lambda-calculus & categories 8

Monday 23 November 2020

# the simply-typed λ-calculus

# The simply-typed $\lambda$-calculus

the calculus of functions with types.

We start with a set of <u>type variables</u>

$TVar$ ← countable

A simple type is defined by the grammar

$$A ::= \alpha \in TVar \mid A \times B \mid A \Rightarrow B \mid \mathbb{1}$$

hence every type $A$ is a tree constructed with $\times$, $\Rightarrow$, $\mathbb{1}$ and with leaves in $TVar$.

def. a context is a finite sequence

$$x_1 : A_1, \ldots, x_n : A_n$$

of pairs $x_i : A_i$ consisting of

- a term variable $x_i$
- a type $A_i$

We also make the assumption that all the variables $x_i$ are different.

def: a typing judgement is a triple

$$x_1 : A_1, \ldots, x_n : A_n \vdash M : B$$

consisting of

- a context $\Gamma = x_1 : A_1, \dots, x_n : A_n$
- a $\lambda$-term $M$
- a type $B$

One could ask that all free var of $M$ are in $\Gamma$

We want to show that a given $\lambda$-term $M$ has a given type $B$ in a given context $\Gamma$ describing the types of the free variables of $M$.

To that purpose, we construct

derivation trees

establing the typing judgement

Def. a derivation tree is a tree whose branches are labelled by a typing judgement and whose nodes are labelled by the following typing rules:

6 rules

## Logical rules

$$\text{var}\ \frac{}{x : A \vdash x : A}$$

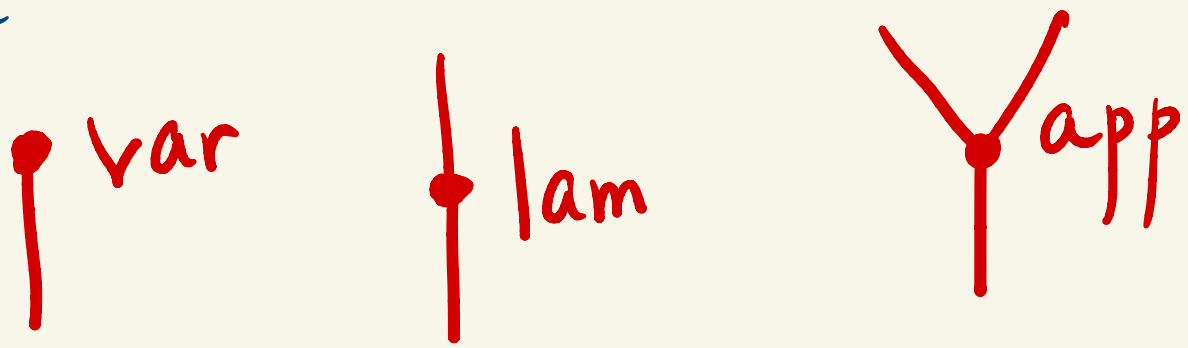
$$\text{lam}\ \frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \Rightarrow B}$$

introduction of $\Rightarrow$

$$\text{app}\ \frac{\Gamma \vdash M : A \Rightarrow B \quad \Delta \vdash P : A}{\Gamma, \Delta \vdash \text{App}(M, P) : B}$$

