# Implemntation of dependent type theory

27/09/2023

## Introduction

We try to describe what should be the values for an implementation of the poset model and then what should be the main algorithms.

We write $\varphi,\ \psi, \dots$ for the cofibrations, and we have a type of cofibrations $\Phi$. We also have an interval type $\mathbf{I}$.

We have two levels of closures $(\lambda_{x:A}t)\rho$ where $t$ is a term and $\rho$ an environment, and $(\lambda_z v)\alpha$ where $v$ is a value and $\alpha$ an interval substitution of the form $z_1 = e_1, \dots, z_n = e_n$.

Ideally, I would like an implementation with head linear reduction and with the new representation of proposition truncation, as the data type generated by constructors $\mathsf{inc}\ u$ and $\mathsf{ext}\ u\ [\psi \mapsto v]$ (which is equal to $v$ on $\psi$).

## Values

$$\begin{aligned}
v &::= k^v[\psi \mapsto v] \mid (\lambda_x t)\rho \mid (\lambda_z\ v)\alpha \mid v, v \mid V \mid cv \mid hcv \\
V &::= \Pi\ v\ v \mid \Sigma\ v\ v \mid U \mid \mathsf{Path}\ v\ v\ v \mid \mathsf{Ext}\ v\ [\psi \mapsto (v, w, w')] \\
cv &::= \mathsf{coe}\ r_0\ r_1\ v \mid \mathsf{coe}\ r_0\ r_1\ (\lambda_z \Pi\ v\ v)\alpha\ v_0 \mid \mathsf{coe}\ r_0\ r_1\ (\lambda_z \Sigma\ v\ v)\alpha\ v_0 \\
hcv &::= \mathsf{hcomp}\ r_0\ r_1\ v\ [\psi \mapsto v] \mid \mathsf{hcomp}\ r_0\ r_1\ (\Pi\ v\ v)\ [\psi \mapsto v]\ v_0 \mid \mathsf{hcomp}\ r_0\ r_1\ (\Sigma\ v\ v)\ [\psi \mapsto v]\ v_0
\end{aligned}$$

$$k ::= x \mid k\ v \mid k\ r \mid \mathsf{coe}\ r_0\ r_1\ (\lambda_z k)\alpha\ v_0 \mid \mathsf{hcomp}\ r_0\ r_1\ k\ [\psi \mapsto v]\ v_0 \mid k.1 \mid k.2$$

$$\rho ::= () \mid D\rho \mid \rho, x = v \qquad \alpha ::= () \mid \alpha, z = r$$

Here $r$ is an interval (lattice) expression.

We can choose to have interval expression as values.

We define substitution on values; the main clauses are for $\rho\alpha$

$$()\alpha = () \quad (\rho, x = v)\alpha = (\rho\alpha, x = v\alpha) \quad (D\rho)\alpha = D(\rho\alpha)$$

and for $\beta\alpha$

$$()\alpha = \alpha \qquad (\beta, x = e)\alpha = (\beta\alpha, x = e\alpha)$$

$\alpha$ represents a map between two stages. The substitution $()$ corresponds to going from a stage $X$ to a stage $X, z_1, \psi, z_2, \dots$ obtained by adding more interval variables and constraints. A substitution $(z = r)$ should correspond to a substitution $X \to X, z, \psi$ so that $\psi(z = r)$ is true at stage $X$.

## Main functions

The suggestion is to follow the algorithms for the cartesian version, but to use connections for the fact that singleton are contractible. Actually, the contractibility of singleton is also expressed by the hcomp function. (There are two notions of contractibility: the one expressing that any partial element can be extended to a total element, and the one coming from type theory with a center of contraction.)

The main functions seem to be eval and application on values.

These two functions on values have as parameter the stage of evaluation $X$ (which is a presentation of a distributive lattice).

For instance when we evaluate $[\psi \mapsto t]$ at stage $X$ and environment $\rho$ we should evaluate $t$ at stage $X, \psi$ and environment $\rho$.

