# M2 LMFI – SOFIX:
## Second-order quantification and fixed-points in logic
# First lecture: Gödel's System T
(preliminary version of the 27/12/2022)

Alexis Saurin

january 2023

## Contents

# 1 Preliminary and motivating remarks

## 1.1 On the weak expressiveness of the simply typed $\lambda$-calculus

Simply typed lambda-calculus (STLC) has good properties but a poor expressiveness:

— due to strong normalization, only total recursive functions can be represented, of course. That is a feature of the calculus, but due to the properties of the type system and the strong normalization proof, some total recursive functions cannot be represented. Actually **lots** of them cannot be represented...

— when typing the encoding of pairs, there were constraints on types: for types $A, B, C$, *paire* has type $A \to (B \to ((A \to (B \to C)) \to C))$. That is, given $t : A$ and $u : B$, $(paire)tu$ had type $(A \to (B \to C)) \to C$ Therefore, unless $A = B$, one cannot find projections with the expected types: it is not possible, in the typed version of the pair encoding, to access the components of the pair...

— for arithmetical functions, there were also strong restrictions: given a base type $o$, and writing $[0] = o$ and $[n+1] = [n] \to [n]$, one saw that every $n \geq 2$ allows to type Church numerals. In Church's style $\lambda$-calculus, one can type addition, product with the expected types $[n] \to [n] \to [n]$ for any $n \geq 2$, but exponentiation cannot be typed with such a type... one has to use types of different levels for $a$ and $b$ in $a^b$: there are restrictions of the typed use of iteration.

More precisely, Schwichtenberg and Statman proved that the expressible functions of type $\mathsf{Nat}^k \to \mathsf{Nat}$ (with $\mathsf{Nat} = [2]$) are exactly the ***extended polynomials***:

**Definition 1.1**

> **Extended polynomials** *are the functions generated by* 0, 1,*the identity function as well as the operations of addition, multiplication and conditional.*

**Theorem 1.2 (*Schwichtenberg and Statman*)**

> *The arithmetical functions definable in simply-typed $\lambda$-calculus over type* $\mathsf{Nat}$ *are exactly the extended polynomials.*

If one relaxes the type for natural numbers to be some type for Church numeral, *ie.* allowing to define function as $\lambda$-terms of type $[n+2] \to \ldots ([n+2] \to [n+2])$ for $n \geq 0$, then one can define more functions in simply-typed $\lambda$-calculus. In particular, the predecessor function and the exponentiation are now definable.

But there is still a big gap. For instance, one can represent neither the equality predicate, nor the less-than predicate (*ie.* their characteristic functions) nor the subtraction function...

Several solutions are available to improve this expressiveness issue:

— We shall now consider an option investigated by Gödel, extending the simply-typed $\lambda$-calculus with types for pairs of objects, atomic types for booleans and naturals and constructions for conditional branching and a recursor.

— Another option that will be investigated in the following lectures will consist in allowing the $\lambda$-terms to be *polymorphic*, that is to be applied to arguments of variable types: this will be the core of System F and of the connection with second-order logic.

## 1.2 Arithmetic and the induction axiom

The well-known Peano's Induction axiom schema is usually presented like:

$$\phi(0) \Rightarrow \forall a.(\phi(a) \Rightarrow \phi(s(a)) \Rightarrow \forall a.\phi(a)$$

In his works (1889 and 1891), Peano formulated the induction axiom in slightly different ways, which can be reformulated as:

**1889:** $\phi(0) \Rightarrow \forall a.[\mathsf{Nat}(a) \Rightarrow (\phi(a) \Rightarrow \phi(s(a))] \Rightarrow \forall a.[\mathsf{Nat}(a) \Rightarrow \phi(a)]$;

**1891:** $\phi(0) \Rightarrow \forall a.(\phi(a) \Rightarrow \phi(s(a))) \Rightarrow \forall a.[\mathsf{Nat}(a) \Rightarrow \phi(a)]$.

In fact, Peano formulated his arithmetic in a form of second-logic (at least allowing quantification over sets, or *classes* of elements), his axioms of Induction were closer to:

**1889':** $\forall k \in K(1 \in k \Rightarrow \forall a.(a \in \mathbb{N} \Rightarrow a \in k \Rightarrow s'a) \in k) \Rightarrow \mathbb{N} \subseteq k)$

**1891':** $\forall s \in K(1 \in s \Rightarrow s(s) \subseteq s \Rightarrow \mathbb{N} \subseteq s)$.

**Exercice 1.1**

*By an analysis of the first-order reformulation of Peano's axiom, justify the reductions to come for the recursor of system* $\mathsf{T}$.

# 2 Gödel's system $\mathsf{T}$

An important defect of the simply-typed $\lambda$-calculus considered during the course is its poor expressiveness as discussed above.

Several systems have been considered to increase the class of (total) functions that can be represented in the typed setting. ***Gödel's System T*** is such a system, extending the simply-typed $\lambda$-calculus with product types ($U \times V$), a type for booleans ($\mathsf{Bool}$), with a type for natural numbers ($\mathsf{Nat}$) and with the following term constructions:

(i) pairs and projections: $\langle t, u\rangle, \pi_1(t), \pi_2(t)$;

(ii) boolean constants and a boolean test: $\mathsf{true}, \mathsf{false}, \mathsf{if}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v$;

(iii) constants for representing natural numbers and a recursor for each type $A$: $\mathsf{S}(t), \mathsf{0}, \mathsf{Rec}(t, u, v)$.

In the following, one will define System $\mathsf{T}$, and then study its strong normalization property.

## 2.1 Types and terms of system $\mathsf{T}$

The types of system $\mathsf{T}$ are just the types of simply-typed $\lambda$-calculus, with two specific atomic types: $\mathsf{Bool}$ and $\mathsf{Nat}$.

