# Introduction to Type Theory

August 2007

Types Summer School

Bertinoro, It

Herman Geuvers

Nijmegen NL

Lecture 1: Introduction, Overview, Simple Type Theory

## Types are not sets.

- Types are a bit like sets, but: ...
- types give "syntactic information"

$$3 + (7 * 8)^5 : \mathsf{nat}$$

- sets give "semantic information"

$$3 \in \{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^+(x^n + y^n \neq z^n)\}$$

Sets are about semantics:

$$3 \in \{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)\}$$

because there are no positive $x, y, z$ such that $x^n + y^n = z^n$.

- set theory talks about what things exist (semantics, ontology). A set $X$ such that for all sets $Y$ with $|Y| < |X|$, $|2^Y| < |X|$?

- sets are extensional:

$$\{n \in \mathbb{N} \mid \exists x, y, z \in \mathbb{N}^+ (x^n + y^n = z^n)\} = \{0, 1, 2\}$$

- sets are "collections of things".

- membership is undecidable

Types are about syntax:

$$3 + (7 * 8)^5 : \mathsf{nat}$$

because $3$, $7$, $8$ are of type nat and the operations take objects of type nat to nat.

$$\frac{1}{2}\Sigma_{n=0}^{\infty} 2^{-n} : \mathbb{N}$$

is not a typing judgment.

- type theory talks about how things can be constructed (syntax, formal language, expressions)
- types are intensional

  $$\{n | \exists x, y, z \in \mathsf{nat}^+ (x^n + y^n \neq z^n)\} \neq \mathsf{nat}.\{n | n = 0 \vee n = 1 \vee n = 2\}$$

- types are "predicates over expressions"
- typing (and type checking) is decidable

Note. The distinction between syntax and semantics is not always as sharp as it seems.
The more we know about a model, the more we can formalize of it and "turn it into syntax".
We can turn

$$\{n \in \mathbb{N} \mid \exists x, y, z \in \mathbb{N}^+(x^n + y^n = z^n)\}$$

into a (syntactic) type, with decidable type checking, if we take as its terms pairs

$$\langle n, p\rangle : \{n \in \mathbb{N} \mid \exists x, y, z \in \mathbb{N}^+(x^n + y^n = z^n)\}$$

where $p$ is a proof of $\exists x, y, z \in \mathbb{N}^+(x^n + y^n = z^n$.

Proof checking is decidable; proof finding not.

Overview of these lectures.

Problem: there so many type systems and so many ways of defining them . . . .

Central theme: two readings of typing judments

$$M : A$$

- $M$ is a term (program, expression) of the data type $A$
- $M$ is a proof (derivation) of the formula $A$

Curry-Howard isomorphism of formulas-as-types
(and proofs-as-terms)

Overview of these lectures.

| Logic | | TT a la Church | AKA | | TT a la Curry |
|---|---|---|---|---|---|
| PROP | $\xrightarrow{\text{f}-\text{as}-\text{t}}$ | $\lambda\!\rightarrow$ | STT | | $\lambda\!\rightarrow$ |
| PROP2 | $\xrightarrow{\text{f}-\text{as}-\text{t}}$ | $\lambda 2$ | system F | | $\lambda 2$ |

| | | | | | Extra features! |
|---|---|---|---|---|---|
| PRED | $\xrightarrow{\text{f}-\text{as}-\text{t}}$ | $\lambda$P | LF | $\xleftarrow{\text{f}-\text{as}-\text{t}}$ | Many logics |
| HOL | $\xrightarrow{\text{f}-\text{as}-\text{t}}$ | $\lambda$HOL | | | language of HOL is STT |
| HOL | $\xrightarrow{\text{f}-\text{as}-\text{t}}$ | CC | Calc. of Constr. | | |
| | | PTS | | | different PTSs for HOL |

Simplest system: $\lambda\!\to$ or STT. Just arrow types

$$\mathsf{Typ} := \mathsf{TVar} \mid (\mathsf{Typ}{\to}\mathsf{Typ})$$

- Examples: $(\alpha{\to}\beta){\to}\alpha$, $(\alpha{\to}\beta){\to}((\beta{\to}\gamma){\to}(\alpha{\to}\gamma))$
- Brackets associate to the right and outside brackets are omitted: $(\alpha{\to}\beta){\to}(\beta{\to}\gamma){\to}\alpha{\to}\gamma$
- Types are denoted by $\sigma, \tau, \ldots$.

