Anders: Modal Homotopy Type System

Namdak Tonpa
Groupoid Infinity

Siegmentation Fault
Groupoid Infinity

Abstract

Here is presented a reincarnation of **cubicaltt** called **anders**.

2012 ACM Subject Classification Theory of computation → Lambda calculus

Keywords and phrases Homotopy Type System, Cubical Type Theory

1 Introduction

Anders is a Modal HoTT proof assistant based on: classical MLTT-80 [3] with 0, 1, 2, W types; CCHM [2] in CHM flavour as cubical type system with hcomp/transp Kan operations; HTS [5] strict equality on pretypes; de Rham [**?**] stack modality primitives. We tend not to touch general recursive higher inductive schemes yet, instead we will try to express as much HIT as possible through W, Coequlizer and HubSpokes Disc in the style of HoTT/Coq homotopy library and Three-HIT theorem.

The HTS language proposed by Voevodsky exposes two different presheaf models of type theory: the inner one is homotopy type system presheaf that models HoTT and the outer one is traditional Martin-Löf type system presheaf that models set theory with UIP. The motivation behind this doubling is to have an ability to express semisemplicial types. Theoretical work on merging inner and outer languages was continued in 2LTT [1].

While we are on our road to Lean-like tactic language, currently we are at the stage of regular cubical HTS type checker with CHM-style [5] primitives. You may try it from Github sources: [1]groupoid/anders or install through OPAM package manager. Main commands are **check** (to check a program) and **repl** (to enter the proof shell).

Anders is fast, idiomatic and educational (think of optimized Mini-TT). We carefully draw the favourite Lean-compatible syntax to fit 200 LOC in Menhir. The CHM kernel is 1K LOC. Whole Anders compiles under 2 seconds and checks all the base library under 1 second [i7-8700]. Anders proof assistant as Homotopy Type System comes with its own Homotopy Library.

2 Syntax

The syntax resembles original syntax of the reference CCHM type checker cubicaltt, is slightly compatible with Lean syntax and contains the full set of Cubical Agda [4] primitives (except generic higher inductive schemes).

Here is given the mathematical pseudo-code notation of the language expressions that come immediately after parsing. The core syntax definition of HTS language corresponds to the type defined in OCaml module:

Further Menhir BNF notation will be used to describe the top-level language E parser.

[1] `https://github.com/groupoid/anders`

$$
\begin{aligned}
\text{cosmos} &:= \mathbf{U}_j \mid \mathbf{V}_k \\
\text{var} &:= \mathbf{var}\ name \mid \mathbf{hole} \\
\text{pi} &:= \Pi\ name\ E\ E \mid \lambda\ name\ E\ E \mid E\ E \\
\text{sigma} &:= \Sigma\ name\ E\ E \mid (E, E) \mid E.1 \mid E.2 \\
\text{id} &:= \mathbf{Id}\ E \mid \mathbf{ref}\ E \mid \mathbf{idJ}\ E \\
\text{path} &:= \mathbf{Path}\ E \mid E^i \mid E\ @\ E \\
\text{I} &:= \mathbf{I} \mid 0 \mid 1 \mid E \bigvee E \mid E \bigwedge E \mid \neg E \\
\text{part} &:= \mathbf{Partial}\ E\ E \mid [\ (E = I) \rightarrow E, \ldots\ ] \\
\text{sub} &:= \mathbf{inc}\ E \mid \mathbf{ouc}\ E \mid E\ [\ I\ \mapsto\ E\ ] \\
\text{kan} &:= \mathbf{transp}\ E\ E \mid \mathbf{hcomp}\ E \\
\text{glue} &:= \mathbf{Glue}\ E \mid \mathbf{glue}\ E \mid \mathbf{unglue}\ E\ E
\end{aligned}
$$

$$
\begin{aligned}
\text{E} &:= cosmos \mid var \mid MLTT \mid CCHM \mid HIT \\
\text{HIT} &:= \mathbf{inductive}\ E\ E \mid \mathbf{ctor}\ name\ E \mid \mathbf{match}\ E\ E \\
\text{CCHM} &:= path \mid I \mid part \mid sub \mid kan \mid glue \\
\text{MLTT} &:= pi \mid sigma \mid id
\end{aligned}
$$

Keywords. The words of a top-level language (file or repl) consist of keywords or identifiers. The keywords are following: **module**, **where**, **import**, **option**, **def**, **axiom**, **postulate**, **theorem**, (,), [,], <, >, /, **.1**, **.2**, Π, Σ, **,**, λ, ×, →, **:**, **:=**, ↦, **U**, **V**, ⋁, ⋀, **-**, +, **@**, **PathP**, **transp**, **hcomp**, **zero**, **one**, **Partial**, **inc**, **ouc**, **interval**, **inductive**, **Glue**, **glue**, **unglue**.

Indentifiers. Identifiers support UTF-8. Indentifiers couldn't start with :, −, →. Sample identifiers: ¬-of-⋁, 1→1, is-?, =, $∼]!005x, ∞, x→Nat.

