. We define

$$\mathsf{TyP}\,\Gamma :\equiv (A : \mathsf{Ty}\,\Gamma) \times \mathsf{isExtPfProp}\,A.$$

Substitution is defined by ordinary type substitution of the first component and the equation for substituted types holds by the following argument.

$$\begin{aligned}
&\mathsf{q}_{A[\gamma]}[\mathsf{p}_{A[\gamma][\mathsf{p}]}] && \equiv ([\circ], \triangleright\beta_1, \triangleright\beta_2)\\
&\mathsf{q}_A[\mathsf{p}_{A[\mathsf{p}]}][\gamma \circ \mathsf{p} \circ \mathsf{p}, \mathsf{q}_{A[\gamma]}[\mathsf{p}_{A[\gamma][\mathsf{p}]}], \mathsf{q}_{A[\gamma][\mathsf{p}]}] && \equiv (\text{assumption})\\
&\mathsf{q}_{A[\mathsf{p}]}[\gamma \circ \mathsf{p} \circ \mathsf{p}, \mathsf{q}_{A[\gamma]}[\mathsf{p}_{A[\gamma][\mathsf{p}]}], \mathsf{q}_{A[\gamma][\mathsf{p}]}] && \equiv (\triangleright\beta_2)\\
&\mathsf{q}_{A[\gamma][\mathsf{p}]})
\end{aligned}$$

The $\uparrow$ operation is defined by $\uparrow(A, p_A) :\equiv A$. Irrelevance holds by Proposition 7. $\top\mathsf{P}$ is defined as $\top$ and $\mathsf{isExtPfProp}\,\top$ holds by $\top\eta$. We define $\Sigma\mathsf{P}\,(A, p_A)\,(B, p_B)$ by $(\Sigma\,A\,B, p_{\Sigma AB})$ where $p_{\Sigma AB}$ is proven using Proposition 7 for $u, v : \mathsf{Tm}\,\Delta\,(\Sigma\,A\,B[\gamma])$ by

$$u \overset{\Sigma\eta}{\equiv} (\pi_1\,u, \pi_2\,u) \overset{p_A, p_B}{\equiv} (\pi_1\,v, \pi_2\,v) \overset{\Sigma\eta}{\equiv} v.$$

We define $\Pi\mathsf{P}\,A\,(B, p_B)$ by $(\Pi\,A\,B, p_{\Pi AB})$ where $p_{\Pi AB}$ is proven using Proposition 7 for $u, v : \mathsf{Tm}\,\Delta\,(\Pi\,A\,B[\gamma])$ by

$$u \overset{\Pi\eta}{\equiv} \mathsf{lam}\,(\mathsf{app}\,u) \overset{p_B}{\equiv} \mathsf{lam}\,(\mathsf{app}\,v) \overset{\Pi\eta}{\equiv} v.$$ ◀

5 Relationship of different notions of being a proposition

For a type family $A : D \to \mathsf{Type}$, being a family of homotopy propositions and a family of point-free propositions were defined internally as follows.

$$\begin{aligned}
\mathsf{isHPropd}\,A &\equiv (d : D)(a\,a' : A\,d) \to \mathsf{Id}_{(A\,d)}\,a\,a'\\
\mathsf{isPfPropd}\,A &\equiv \mathsf{Id}_{((d:D)\to A\,d \to A\,d \to A\,d)}\,(\lambda d\,a\,a'.a)\,(\lambda d\,a\,a'.a')
\end{aligned}$$

Externally, these can be seen as the following two elements of $\mathsf{Ty}\,\bullet$ for $A : \mathsf{Ty}\,(\bullet \triangleright D)$. We also repeat the definition of $\mathsf{isExtPfProp}$ for comparison which is a metatheoretic equality.

$$\begin{aligned}
\mathsf{isHPropd}\,A &\equiv \Pi\,D\,\Big(\Pi\,A\,(\Pi\,(A[\mathsf{p}])\,(\mathsf{Id}_{A[\mathsf{p}][\mathsf{p}]}\,(\mathsf{q}[\mathsf{p}])\,\mathsf{q}))\Big)\\
\mathsf{isPfPropd}\,A &\equiv \mathsf{Id}_{\Pi\,D\,(A \Rightarrow A \Rightarrow A)}\,\big(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])))\big)\,\big(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q}))\big)\\
\mathsf{isExtPfProp}\,A &\equiv (\mathsf{q}[\mathsf{p}] \equiv \mathsf{q}) \qquad\qquad (\text{both sides in } \mathsf{Tm}\,(\bullet \triangleright D \triangleright A \triangleright A[\mathsf{p}])\,(A[\mathsf{p}][\mathsf{p}]))
\end{aligned}$$

We first compare internal point-free propositions and external ones. They coincide for a type where we collect all dependencies into a single closed type D.

▶ **Proposition 10.** *In a model of type theory with* Π, Id *and canonicity, given an* $A : \mathsf{Ty}\,(\diamond \triangleright D)$*, there is a* $\mathsf{Tm}\ \diamond\ (\mathsf{isPfPropd}\,A)$ *if and only if* $\mathsf{isExtPfProp}\,A$*.*

Proof. Right to left: if $\mathsf{q}[\mathsf{p}] \equiv \mathsf{q}$ in $\mathsf{Tm}\,(\diamond \triangleright D)\,A$, then $\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}]))) \equiv \mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q}))$, hence $\mathsf{refl} : \mathsf{Tm}\ \diamond\ (\mathsf{isPfPropd}\,A)$.

Left to right: we have $t : \mathsf{Tm}\,\diamond\,\Big(\mathsf{Id}_{\Pi\,D\,(A\Rightarrow A\Rightarrow A)}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}]))))(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q})))\Big)$. Canonicity for Id implies that $\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}]))) \equiv \mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q}))$, hence

$$\mathsf{app}\,(\mathsf{app}\,(\mathsf{app}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])))))) \equiv \mathsf{app}\,(\mathsf{app}\,(\mathsf{app}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q}))))).$$

Now using $\Pi\beta$ three times on both sides we obtain $\mathsf{q}[\mathsf{p}] \equiv \mathsf{q}$ where both sides are in $\mathsf{Tm}\,(\diamond \triangleright D \triangleright A \triangleright A[\mathsf{p}])\,(A[\mathsf{p}][\mathsf{p}])$, and this is $\mathsf{isExtPfProp}\,A$. ◀

▶ **Corollary 11.** *In a model of type theory with* Π, Id *and canonicity, given a closed type* A*,* $\mathsf{Tm}\ \diamond\ (\mathsf{isPfProp}\,A)$ *if and only if* $\mathsf{isExtPfProp}\,A$*.*

In an open context, external point-free propositions are stronger than internal ones.

