Types, Frames, and
Applicative Structures

Def.: Types

Let $\mathcal{T}$ be the least set s.t:

$$\mathsf{o} \in \mathcal{T}$$

$$\iota \in \mathcal{T}$$

$$\forall \alpha, \beta \in \mathcal{T} : (\alpha\beta) \in \mathcal{T}$$

Def.: Types

Let $\mathcal{T}$ be the least set s.t:

$$\mathsf{o} \in \mathcal{T}$$

$$\iota \in \mathcal{T}$$

$$\forall \alpha, \beta \in \mathcal{T} : (\alpha\beta) \in \mathcal{T}$$

We say that $\alpha \in \mathcal{T}$ is a simple type (or type).
$(\alpha\beta)$ is called a function type.

Def.: Types

Let $\mathcal{T}$ be the least set s.t:

$$\mathsf{o} \in \mathcal{T}$$

$$\iota \in \mathcal{T}$$

$$\forall \alpha, \beta \in \mathcal{T} : (\alpha\beta) \in \mathcal{T}$$

We say that $\alpha \in \mathcal{T}$ is a simple type (or type).
$(\alpha\beta)$ is called a function type.

- The set $\mathcal{T}$ is defined inductively.
- The set $\mathcal{T}$ is "freely generated".

Ex.: Freely Generated

Consider the set $\mathbb{N} = \{0, 1, 2 \ldots\}$.

- $0 \in \mathbb{N}$

Ex.: Freely Generated

Consider the set $\mathbb{N} = \{0, 1, 2 \ldots\}$.

- $0 \in \mathbb{N}$
- $\forall \mathsf{n} \in \mathbb{N} : \mathsf{s}(\mathsf{n}) \in \mathbb{N}$.

Ex.: Freely Generated

Consider the set $\mathbb{N} = \{0, 1, 2\ldots\}$.

- $0 \in \mathbb{N}$
- $\forall n \in \mathbb{N} : s(n) \in \mathbb{N}$.
- $\forall n : 0 \neq s(n)$.

Ex.: Freely Generated

Consider the set $\mathbb{N} = \{0, 1, 2 \ldots\}$.

- $0 \in \mathbb{N}$
- $\forall \mathsf{n} \in \mathbb{N} : \mathsf{s}(\mathsf{n}) \in \mathbb{N}$.
- $\forall \mathsf{n} : 0 \neq \mathsf{s}(\mathsf{n})$.
- $\forall \mathsf{m}, \mathsf{n} : \mathsf{s}(\mathsf{m}) = \mathsf{s}(\mathsf{n}) \Rightarrow \mathsf{m} = \mathsf{n}$.

Ex.: Freely Generated

Consider the set $\mathbb{N} = \{0, 1, 2...\}$.

- $0 \in \mathbb{N}$
- $\forall n \in \mathbb{N} : s(n) \in \mathbb{N}$.
- $\forall n : 0 \neq s(n)$.
- $\forall m, n : s(m) = s(n) \Rightarrow m = n$.

The set $\mathbb{N}$ is "freely generated".

Ex.: Freely Generated

Consider the set $\mathbb{N} = \{0, 1, 2...\}$.

- $0 \in \mathbb{N}$
- $\forall n \in \mathbb{N} : s(n) \in \mathbb{N}$.
- $\forall n : 0 \neq s(n)$.
- $\forall m, n : s(m) = s(n) \Rightarrow m = n$.

The set $\mathbb{N}$ is "freely generated".

Contrast $\mathbb{N}$ to $\mathbb{Z} = \{..., -1, 0, 1, ...\}$.
Note that $\mathbb{Z}$ contains 0 and is closed under successor, but is not the least such set.

Ex.: Freely Generated

The set $\mathcal{T}$ is "freely generated":

- $\mathrm{o} \neq \iota$

Ex.: Freely Generated

The set $\mathcal{T}$ is "freely generated":

- $o \neq \iota$
- $o \neq (\alpha\beta)$

Ex.: Freely Generated

The set $\mathcal{T}$ is "freely generated":

- $\mathsf{o} \neq \iota$
- $\mathsf{o} \neq (\alpha\beta)$
- $\iota \neq (\alpha\beta)$

Ex.: Freely Generated

The set $\mathcal{T}$ is "freely generated":

- $o \neq \iota$
- $o \neq (\alpha\beta)$
- $\iota \neq (\alpha\beta)$
- $(\alpha\beta) = (\gamma\delta) \Rightarrow \alpha = \gamma \wedge \beta = \delta$

Ex.: Types

- $(o\iota) \in \mathcal{T}$

Ex.: Types

- $(o\iota) \in \mathcal{T}$
- $(o(o\iota)) \in \mathcal{T}$

Ex.: Types

- $(\mathsf{o}\iota) \in \mathcal{T}$
- $(\mathsf{o}(\mathsf{o}\iota)) \in \mathcal{T}$
- $(\iota\iota) \in \mathcal{T}$

Ex.: Types

- $(o\iota) \in \mathcal{T}$
- $(o(o\iota)) \in \mathcal{T}$
- $(\iota\iota) \in \mathcal{T}$
- $((o\iota)\iota) \in \mathcal{T}$

Ex.: Types

- $(o\iota) \in \mathcal{T}$
- $(o(o\iota)) \in \mathcal{T}$
- $(\iota\iota) \in \mathcal{T}$
- $((o\iota)\iota) \in \mathcal{T}$

Is $(o\iota\iota)$ also a type?

Ex.: Types

- $(o\iota) \in \mathcal{T}$
- $(o(o\iota)) \in \mathcal{T}$
- $(\iota\iota) \in \mathcal{T}$
- $((o\iota)\iota) \in \mathcal{T}$

Is $(o\iota\iota)$ also a type? – no

Ex.: Types

- $(\mathsf{o}\iota) \in \mathcal{T}$
- $(\mathsf{o}(\mathsf{o}\iota)) \in \mathcal{T}$
- $(\iota\iota) \in \mathcal{T}$
- $((\mathsf{o}\iota)\iota) \in \mathcal{T}$

Is $(\mathsf{o}\iota\iota)$ also a type? – no

But we can and will consider it shorthand by replacing missing parenthesis, associating to the left: $(\mathsf{o}\iota\iota) = ((\mathsf{o}\iota)\iota) \neq (\mathsf{o}(\iota\iota))$.