**Definition 2.1 (*Simple types for system* $\mathsf{T}$)**

*We consider a countable set* $\mathcal{T}_{\mathsf{At}}$ *of atomic types containing* $\mathsf{Nat}$ *and* $\mathsf{Bool}$. $\mathsf{T}$*-types are defined inductively as*

$$T, U, V ::= A \mid U \times V \mid U \to V \qquad A \in \mathcal{T}_{\mathsf{At}}.$$

Terms of system $\mathsf{T}$ are defined by extending the simply-typed $\lambda$-calculus à la Church with:

**Definition 2.2 (*Terms of System* $\mathsf{T}$)**

*For each* $\mathsf{T}$*-type* $T$*, one considers a countable set of variables of type* $T$*,* $\mathcal{V}^T$*, those sets being pairwise disjoint.*

*Similarly to the case of the simply-typed* $\lambda$*-calculus, we define by mutual induction, (i) the set of terms of System* $\mathsf{T}$ *(called* $\mathsf{T}$*-terms), (ii) the typing relation (written* $u : U$*) and the set of free variables of a* $\mathsf{T}$*-term:*

($\mathsf{Var}$) $\forall x \in \mathcal{V}^U$, $x^U$ *is* $\mathsf{T}$*-term of type* $U$ *(of free variables* $\{x\}$*):* $x^U : U$

($\mathsf{Abs}$) *For every* $\mathsf{T}$*-term* $v$ *such that* $v : V$ *and every variable* $x \in \mathcal{V}^U$, $\lambda x^U.v$ *is a* $\mathsf{T}$*-term of type* $U \to V$ *(of free variables* $fv(v) \setminus \{x\}$*):* $\lambda x^U.v : U \to V$

($\mathsf{App}$) *For every* $\mathsf{T}$*-terms* $t$ *and* $u$ *such that* $t : U \to T$ *and* $u : U$, $(t)u$ *is a* $\mathsf{T}$*-term of type* $T$ *(of free variables* $fv(t) \cup fv(u)$*):* $(t)u : T$

($\mathsf{Prod}$) *For every* $T$*-terms* $u$ *and* $v$ *such that* $u : U$ *and* $v : V$, $\langle u, v\rangle$ *is a* $T$*-term of type* $U \times V$ *(of free variables* $fv(t) \cup fv(u)$*):* $\langle u, v\rangle : U \times V$

($\mathsf{Proj}$) *For every* $T$*-term* $t$ *such that* $t : T_1 \times T_2$, $\pi_1(t)$ *and* $\pi_2(t)$ *are* $T$*-terms of respective types* $T_1$ *and* $T_2$ *(of free variables* $fv(t)$*):* $\pi_1(t) : T_1$ *and* $\pi_2(t) : T_2$

(BoolCst) $\mathsf{true}$ *and* $\mathsf{false}$ *are* **closed** $\mathsf{T}$*-terms of type* $\mathsf{Bool}$*:* $V : \mathsf{Bool}, F : \mathsf{Bool}$

(If) *For every* $\mathsf{T}$*-term* $t, u, v$ *such that* $t : \mathsf{Bool}$, $u : U$ *and* $v : U$, $\mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v$ *is a* $\mathsf{T}$*-term of type* $U$ *(of free variables* $fv(t) \cup fv(u) \cup fv(v)$*):* $\mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v : U$

(0) $0$ *is a* **closed** $\mathsf{T}$*-term of type* $\mathsf{Nat}$*:* $0 : \mathsf{Nat}$

(S) *For every* $\mathsf{T}$*-term* $t$ *such that* $t : \mathsf{Nat}$, $\mathsf{S}(t)$ *is a* $\mathsf{T}$*-term of type* $\mathsf{Nat}$ *(of free variables* $fv(t)$*):* $\mathsf{S}(t) : \mathsf{Nat}$

(Rec) *For every* $\mathsf{T}$*-term* $t, u, v$ *such that* $t : \mathsf{Nat}$, $u : \mathsf{Nat} \to (U \to U)$ *and* $v : U$, $\mathsf{Rec}(t, u, v)$ *is a* $\mathsf{T}$*-term of type* $U$ *(of free variables* $fv(t) \cup fv(u) \cup fv(v)$*):* $\mathsf{Rec}(t, u, v) : U$

*This can be summed up in the following inference system:*

$$\frac{}{x^U : U}\ (Var) \quad (x \in \mathcal{V}^U) \qquad \frac{t : T}{\lambda x^U.t : U \to T}\ (Abs) \quad (x \in \mathcal{V}^U) \qquad \frac{t : U \to T \quad u : U}{(t)u : T}\ (App)$$

$$\frac{u : U \quad v : V}{\langle u, v\rangle : U \times V}\ (Prod) \qquad \frac{t : U_1 \times U_2}{\pi_1(t) : U_1}\ (Proj_1) \qquad \frac{t : U_1 \times U_2}{\pi_2(t) : U_2}\ (Proj_2)$$

$$\frac{}{\mathsf{true} : \mathsf{Bool}}\ (\mathsf{true}) \qquad \frac{}{\mathsf{false} : \mathsf{Bool}}\ (\mathsf{false}) \qquad \frac{}{0 : \mathsf{Nat}}\ (0) \qquad \frac{t : \mathsf{Nat}}{\mathsf{S}(t) : \mathsf{Nat}}\ (\mathsf{S})$$

$$\frac{t : \mathsf{Bool} \quad u : U \quad v : U}{\mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v : U}\ (\mathsf{If}) \qquad \frac{t : \mathsf{Nat} \quad u : \mathsf{Nat} \to (U \to U) \quad v : U}{\mathsf{Rec}(t, u, v) : U}\ (\mathsf{Rec})$$

## 2.2 T-reduction

The notion of ***compatible relation*** is extended to the syntax of T-terms in a straightforward way.

**Definition 2.3 (T-*reduction relation*)**

*We define the* **T-reduction**, *written* $\longrightarrow_{\mathsf{T}}$, *as the least compatible relation on* T*-terms, containing typed* $\beta$*-reduction as well as:*

$$\begin{array}{rcl}
(\lambda x^U.t)u & \longrightarrow_{\mathsf{T}} & t\{u/x\} \\
\pi_i(\langle t_1, t_2\rangle) & \longrightarrow_{\mathsf{T}} & t_i \\
\mathsf{if\ true\ then}\, t\, \mathsf{else}\, u & \longrightarrow_{\mathsf{T}} & t \\
\mathsf{if\ false\ then}\, t\, \mathsf{else}\, u & \longrightarrow_{\mathsf{T}} & u \\
\mathsf{Rec}(0, v, w) & \longrightarrow_{\mathsf{T}} & w \\
\mathsf{Rec}(\mathsf{S}(t), v, w) & \longrightarrow_{\mathsf{T}} & (v)t\mathsf{Rec}(t, v, w)
\end{array}$$

*A* **T-normal form** *is a* T*-term that does not* $\longrightarrow_{\mathsf{T}}$*-reduce to any* T*-term.*

**Proposition 2.4 (*Type preservation*)**

*If* $t : T$ *and* $t \longrightarrow_{\mathsf{T}} u$, *then* $u : T$ *(and* $fv(u) \subseteq fv(t)$*).*

**Proposition 2.5**

*Assume that* $t$ *is a* **closed** T*-normal. We have the following properties:*

- *If* $t : \mathsf{Nat}$, *then there exists* $n \in \mathbb{N}$ *such that* $t = \mathsf{S}^n(0)$*;*
- *If* $t : \mathsf{Bool}$, *then* $t = \mathsf{true}$ *or* $t = \mathsf{false}$*;*
- *If* $\vdash t : A \times B$, *then* $t = \langle u, v\rangle$*;*
- *If* $t : U \to V$, *then* $t = \lambda x.\, u$.