▶ **Proposition 12.** *In a model of type theory with* Π, Id*, a type* U *and a family over it* El *(a possibly empty universe) and normalisation, we have* $A : \mathsf{Ty}\,\Gamma$ *such that* $\mathsf{Tm}\,\Gamma\,(\mathsf{isPfProp}\,A)$*, but not* $\mathsf{isExtPfProp}\,A$*.*

Proof. Pick $\Gamma :\equiv \diamond \triangleright \mathsf{U} \triangleright \mathsf{Id}_{\mathsf{El}\,\mathsf{q}\Rightarrow\mathsf{El}\,\mathsf{q}\Rightarrow\mathsf{El}\,\mathsf{q}}\,(\mathsf{lam}\,(\mathsf{lam}\,(\mathsf{q}[\mathsf{p}])))\,(\mathsf{lam}\,(\mathsf{lam}\,\mathsf{q}))$ and $A :\equiv \mathsf{El}\,(\mathsf{q}[\mathsf{p}])$. Now $\mathsf{q}[\mathsf{p}]$ and q both in $\mathsf{Tm}\,(\Gamma \triangleright A \triangleright A[\mathsf{p}])\,(A[\mathsf{p}\circ\mathsf{p}])$ have different normal forms. ◀

Next, we describe the relationship of homotopy and point-free propositions. Here we use the non-dependent variants.

▶ **Proposition 13.**

(i) *In a model of type theory with* Π *and* Id*,* $\mathsf{isPfProp}\,A$ *implies* $\mathsf{isHProp}\,A$*.*

(ii) *In a model of type theory with* Π*,* Id*, an inductively defined* $\bot$ *and normalisation,*

- **(a)** *we have* $\mathsf{isHProp}\,\bot$*, but not* $\mathsf{isPfProp}\,\bot$*.*
- **(b)** *we don't have that for any type* A*,* $\mathsf{isPfProp}\,(\mathsf{isPfProp}\,A)$*.*

(iii) *In a model of type theory with* Π*,* Id *and function extensionality,* $\mathsf{isHProp}\,A$ *implies* $\mathsf{isPfProp}\,A$*.*

Proof.

(i) We work internally. Given $p_A : \mathsf{isPfProp}\,A \equiv (\lambda(a\,a' : A).a) = (\lambda a\,a'.a')$, we define $\lambda a\,a'.\mathsf{ap}\,(\lambda z.z\,a\,a')\,p_A : \mathsf{isHProp}\,A$.

(ii) **(a)** Internally, we have $\lambda b.\mathsf{elim}_\bot\,b : \mathsf{isHProp}\,\bot$. Let's assume $\mathsf{Tm}\ \diamond\ (\mathsf{isPfProp}\,\bot)$. From Corollary 11 and Proposition 7, any two elements of $\bot$ in any context are equal. But from normalisation we have $\mathsf{q}[\mathsf{p}] \not\equiv \mathsf{q} : \mathsf{Tm}\,(\diamond \triangleright \bot \triangleright \bot)\,\bot$ as they have different normal forms.

(b) Assuming $\mathsf{isPfProp}\,(\mathsf{isPfProp}\,\bot)$, we obtain

$$\mathsf{q}[\mathsf{p}] \equiv \mathsf{q} : \mathsf{Tm}\,(\diamond \triangleright \mathsf{isPfProp}\,\bot \triangleright \mathsf{isPfProp}\,\bot)\,(\mathsf{isPfProp}\,\bot)$$

the same way as in (a), but they have different normal forms.

(iii) We have to show that $\mathsf{isHProp}\,A$ implies $\mathsf{isPfProp}\,A$. We work internally by the following double application of function extensionality.

$$\begin{aligned}
&\mathsf{isHProp}\,A && \equiv\\
&\big((a\,a' : A) \to a = a'\big) && \equiv\\
&\big((a : A)(a' : A) \to (\lambda a'.a)\,a' = (\lambda a'.a')\,a'\big) && \to \text{(function extensionality)}\\
&\big((a : A) \to (\lambda a'.a) = (\lambda a'.a')\big) && \equiv\\
&\big((a : A) \to (\lambda a\,a'.a)\,a = (\lambda a\,a'.a')\,a\big) && \to \text{(function extensionality)}\\
&(\lambda a\,a'.a') = (\lambda a\,a'.a') && \equiv\\
&\mathsf{isPfProp}\,A
\end{aligned}$$

◀

From the previous section, we know that TyP can be defined using isExtPfProp. If we start with a model that already has TyP, it is natural to ask about the relationship of TyP and the other notions of being a proposition.

▶ **Proposition 14.**

(i) *In a model of type theory with* Π, Id *and* TyP, *if* $A : \mathsf{TyP}\,\Gamma$, *then* $\mathsf{Tm}\,\Gamma\,(\mathsf{isHProp}\,(\uparrow A))$, $\mathsf{Tm}\,\Gamma\,(\mathsf{isPfProp}\,(\uparrow A))$ *and* $\mathsf{isExtPfProp}\,(\uparrow A)$.

(ii) *In a model of type theory with* Π, Σ *and* Id, *if for every type* A, $\mathsf{isHProp}\,A$ *implies* $\mathsf{isExtPfProp}\,A$, *then the model has equality reflection.*

Proof.

(i) Because any two terms of type $\uparrow A$ are definitionally equal by irr, internally $\lambda a\,a'.\mathsf{refl} : \mathsf{isHProp}\,A$ and $\mathsf{refl} : \mathsf{isPfProp}\,A$.

(ii) The proof is from [13]. Singleton types are in hProp, that is, internally $\mathsf{isHProp}\,((a' : A) \times \mathsf{Id}_A\,a\,a')$ holds for any a, but if $\mathsf{isExtPfProp}\,((a' : A) \times \mathsf{Id}_A\,a\,a')$, then for any $e : \mathsf{Id}_A\,a\,a'$, we have $(a, \mathsf{refl}) \equiv (a', e)$, hence $a \equiv \pi_1\,(a, \mathsf{refl}) \equiv \pi_1\,(a', e) \equiv a'$.

◀

6 The setoid model externally

In this section, from a model of type theory with $\top$, Σ and Π, we build another model of type theory with the same type formers and a strict identity type, a strong transport rule and function extensionality. Strictness of the identity type means that any two elements of the identity type are definitionally equal (it is an external point-free proposition, isExtPfProp). Strength of transport means that we can transport an element of any family of types, not only families of strict propositions. In contrast, Agda and the method described in [13] only support a strict identity type with a weak transport: the identity type is Prop-valued and we can only transport along Prop-valued families.

In Section 7, we describe an internal version of this model construction where we define a model internally to an intensional metatheory. Section 7 relates to this section as the section on internal point-free propositions (Section 3) relates to the section on external point-free propositions (Section 4). The model construction in this section follows those in [4, 3] with some small improvements, but is defined in a more restricted setting: we do not assume that the input model has a universe of strict propositions.

Note that even if our metatheory is extensional type theory, we do not rely on any extensionality features in the input model. We only use an extensional metatheory for convenience. Following Hofmann's conservativity theorem [14], our arguments can be replayed in an intensional metatheory with function extensionality and uniqueness of identity proofs.

