编程语言的设计原理
Design Principles of Programming Languages

Haiyan Zhao, Di Wang
赵海燕，王迪

Peking University, Spring Term 2023

Chap 29: Type Operators and Kinding

Type-Level Functions

Kinding

λ_ω

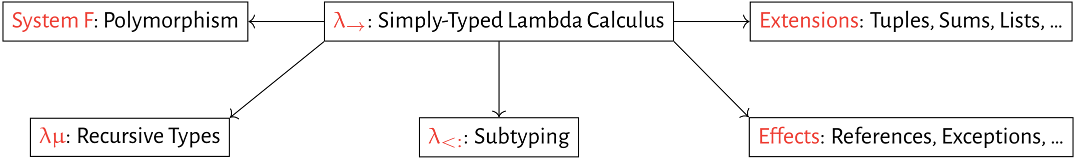
The Essence of λ

We Have Studied ...

Principle

The uses of type systems **go far beyond** their role in detecting errors.

- Type systems offer **crucial support** for programming: **abstraction, safety, efficiency, ...**
- Language design shall go **hand-in-hand** with type-system design.



Remark

- Different combinations of features lead to different languages.
- Some combinations turn out to be very **tricky**!

The Essence of λ

Principle

- **Types** characterize **terms**.
- Building abstractions:
  - In $\lambda_{\rightarrow}$, we use $\lambda x{:}T.\ t$ to abstract **terms** out of **terms**.
  - In System F, we use $\lambda X.\ t$ to abstract **terms** out of **types**.

Question

- Is it possible to further characterize **types**?
- Are these combinations meaningful?
  - Abstract **types** out of **types**? "$\lambda X.\ T$"?
  - Abstract **types** out of **terms**? "$\lambda x{:}T.\ T$"?

Characterization of Types

```
CBool = ∀X. X → X → X;
Pair Y Z = ∀X. (Y→Z→X) → X;
```

Abbreviation

Parametric Abbreviation

Observation

- `Pair` is like a **type-level function**.
- Similar notions include `Array` T and `Ref` T.

Abstract **Types** out of **Types**!

```
Pair = λY. λZ. ∀X. (Y→Z→X) → X
```

Type-Level Computation

Observation

Introducing abstraction and application at the type level allows **writing the same type in different ways**.

Example

$$\texttt{Id} = \lambda\texttt{X}.\ \texttt{X}$$

Then, the following types are all equivalent:

$$\texttt{Nat} \to \texttt{Bool} \qquad \texttt{Nat} \to \texttt{Id Bool} \qquad \texttt{Id Nat} \to \texttt{Id Bool}$$

$$\texttt{Id Nat} \to \texttt{Bool} \qquad \texttt{Id (Nat} \to \texttt{Bool)} \qquad \texttt{Id (Id (Id Nat} \to \texttt{Bool))}$$

Proposition

Let us denote the type-level reduction by $\Rrightarrow$.

Then two types S and T are equivalent iff there exists some U such that $S \Rrightarrow^* U$ and $T \Rrightarrow^* U$.

Kinding

Kinds

Observation

Type-level computation brings the issue of writing **meaningless type expressions**.

`(Bool Nat)`   `(Pair Bool Bool Nat)`   `(Pair Pair)`

Kinds: "**The Types of Types**"

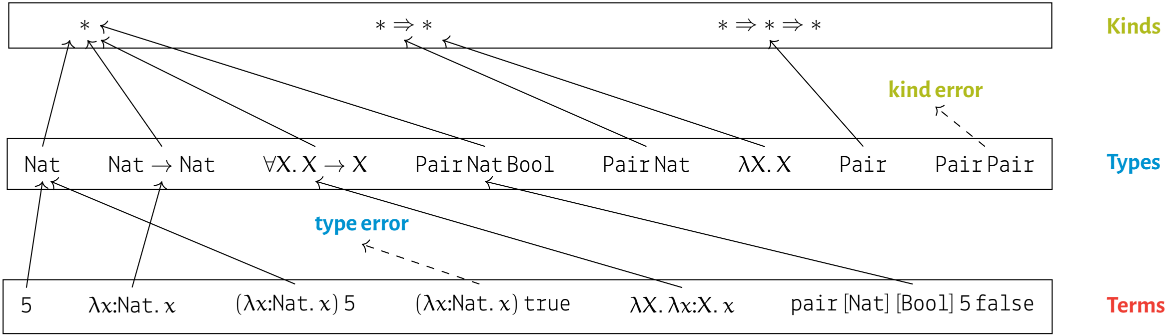
Kinds characterize **types**, in the same sense as **types** characterize **terms**.