The stage can only be modified by adding fresh interval variables or adding new constraints.

I explain application on values $w\ u$.

If $w$ is a closure $(\lambda_{x:A} t)\rho$ then we evaluate $t(\rho, x = u)$.

If $w$ is a closure $(\lambda_z v)\alpha$ then $u$ is an interval expression $r$ and we evaluate $v(\alpha, z = r)$.

If $w$ is of the form $\mathsf{coe}\ r_0\ r_1\ L$ then $L$ is a line value. If it is not in the form $(\lambda_z V)\alpha$ where $V$ is $\Pi, \Sigma$ or $\mathsf{Path}$ then we generate a fresh interval variable $z$ and evaluate $L\ z$ at the stage $X, z$ getting a value $V$. We put then $L$ in the form $(\lambda_z V)()$.

## Some examples

We define an auxiliary function comp

$$\mathsf{comp}\ r_0\ r_1\ L\ [\psi \mapsto u]\ \ u_0 = \mathsf{hcomp}\ r_0\ r_1\ (L\ r_1)\ [\psi \mapsto (\lambda_z \mathsf{coe}\ z\ r_1\ L\ (u\ z))()]\ (\mathsf{coe}\ r_0\ r_1\ L\ u_0)$$

This function takes an extra argument $X$ which is the stage at which we do the computation in order to be able to generate the fresh variable $z$.

Here are some clauses for coe

$$\mathsf{coe}\ r_0\ r_1\ (\lambda_z \Pi\ A\ B)\alpha\ u_0\ a_1 =$$
$$\mathsf{coe}\ r_0\ r_1\ (\lambda_{z'} B\ (\alpha, z = z')(\mathsf{coe}\ r_1\ z'\ (\lambda_z A)\alpha\ a_1))()\ (u_0\ (\mathsf{coe}\ r_1\ r_0\ (\lambda_z A)\alpha\ a_1))$$

$$(\mathsf{coe}\ r_0\ r_1\ (\lambda_z \Sigma\ A\ B)\alpha\ u_0).1 =$$
$$\mathsf{coe}\ r_0\ r_1\ (\lambda_z A)\alpha\ u_0.1$$

$$(\mathsf{coe}\ r_0\ r_1\ (\lambda_z \Sigma\ A\ B)\alpha\ u_0).2 =$$
$$\mathsf{coe}\ r_0\ r_1\ (\lambda_{z'} B\ (\alpha, z = z')(\mathsf{coe}\ r_0\ z'\ (\lambda_z A)\alpha\ u_0.1))()\ u_0.2$$

$$\mathsf{coe}\ r_0\ r_1\ (\lambda_z \mathsf{Path}\ A\ a\ b)\alpha\ u_0\ r =$$
$$\mathsf{comp}\ r_0\ r_1\ (\lambda_z A)\alpha\ [r = 0 \mapsto (\lambda_z a)\alpha, r = 1 \mapsto (\lambda_z b)\alpha]\ (u_0\ r)$$

The definition of $\mathsf{hcomp}\ U$ will use the Ext (Glue) constructor and the most complex functions ar coe and hcomp for Ext types.

Here are some clauses for hcomp.

$$\mathsf{hcomp}\ r_0\ r_1\ (\Pi\ A\ B)\ [\psi \mapsto u]\ u_0\ a =$$
$$\mathsf{hcomp}\ r_0\ r_1\ (B\ a)\ [\psi \mapsto (\lambda_z\ (u\ z\ a))()](u_0\ a)$$

$$(\mathsf{hcomp}\ r_0\ r_1\ (\Sigma\ A\ B)\ [\psi \mapsto u]\ u_0).1 =$$
$$\mathsf{hcomp}\ r_0\ r_1\ A\ [\psi \mapsto u.1]\ u_0.1$$