▶ **Construction 15.** *From an input model of type theory with* $\top$*,* Σ*,* Π*, a sort* TyP *closed under* $\top\mathsf{P}$*,* $\Sigma\mathsf{P}$ *and* TyP *(as in Figure 3), we construct a model of type theory with the same type formers and a* TyP*-valued identity type with a strong transport rule and function extensionality.*

Construction. A context in the output model is a context in the input model together with an hProp-valued equivalence relation on substitutions into that context. Note that as our metatheory is extensional type theory, hProp and pfProp coincide.

$$\begin{array}{rll} \mathsf{Con} :\equiv & (|\Gamma| & : \mathsf{Con}) \\ & \times(\Gamma^\sim & : \mathsf{Sub}\,\Xi\,|\Gamma| \to \mathsf{Sub}\,\Xi\,|\Gamma| \to \mathsf{hProp}) \\ & \times(-[-]_\Gamma & : \Gamma^\sim\,\gamma_0\,\gamma_1 \to (\xi : \mathsf{Sub}\,\Xi'\,\Xi) \to \Gamma^\sim\,(\gamma_0 \circ \xi)\,(\gamma_1 \circ \xi)) \\ & \times(\mathsf{R}_\Gamma & : (\gamma : \mathsf{Sub}\,\Xi\,|\Gamma|) \to \Gamma^\sim\,\gamma\,\gamma) \\ & \times(\mathsf{S}_\Gamma & : \Gamma^\sim\,\gamma_0\,\gamma_1 \to \Gamma^\sim\,\gamma_1\,\gamma_0) \\ & \times(\mathsf{T}_\Gamma & : \Gamma^\sim\,\gamma_0\,\gamma_1 \to \Gamma^\sim\,\gamma_1\,\gamma_2 \to \Gamma^\sim\,\gamma_0\,\gamma_2) \end{array}$$

In [4], the relation for contexts was TyP-valued, not metatheoretic proposition (hProp)-valued. We chose to use hProp for reasons of modularity: now the category part of the output model (Con, Sub) only refers to the category part of the input model. Note that the relation for types is TyP-valued.

Substitutions are substitutions in the input model which respect the relation.

$$\mathsf{Sub}\,\Delta\,\Gamma :\equiv (|\gamma| : \mathsf{Sub}\,|\Delta|\,|\Gamma|) \times (\gamma^\sim : \Delta^\sim\,\delta_0\,\delta_1 \to \Gamma^\sim\,(|\gamma| \circ \delta_0)\,(|\gamma| \circ \delta_1))$$

Composition and identities are composition and identities from the input model where the $^\sim$ components are defined by function composition and the identity function. In fact, up to Π types, all the $|-|$ components in the output model are the corresponding components of the input model.

The empty context is defined as $|\diamond| :\equiv \diamond$ and $\diamond^\sim\,\sigma_0\,\sigma_1 :\equiv \top$ which is trivially an equivalence relation.

Types are displayed setoids with TyP-valued relations together with coercion and coherence operations.

$$\begin{array}{ll} \multicolumn{2}{l}{\mathsf{Ty}\,\Gamma :\equiv} \\ \quad(|A| & : \mathsf{Ty}\,|\Gamma|) \\ \times(A^\sim & : \Gamma^\sim\,\gamma_0\,\gamma_1 \to \mathsf{Tm}\,\Xi\,(|A|[\gamma_0]) \to \mathsf{Tm}\,\Xi\,(|A|[\gamma_1]) \to \mathsf{TyP}\,\Xi) \\ \times(A^\sim[] & : (A^\sim\,\gamma_{01}\,a_0\,a_1)[\xi] \equiv A^\sim\,(\gamma_{01}[\xi]_\Gamma)\,(a_0[\xi])\,(a_1[\xi])) \\ \times(\mathsf{R}_A & : (a : \mathsf{Tm}\,\Xi\,(|A|[\gamma])) \to \mathsf{Tm}\,\Xi\,(\uparrow A^\sim\,(\mathsf{R}_\Gamma\,\gamma)\,a\,a)) \\ \times(\mathsf{S}_A & : \mathsf{Tm}\,\Xi\,(\uparrow A^\sim\,\gamma_{01}\,a_0\,a_1) \to \mathsf{Tm}\,\Xi\,(\uparrow A^\sim\,(\mathsf{S}_\Gamma\,\gamma_{01})\,a_1\,a_0)) \\ \times(\mathsf{T}_A & : \mathsf{Tm}\,\Xi\,(\uparrow A^\sim\,\gamma_{01}\,a_0\,a_1) \to \mathsf{Tm}\,\Xi\,(\uparrow A^\sim\,\gamma_{12}\,a_1\,a_2) \to \mathsf{Tm}\,\Xi\,(\uparrow A^\sim\,(\mathsf{T}_\Gamma\,\gamma_{01}\,\gamma_{12})\,a_0\,a_2)) \\ \times(\mathsf{coe}_A & : \mathsf{Tm}\,\Xi\,(\uparrow\Gamma^\sim\,\gamma_0\,\gamma_1) \to \mathsf{Tm}\,\Xi\,(|A|[\gamma_0]) \to \mathsf{Tm}\,\Xi\,(|A|[\gamma_1])) \\ \times(\mathsf{coe}_A[] & : \mathsf{coe}_A\,\gamma_{01}\,a_0[\xi] \equiv \mathsf{coe}_A\,(\gamma_{01}[\xi]_\Gamma)\,(a_0[\xi])) \\ \times(\mathsf{coh}_A & : (\gamma_{01} : \mathsf{Tm}\,\Xi\,(\uparrow\Gamma^\sim\,\gamma_0\,\gamma_1))(a_0 : \mathsf{Tm}\,\Xi\,(|A|[\gamma_0])) \to \mathsf{Tm}\,\Xi\,(\uparrow A^\sim\,\gamma_{01}\,a_0\,(\mathsf{coe}_A\,\gamma_{01}\,a_0))) \end{array}$$

Type substitution is given by type substitution in the input model and function composition for the other components.

Terms are terms which respect the (displayed) equivalence relations.

$$\mathsf{Tm}\,\Gamma\,A :\equiv (|t| : \mathsf{Tm}\,|\Gamma|\,|A|) \times (t^\sim : (\gamma_{01} : \Gamma^\sim\,\gamma_0\,\gamma_1) \to \mathsf{Tm}\,\Xi\,(\uparrow A^\sim\,\gamma_{01}\,(|t|[\gamma_0])\,(|t|[\gamma_1])))$$

Term substitution is given by term substitution in the input model and function composition for the $^\sim$ component.

