Implementing Martin-Löf's Meaning Explanations for Intuitionistic Type Theory in Agda

Peter Dybjer[1] and Anton Setzer[2]

[1] Dept. of Computer Science and Engineering, University of Gothenburg and Chalmers University of Technology, Gothenburg, Sweden
http://www.cse.chalmers.se/~peterd/
[2] Dept. of Computer Science, Swansea University, Swansea, UK
http://www.cs.swan.ac.uk/~csetzer/

Types 2022 (Nantes, France)
21 June 2022

Meaning Explanations of Type Theory

- Introduced by Martin-Löf in his "Constructive mathematics and computer programming" article [1].
  - Adapted version in the Bibliopolis book [2].
- An attempt to formalise the reasons why we believe in the correctness of type theory.
- Because of Gödel's incompleteness theorem only a philosophical argument is possible.
- Mathematically it is a PER model of type theory, but the intention is a philosophical correctness argument.
- **1st Goal:** formalise Meaning Explanations in Agda in order to work out full details.
- **2nd Goal:** Formalise a type theory with access to the set of untyped terms.

Disclaimer

- In discussion with Peter Dybjer we found:
  - We (and probably everybody else in this area) disagree on everything philosophically.
  - We agree on everything technical.

Raw Terms (in n Variables)

```
data Tm : ℕ → Set where
  var    : {n : ℕ} → Fin n → Tm n
  _#_    : {n : ℕ} → Tm n → Tm n → Tm n
  λ'     : {n : ℕ} → Tm (succ n) → Tm n
  zero   : {n : ℕ} → Tm n
  succ   : {n : ℕ} → Tm n → Tm n
  rec    : {n : ℕ} → Tm n → Tm n → Tm n → Tm n
  nat    : {n : ℕ} → Tm n
  _→'_   : {n : ℕ} → Tm n → Tm n → Tm n
  Π      : {n : ℕ} → Tm n → Tm (succ n) → Tm n
K-comb : Tm 0
K-comb = λ' (λ' x2)
           where x2 = var (suc zer)

appExample : Tm 0
appExample = (K-comb # zero) # (succ zero)
```

Raw Canonical Terms and Contexts (Weak Head Normalform)

```
data CanTm : Set where
  λc      : Tm 1 → CanTm
  zeroc   : CanTm
  succc   : Tm 0 → CanTm
  natc    : CanTm
  _→c_    : Tm 0 → Tm 0 → CanTm
  Πc      : Tm 0 → Tm 1 → CanTm
```

```
data Ctx : (n : ℕ) → Set where
  <> : Ctx 0
  <_,_> : {n : ℕ} → Ctx n → Tm n → Ctx (succ n)
```

Homomorphisms (Substitutions – n variables into Tm m)

```
data Hom : (m n : ℕ) → Set where
  emptysub : {m : ℕ}     → Hom m 0
  conssub  : {m n : ℕ} → Hom m n → Tm m → Hom m (succ n)

_[_] : {n m : ℕ} → Tm n → Hom m n → Tm m

_∘s_ : {m n p : ℕ} → Hom n m → Hom p n → Hom p m

Env : (n : ℕ) → Set
Env n = Hom 0 n

_[_]1 : Tm 1 → Tm 0 → Tm 0
```

Reductions (Big Step Semantics, Lazy Evaluation)

```
data _=>_ : Tm 0 → CanTm → Set where
  β : {c a : Tm 0} → {b : Tm 1} → {v : CanTm}
      → c => λc b
      → (b [ a ]1) => v
      → (c # a) => v
  λcp     : {b : Tm 1} → λ' b => λc b
  zerocp : zero => zeroc
  succcp : {a : Tm 0} → succ a => succc a
  rec-zero : {c d e : Tm 0} → {v : CanTm}
               → c => zeroc
               → d => v
               → rec d e c => v
```

Reductions (Cont.)

$$
\begin{aligned}
&\mathsf{rec\text{-}succ} : \{a\ c\ d\ e : \mathsf{Tm}\ 0\} \to \{v : \mathsf{CanTm}\} \\
&\qquad \to c \Rightarrow \mathsf{succc}\ a \\
&\qquad \to e \#\ a \#\ (\mathsf{rec}\ d\ e\ a) \Rightarrow v \\
&\qquad \to \mathsf{rec}\ d\ e\ c \Rightarrow v \\
&\mathsf{natcp} \quad : \mathsf{nat} \Rightarrow \mathsf{natc} \\
&\mathsf{arrowcp} : \{a\ b : \mathsf{Tm}\ 0\} \to (a \to' b) \Rightarrow (a \to \mathsf{c}\ b) \\
&\Pi\mathsf{cp} \quad : \{a : \mathsf{Tm}\ 0\} \to \{b : \mathsf{Tm}\ 1\} \to \Pi\ a\ b \Rightarrow \Pi\mathsf{c}\ a\ b
\end{aligned}
$$

Non-canonical Elements from a Collection of Canonical Elements

```
record _∈_ (a : Tm 0 ) (A : CanTm → Set) : Set where
  constructor <_,_,_>
  field can  : CanTm
        red  : a => can
        canp : A can
```