Modules. Modules represent files with declarations. More accurate, BNF notation of module consists of imports, options and declarations.

```
menhir
   start <Module.file> file
   start <Module.command> repl
   repl : COLON IDENT exp1 EOF | COLON IDENT EOF | exp0 EOF | EOF
   file : MODULE IDENT WHERE line* EOF
   path : IDENT
   line : IMPORT path+ | OPTION IDENT IDENT | declarations
```

Imports. The import construction supports file folder structure (without file extensions) by using reserved symbol / for hierarchy walking.

Options. Each option holds bool value. Language supports following options: 1) girard (enables U : U); 2) pre-eval (normalization cache); 3) impredicative (infinite hierarchy with impredicativity rule); In Anders you can enable or disable language core types, adjust syntaxes or tune inner variables of the type checker. Here is the example how to setup minimal core able to prove internalization of **MLTT-73** variation (Path instead of Id and no inductive types, see base library): In order to turn HIT into ordinary CiC calculus you may say:

Declarations. Language supports following top level declarations: 1) axiom (non-computable declaration that breakes normalization); 2) postulate (alternative or inverted axiom that can preserve consistency); 3) definition (almost any explicit term or type in type theory); 4) lemma (helper in big game); 5) theorem (something valuable or complex enough).

Sample declarations. For example, signature isProp (A : U) of type U could be defined as normalization-blocking axiom without proof-term or by providing proof-term as definition.

```
option HIT false
option CCHM false
option MLTT true

axiom isProp (A : U) : U
def isSet (A : U) : U := Π (a b : A) (x y : Path A a b), Path (Path A a b) x y
```

In this example (A : U), (a b : A) and (x y : Path A a b) are called telescopes. Each telescope consists of a series of lenses or empty. Each lense provides a set of variables of the same type. Telescope defines parameters of a declaration. Types in a telescope, type of a declaration and a proof-terms are a language expressions exp1.

```
menhir
   ident : IRREF | IDENT
   lense : LPARENS ident+ COLON exp1 RPARENS
   telescope : lense telescope
   params : telescope | []
   declarations :
      | DEF IDENT params DEFEQ exp1
      | DEF IDENT params COLON exp1 DEFEQ exp1
      | AXIOM IDENT params COLON exp1
```

Expressions. All atomic language expressions are grouped by four categories: exp0 (pair constructions), exp1 (non neutral constructions), exp2 (path and pi applcations), exp3 (neutral constructions).

```
menhir
   face : LPARENS IDENT IDENT IDENT RPARENS
   part : face+ ARROW exp1
   exp0 : exp1 COMMA exp0 | exp1
   exp1 : LSQ separated(COMMA, part) RSQ
      | LAM telescope COMMA exp1     | PI telescope COMMA exp1
      | SIGMA telescope COMMA exp1   | LSQ IRREF ARROW exp1 RSQ
      | LT ident+ GT exp1            | exp2 ARROW exp1
      | exp2 PROD exp1               | exp2
```

The LR parsers demand to define exp1 as expressions that cannot be used (without a parens enclosure) as a right part of left-associative application for both Path and Pi lambdas. Universe indicies U_j (inner fibrant), V_k (outer pretypes) and S (outer strict omega) are using unicode subscript letters that are already processed in lexer.

```
menhir
   exp2 : exp2 exp3 | exp2 APPFORMULA exp3 | exp3
   exp3 : LPARENS exp0 RPARENS LSQ exp0 MAP exp0 RSQ
      | HOLE           | PRE            | KAN            | IDJ exp3
      | exp3 FST       | exp3 SND       | NEGATE exp3    | INC exp3
      | exp3 AND exp3  | exp3 OR exp3   | ID exp3        | REF exp3
      | OUC exp3       | PATHP exp3     | PARTIAL exp3   | IDENT
      | IDENT LSQ exp0 MAP exp0 RSQ                      | HCOMP exp3
      | LPARENS exp0 RPARENS                             | TRANSP exp3 exp3
```

3 Semantics

The idea is to have a unified layered type checker, so you can disbale/enable any MLTT-style inference, assign types to universes and enable/disable hierachies. This will be done by providing linking API for pluggable presheaf modules. We selected 5 levels of type checker awareness from universes and pure type systems up to synthetic language of homotopy type theory. Each layer corresponds to its presheaves with separate configuration for universe hierarchies. We want to mention here with homage to its authors all categorical models of

```
inductive lang : U   :=  UNI: cosmos → lang
                      |  PI: pure lang → lang
                      |  SIGMA: total lang → lang
                      |  ID: uip lang → lang
                      |  PATH: homotopy lang → lang
                      |  GLUE: gluening lang → lang
                      |  HIT: hit lang → lang
```

dependent type theory: Comprehension Categories (Grothendieck, Jacobs), LCCC (Seely), D-Categories and CwA (Cartmell), CwF (Dybjer), C-Systems (Voevodsky), Natural Models (Awodey). While we can build some transports between them, we leave this excercise for our mathematical components library. We will use here the Coquand's notation for Presheaf Type Theories in terms of restriction maps.