elimination of $\Rightarrow$

have different variables in the context



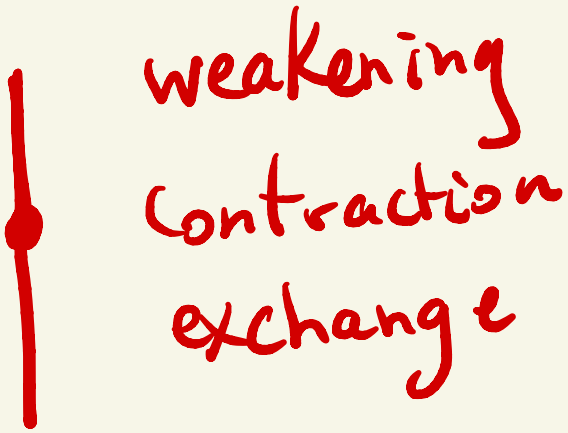

the three "logical" rules

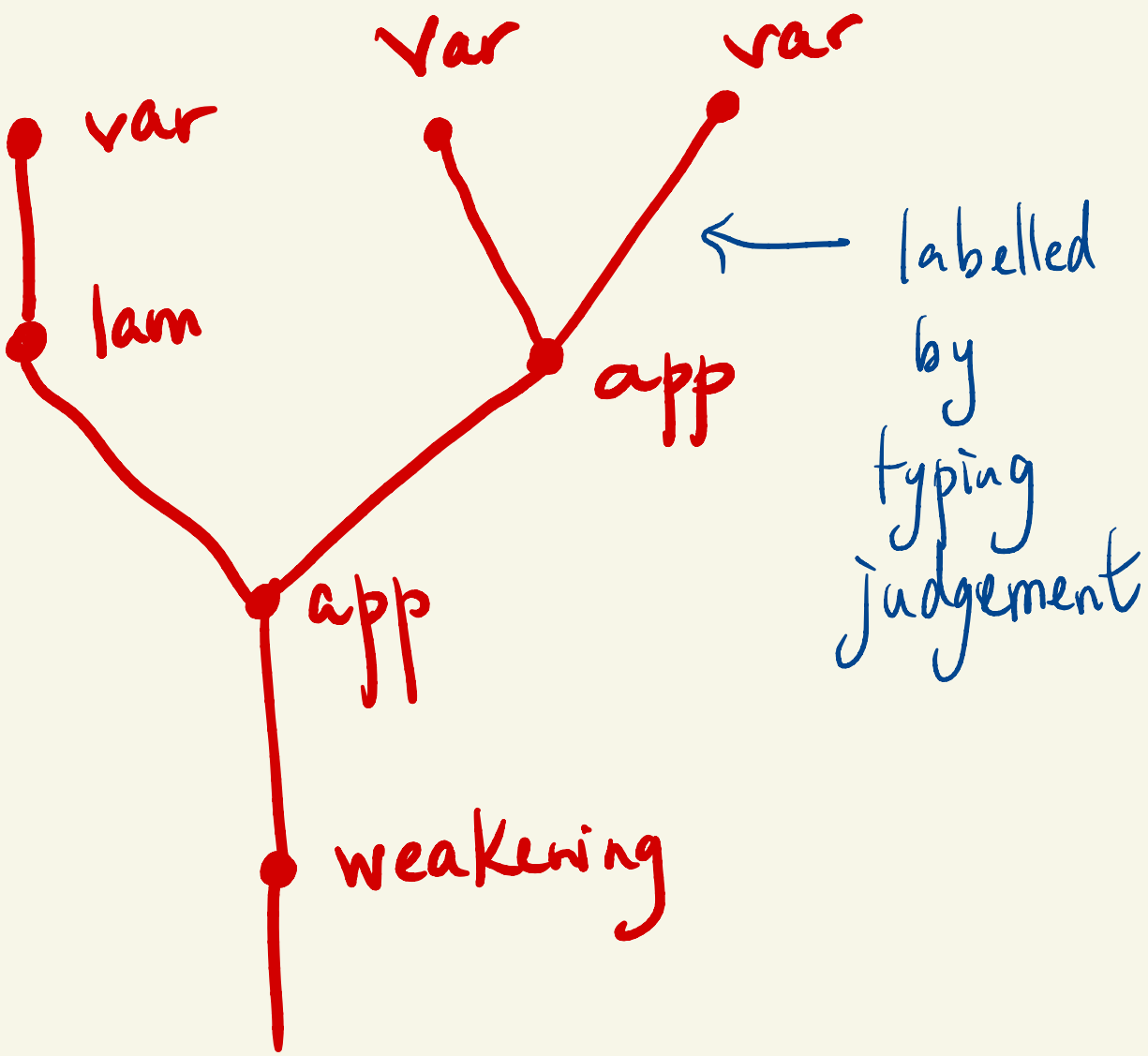
Structural rules

$$\frac{\Gamma, \Delta \vdash M : B}{\Gamma, x : A, \Delta \vdash M : B}\ \text{weakening}$$