Terms:

- typed variables $x_1^\sigma, x_2^\sigma, \ldots$, countably many for every $\sigma$.
- application: if $M : \sigma{\to}\tau$ and $N : \sigma$, then $(MN) : \tau$
- abstraction: if $P : \tau$, then $(\lambda x^\sigma.P) : \sigma{\to}\tau$

Examples:

$$\begin{array}{rcl} \lambda x^{\sigma}.\lambda y^{\tau}.x & : & \sigma{\to}\tau{\to}\sigma \\ \lambda x^{\alpha\to\beta}.\lambda y^{\beta\to\gamma}.\lambda z^{\alpha}.y(xz) & : & (\alpha{\to}\beta){\to}(\beta{\to}\gamma){\to}\alpha{\to}\gamma \\ \lambda x^{\alpha}.\lambda y^{(\beta\to\alpha)\to\alpha}.y(\lambda z^{\beta}.x) & : & \alpha{\to}((\beta{\to}\alpha){\to}\alpha){\to}\alpha \end{array}$$

For every type there is a term of that type:

$$x^{\sigma} : \sigma$$

Not for every type there is a closed term of that type:

$(\alpha{\to}\alpha){\to}\alpha$ is not inhabited

[That is: there is no closed term of type $(\alpha{\to}\alpha){\to}\alpha$.]

Typed Terms versus Type Assignment:

- With typed terms also called typing à la Church, we have terms with type information in the $\lambda$-abstraction

$$\lambda x^{\alpha}.x : \alpha{\to}\alpha$$

As a consequence:

  - Terms have unique types,
  - The type is directly computed from the type info in the variables.

- With typed assignment also called typing à la Curry, we assign types to untyped $\lambda$-terms

$$\lambda x.x : \alpha{\to}\alpha$$

As a consequence:

  - Terms do not have unique types,
  - A principal type can be computed using unification.

Examples:

- Typed Terms:

$$\lambda x^{\alpha}.\lambda y^{(\beta\to\alpha)\to\alpha}.y(\lambda z^{\beta}.x))$$

  has only the type $\alpha\to((\beta\to\alpha)\to\alpha)\to\alpha$

- Type Assignment:

$$\lambda x.\lambda y.y(\lambda z.x))$$

  can be assigned the types

  - $\alpha\to((\beta\to\alpha)\to\alpha)\to\alpha$
  - $\alpha\to((\beta\to\alpha)\to\gamma)\to\gamma$
  - $(\alpha\to\alpha)\to((\beta\to\alpha\to\alpha)\to\gamma)\to\gamma$
  - ...

  with $\alpha\to((\beta\to\alpha)\to\gamma)\to\gamma$ being the principal type

Connection between Church and Curry typed STT:

Definition The erasure map $|-|$ from STT à la Church to STT à la Curry is defined by erasing all type information.

$$\begin{array}{rcl} |x^\alpha| & := & x \\ |M\,N| & := & |M|\,|N| \\ |\lambda x^\alpha.M| & := & \lambda x.|M| \end{array}$$

So, e.g.

$$|\lambda x^\alpha.\lambda y^{(\beta\to\alpha)\to\alpha}.y(\lambda z^\beta.x))| = \lambda x.\lambda y.y(\lambda z.x))$$

Theorem If $M:\sigma$ in STT à la Church, then $|M|:\sigma$ in STT à la Curry.

Theorem If $P:\sigma$ in STT à la Curry, then there is an $M$ such that $|M|\equiv P$ and $M:\sigma$ in STT à la Church.

Example of computing a principal type:

$$\lambda x^\alpha . \lambda y^\beta . \underbrace{y^\beta (\lambda z^\gamma . \overbrace{y^\beta x^\alpha}^{\delta})}_{\varepsilon}$$

1. Assign type vars to all variables: $x : \alpha, y : \beta, z : \gamma$.
2. Assign type vars to all applicative subterms: $y\,x : \delta$, $y(\lambda z.y\,x) : \varepsilon$.
3. Generate equations between types, necessary for the term to be typable: $\beta = \alpha{\to}\delta$ $\quad\quad$ $\beta = (\gamma{\to}\delta){\to}\varepsilon$
4. Find a most general unifier (a substitution) for the type vars that solves the equations: $\alpha := \gamma{\to}\delta,\ \ \beta := (\gamma{\to}\delta){\to}\varepsilon,\ \ \delta := \varepsilon$
5. The principal type of $\lambda x.\lambda y.y(\lambda z.yx)$ is now

$$(\gamma{\to}\varepsilon){\to}((\gamma{\to}\varepsilon){\to}\varepsilon){\to}\varepsilon$$

