M2 LMFI – SOFIX: Second-order quantification and fixed-points in logic
First lecture: Gödel's System T
(preliminary version of the 7/01/2022)

Alexis Saurin

4th january 2022

Contents

1 On the weak expressiveness of the simply typed λ-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 see that every $n \geq 2$ allows to type Church numerals. In Church's style λ-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 type 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, *and the identity function the operations of addition, multiplication and conditional.*

Theorem 1.2 (*Schwichtenberg and Statman*)

The arithmetical functions definable in simply-typed λ-calculus over type 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 λ-terms of type $[n+2] \to \dots ([n+2] \to [n+2])$ for $n \geq 0$, then one can define more functions in simply-typed λ-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 λ-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 lecture will consist in allowing the λ-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.

2 Gödel's system T.

An important defect of the simply-typed λ-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 λ-calculus with product types $(U \times V)$, a type for booleans (Bool), with a type for natural numbers (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 T, and then study its strong normalization property.

2.1 Types and terms of system T

The types of system T are just the types of simply-typed λ-calculus, with two specific atomic types: Bool and Nat.

Definition 2.1 (*simple types for system* T)

We consider a countable set $\mathcal{T}_{\mathsf{At}}$ of atomic types containing Nat *and* Bool. 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 T are defined by extending the simply-typed λ-calculus à la Church with:

Definition 2.2 (*Terms of System* T)

For each 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 λ-calculus, we define by mutual induction, (i) the set of terms of System T *(called* T*-terms), (ii) the typing relation (written $u : U$) and the set of free variables of a* T*-term:*

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

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

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

(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 $fv(t) \cup fv(u)$): $\langle u, v\rangle : U \times V$;*

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

(If) *For every* 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* 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** T-*term of type* Nat*:* $0 : \mathsf{Nat}$

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

(Rec) *For every* 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* 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:

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

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

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

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

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 β-reduction as well as:*

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

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

Proposition 2.4

Assume that t is a **closed** T-*normal. Prove that:*
- *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 $t = \lambda x.\, u$.*

Démonstration : By induction on the structure of terms in normal forms.

□

Notation 2.5 ($\ell(t)$)

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 λ-calculus, as the reduction graph of a* T-*term is finitely branching and by König's lemma.)*

2.2 Strong normalization theorem

The following paragraph generalizes the strong normalization for the simply typed λ-calculus to System T, by adapting the proof by reducibility.

One shall first adapt the definition of neutral terms:

Definition 2.6 (*Neutral* T*-term*)

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

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

Definition 2.7 ($\mathsf{Neut}^{\mathsf{SN}}(U)$)

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

Definition 2.8 ($\mathsf{SNorm}(U)$)

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

$\mathsf{RED}^{\mathsf{SN}}(U)$ is also defined as for STLC but for the use of the previous definition of neutral terms and a treatment of product types:

Definition 2.9

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

Lemma 2.10 (*Adaptation*)

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

Adaptation lemma relies on

Lemma 2.11

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

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

<u>Démonstration :</u> The lemma is proved by induction on the structure of type T.

- If T is atomic, as in STLC
- If $T = U \to V$, as in STLC
- If $T = U_1 \times U_2$, then let $t : T$ such that $t \longrightarrow t'$. Since t is reducible, its projection are: $\pi_i(t) \in \mathsf{RED}^{\mathsf{SN}}(U_i), i \in \{1,2\}$. By applying induction hypothesis on U_1 and U_2, we know that $\mathsf{RED}^{\mathsf{SN}}(U_i)$ are closed by reduction and since $\pi_i(t) \longrightarrow \pi_i(t')$ with $i \in \{1,2\}$, we have that $\pi_i(t') \in \mathsf{RED}^{\mathsf{SN}}(U_i)$ for $i \in \{1,2\}$. Therefore $t' \in \mathsf{RED}^{\mathsf{SN}}(T)$.