**Proof:** By induction on the structure of terms in normal forms. □

**Notation 2.6 ($\ell(t)$)**

*If* $t$ *is a strongly normalizable* T*-term, one writes* $\ell(t)$ *for the maximal length of a* T*-reduction*

*from $t$. (This is well defined, as in the $\lambda$-calculus, as the reduction graph of a $\mathsf{T}$-term is finitely branching and by König's lemma.)*

# 3 Strong normalization theorem

The following section generalizes the strong normalization for the simply typed $\lambda$-calculus to System $\mathsf{T}$, by adapting the proof by reducibility for the simply typed $\lambda$-calculus.

## 3.1 Preliminary comments

Let us first recall that:

- a $\mathsf{T}$-term $t$ is ***weakly normalizing*** if there is a finite $\mathsf{T}$-reduction sequence from $t$ ending in a normal form.
- a $\mathsf{T}$-term $t$ is ***strongly normalizing*** if there is no infinite $\mathsf{T}$-reduction sequence from $t$, that is whatever choice of redex is made at each step, we are bound to reach a normal form ultimately. It is also the least set $\mathcal{N}$ of $\mathsf{T}$-terms which contains normal forms and such that $t \in \mathcal{N}$ if for any $t'$ such that $t \longrightarrow_{\mathsf{T}} t'$, $t' \in \mathcal{N}$.
- A calculus (here system $\mathsf{T}$) will be called weakly (resp. strongly) normalizing if all its terms are weakly (resp. strongly) normalizing.
- Contrarily to WN, SN is not stable by $\beta$-expansion in general (otherwise SN and WN would be equivalent simply because a normal form is always SN and all normalizable term is the expansion of a normal form).
- On the other hand, SN is stable by $\mathsf{T}$-reduction (which is not the case for WN in general...)
- One of the crux for proving normalization is to transfer normalization properties through elimination rules/destructors, that is proving that if $t \in \mathsf{SN}(A \to B)$ and $u \in \mathsf{SN}(A)$, then $(t)u \in \mathsf{SN}(B)$ (and similarly for product types and atomic types). It is typically such a difficulty that makes an attempt for proving strong normalization by induction in the structure of terms (or of type derivation, which is the same here) to fail.
- On the other hand, there are certainly subsets of $\mathsf{SN}(A)$ for which this proprerty holds (starting with variables of type $A$ for instance).
- As such it may seem interesting to identify a subset of SN terms that would be closed by elimination rulesn and that would have some desirable properties ensuring that every typed term is in this set. We shall call such terms reducible and carry the proof by induction on the structure of terms for this sets. In paritcular, some properties will be important:
  - for the proof by induction to go through, we certainly need the sets of reducible terms to be closed by the constructors: the pair of two reducible terms should be reducible, the abstraction of a reducible term should be reducible, etc.
  - the sets should be closed by reduction as well as by introduction rules,
  - plus some additional properties...

## 3.2 Reducible, neutral and (strongly) normalisable terms

One shall first adapt the definition of neutral terms, which are those terms whose topmost construction is not an introduction rule (in natural deduction terms):

The sets $\mathsf{Neut}(U)$, $\mathsf{SN}(U)$ are adapted to $\mathsf{T}$-terms ***without any change*** (but the dependency of $\mathsf{Neut}(U)$ with $\mathsf{RED}(U)$...):

**Definition 3.1 ($\mathsf{SN}(U)$)**

$$\mathsf{SN}(U) = \{u \in \mathsf{T};\ u \textit{ strongly normalizing of type } U\}.$$

$\mathsf{RED}(U)$ is also defined as for STLC but for a treatment of product types:

**Definition 3.2**

— $\mathsf{RED}(X) = \mathsf{SN}(X)$
— $\mathsf{RED}(U \to V) = \{t : U \to V; \forall u \in \mathsf{RED}(U), (t)\, u \in \mathsf{RED}(V)\}$.
— $\mathsf{RED}(U_1 \times U_2) = \{t : U_1 \times U_2 \mid \forall i \in \{1,2\}, \pi_i(t) \in \mathsf{RED}(U_i)\}$.

**Definition 3.3 (*Neutral* T*-term*)**

*A* T*-term is* **neutral** *if it is not of the form* $\lambda x^U : t$, $\langle t, u\rangle$, $\mathsf{true}$, $\mathsf{false}$, $\mathsf{0}$ *or* $\mathsf{S}(t)$.

The essential property of a neutral term is that if $t$ is neutral, it cannot readily interact with its context and as a consequence, for any context $E[]$, the one-step reducts of $E[t]$ are either of the form $E[t']$ or $E'[t]$ where $E'$ and $t'$ are one-step reducts of $E$ and $t$ respectively. Neutral terms cannot interact/react with their context during the first step of computation.

**Definition 3.4 ($\mathsf{Neut}(U)$)**

$$\mathsf{Neut}(U) = \{u \in \mathsf{T};\ u \text{ is neutral of type } U \text{ and } \forall u', u \longrightarrow_\beta u', u' \in \mathsf{RED}(U)\}$$

## 3.3 Adaptation lemma

In this subsection, we prove the following relation between neutral, reductible and strongly normalisable terms:

**Lemma 3.5 (*Adaptation*)**

*For every type* $T$*, one has* $\mathsf{Neut}(T) \subseteq \mathsf{RED}(T) \subseteq \mathsf{SN}(T)$.

Adaptation lemma relies on

**Lemma 3.6**

*For any type* $U$, $\mathsf{RED}(U)$ *is closed by* T*-reduction:*

$$u \in \mathsf{RED}(U), \qquad u \longrightarrow_{\mathsf{T}} u' \qquad \Rightarrow \qquad u' \in \mathsf{RED}(U).$$

**Proof:** Lemma 3.6 is proved by induction on the structure of type $T$.
— If T is atomic, the base case is trivial: since $T$ is an atomic type, $\mathsf{RED}(T) = \mathsf{SN}(T)$ and begin strongly normalizing is closed by $\beta$-reduction.
— If $T = U \to V$, then let $t \in \mathsf{RED}(T)$ such that $t \longrightarrow t'$: one wishes to prove that $t' \in \mathsf{RED}(T)$. Let $u \in \mathsf{RED}(U)$. One has $(t)u \in \mathsf{RED}(V)$ by definition and the induction hypothesis on $V$ which ensures that $\mathsf{RED}(V)$ is closed by $\beta$-reduction so that $(t')u \in \mathsf{RED}(V)$ and this is true for any $u \in \mathsf{RED}(U)$: one concludes that $t' \in \mathsf{RED}(T)$.
— If $T = U_1 \times U_2$, then let $t \in \mathsf{RED}(T)$ such that $t \longrightarrow t' : T$. Since $t$ is reducible, its projections are also reducible:

$$\pi_i(t) \in \mathsf{RED}(U_i), i \in \{1,2\}.$$

By applying induction hypothesis on $U_1$ and $U_2$, we know that $\mathsf{RED}(U_i)$ are closed by reduction and since $\pi_i(t) \longrightarrow \pi_i(t')$ with $i \in \{1,2\}$, we have that $\pi_i(t') \in \mathsf{RED}(U_i)$ for $i \in \{1,2\}$. Therefore $t' \in \mathsf{RED}(T)$.

