

Semi-simplicial Types in Logic-enriched Homotopy Type Theory

Fedor Part¹ and Zhaohui Luo^{*2}

- 1 Royal Holloway, University of London
Email: fedor.part@gmail.com
- 2 Royal Holloway, University of London
Email: zhaohui.luo@hotmail.co.uk

Abstract

The problem of defining Semi-Simplicial Types (SSTs) in Homotopy Type Theory (HoTT) has been recognized as important during the Year of Univalent Foundations at the Institute of Advanced Study [14]. According to the interpretation of HoTT in Quillen model categories [5], SSTs are type-theoretic versions of Reedy fibrant semi-simplicial objects in a model category and simplicial and semi-simplicial objects play a crucial role in many constructions in homotopy theory and higher category theory. Attempts to define SSTs in HoTT lead to some difficulties such as the need of infinitary assumptions which are beyond HoTT with only non-strict equality types.

Voevodsky proposed a definition of SSTs in Homotopy Type System (HTS) [26], an extension of HoTT with non-fibrant types, including an extensional strict equality type. However, HTS doesn't have the desirable computational properties such as decidability of type checking and strong normalization. In this paper, we study a logic-enriched homotopy type theory, an alternative extension of HoTT with equational logic based on the idea of logic-enriched type theories [1, 17]. In contrast to Voevodsky's HTS, all types in our system are fibrant and it can be implemented in existing proof assistants. We show how SSTs can be defined in our system and outline an implementation in the proof assistant Plastic [8].

1998 ACM Subject Classification F.4.1 Lambda calculus and related systems

Keywords and phrases Homotopy type theory, Semi-simplicial types, Logic-enriched type theory

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Homotopy Type Theory (HoTT) [25], an extension of Martin-Löf's intensional Type Theory (MLTT), lies in the center of the research area that explores the striking connections between homotopy theory and type theory in the study of Univalent Foundations of mathematics. Providing a direct language for formalization of homotopy theory, HoTT inherits constructivity and some computational properties of MLTT so that proof assistants like Coq [10] and Agda [3] can be used for proof verification and partial automatization. Sophisticated constructions from homotopy theory, whose complete formal description in set theory like ZF would be hopelessly cumbersome, can be expressed very concisely in HoTT, where the notion of a space is taken as primitive. For example, homotopy groups of spheres, fiber sequences, van Kampen theorem and many other things have been formalized in this way in HoTT [25].

* Partially supported by research grants from Royal Academy of Engineering and the CAS/SAFEA International Partnership Program for Creative Research Teams.



© F. Part and Z. Luo;
licensed under Creative Commons License CC-BY

Conference title on which this volume is based on.

Editors: Billy Editor and Bill Editors; pp. 1–14



Leibniz International Proceedings in Informatics

LIPIcs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Unfortunately, these developments are obstructed by a substantial problem. A type in HoTT is not only characterised by its elements, but by the whole structure of weak ∞ -groupoid, generated by the type of paths. Correspondingly, a function in HoTT is not just one on elements of types, but a proper ∞ -functor between ∞ -groupoids and, as a result, the types form a weak ∞ -category. However, ∞ -functors and ∞ -categories are meta-level notions for HoTT, whereas attempts to define them internally lead to the need to encode an infinite amount of coherence data, which is unclear how to do in HoTT. Only weak n -categories for concrete n can be defined so far in HoTT. As a consequence, the notion of homotopy coherent diagram of types, which is a ∞ -functor from homotopy coherent nerve of a 1-category to the ∞ -category of types, is also problematic. Therefore, no general notion of homotopy limit can be formulated in HoTT, although for freely generated diagrams it is possible [4].

A particular case of the problem is the internalisation of the notion of (semi-)simplicial objects in a type universe \mathcal{U} or, in short, (semi-)simplicial types. A simplicial type is the family of types $X_i : \mathcal{U}$ ($i \in \omega$), together with maps $d_n^i : X_{n+1} \rightarrow X_n$ and $s_n^i : X_n \rightarrow X_{n+1}$, where X_n should be thought of as the type of n -dimensional simplices, d_n^i assigns to a simplex $x : X_{n+1}$ the n -dimensional simplex, which should be thought of as i -th face of x , and s_n^i assigns to a simplex $x : X_n$ its degenerated version at the dimension $n + 1$. Formally, d_n^i and s_n^i are just some maps that satisfy certain equational conditions which, if expressed by means of the path-equality in HoTT, should be accompanied by coherence conditions for these paths and this cannot be expressed in HoTT. Among many other things, simplicial types may serve as a useful tool for talking about weak ∞ -categories in HoTT in terms of complete Segal spaces [24].

If one omits degeneracy maps in the definition of simplicial types, one would obtain Semi-Simplicial Types (SSTs). It simplifies the definition, but is still interesting as SSTs may be used, for example, to define complete semi-Segal spaces, which are ∞ -categories without identities. What seems to be missing in HoTT while trying to define SSTs inductively is some kind of proof-irrelevant or strict equality. Two definitions of SSTs have been proposed independently by Herbelin [12] and Voevodsky [27]. Both definitions rely on a notion of strict equality which, in the former case, is proof-irrelevant equality of the universe of propositions **Prop** in CIC¹ and, in the latter case, is the extensional equality of Voevodsky's Homotopy Type System (HTS) [27]. Unfortunately, CIC with the proof-irrelevant equality in **Prop** is known to be inconsistent with univalent universes. HTS extends HoTT with auxiliary types, which do not carry the structure of weak ∞ -groupoid and correspond to non-fibrant objects in Quillen model category. The major disadvantage of HTS is the lack of basic computational properties such as decidability of type checking or strong normalization, which makes it not implementable in any existing proof assistant.²

In this paper we exploit the ideas from Logic-enriched Type Theories (LTTs) [1, 17] to study an extension of HoTT, HoTT_{Eq}, where all types are fibrant and SSTs are definable.³ LTTs are type theories in which logical propositions and datatypes are completely separate. It is thus possible to introduce axioms or new rules of deduction without affecting the world of

¹ The Calculus of Inductive Constructions (CIC) is the type theory implemented in the Coq proof assistant. More accurately, the current Coq system [10] implements the predicative CIC (pCIC) where Set has become a predicative universe (we omit the details here).