□

<u>Démonstration of lemma 2.10 :</u> The proof is by induction on the structure of type T:

- If $T = X$, as for STLC
- If $T = U \to V$, as for STLC
- If $T = U_1 \times U_2$, then:
  - $\mathsf{Neut}^{\mathsf{SN}}(T) \subseteq \mathsf{RED}^{\mathsf{SN}}(T)$:
    Let $t \in \mathsf{Neut}^{\mathsf{SN}}(T)$. 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 $t \in \mathsf{Neut}^{\mathsf{SN}}(T)$, $t' \in \mathsf{RED}^{\mathsf{SN}}(T)$ and $\pi_i(t') \in \mathsf{RED}^{\mathsf{SN}}(U)_i, i \in \{1,2\}$.
    Therefore we have that $\pi_i(t), i \in \{1,2\}$ are neutral and all their one-step reducts are reducible: $\pi_i(t) \in \mathsf{Neut}^{\mathsf{SN}}(U)_i, i \in \{1,2\}$.
    By induction hypothesis on U_1 and U_2, $\pi_i(t) \in \mathsf{RED}^{\mathsf{SN}}(U)_i, i \in \{1,2\}$.
    By definition of reducibility at product type, one concludes that $t \in \mathsf{RED}^{\mathsf{SN}}(T)$ as expected.

— $\mathsf{RED}^{\mathsf{SN}}(T) \subseteq \mathsf{SNorm}(T)$:
Assume that $t \in \mathsf{RED}^{\mathsf{SN}}(T)$. The $\pi_1(t) \in \mathsf{RED}^{\mathsf{SN}}(U)_1$ by definition and, by induction hypothesis on U, $\pi_1(t) \in \mathsf{SNorm}(U)_1$. The longest reduction from t is certainly at least as long as that from $\pi_1(t)$ so there is only finite reduction sequence from t and $t \in \mathsf{SNorm}(T)$.

□

Proposition 2.12

The following holds:

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

Démonstration :

1. $0 \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Nat})$ since $\mathsf{RED}^{\mathsf{SN}}(\mathsf{Nat}) = \mathsf{SNorm}(\mathsf{Nat})$ and 0 is a T-normal form.
2. $\mathsf{true}, \mathsf{false} \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Bool})$ since $\mathsf{RED}^{\mathsf{SN}}(\mathsf{Bool}) = \mathsf{SNorm}(\mathsf{Bool})$ and $\mathsf{true}, \mathsf{false}$ are T-normal forms.
3. let $t \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Nat})$, then t is strongly normalizable since Nat is an atomic type. Since any reduction from $\mathsf{S}(t)$ is of the form $\mathsf{S}(t) \longrightarrow \mathsf{S}(t_1) \longrightarrow \mathsf{S}(t_2) \longrightarrow \dots \mathsf{S}(t_n) \longrightarrow \dots$, with $t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \cdots \longrightarrow t_n \longrightarrow \dots$, $\mathsf{S}(t)$ is also strongly normalizable and therefore $\mathsf{S}(t) \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Nat})$.
4. let $t : \mathsf{Bool}$, $u, v : U$ be such that $t \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Bool})$, $u, v \in \mathsf{RED}^{\mathsf{SN}}(U)$. By adaptation lemma, it is sufficient to prove that $w = \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v \in \mathsf{Neut}^{\mathsf{SN}}(U)$ to deduce $w \in \mathsf{RED}^{\mathsf{SN}}(U)$. 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 $\forall t \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Bool})$, $\forall u, v \in \mathsf{RED}^{\mathsf{SN}}(U)$, $\mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v \in \mathsf{RED}^{\mathsf{SN}}(U)$.
   $w = \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v$ 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 $\ell(t') + \ell(u) + \ell(v) < \ell(t) + \ell(u) + \ell(v)$, $w' \in \mathsf{RED}^{\mathsf{SN}}(U)$ by induction hypothesis;
   — or $w' = \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}'\, v$ with $u \longrightarrow_{\mathsf{T}} u'$ and since $\ell(t) + \ell(u') + \ell(v) < \ell(t) + \ell(u) + \ell(v)$, $w' \in \mathsf{RED}^{\mathsf{SN}}(U)$ by induction hypothesis;
   — or $w' = \mathsf{if}\, t\, \mathsf{then}\, u\, \mathsf{else}\, v'$ with $v \longrightarrow_{\mathsf{T}} v'$ and since $\ell(t) + \ell(u) + \ell(v') < \ell(t) + \ell(u) + \ell(v)$, $w' \in \mathsf{RED}^{\mathsf{SN}}(U)$ by induction hypothesis.
5. let $t : \mathsf{Nat}$, $u : \mathsf{Nat} \to (U \to U)$ and $v : U$ be such that $t \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Nat})$, $u \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Nat} \to (U \to U))$ and $v \in \mathsf{RED}^{\mathsf{SN}}(U)$. As above, it is sufficient to prove that $w = \mathsf{Rec}(t, u, v) \in \mathsf{Neut}^{\mathsf{SN}}(U)$. w being neutral we simply have to prove that any of its one-step reducts is in $\mathsf{RED}^{\mathsf{SN}}(U)$ 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 measuer as well):

