Lambda Calculus and Types

Nikos Tzeveλekos
Queen Mary, University of London

Lecture 4: Simple Types

Some Words about Types

- The untyped λ-calculus is very unconstrained, allowing e.g. self-application (xx). As a result:
  - The calculus is very expressive (see Definability Theorem),
  - but it is also *ill-behaved* (see behaviour of Ω).
- The solution, due to Curry and Church following Russell is to introduce *types*. The idea is that:

  Types are there to stop you doing bad things

  - In particular, types impose the intuitive distinction between a *function* and its *argument*.
  - It has turned out that types constitute one of the most fruitful *positive* ideas in Computer Science!

- There are very many type systems for the λ-calculus.
  (See H. P. Barendregt's book *Lambda Calculi With Types*, available online.)
- We are only interested in ***simple*** (no polymorphism) type systems, to which there are two non-equivalent approaches:
  - Curry's "type inference/assignment" system;
  - Church's "typed terms" system.

  We focus on the former and proceed to define a *simple-types system* $\boldsymbol{\lambda}^{\mathbb{A}}_{\rightarrow}$ which will consist of:
  - A set of types **Typ**.
  - A set of terms Λ (i.e. the set of λ-terms).
  - A *typing relation* $\vdash$ which will allow us to assign types to (some) terms, i.e. prove *typing judgements* of the form:

$$x_1 : A_1, \ldots, x_n : A_n \vdash s : A$$

Types

We assume a countable set $\mathbb{A}$ of ***type variables***, distinct from λ-variables.
From them we build **Typ**, the set of ***types***, by the following rules.

$$\frac{}{a \in \textbf{Typ}}\ a \in \mathbb{A} \qquad \frac{A \in \textbf{Typ} \quad B \in \textbf{Typ}}{(A \to B) \in \textbf{Typ}}$$

Notes:

- We use lowercase Roman letters a, b, c, etc, to range over type variables.
- We use uppercase Roman letters A, B, C, etc, to range over types.
- We usually omit outermost parentheses: $A \to B \equiv (A \to B)$.
- We use the convention that, unless otherwise bracketed, nested arrows associate to the **right**:
$$A \to B \to C \ \equiv \ (A \to (B \to C))$$
- Types which are type variables are sometimes called ***atomic*** types.
- Types of the form $A \to B$ are called ***composite types***, ***function types***, etc.

Type Contexts

A ***type context*** is a finite (possibly empty) set of λ-variable/type pairs, i.e:

$$\Gamma = \{\, x_1 : A_1, \ldots, x_n : A_n \,\}$$

Notes:

- We use uppercase Greek letters Γ, Δ, Δ', etc, to range over type contexts.
- The x_i's must be **distinct**.
  Thus, contexts can be viewed as *finite partial functions:* $\mathcal{V} \to$ **Typ**.
  In fact, when $(x : A) \in \Gamma$ we say that x is *assigned the type* A *in* Γ.
- When $\Gamma = \{\, x_1 : A_1, \ldots, x_n : A_n \,\}$ we write $\mathrm{Subj}(\Gamma)$ for the set of variables $\{\, x_1, \ldots, x_n \,\}$. These are the ***subjects*** of the context Γ.
- We write Γ, Γ' for the context $\Gamma \cup \Gamma'$, where it is assumed that $\mathrm{Subj}(\Gamma)$ and $\mathrm{Subj}(\Gamma')$ are disjoint.
- We usually omit the empty context altogether from typing judgements.

Typing Rules

We say that a λ-term s is ***typable*** if there is a type A and context Γ such that:

$$\Gamma \vdash s : A$$

We define the typing relation by the following rules.

$$\text{(var)}\ \frac{}{\Gamma, x : A \vdash x : A} \qquad \text{(app)}\ \frac{\Gamma \vdash s : B \to A \quad \Gamma \vdash t : B}{\Gamma \vdash st : A} \qquad \text{(abs)}\ \frac{\Gamma, x : B \vdash s : A}{\Gamma \vdash \lambda x.s : B \to A}$$

Notes:

- We apply the **variable convention** specifically as follows:

  In any typing judgement $\Gamma \vdash s : A$, we silently α-convert s so that its bound variables are not the same as the subjects of Γ.