² Please also note that decidability of type checking is a basic requirement for a type theory to be a reasonable logical system and one may regard this as not only a desirable but necessary property.

³ LTTs are related to the logic with dependent sorts [6], Cartmell's Generalized Algebraic Theories [9], Makkai's First Order Logic with Dependent Sorts [21] and Maietti's PhD thesis [20]. We omit a detailed analysis here.

datatypes. For instance, classical principles can be applied merely to logic without destroying the constructive nature of types (see, for example, [17, 2]). LTTs have been implemented in the Plastic proof assistant [8, 7] where formalization tasks based on LTTs have been done including, for instance, formalization of Weyl’s predicative foundation of mathematics [2].

We shall consider a strict equality in an LTT-setting whose datatype part is HoTT. Our system HoTTTq extends HoTT with two kinds of logical formulas, logical equality and universal quantification, and a new induction principle for \mathbf{N} , the type of natural numbers. The inductive definition of SSTs in HoTTTq may be related to Voevodsky’s definition in HTS as follows. Let $T = \sum_{x:\mathcal{U}} A(x)$, where $A(x)$ encodes some data over a type x . In HTS the type family sst_n of all n -truncated SSTs is defined by recursion on natural numbers $sst_rec: \mathbf{N} \rightarrow \sum_{t:T} P(t)$, where $sst_n := \pi_1(\pi_1(sst_rec(n)))$ and $P(t)$ is essentially a predicate, which expresses the functoriality of some maps in $\pi_2(\pi_1(t))$ by means of the strict equality. As $P(t)$ contains the strict equality, $\sum_{t:T} P(t)$ is non-fibrant. In HoTTTq such non-fibrant types are avoided by means of the new induction principle for natural numbers, which allows one to define a function $sst_rec: \mathbf{N} \rightarrow T$ and to prove that $\forall n : \mathbf{N}. P(sst_rec(n))$ simultaneously.

The plan of the paper is as follows. Section 2 contains some background information on SSTs and LTTs. HoTTTq is described in Section 3. Section 4 demonstrates how SSTs can be defined in HoTTTq and Section 5 outlines an implementation in Plastic.

2 Background: Semi-simplicial Types and Logic-enriched Type Theories

Originally, Martin-Löf’s intensional type theory (MLTT) was developed as a constructive foundation of mathematics with meaning explanations based on the notions of canonical object and computation [22, 23]. Surprisingly, the system admits a non-trivial interpretation in abstract homotopy theory, with types as abstract spaces, terms as continuous functions and identity types as spaces of paths. This unveils intensional type theories as a general syntactic framework for formal reasoning about constructions of homotopy theory. Formally, such a theory is modelled in a category with the Quillen model structure, allowing to talk about homotopical constructions internal to the category [5]. A Quillen model structure consists of three classes of maps - fibrations, cofibrations and weak equivalences, which satisfy certain axioms [13]. Examples of Quillen model categories include categories of topological spaces, groupoids and simplicial sets.

In particular, HoTT extends MLTT to reflect some important properties of the simplicial set model [15] and, among other things, adds the Univalence Axiom, which is a type-theoretic version of the existence of object classifier in an elementary ∞ -topos [19]. HoTT hence provides a direct language for formalization of homotopy theory and higher category theory [25] but, as mentioned in the Introduction, there is an obstacle in defining (semi-)simplicial types, a notion to be introduced below. We shall also review the LTT-framework whose ideas are used in this paper.

2.1 Semi-simplicial types

Let Δ denote the simplicial category with finite ordinals $Ob_\Delta = \{[n] | n \in \mathbf{N}\}$ as objects and with morphisms generated by two classes of monotonic maps: face maps $d_n^i: [n] \hookrightarrow [n+1]$, omitting i in the image, and degeneracy maps $s_n^i: [n+1] \rightarrow [n]$, merging i and $i+1$. A simplicial type is a simplicial object in a universe \mathcal{U} , that is a contravariant functor $X: \Delta^{op} \rightarrow \mathcal{U}$, corresponding to a sequence of types $X([0]): \mathcal{U}$, $X([1]): \mathcal{U}$, $X([2]): \mathcal{U}$, \dots ,

$X([n]) : \mathcal{U}, \dots$, together with maps $\bar{d}_n^i : X([n+1]) \rightarrow X([n])$ and $\bar{s}_n^i : X([n]) \rightarrow X([n+1])$, satisfying equational conditions as determined by functoriality. $X([0])$ is the type of points, $X([1])$ the type of segments, \dots , and $X([n])$ the type of n -simplices, and so on. The maps \bar{d}_n^i assign a face to $(n+1)$ -simplices by omitting the i -th vertex and the maps \bar{s}_n^i form the degenerated $(n+1)$ -simplices out of n -simplices by repeating the i -th vertex.

In the case of semi-simplicial types (SSTs), one only needs to define the notion of a functor from the semi-simplicial category, which is a subcategory of the simplicial category on face maps. In this paper we are mostly concerned with SSTs and shall use Δ to denote the semi-simplicial category, unless stated otherwise.