Def.: Functions

Let A, B be sets.

Def.: Functions

Let A, B be sets.

$f : B \longrightarrow A$: a function from B to A.

Def.: Functions

Let A, B be sets.

$f : B \longrightarrow A$: a function from B to A.

A^B: set of functions from B to A.

Def.: Functions

Let A, B be sets.

$f : B \longrightarrow A$: a function from B to A.

A^B: set of functions from B to A.

Assume (only for the moment) that A, B are finite.

Def.: Functions

Let A, B be sets.

$f : B \longrightarrow A$: a function from B to A.

A^B: set of functions from B to A.

Assume (only for the moment) that A, B are finite.

Let $|A| = m, |B| = n$. Then $|A^B| = m^n = |A|^{|B|}$.

Def.: Functions

Let A, B be sets.

$f : B \longrightarrow A$: a function from B to A.

A^B: set of functions from B to A.

Assume (only for the moment) that A, B are finite.

Let $|A| = m, |B| = n$. Then $|A^B| = m^n = |A|^{|B|}$.

Example:

Def.: Functions

Let A, B be sets.

$f : B \longrightarrow A$: a function from B to A.

A^B: set of functions from B to A.

Assume (only for the moment) that A, B are finite.

Let $|A| = m, |B| = n$. Then $|A^B| = m^n = |A|^{|B|}$.

Example:

- $f : \{0, 1, 2\} \longrightarrow \{0, 1\}$

Def.: Functions

Let A, B be sets.

$f : B \longrightarrow A$: a function from B to A.

A^B: set of functions from B to A.

Assume (only for the moment) that A, B are finite.

Let $|A| = m, |B| = n$. Then $|A^B| = m^n = |A|^{|B|}$.

Example:

- $f : \{0, 1, 2\} \longrightarrow \{0, 1\}$
- $f(0), f(1), f(2) \in \{0, 1\}$

Def.: Functions

Let A, B be sets.

$f : B \longrightarrow A$: a function from B to A.

A^B: set of functions from B to A.

Assume (only for the moment) that A, B are finite.

Let $|A| = m, |B| = n$. Then $|A^B| = m^n = |A|^{|B|}$.

Example:

- $f : \{0, 1, 2\} \longrightarrow \{0, 1\}$
- $f(0), f(1), f(2) \in \{0, 1\}$
- $A = \{0, 1\}, B = \{0, 1, 2\}$

Def.: Functions

Let A, B be sets.

$f : B \longrightarrow A$: a function from B to A.

A^B: set of functions from B to A.

Assume (only for the moment) that A, B are finite.

Let $|A| = m, |B| = n$. Then $|A^B| = m^n = |A|^{|B|}$.

Example:

- $f : \{0, 1, 2\} \longrightarrow \{0, 1\}$
- $f(0), f(1), f(2) \in \{0, 1\}$
- $A = \{0, 1\}, B = \{0, 1, 2\}$
- $|A^B| = 2 \cdot 2 \cdot 2 = 2^3 = 8$

Ex.: Sets of Functions

Let $F = \{f : B \rightarrow A | \forall x, y \in B : x \leq y \Rightarrow f(x) \leq f(y)\} \subseteq A^B$.

Ex.: Sets of Functions

Let $F = \{f : B \to A | \forall x, y \in B : x \leq y \Rightarrow f(x) \leq f(y)\} \subseteq A^B$.

$|F| = ?$

Ex.: Sets of Functions

Let $F = \{f : B \to A | \forall x, y \in B : x \leq y \Rightarrow f(x) \leq f(y)\} \subseteq A^B$.

$|F| = ?$

A^B	$f(0)$	$f(1)$	$f(2)$
$K_0 \in F$	0	0	0
$\in F$	0	0	1
$\notin F$	0	1	0
$\in F$	0	1	1
$g \notin F$	1	0	0
$\notin F$	1	0	1
$\notin F$	1	1	0
$K_1 \in F$	1	1	1

Consider:

$g : x = 0, y = 1, x \leq y$, but $f(x) \geq f(y) \Rightarrow g \notin F$.

Ex.: Sets of Functions

Let $F = \{f : B \to A | \forall x, y \in B : x \leq y \Rightarrow f(x) \leq f(y)\} \subseteq A^B$.

$|F| =?$

A^B	$f(0)$	$f(1)$	$f(2)$
$K_0 \in F$	0	0	0
$\in F$	0	0	1
$\notin F$	0	1	0
$\in F$	0	1	1
$g \notin F$	1	0	0
$\notin F$	1	0	1
$\notin F$	1	1	0
$K_1 \in F$	1	1	1

Consider:
$g : x = 0, y = 1, x \leq y$, but $f(x) \geq f(y) \Rightarrow g \notin F$.

$|F| = 4$

Ex.: Sets of Labelled Functions

$$C = \{\mathsf{red}, \mathsf{blue}, \mathsf{green}\}$$

Ex.: Sets of Labelled Functions

$$C = \{\mathsf{red}, \mathsf{blue}, \mathsf{green}\}$$

$$F_C = \{\langle c, f\rangle | c \in C, f \in F\}$$

Ex.: Sets of Labelled Functions

$$C = \{\text{red}, \text{blue}, \text{green}\}$$

$$F_C = \{\langle c, f\rangle | c \in C, f \in F\}$$

$$|F_C| = 3 \cdot 4 = 12$$

Def.: Frames

A frame is a family $(\mathrm{D}_\alpha)_{\alpha \in \mathcal{T}}$ of nonempty sets s.t:

Def.: Frames

A frame is a family $(\mathsf{D}_\alpha)_{\alpha \in \mathcal{T}}$ of nonempty sets s.t:

$$\forall \alpha, \beta \in \mathcal{T} : \mathsf{D}_{\alpha\beta} \subseteq \mathsf{D}_\alpha^{\mathsf{D}_\beta}$$

