an Intensional Dependent Type Theory with Type-in-Type and Recursion

March 8, 2021

1 Examples

1.1 Pretending $\star =_\star \bot$

spoofing an equality

$(\lambda pr : (\star =_\star \bot) .pr\ (\lambda x.x)\ \bot \quad : \neg\star =_\star \bot)\, refl_{\star:\star}$

elaborates to

$(\lambda pr : (\star =_\star \bot) .pr\ (\lambda x.x)\ \bot \quad : \neg\star =_\star \bot)\, (refl_{\star:\star} :: (\star =_\star \star) =_l (\star =_\star \bot))$

$refl_{\star:\star} :: (\star =_\star \star) =_l (\star =_\star \bot)\ (\lambda x.x)\ \bot \quad :\bot$

$(\lambda C : (\star \to \star) .\lambda x : C\ \star .x :: (\Pi C : (\star \to \star) .C \star \to C \star) =_l (\Pi C : (\star \to \star) .C \star \to C\ \bot))\ (\lambda x.x)\ \bot :\bot$

$(\lambda x : \star .x :: (\star \to \star) =_{l,bod} (\star \to \bot))\ \bot \quad :\bot$

$(\bot :: \star =_{l,bod.bod} \bot) \quad :\bot$

note that the program has not yet "gotten stuck". to exercise this error, $\bot$ must be eliminated, this can be done by tying to summon another type by applying it to $\bot$

$((\bot :: \star =_{l,bod.bod} \bot) \quad :\bot)\ \star$

$((\Pi x : \star .x) :: \star =_{l,bod.bod} (\Pi x : \star .x))\ \star$

the computation is stuck, and the original application can be blamed on account that the "proof" has a discoverable type error at the point of application l

$\Pi C : (\star \to \star) .C \star \to \underline{C \star} \neq \Pi C : (\star \to \star) .C \star \to \underline{C\ \bot}$

when

$C \coloneqq \lambda x.x$

$C\ \bot = \bot \neq \star = C \star$

1.2 Pretending $true_c =_{\mathbb{B}_c} false_c$

spoofing an equality, evaluating $\neg true_c =_{\mathbb{B}_c} false_c$ with an incorrect proof.

$(\lambda pr : (\Pi C : (\mathbb{B}_c \to \star) .C\, true_c \to C\, false_c) .pr\ (\lambda b : \mathbb{B}_c.b \star \star \bot)\ \bot \quad : \neg true_c =_{\mathbb{B}_c} false_c)\, refl_{true_c:\mathbb{B}_c}$

is elaborated to

$(\lambda pr : (\Pi C : (\mathbb{B}_c \to \star) .C\, true_c \to C\, false_c) .pr\ (\lambda b : \mathbb{B}_c.b \star \star \bot)\ \bot \quad : \neg true_c =_{\mathbb{B}_c} false_c)\, (refl_{true_c:\mathbb{B}_c} ::$

$(refl_{true_c:\mathbb{B}_c} :: true_c =_{\mathbb{B}_c} true_c =_l true_c =_{\mathbb{B}_c} false_c)\ (\lambda b : \mathbb{B}_c.b \star \star \bot)\ \bot$

$((\lambda C : (\mathbb{B}_c \to \star) . \lambda x : C\, true_c . x) :: \Pi C : (\mathbb{B}_c \to \star) . C\, true_c \to C\, true_c =_l \Pi C : (\mathbb{B}_c \to \star) . C\, true_c \to C\, false$
$((. \lambda x : \star . x) :: \star \to \star =_{l,bod} \star \to \bot)\ \bot$
$(\bot :: \star =_{l,bod.bod} \bot)$

As in the above the example has not yet "gotten stuck". As above, applying $\star$ will discover the error, which would result in an error like

$\Pi C : (\mathbb{B}_c \to \star) . C\, true_c \to \underline{C\, true_c} \neq \Pi C : (\mathbb{B}_c \to \star) . C\, false_c \to \underline{C\, false_c}$

when

$C \coloneqq \lambda b : \mathbb{B}_c . b \star \star \bot$

$C\, true_c = \bot \neq \star = C\, false_c$

2 Data problems

The only way to induce a type soundness error is by feeding the functions more functions until they get stuck. This indirection means that we cannot say "$\underline{C \star} \neq \underline{C\ \bot}$ since $\star \neq \bot$" without giving a value to C. This means that we can't observe term level differences directly.