- For any types A and B:

$$\dfrac{\dfrac{}{x : A, y : B \vdash x : A}}{x : A \vdash \lambda y.x : B \to A}$$
$$\vdash \lambda xy.x : A \to B \to A$$

  We call the above tree a *derivation* of $\vdash \lambda xy.x : A \to B \to A$.

- For any A, B, C and D:

$$\vdash \lambda xyz.z(\lambda p.p) : A \to B \to ((C \to C) \to D) \to D$$

- Thus, $\lambda xy.x$ and $\lambda xyz.z(\lambda p.p)$ are typable. Other examples of typable terms:

$$x,\ \mathbf{i},\ \mathbf{k},\ (\lambda x.x)(\lambda y.y),\ \ulcorner n \urcorner$$

- Some examples of untypable terms:

$$xx,\ \lambda x.xx,\ \mathbf{\Omega},\ (\lambda xy.y)\mathbf{\Omega},\ \mathbf{y}$$

Weakening. If $\Gamma \vdash s : A$ and $\Gamma \subseteq \Gamma'$ then $\Gamma' \vdash s : A$.

Subject Construction. Consider the derivation tree of $\Gamma \vdash s : A$.

- If s is a variable, say $s \equiv x$, then the whole derivation is:

$$\frac{}{\Gamma \vdash x : A}$$

- If s is an application, say $s \equiv pq$, then the final step of the derivation must be of the form:

$$\frac{\Gamma \vdash p : B \to A \qquad \Gamma \vdash q : B}{\Gamma \vdash pq : A}$$

for *some* type B

- If s is an abstraction, say $s \equiv \lambda x.p$ with $x \notin \mathrm{Subj}(\Gamma)$, then A must be of the form $C \to D$ and the final step of the derivation must be of the form:

$$\frac{\Gamma, x : C \vdash p : D}{\Gamma \vdash \lambda x.p : C \to D}$$

Derivations are not Unique

- The Subject Construction property tells us that the structure of s determines much of the derivation $\Gamma \vdash s : A$.
- However, not quite all is determined:

$$\dfrac{\dfrac{\dfrac{}{x : B \to B, y : A \vdash y : A}}{x : B \to B \vdash \lambda y.y : A \to A}}{\vdash \lambda xy.y : (B \to B) \to A \to A} \qquad \dfrac{\dfrac{}{z : B \vdash z : B}}{\vdash \lambda z.z : B \to B}$$
$$\vdash (\lambda xy.y)(\lambda z.z) : A \to A$$

  (the type B, not appearring anywhere in the conclusion, is arbitrary).

- Nevertheless, it is the case that derivations *are* unique when the term in the conclusion is a β-normal form.

- Typable terms do not have unique types. For example,

$$\dfrac{\dfrac{x : A, y : B \vdash x : A}{x : A \vdash \lambda y.x : B \to A}}{\vdash \lambda xy.x : A \to B \to A}$$

  is a valid derivation for **all** types A and B.

- But we cannot find a derivation of:

$$\vdash \lambda xy.x : A \to A \qquad \text{nor of:} \quad \vdash \lambda xy.x : A \to B \to C \ \ (\text{if } C \not\equiv A)$$

- One show that every typable term has a ***principal type***, of which every other type it can be given is an ***instance***.
  In the case of $\lambda xy.x$, the principal type is $a \to b \to a$.
- In fact, one can construct a ***principal typing algorithm*** which, given a term s:
  - if s is typable then it computes a principal type for s,
  - or correctly states that s is not typable.

  The algorithm is overly intricate to be covered here...

Lemma. *For any* $x \notin FV(t) \cup Subj(\Gamma)$,

- *if* $\Gamma, x : B \vdash s : A$ *and* $\Gamma \vdash t : B$ *then* $\Gamma \vdash s[t/x] : A$.

Proof. By induction on s.