Context extension is context extension $|\Gamma \rhd A| :\equiv |\Gamma| \rhd |A|$, the relation is given by metatheoretic Σ types: $(\Gamma \rhd A)^\sim\,(\gamma_0, a_0)\,(\gamma_1, a_1) :\equiv (\gamma_{01} : \Gamma^\sim\,\gamma_0\,\gamma_1) \times \mathsf{Tm}\,\Xi\,(\uparrow\! A^\sim\,\gamma_{01}\,a_0\,a_1)$. This is an equivalence relation because $\Gamma^\sim$ is an equivalence relation and $A^\sim$ is a displayed equivalence relation. The $^\sim$ components of $-,-$, p and q are given by pairing and projections for metatheoretic Σ types. The equations $\rhd\beta_1$, $\rhd\beta_2$, $\rhd\eta$ follow from β, η for metatheoretic Σ types. The unit type $\top$ is given by $|\top| :\equiv \top$, $\top^\sim\,\gamma_{01}\,t_0\,t_1 :\equiv \top\mathsf{P}$.

Σ types use $\Sigma\mathsf{P}$ for the relation: we define $|\Sigma\,A\,B| :\equiv \Sigma\,|A|\,|B|$ and

$$(\Sigma\,A\,B)^\sim\,\gamma_{01}\,(a_0, b_0)\,(a_1, b_1) :\equiv \Sigma\mathsf{P}\,(A^\sim\,\gamma_{01}\,a_0\,a_1)\,(B^\sim\,(\gamma_{01}[\mathsf{p}], \mathsf{q})\,(b_0[\mathsf{p}])\,(b_1[\mathsf{p}])).$$

All the other components are pointwise, for example $\mathsf{R}_{\Sigma\,A\,B}\,(a, b) :\equiv (\mathsf{R}_A\,a\,{,}\mathsf{P}\,\mathsf{R}_B\,b)$ and

$$\mathsf{coe}_{\Sigma\,A\,B}\,\gamma_{01}\,(a_0, b_0) :\equiv (\mathsf{coe}_A\,\gamma_{01}\,a_0, \mathsf{coe}_B\,(\gamma_{01}, \mathsf{coh}_A\,\gamma_{01}\,a_0)\,b_0).$$

Pairing, first and second projection and the computation rules are straightforward. Note that to prove e.g. $\pi_1\,(a, b) \equiv a$, it is enough to compare the first components, i.e. $|\pi_1\,(a, b)| \equiv |a|$ as the second components are equal by irr.

For Π types, the $|-|$ component includes $^\sim$ components of the constituent types:

$$\begin{aligned}&|\Pi\,A\,B| :\equiv\\ &\quad \Sigma\,(\Pi\,|A|\,|B|)\\ &\quad\quad \Big(\Pi\mathsf{P}\,(|A|[\mathsf{p}])\,\Big(\Pi\mathsf{P}\,(|A|[\mathsf{p}^2])\,\big(\Pi\mathsf{P}\,(\uparrow\! A^\sim\,(\mathsf{R}_\Gamma\,\mathsf{p}^3)\,(\mathsf{q}[\mathsf{p}])\,\mathsf{q})\\ &\qquad\qquad\qquad\qquad (B^\sim\,(\mathsf{R}_\Gamma\,\mathsf{p}^4, \mathsf{q})\,(\mathsf{q}[\mathsf{p}^3]\,\$\,\mathsf{q}[\mathsf{p}^2])\,(\mathsf{q}[\mathsf{p}^3]\,\$\,\mathsf{q}[\mathsf{p}]))))\Big)\Big)\end{aligned}$$

Functions are given by functions which respect the relation: for any two elements of $|A|$ that are related by $A^\sim$, the outputs of the function are related by $B^\sim$. We wrote p^2 for $\mathsf{p} \circ \mathsf{p}$. With variable names and without weakenings, the same definition is written

$$\Sigma(f : \Pi(a : |A|).|B|).\Pi\mathsf{P}(a_0\,a_1 : |A|, a_{01} : \uparrow\! A^\sim\,(\mathsf{R}_\Gamma\,\mathsf{id})\,a_0\,a_1).B^\sim\,(\mathsf{R}_\Gamma\,\mathsf{id}, a_{01})\,(f\,\$\,a_0)\,(f\,\$\,a_1)).$$

The relation for Π types says that two functions are related if they map related inputs to related outputs:

$$\begin{aligned}&(\Pi\,A\,B)^\sim\,\gamma_{01}\,t_0\,t_1 :\equiv\\ &\ \Pi\mathsf{P}\,(|A|[\gamma_0])\,\Big(\Pi\mathsf{P}\,(|A|[\gamma_1 \circ \mathsf{p}])\,(\Pi\mathsf{P}\,(\uparrow\! A^\sim\,(\gamma_{01}[\mathsf{p}^2]_\Gamma)\,(\mathsf{q}[\mathsf{p}])\,\mathsf{q})\\ &\qquad\qquad\qquad\qquad (B^\sim\,(\gamma_{01}[\mathsf{p}^3]_\Gamma, \mathsf{q})\,(t_0[\mathsf{p}^3]\,\$\,(\mathsf{q}[\mathsf{p}^2]))\,(t_1[\mathsf{p}^3]\,\$\,(\mathsf{q}[\mathsf{p}])))))\Big)\end{aligned}$$

Reflexivity for Π types is second projection: $\mathsf{R}_{\Pi\,A\,B}\,t :\equiv \pi_2\,t$. The other components are defined as in [4]. The definition of lam and app are straightforward. Just as for Π, the definition of $|\mathsf{lam}\,t|$ involves both $|t|$ and $t^\sim$. When comparing two elements of $|\Pi\,A\,B|$ for equality, only the first components of the Σ types have to be compared, the second components are equal by irr.

The sort TyP is defined by TyPs in the input model together with coercion.

$$\mathsf{TyP}\,\Gamma :\equiv (|A| : \mathsf{TyP}\,|\Gamma|) \times (\mathsf{coe}_A : \Gamma^\sim\,\gamma_0\,\gamma_1 \to \mathsf{Tm}\,\Xi\,(\uparrow\!|A|[\gamma_0]) \to \mathsf{Tm}\,\Xi\,(\uparrow\!|A|[\gamma_1]))$$

Compared to Ty which had nine components, TyP has only two. All the other components that Ty had are irrelevant for propositional types. Lifting is given by lifting in the input model, the relation is trivial and coercion comes from the coercion component in TyP:

$$|\uparrow\! A| :\equiv \uparrow\!|A| \qquad\qquad (\uparrow\! A)^\sim\,\gamma_{01}\,a_0\,a_1 :\equiv \top\mathsf{P} \qquad\qquad \mathsf{coe}_{\uparrow A}\,\gamma_{01}\,a_0 :\equiv \mathsf{coe}_A\,\gamma_{01}\,a_0$$

TyP is closed under $\top\mathsf{P}$, $\Sigma\mathsf{P}$ and $\Pi\mathsf{P}$.

Thus we constructed a model of type theory with $\top$, Π, Σ and a sort TyP closed under the same type formers.