$*$	the kind of proper types (like `Bool` and `Bool`$\to$`Bool`)
$* \Rightarrow *$	the kind of type operators (i.e., functions from proper types to proper types)
$* \Rightarrow * \Rightarrow *$	the kind of functions from proper types to type operators (i.e., two-argument operators)
$(* \Rightarrow *) \Rightarrow *$	the kind of functions from type operators to proper types

Type-Level Functions (λX::K. T)

$$\texttt{Pair} = \lambda Y :: *.\ \lambda Z :: *.\ \forall X.\ (Y \to Z \to X) \to X$$

Terms, Types, and Kinds



Question

- What is the difference between $\forall X.\, X \to X$ and $\lambda X.\, X \to X$?
- Why doesn't an arrow type $\mathsf{Nat} \to \mathsf{Nat}$ have an arrow kind like $* \Rightarrow *$?

λ_ω

$\lambda_\rightarrow$ with Type-Level Functions and Kinding

Syntax

t	$::=$		*terms:*
		x	*variable*
		$\lambda x{:}T.\,t$	*abstraction*
		$t\ t$	*application*
v	$::=$		*values:*
		$\lambda x{:}T.\,t$	*abstraction value*
T	$::=$		*types:*
		X	*type variable*
		$\lambda X{::}K.\,T$	*operator abstraction*
		$T\ T$	*operator application*
		$T \to T$	*type of functions*
Γ	$::=$		*contexts:*
		$\varnothing$	*empty context*
		$\Gamma, x : T$	*term variable binding*
		$\Gamma, X :: K$	*type variable binding*
K	$::=$		*kinds:*
		$*$	*kind of proper types*
		$K \Rightarrow K$	*kind of operators*

Evaluation and Typing

Evaluation

$$\frac{t_1 \longrightarrow t_1'}{t_1\ t_2 \longrightarrow t_1'\ t_2}\ \text{E-App1} \qquad \frac{t_2 \longrightarrow t_2'}{v_1\ t_2 \longrightarrow v_1\ t_2'}\ \text{E-App2} \qquad \frac{}{(\lambda x{:}T_{11}.\,t_{12})\ v_2 \longrightarrow [x \mapsto v_2]t_{12}}\ \text{E-AppAbs}$$

Typing

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T}\ \text{T-Var} \qquad \frac{\Gamma \vdash T_1 :: * \qquad \Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1.\,t_2 : T_1 \to T_2}\ \text{T-Abs} \qquad \frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}}\ \text{T-App}$$

$$\frac{\Gamma \vdash t : S \qquad S \equiv T \qquad \Gamma \vdash T :: *}{\Gamma \vdash t : T}\ \text{T-Eq}$$

Kinding

$\Gamma \vdash T :: K$: "type T has kind K in context Γ"

$$\frac{X :: K \in \Gamma}{\Gamma \vdash X :: K} \text{ K-TVar} \qquad \frac{\Gamma, X :: K_1 \vdash T_2 :: K_2}{\Gamma \vdash \lambda X{::}K_1 . T_2 :: K_1 \Rightarrow K_2} \text{ K-Abs}$$

$$\frac{\Gamma \vdash T_1 :: K_{11} \Rightarrow K_{12} \qquad \Gamma \vdash T_2 :: K_{11}}{\Gamma \vdash T_1\ T_2 :: K_{12}} \text{ K-App} \qquad \frac{\Gamma \vdash T_1 :: * \qquad \Gamma \vdash T_2 :: *}{\Gamma \vdash T_1 \to T_2 :: *} \text{ K-Arrow}$$

Remark

Those rules are very similar to the typing rules of $\lambda_\to$.

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \text{ T-Var} \qquad \frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x{:}T_1 . t_2 : T_1 \to T_2} \text{ T-Abs} \qquad \frac{\Gamma \vdash t_1 : T_{11} \to T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\ t_2 : T_{12}} \text{ T-App}$$

At the kinding level, the arrow $\to$ is like a **type operator with two arguments**!
We can assign a kind $* \Rightarrow * \Rightarrow *$ to $(\to)$ and an arrow type can be thought of as an operator application $(\to)\ T_1\ T_2$.