$$\frac{\Gamma, x : A, y : A, \Delta \vdash M : B}{\Gamma, z : A, \Delta \vdash M[x, y := z] : B}\ \text{contraction}$$

$$\frac{\Gamma, x : A, y : B, \Delta \vdash M : C}{\Gamma, y : B, x : A, \Delta \vdash M : C}\ \text{exchange}$$




the shape of a derivation tree

Let us construct a derivation tree establishing that the Church numeral

$$[1] = \lambda f.\, \lambda x.\, App(f, x)$$

function ↑ argument ↑

has type

$$(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$$

with type variables $\alpha$ and $\beta$.

a machine that establishes it!

The intuition is that the argument $x$ has type $\alpha$ while the function $f$ has type $\alpha \Rightarrow \beta$.

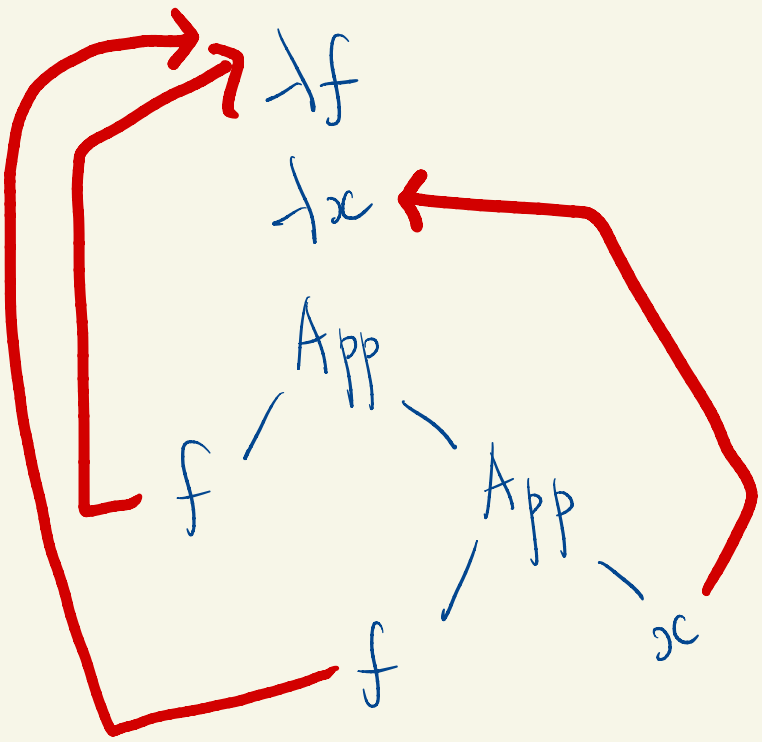
$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{f : \alpha \Rightarrow \beta \vdash f : \alpha \Rightarrow \beta}\,\text{var} \quad \dfrac{}{x : \alpha \vdash x : \alpha}\,\text{var}}{f : \alpha \Rightarrow \beta,\ x : \alpha \vdash App(f, x) : \beta}\,\text{app}}{f : \alpha \Rightarrow \beta \vdash \lambda x.\, App(f, x) : \alpha \Rightarrow \beta}\,\text{lam}}{\vdash \lambda f.\, \lambda x.\, App(f, x) : (\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta}\,\text{lam}$$

In a polymorphic type system like System F we could write

$$\vdash [1] : \underbrace{\forall\alpha\,\forall\beta}_{\text{quantification on types}}\ (\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta.$$

What about the Church numeral $[2]$?