If $s \equiv y \not\equiv x$ then $\Gamma, x : B \vdash y : A$ implies $\Gamma \vdash y : A$, which is $\Gamma \vdash s[t/x] : A$.

If $s \equiv x$ then it must be the case that $A \equiv B$ and, therefore, $\Gamma \vdash s[t/x] : A$ is in fact $\Gamma \vdash t : B$.

If $s \equiv pq$ then, by subject construction, the final step in the derivation for s must be:

$$\frac{\Gamma, x : B \vdash p : C \to A \qquad \Gamma, x : B \vdash q : C}{\Gamma, x : B \vdash pq : A}$$

for some type C. Applying the induction hypothesis we obtain:

$$\frac{\Gamma \vdash p[t/x] : C \to A \qquad \Gamma \vdash q[t/x] : C}{\Gamma, x : B \vdash p[t/x]q[t/x] : A \quad (\equiv s[t/x])}$$

The final case, $s \equiv \lambda y.p$, is dealt with in a similar manner. □

Theorem. *If* $\Gamma \vdash s : A$ *and* $s \to_\beta t$ *then* $\Gamma \vdash t : A$.

Proof. Use induction on the derivation of $s \to_\beta t$ applying the Substitution Lemma in the case of the last rule in the derivation being (β). □

Note, though, that ***Subject Expansion*** does not hold:

$$\Gamma \vdash t : A \ \wedge \ s \to_\beta t \implies \Gamma \vdash s : A$$

If terms which receive no type are considered “unsafe”, then it would be worrying for “safe” terms to reduce to “unsafe” terms (although the converse might happen).

We can now bring the theory of reduction from the untyped language into $\boldsymbol{\lambda}^{\mathbb{A}}_{\rightarrow}$.
Everything goes through:

- There is the same definition for **β-reduction**.
  The Subject Reduction Theorem ensures that it interacts properly with types.
- The **Church-Rosser** proof carries over.
- β-conversion is still **consistent** (the types don't affect it).

  However, as the converse to the Subject Reduction Theorem (*Subject Expansion*) does *not* hold, β-conversion does not interact properly with the type system.

  This might be unsatisfactory from a theoretical point of view, and people have considered ***Type Assignment with Equality*** systems.

- Recall that a term $s \in \Lambda$ is ***typable*** if there is a context Γ and a type A such that:

$$\Gamma \vdash s : A$$

- A term $s \in \Lambda$ is ***strongly normalising*** (w.r.t. β-reduction) if there is no infinite reduction sequence

$$s_0 \to_\beta s_1 \to_\beta s_2 \to_\beta \cdots$$

such that $s \equiv s_0$.

Thus, if s is SN then the *reduction tree of* s is finite (by König's Lemma).
In particular we can define the maximum length of path to normal form:

$$\nu(s) = \max\{n \in \mathbb{N} \mid \exists t_0, ..., t_n.\, s \equiv t_0 \to_\beta t_1 \to_\beta \cdots \to_\beta t_n \wedge t_n \text{ in } \beta\text{-nf}\}$$

Theorem. *All typable terms are strongly normalising w.r.t. β-reduction.*

To prove the SN-Theorem, we follow the strategy below.

- Define a set of *reducible terms* (by recursion on types).
- Prove that every reducible term is strongly normalising (induction on types).
- Prove that every typable term is reducible (induction on terms).

The proof we give here is an adaptation to Curry-style types of the one presented in the book *Proofs and Types* by J.-Y. Girard.

Reducible terms

Definition. For any type A and context Γ we define the set of ***reducible terms at*** Γ, A, written $\mathtt{Red}_{\Gamma,A}$, recursively over A:

- If A is atomic (a type variable) then $s \in \mathtt{Red}_{\Gamma,A}$ if $\Gamma \vdash s : A$ and s is strongly normalising.
- If $A \equiv B \to C$ then $s \in \mathtt{Red}_{\Gamma,A}$ if $\Gamma \vdash s : A$ and for all $t \in \Lambda$ and $\Gamma' \supseteq \Gamma$:

$$t \in \mathtt{Red}_{\Gamma',B} \implies st \in \mathtt{Red}_{\Gamma',C}$$

Lemma. *If* $s \in \mathtt{Red}_{\Gamma,A}$ *then* $\Gamma \vdash s : A$.