The solution is likely to add dependent data where the motive can observe type level issues via term computation and case elimination observes data constructor discrepancies.

consider some data with

$T : \mathbb{N} \to \mathbb{N} \to \mathbb{N} \to \star$

and some data constructor

$D : (x : \mathbb{N}) \to (y : \mathbb{N}) \to (z : \mathbb{N}) \to T\, x\, y\, z$

and an elimination like

$$case\ (D\, 0\, 1\, 0 : T\, 0\, 1\, 0) :: T\, 0\, 0\, 0 < - : T\, x\, y\, z . case\, x + y + z\, of\, 0 . \mathbb{N} \to \mathbb{N} \mid - . \star > of$$
$$D\, x\, y\, z . case\, x + y + z < ... > of\, 0 . \lambda x . x \mid - . \star$$

this term should have the apparent type of

$\mathbb{N} \to \mathbb{N}$

the goal is to track the problem to the 2nd position of T when the expression in normalized, by computing the motive at runtime.

While this problem seems easy we need to consider all possibilities for the syntax. For instance:

- $T : (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to \mathbb{N}) \to \star$, needs to support observing functions directly by application (rather then inspecting Π under assumption)
- $T : (\mathbb{N} * (\mathbb{N} * \mathbb{N})) \to \star$, data constructors can be embedded in types, so they should also be inspected
- $T : (A : \star) \to A \to A \to \star$, full dependent typing

Further we should expect that

- Normalization "preserves" the apparent type.
- the direct scrutinee is compatible with the typing of the motive.
  - for instance $case\ (D\,0\,1\,0\ :T\,0\,1\,0) :: T\,0\,0\,0\ <\underline{a} : T\,\underline{x}\,\underline{y}\,\underline{z}.\,M>$
    - Then the apparent type is $M\left[a \coloneqq (D\,0\,1\,0\ :T\,0\,1\,0) :: T\,0\,0\,0, x \coloneqq 0, \underline{y \coloneqq 0}, z \coloneqq 0\right]$ not $M\left[a \coloneqq (D\,0\,1\,0\ :T\,0\,1\,0) :: T\,0\,0\,0, x \coloneqq 0, \underline{y \coloneqq 1}, z \coloneqq 0\right]$

Finally not all casts will be available until the entire expression can be eliminated:

$$case\ ((\lambda x.\,(D\,0\,x\,0\ :T\,0\,x\,0))\,1) :: T\,0\,0\,0\ <x : T\,x\,y\,z.\,case\,x+y+z\,of\,0.\,\mathbb{N}\to\mathbb{N}\mid -.\star>\ of$$
$$D\,x\,y\,z.case\,x+y+z\ <...>\ of\,0.\,\lambda x.x\mid -.\star$$

At this point a number of mediocre solutions have been considered

2.1 Big Stack

One possibility is to expand the stack on elimination, interpolating between different positions

For instance, if $(D\,1\,2\,3\ :T\,1\,2\,3 :: T\,4\,5\,6 :: T\,7\,8\,9)$ could project into the motive $<\underline{a} : T\,\underline{x}\,\underline{y}\,\underline{z}.\,M>$ that will result in a stack of casts:

$$\begin{array}{llllllllll}
M[a \coloneqq & D\,1\,2\,3 : T\,1\,2\,3 :: T\,4\,5\,6 :: T\,7\,8\,9 & , x \coloneqq & 7 & , y \coloneqq & 8 & , z \coloneqq & 9 &] \\
M[a \coloneqq & D\,1\,2\,3 : T\,1\,2\,3 :: T\,4\,5\,6 :: T\,7\,8\,6 & , x \coloneqq & 7 & , y \coloneqq & 8 & , z \coloneqq & 6 &] \\
M[a \coloneqq & D\,1\,2\,3 : T\,1\,2\,3 :: T\,4\,5\,6 :: T\,7\,5\,6 & , x \coloneqq & 7 & , y \coloneqq & 5 & , z \coloneqq & 6 &] \\
M[a \coloneqq & D\,1\,2\,3 : T\,1\,2\,3 :: T\,4\,5\,6 & , x \coloneqq & 4 & , y \coloneqq & 5 & , z \coloneqq & 6 &] \\
M[a \coloneqq & D\,1\,2\,3 : T\,1\,2\,3 :: T\,4\,5\,3 & , x \coloneqq & 4 & , y \coloneqq & 5 & , z \coloneqq & 3 &] \\
M[a \coloneqq & D\,1\,2\,3 : T\,1\,2\,3 :: T\,4\,2\,3 & , x \coloneqq & 4 & , y \coloneqq & 2 & , z \coloneqq & 3 &] \\
M[a \coloneqq & D\,1\,2\,3 : T\,1\,2\,3 & , x \coloneqq & 1 & , y \coloneqq & 2 & , z \coloneqq & 3 &]
\end{array}$$