This model has an identity type $\mathsf{Id}_A\,a\,a' : \mathsf{TyP}\,\Gamma$ for $a, a' : \mathsf{Tm}\,\Gamma\,A$.

$$
\begin{aligned}
&|\mathsf{Id}_A\,a\,a'| :\equiv A^{\sim}\,(\mathsf{R}_\Gamma\,\mathsf{id})\,a\,a' \qquad\qquad (\mathsf{Id}_A\,a\,a')^{\sim}\,\gamma_{01}\,e_0\,e_1 :\equiv \top\mathsf{P}\\
&\mathsf{coe}_{\mathsf{Id}_A\,a\,a'}\,\gamma_{01}\,\underbrace{e}_{:A^{\sim}\,(\mathsf{R}_\Gamma\,\gamma_0)\,(a[\gamma_0])\,(a'[\gamma_0])} :\equiv \underbrace{\mathsf{T}_A\,\underbrace{(a^{\sim}\,(\mathsf{S}_\Gamma\,\gamma_{01}))}_{:A^{\sim}\,(\mathsf{S}_\Gamma\,\gamma_{01})\,(a[\gamma_1])\,(a[\gamma_0])}\,\big(\mathsf{T}_A\,e\,\underbrace{(a'^{\sim}\,\gamma_{01})}_{:A^{\sim}\,\gamma_{01}\,(a'[\gamma_0])\,(a'[\gamma_1])}\big)}_{:A^{\sim}\,(\mathsf{R}_\Gamma\,\gamma_1)\,(a[\gamma_1])\,(a'[\gamma_1])}
\end{aligned}
$$

It has a constructor refl and an eliminator transp (J is a consequence of transport as equality is proof-irrelevant).

$$
\begin{aligned}
&\mathsf{refl} && : \mathsf{Tm}\,\Gamma\,(\uparrow\!\mathsf{Id}_A\,a\,a)\\
&|\mathsf{refl}| && :\equiv \mathsf{R}_A\,a\\
&\mathsf{refl}^{\sim}\,\gamma_{01} && :\equiv \mathsf{ttP}\\
&\mathsf{transp} && : (P : \mathsf{Ty}\,(\Gamma \triangleright A)) \to \mathsf{Tm}\,\Gamma\,(\uparrow\!\mathsf{Id}_A\,a\,a') \to \mathsf{Tm}\,\Gamma\,(P[\mathsf{id}, a]) \to \mathsf{Tm}\,\Gamma\,(P[\mathsf{id}, a'])\\
&|\mathsf{transp}\,P\,e\,u| && :\equiv \mathsf{coe}_P\,(\mathsf{R}_\Gamma\,\mathsf{id}, |e|)\,|u|
\end{aligned}
$$

The computation rule of transp only holds up to Id, but as described in [4], the model can be refined to support a definitional computation rule. Note that transport works with arbitrary Ty-motive, the motive does not have to be TyP (as opposed to the inductively defined Prop-valued identity type in Agda). Function extensionality holds by definition of the identity type. ◀

▶ **Construction 16.** *From an input model of type theory with $\top$, Σ, Π, we construct a model of type theory with $\top$, Σ, Π, a sort of propositions* TyP *closed under $\top$, Σ, Π and a* TyP*-valued identity type with a strong transport rule and function extensionality.*

Construction. We take the input model, equip it with TyP using Construction 9, then invoke Construction 15. ◀

The above construction can be extended with the empty type: if the input model has $\bot : \mathsf{Ty}$, the output model also supports $\bot : \mathsf{Ty}$ with its elimination rule, but we do not have $\bot : \mathsf{TyP}$ (unless $\mathsf{isExtPfProp}\,\bot$ in the input model). Similarly, to justify booleans in the output model, we need that the input model has booleans and a definitionally proof-irrelevant family over booleans that we can use to define identity for booleans:

$$
\begin{aligned}
&\mathsf{IdBool} && : \mathsf{Ty}\,(\Gamma \triangleright \mathsf{Bool} \triangleright \mathsf{Bool})\\
&\mathsf{Idtrue} && : \mathsf{Tm}\,\Gamma\,(\mathsf{IdBool}[\mathsf{id}, \mathsf{true}, \mathsf{true}])\\
&\mathsf{Idfalse} && : \mathsf{Tm}\,\Gamma\,(\mathsf{IdBool}[\mathsf{id}, \mathsf{false}, \mathsf{false}])\\
&\mathsf{Idirr} && : (e\,e' : \mathsf{Tm}\,(\Gamma \triangleright \mathsf{Bool} \triangleright \mathsf{Bool})\,\mathsf{IdBool}) \to e \equiv e'
\end{aligned}
$$

But then we might as well require TyP in the input model with closure under inductive types.

7 The setoid model internally

In the previous section we showed how a setoid model can be constructed without requiring a sort TyP in the input model. Can we redo the same internally to intensional type theory using point-free propositions? That is, can we define a setoid model in Agda (which can be viewed as the initial model of intensional type theory) without using strict propositions (Prop, TyP)?

Compared to Construction 15 of the previous section, the role of the input model is taken by our metatheory (Agda), the role of the output model is the model we construct. The equations of our model are given by the identity type of the metatheory. If all the equations can be proven by refl, it means that the model is strict. In such a case an external model construction can be obtained from the internal model (see [4, Section 3] for an exposition of model constructions vs. internal models through the example of the graph model). Model constructions are also called syntactic translations, see [7] for such a presentation.

The notion of model we construct is described in Figures 1, 2, 3 in extensional type theory. As some operations and equations typecheck only because of previous equations (e.g. $\mathsf{lam}[]$ depends on $\Pi[]$), the complete intensional description of the notion of model has many transports compared to this (see [5] for an exposition using explicit transports). However if an equation is proved by refl in the model, then transports over it disappear, so *concrete* strict models can be defined in Agda without using any transports.

External model constructions where the definitions of types (and substitutions and terms) don't involve equations can be internalised immediately as strict models. This is the case for the setoid model using TyP, see [4]. In our case however, there is an equation expressing that the equivalence relation is a proposition. This makes the construction more involved as we have to prove that the witnesses of propositionality are equal.

The answer to the above question is yes. This section was formalised in Agda [12].

▶ **Construction 17.** *We construct a model of type theory with $\bot$, $\top$, Σ, Π, a sort of propositions* TyP *closed under $\top$, Σ, Π, a* TyP*-valued identity type with a strong transport rule and function extensionality. All equations of our model hold definitionally, with the exceptions* irr, $\Sigma[]$, $,[]$, $\Pi\eta$, $\Pi[]$, $\mathsf{lam}[]$.

Construction. We explain the main components, for details consult the formalisation.

We define contexts as setoids where the equivalence relation is a point-free proposition. Compare it with how contexts were defined in the external Construction 15.

$$\begin{aligned}
\mathsf{Con} :\equiv \quad & (|\Gamma| : \mathsf{Type}) \\
\times\, & (\Gamma^{\sim} : |\Gamma| \times |\Gamma| \to \mathsf{Type}) \\
\times\, & (\Gamma^{\mathsf{p}} \;: \mathsf{isPfPropd}\, \Gamma^{\sim}) \\
\times\, & (\mathsf{R}_\Gamma : (\gamma_x : |\Gamma|) \to \Gamma^{\sim}(\gamma_x, \gamma_x)) \\
\times\, & (\mathsf{S}_\Gamma : \Gamma^{\sim}(\gamma_0, \gamma_1) \to \Gamma^{\sim}(\gamma_1, \gamma_0)) \\
\times\, & (\mathsf{T}_\Gamma : \Gamma^{\sim}(\gamma_0, \gamma_1) \to \Gamma^{\sim}(\gamma_1, \gamma_2) \to \Gamma^{\sim}(\gamma_0, \gamma_2))
\end{aligned}$$

We don't have equations on contexts, so it is not an issue that there is an equation (Γ^{p}) as one of the components. There will be an issue for types, see below.