During the Year of Univalent Foundations at IAS [14] the following presentation of SSTs as a family of dependency structures of HoTT has been proposed. Consider the following boundary map:

$$\bar{d}_n := \bar{d}_n^0 \times \dots \times \bar{d}_n^n : X([n+1]) \rightarrow X([n]) \times \dots \times X([n])$$

that assigns to each $(n+1)$ -simplex a tuple of its n -faces. Then $X([n+1])$ is equivalent to the total space of fibration:

$$X([n+1]) \simeq \sum_{x_0 : X([n]), \dots, x_n : X([n])} (\bar{d}_n)^{-1}(x_0, \dots, x_n)$$

where fiber $(\bar{d}_n)^{-1}(x_0, \dots, x_n)$ is the type of all $(n+1)$ -simplices with boundary (x_0, \dots, x_n) and is non-empty iff (x_0, \dots, x_n) forms a valid boundary. In the case that there's a type of all valid boundaries bnd_n , $X([n+1])$ can be defined by means of the dependent type of fillings of boundaries $Y_n : \text{bnd}_n \rightarrow \mathcal{U}$:

$$X([n+1]) := \sum_{x : \text{bnd}_n} Y_n(x)$$

Thus an SST is the following dependency structure (Y_n) :

$$\begin{aligned} Y_0 &: \mathcal{U} \\ Y_1 &: Y_0 \rightarrow Y_0 \rightarrow \mathcal{U} \\ Y_2 &: \prod_{a, b, c : Y_0} Y_1(a, b) \rightarrow Y_1(b, c) \rightarrow Y_1(a, c) \rightarrow \mathcal{U} \\ &\dots \end{aligned} \tag{1}$$

In this way one may define n -truncated SSTs in HoTT for any *concrete* n , but not for hypothesised n . This only constitutes a meta-level definition.

2.2 Logic-enriched type theories

The concept of an LTT, an extension of the notion of type theory, was proposed by Aczel and Gambino in their study of type-theoretic interpretations of constructive set theory [1]. It provided them with flexibility to consider logics distinct from the propositions-as-types logic. A type-theoretic framework, which formulates LTTs in a logical framework, has been proposed in [17] to support formal reasoning with different logical foundations.

An LTT is a dependent type theory such as MLTT extended with judgements of the following forms:

- $\Gamma \vdash P : \mathbf{Prop}$, asserting that P is a logical proposition under context Γ .

- $\Gamma \vdash p : \mathbf{Prf}(P)$, asserting that p is a proof of proposition P (we often omit the \mathbf{Prf} -operator.)⁴

In this paper, we shall employ the LTT-framework as studied in [17] where a logical framework is used to specify LTTs, which is implemented in the proof assistant Plastic [8, 7]. A logical framework is a meta-language for specifying type theories, which is itself a dependent type theory. The rules of this framework for LTTs can be found in [2] (particularly its Appendix A). Besides the kind **Type** of types, we also have the kind **Prop** whose objects are logical propositions.⁵ These kinds are governed by the following rules:

$$\frac{\Gamma \text{ valid}}{\Gamma \vdash \mathbf{Type} \text{ kind}} \quad \frac{\Gamma \vdash A : \mathbf{Type}}{\Gamma \vdash \mathbf{El}(A) \text{ kind}} \quad \frac{\Gamma \vdash A = B : \mathbf{Type}}{\Gamma \vdash \mathbf{El}(A) = \mathbf{El}(B)}$$

$$\frac{\Gamma \text{ valid}}{\Gamma \vdash \mathbf{Prop} \text{ kind}} \quad \frac{\Gamma \vdash P : \mathbf{Prop}}{\Gamma \vdash \mathbf{Prf}(P) \text{ kind}} \quad \frac{\Gamma \vdash P = Q : \mathbf{Prop}}{\Gamma \vdash \mathbf{Prf}(P) = \mathbf{Prf}(Q)}$$

The rules for valid context formation is as usual:

$$\frac{}{\langle \rangle \text{ valid}} \quad \frac{\Gamma \vdash K \text{ kind} \quad x \notin FV(\Gamma)}{\Gamma, x : K \text{ valid}}$$

Please note, however, since there are new kinds **Prop** and **Prf**(P), one can assume propositions and their proofs as well as types and their objects. There are also parametric kinds of the form $(x : K)K'$, which are Π -constructors at the level of kinds. They are also called Π -kinds, not to be confused with Π -types: the former is meta-level constructs while the latter is at the object level. Π -types can be introduced as follows:

$$\begin{aligned} \Pi : (A : \mathbf{Type}, B : (x : A)\mathbf{Type})\mathbf{Type} \\ \lambda : (A : \mathbf{Type}, B : (x : A)\mathbf{Type}, f : (x : A)B[x])\Pi[A, B] \\ \varepsilon_{\Pi} : (A : \mathbf{Type}, B : (x : A)\mathbf{Type}, C : (F : \Pi[A, B])\mathbf{Type}, \\ f : (g : (x : A)B[x])C[\lambda[A, B, g]], z : \Pi[A, B])C[z] \end{aligned}$$

plus a computation rule which we omit here.

The nice property of LTTs is that the introduction of new logical axioms or logical rules doesn't affect world of types. This has given birth to an interesting direction of research. For instance, traditionally, the propositions-as-types logic in dependent type theories cannot be made classical without introducing unwanted closed terms in the data-types such as that of natural numbers, which do not compute to any canonical objects. For example, in MLTT the law of Excluded Middle would generate an element of $\mathbf{N} + (\mathbf{N} \rightarrow \mathbf{0})$, which doesn't compute to any canonical element of the type. As LTTs are free of such a problem, they are better suited for classical reasoning with type theory. Work on the subject includes [17, 2] and the Plastic proof assistant has been developed to support LTTs [8, 7] and it was used to give computer checked formalization of Weyl's classical predicative mathematics [2].

In this work we investigate another interesting application of the LTT-framework: resolving the coherence problem in HoTT. For this purpose we slightly diverge from what has been understood as an LTT in [1] or [17] in the following sense.

⁴ It is possible to consider a system without proof terms by only considering judgements of the form $\Gamma \vdash \phi_1, \dots, \phi_n \Rightarrow \phi \text{ true}$, as in [1]. Here we consider proof terms and they do not make any essential difference in most of the cases (and definitely not in this paper).

⁵ Note that, different from Coq, **Prop** is not a type and, hence, we do not automatically have higher-order logic in LTTs.