This method has the advantage that

- the apparent type of the scrutinee matches the collected type annotations
- the apparent type is stable under evaluation

Unfortunately

- this doesn't seem to naturally handle cases of nested type constructors or data constructors (these could also be expanded)
- this doesn't seem to handle cases of nested function type
- though each step is only linearly expands the stack of casts, it is easy to imagine these compounded linear expansions being far too slow

2.2 Remember and Peek

As an alternative to preemptively expanding the stack each time, we can thunk the information needed and then lazily expand the stack later, when needed.

It has all the downsides of the "big stack approach" except a little more efficient (but still very inefficient)

- doesn't seem to naturally handle cases of nested type type constructors or data constructors (these could also be expanded)
- doesn't seem to handle cases of nested function type
- the theory becomes very messy

An earlier version of the implementation used this, it was pretty buggy

2.3 "non-determinism"

Another alternative would to embed the possible non-determinism into the cast syntax. This seams closer to the intended meaning. So for instance, $(D\,1\,2\,3\;:T\,1\,2\,3 :: T\,4\,5\,6 :: T\,7\,8\,9)$ could project into the motive $<\underline{a} : T\,\underline{x}\,\underline{y}\,\underline{z}.\,M>$ with $M\left[a \coloneqq (D\,1\,2\,3\;:T\,1\,2\,3 :: T\,4\,5\,6 :: T\,7\,8\,9), x \coloneqq \{7,4,1\}, y \coloneqq \{8,5,\overline{2}\}, z \coloneqq \{9,6,3\}\right]$.

This approach has the advantage that terms are handled naturally, and

- it is easy to see how to project blame out of
  - functions $\{\lambda x.x, \lambda - .0\}\ 1$
  - data *case* $\{s\ (s\,0), 0\}$ ...
- it would be easy to increase the conservativity of runtime checking
- it seems possible to see how non-determinism can be optimized out of terms for some value syntax
  - $\{7,7,7\} = 7$
  - $\{\lambda x.x, \lambda - .0\} = \lambda x.\,\{x, 0\}$
  - $\{f\,a,\ f'\,a'\} \neq \{f\ f'\}\ \{a\,a'\}$ in general

However there are issues with this formulation

- Like casts, it seems best to fix the non-determinism against head terms, to avoid superfluous syntax.
  - such as $\{7, \{\{7, \{7,7\}\}, \{7,7\}\}\}$
- the type must be consistent over formalization this leads to something like a 2 dimensional terms
  - like
    - $\{7 : \mathbb{N} :: \mathbb{N}, 7 : \mathbb{N} :: \mathbb{B} :: \mathbb{N}, \mathit{true} : \mathbb{B} :: \mathbb{N}\}$

    - $\{7 : \mathbb{N} :: \mathbb{N}, 7 : \mathbb{N} :: \mathbb{B} :: \mathbb{N}\}$
    - but $\{7 : \mathbb{N}, \mathit{true} : \mathbb{B}\}$ should be disallowed

- motives don't compute straightforwardly,

$$\mathit{case}\ (D\,0\,1\,0\ :T\,0\,1\,0) :: T\,0\,0\,0\ <-: T\,x\,y\,z.\,\mathit{case}\,x+y+z\,\mathit{of}\,0.\,\mathbb{N}\to\mathbb{N}\mid -.\star>\ \mathit{of}$$
$$D\,x\,y\,z.\mathit{case}\,x+y+z\ <\ldots>\ \mathit{of}\,0.\,\lambda x.x\mid -.\star$$