Canonical Elements of Nat and Π

```
data Nat : CanTm → Set where
  zeroN : Nat zeroc
  succN : {b : Tm 0} → b ∈ Nat → Nat (succc b)

data Pi (A : Tm 0 → Set)
        (B : (x : Tm 0) → A x → Tm 0 → Set) : CanTm → Set where
  λPi : (b : Tm 1)
        → ((x : Tm 0) → (xA : A x) → B x xA (b [ x ]1))
        → Pi A B (λc b)
```

```
mutual
  data Ty : CanTm → Set where
    natTy   : Ty natc
    arrowTy : (A B : Tm 0) → A ∈ Ty → B ∈ Ty → Ty (A →c B)
    ΠTy     : (A : Tm 0)(B : Tm 1) (ATy : A ∈ Ty)
              → ((x : Tm 0)
                   → x ∈ El (ATy .can) (ATy .canp)
                   → ( B [ x ]₁) ∈ Ty )
              → Ty (Πc A B)

  El : (A : CanTm) → Ty A → CanTm → Set
  El natc natTy = Nat
  El (A →c B) (arrowTy .A .B ATy BTy)
     = Arrow (λ x → x ∈ El (ATy .can) (ATy .canp))
             (λ y → y ∈ El (BTy .can) (BTy .canp))
  El (Πc A B) (ΠTy .A .B ATy BTy)
     = Pi (λ x → x ∈ El (ATy .can) (ATy .canp))
          (λ x xA y → y ∈ El (BTy x xA .can) (BTy x xA .canp))
```

Contexts and Environments

- Meaning of $\vdash \Gamma$ Context

$$
\begin{aligned}
&\text{is-Ctx} : \{n : \mathbb{N}\} \to \text{Ctx } n \to \text{Set} \\
&\text{is-Ctx } <> = \top \\
&\text{is-Ctx } \{\text{succ } n\} < \Gamma\ ,\ A > = \\
&\qquad \Sigma[\ \Gamma\text{-}ty \in \text{is-Ctx } \Gamma\ ] \\
&\qquad ((\gamma : \text{Env } n) \to \text{is-Env } \Gamma\ \Gamma\text{-}ty\ \gamma \to (A\ [\ \gamma\ ]) \in \text{Ty})
\end{aligned}
$$

- Meaning of $\vdash \gamma : \Gamma$

$$
\begin{aligned}
&\text{is-Env} : \{n : \mathbb{N}\}(\Gamma : \text{Ctx } n)(\Gamma\text{-}ty : \text{is-Ctx } \Gamma)(\gamma : \text{Env } n) \to \text{Set} \\
&\text{is-Env } <>\ \Gamma\text{-}ty\ \text{emptysub} = \top \\
&\text{is-Env } < \Gamma\ ,\ A >\ (\Gamma\text{-}ty\ ,\ A\text{-}ty)\ (\text{conssub } \gamma\ a) = \\
&\qquad \Sigma[\ \gamma\text{-}t \in \text{is-Env } \Gamma\ \Gamma\text{-}ty\ \gamma\ ] \\
&\qquad (a \in\!< A\ [\ \gamma\ ]\ ,\ A\text{-}ty\ \gamma\ \gamma\text{-}t >)
\end{aligned}
$$

Meaning of Judgments

- Meaning of $\Gamma \vdash A\ \mathrm{Type}$

$$\begin{aligned}&\mathsf{is\text{-}Ty} : \{n : \mathbb{N}\}\ (\Gamma : \mathsf{Ctx}\ n)\ (\Gamma\text{-}ty : \mathsf{is\text{-}Ctx}\ \Gamma)\ (A : \mathsf{Tm}\ n) \to \mathsf{Set}\\ &\mathsf{is\text{-}Ty}\ \{n\}\ \Gamma\ \Gamma\text{-}ty\ A = (\gamma : \mathsf{Env}\ n) \to \mathsf{is\text{-}Env}\ \Gamma\ \Gamma\text{-}ty\ \gamma\\ &\qquad\qquad\qquad\qquad \to (A\ [\ \gamma\ ]) \in \mathsf{Ty}\end{aligned}$$

- Meaning of $\Gamma \vdash a : A$

$$\begin{aligned}&\mathsf{is\text{-}Tm} : \{n : \mathbb{N}\}(\Gamma : \mathsf{Ctx}\ n)(\Gamma\text{-}ty : \mathsf{is\text{-}Ctx}\ \Gamma)(A : \mathsf{Tm}\ n)\\ &\qquad \to \mathsf{is\text{-}Ty}\ \Gamma\ \Gamma\text{-}ty\ A \to \mathsf{Tm}\ n \to \mathsf{Set}\\ &\mathsf{is\text{-}Tm}\ \{n\}\ \Gamma\ \Gamma\text{-}ty\ A\ A\text{-}ty\ a =\\ &\quad (\gamma : \mathsf{Env}\ n)\ (\gamma\text{-}t : \mathsf{is\text{-}Env}\ \Gamma\ \Gamma\text{-}ty\ \gamma)\\ &\quad \to (a\ [\ \gamma\ ]) \in\!< A\ [\ \gamma\ ]\ ,\ A\text{-}ty\ \gamma\ \gamma\text{-}t >\end{aligned}$$

