Pure Type Systems to Homotopy Calculus: The Infinity Language Stack

Maksym Sokhatskyi[1,a),c)] and Pavlo Maslianko[1,b)]

[1]*National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnical Institute"*

[a)]maxim@synrc.com
[b)]mppdom@i.ua
[c)]URL: https://groupoid.space

Abstract. We expose the layered models of type systems with the arising complexity. While designing the language or type checker it is useful to do a separation or reusing of AST trees. In the resulting tower we have four languages: 1) PTS[1] [1] with infinite number of universes; 2) MLTT[2] [2] system with Pi, Sigma and Equ; 3) Calculus of Inductive Constructions [3] — system with induction principle using backends as encodings: IR-types[3] [4] and W-types[4] backends; 4) Homotopy Calculus with interval [0,1] [5]. The models are given in cubical type checker, which is CCHM-fibrations[5] [6] based type checker. The central part and main motivation in type theory is Equality type that is fully derivable with its eliminators and computational rules in CCHM model. We show examples of Path Equality from a future base library and give some insights on future development.

Introduction

The idea of Infinity Langauge came from the needs of unification and arranging different calculuses as extensions to the core of the language with dependent types (or MLTT core). E.g. pi-calculus (spawn, send, recv, pub, sub) and stream calculus (map, scan, concat, reduce, iota, fold, split) are connected as streams represent variables in pi-calculus. This gives us a direction to build a solution towards unified layered type checker.

This type checker should introduce new complexity on each layer while remaining compatible with properties on previous layer. Here is shown: 1) Pure Type System with infinite number of universes for consistency as a base calculus; 2) Martin-Löf Type Theory as intermediate calculus with sigma and identity reasoning for pattern matching; 3) EXE calculus with inductive types, inductively recursive IR-encoding and W-types encoding and derivable induction principle, therefore fixpoint type. 4) Homotopy calculus based on interval $[0, 1]$, its introduction, eliminators, composition and gluening.

TABLE 1. Type Systems

PTS	MLTT	CiC	CCHM
PI	PI SIGMA ID[6]	PI SIGMA ID INDUCTION	PI SIGMA PATH COMPOSITION GLUENING

[1]Pure Type System, Henk Barendregt
[2]Martin-Löf Type Theory, as for 1984
[3]Induction-Recursion
[4]Wellfounded Trees
[5]Cohen-Coquand-Huber-Mörtberg fibrations

Pure Type Systems

From the time Coquand at all discovered Calculus of Constructions [7], and Barendregt [8] systemized its variations, a Pure Type System theory was developed. It is known also as Single Axiom System with only Pi-Type of MLTT [2], representing functional completeness.

```
data pts
   = star              (n: nat)
   | var     (x: name) (l: nat)
   | pi      (x: name) (l: nat) (d c: lang)
   | lambda  (x: name) (l: nat) (d c: lang)
   | app                        (f a: lang)
```

This language is called one axiom language (or pure) as eliminator and introduction rules inferred from type formation rule. The only computation rule of Pi type is called beta-reduction.

$$\frac{x : A \vdash B : Type}{\Pi\,(x : A) \to B : Type} \tag{Π-formation}$$

$$\frac{x : A \vdash b : B}{\lambda\,(x : A) \to b : \Pi\,(x : A) \to B} \tag{λ-intro}$$

$$\frac{f : (\Pi\,(x : A) \to B) \quad a : A}{f\ a : B\,[a/x]} \tag{App-elimination}$$

$$\frac{x : A \vdash b : B \quad a : A}{(\lambda\,(x : A) \to b)\ a = b\,[a/x] : B\,[a/x]} \tag{β-computation}$$

$$\frac{\pi_1 : A \quad u : A \vdash \pi_2 : B}{[\pi_1/u]\,\pi_2 : B} \tag{subst}$$

Sigma Types

The basic Core primitive which is needed for proving things is MLTT Sigma-Type, representing Contextual Completeness. This is needed for building Sigma pairs which are curried records. Usually, type checkers called Pi-Sigma provers as it contains PTS enriched with Sigma primitive.

```
data exists
   = sigma (n: name) (a b: lang)
   | pair (a b: lang)
   | fst (p: lang)
   | snd (p: lang)
```

E.g. one may want to define vectors as refinement type of list type using sigma pair construction:

```
vector (n: nat) (A: U): U
   = (c: list nat)
   * (Path nat (length nat c) n)
```

instead of using initial object with restriction in second parameter of vcons constructor:

```
data vector (A: U) (n: nat)
   = vnil
   | vcons (x: A) (xs: vector A (pred n))
```

We created an PTS type checker for Erlang/OTP platform [7]. The download and run instructions could be obtained from Github page.