Def.: Frames

A frame is a family $(\mathsf{D}_\alpha)_{\alpha \in \mathcal{T}}$ of nonempty sets s.t:

$$\forall \alpha, \beta \in \mathcal{T} : \mathsf{D}_{\alpha\beta} \subseteq \mathsf{D}_\alpha^{\mathsf{D}_\beta}$$

A Frame is called standard if

Def.: Frames

A frame is a family $(\mathsf{D}_\alpha)_{\alpha \in \mathcal{T}}$ of nonempty sets s.t:

$$\forall \alpha, \beta \in \mathcal{T} : \mathsf{D}_{\alpha\beta} \subseteq \mathsf{D}_\alpha^{\mathsf{D}_\beta}$$

A Frame is called standard if

$$\mathsf{D}_{\alpha\beta} = \mathsf{D}_\alpha^{\mathsf{D}_\beta} \quad \forall \alpha, \beta \in \mathcal{T}$$

Ex.: Frames

$D_o = \{\perp, \top\}$

Ex.: Frames

$\mathsf{D}_o = \{\bot, \top\}$

$\mathsf{D}_\iota = \{1\}$

Ex.: Frames

$$\mathsf{D}_o = \{\bot, \top\}$$

$$\mathsf{D}_\iota = \{1\}$$

$$\mathsf{D}_{\alpha\beta} = \mathsf{D}_\alpha^{\mathsf{D}_\beta}$$

Ex.: Frames

$\mathsf{D}_o = \{\bot, \top\}$

$\mathsf{D}_\iota = \{1\}$

$\mathsf{D}_{\alpha\beta} = \mathsf{D}_\alpha^{\mathsf{D}_\beta}$

D: the standard frame with $\mathsf{D}_o = \{\bot, \top\}, \mathsf{D}_i = \{1\}$

Ex.: Frames (Contd.)

Consider the set $\mathsf{D}_{\mathsf{o}(\iota\iota)((\mathsf{o}(\iota\mathsf{o})))}$. Is the set empty?

Ex.: Frames (Contd.)

Consider the set $\mathsf{D}_{\mathsf{o}(\iota\iota)((\mathsf{o}(\iota\mathsf{o})))}$. Is the set empty? — no!

Ex.: Frames (Contd.)

Consider the set $\mathsf{D}_{o(\iota\iota)((o(\iota o)))}$. Is the set empty? — no!

Claim: $\forall \alpha \in \mathcal{T} : \mathsf{D}_\alpha \neq \emptyset$.

Ex.: Frames (Contd.)

Consider the set $\mathsf{D}_{\mathsf{o}(\iota\iota)((\mathsf{o}(\iota\mathsf{o})))}$. Is the set empty? — no!

Claim: $\forall \alpha \in \mathcal{T} : \mathsf{D}_\alpha \neq \emptyset$.
Proof: induction on type.

Ex.: Frames (Contd.)

Consider the set $\mathsf{D}_{\mathsf{o}(\iota\iota)((\mathsf{o}(\iota\mathsf{o})))}$. Is the set empty? — no!

Claim: $\forall \alpha \in \mathcal{T} : \mathsf{D}_\alpha \neq \emptyset$.
Proof: induction on type.

- Base: $\mathsf{D}_\mathsf{o} = \{\bot, \top\} \neq \emptyset, \mathsf{D}_\mathsf{i} = \{1\} \neq \emptyset$.

Ex.: Frames (Contd.)

Consider the set $D_{o(\iota\iota)((o(\iota o)))}$. Is the set empty? — no!

Claim: $\forall \alpha \in \mathcal{T} : D_\alpha \neq \emptyset$.
Proof: induction on type.

- Base: $D_o = \{\bot, \top\} \neq \emptyset, D_i = \{1\} \neq \emptyset$.
- Step: Assume $D_\alpha \neq \emptyset \wedge D_\beta \neq \emptyset$. Want to show: $D_{\alpha\beta} \neq \emptyset$.

Ex.: Frames (Contd.)

Consider the set $\mathsf{D}_{\mathsf{o}(\iota\iota)((\mathsf{o}(\iota\mathsf{o})))}$. Is the set empty? — no!

Claim: $\forall\alpha \in \mathcal{T} : \mathsf{D}_\alpha \neq \emptyset$.
Proof: induction on type.

- Base: $\mathsf{D}_\mathsf{o} = \{\bot, \top\} \neq \emptyset, \mathsf{D}_\mathsf{i} = \{1\} \neq \emptyset$.
- Step: Assume $\mathsf{D}_\alpha \neq \emptyset \wedge \mathsf{D}_\beta \neq \emptyset$. Want to show: $\mathsf{D}_{\alpha\beta} \neq \emptyset$.
  Since $\mathsf{D}_\alpha \neq \emptyset \Rightarrow \exists \mathsf{a} \in \mathsf{D}_\alpha$,

Ex.: Frames (Contd.)

Consider the set $D_{o(\iota\iota)((o(\iota o)))}$. Is the set empty? — no!

Claim: $\forall \alpha \in \mathcal{T} : D_\alpha \neq \emptyset$.
Proof: induction on type.

- Base: $D_o = \{\bot, \top\} \neq \emptyset, D_i = \{1\} \neq \emptyset$.
- Step: Assume $D_\alpha \neq \emptyset \wedge D_\beta \neq \emptyset$. Want to show: $D_{\alpha\beta} \neq \emptyset$.
  Since $D_\alpha \neq \emptyset \Rightarrow \exists a \in D_\alpha$, hence $K_a \in D_{\alpha\beta}$.

Ex.: Frames (Contd.)

Consider the set $D_{o(\iota\iota)((o(\iota o)))}$. Is the set empty? — no!

Claim: $\forall\alpha \in \mathcal{T} : D_\alpha \neq \emptyset$.
Proof: induction on type.

- Base: $D_o = \{\bot, \top\} \neq \emptyset, D_i = \{1\} \neq \emptyset$.
- Step: Assume $D_\alpha \neq \emptyset \wedge D_\beta \neq \emptyset$. Want to show: $D_{\alpha\beta} \neq \emptyset$.
  Since $D_\alpha \neq \emptyset \Rightarrow \exists a \in D_\alpha$, hence $K_a \in D_{\alpha\beta}$.