Type Equivalence, Definitionally

$S \equiv T$: "types S and T are definitionally equivalent"

$$\frac{}{T \equiv T} \text{ Q-REFL} \qquad \frac{T \equiv S}{S \equiv T} \text{ Q-SYMM} \qquad \frac{S \equiv U \qquad U \equiv T}{S \equiv T} \text{ Q-TRANS} \qquad \frac{S_1 \equiv T_1 \qquad S_2 \equiv T_2}{S_1 \to S_2 \equiv T_1 \to T_2} \text{ Q-ARROW}$$

$$\frac{S_2 \equiv T_2}{\lambda X{::}K_1 .\, S_2 \equiv \lambda X{::}K_1 .\, T_2} \text{ Q-ABS} \qquad \frac{S_1 \equiv T_1 \qquad S_2 \equiv T_2}{S_1\ S_2 \equiv T_1\ T_2} \text{ Q-APP}$$

$$\frac{}{(\lambda X{::}K_{11} .\, T_{12})\ T_2 \equiv [X \mapsto T_2]T_{12}} \text{ Q-APPABS}$$

PROPOSITION

Let us denote the type-level reduction by $\Rrightarrow$.
Then two types S and T are equivalent iff there exists some U such that $S \Rrightarrow^* U$ and $T \Rrightarrow^* U$.

Type Equivalence, Computationally

$S \Rrightarrow T$: "type S parallelly reduces to type T"

$$\frac{}{T \Rrightarrow T}\ \text{QR-Refl} \qquad \frac{S_1 \Rrightarrow T_1 \qquad S_2 \Rrightarrow T_2}{S_1 \to S_2 \Rrightarrow T_1 \to T_2}\ \text{QR-Arrow} \qquad \frac{S_2 \Rrightarrow T_2}{\lambda X{::}K_1.\, S_2 \Rrightarrow \lambda X{::}K_1.\, T_2}\ \text{QR-Abs}$$

$$\frac{S_1 \Rrightarrow T_1 \qquad S_2 \Rrightarrow T_2}{S_1\ S_2 \Rrightarrow T_1\ T_2}\ \text{QR-App} \qquad \frac{S_{12} \Rrightarrow T_{12} \qquad S_2 \Rrightarrow T_2}{(\lambda X{::}K_{11}.\, S_{12})\ S_2 \Rrightarrow [X \mapsto T_2]T_{12}}\ \text{QR-AppAbs}$$

Example

Let $S \stackrel{\text{def}}{=} \mathtt{Id}\,\mathtt{Nat} \to \mathtt{Bool}$ and $T \stackrel{\text{def}}{=} \mathtt{Id}\,(\mathtt{Nat} \to \mathtt{Bool})$. Then

$$S = ((\lambda X{::}{*}.\, X)\ \mathtt{Nat}) \to \mathtt{Bool} \Rrightarrow \mathtt{Nat} \to \mathtt{Bool},$$

$$T = (\lambda X{::}{*}.\, X)\ (\mathtt{Nat} \to \mathtt{Bool}) \Rrightarrow \mathtt{Nat} \to \mathtt{Bool},$$

by rule (QR-AppAbs).

The Essence of λ

The Essence of λ: Characterization

Principle

Types characterize **terms**. **Kinds** characterize **types**.

Question

Can we have more than **three** levels of expressions?

Aside (Pure Type Systems, Part I)

Let S be a set of **sorts**, e.g., $S = \{*, \square\}$ where

- $*$ represents the sort of **all (proper) types** and
- $\square$ represents the sort of **all kinds**.

Let M be a set of **axioms**, e.g., $M = \{(\varnothing \vdash * : \square)\}$, meaning "$*$ is a kind for (proper) types."

One can definitely add more sorts to S and more axioms to M accordingly!

The Essence of λ: Abstraction

Principle

- In $\lambda_{\rightarrow}$, we use $\lambda x{:}T.\ t$ to abstract **terms** out of **terms**.
- In λ_{ω}, we use $\lambda X{::}K.\ T$ to abstract **types** out of **types**.

