An implementation of Cubical Type Theory

Mattia Furlan

19 aprile 2022

Indice

1 Syntax of the language

The syntax of the language is given by the following grammar, where A, B, M, K are meta-variables for terms and K represents (possibly) neutral terms.

$$
\begin{aligned}
A, B, M, K ::= {} & \mathtt{U} \mid K \mid [x : A]M \\
& \mid \mathtt{N} \mid \mathtt{0} \mid \mathtt{S} \\
& \mid \mathbb{I} \mid \mathtt{0}_{\mathbb{I}} \mid \mathtt{1}_{\mathbb{I}} \mid \mathit{Sys} \mid \mathit{Partial} \mid \mathit{Restr} \\
K ::= {} & x \mid KM \mid \mathtt{ind}_{\mathtt{N}}\ MMMK \\
\mathit{Sys} ::= {} & [\varphi_1 \mapsto M_1, \ldots, \varphi_n \mapsto M_n] \\
\mathit{Partial} ::= {} & [\varphi]A \\
\mathit{Restr} ::= {} & [\varphi_1 \mapsto M_1, \ldots, \varphi_n \mapsto M_n]A
\end{aligned}
$$

$$
\begin{aligned}
\varphi, \psi ::= {} & (i = 0) \mid (i = 1) \mid (i = j) \\
& \mid \varphi \vee \psi \mid \varphi \wedge \psi
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Program} ::= {} & [\mathit{TopLevel}] \\
\mathit{TopLevel} ::= {} & \mathit{Definition} \mid \mathit{Declaration} \mid \mathit{Example} \\
\mathit{Definition} ::= {} & x : A = B \\
\mathit{Declaration} ::= {} & x : A \\
\mathit{Example} ::= {} & M
\end{aligned}
$$

In more detail, the language has symbols:

- $\mathtt{U}$ for the universe of types.[1]
- $\mathtt{N}$ for the type of natural numbers, with costants $\mathtt{0}$ and $\mathtt{S}$ respectively for zero and the successor operator. $\mathtt{ind}_{\mathtt{N}}\ F\ c_0\ c_s\ n$ names the induction principle for naturals (where $F : \mathtt{N} \to \mathtt{U}$ is a type family).
- $[x : A]M$ both for the λ and Π abstractions.
- $\mathbb{I}$ for the formal $[0, 1]$ interval, with the endpoints $\mathtt{0}_{\mathbb{I}}$ and $\mathtt{1}_{\mathbb{I}}$.

We consider values, i.e. the results of the evaluation of terms, as terms extended with a closure operator of the form $\langle x : ty^{val}, M, (\Gamma; \Theta)\rangle$, where the abstracted variable x is paired with its evaluated type ty, the abstraction body is M (unevaluated) and the closure enviroment is specified by the context Γ and the directions enviroment Θ (see next section).

The only change we make in the abstract syntax is that we annotate each variable with its (evaluated) type. Variables get annotated during evaluation, so the eval function needs the context to lookup the types; instead of having to pass both the context and the enviroment, we use only the context, adding values bindings to it. Type annotations are required in order to simplify cubical expressions: for example if the variable x has type $[i : \mathbb{I}][(i = 0) \mapsto a, (i = 1) \mapsto b]A$, then $x\ \mathtt{0}_{\mathbb{I}}$ shall reduce to a.

[1] As of now, it holds (inconsistently) that $\mathtt{U} : \mathtt{U}$.

2 Contexts and enviroments

The implementation uses two kinds of similar data structures to store information: a directions enviroment, which is only used to store bindings of the form $i \mapsto 0$, $i \mapsto 1$, $i \mapsto j$, and a context, which is used during typechecking and evaluation; a context stores definitions, declarations and values bindings.

Definition 2.1 (Context)**.**

1. $()$ *is the empty context;*
2. $\Gamma, x : A = B$ *extends the context* Γ *with a definition, where both the type* A *and the body* B *are terms;*
3. $\Gamma, x : A$ *extends the context* Γ *with a declaration, where the type* A *is a term;*
4. $\Gamma, x : A^{val}$ *extends the context* Γ *with a declaration, where the type* A *is a value;*
5. $\Gamma, x \mapsto t^{val}$ *extends the context* Γ *binding the variable* x *to the value* t*.*

Item (4) is used to optimize the code, since when type-checking a λ-abstraction of the form $[x : A]B$, A gets already evaluated by the type-checker, and it shall not be evaluated again later during the evaluation of the body B (this may happen when the eval function wants to know the type of x). Item (5) is used only in the evalutation.