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

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

$w = \mathsf{Rec}(t, u, v)$ is neutral, let us consider its one-step reducts: if $w \longrightarrow_{\mathsf{T}} w'$ then
— either w' is v if $t = 0$ 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 $t \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Nat})$, $u \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Nat} \to (U \to U))$, ensures that $w' \in \mathsf{RED}^{\mathsf{SN}}(U)$. *[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 $w' = \mathsf{Rec}t'uv$ with $t \longrightarrow_{\mathsf{T}} t'$ and since $\ell(t') + \ell(u) + \ell(v) < \ell(t) + \ell(u) + \ell(v)$ and since $n(t) = n(t')$, $w' \in \mathsf{RED}^{\mathsf{SN}}(U)$ 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 $w' = \mathsf{Rec}tu'v$ with $u \longrightarrow_{\mathsf{T}} u'$ and since $\ell(t) + \ell(u') + \ell(v) + n(t) < \ell(t) + \ell(u) + \ell(v) + n(t)$, $w' \in \mathsf{RED}^{\mathsf{SN}}(U)$ 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 $w' = \mathsf{Rec}tuv'$ with $v \longrightarrow_{\mathsf{T}} v'$ and since $\ell(t) + \ell(u) + \ell(v') + n(t) < \ell(t) + \ell(u) + \ell(v) + n(t)$, $w' \in \mathsf{RED}^{\mathsf{SN}}(U)$ 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 suffice to deduce that $w \in \mathsf{Neut}^{\mathsf{SN}}(U) \subseteq \mathsf{RED}^{\mathsf{SN}}(U)$.

□

For STLC, the following lemma was used in the proof of adequation:

Lemma 2.13

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

The following is a corresponding result for pairs:

Lemma 2.14

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

Démonstration : 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 $\pi_i(\langle u, v\rangle) \in \mathsf{Neut}^{\mathsf{SN}}(U)$ 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 $\langle u, v\rangle \in \mathsf{RED}^{\mathsf{SN}}(U) \times V$.

□

Lemma 2.15 (*Adequation*)

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

Démonstration du lemme 2.15 : One reason by induction on the structure of $t : T$.
— If $t = x_i^{T_i}$, as for STLC.
— If $t = \lambda x^U.t'$, as for STLC.
— If $t = (u)v$, as for STLC.

- 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 T-constant, it is reducible (since $\mathsf{0} \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Nat})$, $\mathsf{true}, \mathsf{false} \in \mathsf{RED}^{\mathsf{SN}}(\mathsf{Bool})$).
- 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 $\mathsf{Rec}(u\{u_i/x_i\}, v\{u_i/x_i\}, w\{u_i/x_i\})$.

□

Theorem 2.16

System T *is strongly normalizing.*

Démonstration : Let $t : T$ of free variables $(x_i^{T_i})_{1\leq i \leq n}$. Be adaptation lemma (2.10) for any $1 \leq i \leq n$, $x_i^{T_i} \in \mathsf{RED}^{\mathsf{SN}}(T_i)$ since variables of type T are neutral and normal and therefore in $\mathsf{Neut}^{\mathsf{SN}}(T)$.