□

**Proof of lemma 3.5:** The proof is by induction on the structure of type $T$.
— If $T = X$, $\mathsf{RED}(X) = \mathsf{SN}(X)$ by definition. Moreover, if $t \in \mathsf{Neut}(X)$, then for any $t'$ such that $t \longrightarrow_\beta t'$, one has $t' \in \mathsf{RED}(X) = \mathsf{SN}(X)$ therefore $t$ is strongly normalisable and $t \in \mathsf{RED}(X)$.
— If $T = U \to V$, one has:
  — $\mathsf{Neut}(T) \subseteq \mathsf{RED}(T)$: let $t \in \mathsf{Neut}(T)$, that is it is neutral and all its reducts are reducible of type $U \to V$: $\forall t', t \longrightarrow_\beta t'$, one has $t' \in \mathsf{RED}(T)$.
  Let us prove by induction on the length of the longest reduction from $u$, $\ell(u)$, that for all $u \in \mathsf{RED}(U) \subseteq \mathsf{SN}(U)$, $(t)u \in \mathsf{RED}(V)$.

— if $\ell(u) = 0$, then $u$ is normal and the reducts of $(t)u$ are all of the form $(t')u$ where $t$ reduces to $t'$, so that $t' \in \mathsf{RED}(U \to V)$ and $(t')u \in \mathsf{RED}(V)$. As a consequence, all reducts of $(t)u$, which is neutral, are in $\mathsf{RED}(V)$: $(t)u \in \mathsf{Neut}(V) \subseteq \mathsf{RED}(V)$ (by induction hypothesis).
— Let us assume that the property is true of the terms $v$ such that $\ell(v) \leq n$ and consider $u$ such that $\ell(u) = n + 1$. Let $v$ a reduct of $(t)u$, either it is obtained by reducing $t$ to $t'$: $v = (t')u$, or by reducing $u$ to $u'$, $v = (t)u'$, but there is no other optionsince $t$ is neutral. In the first case, as we know that $t' \in \mathsf{RED}(U \to V)$, one has $v \in \mathsf{RED}(V)$. In the second case, since $\ell(u') \leq n$, and since $u' \in \mathsf{RED}(U)$ by closure of $\mathsf{RED}(U)$ by reduction (lemma 3.6), the induction hypothesis ensures that $v \in \mathsf{RED}(V)$. Since all reducts of $(t)u$ are in $\mathsf{RED}(V)$ and $(t)u$ is neutral, one concludes that $(t)u \in \mathsf{Neut}(V) \subseteq \mathsf{RED}(V)$.

— $\mathsf{RED}(T) \subseteq \mathsf{SN}(T)$: let $t \in \mathsf{RED}(T)$, let us prove that it is strongly normalizing. One has $x^U \in \mathsf{Neut}(U) \subseteq \mathsf{RED}(U)$ (by induction hypothesis) so that $(t)x \in \mathsf{RED}(V) \subseteq \mathsf{SN}(V)$ and the previous lemma allows to conclude that $t \in \mathsf{SN}(T)$.

— If $T = U_1 \times U_2$, then:

— $\mathsf{Neut}(T) \subseteq \mathsf{RED}(T)$:
Let $t \in \mathsf{Neut}(T)$. Since $t$ is neutral, $\pi_i(t)$ cannot be a redex itself: its redexes are necessarily in $t$, so that its one-step reducts are all of the form $\pi_i(t')$ with $t \longrightarrow t'$. Since $t \in \mathsf{Neut}(T)$, $t' \in \mathsf{RED}(T)$ and $\pi_i(t') \in \mathsf{RED}(U_i), i \in \{1, 2\}$.
Therefore we have that $\pi_i(t), i \in \{1, 2\}$ are neutral and all their one-step reducts are reducible: $\pi_i(t) \in \mathsf{Neut}(U_i), i \in \{1, 2\}$. $\pi_i(t) \in \mathsf{RED}(U_i), i \in \{1, 2\}$.
By definition of reducibility at product types, one concludes that $t \in \mathsf{RED}(T)$ as expected.

— $\mathsf{RED}(T) \subseteq \mathsf{SN}(T)$:
Assume that $t \in \mathsf{RED}(T)$. The $\pi_1(t) \in \mathsf{RED}(U_1)$ by definition and, by induction hypothesis on $U$, $\pi_1(t) \in \mathsf{SN}(U_1)$. The longest reduction from $t$ is certainly at most as long as that from $\pi_1(t)$ so there is only finite reduction sequence from $t$ and $t \in \mathsf{SN}(T)$.

□

## 3.4 Adequation lemma

In the following section, we prove the key property for strong normalization, namely that typed terms are adequate with respect to reducibility:

**Lemma 3.7 (*Adequation*)**

*Let $t : U$ with free variables among $x_1^{T_1}, \ldots, x_n^{T_n}$. For any $(u_i \in \mathsf{RED}(T_i))_{1 \leq i \leq n}$, one has $t\{u_i/x_i\} \in \mathsf{RED}(U)$.*

**Remark 3.8**

*An immediate consequence of the lemma is that closed* T*-terms are reducible and therefore, by adaptation lemma, they are strongly normalizable. One understands that the proof of strong normalization is a short step from adaptation and adequation lemmas...*

**Proposition 3.9**

*The following holds:*

1. $0 \in \mathsf{RED}(\mathsf{Nat})$.
2. $\mathsf{true}, \mathsf{false} \in \mathsf{RED}(\mathsf{Bool})$.
3. $\forall t \in \mathsf{RED}(\mathsf{Nat}), \mathsf{S}(t) \in \mathsf{RED}(\mathsf{Nat})$.
4. $\forall t \in \mathsf{RED}(\mathsf{Bool}), \forall u, v \in \mathsf{RED}(U)$, $\mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v \in \mathsf{RED}(U)$.
5. $\forall t \in \mathsf{RED}(\mathsf{Nat}), \forall u \in \mathsf{RED}(\mathsf{Nat} \to (U \to U)), \forall v \in \mathsf{RED}(U)$, $\mathsf{Rec}(t, u, v) \in \mathsf{RED}(U)$.