Aside (Pure Type Systems, Part II)

Let S be a set of **sorts**, e.g., $S = \{*, \square\}$. Let M be a set of **axioms**, e.g., $M = \{(\varnothing \vdash * : \square)\}$.

Let $R \subseteq S \times S$ be a set of **rules**: for each $(s_1, s_2) \in R$, we have

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma \vdash B : s_2}{\Gamma \vdash A \rightsquigarrow^{s_1}_{s_2} B : s_2}\ \textsc{Arrow} \qquad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash A \rightsquigarrow^{s_1}_{s_2} B : s_2}{\Gamma \vdash \lambda x{:}A.\ b : A \rightsquigarrow^{s_1}_{s_2} B}\ \textsc{Abs}$$

$$\frac{\Gamma \vdash F : A \rightsquigarrow^{s_1}_{s_2} B \qquad \Gamma \vdash a : A}{\Gamma \vdash F\,a : B}\ \textsc{App}$$

Let $R \subseteq S \times S$ be a set of **rules**: for each $(s_1, s_2) \in R$, we have

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma \vdash B : s_2}{\Gamma \vdash A \rightsquigarrow^{s_1}_{s_2} B : s_2}\ \textsc{Arrow} \qquad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash A \rightsquigarrow^{s_1}_{s_2} B : s_2}{\Gamma \vdash \lambda x{:}A.\, b : A \rightsquigarrow^{s_1}_{s_2} B}\ \textsc{Abs}$$

$$\frac{\Gamma \vdash F : A \rightsquigarrow^{s_1}_{s_2} B \qquad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B}\ \textsc{App}$$

$\lambda_\to$: Abstracting Terms out of Terms

Let $R \stackrel{\text{def}}{=} \{(*, *)\}$. Then $\rightsquigarrow^*_*$ represents arrow types $\to$.

$$\frac{\Gamma \vdash T_1 : * \qquad \Gamma \vdash T_2 : *}{\Gamma \vdash T_1 \rightsquigarrow^*_* T_2 : *}$$ means "if T_1, T_2 are types, then $T_1 \to T_2$ is a type"

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2 \qquad \Gamma \vdash T_1 \rightsquigarrow^*_* T_2 : *}{\Gamma \vdash \lambda x{:}T_1 .\, t_2 : T_1 \rightsquigarrow^*_* T_2}$$ means the typing rule (T-Abs)

$$\frac{\Gamma \vdash t_1 : T_{11} \rightsquigarrow^*_* T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\, t_2 : T_{12}}$$ means the typing rule (T-App)

Let $R \subseteq S \times S$ be a set of **rules**: for each $(s_1, s_2) \in R$, we have

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma \vdash B : s_2}{\Gamma \vdash A \rightsquigarrow^{s_1}_{s_2} B : s_2}\ \textsc{Arrow} \qquad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash A \rightsquigarrow^{s_1}_{s_2} B : s_2}{\Gamma \vdash \lambda x{:}A.\, b : A \rightsquigarrow^{s_1}_{s_2} B}\ \textsc{Abs}$$

$$\frac{\Gamma \vdash F : A \rightsquigarrow^{s_1}_{s_2} B \qquad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B}\ \textsc{App}$$

λ_ω: Abstracting Types out of Types

Let $R \stackrel{\text{def}}{=} \{(*, *), (\Box, \Box)\}$. Then $\rightsquigarrow^*_*$ represents arrow types $\to$ and $\rightsquigarrow^\Box_\Box$ represents arrow kinds $\Rightarrow$.

$$\frac{\Gamma \vdash K_1 : \Box \qquad \Gamma \vdash K_2 : \Box}{\Gamma \vdash K_1 \rightsquigarrow^\Box_\Box K_2 : \Box}$$ means "if K_1, K_2 are kinds, then $K_1 \Rightarrow K_2$ is a kind"

$$\frac{\Gamma, X : K_1 \vdash T_2 : K_2 \qquad \Gamma \vdash K_1 \rightsquigarrow^\Box_\Box K_2 : \Box}{\Gamma \vdash \lambda X{:}K_1.\, T_2 : K_1 \rightsquigarrow^\Box_\Box K_2}$$ means the typing rule (K-Abs)

$$\frac{\Gamma \vdash T_1 : K_{11} \rightsquigarrow^\Box_\Box K_{12} \qquad \Gamma \vdash T_2 : K_{11}}{\Gamma \vdash T_1\, T_2 : K_{12}}$$ means the typing rule (K-App)

The Essence of λ: Abstraction

Principle

In System F, we use $\lambda X.\ t$ to abstract **terms** out of **types**.