Lemma (Γ-Lemma)**.** *If* $s \in \mathtt{Red}_{\Gamma,A}$ *and* $\Gamma \subseteq \Gamma'$ *then* $s \in \mathtt{Red}_{\Gamma',A}$.

Reducible Terms are Strongly Normalising

Defn. $s \in \mathtt{Red}_{\Gamma,A}$ if $\Gamma \vdash s : A$ and:

- If $A \equiv a$ then s is SN.
- If $A \equiv B \to C$ then $\forall t, \Gamma' \supseteq \Gamma$, $t \in \mathtt{Red}_{\Gamma',B} \implies st \in \mathtt{Red}_{\Gamma',C}$

Theorem. *For all* $\Gamma \vdash s : A$*:*

(**CR1**) *If* $s \in \mathtt{Red}_{\Gamma,A}$ *then* s *is strongly normalising.*

(**CR2**) *If* $s \in \mathtt{Red}_{\Gamma,A}$ *and* $s \to_\beta s'$ *then* $s' \in \mathtt{Red}_{\Gamma,A}$.

(**CR3**) *If* s *is not a* λ*-abs. and* $\forall s'.(s \to_\beta s' \implies s' \in \mathtt{Red}_{\Gamma,A})$ *then* $s \in \mathtt{Red}_{\Gamma,A}$.

Proof. By induction on the type A. If A is a type variable:

1. Immediate from the definition of reducibility.
2. As s is strongly normalising, so are its one-step β-reducts (and by Subject Reduction they can be typed with Γ, A).
3. If all the one-step β-reducts of s are strongly normalising then so is s.

Defn. $s \in \mathtt{Red}_{\Gamma,A}$ if $\Gamma \vdash s : A$ and:

- If $A \equiv a$ then s is SN.
- If $A \equiv B \to C$ then $\forall t, \Gamma' \supseteq \Gamma$, $t \in \mathtt{Red}_{\Gamma',B} \implies st \in \mathtt{Red}_{\Gamma',C}$

Theorem. *For all* $\Gamma \vdash s : A$*:*

(**CR1**) *If* $s \in \mathtt{Red}_{\Gamma,A}$ *then* s *is strongly normalising.*

(**CR2**) *If* $s \in \mathtt{Red}_{\Gamma,A}$ *and* $s \to_\beta s'$ *then* $s' \in \mathtt{Red}_{\Gamma,A}$.

(**CR3**) *If* s *is not a* λ*-abs. and* $\forall s'.(s \to_\beta s' \implies s' \in \mathtt{Red}_{\Gamma,A})$ *then* $s \in \mathtt{Red}_{\Gamma,A}$.

Proof. By induction on the type A. If $A \equiv B \to C$:

1. Let $s \in \mathtt{Red}_{\Gamma,A}$ and take $\Gamma' = \Gamma, x : B$ for some fresh x.
   By IH on (**CR3**), $x \in \mathtt{Red}_{\Gamma',B}$. Hence, by definition, $sx \in \mathtt{Red}_{\Gamma',C}$.
   Then by IH on (**CR1**) we have that sx is SN and therefore s is SN.
2. Let $s \in \mathtt{Red}_{\Gamma,A}$ and $s \to_\beta s'$ so, by Subject Reduction, $\Gamma \vdash s' : A$.
   Now take some $t \in \mathtt{Red}_{\Gamma',B}$ for $\Gamma' \supseteq \Gamma$.
   By definition, $s \in \mathtt{Red}_{\Gamma,A}$ implies that $st \in \mathtt{Red}_{\Gamma',C}$ and therefore, by IH on (**CR2**), $s't \in \mathtt{Red}_{\Gamma',C}$.