  (Here K_a is the constant function which always returns a. We will often use this notation for constant functions.)

Def.: Typed Applicative Structure

A (typed) applicative structure is a tupel

Def.: Typed Applicative Structure

A (typed) applicative structure is a tupel

$$\langle \mathcal{D}, @ \rangle$$

where

Def.: Typed Applicative Structure

A (typed) applicative structure is a tupel

$$\langle \mathrm{D}, @ \rangle$$

where

- $\mathrm{D} := (\mathrm{D}_\alpha)_{\alpha \in \mathcal{T}}$ is a family of nonempty sets

Def.: Typed Applicative Structure

A (typed) applicative structure is a tupel

$$\langle \mathsf{D}, @\rangle$$

where

- $\mathsf{D} := (\mathsf{D}_\alpha)_{\alpha\in\mathcal{T}}$ is a family of nonempty sets
- $@ := (@^{\alpha\beta} : \mathsf{D}_{\alpha\beta} \times \mathsf{D}_\beta \longrightarrow \mathsf{D}_\alpha)_{\alpha,\beta\in\mathcal{T}}$

Def.: Typed Applicative Structure

A (typed) applicative structure is a tupel

$$\langle \mathsf{D}, @ \rangle$$

where

- $\mathsf{D} := (\mathsf{D}_\alpha)_{\alpha \in \mathcal{T}}$ is a family of nonempty sets
- $@ := (@^{\alpha\beta} : \mathsf{D}_{\alpha\beta} \times \mathsf{D}_\beta \longrightarrow \mathsf{D}_\alpha)_{\alpha,\beta \in \mathcal{T}}$

Usually we write $\mathsf{f}@\mathsf{b}$ for $@^{\alpha\beta}(\mathsf{f}, \mathsf{b})$ when $\mathsf{f} \in \mathsf{D}_{\alpha\beta} \wedge \mathsf{b} \in \mathsf{D}_\beta$

Rem.: Currying

The application operator $@$ in an applicative structure is an abstract version of function application.

Rem.: Currying

The application operator $@$ in an applicative structure is an abstract version of function application. It is no restriction to exclusively use a binary application operator, which corresponds to unary function application,

Rem.: Currying

The application operator $@$ in an applicative structure is an abstract version of function application. It is no restriction to exclusively use a binary application operator, which corresponds to unary function application, since we can define higher-arity application operators from the binary one by setting $\mathsf{f}@(\mathsf{a}^1,\ldots,\mathsf{a}^n) := (\ldots(\mathsf{f}@\mathsf{a}^1)\ldots@\mathsf{a}^n)$ ("Currying").

Interesting Properties

Let D be a frame.

Interesting Properties

Let D be a frame.

$$\forall f, g \in D_{\alpha\beta} \quad (\forall b \in D_\beta : f(b) = g(b)) \Rightarrow f = g.$$

Interesting Properties

Let D be a frame.

$$\forall \mathsf{f}, \mathsf{g} \in \mathrm{D}_{\alpha\beta} \quad (\forall \mathsf{b} \in \mathrm{D}_{\beta} : \mathsf{f}(\mathsf{b}) = \mathsf{g}(\mathsf{b})) \Rightarrow \mathsf{f} = \mathsf{g}.$$

Let $\langle \mathrm{D}, @ \rangle$ be an applicative structure. Consider the property:

Interesting Properties

Let D be a frame.

$$\forall f, g \in D_{\alpha\beta} \quad (\forall b \in D_\beta : f(b) = g(b)) \Rightarrow f = g.$$

Let $\langle D, @ \rangle$ be an applicative structure. Consider the property:

$$\forall f, g \in D_{\alpha\beta} \quad (\forall b \in D_\beta : f@b = g@b) \Rightarrow f = g.$$

Def.: Functional Applicative Structures

Given an applicative structure $\langle \mathcal{D}, @ \rangle$.

Def.: Functional Applicative Structures

Given an applicative structure $\langle \mathcal{D}, @ \rangle$. We say that $\langle \mathcal{D}, @ \rangle$ is functional if

Def.: Functional Applicative Structures

Given an applicative structure $\langle \mathsf{D}, @\rangle$. We say that $\langle \mathsf{D}, @\rangle$ is functional if

$$\forall \alpha, \beta \in \mathcal{T} : \forall \mathsf{f}, \mathsf{g} \in \mathsf{D}_{\alpha\beta}(\forall \mathsf{b} \in \mathsf{D}_{\beta} : \mathsf{f}@\mathsf{b} = \mathsf{g}@\mathsf{b}) \Rightarrow \mathsf{f} = \mathsf{g}$$

Def.: Full Applicative Structures

Given an applicative structure $\langle \mathcal{D}, @ \rangle$.

Def.: Full Applicative Structures

Given an applicative structure $\langle \mathrm{D}, @ \rangle$. We say that $\langle \mathrm{D}, @ \rangle$ is full if

Def.: Full Applicative Structures

Given an applicative structure $\langle \mathsf{D}, @ \rangle$. We say that $\langle \mathsf{D}, @ \rangle$ is full if

$$\forall \alpha, \beta \quad \forall \mathsf{h} : \mathsf{D}_\beta \longrightarrow \mathsf{D}_\alpha \quad \exists \mathsf{f} \in \mathsf{D}_{\alpha\beta} \forall \mathsf{b} \in \mathsf{D}_\beta : \mathsf{f}@\mathsf{b} = \mathsf{h}(\mathsf{b})$$

Def.: Standard Applicative Structures

An applicative structure $\mathcal{A} := \langle \mathrm{D}, @ \rangle$ is called standard if

Def.: Standard Applicative Structures

An applicative structure $\mathcal{A} := \langle \mathrm{D}, @ \rangle$ is called standard if it is a frame structure (i.e. $@$ is function application) where $\mathcal{D}$ is standard.

Def.: Standard Applicative Structures

An applicative structure $\mathcal{A} := \langle \mathrm{D}, @\rangle$ is called standard if it is a frame structure (i.e. $@$ is function application) where $\mathcal{D}$ is standard.