Exercise: Compute principal types for $\mathbf{S} := \lambda x.\lambda y.\lambda z.x\,z(y\,z)$ and for $M := \lambda x.\lambda y.x(y(\lambda z.x\,z\,z))(y(\lambda z.x\,z\,z))$.

Definition

- A type substitution (or just substitution) is a map $S$ from type variables to types.
  Note: we can compose substitutions.

- A unifier of the types $\sigma$ and $\tau$ is a substitution that "makes $\sigma$ and $\tau$ equal", i.e. an $S$ such that $S(\sigma) = S(\tau)$

- A most general unifier (or mgu) of the types $\sigma$ and $\tau$ is the "simplest substitution" that makes $\sigma$ and $\tau$ equal, i.e. an $S$ such that
  - $S(\sigma) = S(\tau)$
  - for all substitutions $T$ such that $T(\sigma) = T(\tau)$ there is a substitution $R$ such that $T = R \circ S$.

All these notions generalize to lists of types $\sigma_1, \ldots, \sigma_n$ in stead of pairs $\sigma, \tau$.

**Theorem** **Computability of most general unifiers**

There is an algorithm $U$ that, when given types $\sigma_1, \ldots, \sigma_n$ outputs

- A most general unifier of $\sigma_1, \ldots, \sigma_n$, if $\sigma_1, \ldots, \sigma_n$ can be unified.
- "Fail" if $\sigma_1, \ldots, \sigma_n$ can't be unified.

**Definition** $\sigma$ is a principal type for the untyped $\lambda$-term $M$ if

- $M : \sigma$ in STT à la Curry
- for all types $\tau$, if $M : \tau$, then $\tau = S(\sigma)$ for some substitution $S$.

Theorem Principal Types There is an algorithm PT that, when given an (untyped) $\lambda$-term $M$, outputs

- A principal type $\sigma$ such that $M : \sigma$ in STT à la Curry.
- "Fail" if $M$ is not typable in STT à la Curry.

Typical problems one would like to have an algorithm for:

| | | |
|---|---|---|
| $M : \sigma?$ | Type Checking Problem | TCP |
| $M : ?$ | Type Synthesis or Type Assgnment Problem | TSP, TAP |
| $? : \sigma$ | Type Inhabitation Problem (by a closed term) | TIP |

For $\lambda{\rightarrow}$, all these problems are decidable,
both for the Curry style and for the Church style presentation.
Remarks:

- TCP and TSP are (usually) equivalent: To solve $MN : \sigma$, one has to solve $N :?$ (and if this gives answer $\tau$, solve $M : \tau{\rightarrow}\sigma$).
- For Curry systems, TCP and TSP soon become undecidable if we go beyond $\lambda{\rightarrow}$.
- TIP is undecidable for most extensions of $\lambda{\rightarrow}$, as it corresponds to provability in some logic.

In this course we will mainly focus on the Church formulation of simple type theory:terms with type information.

Inductive definition of the terms:

- typed variables $x_1^\sigma, x_2^\sigma, \ldots$, countably many for every $\sigma$.
- application: if $M : \sigma \to \tau$ and $N : \sigma$, then $(MN) : \tau$
- abstraction: if $P : \tau$, then $(\lambda x^\sigma . P) : \sigma \to \tau$

Alternative: Inductive definition of the terms in rule form:

$$\frac{}{x^\sigma : \sigma} \qquad \frac{M : \sigma \to \tau \quad N : \sigma}{MN : \tau} \qquad \frac{P : \tau}{\lambda x^\sigma . P : \sigma \to \tau}$$

Advantage: We also have a derivation tree, a proof of the fact that the term has that type.
We can reason over derivations.

Simple type theory a la Church.

Formulation with contexts to declare the free variables:

$$x_1 : \sigma_1, x_2 : \sigma_2, \ldots, x_n : \sigma_n$$

is a context, usually denoted by $\Gamma$.