Defn. $s \in \mathtt{Red}_{\Gamma,A}$ if $\Gamma \vdash s : A$ and:

- If $A \equiv a$ then s is SN.
- If $A \equiv B \to C$ then $\forall t, \Gamma' \supseteq \Gamma$, $t \in \mathtt{Red}_{\Gamma',B} \implies st \in \mathtt{Red}_{\Gamma',C}$

Theorem. *For all* $\Gamma \vdash s : A$*:*

(**CR1**) *If* $s \in \mathtt{Red}_{\Gamma,A}$ *then* s *is strongly normalising.*

(**CR2**) *If* $s \in \mathtt{Red}_{\Gamma,A}$ *and* $s \to_\beta s'$ *then* $s' \in \mathtt{Red}_{\Gamma,A}$.

(**CR3**) *If* s *is not a* λ*-abs. and* $\forall s'.(s \to_\beta s' \implies s' \in \mathtt{Red}_{\Gamma,A})$ *then* $s \in \mathtt{Red}_{\Gamma,A}$.

Proof. By induction on the type A. If $A \equiv B \to C$:

3. Finally, let s be a term that is not a λ-abstraction, and let all its one-step β-reducts be inside $\mathtt{Red}_{\Gamma,A}$.

   Take some $t \in \mathtt{Red}_{\Gamma',B}$ with $\Gamma' \supseteq \Gamma$. By IH on (**CR1**), t is SN.

   We show that $st \in \mathtt{Red}_{\Gamma',C}$ by induction on $\nu(t)$. By IH on (**CR3**), it suffices to show that if $st \to_\beta u$ then $u \in \mathtt{Red}_{\Gamma',C}$. There are two cases:

   - $u \equiv st'$ with $t \to_\beta t'$. Then $t' \in \mathtt{Red}_{\Gamma',B}$ by IH on (**CR2**), so use IH on $\nu(t')$.
   - $u \equiv s't$ with $s \to_\beta s'$, in which case use the hypothesis.

   In both cases, $u \in \mathtt{Red}_{\Gamma',C}$.

This completes the proof. □

Lemma. *Let* $\Gamma \vdash \lambda y.s : B \to A$. *Then if, for all* $t \in \mathtt{Red}_{\Gamma',B}$ *with* $\Gamma' \supseteq \Gamma$, *we have* $s[t/y] \in \mathtt{Red}_{\Gamma',A}$ *then* $\lambda y.s \in \mathtt{Red}_{\Gamma,B\to A}$.

Proof. Suppose $t \in \mathtt{Red}_{\Gamma',B}$ with $\Gamma' \supseteq \Gamma$; we show that $(\lambda y.s)t \in \mathtt{Red}_{\Gamma',A}$.

By (**CR3**), $y \in \mathtt{Red}_{\Gamma\cup\{y:B\},B}$ and thus, by hypothesis, $s \in \mathtt{Red}_{\Gamma\cup\{y:B\},A}$. So, by (**CR1**), both s, t are SN and therefore we can do induction on $\nu(s)+\nu(t)$.

Now let $(\lambda y.s)t \to_\beta u$; we show that $u \in \mathtt{Red}_{\Gamma',A}$. There are three cases:

- $u \equiv s[t/y]$, in which case use the hypothesis.
- $u \equiv (\lambda y.s)t'$ with $t \to_\beta t'$, in which case use the IH on $\nu(s)+\nu(t')$ and obtain $(\lambda y.s)t' \in \mathtt{Red}_{\Gamma',A}$.
- $u \equiv (\lambda y.s')t$ with $s \to_\beta s'$. Note that, for all $t' \in \mathtt{Red}_{\Gamma'',B}$ with $\Gamma'' \supseteq \Gamma$, we have that $s[t'/y] \twoheadrightarrow_\beta s'[t'/y]$ and therefore $s'[t'/y] \in \mathtt{Red}_{\Gamma'',A}$ by using the hypothesis and (**CR2**).
  Thus, we can use the IH on $\nu(s')+\nu(t)$ and obtain $(\lambda y.s')t \in \mathtt{Red}_{\Gamma',A}$.