3.1 Universe Hierarchies

Language supports Agda-style hierarchy of universes: prop, fibrant (U), interval pretypes (V) and strict omega with explicit level manipulation. All universes are bounded with preorder

$$Prop_i \prec Fibrant_j \prec Pretypes_k \prec Strict_l, \tag{1}$$

in which i, j, k, l are bounded with equation:

$$i < j < k < l. \tag{2}$$

Large elimination to upper universes is prohibited. This is extendable to Agda model:

```
inductive cosmos : U   :=  prop: nat → cosmos
                        |  fibrant: nat → cosmos
                        |  pretypes: nat → cosmos
                        |  strict: nat → cosmos
                        |  omega: cosmos
                        |  lock: cosmos
```

The **anders** model contains only fibrant U_j and pretypes V_k universe hierarchies.

3.2 Dependent Types

▶ **Definition 1** (Type). *A type is interpreted as a presheaf A, a family of sets A_I with restriction maps $u \mapsto u\ f, A_I \to A_J$ for $f : J \to I$. A dependent type B on A is interpreted by a presheaf on category of elements of A: the objects are pairs (I, u) with $u : A_I$ and morphisms $f : (J, v) \to (I, u)$ are maps $f : J \to$ such that $v = u\ f$. A dependent type B is thus given by a family of sets $B(I, u)$ and restriction maps $B(I, u) \to B(J, u\ f)$.*

We think of A as a type and B as a family of presheves $B(x)$ varying $x : A$. The operation $\Pi(x : A)B(x)$ generalizes the semantics of implication in a Kripke model.

▶ **Definition 2** (Pi). *An element $w : [\Pi(x : A)B(x)](I)$ is a family of functions $w_f : \Pi(u : A(J))B(J, u)$ for $f : J \to I$ such that $(w_f u)g = w_{f\ g}(u\ g)$ when $u : A(J)$ and $g : K \to J$.*

```
inductive pure (lang : U) : U   :=   pi: name → nat → lang → lang → pure lang
                                |    lambda: name → nat → lang → lang → pure lang
                                |    app: lang → lang → pure lang
```

▶ **Definition 3** (Sigma). *The set $\Sigma(x : A)B(x)$ is the set of pairs (u, v) when $u : A(I), v : B(I, u)$ and restriction map $(u, v)\ f = (u\ f, v\ f)$.*

```
inductive total (lang : U) : U   :=   sigma: name → lang → total lang
                                 |    pair: lang → lang → total lang
                                 |    fst: lang → total lang
                                 |    snd: lang → total lang
```

The presheaf with only Pi and Sigma is called **MLTT-72**. Its internalization in **anders** is as follows:

```
def MLTT-72 (A : U) (B : A → U) : U := Σ
  (Π-form₁ : U)
  (Π-ctor₁ : Pi A B → Pi A B)
  (Π-elim₁ : Pi A B → Pi A B)
  (Π-comp₁ : (a : A) (f : Pi A B), Π-elim₁ (Π-ctor₁ f) a = f a)
  (Π-comp₂ : (a : A) (f : Pi A B), f = λ (x : A), f x)
  (Σ-form₁ : U)
  (Σ-ctor₁ : Π (a : A) (b : B a) , Sigma A B)
  (Σ-elim₁ : Π (p : Sigma A B), A)
  (Σ-elim₂ : Π (p : Sigma A B), B (pr₁ A B p))
  (Σ-comp₁ : Π (a : A) (b: B a), a = Σ-elim₁ (Σ-ctor₁ a b))
  (Σ-comp₂ : Π (a : A) (b: B a), b = Σ-elim₂ (a, b))
  (Σ-comp₃ : Π (p : Sigma A B), p = (pr₁ A B p, pr₂ A B p)), 1
```

3.3 Path Equality

3.4 Strict Equality

3.5 Glue Types

3.6 Higher Inductive Types

References

1. Danil Annenkov, Paolo Capriotti, Nicolai Kraus, and Christian Sattler. Two-level type theory and applications, 2019. `arXiv:1705.03307`.
2. Cyril Cohen, Thierry Coquand, Simon Huber, and Anders Mörtberg. Cubical Type Theory: A Constructive Interpretation of the Univalence Axiom. In Tarmo Uustalu, editor, *21st International Conference on Types for Proofs and Programs (TYPES 2015)*, volume 69 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 5:1–5:34, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: `http://drops.dagstuhl.de/opus/volltexte/2018/8475`, `doi:10.4230/LIPIcs.TYPES.2015.5`.
3. Martin-Löf. Intuitionistic type theory, 1980.
4. Andrea Vezzosi, Anders Mörtberg, and Andreas Abel. Cubical agda: A dependently typed programming language with univalence and higher inductive types. *Proc. ACM Program. Lang.*, 3(ICFP), July 2019. `doi:10.1145/3341691`.
5. Vladimir Voevodsky. A simple type system with two identity types, 2013.