- We form HoTTEq by adding a logical equality and universal quantification, but it doesn't have full first-order logic of LTTs. In another words, in HoTTEq, only equational reasoning is considered and, in a sense, it is closely related to generalized algebraic theories.
- Logically equal terms in HoTTEq are interchangeable in both propositions and types, whereas in LTTs they are interchangeable only in propositions.
- Induction principles of HoTTEq are more sophisticated than those in LTTs. In LTTs an inductive type is characterised by an elimination principle and logical axioms of induction. These are two separate mechanisms for constructing functions and proving propositions. In HoTTEq these mechanisms are merged into one.

However, having said the above, the key idea of LTTs that propositions and datatypes are separated plays a key role in the work to be described below.

3 HoTTEq: HoTT Enriched with Logical Equality

As in the LTT-framework (see §2.2), HoTTEq extends HoTT with two forms of judgements: $\Gamma \vdash P : \mathbf{Prop}$ and $\Gamma \vdash p : P$. Contexts of HoTTEq can contain both type/object variables and proposition/proof variables. We shall follow the rules for LTT₁ [17]. In LTT₁, however, every context can be split: Γ is equivalent to $(\Gamma_{\mathbf{Type}}, \Gamma_{\mathbf{Prop}})$, where $\Gamma_{\mathbf{Type}}$ contains type/object variables and $\Gamma_{\mathbf{Prop}}$ contains proposition/proof variables. This is because that, in traditional LTTs, propositions or their proofs do not occur in types or their objects. For a type theory like LTT₁, we have: $\Gamma_{\mathbf{Type}}, \Gamma_{\mathbf{Prop}} \vdash J_{\mathbf{Type}}$ if, and only if, $\Gamma_{\mathbf{Type}} \vdash J_{\mathbf{Type}}$, where $J_{\mathbf{Type}}$ is either $A : \mathbf{Type}$ or $a : A$. In contrast, in HoTTEq one can construct more objects with assumptions of the logical equality between terms. This property will be crucial for the definition of SSTs.

We shall also have the rule for proof irrelevance:

$$\frac{\Gamma \vdash P : \mathbf{Prop} \quad \Gamma \vdash p : P \quad \Gamma \vdash q : P}{\Gamma \vdash p \equiv q : P} \quad (\text{PI})$$

which states that any two proofs of a proposition are definitionally equal.

3.1 Logical operators

HoTTEq contains only two logical operators: equality $x =_A y : \mathbf{Prop}$ and universal quantification $\forall x : \sigma. P(x) : \mathbf{Prop}$, where σ is either a type or a proposition. The rules for the logical equality are reminiscent of those of the Martin-Löf identity, where the J -like eliminator is duplicated for propositions and types. Additionally, the logical equality is defined to satisfy function extensionality by the rule (LEqFE). The rules are as follows:

$$\frac{\Gamma \vdash A : \mathbf{Type} \quad \Gamma \vdash a : A \quad \Gamma \vdash b : A}{\Gamma \vdash a =_A b : \mathbf{Prop}} \quad (\text{LEqForm})$$

$$\frac{\Gamma \vdash A : \mathbf{Type} \quad \Gamma \vdash a : A}{\Gamma \vdash \mathbf{r}_a^A : a =_A a} \quad (\text{LEqIntro})$$

$$\frac{\Gamma, x : A, y : A, p : x =_A y \vdash T[x, y, p] : \mathbf{Type} \quad \Gamma, x : A \vdash t(x) : T[x, x, \mathbf{r}]}{\Gamma, x : A, y : A, p : x =_A y \vdash \mathcal{E}_{\equiv}^T(x, y, p, t) : T[x, y, p]} \quad (\text{LEqElimT})$$

$$\frac{\Gamma, x : A, y : A, p : x =_A y \vdash P[x, y, p] : \mathbf{Prop} \quad \Gamma, x : A \vdash q(x) : P[x, x, \mathbf{r}]}{\Gamma, x : A, y : A, p : x =_A y \vdash \mathcal{E}_{\equiv}^P(x, y, p, q) : P[x, y, p]} \quad (\text{LEqElimP})$$

$$\frac{\Gamma, x : A, y : A, p : x =_A y \vdash T[x, y, p] : \mathbf{Type} \quad \Gamma, x : A \vdash t(x) : T[x, x, \mathbf{r}]}{\Gamma, x : A \vdash \text{Cmpt}(x) : \mathcal{E}_{=}^T(x, x, \mathbf{r}, t) =_{T[x, x, \mathbf{r}]} t(x)} \quad (\text{LEqComp})$$

$$\frac{\Gamma \vdash A : \mathbf{Type} \quad \Gamma, x : A \vdash B(x) : \mathbf{Type}}{\Gamma, f, g : F, p : (\forall x : A. f(x) =_{B(x)} g(x)) \vdash FE(f, g, p) : f =_F g} \quad (\text{LEqFE})$$

where $F \equiv \prod_{x:A} B(x)$.

And for the universal quantification (where σ is either **Type** or **Prop**):

$$\frac{\Gamma \vdash X : \sigma \quad \Gamma, x : X \vdash P(x) : \mathbf{Prop}}{\Gamma \vdash \forall x : X. P(x) : \mathbf{Prop}} \quad (\text{FAForm})$$

$$\frac{\Gamma \vdash X : \sigma \quad \Gamma, x : X \vdash p(x) : P(x)}{\Gamma \vdash I_V(p) : (\forall x : X. P(x))} \quad (\text{FAIntro})$$

$$\frac{\Gamma \vdash X : \sigma \quad \Gamma \vdash p : (\forall x : X. P(x))}{\Gamma, x : X \vdash \mathcal{E}_V(p, x) : P(x)} \quad (\text{FAElim})$$

Using the eliminators $\mathcal{E}_{=}^P$ and $\mathcal{E}_{=}^T$ and the universal quantification, one can construct the transitivity proof: for $p : a =_A b$ and $q : b =_A c$, the proof $q \cdot p : a =_A c$. Also, for $p : a =_A b$ and $(x : A)Y(x) : \sigma$, one can construct the substitution $\text{subst}_p^Y(y) : Y(b)$, where $y : Y(a)$.