Thus, by (**CR3**), $(\lambda y.s)t \in \mathtt{Red}_{\Gamma',A}$. □

Theorem. *Suppose* $x_1 : A_1, \cdots, x_n : A_n \vdash s : A$. *Then, for all* $u_1, \ldots, u_n \in \Lambda$ *and* Γ' *such that* $u_i \in \mathtt{Red}_{\Gamma',A_i}$, *we have* $s[u_1/x_1, ..., u_n/x_n] \in \mathtt{Red}_{\Gamma',A}$.

Proof. By induction on s. Let $\Gamma \equiv \{x_1 : A_1, \cdots, x_n : A_n\}$ and let $u_1, \ldots, u_n$ be as above. There are three cases:

- $s \equiv x_i$ for some i, in which case $s[\vec{u}/\vec{x}] \equiv u_i \in \mathtt{Red}_{\Gamma',A}$ by hypothesis.
- $s \equiv pq$. By Subject Construction, $\Gamma \vdash p : B \to A$ and $\Gamma \vdash q : B$ for some type B. Thus, $p[\vec{u}/\vec{x}] \in \mathtt{Red}_{\Gamma',B\to A}$ and $q[\vec{u}/\vec{x}] \in \mathtt{Red}_{\Gamma',B}$ by IH. But then, by definition of reducibility, $p[\vec{u}/\vec{x}]\, q[\vec{u}/\vec{x}] \in \mathtt{Red}_{\Gamma',A}$, as required.
- $s \equiv \lambda y.p$ (y not in Γ, Γ'). Then, $\Gamma, y : B \vdash p : C$ with $A \equiv B \to C$. Now, for all $t \in \mathtt{Red}_{\Gamma'',B}$ with $\Gamma'' \supseteq \Gamma'$, by the IH and the Γ-Lemma:

$$p[\vec{u}/\vec{x}, t/y] \in \mathtt{Red}_{\Gamma'',A}$$

that is, $(p[\vec{u}/\vec{x}])[t/y] \in \mathtt{Red}_{\Gamma'',A}$.
But then, by previous Lemma, $\lambda y.p[\vec{u}/\vec{x}] \in \mathtt{Red}_{\Gamma',B\to A}$. □

Theorem. *Suppose* $x_1 : A_1, \ldots, x_n : A_n \vdash s : A$. *Then, for all* $u_1, \ldots, u_n \in \Lambda$ *and* Γ' *such that* $u_i \in \mathtt{Red}_{\Gamma', A_i}$, *we have* $s[u_1/x_1, ..., u_n/x_n] \in \mathtt{Red}_{\Gamma', A}$.

Corrolary. *Let* $s \in \Lambda$ *be typable. Then* s *is strongly normalising.*

Proof. Suppose $\Gamma \vdash s : A$ with $\Gamma \equiv \{x_1 : A_1, \ldots, x_n : A_n\}$.

Then, by (**CR3**), $x_i \in \mathtt{Red}_{\Gamma, A_i}$ for each i and thus, by Theorem, $s \equiv s[\vec{x}/\vec{x}] \in \mathtt{Red}_{\Gamma, A}$.

By (**CR1**), s is strongly normalising. □

Consequences of Strong Normalisation

Strong normalisation marks the simply-typed λ-calculus very different from the untyped λ-calculus:

- Equality under β-conversion is decidable.
- There are no (typable) fixed point combinators in $\boldsymbol{\lambda}_{\rightarrow}^{\mathbb{A}}$.

The standard **Church numerals** are all typable and the successor combinator **succ** is typable and compatible with application to numerals.

Lemma. *The projection functions, constant functions, addition and multiplication are all definable in the simply-typed λ-calculus (w.r.t. Church numerals).*

Theorem (Schwichtenberg & Statman)**.** *Define the set of functions on the natural numbers called the **extended polynomials** to be the smallest set:*

- *containing projection functions, constant functions and the function $ifzero$, where $ifzero := \{\langle 0, m, n\rangle \mapsto m,\ \langle k+1, m, n\rangle \mapsto n\}$,*
- *and closed under addition and multiplication of functions.*

Then the functions definable in the simply typed λ-calculus are exactly the extended polynomials.