Substitutions are functions that respect the relations.

$$\mathsf{Sub}\,\Delta\,\Gamma :\equiv (|\gamma| : |\Delta| \to |\Gamma|) \times (\gamma^{\sim} : \Delta^{\sim}(\delta_0, \delta_1) \to \Gamma^{\sim}(|\gamma|\,\delta_0, |\gamma|\,\delta_1))$$

They form a category with function composition (for both $|{-}|$ and $-^{\sim}$ components) and the identity function. The categorical laws are definitional. The empty context is given by $\top$ with the constant $\top$ relation.

Types are displayed setoids with coercion and coherence (note that later we will replace types by their strictified variants).

$$
\begin{aligned}
&\mathsf{Ty}\,\Gamma :\equiv\\
&\quad (|A| \quad : |\Gamma| \to \mathsf{Type})\\
&\times (A^{\sim} \quad : (\gamma_0 : |\Gamma|) \times (\gamma_1 : |\Gamma|) \times \Gamma^{\sim}\,\gamma_0\,\gamma_1 \times |A|\,\gamma_0 \times |A|\,\gamma_1 \to \mathsf{Type})\\
&\times (A^{\mathsf{p}} \quad : \mathsf{isPfPropd}\,A^{\sim})\\
&\times (\mathsf{R}_A \quad : (a_x : |A|\,\gamma_x) \to A^{\sim}\,(\gamma_x, \gamma_x, \mathsf{R}_\Gamma\,\gamma_x, a_x, a_x))\\
&\times (\mathsf{S}_A \quad : A^{\sim}\,(\gamma_0, \gamma_1, \gamma_{01}, a_0, a_1) \to A^{\sim}\,(\gamma_1, \gamma_0, \mathsf{S}_\Gamma\,\gamma_{01}, a_1, a_0))\\
&\times (\mathsf{T}_A \quad : A^{\sim}\,(\gamma_0, \gamma_1, \gamma_{01}, a_0, a_1) \to A^{\sim}\,(\gamma_1, \gamma_2, \gamma_{12}, a_1, a_2) \to A^{\sim}\,(\gamma_0, \gamma_2, \mathsf{T}_\Gamma\,\gamma_{01}\,\gamma_{12}, a_0, a_2))\\
&\times (\mathsf{coe}_A : \Gamma^{\sim}\,(\gamma_0, \gamma_1) \to |A|\,\gamma_0 \to |A|\,\gamma_1)\\
&\times (\mathsf{coh}_A : (\gamma_{01} : \Gamma^{\sim}\,(\gamma_0, \gamma_1))(a_0 : |A|\,\gamma_0) \to A^{\sim}\,(\gamma_0, \gamma_1, \gamma_{01}, a_0, \mathsf{coe}_A\,\gamma_{01}\,a_0))
\end{aligned}
$$

Compared to the external version, we don't need substitution laws ($A^{\sim}[]$ and $\mathsf{coe}_A[]$) and instead of making the relation Prop-valued we add an element of the identity type saying that $A^{\sim}$ is a point-free proposition. We can prove that two types are equal if their $|{-}|$, $-^{\sim}$, $-^{\mathsf{p}}$, coe components are equal. The other components will be equal by $-^{\mathsf{p}}$. Unfortunately, due to Proposition 13 part (ii) (b), we have to show that the proofs of propositionalities $-^{\mathsf{p}}$ coincide.

Substitution of types is given by function composition for the $|{-}|$ and $-^{\sim}$ components, for the $-^{\mathsf{p}}$ component we use the fact that dependent point-free propositions are closed under reindexing. The reflexivity, symmetry and transitivity components of $A[\gamma]$ are constructed using transport and the corresponding components of A. The exact way they are constructed does not matter as they are proof irrelevant by A^{p}. We prove the substitution laws $[\circ]$ and $[\mathsf{id}]$ up to the identity type using J.

Terms are like substitutions, but with dependent functions.

$$\mathsf{Tm}\,\Gamma\,A :\equiv (|t| : (\gamma_x : |\Gamma|) \to |A|\,\gamma_x) \times (t^{\sim} : (\gamma_{01} : \Gamma^{\sim}\,(\gamma_0, \gamma_1)) \to A^{\sim}\,(\gamma_0, \gamma_1, \gamma_{01}, |t|\,\gamma_0, , |t|\,\gamma_1))$$

The $|{-}|$ and $-^{\sim}$ components of context extension are given by Σ types, the propositionality component is using the fact that point-free propositions are closed under Σ.

Analogously to the model in the previous section, we can show that we have $\bot$, $\top$, Σ and Π types. The β rules are definitional for both Σ and Π, however for Π the η rule only holds up to the metatheoretic identity type. The reason is that $|\Pi\,A\,B|$ is defined as a Σ type consisting of a function from $|A|$ to $|B|$ and a proof that it respects the relation.

$$
\begin{aligned}
|\Pi\,A\,B|\,\gamma_x :\equiv \big(f : (a_x : |A|\,\gamma_x) \to |B|\,(\gamma_x, a_x)\big) \times\\
(a_{01} : A^{\sim}\,(\gamma_x, \gamma_x, \mathsf{R}_\Gamma\,\gamma_x, a_0, a_1)) \to B^{\sim}\,((\gamma_x, a_0), (\gamma_x, a_1), (\mathsf{R}_\Gamma\,\gamma_x, a_{01}), f\,a_0, f\,a_1)
\end{aligned}
$$

Two functions are related by $(\Pi\,A\,B)^{\sim}$ if they map related inputs to related outputs. Hence there are two (definitionally) different ways of proving that a $t : \mathsf{Tm}\,\Gamma\,(\Pi\,A\,B)$ respects a (homogeneous) relation $a_{01} : A^{\sim}\,(\gamma_x, \gamma_x, \mathsf{R}_\Gamma\,\gamma_x, a_0, a_1)$. One is $\pi_2\,(|t|\,\gamma_x)\,a_{01}$, the other is $t^{\sim}\,(\mathsf{R}_\Gamma\,\gamma_x)\,a_{01}$. Because $B^{\sim}$ is a proposition, these are equal, but only up to the identity type. And the eta rule computes to the usage of the two different versions on the two sides of the equation. We do not prove the substitution laws $\bot[]$, $\top[]$, $\Sigma[]$, $,[]$, $\Pi[]$, $\mathsf{lam}[]$ yet. There is no need to worry, we will prove them after replacing Ty with its strictified variant.