The following straightforward lemmas assure that logical equality, defined in this way, behaves as expected. They will be required in Section 4.

► **Lemma 1.** *For any type $A : \mathbf{Type}$, any type family $T : (A) \mathbf{Type}$ over A , if $p : x =_A y$, $q : y =_A z$, then $\text{subst}_q^T \circ \text{subst}_p^T = \text{subst}_{q \cdot p}^T$.*

► **Lemma 2.** *For any family of functional types $T \equiv (x : A) \prod_{y:B} C(x, y)$, any object $a : A$, any function $f : \prod_{y:B} C(a, y)$ and any equality proof $p : a =_A b$, the following holds:*

$$\forall y : B. \text{subst}_p^T(f)(y) = \text{subst}_p^{C(\cdot, y)}(f(y)).$$

3.2 Induction

In LTTs, an inductive type is characterised by an elimination rule that specifies how one can construct elements of other types out of the objects of the inductive type, and an induction rule that specifies how propositions about objects of the inductive type can be proven. For example, for the type of natural numbers \mathbf{N} , it does not only have the familiar elimination rule but have the following induction rule:

$$\frac{\Gamma, n : \mathbf{N} \vdash P_n : \mathbf{Prop} \quad \Gamma \vdash b : P_0 \quad \Gamma, n : \mathbf{N}, p : P_n \vdash ih : P_{n+1}}{\Gamma, n : \mathbf{N} \vdash \text{Ind}_{\mathbf{N}}(b, ih, n) : P_n}$$

This is fine as long as proof terms do not occur in types or their objects. In HoTTTEq this property doesn't hold and the above form of induction becomes insufficient. This happens because, if inductive hypothesis has the form:

$$\begin{aligned} n : N, t : T_n, p : P_n(t) &\vdash ih_T(t, p) : T_{n+1} \\ n : N, t : T_n, p : P_n(t) &\vdash ih_P(t, p) : P_{n+1}(ih_T(t, p)) \end{aligned}$$

where T_n are types and $P_n(t)$ are propositions, then the elimination rule for \mathbf{N} cannot be applied to obtain a function of type $\prod_{n:\mathbf{N}} T_n$. However note, that if propositions were types, this inductive hypothesis could have been rewritten in terms of type family $\sum_{t:T_n} P_n(t)$. Thus, instead, we introduce a mechanism that not only constructs a function from an inductive type but, simultaniously, proves a property of the function (the premises are the same):

$$\frac{\begin{array}{c} \Gamma \vdash b: T_0 \quad \Gamma \vdash pb: P_0(b) \\ \Gamma, n: N, t: T_n, p: P_n(t) \vdash ih_T(t, p): T_{n+1} \\ \Gamma, n: N, t: T_n, p: P_n(t) \vdash ih_P(t, p): P_{n+1}(ih_T(t, p)) \end{array}}{\Gamma, n: N \vdash \mathcal{E}_N^T(b, pb, ih_T, ih_P, n): T_n}$$

and

$$\frac{\begin{array}{c} \Gamma \vdash b: T_0 \quad \Gamma \vdash pb: P_0(b) \\ \Gamma, n: N, t: T_n, p: P_n(t) \vdash ih_T(t, p): T_{n+1} \\ \Gamma, n: N, t: T_n, p: P_n(t) \vdash ih_P(t, p): P_{n+1}(ih_T(t, p)) \end{array}}{\Gamma, n: N \vdash \mathcal{E}_N^P(b, pb, ih_T, ih_P, n): P_n(\mathcal{E}_N^T(b, pb, ih_T, ih_P, n))}$$

with the following computation rules:

$$\Gamma \vdash \mathcal{E}_N(b, pb, ih_T, ih_P, 0) \equiv b$$

$$\Gamma, n: N \vdash \mathcal{E}_N(b, pb, ih_T, ih_P, n+1) \equiv ih_T(\mathcal{E}_N^T(b, pb, ih_T, ih_P, n), \mathcal{E}_N^P(b, pb, ih_T, ih_P, n))$$

4 Inductive Definition of Semi-Simplicial Types

We shall first give an outline on how to define SSTs in HoTTTEq and then describe the constructions in more details.

4.1 Outline

In this subsection we show how the inductive construction, as described by Voevodsky [27], of the tower of dependency structures of SSTs (see (1) at the end of §4) fails in HoTT, but can be done in HoTTTEq.

The inductive procedure constructs $(n+1)$ -truncated SSTs out of an n -truncated ones. Explicitly, types sst_n of all n -truncated SSTs are defined inductively by assigning a type of $(n+1)$ -simplices to every valid n -dimensional boundary of a $(n+1)$ -simplex:

$$sst_{n+1} \equiv \sum_{x:sst_n} bnd_n^{m+1}(x) \rightarrow \mathcal{U}$$

where $bnd_n^m(x)$ is the type of n -boundaries of a m -simplex in x . The type $bnd_n^m(x)$ is defined inductively by representing $bnd_{n+1}^m(x)$ as the type of pairs (y, aug) , where $y: bnd_n^m(x)$ and $aug: \prod_{f:[n+1] \rightarrow [m]} Fill(rest_n^{f,x}(y))$ is an augmentation of n -boundary y , obtained by choosing

a $(n+1)$ -simplex for every restriction of y to $(n+1)$ -dimensional subsimplex f of a m -dimensional simplex. Here restriction maps $rest_n^{f,x}(y)$ can only be defined inductively in the assumption of functoriality of them. Then functoriality can be defined inductively only with further coherence assumptions and so on.

We avoid this obstacle in HoTTEq by using the induction principle for natural numbers as given in §3.2, which makes it possible to prove functoriality mutually with the type construction. In §1 P denotes this functoriality predicate.

The rest of the section is devoted to the detailed exposition of how to construct the type family sst_n .