Derivation rules of $\lambda\!\rightarrow$ (à la Church):

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \sigma{\rightarrow}\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau} \qquad \frac{\Gamma, x{:}\sigma \vdash P : \tau}{\Gamma \vdash \lambda x{:}\sigma.P : \sigma{\rightarrow}\tau}$$

$\Gamma \vdash_{\lambda\rightarrow} M : \sigma$ if there is a derivation using these rules with conclusion $\Gamma \vdash M : \sigma$

## Derivation rules Church vs. Curry

$\lambda\!\rightarrow$ (à la Church):

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \sigma{\rightarrow}\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M N : \tau} \qquad \frac{\Gamma, x{:}\sigma \vdash P : \tau}{\Gamma \vdash \lambda x{:}\sigma.P : \sigma{\rightarrow}\tau}$$

$\lambda\!\rightarrow$ (à la Curry):

$$\frac{x{:}\sigma \in \Gamma}{\Gamma \vdash x : \sigma} \qquad \frac{\Gamma \vdash M : \sigma{\rightarrow}\tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash M N : \tau} \qquad \frac{\Gamma, x{:}\sigma \vdash P : \tau}{\Gamma \vdash \lambda x.P : \sigma{\rightarrow}\tau}$$

Exercise: Give a full derivation of

$$\vdash \lambda x.\lambda y.y(\lambda z.y\,x) : (\gamma{\rightarrow}\varepsilon){\rightarrow}((\gamma{\rightarrow}\varepsilon){\rightarrow}\varepsilon){\rightarrow}\varepsilon$$

in Curry style $\lambda\!\rightarrow$

Formulas-as-Types (Curry, Howard):

There are two readings of a judgement $M : \sigma$

1. term as algorithm/program, type as specification:
   $M$ is a function of type $\sigma$

2. type as a proposition, term as its proof:
   $M$ is a proof of the proposition $\sigma$

- There is a one-to-one correspondence:

  typable terms in $\lambda\!\rightarrow$ $\simeq$ derivations in minimal proposition logic

- The judgement

$$x_1 : \tau_1, x_2 : \tau_2, \ldots, x_n : \tau_n \vdash M : \sigma$$

  can be read as
  $M$ is a proof of $\sigma$ from the assumptions $\tau_1, \tau_2, \ldots, \tau_n$.

Example

$$\cfrac{\cfrac{\cfrac{\cfrac{\cfrac{[\alpha\to\beta\to\gamma]^3 \quad [\alpha]^1}{\beta\to\gamma} \qquad \cfrac{[\alpha\to\beta]^2 \quad [\alpha]^1}{\beta}}{\gamma}\;1}{\alpha\to\gamma}\;2}{(\alpha\to\beta)\to\alpha\to\gamma}\;3}{(\alpha\to\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma} \quad \simeq \quad \begin{array}{c} \lambda x{:}\alpha\to\beta\to\gamma.\lambda y{:}\alpha\to\beta.\lambda z{:}\alpha.xz(yz) \\ :(\alpha\to\beta\to\gamma)\to(\alpha\to\beta)\to\alpha\to\gamma \end{array}$$

Example

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{[x : \alpha{\to}\beta{\to}\gamma]^3 \quad [z : \alpha]^1}{xz : \beta{\to}\gamma} \qquad \dfrac{[y : \alpha{\to}\beta]^2 \quad [z : \alpha]^1}{yz : \beta}}{xz(yz) : \gamma}}{\lambda z{:}\alpha.xz(yz) : \alpha{\to}\gamma}\,1}{\lambda y{:}\alpha{\to}\beta.\lambda z{:}\alpha.xz(yz) : (\alpha{\to}\beta){\to}\alpha{\to}\gamma}\,2}{\lambda x{:}\alpha{\to}\beta{\to}\gamma.\lambda y{:}\alpha{\to}\beta.\lambda z{:}\alpha.xz(yz) : (\alpha{\to}\beta{\to}\gamma){\to}(\alpha{\to}\beta){\to}\alpha{\to}\gamma}\,3$$

Exercise: Give the derivation that corresponds to

$$\lambda x.\lambda y.y(\lambda z.y\,x) : (\gamma{\to}\varepsilon){\to}((\gamma{\to}\varepsilon){\to}\varepsilon){\to}\varepsilon$$

Computation:

- $\beta$-reduction: $(\lambda x{:}\sigma.M)P \longrightarrow_\beta M[P/x]$
- $\eta$-reduction: $\lambda x{:}\sigma.Mx \longrightarrow_\eta M$ if $x \notin \mathsf{FV}(M)$

Cut-elimination in minimal logic $=$ $\beta$-reduction in $\lambda{\to}$.

$$
\dfrac{\dfrac{\begin{matrix}[\sigma]^1 \\ \mathcal{D}_1 \\ \tau\end{matrix}}{\sigma\to\tau}\,1 \qquad \dfrac{\mathcal{D}_2}{\sigma}}{\tau}
\quad\longrightarrow\quad
\begin{matrix}\mathcal{D}_2 \\ \sigma \\ \mathcal{D}_1 \\ \tau\end{matrix}
$$

$$
\dfrac{\dfrac{\begin{matrix}[x : \sigma]^1 \\ \mathcal{D}_1 \\ M : \tau\end{matrix}}{\lambda x{:}\sigma.M : \sigma\to\tau}\,1 \qquad \dfrac{\mathcal{D}_2}{P : \sigma}}{(\lambda x{:}\sigma.M)P : \tau}
\quad\longrightarrow_\beta\quad
\begin{matrix}\mathcal{D}_2 \\ P : \sigma \\ \mathcal{D}_1 \\ M[P/x] : \tau\end{matrix}
$$

Properties of $\lambda\!\rightarrow$.

- Uniqueness of types

  If $\Gamma \vdash M : \sigma$ and $\Gamma \vdash M : \tau$, then $\sigma = \tau$.

- Subject Reduction

  If $\Gamma \vdash M : \sigma$ and $M \longrightarrow_{\beta\eta} N$, then $\Gamma \vdash N : \sigma$.

- Strong Normalization

  If $\Gamma \vdash M : \sigma$, then all $\beta\eta$-reductions from $M$ terminate.

- Substitution property

  If $\Gamma, x : \tau, \Delta \vdash M : \sigma$, $\Gamma \vdash P : \tau$, then $\Gamma, \Delta \vdash M[P/x] : \sigma$.

- Thinning

  If $\Gamma \vdash M : \sigma$ and $\Gamma \subseteq \Delta$, then $\Delta \vdash M : \sigma$.

- Strengthening

  If $\Gamma, x : \tau \vdash M : \sigma$ and $x \notin \mathsf{FV}(M)$, then $\Gamma \vdash M : \sigma$.

Normalization of $\beta$ for $\lambda\!\rightarrow$.

Note:

- Terms may get larger under reduction
  $(\lambda f.\lambda x.f(fx))P \longrightarrow_\beta \lambda x.P(Px)$
- Redexes may get multiplied under reduction.
  $(\lambda f.\lambda x.f(fx))((\lambda y.M)Q) \longrightarrow_\beta \lambda x.((\lambda y.M)Q)(((\lambda y.M)Q)x)$
- New redexes may be created under reduction.
  $(\lambda f.\lambda x.f(fx))(\lambda y.N) \longrightarrow_\beta \lambda x.(\lambda y.N)((\lambda y.N)x)$

First: Weak Normalization

- Weak Normalization: there is a reduction sequence that terminates,
- Strong Normalization: all reduction sequences terminate.

## Towards Weak Normalization

There are three ways in which a "new" $\beta$-redex can be created.

- Creation

$$(\lambda x. \dots x\, P \dots)(\lambda y.Q) \longrightarrow_\beta \dots (\lambda y.Q)P \dots$$

- Multiplication

$$(\lambda x. \dots x \dots x \dots)((\lambda y.Q)R) \longrightarrow_\beta \dots (\lambda y.Q)R \dots (\lambda y.Q)R \dots$$

- Identity

$$(\lambda x.x)(\lambda y.Q)R \longrightarrow_\beta (\lambda y.Q)R$$

Towards Weak Normalization

**Definition**

The height (or order) of a type $h(\sigma)$ is defined by

- $h(\alpha) := 0$
- $h(\sigma_1 \to \ldots \to \sigma_n \to \alpha) := \max(h(\sigma_1), \ldots, h(\sigma_n)) + 1.$

NB [Exercise] This is the same as defining

- $h(\sigma \to \tau) := \max(h(\sigma) + 1, h(\tau)).$

**Definition**

The height of a redex $(\lambda x{:}\sigma.P)Q$ is the height of the type of $\lambda x{:}\sigma.P$

Towards Weak Normalization

Definition

We give a measure $m$ to the terms by defining $m(N) := (h(N), \#N)$ with

- $h(N)$ = the maximum height of a redex in $N$,
- $\#N$ = the number of redexes of height $h(N)$ in $N$.