Note that the definitions of functional, full, and standard impose restrictions on the domains for function types only.

Rem.: Frames and Applicative Structures

It is easy to show that every frame is functional.

Rem.: Frames and Applicative Structures

It is easy to show that every frame is functional.

Furthermore, an applicative structure is standard iff it is a full frame.

Example: Full Functional Appl. Structure

Let $\mathrm{D}_\alpha = \{1\} \quad \forall\alpha$

Example: Full Functional Appl. Structure

Let $\mathsf{D}_\alpha = \{1\} \quad \forall \alpha$

Let $\mathsf{f}@\mathsf{b} = 1 \quad \forall \mathsf{f} \in \mathsf{D}_{\alpha\beta} \quad \forall \mathsf{b} \in \mathsf{D}_\beta$

Example: Full Functional Appl. Structure

Let $D_\alpha = \{1\} \quad \forall\alpha$

Let $f@b = 1 \quad \forall f \in D_{\alpha\beta} \quad \forall b \in D_\beta$

$\langle D, @\rangle$ is a full functional applicative structure, but it is not a frame.

Example: Full Functional Appl. Structure

Let $D_\alpha = \{1\} \quad \forall\alpha$

Let $f@b = 1 \quad \forall f \in D_{\alpha\beta} \quad \forall b \in D_\beta$

$\langle D, @\rangle$ is a full functional applicative structure, but it is not a frame.

$1 \in D_{oo}$ but $1 \notin D_o^{D_o} \Rightarrow D_{oo} \not\subseteq D_o^{D_o}$

Def.: Homomorphic Appl. Structures

Let $\langle \mathrm{D}^1, @^1 \rangle$ and $\langle \mathrm{D}^2, @^2 \rangle$ are applicative structures.

Def.: Homomorphic Appl. Structures

Let $\langle \mathrm{D}^1, @^1 \rangle$ and $\langle \mathrm{D}^2, @^2 \rangle$ are applicative structures. We say that κ is a homomorphism from $\langle \mathrm{D}^1, @^1 \rangle$ to $\langle \mathrm{D}^2, @^2 \rangle$ if

Def.: Homomorphic Appl. Structures

Let $\langle \mathsf{D}^1, @^1 \rangle$ and $\langle \mathsf{D}^2, @^2 \rangle$ are applicative structures. We say that κ is a homomorphism from $\langle \mathsf{D}^1, @^1 \rangle$ to $\langle \mathsf{D}^2, @^2 \rangle$ if

- $\kappa_\alpha : \mathsf{D}^1_\alpha \longrightarrow \mathsf{D}^2_\alpha \quad \forall \alpha \in \mathcal{T}$

Def.: Homomorphic Appl. Structures

Let $\langle \mathsf{D}^1, @^1 \rangle$ and $\langle \mathsf{D}^2, @^2 \rangle$ are applicative structures. We say that κ is a homomorphism from $\langle \mathsf{D}^1, @^1 \rangle$ to $\langle \mathsf{D}^2, @^2 \rangle$ if

- $\kappa_\alpha : \mathsf{D}^1_\alpha \longrightarrow \mathsf{D}^2_\alpha \quad \forall \alpha \in \mathcal{T}$
- $\forall \alpha, \beta \in \mathcal{T}, \quad \forall \mathsf{f} \in \mathsf{D}^1_{\alpha\beta}, \quad \forall \mathsf{b} \in \mathsf{D}^1_\beta$:

$$\kappa(\mathsf{f})@^2\kappa(\mathsf{b}) = \kappa(\mathsf{f}@^1\mathsf{b})$$

Def.: Isomorphic Appl. Structures

We say that $\langle \mathcal{D}^1, @^1 \rangle$ and $\langle \mathcal{D}^2, @^2 \rangle$ are isomorphic if $\exists i, j$ s.t:

Def.: Isomorphic Appl. Structures

We say that $\langle \mathcal{D}^1, @^1 \rangle$ and $\langle \mathcal{D}^2, @^2 \rangle$ are isomorphic if $\exists i, j$ s.t:

- i is a homomorphism from $\langle \mathcal{D}^1, @^1 \rangle$ to $\langle \mathcal{D}^2, @^2 \rangle$

Def.: Isomorphic Appl. Structures

We say that $\langle D^1, @^1\rangle$ and $\langle D^2, @^2\rangle$ are isomorphic if $\exists i, j$ s.t:

- i is a homomorphism from $\langle D^1, @^1\rangle$ to $\langle D^2, @^2\rangle$
- j is a homomorphism from $\langle D^2, @^2\rangle$ to $\langle D^1, @^1\rangle$

Def.: Isomorphic Appl. Structures

We say that $\langle D^1, @^1\rangle$ and $\langle D^2, @^2\rangle$ are isomorphic if $\exists i, j$ s.t:

- i is a homomorphism from $\langle D^1, @^1\rangle$ to $\langle D^2, @^2\rangle$
- j is a homomorphism from $\langle D^2, @^2\rangle$ to $\langle D^1, @^1\rangle$
- i and j are inverses (i.e $i(j(a^2)) = a^2$ and $j(i(a^1)) = a^1$).