$$(\mathsf{hcomp}\ r_0\ r_1\ (\Sigma\ A\ B)\ [\psi \mapsto u]\ u_0).2 =$$
$$\mathsf{comp}\ r_0\ r_1\ (\lambda_z B\ (\mathsf{hcomp}\ r_0\ z\ [\psi \mapsto u.1]\ u_0.1))()\ [\psi \mapsto u.2]\ u_0.2$$

$$\mathsf{hcomp}\ r_0\ r_1\ (\mathsf{Path}\ A\ a\ b)\ [\psi \mapsto u]\ u_0\ r =$$
$$\mathsf{hcomp}\ r_0\ r_1\ A\ [\psi \mapsto (\lambda_z u\ z\ r)(), r = 0 \mapsto (\lambda_z a)(), r = 1 \mapsto (\lambda_z b)()]\ (u_0\ r)$$

Here to simplify, we have written $u.1$ for $(\lambda_z(u\ z).1)()$ and $u.2$ for $(\lambda_z(u\ z).2)()$.

## Some combinators

In the implementation of cubicaltt, it was found convenient to introduce some combinators that are obtained as evaluation of terms. For instance we have

$$\mathsf{id} = (\lambda_A \lambda_x x)()$$

so that $\mathsf{id}\ A$ is the identity function for $A$. We can also define

$$\mathsf{Fib} = (\lambda_A \lambda_B \lambda_f \lambda_a \Sigma_{b:B} \mathsf{Path}\ A\ (f\ b)\ a)()$$

so that $\mathsf{Fib}\ A\ B\ w\ u$ is the fiber of $w$ at $u$, and let $D$ be the definition

$$D = [\mathsf{isContr} : U \to U = \lambda_A \Sigma_a \Pi_x \mathsf{Path}\ A\ a\ x]$$

so that $\mathsf{isContr} D\ A$ is the value for the fact that $A$ is contractible.

We also have

$$\mathsf{isEquiv} = (\lambda_{A:U} \lambda_{B:U} \lambda_{f:B\to A} \Pi_{a:A} \mathsf{isContr}(\Sigma_{b:B} \mathsf{Path}\ A\ (f\ b)\ a))D$$

so that $\mathsf{isEquiv}\ A\ B\ w$ expresses that $w$ is an equivalence.

We shall also need a combinator expressing that the identity is an equivalence. This is the proof that singleton are contractible, which is simple if we have connections.

$$\mathsf{isEquivId} = (\lambda_{A:U} \lambda_{a:A} ((a, z.a), \lambda_{v:\Sigma_{x:A} \mathsf{Path}\ A\ a\ x} (v.2, z'.v.2(z \wedge z'))))()$$

If $c$ a value of type $\mathsf{isContr}\ A$, so that $c$ is convertible to a pair $a_0, p$, with $p\ a$ being a path in $\mathsf{Path}\ A\ a_0\ a$, we can use this to define a function that extends a partial element of $A$ to a total element

$$\mathsf{wid}\ A\ (a_0, p)\ [\psi \mapsto u] = \mathsf{hcomp}\ 0\ 1\ A\ [\psi \mapsto p\ u]\ a_0$$

## Glue/Ext type

A canonical type at stage $X$ can be of the form $E = \mathsf{Ext}\ A\ [\varphi \mapsto (B, w, w')]$ where $\varphi \neq 1$ at stage $X$ and we have $A = B$ on $\varphi$ and $w : B \to A$ and $w'$ a proof that $w$ is an equivalence. The elements of this type are pairs $(a, b)$ where $a$ is in $A$ and $b$ in $B$ and $w\ b = a$ on $\varphi$.

We have the function $\mathsf{ext}\ w$ of type $E \to A$ which is defined by $\mathsf{ext}\ w\ (a, b) = a$ in such a way that $\mathsf{ext}\ w : E \to A$ extends the given partial function $w : B \to A$.