  - has apparent type of $\mathbb{N}\to\mathbb{N}$
  - but after evaluation has apparent type $\mathit{case}\,\{0,0\}+\{0,1\}+\{0,0\}\,\mathit{of}\,0.\,\mathbb{N}\to\mathbb{N}\mid -.\star$
  - in this specific case the motive could produce a direct error but in general that is not possible,

$$a:\mathbb{N}\vdash \mathit{case}\ (D\,0\,a\,0\ :T\,0\,a\,0) :: T\,0\,0\,0\ <-: T\,x\,y\,z.\,\mathit{case}\,x+y+z\,\mathit{of}\,0.\,\mathbb{N}\to\mathbb{N}\mid -.\star>\ \mathit{of}$$
$$D\,x\,y\,z.\mathit{case}\,x+y+z\,\mathit{of}\,0.\,\lambda x.x\mid -.\star$$

    - after evaluation has apparent type $\mathit{case}\,\{0,0\}+\{0,a\}+\{0,0\}\,\mathit{of}\,0.\,\mathbb{N}\to\mathbb{N}\mid -.\star$
  - this suggests that the top of the casts need to have some "special" behavior that allows computation for the purposes of typing but retains the uncertainty if a term of that type is eliminated for instance $\mathit{case}\,\{\underline{0},0\}+\{\underline{0},a\}+\{\underline{0},0\}\,\mathit{of}\,0.\,\mathbb{N}\to\mathbb{N}\mid -.\star$ should in some way be equivalent to $\{\mathbb{N}\to\mathbb{N}, \mathit{case}\,\{0,0\}+\{0,a\}+\{0,0\}\,\mathit{of}\,0.\,\mathbb{N}\to\mathbb{N}\mid -.\star\}$

- also the motive scrutinee is not necessarily type compatible as it was in the stack case

Currently this is the closest to the prototype implementation

3 Proof summery

elaboration produces a well typed term

type soundness goes through as before but, all head constructors match or blame is available in an empty ctx

4 TODO

- syntax and rules
- Example of why function types alone are underwhelming
  - pair,singleton
  - walk through of various examples
- theorem statements

  - substitution!

- proofs
  - top level summery!!
  - consistency is a little too restricted
  - clean up, the first type cast is unneeded and dropping it might make induction easier?
  - consider separating annotation evaluation and term evaluation
- more specifics about var binding
  - presented in an almost capture avoiding way
  - more explicit about the connection to heterogeneous substitution
- exposition
- archive
- paper target

5 Scratch

5.1 pretending $\lambda x.x =_{\mathbb{B}_c \to \mathbb{B}_c} \lambda x.true_c$

a difference can be observed via
$(\lambda pr : (\Pi C : ((\mathbb{B}_c \to \mathbb{B}_c) \to \star).C\ (\lambda x.x) \to C\ (\lambda x.true_c)).pr\ (\lambda f : \mathbb{B}_c \to \mathbb{B}_c.f\ \star\ \star\ \bot)\ \bot \quad : \neg \lambda x.x =_{\mathbb{B}_c \to \mathbb{B}}$
...
$S_{a:A} \coloneqq a$
$S_{a:A} \coloneqq \Pi P : A \to \star.P\,a \to \star$
..
$\neg\star =_\star (\star \to \star)$ is provable?
$\lambda pr : (\Pi C : (\star \to \star).C\,Unit \to C\ \bot).pr\ (\lambda x.x)\ \bot \quad : \neg\star =_\star \bot$
.
evaluating $\neg true_c =_{\mathbb{B}_c} false_c$ with an incorrect proof.
$(\lambda pr : (\Pi C : (\mathbb{B}_c \to \star).C\,true_c \to C\,false_c).pr\ (\lambda b : \mathbb{B}_c.b\ \star\ Unit\ \bot)\ tt \quad : \neg true_c =_{\mathbb{B}_c} false_c)\,refl_{true_c:\mathbb{B}}$
is elaborated to
$(\lambda pr : (\Pi C : (\mathbb{B}_c \to \star).C\,true_c \to C\,false_c).pr\ (\lambda b : \mathbb{B}_c.b\ \star\ Unit\ \bot)\ tt \quad : \neg true_c =_{\mathbb{B}_c} false_c)\,(refl_{true_c:}$
$\rightsquigarrow ((refl_{true_c:\mathbb{B}_c} :: true_c =_{\mathbb{B}_c} true_c =_l true_c =_{\mathbb{B}_c} false_c)\ (\lambda b : \mathbb{B}_c.b\ \star\ Unit\ \bot)\ tt \quad :\bot)$
de-sugars to
$(((\lambda C : (\mathbb{B}_c \to \star).\lambda x : C\,true_c.x) :: (\Pi C : (\mathbb{B}_c \to \star).C\,true_c \to C\,true_c) =_l (\Pi C : (\mathbb{B}_c \to \star).C\,true_c \to C\,fa$
$\rightsquigarrow (((\lambda x : (true_c\ \star\ Unit\ \bot).x) :: (true_c\ \star\ Unit\ \bot \to true_c\ \star\ Unit\ \bot) =_{l,bod} ((true_c\ \star\ Unit\ \bot) \to false_c\ \star$
def eq to
$(((\lambda x : Unit.x) :: (Unit \to Unit) =_{l,bod} (Unit \to \bot))\ tt \quad :\bot)$
$\rightsquigarrow (tt :: Unit =_{l,bod.bod} \bot \quad :\bot)$