4.2 Face maps

We define combinatorics of semi-simplicial category in type theory by the type $\Delta(i, j) : \mathcal{U}_0$ of all increasing functions between standard intervals $Stn(i)$ and $Stn(j)$, namely:

$$\Delta(i, j) := \sum_{f: Stn(i) \rightarrow Stn(j)} is_incr_T(f)$$

where:

$$\begin{aligned} Stn(i) &:= \sum_{n:N} leq(n, i) \\ is_incr_T(f) &:= \prod_{n, m: Stn(i)} ls(n, m) \rightarrow ls(f(n), f(m)) \end{aligned}$$

Composition $- \circ_{\Delta} -$ of two face maps $(f, p_f) : \Delta(i, j)$ and $(g, p_g) : \Delta(j, k)$ is defined as follows:

$$\begin{aligned} (g, p_g) \circ_{\Delta} (f, p_f) &:= \\ (g \circ f, \lambda n, m : Stn(i). p_g(f(n), f(m)) \circ p_f(n, m)) \end{aligned}$$

Note that, by the η -rule, the associativity of composition holds *definitionally*.

4.3 Inductive construction

We use induction for natural numbers HoTTEq to define simultaneously the data and a proof of a predicate on this data. Data construction is as follows:

- Type family $sst_n : \mathcal{U}_1$ of all n -truncated SST, such that sst_{n+1} computes to $\sum_{x: sst_n} bnd_n^{n+1}(x) \rightarrow \mathcal{U}_0$ and $sst_0 \equiv \mathcal{U}_0$.
- Type family $bnd_n^m(x : sst_n) : \mathcal{U}_1$ of n -boundaries of m -simplex in x , such that:
 - $bnd_0^m(x : sst_0) \equiv Stn(m) \rightarrow x$.
 - A $(n+1)$ -boundary is a pair (y, aug) , consisting of n -boundary y and its augmentation aug . Precisely, $bnd_{n+1}^m(x, Fill)$, where $x : sst_n$, $Fill : bnd_n^{n+1}(x) \rightarrow \mathcal{U}_0$, computes to:

$$\sum_{y: bnd_n^m(x)} \prod_{f: [n+1] \rightarrow [m]} Fill(rest_n^{f,x}(y))$$

- Family of maps $rest_n^{f,x: sst_n}(y) : bnd_n^k(x)$, where $y : bnd_n^m(x)$, restricting n -boundary of m -simplex y to the n -subboundary of face $f : \Delta(k, m)$ of m -simplex, such that:
 - $rest_0^{f,x: sst_0}(y) \equiv y \circ \pi_1(f)$

- Inductive step here deserves a comment as this is the point, where functoriality is required. At this stage the value of the following expression should be specified, where (y, aug) is unfolded presentation of $(n+1)$ -boundary $bnd_{n+1}^m(x)$:

$$rest_{n+1}^{f:\Delta(k,m),x}(y, aug)$$

This can be done by taking n -boundary $rest_n^{f,x}(y): bnd_n^k(x)$ together with some augmentation of it, which should be of type $\prod_{h:\Delta(n+1,k)} Fill(rest_n^{h,x}(rest_n^{f,x}(y)))$. But if

we try to use given augmentation $\lambda h. aug(f \circ h)$, we would obtain a term of type $\prod_{h:\Delta(n+1,k)} Fill(rest_n^{f \circ h,x}(y))$. Thus we require, that $rest_n$ satisfy the following functoriality condition:

$$\begin{aligned} func_n: \forall k, l, m : \mathbf{N}. \forall f : \Delta(k, l), g : \Delta(l, m). \\ \forall y : bnd_n^m(x). rest_n^{g \circ f, x}(y) = rest_n^{f, x}(rest_n^{g, x}(y)) \end{aligned}$$

The correct augmentation can be obtained by using substitution:

$$\lambda h. subst_{func_n^{h,f}}^{Fill}(aug(f \circ h))$$

To complete the inductive definition, it remains to prove functoriality for $rest_0$ and for $rest_{n+1}$, provided it holds for $rest_n$. Assume that $f : \Delta(k, l)$ and $g : \Delta(l, m)$. The base case is straightforward:

$$\begin{aligned} rest_0^{g \circ f, x: sst_0}(y) &\equiv y \circ \pi_1(g \circ f) \equiv y \circ (\pi_1(g) \circ \pi_1(f)) \equiv \\ &\equiv (y \circ \pi_1(g)) \circ \pi_1(f) \equiv rest_0^{f, x} \circ rest_0^{g, x}(y) \end{aligned}$$

and thus $func_0^{f,g}(y)$ is just reflexivity.

Using $func_n$, we now construct the proof $func_{n+1}^{f,g}(y): (rest_{n+1}^{g \circ f, x}(y) = rest_{n+1}^{f, x} \circ rest_{n+1}^{g, x}(y))$. By definition $rest_{n+1}^{g \circ f, x}(y)$ and $rest_{n+1}^{f, x} \circ rest_{n+1}^{g, x}(y)$, where $y : bnd_{n+1}^m(x)$, compute to the following pairs:

$$rest_{n+1}^{g \circ f, x}(y, aug) \equiv \left(rest_n^{g \circ f, x}(y), \lambda h : \Delta(n+1, k). subst_{func_n^{h, g \circ f}}^{Fill}(aug((g \circ f) \circ h)) \right)$$

$$\begin{aligned} rest_{n+1}^{f, x} \circ rest_{n+1}^{g, x}(y) &\equiv rest_{n+1}^{f, x} \left(rest_n^{g, x}(y), \lambda h : \Delta(n+1, l). subst_{func_n^{h, g}}^{Fill}(aug(g \circ h)) \right) \equiv \\ &\equiv \left(rest_n^{f, x}(rest_n^{g, x}(y)), \lambda h : \Delta(n+1, k). subst_{func_n^{h, f}}^{Fill}(subst_{func_n^{g, f \circ h}}^{Fill}(aug(g \circ (f \circ h)))) \right) \end{aligned}$$

Each of these pairs is a point in a fiber of the family $\Pi F := (y : bnd_n^k(x)) \prod_{h:\Delta(n+1,k)} Fill(rest_n^{h,x}(y))$.