### Homogeneous composition

$\mathsf{hcomp}\ r_0\ r_1\ E\ [\psi \mapsto u]\ u_0$ is defined in the following way. First we can always write $u = (a, b)$ and $u_0 = (a_0, b_0)$. The output should be $a_1, b_1$ where

$$\tilde{b} = \lambda_z \mathsf{hcomp}\ r_0\ z\ B\ [\psi \mapsto b]\ b_0 \qquad b_1 = \tilde{b}\ r_1$$

and

$$a_1 = \mathsf{hcomp}\ r_0\ r_1\ A\ [\psi \mapsto a,\ \varphi \mapsto (\lambda_z w\ (\tilde{b}\ z))()]\ a_0$$

Note that we don't use $w'$ in this computation.

## Coerce function for Glue type

The last case is coerce for $E$ and $\mathsf{hcomp}$ for $U$.

The case $\mathsf{coe}\ r_0\ r_1\ (\lambda_z E)()\ (a_0, b_0)$ is the most complex one.

The value $w'$ should be a witness that $w$ is an equivalence. Using $w'(r_1)$ and the combinators $\mathsf{wid}$ and $\mathsf{Fib}$, we can define a function $f_1$, defined for values at stage $X, \varphi$, which takes as argument $a$ in $A(r_1)$ and $b$ with a path $\omega$ in $\mathsf{Path}\ A(r_1)\ (w(r_1)\ b)\ a$ only defined on $\psi \leqslant \varphi(r_1)$ and which produces as output a pair $\tilde{b}, \tilde{\omega}$ on $\varphi(r_1)$ which extends $b, \omega$.

We start by computing $\delta = \forall_z \varphi$. Using $\mathsf{coe}$ for $A$ and $B$ we compute $\tilde{a}$ line in $A$ which is $a_0$ at $r_0$ and $\tilde{b}$, defined on $\delta$, line in $B$ which is $b_0$ at $r_0$. Using then the type $\mathsf{Path}\ A\ (w\ \tilde{b})\ \tilde{a}$ we can compute by $\mathsf{coe}$ an element in $\mathsf{Path}\ A(r_1)\ (w(r_1)\ \tilde{b}(r_1))\ \tilde{a}(r_1)$ on $\delta$. Using the function $f_1$, we get an element $b_1$ in $B(r_1)$ and a path connecting $w(r_1)\ b_1$ and $\tilde{a}(r_1)$, furthermore such that $b_1 = b_0$ on $r_0 = r_1$. Using $\mathsf{hcomp}$ for $A(r_1)$ we then get an element $a_1$ in $A(r_1)$ which extends $w(r_1)\ b_1$ and is equal to $a_0$ on $r_0 = r_1$. The pair $a_1, b_1$ is the value of $\mathsf{coe}\ r_0\ r_1\ (\lambda_z E)()\ (a_0, b_0)$.

## Homogeneous composition for universes

The remaining case is $\mathsf{hcomp}\ r_0\ r_1\ U\ [\psi \mapsto A]\ A_0$. For this we compute for $z$ a witness $w'(z)$ that $\mathsf{coe}\ \ z\ r_0\ A$ is an equivalence between $A(z)$ and $A(r_0)$. For this, we define the line of types $L = \lambda_z \mathsf{isEquiv}\ A(z)\ A(r_0)\ (\mathsf{coe}\ z\ r_0\ A)$. Note that $L(r_0)$ expresses that the identity function of $A(r_0)$ is an equivalence. We have an element of $L(r_0)$ which is $\mathsf{isEquivId}\ A(r_0)$. We define

$$w'(z) = \mathsf{coe}\ r_0\ z\ L\ (\mathsf{isEquivId}\ A(r_0))$$

The result is then $\mathsf{Ext}\ A_0\ [\psi \mapsto (A(r_1), \mathsf{coe}\ r_0\ r_1\ A, w'(r_0)), r_0 = r_1 \mapsto (A_0, \mathsf{id}\ A_0, \mathsf{isEquivId}\ A_0)]$

## Univalence

Univalence can be formulated as the type $\Pi_{A:U} \mathsf{isContr}\ (\Sigma_{X:U} \mathsf{Equiv}\ X\ A)$.