Definition 2.2 (Directions enviroment)**.**

1. $()$ *is the empty directions enviroment;*
2. $\Theta, i \mapsto \mathtt{0}_{\mathbb{I}}$ *(or* $i \mapsto \mathtt{1}_{\mathbb{I}}$*) extends the directions enviroment* Θ *binding the name* i *to* $\mathtt{0}_{\mathbb{I}}$ *(or to* $\mathtt{1}_{\mathbb{I}}$*);*
3. $\Theta, i \mapsto j$ *extends the directions enviroment* Θ *binding the name* i *to the name* j *(diagonal).*

In the program the directions enviroment is kept distinct from the context because the former is implemented efficiently using a partitions list.

3 Evaluation

Given a context Γ and a directions enviroment Θ, the function $\textsc{eval}_{\Gamma;\Theta}$ gets a term as input and gives the corrisponding value as output. Instead of writing $\textsc{eval}_{\Gamma;\Theta}(t)$, we shall simply write $t_{\Gamma;\Theta}$. I denote with φ_Θ the substitution in the formula φ for the bindings stored in Θ.

$$
\begin{aligned}
x_{\Gamma;\Theta} &= x^{\textsc{lookupType}(\Gamma,x)} \\
U_{\Gamma;\Theta} &= U \\
([x : ty]e)_{\Gamma;\Theta} &= \langle x : ty_{\Gamma;\Theta}, e, (\Gamma;\Theta)\rangle \\
(fun\ a)_{\Gamma;\Theta} &= \textsc{app}(fun_{\Gamma;\Theta}, a_{\Gamma;\Theta}) \\
\mathtt{N}_{\Gamma;\Theta} &= \mathtt{N} \\
\mathtt{0}_{\Gamma;\Theta} &= \mathtt{0} \\
(\mathtt{S}n)_{\Gamma;\Theta} &= \mathtt{S}(n_{\Gamma;\Theta}) \\
(\mathtt{ind}\ F\ c_0\ c_s\ n)_{\Gamma;\Theta} &= \textsc{ind}(\mathtt{ind}\ F_{\Gamma;\Theta}\ (c_0)_{\Gamma;\Theta}\ (c_s)_{\Gamma;\Theta}\ n_{\Gamma;\Theta}) \\
\mathbb{I}_{\Gamma;\Theta} &= \mathbb{I} \\
(\mathtt{0}_{\mathbb{I}})_{\Gamma;\Theta} &= \mathtt{0}_{\mathbb{I}} \\
(\mathtt{1}_{\mathbb{I}})_{\Gamma;\Theta} &= \mathtt{1}_{\mathbb{I}} \\
([\varphi_1 \mapsto M_1, \ldots, \varphi_n \mapsto M_n])_{\Gamma;\Theta} &= \textsc{sys}([(\varphi_1)_\Theta \mapsto (M_1)_{\Gamma;\Theta}, \ldots, (\varphi_n)_\Theta \mapsto (M_n)_{\Gamma;\Theta}]) \\
([\varphi]A)_{\Gamma;\Theta} &= \textsc{partial}([\varphi_\Theta]A_{\Gamma;\Theta}) \\
([\varphi_1 \mapsto M_1, \ldots, \varphi_n \mapsto M_n]A)_{\Gamma;\Theta} &= [(\varphi_1)_\Theta \mapsto (M_1)_{\Gamma;\Theta}, \ldots, (\varphi_n)_\Theta \mapsto (M_n)_{\Gamma;\Theta}]A_{\Gamma;\Theta}
\end{aligned}
$$

The evalutation function uses some helpers:

- APP applied to fun^{val} and a^{val} first checks if fun^{val} is a closure; in that case in evaluates the closure, otherwise it checks if fun^{val} is a variable annotated with a type of the form $[i : I]A$ (where A may depend on i); if that's the case, it reduces the input to the appropriate value, otherwise it does nothing.
- IND evaluates an induction, pattern-matching on the argument n.
- SYS checks if in a system $[\varphi_1 \mapsto t_1^{val}, \ldots, \varphi_n \mapsto t_n^{val}]$ one of the φ_i's is true; if it finds one, lets say φ_i, then it returns t_i^{val}, otherwise it returns the system untouched.
- PARTIAL checks if the formula φ in a partial type $[\varphi]A^{val}$ is true and eventually returns just A^{val}.