$$\lambda f.\ \lambda x.\ App(f, App(f, x))$$



the intuition is that [2] should be of type:

$$(\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$$

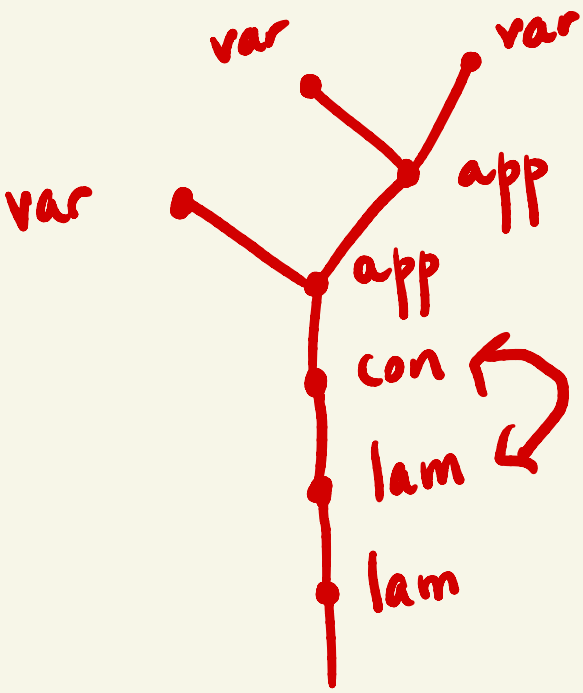
where the argument $x$ is of type $\alpha$ and the function $f$ is of type $\alpha \Rightarrow \alpha$

**our derivation tree**

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{h:\alpha\Rightarrow\alpha \vdash h:\alpha\Rightarrow\alpha}\,\text{var} \quad \dfrac{}{x:\alpha\vdash x:\alpha}\,\text{var}}{h:\alpha\Rightarrow\alpha, x:\alpha \vdash App(h,x):\alpha}\,App \quad \dfrac{}{g:\alpha\Rightarrow\alpha\vdash g:\alpha\Rightarrow\alpha}\,\text{var}}{g:\alpha\Rightarrow\alpha, h:\alpha\Rightarrow\alpha, x:\alpha \vdash App(g, App(h,x)):\alpha}\,App}{f:\alpha\Rightarrow\alpha, x:\alpha \vdash App(f, App(f,x)):\alpha}\,\text{contraction}}{\vdash \lambda f.\lambda x.\, App(f, App(f,x)) : (\alpha\Rightarrow\alpha)\Rightarrow\alpha\Rightarrow\alpha}\,\text{lam twice}$$

our derivation tree

"seen from far above"

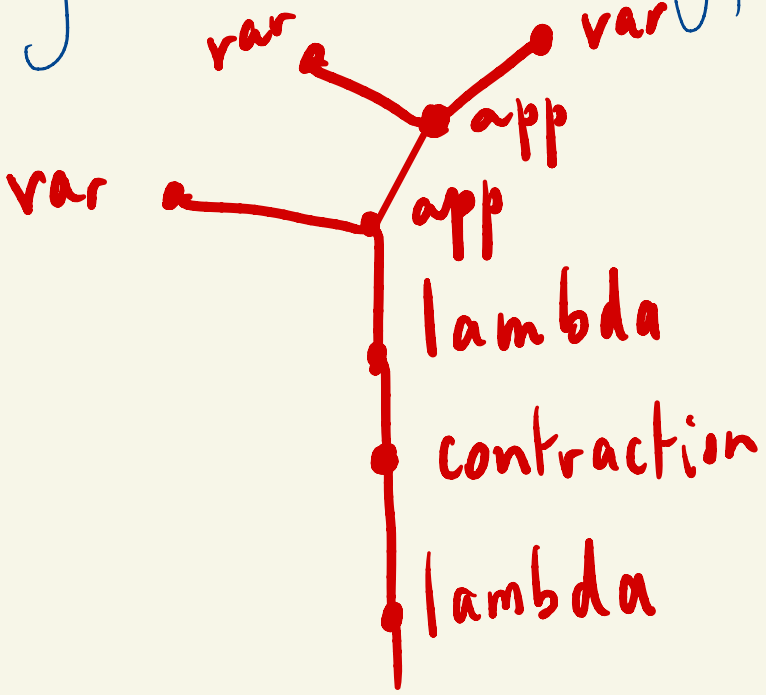



edges labelled by typing judgements

nodes labelled by rules

Let me ask a simple question:

are there other derivation trees establishing the same typing judgement?