Simply Typed λ-Calculus

Def.: Untyped λ-Calculus

Let $\Sigma = (\mathcal{V}, \mathcal{C})$ be a signature where

Def.: Untyped λ-Calculus

Let $\Sigma = (\mathcal{V}, \mathcal{C})$ be a signature where

- $\mathcal{V}$ — countably infinite set of variables

Def.: Untyped λ-Calculus

Let $\Sigma = (\mathcal{V}, \mathcal{C})$ be a signature where

- $\mathcal{V}$ — countably infinite set of variables
- $\mathcal{C}$ — possibly empty set of constants

Def.: Untyped λ-Calculus

Let $\Sigma = (\mathcal{V}, \mathcal{C})$ be a signature where

- $\mathcal{V}$ — countably infinite set of variables
- $\mathcal{C}$ — possibly empty set of constants

We define the set $\Lambda = \mathit{wff}_\Sigma(\Sigma)$ to be the smallest set s.t:

Def.: Untyped λ-Calculus

Let $\Sigma = (\mathcal{V}, \mathcal{C})$ be a signature where

- $\mathcal{V}$ — countably infinite set of variables
- $\mathcal{C}$ — possibly empty set of constants

We define the set $\Lambda = \mathit{wff}_\Sigma(\Sigma)$ to be the smallest set s.t:

- $\mathsf{x} \in \mathcal{V}$ then $\mathsf{x} \in \Lambda$

Def.: Untyped λ-Calculus

Let $\Sigma = (\mathcal{V}, \mathcal{C})$ be a signature where

- $\mathcal{V}$ — countably infinite set of variables
- $\mathcal{C}$ — possibly empty set of constants

We define the set $\Lambda = \mathit{wff}_\Sigma(\Sigma)$ to be the smallest set s.t:

- $\mathsf{x} \in \mathcal{V}$ then $\mathsf{x} \in \Lambda$
- $\mathsf{c} \in \mathcal{C}$ then $\mathsf{c} \in \Lambda$

Def.: Untyped λ-Calculus

Let $\Sigma = (\mathcal{V}, \mathcal{C})$ be a signature where

- $\mathcal{V}$ — countably infinite set of variables
- $\mathcal{C}$ — possibly empty set of constants

We define the set $\Lambda = \mathit{wff}_{\Sigma}(\Sigma)$ to be the smallest set s.t:

- $\mathsf{x} \in \mathcal{V}$ then $\mathsf{x} \in \Lambda$
- $\mathsf{c} \in \mathcal{C}$ then $\mathsf{c} \in \Lambda$
- $\mathsf{A} \in \Lambda$, $\mathsf{B} \in \Lambda$ then $(\mathsf{A}\,\mathsf{B}) \in \Lambda$

Def.: Untyped λ-Calculus

Let $\Sigma = (\mathcal{V}, \mathcal{C})$ be a signature where

- $\mathcal{V}$ — countably infinite set of variables
- $\mathcal{C}$ — possibly empty set of constants

We define the set $\Lambda = wff_\Sigma(\Sigma)$ to be the smallest set s.t:

- $\mathrm{x} \in \mathcal{V}$ then $\mathrm{x} \in \Lambda$
- $\mathrm{c} \in \mathcal{C}$ then $\mathrm{c} \in \Lambda$
- $\mathrm{A} \in \Lambda$, $\mathrm{B} \in \Lambda$ then $(\mathrm{A}\,\mathrm{B}) \in \Lambda$
- $\mathrm{x} \in \mathcal{V}$, $\mathrm{A} \in \Lambda$ then $(\lambda \mathrm{x}.\mathrm{A}) \in \Lambda$

UNIVERSITÄT DES SAARLANDES

Simply Typed λ-Calculus

Let $\Sigma = (\mathcal{V}^\alpha, \mathcal{C}^\alpha)$ be a signature where

Simply Typed λ-Calculus

Let $\Sigma = (\mathcal{V}^\alpha, \mathcal{C}^\alpha)$ be a signature where

- $\mathcal{V}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{V}_\alpha$ — countably infinite sets of variables

Simply Typed λ-Calculus

Let $\Sigma = (\mathcal{V}^\alpha, \mathcal{C}^\alpha)$ be a signature where

- $\mathcal{V}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{V}_\alpha$ — countably infinite sets of variables
- $\mathcal{C}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{C}_\alpha$ — possibly empty sets of constants

Simply Typed λ-Calculus

Let $\Sigma = (\mathcal{V}^\alpha, \mathcal{C}^\alpha)$ be a signature where

- $\mathcal{V}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{V}_\alpha$ — countably infinite sets of variables
- $\mathcal{C}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{C}_\alpha$ — possibly empty sets of constants

We define the set $\Lambda^\alpha = \mathit{wff}_\Sigma(\Sigma)_\alpha = \bigcup_{\alpha \in \mathcal{T}} \Lambda_\alpha$ to be the smallest set s.t:

Simply Typed λ-Calculus

Let $\Sigma = (\mathcal{V}^\alpha, \mathcal{C}^\alpha)$ be a signature where

- $\mathcal{V}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{V}_\alpha$ — countably infinite sets of variables
- $\mathcal{C}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{C}_\alpha$ — possibly empty sets of constants

We define the set $\Lambda^\alpha = \mathit{wff}_\Sigma(\Sigma)_\alpha = \bigcup_{\alpha \in \mathcal{T}} \Lambda_\alpha$ to be the smallest set s.t:

- $\mathsf{x}_\alpha \in \mathcal{V}_\alpha$ then $\mathsf{x}_\alpha \in \Lambda_\alpha$

Simply Typed λ-Calculus

Let $\Sigma = (\mathcal{V}^\alpha, \mathcal{C}^\alpha)$ be a signature where

- $\mathcal{V}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{V}_\alpha$ — countably infinite sets of variables
- $\mathcal{C}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{C}_\alpha$ — possibly empty sets of constants

We define the set $\Lambda^\alpha = wff_\Sigma(\Sigma)_\alpha = \bigcup_{\alpha \in \mathcal{T}} \Lambda_\alpha$ to be the smallest set s.t:

- $x_\alpha \in \mathcal{V}_\alpha$ then $x_\alpha \in \Lambda_\alpha$
- $c_\alpha \in \mathcal{C}_\alpha$ then $c_\alpha \in \Lambda_\alpha$

Simply Typed λ-Calculus

Let $\Sigma = (\mathcal{V}^\alpha, \mathcal{C}^\alpha)$ be a signature where

- $\mathcal{V}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{V}_\alpha$ — countably infinite sets of variables
- $\mathcal{C}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{C}_\alpha$ — possibly empty sets of constants

We define the set $\Lambda^\alpha = \mathit{wff}_\Sigma(\Sigma)_\alpha = \bigcup_{\alpha \in \mathcal{T}} \Lambda_\alpha$ to be the smallest set s.t:

- $\mathsf{x}_\alpha \in \mathcal{V}_\alpha$ then $\mathsf{x}_\alpha \in \Lambda_\alpha$
- $\mathsf{c}_\alpha \in \mathcal{C}_\alpha$ then $\mathsf{c}_\alpha \in \Lambda_\alpha$
- $\mathsf{A}_{\alpha\beta} \in \Lambda_{\alpha\beta}$, $\mathsf{B}_\beta \in \Lambda_\beta$ then $(\mathsf{A}\,\mathsf{B}) \in \Lambda_\alpha$

Simply Typed λ-Calculus

Let $\Sigma = (\mathcal{V}^\alpha, \mathcal{C}^\alpha)$ be a signature where

- $\mathcal{V}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{V}_\alpha$ — countably infinite sets of variables
- $\mathcal{C}^\alpha = \bigcup_{\alpha \in \mathcal{T}} \mathcal{C}_\alpha$ — possibly empty sets of constants

We define the set $\Lambda^\alpha = \mathit{wff}_\Sigma(\Sigma)_\alpha = \bigcup_{\alpha \in \mathcal{T}} \Lambda_\alpha$ to be the smallest set s.t:

- $\mathsf{x}_\alpha \in \mathcal{V}_\alpha$ then $\mathsf{x}_\alpha \in \Lambda_\alpha$
- $\mathsf{c}_\alpha \in \mathcal{C}_\alpha$ then $\mathsf{c}_\alpha \in \Lambda_\alpha$
- $\mathsf{A}_{\alpha\beta} \in \Lambda_{\alpha\beta}$, $\mathsf{B}_\beta \in \Lambda_\beta$ then $(\mathsf{A}\,\mathsf{B}) \in \Lambda_\alpha$
- $\mathsf{x}_\alpha \in \mathcal{V}_\alpha$, $\mathsf{A}_\beta \in \Lambda_\beta$ then $(\lambda \mathsf{x}_\alpha . \mathsf{A}_\beta)_{\beta\alpha} \in \Lambda_{\beta\alpha}$

Notational Conventions

- brackets may be avoided: $\mathsf{A}\,\mathsf{B}\,\mathsf{C} \rightsquigarrow ((\mathsf{A}\,\mathsf{B})\,\mathsf{C})$

Notational Conventions

- brackets may be avoided: $A\,B\,C \rightsquigarrow ((A\,B)\,C)$
- $\lambda x_\iota.A_{o\iota}\,B_\iota\,C_\iota$ — dots as far to the right as is consistent: $((\lambda x_\iota.A_{o\iota}B_\iota)C_\iota)$

Notational Conventions

- brackets may be avoided: $A\,B\,C \rightsquigarrow ((A\,B)\,C)$
- $\lambda x_\iota.A_{o\iota}\, B_\iota\, C_\iota$ — dots as far to the right as is consistent: $((\lambda x_\iota.A_{o\iota}B_\iota)C_\iota)$
- $\lambda x, y.A \rightsquigarrow (\lambda x.(\lambda y.A))$

Notational Conventions

- brackets may be avoided: $A\,B\,C \rightsquigarrow ((A\,B)\,C)$
- $\lambda x_\iota.A_{o\iota}\,B_\iota\,C_\iota$ — dots as far to the right as is consistent: $((\lambda x_\iota.A_{o\iota}B_\iota)C_\iota)$
- $\lambda x, y.A \rightsquigarrow (\lambda x.(\lambda y.A))$
- $\lambda \overline{x}^n.A \rightsquigarrow (\lambda x_1.(\ldots(\lambda x_n.A)\ldots))$

Notational Conventions

- brackets may be avoided: $\mathsf{A\,B\,C} \rightsquigarrow ((\mathsf{A\,B})\,\mathsf{C})$
- $\lambda \mathsf{x}_\iota.\mathsf{A}_{o\iota}\,\mathsf{B}_\iota\,\mathsf{C}_\iota$ — dots as far to the right as is consistent: $((\lambda \mathsf{x}_\iota.\mathsf{A}_{o\iota}\mathsf{B}_\iota)\mathsf{C}_\iota)$
- $\lambda \mathsf{x}, \mathsf{y}.\mathsf{A} \rightsquigarrow (\lambda \mathsf{x}.(\lambda \mathsf{y}.\mathsf{A}))$
- $\lambda \overline{\mathsf{x}}^{\mathsf{n}}.\mathsf{A} \rightsquigarrow (\lambda \mathsf{x}_1.(\ldots(\lambda \mathsf{x}_{\mathsf{n}}.\mathsf{A})\ldots))$
- $\lambda \overline{\mathsf{x}}.\mathsf{A}$ — n is not important

Notational Conventions

- brackets may be avoided: $A\,B\,C \rightsquigarrow ((A\,B)\,C)$
- $\lambda x_\iota.A_{o\iota}\,B_\iota\,C_\iota$ — dots as far to the right as is consistent: $((\lambda x_\iota.A_{o\iota}B_\iota)C_\iota)$
- $\lambda x, y.A \rightsquigarrow (\lambda x.(\lambda y.A))$
- $\lambda \overline{x}^n.A \rightsquigarrow (\lambda x_1.(\ldots(\lambda x_n.A)\ldots))$
- $\lambda \overline{x}.A$ — n is not important
- $(f\,\overline{A}^n) \rightsquigarrow (\ldots((f\,A^1)\,A^2)\ldots A^n)$

Def.: Positions in λ-Terms

Consider the following term:

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

Def.: Positions in λ-Terms

Consider the following term:

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

The position $[212]$ points to the red y in

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

Def.: Positions in λ-Terms

Consider the following term:

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

The position $[212]$ points to the red y in

$$((\lambda x.x)((\lambda y.\color{red}{y})(\lambda z.z)))$$

. . . Graphics on Blackboard . . .

Def.: Position (Contd.)

The expression

$$A_p$$

refers to the subterm of A at position p.

Def.: Position (Contd.)

The expression

$$A_p$$

refers to the subterm of A at position p.

Example: Consider $T := ((\lambda x.x)((\lambda y.y)(\lambda z.z)))$

Def.: Position (Contd.)

The expression

$$A_p$$

refers to the subterm of A at position p.