By inductive hypothesis, there is equality $func_n^{f,g}(y)$ between first components. Denote aug_1 and aug_2 the second components of the first and the second pair respectively. It remains to prove, that $subst_{func_n^{f,g}}^{\Pi F}(aug_1) = aug_2$. By Lemma 2 and (LEqFE) rule this equality is equivalent to $(h : \Delta(n+1, k)) subst_{func_n^{f,g}}^{Fill}(aug_1 h) = aug_2 h$. $aug(g \circ (f \circ h))$ computationally equals to $aug((g \circ f) \circ h)$ and by Lemma 1 $subst_{func_n^{h, f}}^{Fill} \circ subst_{func_n^{g, f \circ h}}^{Fill} = subst_{func_n^{h, f} \cdot func_n^{g, f \circ h}}^{Fill}$. Thus $subst_{func_n^{f,g}}^{Fill}(aug_1 h)$ and $aug_2 h$ are two substitutions of the same term, therefore by (PI) they are equal.

5 Implementation in Plastic

Plastic is a light-weight proof assistant, developed by Paul Callaghan in 1999 [7]. It's underlying system is Luo's Logical Framework [16] and it's syntax is very similar to that of Lego [18]. Plastic is the only proof assistant that supports LTTs and is flexible enough for the developments, described in this paper.

In this section we outline the key aspects of the implementation of HoTTEq and SSTs in Plastic. The source code of Plastic together with this implementation is available at github [11] (the SST implementation is located at the subdirectory `lib/Univalence/SimplicialTypes`).

5.1 HoTTEq: Implementation in Plastic

Rules of a type theory are expressed in Plastic by means of LF. For example, rules (LEqForm), (LEqIntro), (LEqElimT) and (LEqElimP) for logical equality are expressed as the following code in Plastic (here $A \rightarrow B$ is the abbreviation for the kind $(_ : A)B$):

```
[Eq : (A : Type) A -> A -> Prop];
[EqR : (A : Type) (a : A) Eq A a a];
[EqE_T : (A : Type) (T : (x:A)(y:A)(_:Eq ? x y)Type)
  (_(x:A)T x x (EqR ? x))(x:A)(y:A)(p:Eq ? x y) T x y p];
[EqE_P : (A : Type) (P : (x:A)(y:A)(_:Eq ? x y)Prop)
  (_(x:A)P x x (EqR ? x))(x:A)(y:A)(p:Eq ? x y) P x y p];
```

Plastic supports standard pattern of inductive definitions. For example, the type of paths is defined as follows:

```
Inductive [A:Type][Id_ : (x:El A)(y:El A)Type]
Constructors
  [IdR : (z:El A)Id_ z z];
```

One of key features of Plastic is the mechanism to specify computation rules for customised eliminators of inductive types. This makes it possible to implement induction of HoTTEq (and even higher inductive types). Specialised computation rules can be defined by the command **SimpleElimRule**. This gives access to the underlying mechanism of Plastic, used for inductive types and universes. Basically, the term, which is to be used as the combinator for elimination operator and constructor arguments, is specified in this way. The general syntax is as follows:

```
SimpleElimRule TYPE_NAME ELIM_NAME ELIM_ARITY
  [CONSTR_1 CONSTR_1_ARITY = TERM : TYPE]
  ...
  [CONSTR_n CONSTR_n_ARITY = TERM : TYPE];
```

For example, this command allows one to define the recursion principle for a circle, so that it computes on the point constructor:

```
[Circle : Type];
[base : Circle];
[loop : Id_ ? base base];
[CircRec : (A:Type)(a:A)(l:Id_ ? a a)Circle->A];

SimpleElimRule Circle CircRec 4
  [base 0 = [A:Type][a:A][l:Id_ A a a] a :
    (A:Type)(a:A)(_:Id_ A a a)A];
```

Because there's no way to implement the strong proof-irrelevance (PI) in the current version of Plastic, we replace it with the following rule:

```
[PI : (P : Prop) (A : Type) (t : P -> A) (p , q : P)
      Eq ? (t p) (t q)];
```

5.2 Semi-simplicial types

The definition of SSTs by means of an inductive procedure, as described in §4, is implemented as two maps: $t : T, p : P(t) \vdash ih_T(t, p) : T$ and $t : T, p : P(t) \vdash ih_P(t, p) : P(ih_T(t, p))$, where:

$$T := \sum_{SST : \mathcal{U}_1} \sum_{bnd : SST \rightarrow \mathbf{N} \rightarrow \mathcal{U}_1} \prod_{x : SST, k, m : \mathbf{N}, f : \Delta(k, m)} bnd^m(x) \rightarrow bnd^k(x)$$

$$P((SST, bnd, rest) : T) := \forall x : SST. \forall k, l, m : \mathbf{N}. \forall f : \Delta(k, l), g : \Delta(l, m). \\ \forall y : bnd^m(x). rest^{g \circ f, x}(y) = rest^{f, x}(rest^{g, x}(y))$$

Then, desired functions $sst_rec_T : \mathbf{N} \rightarrow T$ and $sst_rec_P(n : \mathbf{N}) : P(sst_rec_T(n))$, such that $sst_rec_T(n+1) \equiv ih_T(sst_rec_T(n), sst_rec_P(n))$, are obtained by applying induction principle of Section 3. With proof scripts omitted as they are too cumbersome (see [11]) the code is as follows:

```
[T = Sigma ? ([SST : Type^1] Sigma ? ([bnd : (T^1 SST) ==> (Nat ==> Type^1)]
    Pi3 ? ? ? ([x : El (T^1 SST)] [k : Nat] [m : Nat]
    Pi ? ([f : El (Delta k m)]
    T^1 (ap2_ ? ? ? bnd x m) ==> T^1 (ap2_ ? ? ? bnd x k)
    )))) ];

[P = [t : T] All_T ? ([x : (SST_pr t)]
    AllNat3 ([k, l, m : Nat]
    All_T ? ([f : El (Delta k l)] All_T ? ([g : El (Delta l m)]
    All_T ? ([y : (T^1 (bnd_pr t x m))]
    Eq ? (rest_pr t x k m (fm_comp k l m f g) y)
    (rest_pr t x k l f (rest_pr t x l m g y))))))]);

Claim ih_T : (n : Nat)(t : T)(p : (P t))T;
...

Claim ih_P : (n : Nat)(t : T)(p : (P t))P (ih_T n t p);
...

Claim t0 : T;
...

Claim p0 : P t0;
...]
```