The measures of terms are ordered lexicographically:

$$(h_1, x) <_l (h_2, y) \text{ iff } h_1 < h_2 \text{ or } (h_1 = h_2 \text{ and } x < y)$$

.

Theorem [Weak Normalization]
If $P$ is a typable term in $\lambda\!\rightarrow$, then there is a terminating reduction starting from $P$.

Proof
Pick a redex of height $h(P)$ inside $P$ that does not contain any other redex of height $h(P)$. [Note that this is always possible!]
Reduce this redex, to obtain $Q$. This does not create a new redex of height $h(P)$. [This is the important step. Exercise: check this; use the three ways in which new redexes can be created.]
So $m(Q) <_l m(P)$
As there are no infinitely decreasing $<_l$ sequences, this process must terminate and then we have arrived at a normal form.

## Strong Normalization for $\lambda\!\rightarrow$ à la Curry

This is proved by constructing a model of $\lambda\!\rightarrow$.

### Definition

- $[\![\alpha]\!] :=$ SN (the set of strongly normalizing $\lambda$-terms).
- $[\![\sigma\!\rightarrow\!\tau]\!] := \{M \mid \forall N \in [\![\sigma]\!](MN \in [\![\tau]\!])\}$.

### Lemma (both by induction on $\sigma$)

- $[\![\sigma]\!] \subseteq$ SN
- If $M[N/x]\vec{P} \in [\![\sigma]\!]$, $N \in [\![\tau]\!]$, then $(\lambda x.M)N\vec{P} \in [\![\sigma]\!]$.

### Proposition

$$\left.\begin{array}{r} x_1{:}\tau_1,\ldots,x_n{:}\tau_n \vdash M : \sigma \\ N_1 \in [\![\tau_1]\!],\ldots,N_n \in [\![\tau_n]\!] \end{array}\right\} \Rightarrow M[N_1/x_1,\ldots N_n/x_n] \in [\![\sigma]\!]$$

**Proof** By induction on the derivation of $\Gamma \vdash M : \sigma$.

Proposition

$$\left.\begin{array}{r} x_1{:}\tau_1, \ldots, x_n{:}\tau_n \vdash M : \sigma \\ N_1 \in [\![\tau_1]\!], \ldots, N_n \in [\![\tau_n]\!] \end{array}\right\} \Rightarrow M[N_1/x_1, \ldots N_n/x_n] \in [\![\sigma]\!]$$

Corollary $\lambda\!\rightarrow$ is SN

Proof By taking $N_i := x_i$ in the Proposition.
Of course, then we first have to show that $x \in [\![\sigma]\!]$ for all $x$ and $\sigma$.

This is a consequence of the following
Lemma
$xN_1 \ldots N_k \in [\![\sigma]\!]$ for all $x$, $\sigma$ and $N_1, \ldots, N_k \in$ SN.
Proof Induction on $\sigma$.

A little bit on semantics

$\lambda\!\rightarrow$ has a simple set-theoretic model. Given sets $[\![\alpha]\!]$ for all type variables $\alpha$, define

$$[\![\sigma\!\rightarrow\!\tau]\!] := [\![\tau]\!]^{[\![\sigma]\!]} \ \text{( set theoretic function space } [\![\sigma]\!] \rightarrow [\![\tau]\!])$$

If any of the base sets $[\![\alpha]\!]$ is infinite, then there are higher and higher (uncountable) cardinalities among the $[\![\sigma]\!]$

There are smaller models, e.g.

$$[\![\sigma\!\rightarrow\!\tau]\!] := \{f \in [\![\sigma]\!] \rightarrow [\![\tau]\!] | f \text{ is definable}\}$$

where definability means that it can be constructed in some formal system. This restricts the collection to a countable set.

For example

$$[\![\sigma\!\rightarrow\!\tau]\!] := \{f \in [\![\sigma]\!] \rightarrow [\![\tau]\!] | f \text{ is } \lambda\text{-definable}\}$$