LEOG

Conor Mc Bride

March 9, 2023

1 introduction

I'm trying to cook up a variant of the Calculus of (Inductive, etc) Constructions which is *bidirectional* in Curry style, and *observational* in that equality reflects what you can do with things, rather than how they are constructed (which makes a difference when the elimination form does not permit observation of construction).

The purpose of this note is simply to record what I think the rules are.

2 syntax

2.1 sorts

We have sorts **Prop** and $\mathbf{Type}_n$ for natural n. Administratively, we have a 'topsort' $\star$, which should appear only as the type in a checking judgement (i.e., morally, there is a separate type formation judgement) and as the upper bound in a cumulativity judgement (because a type of any sort is a type).

As in CIC, the sort **Prop** admits impredicative universal quantification.

2.2 canonical things

We have *atoms* `'atom`. We have pairs `[`s`|`t`]`, with the usual LISP right-bias convention that `[]` the syntacitcally valid name for the atom `'`, and `|[` may be omitted along with the matching `]`.

Invariance up to alpha-equivalence is quite helpful, so let us not use magical atoms for abstraction. Instead, let us have `\`x`.` t abstracting the *variable* x from t.

Given these ingredients, we can construct canonical forms such as `['Pi` S `\`x`.` T`]`. Note that canonical forms are not expected to make sense in and of themselves: the best we can hope is that sense is made of them. They will always be type*checked*.

Of course, once we have free variables, we have *noncanonical* things. They embed in the canonical things as e`{`q`}`, where e is noncanonical and q is a canonical proof that e fits where it's put. We may shorten e`{[]}` to e, as `[]` will always serve as the proof term when judgemental cumulativity/equality is enough.

2.3 noncanonical things

We have variables x within t for some `\`x`.` t. We also have e`-`s for noncanonical s, an eliminator appropriate to the type of e (e.g., if e is a function, s is its argument).

Finally, we have t`:`T, a *radical*, allowing us to annotate a canonical term with a canonical type, and thus form redexes.

3 typing awaiting computation

We shall have judgement forms for checking $T \ni t$ (*relying* on $\star \ni T$) and synthesis $e \in S$ (guaranteeing that $\star \ni S$). At some point, these will be closed appropriately under computation, but let's leave that to the next section.

3.1 type checking

For sorts w, u, we have

$$\frac{w > u}{w \ni u} \qquad \frac{}{\star > \mathbf{Type}_n} \qquad \frac{}{\star > \mathbf{Prop}} \qquad \frac{m > n}{\mathbf{Type}_m > \mathbf{Type}_n} \qquad \frac{}{\mathbf{Type}_n > \mathbf{Prop}}$$

I'm aware that the Coq tradition (from the days of impredicative **Set**) is to place **Prop** on a par with $\mathbf{Type}_0$ and below $\mathbf{Type}_1$, but the above formulation says yes to all those things and more. Besides, this system isn't called LEOG for nothing.

Meanwhile, for function types,

$$\frac{w \ni S \quad x \in S \vdash u \ni T}{u \ni \texttt{['Pi}\; S \; \backslash x.\, T\texttt{]}} \text{ where } w = \begin{cases} \star & \text{if } u = \mathbf{Prop} \\ u & \text{otherwise} \end{cases}$$

making **Prop** impredicative.

For functions, we have

$$\frac{x \in S \vdash T \ni t}{\texttt{['Pi}\; S \; \backslash x.\, T\texttt{]} \ni \backslash x.\, t}$$

On the checking side, that leaves us with only the shipment of synthesizable terms.

$$\frac{e \in S \quad \texttt{['Le}\; S\; T\texttt{]} \ni q}{T \ni e\{q\}}$$

where this `'Le` thing is *propositional cumulativity.* For all sorts u, because for $u = \mathbf{Prop}$

$$\frac{\star \ni S \quad \star \ni T}{u \ni \texttt{['Le}\; S\; T\texttt{]}}$$

Let us have

$$\frac{}{\texttt{['Le}\; \mathbf{Prop}\; \mathbf{Type}_n\texttt{]} \ni \texttt{[]}} \qquad \frac{i < j}{\texttt{['Le}\; \mathbf{Type}_i\; \mathbf{Type}_j\texttt{]} \ni \texttt{[]}}$$

and the (domain contravariant!)

$$\frac{\texttt{['Le}\; S'\; S\texttt{]} \ni \texttt{[]} \quad x \in S' \vdash \texttt{['Le}\; T\; T'\texttt{]} \ni \texttt{[]}}{\texttt{['Le ['Pi}\; S \; \backslash x.\, T\texttt{] ['Pi}\; S' \; \backslash x.\, T'\texttt{]]} \ni \texttt{[]}}$$

along with the inclusion of judgemental equality

$$\frac{\star \ni S \equiv T}{\texttt{['Le}\; S\; T\texttt{]} \ni \texttt{[]}}$$

Nontrivial proofs of cumulativity will arrive in due course.

3.2 type synthesis

I write a reverse turnstile to indicate an appeal to a local hypothesis, which is as close as I get to mentioning the context. Radicals are straightforward, thanks to the 'topsort'.

$$\frac{\dashv\ x \in S}{x \in S} \qquad \frac{\star \ni T \quad T \ni t}{t\!:\!T \in T} \qquad \frac{e \in \texttt{['Pi}\ S\ \backslash x\texttt{.}\ T\texttt{]} \quad S \ni s}{e\texttt{-}s \in T[s\!:\!S]}$$

Application is unremarkable, except to note that (i) there is no need to mention x by name when substituting it, because it is clear that it is x which has moved out of scope, and (ii) that s has to be radicalised before it can be substituted for x, from sheer syntactic compatibility and to create new redexes.