
Partial Recursive Functions in Martin-Löf Type Theory

Anton Setzer
(University of Wales Swansea)

1. Introduction.
2. Inductive-Recursive Definitions.
3. Restrictions on Inductive-Recursive Definitions.
4. Conclusion.

1. Introduction

- **Problem:**

- All functions in Martin-Löf type theory (MLTT) terminate.
- Otherwise we would obtain non-terminating and therefore possibly false proofs.
- Need to reason about possibly non-terminating partial recursive functions, e.g. in order to carry out correctness proofs.

- **Solution:** Based on approach by Bove/Capretta.

Notations



$$\underline{(x : A) \rightarrow B}$$

= type of total functions f , mapping $a : A$ to $f(a) : B[x := a]$.



Abbreviation:

We write $(x : A, y : B) \rightarrow C$ for $(x : A) \rightarrow (y : B) \rightarrow C$.



$$\underline{(x : A) \rightharpoonup B}$$

is the intended type of partial recursive functions mapping $a : A$ possibly to $f(a) : B[x := a]$.

Repres. of Part. Rec. Functions

- Consider

$$\begin{aligned} f & : \mathbb{N} \rightarrow \mathbb{N} \\ f(0) & \simeq 0 \\ f(n+1) & \simeq f(f(n)) \end{aligned}$$

- Done by defining

$$\begin{aligned} f(\cdot)\downarrow & : \mathbb{N} \rightarrow \text{Set} && \text{domain of } f \\ \text{eval}_f & : (n : \mathbb{N}, p : f(n)\downarrow) \rightarrow \mathbb{N} && \text{result of } f(n) \end{aligned}$$

- We have the following constructors and equality rules:

$$C_0 : f(0)\downarrow \quad \text{eval}_f(0, C_0) = 0$$

$$C_S : (n : \mathbb{N}, p : f(n)\downarrow, q : f(\text{eval}_f(n, p))\downarrow) \rightarrow f(n+1)\downarrow$$

$$\text{eval}_f(n+1, C_S(n, p, q)) = \text{eval}_f(\text{eval}_f(n, p), q)$$

2. Inductive-Recursive Definitions

- C_S is no longer a constructor of a strictly positive inductive definition:

$$C_S : (n : \mathbb{N}, p : f(n) \downarrow, q : f(\text{eval}_f(n, p)) \downarrow) \rightarrow f(n + 1) \downarrow$$

The type of q depends on $\text{eval}_f(n, p)$.

- This is an example of an indexed inductive-recursive definition.

Inductive Definitions

- Without dependent types, strictly positive inductive definitions are given as sets A with constructors of the form

$$C : A_1 \rightarrow A_2 \rightarrow \dots \rightarrow A_n \rightarrow A$$

where

- A_i are defined before introducing A ,
 - called an non-inductive argument,
- or A_i is of the form $B_1 \rightarrow \dots \rightarrow B_n \rightarrow A$,
 - called an inductive argument.

Ind. Defs. in Dep. Type Theory

- In dependent type theory, A_i might depend on previous arguments $a_j : A_j$:

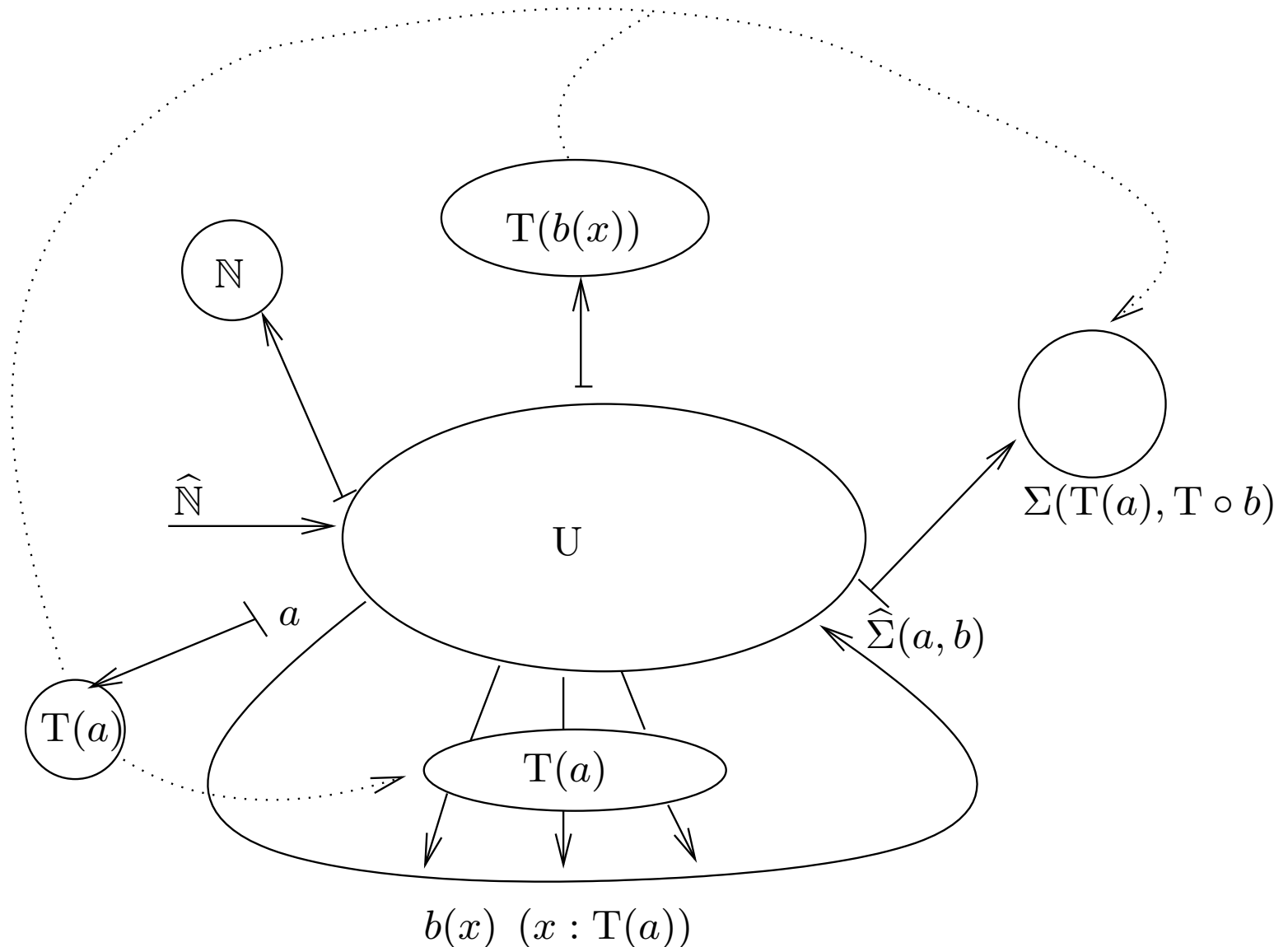
$$C : (a_1 : A_1, a_2 : A_2, \dots, a_n : A_n) \rightarrow A$$