**Proof:**

1. $0 \in \mathsf{RED}(\mathsf{Nat})$ since $\mathsf{RED}(\mathsf{Nat}) = \mathsf{SN}(\mathsf{Nat})$ and $0$ is a T-normal form.
2. $\mathsf{true}, \mathsf{false} \in \mathsf{RED}(\mathsf{Bool})$ since $\mathsf{RED}(\mathsf{Bool}) = \mathsf{SN}(\mathsf{Bool})$ and $\mathsf{true}, \mathsf{false}$ are T-normal forms.
3. let $t \in \mathsf{RED}(\mathsf{Nat})$, then $t$ is strongly normalizable since $\mathsf{Nat}$ is an atomic type. Since any reduction from $\mathsf{S}(t)$ is of the form $\mathsf{S}(t) \longrightarrow \mathsf{S}(t_1) \longrightarrow \mathsf{S}(t_2) \longrightarrow \ldots \mathsf{S}(t_n) \longrightarrow \ldots$, with

$t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \cdots \longrightarrow t_n \longrightarrow \ldots$, $\mathsf{S}(t)$ is also strongly normalizable and therefore $\mathsf{S}(t) \in \mathsf{RED}(\mathsf{Nat})$.

4. let $t : \mathsf{Bool}$, $u, v : U$ be such that $t \in \mathsf{RED}(\mathsf{Bool})$, $u, v \in \mathsf{RED}(U)$. By adaptation lemma, it is sufficient to prove that $w = \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v \in \mathsf{Neut}(U)$ to deduce $w \in \mathsf{RED}(U)$. By adaptation lemma, we know that $t, u, v$ are all strongly normalizing so that we can reason by induction on $\ell(t) + \ell(u) + \ell(v)$ to prove that $\forall t \in \mathsf{RED}(\mathsf{Bool})$, $\forall u, v \in \mathsf{RED}(U)$, $\mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v \in \mathsf{RED}(U)$.

   $w = \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v$ is neutral, let us consider its one-step reducts: if $w \longrightarrow_{\mathsf{T}} w'$ then
   — either $w'$ is $u$ (resp. $v$) if $t = \mathsf{true}$ (resp. $t = \mathsf{false}$) which is reducible of type $U$
   — or $w' = \mathsf{if}\, t\, \mathsf{then}'\, \mathsf{else}\, uv$ with $t \longrightarrow_{\mathsf{T}} t'$ and since $\ell(t') + \ell(u) + \ell(v) < \ell(t) + \ell(u) + \ell(v)$, $w' \in \mathsf{RED}(U)$ by induction hypothesis;
   — or $w' = \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}'\, v$ with $u \longrightarrow_{\mathsf{T}} u'$ and since $\ell(t) + \ell(u') + \ell(v) < \ell(t) + \ell(u) + \ell(v)$, $w' \in \mathsf{RED}(U)$ by induction hypothesis;
   — or $w' = \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v'$ with $v \longrightarrow_{\mathsf{T}} v'$ and since $\ell(t) + \ell(u) + \ell(v') < \ell(t) + \ell(u) + \ell(v)$, $w' \in \mathsf{RED}(U)$ by induction hypothesis.

5. let $t : \mathsf{Nat}$, $u : \mathsf{Nat} \to (U \to U)$ and $v : U$ be such that $t \in \mathsf{RED}(\mathsf{Nat})$, $u \in \mathsf{RED}(\mathsf{Nat} \to (U \to U))$ and $v \in \mathsf{RED}(U)$. As above, it is sufficient to prove that $w = \mathsf{Rec}(t, u, v) \in \mathsf{Neut}(U)$. $w$ being neutral we simply have to prove that any of its one-step reducts is in $\mathsf{RED}(U)$ which is done by induction, in a slightly more complex way as for the boolean test.

   Indeed, consider the case when $t$ is of the form $\mathsf{S}(t')$, then $w$ can reduce to $(u)t'\mathsf{Rec}(t', u, v)$. To prove that the term is reducible, the inductive measure considered for the boolean destructor is not suitable: indeed, in that case one has to rely on the reducibility of $u$ (by hypothesis) and $t'$ (reducible because strongly normalizable and of atomic type) and we need to establish reducibility of $\mathsf{Rec}(t', u, v)$ but $\ell(t') + \ell(u) + \ell(v) = \ell(t) + \ell(u) + \ell(v)$: the measure did not decrease... One way out, it to add to the measure the information on the complexity of $t$ (or of its normal form):
   — either by taking $\ell(t) + \ell(u) + \ell(v) + n(t)$ where $n(t)$ is the size of ***the normal form*** of $t$ (indeed, the size of $t$ may vary over the reduction and is not necessarily decreasing through the reduction...)
   — or by considering $(\ell(t) + \ell(u) + \ell(v), s(t))$ ordered lexicographically, where $s(t)$ is the size of $t$ (here it is sufficient to consider the size of $t$, and not of its normal form, since one uses this component of the measure only when the term can be structurally compared, one being a subterm of the other, see below).

   Let us consider the first option which is simpler and sufficient (we comment on applicability of the other measure as well):

   let us prove by induction on $\ell(t) + \ell(u) + \ell(v) + n(t)$, that for all $\forall t \in \mathsf{RED}(\mathsf{Nat})$, $\forall u \in \mathsf{RED}(\mathsf{Nat} \to (U \to U))$, $\forall v \in \mathsf{RED}(U)$, $\mathsf{Rec}(t, u, v) \in \mathsf{RED}(U)$.

   Let thus consider $t : \mathsf{Nat}$, $u : \mathsf{Nat} \to (U \to U)$ and $v : U$ be such that $t \in \mathsf{RED}(\mathsf{Nat})$, $u \in \mathsf{RED}(\mathsf{Nat} \to (U \to U))$ and $v \in \mathsf{RED}(U)$.