And the type of all n -truncated SSTs is obtained by application of the data component **IE_NatT** of induction for natural numbers:

```
pi1 ? ? (IE_NatT T P t0 p0 ih_T ih_P n);
```

6 Conclusion

The system HoTTEq and the successful definition of SSTs in HoTTEq, presented in this paper, is just one example of how HoTT can benefit from a logic enrichment. In general, strict **Prop** of logic-enriched HoTT allows to talk about subsets $(\Gamma_{\mathbf{Type}}, \Gamma_{\mathbf{Prop}})$ of the fibrant object, which represents a context of types $\Gamma_{\mathbf{Type}}$. In further work we plan to explore in more detail a categorical semantics of the logic-enriched HoTT.

Definition of SSTs in HoTTEq clears up also many interesting directions for a further formalisation work.

Extending this definition to the definition of full simplicial types seems to us to be possible, although it requires some technical details to be worked out.

Another possibility is to explore, what can be already done by means of SSTs. For example, straightforward definitins can be done for natural transformations between SSTs and complete semi-Segal types, which reflect the structure of weak ∞ -category without identities. In further work we plan to figure out, how the globular structure of the type of paths on a universe can be realized as a complete semi-Segal type.

References

- 1 Peter Aczel and Nicola Gambino. The generalised type-theoretic interpretation of constructive set theory. *J. Symb. Log.*, 71(1):67–103, 2006.
- 2 R. Adams and Z. Luo. Weyl’s predicative classical mathematics as a logic-enriched type theory. *ACM Trans. on Computational Logic*, 11(2), 2010.
- 3 The Agda proof assistant (version 2). Available from the web page: <http://appserv.cs.chalmers.se/users/ulfn/wiki/agda.php>, 2008.
- 4 J. Avigad, K. Kapulkin, and P.-L. Lumsdaine. Homotopy limits in Coq. *CoRR*, abs/1304.0680, 2013.
- 5 Steven Awodey. Type theory and homotopy. In *Epistemology versus Ontology - Essays on the Philosophy and Foundations of Mathematics in Honour of Per Martin-Löf*, pages 183–201. 2012.
- 6 Joao Belo. Dependently sorted logic. In *Proceedings of the 2007 international conference on Types for proofs and programs*, Springer- Verlag Berlin, Heidelberg, 2007.
- 7 P. Callaghan. Plastic proof assistant. <http://homepages.inf.ed.ac.uk/wadler/realworld/plastic.html>, 1999.
- 8 P. Callaghan and Z. Luo. An implementation of LF with coercive subtyping and universes. *Journal of Automated Reasoning*, 27(1):3–27, 2001.
- 9 J. Cartmell. Generalized algebraic theories and contextual categories. *Annals of Pure and Applied Logic*, 32:209–243, 1986.
- 10 The Coq Development Team. *The Coq Proof Assistant Reference Manual (Version 8.1)*, INRIA, 2007.
- 11 F. Part. Implementation of HoTTEq and SSTs in Plastic. <https://github.com/part-xx/hott-plastic>, 2015.
- 12 Hugo Herbelin. A dependently-typed construction of semi-simplicial types. *Mathematical Structures in Computer Science*, 2014. to appear.
- 13 M. Hovey. *Model categories*. American Mathematical Soc., 2007.
- 14 IAS-web-on-SST. Semi-Simplicial Types, Year of Univalent Foundations, IAS at Princeton. <http://uf-ias-2012.wikispaces.com/Semi-simplicial+types>, 2012.
- 15 C. Kapulkin, P.-L. Lumsdaine, and V. Voevodsky. The simplicial model of univalent foundations. 2012.

- 16 Z. Luo. *Computation and Reasoning: a Type Theory for Computer Science*. Oxford University Press, 1994.
- 17 Z. Luo. A type-theoretic framework for formal reasoning with different logical foundations. In *Advances in Computer Science - ASIAN 2006. Secure Software and Related Issues, 11th Asian Computing Science Conference, Tokyo, Japan, December 6-8, 2006, Revised Selected Papers*, pages 214–222, 2006.
- 18 Z. Luo and R. Pollack. LEGO Proof Development System: User’s Manual. LFCS Report ECS-LFCS-92-211, Dept of Comp. Sci., U of Edinburgh, 1992.
- 19 Jacob Lurie. *Higher Topos Theory*. Annals of Mathematics Studies. Princeton University Press, Princeton, NJ, 2009.
- 20 M. Maietti. *The type theory of categorical universes*. PhD thesis, University of Padova, 1998.
- 21 M. Makkai. First order logic with dependent sorts. preprint, 1995.
- 22 P. Martin-Löf. *Intuitionistic Type Theory*. Bibliopolis, 1984.
- 23 P. Martin-Löf. Truth of a proposition, evidence of a judgement, validity of a proof. Talk given at Workshop of Theories of Meaning, Florence, June 1985.
- 24 C. Rezk. A model for the homotopy theory of homotopy theory. In *Trans. Amer. Math. Soc.*, pages 973–1007.
- 25 The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. <http://homotopytypetheory.org/book>, Institute for Advanced Study, 2013.
- 26 V. Voevodsky. Homotopy Type System. <http://ncatlab.org/homotopytypetheory/show/Homotopy+Type+System>, 2012.
- 27 V. Voevodsky. Semi-simplicial types in HTS. <http://uf-ias-2012.wikispaces.com/file/detail/semisimplicial.v>, 2012.