If an equation is not definitional and there are later components in the model that depend on it (as $\mathsf{lam}[]$ depends on $\Pi[]$), it makes the model construction extremely tedious. The situation one ends up in is also known as "transport hell". As the functor laws $[\circ]$, $[\mathsf{id}]$ for types and terms are not definitional, almost every operation that mentions substitutions involves transports. Instead of fighting in transport hell and proving the transported versions of the laws $\bot[]$, $\dots$, $\mathsf{lam}[]$, we follow the local universes approach [20]. We wrap Ty into Ty' which contains a base context, a substitution into this context and a Ty in this base context.

$$\mathsf{Ty}'\,\Gamma :\equiv (\mathsf{con}_A : \mathsf{Con}) \times (\mathsf{sub}_A : \mathsf{Sub}\,\Gamma\,\mathsf{con}_A) \times (\mathsf{ty}_A : \mathsf{Ty}\,\mathsf{con}_A)$$

Substitution for Ty' is defined as composition in the sub component, and as composition in the category is definitional, the laws $[\circ]$, $[\mathsf{id}]$ become definitional. Terms Tm' and context extension $- \rhd' -$ can be defined, and all the CwF equations are definitional. The type formers can be redefined as their primed versions $\bot'$, Σ' and Π'. $\bot'[]$ and $\top'[]$ hold definitionally, but $\Sigma'[]$ and $\Pi'[]$ rely on definitional β and η for Σ and Π (the ones defined for Ty), and we are missing an η for Π. Hence $\Sigma[]$, $,[]$, $\Pi\eta$, $\Pi[]$, $\mathsf{lam}[]$ only hold up to the identity type.

We define $\mathsf{TyP}\,\Gamma$ as those families over $|\Gamma|$ that are (point-free) propositional and which have coercion.

$$\mathsf{TyP}\,\Gamma :\equiv (|A| : |\Gamma| \to \mathsf{Type}) \times (A^{\mathsf{p}} : \mathsf{isPfPropd}\,|A|) \times (\mathsf{coe}_A : \Gamma^{\sim}\,(\gamma_0, \gamma_1) \to |A|\,\gamma_0 \to |A|\,\gamma_1)$$

$\uparrow$ is given by letting the relation be constant $\top$, and showing closure under $\top$, Σ and Π is straightforward. Proof irrelevance irr comes from the assumed equation A^{p}, hence it is not definitional. Definition of the TyP-valued identity type is analogous to the construction in the previous section. Strictification of TyP is analogous to that of Ty. ◀

We conjecture that without strictification (the replacement of Ty by Ty') we can still prove all the equations, however this seems to be very difficult due to "transport hell".

8 Examples of strict algebraic structures

Point-free equations can be used to define strict variants of algebraic structures. For example, internally to a model of type theory with a universe Type closed under Π, Σ, Id, a strict monoid is defined as follows.

$$\begin{aligned}
&\mathsf{M} &&: \mathsf{Type}\\
&- \otimes - &&: \mathsf{M} \to \mathsf{M} \to \mathsf{M}\\
&\mathsf{ass} &&: \mathsf{Id}_{\mathsf{M}\to\mathsf{M}\to\mathsf{M}\to\mathsf{M}}\,(\lambda x\,y\,z.(x \otimes y) \otimes z)\,(\lambda x\,y\,z.x \otimes (y \otimes z))\\
&\mathsf{o} &&: \mathsf{M}\\
&\mathsf{idl} &&: \mathsf{Id}_{\mathsf{M}\to\mathsf{M}}\,(\lambda x.\mathsf{o} \otimes x)\,(\lambda x.x)\\
&\mathsf{idr} &&: \mathsf{Id}_{\mathsf{M}\to\mathsf{M}}\,(\lambda x.x \otimes \mathsf{o})\,(\lambda x.x)
\end{aligned}$$

Compare it with the usual definition of monoid where the laws are stated using universal quantification:

$$\begin{aligned}
&\mathsf{ass} : (x\,y\,z : \mathsf{M}) \to \mathsf{Id}_{\mathsf{M}}\,((x \otimes y) \otimes z)\,(x \otimes (y \otimes z))\\
&\mathsf{idl}\ \ : (x : \mathsf{M}) \to \mathsf{Id}_{\mathsf{M}}\,(\mathsf{o} \otimes x)\,x\\
&\mathsf{idr}\ \ : (x : \mathsf{M}) \to \mathsf{Id}_{\mathsf{M}}\,(x \otimes \mathsf{o})\,x
\end{aligned}$$

If our model has canonicity, then in the empty context, for any strict monoid, the laws hold definitionally. For example, booleans where conjunction is defined as $a \wedge b :\equiv \mathsf{if}\, a\, \mathsf{then}\, b\, \mathsf{else}\, \mathsf{false}$ do not form a strict monoid. We do have $\mathsf{idl} : \mathsf{true} \wedge b \equiv b$, but we don't have idr or associativity definitionally, only propositionally. So booleans with $- \wedge -$ form a usual monoid, but not a strict monoid. Similarly, natural numbers with addition form a usual monoid, but not a strict monoid.

In contrast, for any type A, the function space $A \to A$ forms a monoid with $f \otimes g :\equiv \lambda x.f\,(g\,x)$ and $\mathsf{o} :\equiv \lambda x.x$. We have associativity as $\lambda f\,g\,h.(f \otimes g) \otimes g \equiv \lambda f\,g\,h\,x.f\,(g\,(h\,x)) \equiv \lambda f\,g\,h.f \otimes (g \otimes h)$ and the identity laws hold as e.g. $\lambda f.\mathsf{o} \otimes f \equiv \lambda f\,x.f\,x \equiv \lambda f.f$.

Strict monoids are closed under finite products following the η rule for $\times$. We can define displayed strict monoids over a strict monoid, and dependent product of strict monoids. Strict monoids are also closed by A-ary products for any type A. That is, given a strict monoid with carrier M, $A \to M$ is also a strict monoid.

Point-free propositions are another strict algebraic structure with no operations and only one equation: any two elements are equal. Closure under $\top$ and Σ give closure under (dependent) finite products, closure under Π is the same as having A-ary (dependent) products for any type A.

We conjecture that for any (generalised) algebraic structure, we have a CwF with $\top$, Σ and extensional Id of strict algebras internally to any model of intensional type theory. The category part of the CwF is the category of algebras and homomorphisms, terms and types are displayed algebras and sections, context extension is dependent product of algebras, and so on. This semantics was called finite limit CwF in [17].

The term "strict" algebraic structure is only correct in intensional type theory. In a model with function extensionality, strict and usual monoids coincide.

There is a stronger sense in which algebraic structures can be "strict". Obviously, to define a strict monoid in the empty context, all laws have to hold definitionally. However when *assuming* a strict monoid and using it in a construction in this open context, the laws only hold up to propositional equality. It would be convenient to have implementations of type theory with strict algebraic structures in this stronger sense. Currently, Agda only supports one algebraic structure which is strict in this stronger sense: propositions.