   $w = \mathsf{Rec}(t, u, v)$ is neutral, let us consider its one-step reducts: if $w \longrightarrow_{\mathsf{T}} w'$ then
   — either $w'$ is $v$ if $t = \mathsf{0}$ which is reducible of type $U$;
   — or $w'$ is $(u)t'\mathsf{Rec}(t', u, v)$ if $t = \mathsf{S}(t')$. In that case $n(t) = n(t') + 1$ and $\ell(t') = \ell(t)$ so that $\ell(t') + \ell(u) + \ell(v) + n(t') < \ell(t) + \ell(u) + \ell(v) + n(t)$, and therefore induction hypothesis ensures that $\mathsf{Rec}(t', u, v)$ is reducible of type $U$ which together with the fact that $t \in \mathsf{RED}(\mathsf{Nat})$, $u \in \mathsf{RED}(\mathsf{Nat} \to (U \to U))$, ensures that $w' \in \mathsf{RED}(U)$. *[Note that the other measure, $(\ell(t) + \ell(u) + \ell(v), s(t))$ would also have decreased since its first component would be unchanged while its second component has strictly decreased as $s(t) = s(t') + 1$.]*
   — or $w' = \mathsf{Rec}(t', u, v)$ with $t \longrightarrow_{\mathsf{T}} t'$ and since $\ell(t') + \ell(u) + \ell(v) < \ell(t) + \ell(u) + \ell(v)$ and since $n(t) = n(t')$, $w' \in \mathsf{RED}(U)$ by induction hypothesis; *[Note that the other measure, $(\ell(t) + \ell(u) + \ell(v), s(t))$ would also have decreased since its first component would have decreased.]*
   — or $w' = \mathsf{Rec}(t, u', v)$ with $u \longrightarrow_{\mathsf{T}} u'$ and since $\ell(t) + \ell(u') + \ell(v) + n(t) < \ell(t) + \ell(u) + \ell(v) + n(t)$, $w' \in \mathsf{RED}(U)$ by induction hypothesis; *[Note that the other measure, $(\ell(t) + \ell(u) + \ell(v), s(t))$ would also have decreased since its first component would have decreased.]*
   — or $w' = \mathsf{Rec}(t, u, v')$ with $v \longrightarrow_{\mathsf{T}} v'$ and since $\ell(t) + \ell(u) + \ell(v') + n(t) < \ell(t) + \ell(u) + \ell(v) + n(t)$, $w' \in \mathsf{RED}(U)$ by induction hypothesis. *[Note that the other measure,*

$(\ell(t)+\ell(u)+\ell(v), s(t))$ *would also have decreased since its first component would have decreased.]*

From this case analysis, we deduce that any one-step reduct of $w$ is reducible which suffices to deduce that $w \in \mathsf{Neut}(U) \subseteq \mathsf{RED}(U)$.

□

The following lemma will be important:

**Lemma 3.10**

$$(\forall u \in \mathsf{RED}(U), v\{u/x\} \in \mathsf{RED}(V)) \Rightarrow \forall u \in \mathsf{RED}(U), (\lambda x.v)u \in \mathsf{RED}(V).$$

together with its immediate corollary (by definition of $\mathsf{RED}(U \to V)$):

**Corollary 3.11**

$$(\forall u \in \mathsf{RED}(U), v\{u/x\} \in \mathsf{RED}(V)) \Rightarrow \lambda x.v \in \mathsf{RED}(U \to V).$$

**Proof of the lemma :** Let $v : V$ possibly with $x^U$ as free variable, and assume that for all $u \in \mathsf{RED}(U)$, $v\{u/x\} \in \mathsf{RED}(V)$.

Let us first remark that, since variables are reducible for their type, $v = v\{x^U/x\} \in \mathsf{RED}(V)$. In particular, $u$ and $v$ are strongly normalizing by adaptation.

We show the result by induction on $\ell(u) + \ell(v)$: considering $t = (\lambda x.v)u$ which is neutral, it is enough to prove that it is in $\mathsf{Neut}(V)$ to have the result by adaptation lemma. One shall therefore consider its reducts and show that they are all reducible of type $V$: consider therefore $t'$ a one-step reduct of $t$. There are three possible cases:

— $t' = v\{u/x\}$ which is in $\mathsf{RED}(V)$ by the lemma hypothesis;

— $t' = (\lambda x.v')u$ with $v'$ a one-step reduct of $v$. Then $v' \in \mathsf{RED}(V)$ and for any $u \in \mathsf{RED}(U)$, $v'\{u/x\} \in \mathsf{RED}(V)$ since $v\{u/x\} \longrightarrow_\beta v'\{u/x\}$ and since $\mathsf{RED}(V)$ is closed by reduction. Since $\ell(u) + \ell(v') < \ell(u) + \ell(v)$, one can apply the induction hypothesis and conclude que that $t' = (\lambda x.v')u \in \mathsf{RED}(V)$.

— $t' = (\lambda x.v)u'$ with $u'$ a one-step reduct of $u$ which is therefore in $\mathsf{RED}(U)$, we known by hypothesis that $v\{u'/x\} \in \mathsf{RED}(V)$. Since $\ell(u') + \ell(v) < \ell(u) + \ell(v)$, one can apply the induction hypothesis and conclude that $t' \in \mathsf{RED}(V)$.

□

The following is a corresponding result for pairs:

**Lemma 3.12**

$$\forall u \in \mathsf{RED}(U), v \in \mathsf{RED}(V), \langle u, v\rangle \in \mathsf{RED}(U \times V).$$

**Proof :** By adaptation lemma, one can reason using the strong normalisation of $u, v$ and therefore reason by induction on the sum of the length of the longest reductions from $u$ and $v$ to show that $\pi_i(\langle u, v\rangle)$ is reducible.

First notice that this term is neutral. Therefore, to show that is it reducible, it is sufficient to show that every one-step reduct is reducible from which one deduce that $\pi_i(\langle u, v\rangle) \in \mathsf{Neut}(U)$ and, by adaptation, that it is reducible.

$\pi_i(\langle u, v\rangle)$ reduces (i) either to $u$ (resp. $v$) which is reducible, (ii) or to $\pi_i(\langle u', v\rangle)$ with $u \longrightarrow u'$. $u'$ is reducible since reducibility is closed by reduction and its longest reduction is shorter than that of $u$ so by induction hypothesis, $\pi_i(\langle u', v\rangle)$ is reducible, (iii) or to $\pi_i(\langle u, v'\rangle)$ with $v \longrightarrow v'$ which is reducible by exactly the same reasoning as in (ii).

Therefore both projections of $\langle u, v\rangle$ are reducible showing that $\langle u, v\rangle \in \mathsf{RED}(U \times V)$.

□

**Proof of lemma 3.7 :** One can reason by induction on the structure of $t : T$.

— If $t = x_i^{T_i}$, the result is trivial since $u_i$ is in $\mathsf{RED}(T_i)$ by the lemma hypothesis.