What about the Church numeral $[0]$?

$$[0] = \lambda f.\, \lambda x.\, x$$

the intuition is that it should be

$$\gamma \Rightarrow (\alpha \Rightarrow \alpha)$$

where $\alpha$ and $\gamma$ are type variables.

$$\dfrac{\dfrac{\dfrac{}{x : \alpha \vdash x : \alpha}\ \text{var}}{f : \gamma,\ x : \alpha \vdash x : \alpha}\ \text{weakening}}{f : \gamma \vdash \lambda x.\, x : \alpha \Rightarrow \alpha}\ \text{lam}$$

could be permuted

$$\dfrac{f : \gamma \vdash \lambda x.\, x : \alpha \Rightarrow \alpha}{\vdash \lambda f.\, \lambda x.\, x : \gamma \Rightarrow \alpha \Rightarrow \alpha}\ \text{lam}$$

To summarize:

[1] is of type $(\alpha \Rightarrow \beta) \Rightarrow \alpha \Rightarrow \beta$

[2] is of type $(\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$

[0] is of type $\gamma \Rightarrow \alpha \Rightarrow \alpha$

hence [0] is of type $(\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$

(simply substitute $\alpha \Rightarrow \alpha$ for $\gamma$)

and similarly

[1] is of type $(\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$

(simply substitute $\alpha$ for $\beta$)

More generally, every Church numeral [n] has type $(\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$

$[n] = \lambda f. \lambda x. \; f \ldots f \, (x)$

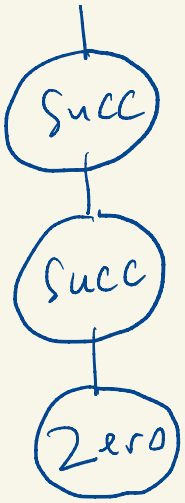
Church numerals translate terms / trees of signature

Succ : 1  Zero : 0

Zero

Succ Zero

Succ Succ Zero




More generally

there is a Church encoding

for terms / trees

associated to any given signature $\Sigma$

def: a signature $\Sigma$ is a family

$$(\Sigma_n)_{n \in \mathbb{N}}$$ of sets.

example: $\Sigma_0 = \{Zero\}$  $\Sigma_1 = \{Succ\}$  $\Sigma_n = \emptyset$ $n \geq 2$

A tree of signature $\Sigma$ is

either an element of $\Sigma_0$ (= a constant)

or a pair $(f, (t_1, \dots, t_n))$

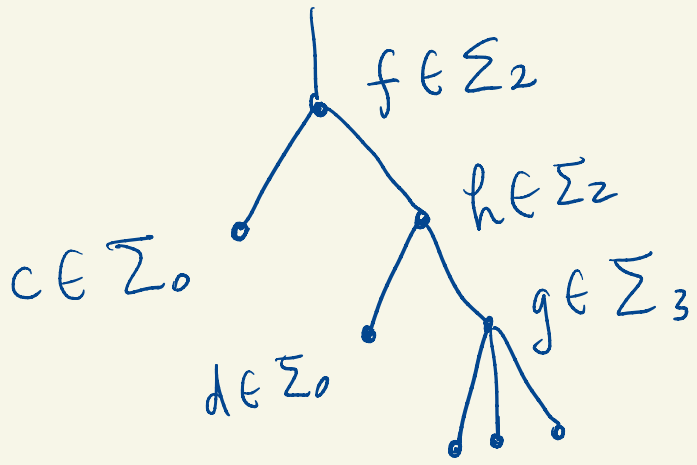
consisting of an element $f \in \Sigma_n$

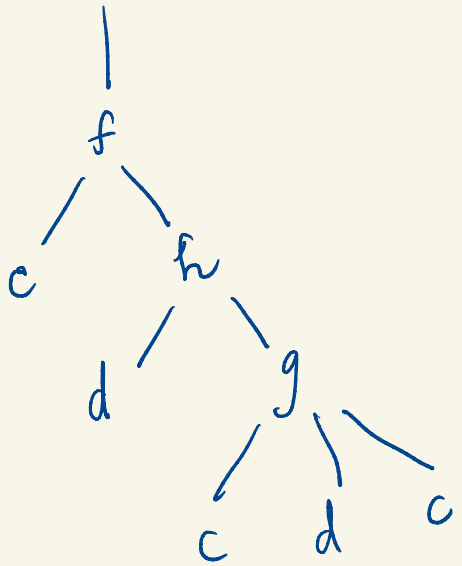
(= an operation of arity $n$)

and a sequence $(t_1, \dots, t_n)$ of trees

of signature $\Sigma$.