[7] https://github.com/groupoid/om

Identity Types

Starting from an appearance of MLTT systems there was a question about derivability of equality along with J eliminator and functional extensionality. While IR-encoding [4] provides the useful descriptive mechanism for handling equality eliminators, homotopical interval as a model of equality is encoded internally within CCHM-fibrations system [6]. The identity type could also be modeled using setoids [9], but it will lacks functional extensionality which is also non-derivable in MLTT systems.

```
data identity
    = id (a b: lang)
    | idpair (a b: lang)
    | idelim (a b pa pb u: lang)
```

The core of elimination rule is a substitution or transport that gives as a fact of equality of points in dependent types indexed by two points with a given equality.

$$\frac{a : A \quad b : A \quad A : Type}{Id(A, a, b) : Type} \qquad (Id\text{-formation})$$

$$\frac{a : A}{refl(A, a) : Id(A, a, a)} \qquad (Id\text{-intro})$$

$$\frac{p : Id(a, b) \quad x, y : A \quad u : Id(x, y) \vdash E : Type \quad x : A \vdash d : E\ [x/y,\ refl(x)/u]}{J(a, b, p, (x, y, u)\ d) : E\ [a/x,\ b/y,\ p/u]} \qquad (J\text{-elimination})$$

$$\frac{a, x, y : A, \quad u : Id(x, y) \vdash E : Type \quad x : A \vdash d : E\ [x/y,\ refl(x)/u]}{J(a, a, refl(a), (x, y, u)\ d) = d\ [a/x] : E\ [a/y,\ refl(a)/u]} \qquad (Id\text{-computation})$$

The elimination rule of indentity type cannot be expressed inside MLTT. That is why in all provers before CCHM-based cubical have encoded J eliminator as part of AST and core language primitive. However lately we where able to model a theory where J is derivable using $[0, 1]$ homotopy interval [5] and CCHM-fibrations. Here is example of Identity Type theory using cubical type checker:

```
Path    (A: U) (a b: A): U = PathP (<i> A) a b
refl    (A: U) (a: A): Path A a a = <i> a
singl   (A: U) (a: A): U = (x : A) * Path A a x
sym     (A: U) (a b: A) (p: Path A a b): Path A b a = <i> p @ -i
trans   (A B: U) (p: Path U A B) (a: A): B = comp p a []
```

The differences between congruence, functional extensionality, substitution and contractablity of singletons are given in explicit manner by their cubical definitions.

```
cong    (A B: U) (f: A -> B) (a b: A) (p: Path A a b):
     Path B (f a) (f b) = <i> f (p @ i)

funExt (A B: U) (f g: A -> B) (p: (x: A) -> Path B (f x) (g x)):
     Path (A -> B) f g = <i> \(a : A) -> p a @ i

mapOnPath (A B: U) (f: A -> B) (a b: A) (p: Path A a b):
     Path B (f a) (f b) = <i> f (p @ i)

subst (A: U) (P: A -> U) (a b: A) (p: Path A a b) (e: P a):
     P b = comp (<i> P (p @ i)) e []

contrSingl (A: U) (a b: A) (p: Path A a b):
     Path (singl A a) (a,refl A a) (b,p) = <i> (p @ i,<j> p @ i/\j)
```

Identity Eliminator J

J is formulated in a form of Paulin-Mohring and implemented using two facts that singleton are contractible and dependent function transport.

```
JPM (A: U) (a b: A) (P: singl A a -> U) (u: P (a,refl A a)) (p: Path A a b):
     P (b,p) = subst (singl A a) T (a,refl A a) (b,p) (contrSingl A a b p) u
     where T (z : singl A a) : U = P z
```