— If $t = \lambda x^U.t'$, with $T = U \to V$ and choosing $x^U$ such that is not free in any of the $u_i$ and distinct from all the $x_i^{T_i}$. Let $u \in \mathsf{RED}(U)$. By induction hypothesis, $t'\{u_i/x_i, 1 \le i \le n\}\{u/x\} = t'\{u_i/x_i, 1 \le i \le n, u/x\} \in \mathsf{RED}(V)$ therefore, by the previous lemma, one has $t\{u_i/x_i, 1 \le i \le n\} = \lambda x.t'\{u_i/x_i, 1 \le i \le n\} \in \mathsf{RED}(U \to V)$

— If $t = (u)v$, with $u : V \to T$ and $v : V$. By induction hypothesis, $u' = u\{u_i/x_i, 1 \le i \le n\} \in \mathsf{RED}(V \to T)$ et $v' = v\{u_i/x_i, 1 \le i \le n\} \in \mathsf{RED}(V)$ therefore $t\{u_i/x_i, 1 \le i \le n\} = (u')v' \in \mathsf{RED}(T)$.

— If $t = \langle u, v\rangle$, then by induction hypothesis, both $u\{u_i/x_i\}$ and $v\{u_i/x_i\}$ are reducible and by the previous lemma $t\{u_i/x_i\}$ is reducible.
— If $t = \pi_1(u)$ (resp $\pi_2(u)$), then by induction hypothesis $u\{u_i/x_i\}$ is reducible which implies that $\pi_1(u\{u_i/x_i\})$ is reducible by definition.
— If $t$ is some $\mathsf{T}$-constant, it is reducible (since $\mathsf{0} \in \mathsf{RED}(\mathsf{Nat})$, $\mathsf{true}, \mathsf{false} \in \mathsf{RED}(\mathsf{Bool})$).
— If $t = \mathsf{S}(u)$, then by induction hypothesis, $u\{u_i/x_i\}$ is reducible and so is $\mathsf{S}(u\{u_i/x_i\})$.
— If $t = \mathsf{if}\, u\, \mathsf{then}\, v\, \mathsf{else}\, w$, then by induction hypothesis, $u\{u_i/x_i\}$, $v\{u_i/x_i\}$, $w\{u_i/x_i\}$ are reducible and so is $\mathsf{if}\, u\{u_i/x_i\}\, \mathsf{then}\, v\{u_i/x_i\}\, \mathsf{else}\, w\{u_i/x_i\}$.
— If $t = \mathsf{Rec}(u, v, w)$, then by induction hypothesis, $u\{u_i/x_i\}$, $v\{u_i/x_i\}$, $w\{u_i/x_i\}$ are reducible and so is $\mathsf{Rec}(u\{u_i/x_i\}, v\{u_i/x_i\}, w\{u_i/x_i\})$.

□

### 3.5 Conclusion of the proof of strong normalization

**Theorem 3.13**

*System* $\mathsf{T}$ *is strongly normalizing.*

**Proof:** Let $t : T$ of free variables $(x_i^{T_i})_{1\leq i\leq n}$. By adaptation lemma (3.5) for any $1 \leq i \leq n$, $x_i^{T_i} \in \mathsf{RED}(T_i)$ since variables of type $T$ are neutral and normal and therefore in $\mathsf{Neut}(T)$.

Adequation lemma (3.7) ensures that $t\left\{x_i^{T_i}/x_i, 1 \leq i \leq n\right\} = t$ is reducible of type $T$ ($\in \mathsf{RED}(T)$).

By using adaptation lemma once more, one has $t \in \mathsf{RED}(T) \subseteq \mathsf{SN}(T)$ which allows to conclude that $t$ is strongly normalizing.

□

## 4 Expressive power of system $\mathsf{T}$

It is easy to write complex programs in $\mathsf{T}$, that cannot be written in simply-typed $\lambda$-calculus. Back to the introduction of this chapter, of course one can manipulate pairs as we are given primitive operations in $\mathsf{T}$, as well as boolean functions as we have the boolean test.

**Exercice 4.1**

*Write* $\mathsf{T}$*-terms for the standard boolean functions.*

### 4.1 Simple arithmetical functions represented by $\mathsf{T}$-terms.

It is simple to defined basic arithmetical functions on type $\mathsf{Nat}$: instead of manipulating Church numerals, one works with the built-in naturalnumbers of $\mathsf{T}$, which does not make a big difference as they are unary integers as well as we have a recursor to replace the ability of a Church numeral to iterate its arguments directly:

**Exercice 4.2**

*Write* $\mathsf{T}$*-terms for the following functions:*
— *successor;*
— *addition;*
— *multiplication;*
— *exponentiation;*
— *predecessor;*
— *subtraction.*

### 4.2 Ackermann-Péter function in $\mathsf{T}$.

Notice here that the type of the recursors we have been using sofar is very simple: $U$ is always taken to be $\mathsf{Nat}$ is the previous examples... We can benefit from the ability to use more complex types, higher-order types in fact, to defined simply much more complex, and fast-growing functions, for instance we shall see now how to represent Ackermann-Péter function in system $\mathsf{T}$.

Let us consider Ackermann-Péter function for a while:

$$A(m,n) \triangleq \begin{cases} n+1 & \text{if } m = 0 \\ A(m-1,1) & \text{if } m > 0 \text{ and } n = 0 \\ A(m-1, A(m,n-1)) & \text{if } m > 0 \text{ and } n > 0 \end{cases}$$

In order to represent $A$ in $\mathsf{T}$, we would need a $\mathsf{T}$-term $\mathsf{A}$ such that

$$\begin{array}{lll} (\mathsf{A})\mathsf{0}n & \longrightarrow_{\mathsf{T}}^{\star} & \mathsf{S}(n) \\ (\mathsf{A})\mathsf{S}(m)\mathsf{0} & \longrightarrow_{\mathsf{T}}^{\star} & (\mathsf{A})m\mathsf{S}(\mathsf{0}) \\ (\mathsf{A})\mathsf{S}(m)\mathsf{S}(n) & \longrightarrow_{\mathsf{T}}^{\star} & (\mathsf{A})m(\mathsf{A})\mathsf{S}(m)n \end{array}$$

However, it is well-known that Ackermann-Peter function is not primitive recursive and in system $\mathsf{T}$, we only have a recursor, not minimization scheme construct. How to find a solution?

Let us consider $A$, by currying, not as a function of two arguments but as a family of unary functions $(A_m)_{m\in\mathbb{N}}$ from $\mathbb{N}$ to $\mathbb{N}$. We then notice that the definition becomes:

$$\begin{array}{l} A_0(n) \triangleq n+1 \\ A_{m+1}(n) \triangleq \begin{cases} A_m(1) & \text{if } n = 0 \\ A_m(A_{m+1}(n-1)) & n > 0 \end{cases} \end{array}$$

And we notice that each $A_i$ is now defined with only a primitive recursive scheme, assuming the $A_0, \ldots, A_{i-1}$ have been defined already. This means that we need to be able to define, not an object in $\mathbb{N}$ by recursion, but an element of $\mathbb{N}^{\mathbb{N}}$, which is exactly what the recursor of system $\mathsf{T}$ allows for when instantiating $U$ with type $\mathsf{Nat} \to \mathsf{Nat}$...