Meaning of Judgments

- Meaning of $\Delta \vdash \gamma : \Gamma$

$$
\begin{aligned}
&\text{is-Hom} : \{m\ n : \mathbb{N}\}(\Delta : \text{Ctx}\ m)\ (\Gamma : \text{Ctx}\ n) \\
&\qquad \to \text{is-Ctx}\ \Delta \to \text{is-Ctx}\ \Gamma \to \text{Hom}\ m\ n \to \text{Set} \\
&\text{is-Hom}\ \{m\}\ \{n\}\ \Delta\ \Gamma\ \Delta\text{-}ty\ \Gamma\text{-}ty\ \gamma = \\
&\quad (\rho : \text{Env}\ m)(\rho\text{-}t : \text{is-Env}\ \Delta\ \Delta\text{-}ty\ \rho) \to \text{is-Env}\ \Gamma\ \Gamma\text{-}ty\ (\gamma \circ \text{s}\ \rho)
\end{aligned}
$$

Formation and Introduction Rules for $\mathbb{N}$

- Justification of Formation Rule:

$$\begin{aligned}&\text{nat-in-Ty} : \text{nat} \in \text{Ty}\\&\text{nat-in-Ty} = < \text{natc} , \text{natcp} , \text{natTy} >\end{aligned}$$

- Justification of Introduction Rules:

$$\begin{aligned}&\text{zero-in-nat} : \text{zero} \in< \text{nat} , \text{nat-in-Ty} >\\&\text{zero-in-nat} = < \text{zeroc} , \text{zerocp} , \text{zeroN} >\end{aligned}$$

$$\begin{aligned}&\text{succ-in-nat} : (a : \text{Tm } 0) \to a \in< \text{nat} , \text{nat-in-Ty} >\\&\qquad\qquad \to \text{succ } a \in< \text{nat} , \text{nat-in-Ty} >\\&\text{succ-in-nat } a \; a\text{-}in\text{-}nat = < \text{succc } a , \text{succcp} , \text{succN } a\text{-}in\text{-}nat >\end{aligned}$$

Elimination Rule for $\mathbb{N}$

$$
\begin{aligned}
&\text{rec-t} : (C\ e\ d\ c : \mathsf{Tm}\ 0) \\
&\quad \to (C\text{-}ty : C \in \mathsf{Ty}) \\
&\quad \to e \in\!< (\mathsf{nat} \to' (C \to' C)) , \\
&\qquad\qquad (\mathsf{arrow\text{-}in\text{-}Ty}\ \mathsf{nat}\ (C \to' C)\ \mathsf{nat\text{-}in\text{-}Ty} \\
&\qquad\qquad (\mathsf{arrow\text{-}in\text{-}Ty}\ C\ C\ C\text{-}ty\ C\text{-}ty)) > \\
&\quad \to d \in\!< C , C\text{-}ty > \\
&\quad \to c \in\!< \mathsf{nat} , \mathsf{nat\text{-}in\text{-}Ty} > \\
&\quad \to \mathsf{rec}\ d\ e\ c \in\!< C , C\text{-}ty >
\end{aligned}
$$

Elimination Rule for $\mathbb{N}$

```
rec-t C e d c C-ty e-t d-t < .zeroc , czero , zeroN >
  = < d-t .can , rec-zero czero (d-t .red) , d-t .canp >
rec-t C e d c C-ty e-t d-t < succc a , csucca , succN a-in-nat >
  = let
      ih : rec d e a ∈< C , C-ty >
      ih = rec-t C e d a C-ty e-t d-t a-in-nat
      ear∈C : (e # a # rec d e a) ∈< C , C-ty >
      ear∈C = #-t C C (e # a) (rec d e a) C-ty C-ty
                (#-t nat (C →' C) e a nat-in-Ty
                (arrow-in-Ty C C C-ty C-ty) e-t a-in-nat) ih
    in < ear∈C .can , rec-succ csucca (ear∈C .red) , ear∈C .canp >
```

Conclusion

- Formalisation of meaning explanations in Agda.
- Substantial work needed in order to get it precise.
  - Definition of raw terms, canonical terms, substitutions.
  - Meaning of judgements.
- Auxiliary goal: Develop a type theory containing partial terms.

Bibliography I


P. Martin-Löf.
Constructive mathematics and computer programming.
In L. J. Cohen, J. Łoś, H. Pfeiffer, and K.-P. Podewski, editors, Logic, Methodology and Philosophy of Science VI, Proceedings of the Sixth International Congress of Logic, Methodology and Philosophy of Science, Hannover 1979, volume 104 of Studies in Logic and the Foundations of Mathematics, pages 153–175. North-Holland, 1982.


P. Martin-Löf.
Intuitionistic type theory, volume 1 of Studies in Proof Theory.
Bibliopolis, 1984.