Another J based on path induction from HoTT book [10].

```
J (A: U) (a: A) (C: (x : A) -> Path A a x -> U)
   (d: C a (refl A a)) (x : A) (p: Path A a x) : C x p =
     subst (singl A a) T (a, refl A a) (x, p) (contrSingl A a x p) d
       where T (z: singl A a): U = C (z.1) (z.2)

composition (A: U) (a b c: A) (p: Path A a b) (q: Path A b c): Path A a c
     = <i> comp (<j>A) (p @ i) [ (i = 1) -> q, (i=0) -> <j> a ]
```

Inductive Types

The further development of induction [11, 12] inside MLTT provers led to the theory of polynomial functors and well-founded trees [13], known in programming languages as inductive types with data and record core primitives of type checker. In fact recursive inductive types [14] could be encoded in PTS using non-recursive representation Berarducci [15] or categorical encoding.

```
data tele (A: U)   = emp | tel (n: name) (b: A) (t: tele A)
data branch (A: U) =        br (n: name) (args: list name) (term: A)
data label (A: U)  =       lab (n: name) (t: tele A)

data ind
     = data_  (n: name) (t: tele lang) (labels:   list (label lang))
     | case   (n: name) (t: lang)      (branches: list (branch lang))
     | ctor   (n: name)                (args:     list lang)
```

$$\frac{A : Type \quad x : A \quad B(x) : Type}{W(x : A) \to B(x) : Type} \tag{W-formation}$$

$$\frac{a : A \quad t : B(a) \to W}{sup(a, t) : W} \tag{W-intro}$$

$$\frac{w : W \vdash C(w) : Type \quad x : A,\ u : B(x) \to W, \quad v : \Pi(y : B(x)) \to C(u(y)) \vdash c(x, u, v) : C(sup(x, u))}{w : W \vdash wrec(w, c) : C(w)} \tag{W-elimination}$$

$$\frac{w : W \vdash C(w) : Type \quad x : A,\ u : B(x) \to W, \quad v : \Pi(y : B(x)) \to C(u(y)) \vdash c(x, u, v) : C(sup(x, u))}{x : A, \quad u : B(x) \to W \vdash wrec(sup(x, u), c) \ = \ c(x, u, \lambda(y : B(x)) \to wrec(u(y), c)) : C(sup(x, u))} \tag{W-computation}$$

The non-well-founded trees or infinite coinductive trees [16, 17] are useful for modeling infinite process running which is part of Milner's Pi-calculus. Coinductive streams are part of any MLTT base library.

We gathered several models in lambek module of base library [8]. It includes: 1) Recursion Schemes; 2) Catamorphism Inductive Encoding; 3) IR-encoding [4]; 4) F-Algebra encoding [18].

[8] https://github.com/groupoid/infinity/blob/master/priv/lambek.ctt

Higher Inductive Types

The fundamental development of equality inside MLTT provers led us to the notion of ∞-groupoid [19] as spaces. In this was Path identity type appeared in the core of type checker along with de Morgan algebra on built-in interval type. Glue, unglue composition and fill operations are also needed in the core for the univalence computability [5].

```
data hts
    = path (a b: lang)
    | path_lam (n: name) (a b: lang)
    | path_app (f: name) (a b: lang)
    | comp_ (a b: lang)
    | fill_ (a b c: lang)
    | glue_ (a b c: lang)
    | glue_elem (a b: lang)
    | unglue_elem (a b: lang)
```

Comparing Homotopical and Inductive proofs

Here we show how to use Path equality from CCHM **cubical** type checker:

```
add_comm (a : nat) : (n : nat) -> Path nat (add a n) (add n a) = split
  zero  -> <i> add_zero a @ -i
  suc m -> <i> comp (<_> nat) (suc (add_comm a m @ i))
                    [ (i = 0) -> <j> suc (add a m)
                    , (i = 1) -> <j> add_suc m a @ -j ]
```

and how it is differ e.g. with inductive based proofs from **Coq**:

```
Theorem plus_comm : forall n m : nat, n + m = m + n.
Proof.
  intros n m.
  induction n as [| n' Sn' ].
  - simpl. rewrite <- plus_n_0. reflexivity.
  - simpl. rewrite <- plus_n_Sm. rewrite <- Sn'. reflexivity.
Qed.
```