$t ::= f(t_1, \dots, t_n)$ where $f \in \Sigma_n$.




$f(c, h(d, g(c, d, c)))$

In fact, it is possible to translate

every signature $\Sigma$ (finite)

into a type $[\Sigma]$ of the simply-typed $\lambda$-calculus

such that every term $t$ of the signature $\Sigma$

can be encoded as a $\lambda$-term $[t]$ of type $[\Sigma]$

the encoding of $\Sigma$ is very nice and simple:

(1) pick a type variable $\alpha$

(2) understand every operation $f \in \Sigma_n$

think of

as a "function" $[f]$

of type $\underbrace{\alpha \Rightarrow \alpha \Rightarrow \alpha \Rightarrow \cdots \Rightarrow \alpha}$

$[f]$ is a n-ary function

example: $[\text{Succ}] : \alpha \Rightarrow \alpha$ $\quad$ $[\text{Zero}] : \alpha$

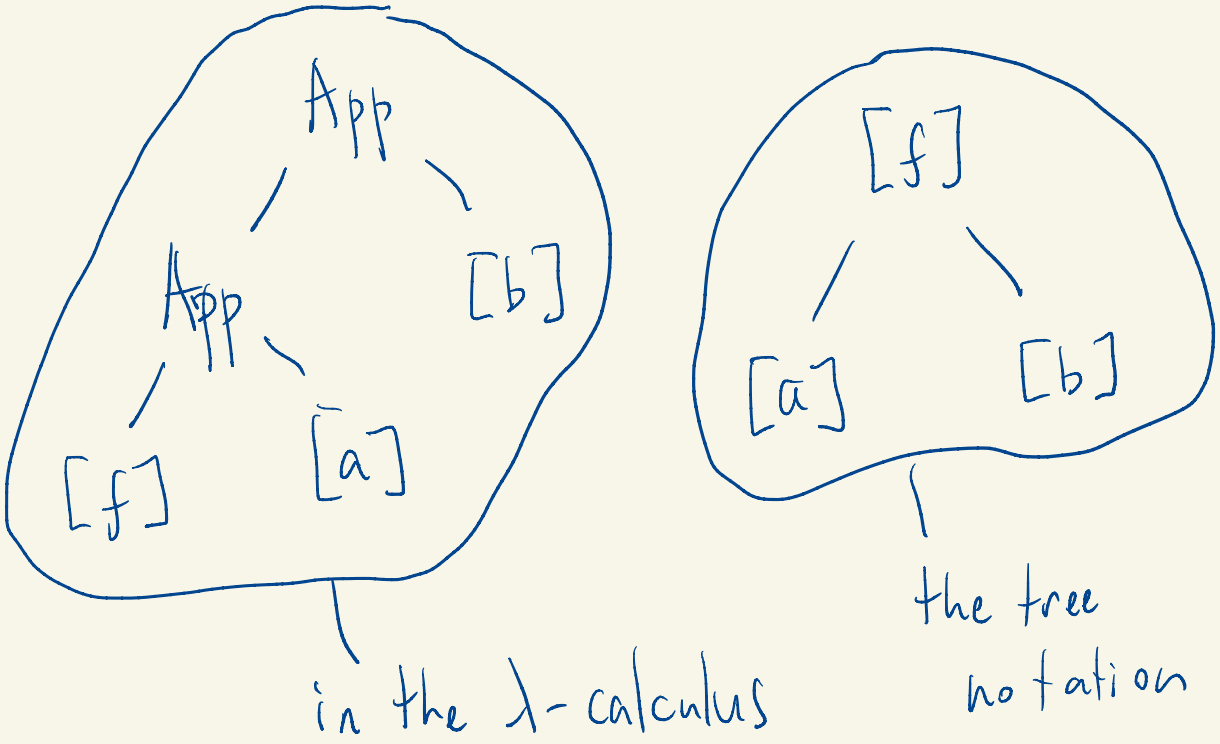
for a binary operation $f \in \Sigma_2$

$$[f] : \alpha \Rightarrow \alpha \Rightarrow \alpha .$$

③ think of a tree of signature $\Sigma$ as a $\lambda$-term obtained by applying the functions $[f]$ in the expected way.

typically:

$$[f(a,b)] = App(App([f],[a]),[b])$$



④ think of the resulting $\lambda$-term $[t]$ as a simply-typed $\lambda$-term

$[f]$ $\underbrace{\alpha \Rightarrow \alpha \ldots \Rightarrow \alpha \Rightarrow \alpha}_{f \in \Sigma_n} \vdash [t] : \alpha$

for all operations $f \in \Sigma_n$ for some $n \in \mathbb{N}$.

not really a notation for variable so just use the notation $f$.

Hence, a context of the $\lambda$-calculus can be seen a higher-order generalization of the notion of signature !

Hence every signature $\Sigma$

can be turned into a context $[\Sigma]$

example: $\mathrm{Zero} \in \Sigma_0$ $\mathrm{Succ} \in \Sigma_1$ $\Sigma_n = \emptyset$ otherwise

$$[\Sigma] = \mathrm{Zero} : \alpha,\ \mathrm{Succ} : \alpha \Rightarrow \alpha$$

in such a way that every term $t$ of signature $\Sigma$

can be Church encoded into a $\lambda$-term $[t]$

with typing judgement:

$$[\Sigma] \vdash [t] : \alpha$$