9 Summary

In this paper we attempted to push the limits of what can be done in intensional type theory without function extensionality or uniqueness of identity proofs. We exploited the fact that in intensional type theory, in the empty context propositional and definitional equality coincide. We used this to define a dynamic universe of strict propositions internally. We expect that other strict algebraic structures with the expected properties can be defined along the same lines. In a strict algebraic structure, all equations are definitional. As we cannot assume definitional equalities in type theory, when we assume a member of a strict algebraic structure, the equations only hold propositionally. This makes it difficult to use such algebraic structures in practice. However we think that model constructions of type theory can be formalised as functions between strict models. We conjecture that the canonicity and normalisation displayed models from the corresponding proofs for type theory [10, 6] can be formalised in pure intensional type theory. These would be displayed over a strict model defined as a point-free algebraic structure. There are other inherent limitations of point-free propositions, e.g. the fact that we cannot prove that being a point-free proposition is a point-free proposition.

Internal strict models can be externalised directly. We would like to understand in which circumstances internal non-strict models can be externalised into model constructions. Another open problem is whether $\mathsf{isHProp}\,(\mathsf{isPfProp}\,A)$ is provable in intensional type theory.

A strict proposition-valued identity type with a strong transport rule was used to define presheaves [22] and a universe of setoids closed under dependent function space [3]. It is not clear whether such a type theory has normalisation [1]. Currently the only justification that we know for this strong transport rule is the setoid model construction. We showed that such an identity type can be derived in intensional type theory using point-free propositions. It seems that our construction is limited, the model we constructed does not include a universe of propositions or inductive types. In the future, we would like to circumscribe the exact conditions that the input model has to satisfy in order to obtain inductive types and universes from the setoid model construction.

References

1 Andreas Abel and Thierry Coquand. Failure of normalization in impredicative type theory with proof-irrelevant propositional equality. *Log. Methods Comput. Sci.*, 16(2), 2020. doi:10.23638/LMCS-16(2:14)2020.

2 Thorsten Altenkirch. Extensional equality in intensional type theory. In *14th Annual IEEE Symposium on Logic in Computer Science, Trento, Italy, July 2-5, 1999*, pages 412–420. IEEE Computer Society, 1999. doi:10.1109/LICS.1999.782636.

3 Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, Christian Sattler, and Filippo Sestini. Constructing a universe for the setoid model. In Stefan Kiefer and Christine Tasson, editors, *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings*, volume 12650 of *Lecture Notes in Computer Science*, pages 1–21. Springer, 2021. doi:10.1007/978-3-030-71995-1_1.

4 Thorsten Altenkirch, Simon Boulier, Ambrus Kaposi, and Nicolas Tabareau. Setoid type theory—a syntactic translation. In Graham Hutton, editor, *Mathematics of Program Construction*, pages 155–196, Cham, 2019. Springer International Publishing.

5 Thorsten Altenkirch and Ambrus Kaposi. Type theory in type theory using quotient inductive types. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 18–29, New York, NY, USA, 2016. ACM. doi:10.1145/2837614.2837638.

6 Thorsten Altenkirch and Ambrus Kaposi. Normalisation by Evaluation for Type Theory, in Type Theory. *Logical Methods in Computer Science*, Volume 13, Issue 4, October 2017. doi:10.23638/LMCS-13(4:1)2017.

7 Simon Boulier, Pierre-Marie Pédrot, and Nicolas Tabareau. The next 700 syntactical models of type theory. In *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs*, CPP 2017, pages 182–194, New York, NY, USA, 2017. ACM. doi:10.1145/3018610.3018620.

8 Simon Castellan, Pierre Clairambault, and Peter Dybjer. Categories with families: Unityped, simply typed, and dependently typed. *CoRR*, abs/1904.00827, 2019. arXiv:1904.00827.

9 Thierry Coquand. About the setoid model, 2013. URL: http://www.cse.chalmers.se/~coquand/setoid.pdf.

10 Thierry Coquand. Canonicity and normalization for dependent type theory. *Theor. Comput. Sci.*, 777:184–191, 2019. doi:10.1016/j.tcs.2019.01.015.

11 Thierry Coquand. Reduction free normalisation for a proof irrelevant type of propositions. *CoRR*, abs/2103.04287, 2021. arXiv:2103.04287.

12 István Donkó and Ambrus Kaposi. Agda formalization for the paper "Internal strict propositions using point-free equations". https://bitbucket.org/akaposi/prop, 2022.

13 Gaëtan Gilbert, Jesper Cockx, Matthieu Sozeau, and Nicolas Tabareau. Definitional proof-irrelevance without K. *Proc. ACM Program. Lang.*, 3(POPL):3:1–3:28, 2019. doi:10.1145/3290316.

14 Martin Hofmann. Conservativity of equality reflection over intensional type theory. In *TYPES 95*, pages 153–164, 1995.

15 Martin Hofmann. *Extensional constructs in intensional type theory.* CPHC/BCS distinguished dissertations. Springer, 1997.

16 Jasper Hugunin. Why not w? In Ugo de'Liguoro, Stefano Berardi, and Thorsten Altenkirch, editors, *26th International Conference on Types for Proofs and Programs, TYPES 2020, March 2-5, 2020, University of Turin, Italy*, volume 188 of *LIPIcs*, pages 8:1–8:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.TYPES.2020.8.

17 Ambrus Kaposi, András Kovács, and Thorsten Altenkirch. Constructing quotient inductive-inductive types. *Proc. ACM Program. Lang.*, 3(POPL):2:1–2:24, January 2019. doi:10.1145/3290315.

18 Ambrus Kaposi and Zongpu Xie. Quotient inductive-inductive types in the setoid model. In Henning Basold, editor, *27th International Conference on Types for Proofs and Programs, TYPES 2021*. Universiteit Leiden, 2021. URL: https://types21.liacs.nl/download/quotient-inductive-inductive-types-in-the-setoid-model/.

19 Nicolai Kraus and Jakob von Raumer. Coherence via well-foundedness: Taming set-quotients in homotopy type theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 662–675. ACM, 2020. doi:10.1145/3373718.3394800.

20 Peter Lefanu Lumsdaine and Michael A. Warren. The local universes model: An overlooked coherence construction for dependent type theories. *ACM Trans. Comput. Logic*, 16(3), July 2015. doi:10.1145/2754931.

21 Erik Palmgren. From type theory to setoids and back. *arXiv e-prints*, page arXiv:1909.01414, September 2019. arXiv:1909.01414.

22 Pierre-Marie Pédrot. Russian constructivism in a prefascist theory. In Holger Hermanns, Lijun Zhang, Naoki Kobayashi, and Dale Miller, editors, *LICS '20: 35th Annual ACM/IEEE Symposium on Logic in Computer Science, Saarbrücken, Germany, July 8-11, 2020*, pages 782–794. ACM, 2020. doi:10.1145/3373718.3394740.

23 The Univalent Foundations Program. Homotopy type theory: Univalent foundations of mathematics. Technical report, Institute for Advanced Study, 2013.