Conclusion

In this article we gave a simple AST definitions for tower of languages on top of Pure Type System, show a cubical terms for Identity types, compare the inductive and homotopical proof styles. The main motivation in layering the complex proving languages is to simplify the reasoning about them, manage and reduce the stack of calculuses.

```
data lang
   = PTS (_: pts)
   | SIGMA (_: exists)
   | ID (_: identity)
   | INDUCTION (_: ind)
   | CUBICAL (_: hts)
```

Such decoupling of top level language brings modularity to its essence, the type checker. We can reason about properties of cacluluses polymorphically disregarding of underlying types in computational equations, such as *Path* (*B a*) ((*app B* (*lambda B f*)) *a*) (*f a*). Here we don't know *a* is term of PTS or SIGMA language, this equation holds for all terms *a*. We treating language core as AST sum of sublanguages, that is extendable to the needed set of language features.

REFERENCES

[1] S. P. Jones and E. Meijer, "Henk: A typed intermediate language," in *In Proc. First Int'l Workshop on Types in Compilation* (1997).

[2] P. Martin-Löf and G. Sambin, *Intuitionistic type theory*, Studies in proof theory (Bibliopolis, 1984).

[3] C. Paulin-Mohring, in *All about Proofs, Proofs for All*, Studies in Logic (Mathematical logic and foundations), Vol. 55, edited by B. W. Paleo and D. Delahaye (College Publications, 2015).

[4] P. Dagand, U. of Strathclyde. Department of Computer, and P. t. Information Sciences, *A Cosmology of Datatypes: Reusability and Dependent Types* (2013).

[5] C. Cohen, T. Coquand, S. Huber, and A. Mörtberg, in *Cubical Type Theory: a constructive interpretation of the univalence axiom*, Vol. abs/1611.02108 (2017).

[6] I. Orton and A. M. Pitts, arXiv preprint arXiv:1712.04864 (2017).

[7] T. Coquand and G. Huet, "The calculus of constructions," in *Information and Computation* (Academic Press, Inc., Duluth, MN, USA, 1988), pp. 95–120.

[8] H. P. Barendregt, in *Handbook of Logic in Computer Science (Vol. 2)*, edited by S. Abramsky, D. M. Gabbay, and S. E. Maibaum (Oxford University Press, Inc., New York, NY, USA, 1992) Chap. Lambda Calculi with Types,, pp. 117–309.

[9] E. Bishop, *Foundations of constructive analysis* (1967).

[10] T. Coquand, P. Martin-Löf, V. Voevodsky, A. Joyal, A. Bauer, S. Awodey, M. Sozeau, M. Shulman, D. Licata, Y. Bertot, P. Dybjer, and N. Gambino, *Homotopy Type Theory: Univalent Foundations of Mathematics* (2013).

[11] P. Dybjer, in *Inductive families*, Vol. 6 (Springer, 1994), pp. 440–465.

[12] V. Vene, *Categorical programming with inductive and coinductive types* (Tartu University Press, 2000).

[13] N. Gambino and M. Hyland, "Wellfounded trees and dependent polynomial functors," in *International Workshop on Types for Proofs and Programs* (Springer, 2003), pp. 210–225.

[14] P. Wadler, in *Recursive types for free* (manuscript, 1990).

[15] C. Böhm and A. Berarducci, "Automatic synthesis of typed lambda-programs on term algebras," in *Theoretical Computer Science*, Vol. 39 (1985), pp. 135–154.

[16] B. Jacobs and J. Rutten, in *A tutorial on (co) algebras and (co) induction*, Vol. 62 (EUROPEAN ASSOCIATION FOR THEORETICAL COMPUTER, 1997), pp. 222–259.

[17] H. Basold and H. Geuvers, "Type theory based on dependent inductive and coinductive types," in *Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science* (ACM, 2016), pp. 327–336.

[18] R. Hinze and N. Wu, "Histo- and dynamorphisms revisited," in *Proceedings of the 9th ACM SIGPLAN Workshop on Generic Programming*, WGP '13 (ACM, New York, NY, USA, 2013), pp. 1–12.

[19] M. Hofmann and T. Streicher, "The groupoid interpretation of type theory," in *Twenty-five years of constructive type theory (Venice, 1995)*, Oxford Logic Guides, Vol. 36 (Oxford Univ. Press, New York, 1998), pp. 83–111.

Contents