An Aside: Simple Types à la Church

Two distinct presentations of the simply-typed λ-calculus:

- Type Assignment (what we've seen so far): simple types *à la Curry*.
- Typed Terms: simple types *à la Church*.

$\mathcal{V}^A$'s disjoint

denoted by x^A, y^A, etc.

For the latter:

- Suppose that for each type A there is a countable set of variables $\mathcal{V}^A$.
- For each type A we define the set of ***terms of type*** A, written Λ^A, by the following mutually inductive rules.

$$\frac{}{x^A \in \Lambda^A}\ x^A \in \mathcal{V}^A \qquad \frac{s \in \Lambda^{A\to B} \quad t \in \Lambda^A}{st \in \Lambda^B} \qquad \frac{s \in \Lambda^B}{\lambda x^A.s \in \Lambda^{A\to B}}\ x^A \in \mathcal{V}^A$$

We have obvious parallel definitions of:

- Free and bound variables, closed terms.
- α-conversion (α-convertible terms are still syntactically equivalent).
- Substitution (we now first need to prove a Substitution Lemma).

- **Typed reduction**: There is a family of relations $\rightarrow_\beta^A$, one for each type A, denoting β-reduction on terms of each type.
- With this presentation, the **Subject Reduction Theorem** is immediate, and also the **Subject Expansion Theorem** holds.
- The **Church-Rosser property** can be proven as in the untyped case (but it cannot be carried over).
- **Typability**: We can relate typed terms to untyped ones by erasing type information.

Definition. Define a family of maps, $|-| : \Lambda^A \rightarrow \Lambda$ for each type A, by:

$$|x^A| = x, \quad |\lambda x^A.s| = \lambda x.|s|, \quad |st| = |s||t|$$

We say that a term $s \in \Lambda$ is ***typable*** if there is a term $s' \in \Lambda^A$, for some type A, with $|s'| = s$.

Exercises

1. Give derivations of the following.
   (a) $x : A \vdash (\lambda y.y)x : A$
   (b) $\vdash \lambda xy.xyy : (A \to A \to B) \to A \to B$
   (c) $\vdash \lambda xy.xy : (A \to B) \to A \to B$
2. Show that there is no type A such that $\vdash \lambda xy.yxy : A$.
   Is the same true for $\vdash \lambda xy.yxx : A$?
3. Write down closed terms s which can be given the following types (in other words, find a closed term s such that $\vdash s : A$).
   (a) $(A \to A) \to A \to A$
   (b) $(A \to B \to C) \to (A \to B) \to A \to C$

   Can you find a closed term s such that $\vdash s : a \to b$, for different type variables a and b?
4. Show that, for all $n \in \mathbb{N}$, $\vdash \ulcorner n \urcorner : (a \to a) \to a \to a$, with a a type variable.
5. Let s be a term in β-normal form.
   (a) Show that every subterm of s which is not a λ-abstraction is of the form $xs_1 \cdots s_n$, for some variable x and some terms $s_1, \ldots, s_n$ with $n \geq 0$.
   (b) Show that, if $\Gamma \vdash s : A$, for some Γ, A, then the derivation of the latter is unique.
6. Prove the Subject Reduction Theorem on slide 12.
7. The Strong Normalisation Theorem shows that every term which is typable is strongly normalising. Show the converse is false, i.e. that there is a term which is strongly normalising but not typable.
8. We saw that a consequence of the Strong Normalisation Theorem is that equality under β-conversion is decidable. How does that reconcile with the Scott-Curry Theorem (and its consequent lemmas)?
9. Show that, if there exists a (untyped) fixed point combinator which has a normal form, then there exists a fixed point combinator which is in normal form.
   Deduce that no fixed point combinator has a normal form.
   [Hint: consider a fixed point combinator F in normal form and suppose that $Fx =_\beta x(Fx)$, for a fresh variable x. By considering reducts of Fx and $x(Fx)$, derive a contradiction.]