- But a_j can only depend on previous **non-inductive arguments**.
 - When defining the constructors for A , we don't know anything about A .
So we cannot define any set depending directly on an inductive-argument $b : B_1 \rightarrow \dots \rightarrow B_n \rightarrow A$.

Inductive-Recursive Definitions

- In inductive-recursive definitions (IRD) we define **simultaneously**
 - **inductively** a set $U : \text{Set}$
 - and for $a : U$ **recursively** an element $T(a) : D$ for some type D .
- The type of later arguments can now depend on T applied to previous **inductive arguments**.

Example: A Universe



Indexed Ind.-Rec. Definitions

- Indexed Inductive-Recursive Definitions (IIRD)
extend IRD by allowing to define
 - simultaneously for $i : I$ inductively sets

$$U(i) : \text{Set}$$

while recursively defining for $a : U(i)$ an element

$$T(i, a) : D[i]$$

where $D[i]$ is a type depending on $i : I$.

Restricted vs. Generalised IIRD

- Restricted IIRD: for every $i : I$ we have a constructor

$$C(i) : A(i) \rightarrow U(i)$$

- Generalised IIRD: we have constructors

$$C : (a : A) \rightarrow U(i(a)).$$

- **Example revisited:**

- If we consider C_S above, it is a constructor of an IIRD

$$C_S : (n : \mathbb{N}, p : f(n) \downarrow, q : f(\text{eval}_f(n, p)) \downarrow) \rightarrow f(n + 1) \downarrow$$

$$\text{eval}_f(n + 1, C_S(n, p, q)) = \text{eval}_f(\text{eval}_f(n, p), q)$$

- An example of a generalised IIRD.

- But it can be transformed into a restricted IIRD.

3. Restrictions on IIRD

- We want to take IIRD as representations of partial recursive functions.
- Which IIRD correspond to partial recursive functions?
 - Elements of $(a : A) \rightarrow B(a)$ correspond to an IIRD of type

$$f(\cdot)\downarrow : A \rightarrow \text{Set} \quad \text{eval}_f : (a : A, p : f(a)\downarrow) \rightarrow B(a)$$

So

$$I = A \quad D[a] = B(a)$$

- It should be a **restricted** IIRD, otherwise we possibly get multivalued functions.
- If we had **non-inductive arguments**, we would obtain possibly multivalued functions.
- Therefore we **disallow** non-inductive arguments.

IIRD Corresp. to Part.-Rec. Func.

- An inductive-argument of the form

$$(c : C) \rightarrow f(a(c)) \downarrow$$

would correspond to recursive calls of $f(a(c))$ for all $c : C$, which is non-computable if C is infinite.

- All restricted IIRD of the above type
 - without non-inductive arguments,
 - only singleton inductive-argumentscorrespond to partial recursive functions.
- Such IIRD correspond directly to recursive equations with dependencies.

Completeness

- The above doesn't allow references to previously defined partial-recursive functions.
- Two approaches:
 - Define simultaneously all functions used in the buildup of one partial recursive functions.
 - Allow specific non-inductive arguments referring to the domain of previously defined partial recursive functions.
- With both approaches we obtain
 - a set of partial rec.-functions
 - closed under the standard constructions of partial recursive functions,
 - and under total functions.

Data Type of IIRD

- There is a data type for codes for IIRD.
- If we restrict it to IIRD corresponding to IIRD we obtain a type of codes for partial recursive functions

$$\text{Rec}_{A,B}^+$$

which represents

$$(x : A) \multimap B(x)$$

4. Conclusion

- Representation of partial rec. functions by IIRD.
- Data type $\text{Rec}_{A,B}^+$.
- However $\text{Rec}_{A,B}^+$ is a true type (no longer a small type, called set in MLTT).
 - Happens, since $\text{Rec}_{A,B}^+$ refers to $(c : C) \multimap D$ for arbitrary sets C, D .
 - If we restrict this approach to sets C, D which are elements of a universe, $\text{Rec}_{A,B}^+$ is a set.
 - This is the case, e.g. if we consider functions $\mathbb{N}^n \multimap \mathbb{N}$.