note that the program has not yet "gotten stuck". to exercise this error, $\bot$ must be eliminated, this can be done by tying to summon another type by applying it to $\bot$

$(tt :: Unit =_{l,bod.bod} \bot \quad :\bot)\ \bot \quad :\bot$

bad attempt

$(tt :: Unit =_{l,bod.bod} \bot \quad :\bot)\ \bot \quad :\bot$

de-sugars to

$((\lambda A : \star.\lambda a : A.a) :: \Pi A : \star.A \to A =_{l,bod.bod} \Pi x : \star.x \quad :\bot)\ \bot \quad :\bot$

$\rightsquigarrow (.\lambda a.a) ::\bot\to\bot=_{l,bod.bod.bod} \bot \quad :\bot$

but still

$((.\lambda a.a) ::\bot\to\bot=_{l,bod.bod.bod} \bot \quad :\bot)\star$

$((.\lambda a.a) ::\bot\to\bot=_{l,bod.bod.bod} \Pi x : \star.x \quad :\bot)\star$

$(\star :: \star =_{l,bod.bod.bod.aty} \bot =_{l,bod.bod.bod.bod} \star)$

bad attempt

$(tt :: Unit =_{l,bod.bod} \bot \quad :\bot)\star \to \star \quad : \star \to \star$

de-sugars to

$((\lambda A : \star.\lambda a : A.a) :: \Pi A : \star.A \to A =_{l,bod.bod} \Pi x : \star.x \quad :\bot)\star \to \star \quad : \star \to \star$

$\rightsquigarrow (.\lambda a.a) :: (\star \to \star) \to (\star \to \star) =_{l,bod.bod.bod} \star \to \star \quad : \star \to \star$

not yet "gotten stuck"

$\rightsquigarrow (.\lambda a.a) :: (\star \to \star) \to (\star \to \star) =_{l,bod.bod.bod} \star \to \star \quad : \star \to \star$

bad attempt

de-sugars to

$((\lambda A : \star.\lambda a : A.a) :: \Pi A : \star.A \to A =_{l,bod.bod} \Pi x : \star.x \quad :\bot)\,\mathbb{B}_c \quad : \mathbb{B}_c$

$\rightsquigarrow (.\lambda a : \mathbb{B}_c.a) :: \mathbb{B}_c \to \mathbb{B}_c =_{l,bod.bod.bod} \mathbb{B}_c \quad : \mathbb{B}_c$

attempt

$(tt :: Unit =_{l,bod.bod} \bot \quad :\bot)\,\mathbb{B}_c \quad : \mathbb{B}_c$

de-sugars to

$((\lambda A : \star.\lambda a : A.a) :: \Pi A : \star.A \to A =_{l,bod.bod} \Pi x : \star.x \quad :\bot)\,\mathbb{B}_c \quad : \mathbb{B}_c$

$\rightsquigarrow (.\lambda a : \mathbb{B}_c.a) :: \mathbb{B}_c \to \mathbb{B}_c =_{l,bod.bod.bod} \mathbb{B}_c \quad : \mathbb{B}_c$

.

.

.

.

.

.