Adequation lemma (2.15) ensures that $t\left\{x_i^{T_i}/x_i, 1 \leq i \leq n\right\} = t$ is reducible of type T ($\in \mathsf{RED}^{\mathsf{SN}}(T)$).

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

□

2.3 Expressive power of system T

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

Exercice 2.1

Write T*-terms for the standard boolean functions.*

Simple arithmetical functions represented by T-terms. It is simple to defined basic arithmetical functions on type Nat: instead of manipulating Church numerals, one works with the built-in naturalnumbers of 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:

The successor function can be written as:
- $\mathsf{Succ} \triangleq \lambda x^{\mathsf{Nat}}.\mathsf{S}(x)$ or
- $\mathsf{Succ}' \triangleq \lambda x^{\mathsf{Nat}}.\mathsf{Rec}(x, \lambda y^{\mathsf{Nat}}.\lambda z^{\mathsf{Nat}}.z, \mathsf{S}(\mathsf{0}))$

Addition can be defined as:

$$\mathsf{Add} \triangleq \lambda x^{\mathsf{Nat}}.\lambda y^{\mathsf{Nat}}.\mathsf{Rec}(x, \lambda z^{\mathsf{Nat}}.\mathsf{Succ}, y).$$

Multiplication can be defined in the same way,

$$\mathsf{Mult} \triangleq \lambda x^{\mathsf{Nat}}.\lambda y^{\mathsf{Nat}}.\mathsf{Rec}(x, \lambda z^{\mathsf{Nat}}.(\mathsf{Add})y, \mathsf{0}).$$

Exponentiation can be defined in the same way,

$$\mathsf{Exp} \triangleq \lambda x^{\mathsf{Nat}}.\lambda y^{\mathsf{Nat}}.\mathsf{Rec}(y, \lambda z^{\mathsf{Nat}}.(\mathsf{Mult})x, \mathsf{S}(\mathsf{0})).$$

Predecessor can be defined easily, taking opportunity of the fact that the natural on which we recurse is passed as the first argument of the functional that we iterate:

$$\mathsf{Pred} \triangleq \lambda x^{\mathsf{Nat}}.\mathsf{Rec}(x, \lambda y^{\mathsf{Nat}}.\lambda z^{\mathsf{Nat}}.y, \mathsf{0}).$$

Contrarily to the simply typed case, the predecessor is computed in constant time in T, not linear time.

Subtraction can be then be defined easily by iterating predecessor:

$$\mathsf{Subt} \triangleq \lambda x^{\mathsf{Nat}}.\lambda y^{\mathsf{Nat}}.\mathsf{Rec}(y, \lambda z^{\mathsf{Nat}}.\mathsf{Pred}, x).$$

Ackermann-Péter function in T. Notice here that the type of the recursors we have been using sofar is very simple: U is always taken to be 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 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 T, we would need a T-term A such that

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

However, it is well-known that Ackermann-Peter function is not primitive recursive and in system 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 exactely what the recursor of system 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: $A_{m+1}(n) = \mathit{iter}(A_m,n)$ where $\mathit{iter}(f,0) = f(1)$ and $\mathit{iter}(f,n+1) = f(\mathit{iter}(f,n))$.

Now, we see clearly how to complete the definition of A:

— Consider $\mathsf{Iter} \triangleq \lambda f^{\mathsf{Nat}\to\mathsf{Nat}}.\lambda x^{\mathsf{Nat}}.\mathsf{Rec}(x, \lambda y^{\mathsf{Nat}}.f, (f)\mathsf{S}(\mathsf{0}))$ to represent the iteration function described above.

— We can define the T-term representing Ackermann-Peter function as

$$\mathsf{A} \triangleq \lambda x^{\mathsf{Nat}}.\mathsf{Rec}(x, \lambda z^{\mathsf{Nat}}.\mathsf{Iter}, \mathsf{Succ}).$$

Exercice 2.2

Prove that A *indeed represents Ackermann-Peter function, that is:*

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

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

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

Todo: develop this...

Characterization of the expressiveness of T. More generally, the extended expressiveness of T that was mentioned in the start is expressed by the following theorem:

Theorem 2.17

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