Observation

We can think of $\lambda X.\ t$ as $\lambda X{::}{*}.\ t$, i.e., a type abstraction should be applied to a proper type.
The type of $\lambda X{::}{*}.\ t$ then has the form $\forall X{::}{*}.\ T$—**not an arrow!**
$\forall X{::}{*}.\ T$ can be thought of as a **dependent arrow** $(X{::}{*}) \rightrightarrows T$: the domain is a **kind** and the range is a **type**.
In next chapter, we will see a generalized form $\forall X{::}K.\ T$, or as a dependent arrow $(X{::}K) \rightrightarrows T$.

Aside (Pure Type Systems, Part III)

Let $R \subseteq S \times S$ be a set of **rules**: for each $(s_1, s_2) \in R$, we have

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma \vdash B : s_2}{\Gamma \vdash A \rightsquigarrow^{s_1}_{s_2} B : s_2}\ \textsc{Arrow} \qquad \text{becomes} \qquad \frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash (x{:}A) \rightsquigarrow^{s_1}_{s_2} B : s_2}\ \textsc{Arrow}^{D}$$

Then $(X : {*}) \rightsquigarrow^{\square}_{*} T$ represents $\forall X{::}{*}.\ T$!

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash A \rightsquigarrow^{s_1}_{s_2} B : s_2}{\Gamma \vdash \lambda x{:}A.\, b : A \rightsquigarrow^{s_1}_{s_2} B}\textsc{Abs} \quad \text{becomes} \quad \frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash (x{:}A) \rightsquigarrow^{s_1}_{s_2} B : s_2}{\Gamma \vdash \lambda x{:}A.\, b : (x{:}A) \rightsquigarrow^{s_1}_{s_2} B}\textsc{Abs}^{\mathsf{D}}$$

$$\frac{\Gamma \vdash F : A \rightsquigarrow^{s_1}_{s_2} B \qquad \Gamma \vdash a : A}{\Gamma \vdash F\, a : B}\textsc{App} \quad \text{becomes} \quad \frac{\Gamma \vdash F : (x{:}A) \rightsquigarrow^{s_1}_{s_2} B \qquad \Gamma \vdash a : A}{\Gamma \vdash F\, a : [x \mapsto a]B}\textsc{App}^{\mathsf{D}}$$

System F: Abstracting Terms out of Types

Let $R \stackrel{\text{def}}{=} \{(*, *), (\square, *)\}$. Then $\rightsquigarrow^{*}_{*}$ represents arrow types $\to$ and $\rightsquigarrow^{\square}_{*}$ represents universal types $\forall$.

$$\frac{\Gamma \vdash K_1 : \square \qquad \Gamma, X : K_1 \vdash T_2 : *}{\Gamma \vdash (X : K_1) \rightsquigarrow^{\square}_{*} T_2 : *}$$

means "if K_1 is a kind and T_2 is a type, then $\forall X{::}K_1 .\, T_2$ is a type"

$$\frac{\Gamma, X : K_1 \vdash t_2 : T_2 \qquad \Gamma \vdash (X{:}K_1) \rightsquigarrow^{\square}_{*} T_2 : *}{\Gamma \vdash \lambda X{:}K_1 .\, t_2 : (X{:}K_1) \rightsquigarrow^{\square}_{*} T_2}$$

means the typing rule (T-TAbs)

$$\frac{\Gamma \vdash t_1 : (X{:}K_{11}) \rightsquigarrow^{\square}_{*} T_{12} \qquad \Gamma \vdash T_2 : K_{11}}{\Gamma \vdash t_1\, [T_2] : [X \mapsto T_2]T_{12}}$$

means the typing rule (T-TApp)

The Essence of λ: Abstraction

Aside (Pure Type Systems, Part IV)

$\lambda_\rightarrow$	abstract **terms** out of **terms**	$\{(*, *)\}$
F	abstract **terms** out of **types**	$\{(*, *), (\square, *)\}$
λ_ω	abstract **types** out of **types**	$\{(*, *), (\square, \square)\}$
F_ω	$F + \lambda_\omega$ **(next chapter)**	$\{(*, *), (\square, *), (\square, \square)\}$

There are eight variants, each of which is $(*, *)$ plus a subset of $\{(\square, *), (\square, \square), (*, \square)\}$!