of course we can also "currify" the context and turn it into a sequence of $\lambda$'s.

example: $\mathrm{Zero} : \alpha,\ \mathrm{Succ} : \alpha \Rightarrow \alpha \vdash \mathrm{Zero} : \alpha$

$$Zero : \alpha,\ Succ : \alpha \Rightarrow \alpha \vdash App(Succ, Zero) : \alpha$$

$$Zero : \alpha,\ Succ : \alpha \Rightarrow \alpha \vdash App(Succ, App(Succ, Zero)) : \alpha$$

after "currification":

$$[0] = \lambda Succ\ .\ \lambda Zero.\ Zero : (\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$$

$$[1] = \lambda Succ.\ \lambda Zero.\ App(Succ, Zero) : (\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha$$

$$[2] = \lambda Succ\ .\ \lambda Zero.\ App(Succ, App(Succ, Zero)) :$$

$$\underbrace{(\alpha \Rightarrow \alpha) \Rightarrow \alpha \Rightarrow \alpha}$$

the type $[\Sigma]$ of the signature $\Sigma$.

deeply connected to automata theory!

this leads to an extension of automata theory from tree language (recognized by a tree automaton) to $\lambda$-term languages. (idea: $\alpha$ is the set of states of the automaton)

**Def.** a $\lambda$-term $M$ is in normal form when there is no $\beta$-redex $M \longrightarrow N$.

in other words, $M$ does not contain any pattern of $\beta$-rule.

**Fact:** given a signature $\Sigma$ there is a **one-to-one** relationship between:

(1) the trees $t$ of signature $\Sigma$

(2) the simply-typed $\lambda$-terms $M$ in normal form of type:

$$[\Sigma] \quad \vdash \quad M : \alpha$$

$[\Sigma]$: Church encoding of the signature $\Sigma$

**Next time:**

we will study/explain how every

**$\lambda$-term $M$** of type:

$$x_1 : A_1, \dots, x_k : A_k \vdash M : B$$

can be turned into

**a morphism**

$$A_1 \times \dots \times A_k \xrightarrow{M} B$$

in any cartesian closed category $\mathcal{C}$.

here $A_1, \dots, A_k$ are the interpretations of the type $A_1, \dots, A_k$ in the category $\mathcal{C}$