4 Type checking/inference rules

The implementation uses a bidirectional type checker: I denote type inference with $e \Rightarrow t^{val}$ (the input is a term a and the output is a value t) and type checking with $e \Leftarrow t^{val}$ (both are inputs, where the term e is checked against value t).

4.1 Type inference

$$\frac{}{\Gamma;\Theta \vdash x \Rightarrow \textsc{lookupType}(\Gamma, x)} \quad \text{(Var)}$$

$$\frac{}{\Gamma;\Theta \vdash \mathtt{U} \Rightarrow \mathtt{U}} \quad \text{(Universe)}$$

$$\frac{\Gamma;\Theta \vdash f \Rightarrow \langle s : ty^{val}, e, \Gamma'; \Theta' \rangle \quad \Gamma;\Theta \vdash a \Leftarrow ty^{val}}{\Gamma;\Theta \vdash f\ a \Rightarrow \textsc{app}(\langle s : ty^{val}, e, \Gamma'; \Theta' \rangle, a_{\Gamma;\Theta})} \quad \text{(App)}$$

$$\frac{}{\Gamma;\Theta \vdash \mathtt{N} \Rightarrow \mathtt{U}} \quad \text{(Nat)}$$

$$\frac{}{\Gamma;\Theta \vdash \mathtt{0} \Rightarrow \mathtt{N}} \quad \text{(Nat-Zero)}$$

$$\frac{\Gamma;\Theta \vdash n \Leftarrow \mathtt{N}}{\Gamma;\Theta \vdash \mathtt{S}n \Rightarrow \mathtt{N}} \quad \text{(Nat-Succ)}$$

$$\frac{\begin{array}{cc}\Gamma;\Theta \vdash F \Leftarrow (\mathtt{N} \to U)_{();()} & \Gamma;\Theta \vdash c_0 \Leftarrow \textsc{app}(F_{\Gamma;\Theta}, \mathtt{0}) \\ \Gamma;\Theta \vdash n \Leftarrow \mathtt{N} & \Gamma;\Theta \vdash c_s \Leftarrow ([m : \mathtt{N}]Fm \to F(\mathtt{S}m))_{\Gamma;\Theta}\end{array}}{\Gamma;\Theta \vdash \mathtt{ind}\ F\ c_0\ c_s\ n \Rightarrow \textsc{app}(F_{\Gamma;\Theta}, n_{\Gamma;\Theta})} \quad \text{(Nat-Ind)}$$

$$\frac{}{\Gamma;\Theta \vdash \mathbb{I} \Rightarrow \mathtt{U}} \quad \text{(Interval)}$$

$$\frac{}{\Gamma;\Theta \vdash \mathtt{0}_{\mathbb{I}} \Rightarrow \mathbb{I}} \quad \text{(Interval-0)}$$

$$\frac{}{\Gamma;\Theta \vdash \mathtt{1}_{\mathbb{I}} \Rightarrow \mathbb{I}} \quad \text{(Interval-1)}$$

4.2 Type checking

$$\frac{\Gamma;\Theta \vdash t \Leftarrow \mathtt{U} \quad \Gamma, x : t;\Theta \vdash e \Leftarrow \mathtt{U}}{\Gamma;\Theta \vdash [x : t]e \Leftarrow \mathtt{U}} \quad \text{(Pi)}$$

$$\frac{\Gamma;\Theta \vdash ty \Leftarrow \mathtt{U} \quad ty_{\Gamma;\Theta} \sim_{\tau(\Gamma)} ty_1^{val} \quad \Gamma, x : ty_{\Gamma;\Theta} = v;\Theta \vdash e \Leftarrow (e_1)_{\Gamma_1, x_1 : ty_1^{val} = v;\Theta_1}}{\Gamma;\Theta \vdash [x : ty]e \Leftarrow \langle x_1 : ty_1^{val}, e_1, (\Gamma_1, \Theta_1) \rangle}\ (v = \textsc{newVar}(x, \Gamma \cup \Gamma_1))$$

(Lambda)

$$\frac{\Gamma, i : \mathbb{I};\Theta, i \mapsto i_1 \vdash e \Leftarrow (e_1)_{\Gamma, i_1 : \mathbb{I};\Theta_1}}{\Gamma;\Theta \vdash [i : \mathbb{I}]e \Leftarrow \langle i_1 : \mathbb{I}, e_1, (\Gamma_1, \Theta_1) \rangle} \quad \text{(Lambda-}\mathbb{I}\text{)}$$