The effect of $A_{m+1}$ on $n$ is to iterate $A_m$ $n+1$ times over 1: $A_{m+1}(n) = A_m(A_{m+1}(n-1)) = A_m(A_m(A_{m+1}(n-2))) = A_m(A_m(A_m(A_{m+1}(n-3)))) = \cdots = A_m(A_m(A_m(A_m(\ldots(A_m(1)\ldots)))))$! That is simply (if $f^{(0)}(x) = x$ and $f^{(n+1)}(x) = f(f^{(n)}(x))$):

$$A_{m+1}(n) = A_m^{(n+1)}(1).$$

which can also be define as: $A_{m+1}(n) = iter(A_m, n)$ where $iter(f,0) = f(1)$ and $iter(f, n+1) = f(iter(f,n))$.

Now, we see clearly how to complete the definition of $\mathsf{A}$:

**Exercice 4.3**

*Define a* $\mathsf{T}$*-term* $\mathsf{Iter}$ *representing iter as described above, that is it takes as input two arguments of type* $\mathsf{Nat} \to \mathsf{Nat}$ *and* $\mathsf{Nat}$ *and iterates its first argument as many time as specified by its second argument.*

**Exercice 4.4**

*Using* $\mathsf{Iter}$*, define a* $\mathsf{T}$*-term* $\mathsf{A}$ *representing Ackermann-Peter function, that is such that for any* $m, n : \mathsf{Nat}$*:*

$$\begin{array}{lll} (\mathsf{A})\mathsf{0}n & =_{\mathsf{T}} & \mathsf{S}(n) \\ (\mathsf{A})\mathsf{S}(m)\mathsf{0} & =_{\mathsf{T}} & (\mathsf{A})m\mathsf{S}(\mathsf{0}) \\ (\mathsf{A})\mathsf{S}(m)\mathsf{S}(n) & =_{\mathsf{T}} & (\mathsf{A})m(\mathsf{A})\mathsf{S}(m)n \end{array}$$

*with* $=_{\mathsf{T}}$ *denoting the least congruence containing* $\longrightarrow_{\mathsf{T}}$*.*

## 4.3 A total recursive function not representable in $\mathsf{T}$.

In this paragraph, we describe the construction of a total recursive function that cannot be represented in $\mathsf{T}$. This construction is general and will be reproduced later in the semester for system $\mathsf{F}$: it amounts on a diagonalization argument, showing that the evaluation function of $\mathsf{T}$ which is (total) recursive cannot be represented in $\mathsf{T}$.

Indeed, consider $\mathsf{g}(\_)$ a Gödel numbering of $\mathsf{T}$-terms and the following functions:

— $\mathsf{eval}(n) = \begin{cases} \mathsf{g}(u) & \text{if } n = \mathsf{g}(t) \text{ and } t \longrightarrow^{\star} u \not\longrightarrow \\ 0 & \text{otherwise} \end{cases}$

— $\mathsf{apply}(m,n) = \begin{cases} \mathsf{g}(v) & \text{if } m = \mathsf{g}(t), n = \mathsf{g}(u) \text{ and } v = (t)u \text{ is a } \mathsf{T}\text{-term.} \\ 0 & \text{otherwise} \end{cases}$
— $\#(n) = \mathsf{g}(\overline{n})$ (where $\overline{n}$ is the $\mathsf{Nat}$ term corresponding to $n$).
— $\mathsf{b}(n) = \begin{cases} m & \text{if } n = \mathsf{g}(\overline{m}) \\ 0 & \text{otherwise} \end{cases}$

Otherwise said, :

— $\mathsf{eval}(\_)$ returns the Gödel number of the normal form of the $\mathsf{T}$-term coded by its input if the input codes a $\mathsf{T}$-term and returns 0 otherwise.
— $\mathsf{apply}(\_)$ returns the Gödel number of the application of the terms coded by its arguments and returns 0 if the arguments are not of the appropriate types.
— $\#(\_)$ returns the Gödel number of its input, viewed as a $\mathsf{T}$-nat: it codes a natural of system $\mathsf{T}$.
— $\mathsf{b}(\_)$ does the opposite of $\#(\_)$, decoding its input: if the input is the code of a $\mathsf{T}$ natual number, it returns the corresponding nat, otherwise it returns 0. In puarticular, $\mathsf{b}(\#(n)) = n$ for any $n \in \mathbb{N}$.

The following proposition is clear and left to the reader:

**Proposition 4.1**

*$\mathsf{g}, \mathsf{eval}, \mathsf{apply}, \#$ and $\mathsf{b}$ are total recursive functions.*

Consider now $\mathsf{diag}$ defined as:

$$\mathsf{diag}(n) = \mathsf{b}(\mathsf{eval}(\mathsf{apply}(n, \#(n)))) + 1$$

Assume $d$ is a $\mathsf{T}$-term representing $\mathsf{diag}$ and let $n = \mathsf{g}(d)$. Then we have:
— $\mathsf{apply}(n, \#(n)) = \mathsf{g}((d)\overline{n})$;
— $\mathsf{eval}(\mathsf{apply}(n, \#(n))) = \mathsf{g}(u)$ such that $(d)\overline{n} \longrightarrow^\star u \not\longrightarrow$;
— $(d)\overline{n} \longrightarrow^\star \mathsf{diag}(n)$ by definition;
— $\mathsf{eval}(\mathsf{apply}(n, \#(n))) = \mathsf{g}(\overline{\mathsf{diag}(n)})$ so
— $\mathsf{diag}(n) = \mathsf{b}(\mathsf{g}(\overline{\mathsf{diag}(n)}))$ and finally
— $\mathsf{diag}(n) = \mathsf{diag}(n) + 1$...

As a consequence, $\mathsf{diag}$ is total recursive which cannot be represented in $\mathsf{T}$.

**Remark 4.2**

*Note that the above construction does not use any thing about $\mathsf{T}$, but uniqueness of its normal forms and will therefore be reused for system $\mathsf{F}$.*

## 4.4 Characterization of the expressiveness of $\mathsf{T}$.

More generally, the extended expressiveness of $\mathsf{T}$ that was mentioned in the start is expressed by the following theorem:

**Theorem 4.3**

*The functions that can be represented in system $\mathsf{T}$ are the recursive functions which can be proved to be total functions in first-order Peano arithmetics ($\mathsf{PA}$).*