Question

What does the rule $(*, \square)$ mean? "Abstracting **types** out of **terms** by $\lambda x{:}T.\,T$?"

$$\frac{\Gamma \vdash T_1 : * \qquad \Gamma, x : T_1 \vdash K_2 : \square}{\Gamma \vdash (x{:}T_1) \rightsquigarrow^*_\square K_2 : \square}\ \text{ArrowD}$$

$$\frac{\Gamma, x : T_1 \vdash T_2 : K_2 \qquad \Gamma \vdash (x{:}T_1) \rightsquigarrow^*_\square K_2 : \square}{\Gamma \vdash \lambda x{:}T_1.\,T_2 : (x{:}T_1) \rightsquigarrow^*_\square K_2}\ \text{AbsD}$$

$$\frac{\Gamma \vdash T_1 : (x{:}T_{11}) \rightsquigarrow^*_\square K_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash T_1\ [t_2] : [x \mapsto t_2]K_{12}}\ \text{AppD}$$

$$K ::= * \mid (x{:}T) \rightsquigarrow^{*}_{\square} K$$
$$T ::= \mathsf{Nat} \mid \lambda x{:}T.\, T \mid T\,[t] \mid (x{:}T) \rightsquigarrow^{*}_{*} T$$
$$t ::= \mathtt{zero} \mid \mathtt{succ}(t) \mid x \mid \lambda x{:}T.\, t \mid t\, t$$

$$\frac{\Gamma, x : T_1 \vdash T_2 :: K_2 \qquad \Gamma \vdash T_1 :: *}{\Gamma \vdash \lambda x{:}T_1.\, T_2 :: (x{:}T_1) \rightsquigarrow^{*}_{\square} K_2} \text{ K-VAbs}$$

$$\frac{\Gamma \vdash T_1 :: (x{:}T_{11}) \rightsquigarrow^{*}_{\square} K_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash T_1\,[t_2] :: [x \mapsto t_2]K_{12}} \text{ K-VApp}$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2 \qquad \Gamma \vdash T_1 :: *}{\Gamma \vdash \lambda x{:}T_1.\, t_2 : (x{:}T_1) \rightsquigarrow^{*}_{*} T_2} \text{ T-Abs}$$

$$\frac{\Gamma \vdash t_1 : (x{:}T_{11}) \rightsquigarrow^{*}_{*} T_{12} \qquad \Gamma \vdash t_2 : T_{11}}{\Gamma \vdash t_1\, t_2 : [x \mapsto t_2]T_{12}} \text{ T-App}$$

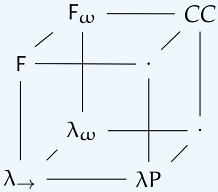
Example (Dependent Types)

Consider the type $\mathtt{NatList}$ and its two introduction terms $\mathtt{nil}$ and $\mathtt{cons}$.

$$\mathtt{NatList} :: \mathsf{Nat} \rightsquigarrow^{*}_{\square} *$$
$$\mathtt{nil} : \mathtt{NatList}\,[\mathtt{zero}]$$
$$\mathtt{cons} : (n{:}\mathsf{Nat}) \rightsquigarrow^{*}_{*} \mathsf{Nat} \rightsquigarrow^{*}_{*} \mathtt{NatList}\,[n] \rightsquigarrow^{*}_{*} \mathtt{NatList}\,[\mathtt{succ}(n)]$$

The Lambda Cube

Aside (Pure Type Systems, Part V)



$\lambda_\to$	simply-typed lambda-calculus	$\{(*, *)\}$
F	parametric polymorphism	$\{(*, *), (\Box, *)\}$
λ_ω	type operators	$\{(*, *), (\Box, \Box)\}$
λP	dependent types	$\{(*, *), (*, \Box)\}$
F_ω	higher-order polymorphism	$\{(*, *), (\Box, *), (\Box, \Box)\}$
CC	calculus of constructions	$\{(*, *), (\Box, *), (\Box, \Box), (*, \Box)\}$