Example: Consider $T := ((\lambda x.x)((\lambda y.y)(\lambda z.z)))$

$$T_{[212]} = y$$

Def.: Replacement at Position

Replacement of A_p in A by a term B is denoted as

$$A[B]_p$$

Def.: Replacement at Position

Replacement of A_p in A by a term B is denoted as

$$A[B]_p$$

Example:

$$T[(f\,x)]_{[212]} = ((\lambda x.x)((\lambda y.(fx))(\lambda z.z)))$$

Def.: Scope of λ-Term

$(\lambda x.A)$: We say that A is in the scope of λ-binder that binds x.

Def.: Free and Bound Variables

An occurrence of a variable x in a term A is called bound if it is in the scope of a λ-binder that binds x.

Def.: Free and Bound Variables

An occurrence of a variable x in a term A is called bound if it is in the scope of a λ-binder that binds x.

Otherwise it is called free.

Def.: Free and Bound Variables

An occurrence of a variable x in a term A is called bound if it is in the scope of a λ-binder that binds x.

Otherwise it is called free.

We denote the set of all free variables in a λ-term as $FV(A)$.

Syntax: Simply Typed
λ-Calculus (Contd.)

Def.: Substitution

Substitution is a map

Def.: Substitution

Substitution is a map

$$[A/x] : \Lambda \longrightarrow \Lambda \quad \text{(untyped)}$$

Def.: Substitution

Substitution is a map

$$[A/x] : \Lambda \longrightarrow \Lambda \quad \text{(untyped)}$$
$$[A_\alpha/x_\alpha] : \Lambda_\alpha \longrightarrow \Lambda_\alpha \quad \text{(typed)}$$

Def.: Substitution

Substitution is a map

$$[A/x] : \Lambda \longrightarrow \Lambda \quad \text{(untyped)}$$
$$[A_\alpha/x_\alpha] : \Lambda_\alpha \longrightarrow \Lambda_\alpha \quad \text{(typed)}$$

and is defined as follows:

Def.: Substitution

Substitution is a map

$$[A/x] : \Lambda \longrightarrow \Lambda \quad \text{(untyped)}$$
$$[A_\alpha/x_\alpha] : \Lambda_\alpha \longrightarrow \Lambda_\alpha \quad \text{(typed)}$$

and is defined as follows:

1. $[N_\alpha/x_\alpha]x_\alpha = N_\alpha$

Def.: Substitution

Substitution is a map

$$[A/x] : \Lambda \longrightarrow \Lambda \qquad \text{(untyped)}$$
$$[A_\alpha/x_\alpha] : \Lambda_\alpha \longrightarrow \Lambda_\alpha \qquad \text{(typed)}$$

and is defined as follows:

1. $[N_\alpha/x_\alpha]x_\alpha = N_\alpha$
2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \wedge a_\beta \in \mathcal{V}_\beta \cup \mathcal{C}_\beta$

Def.: Substitution

Substitution is a map

$$[A/x] : \Lambda \to \Lambda \quad \text{(untyped)}$$
$$[A_\alpha/x_\alpha] : \Lambda_\alpha \to \Lambda_\alpha \quad \text{(typed)}$$

and is defined as follows:

1. $[N_\alpha/x_\alpha]x_\alpha = N_\alpha$
2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \wedge a_\beta \in \mathcal{V}_\beta \cup \mathcal{C}_\beta$
3. $[N_\alpha/x_\alpha](A_{\alpha\alpha}B_\beta) = ([N_\alpha/x_\alpha]A)([N_\alpha/x_\alpha]B)$

Def.: Substitution

Substitution is a map

$$[A/x] : \Lambda \longrightarrow \Lambda \quad \text{(untyped)}$$
$$[A_\alpha/x_\alpha] : \Lambda_\alpha \longrightarrow \Lambda_\alpha \quad \text{(typed)}$$

and is defined as follows:

1. $[N_\alpha/x_\alpha]x_\alpha = N_\alpha$
2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \wedge a_\beta \in \mathcal{V}_\beta \cup \mathcal{C}_\beta$
3. $[N_\alpha/x_\alpha](A_{\alpha\alpha}B_\beta) = ([N_\alpha/x_\alpha]A)([N_\alpha/x_\alpha]B)$
4. $[N_\alpha/x_\alpha](\lambda x_\alpha.A_\gamma) = (\lambda x_\alpha A_\gamma)$

Def.: Substitution

Substitution is a map

$$[A/x] : \Lambda \longrightarrow \Lambda \quad \text{(untyped)}$$
$$[A_\alpha/x_\alpha] : \Lambda_\alpha \longrightarrow \Lambda_\alpha \quad \text{(typed)}$$

and is defined as follows:

1. $[N_\alpha/x_\alpha]x_\alpha = N_\alpha$
2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \wedge a_\beta \in \mathcal{V}_\beta \cup \mathcal{C}_\beta$
3. $[N_\alpha/x_\alpha](A_{\alpha\alpha}B_\beta) = ([N_\alpha/x_\alpha]A)([N_\alpha/x_\alpha]B)$
4. $[N_\alpha/x_\alpha](\lambda x_\alpha.A_\gamma) = (\lambda x_\alpha A_\gamma)$
5. $[N_\alpha/x_\alpha](\lambda y_\beta.A_\gamma) = (\lambda y_\beta.[N_\alpha/x_\alpha]A_\gamma)$ if $x_\alpha \neq y_\beta \wedge (y_\beta \notin FV(N_\alpha) \vee x_\alpha \notin FV(A_\gamma))$

Def.: Substitution

Substitution is a map

$$[A/x] : \Lambda \to \Lambda \quad \text{(untyped)}$$
$$[A_\alpha/x_\alpha] : \Lambda_\alpha \to \Lambda_\alpha \quad \text{(typed)}$$

and is defined as follows:

1. $[N_\alpha/x_\alpha]x_\alpha = N_\alpha$
2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \land a_\beta \in \mathcal{V}_\beta \cup \mathcal{C}_\beta$
3. $[N_\alpha/x_\alpha](A_{\alpha\alpha}B_\beta) = ([N_\alpha/x_\alpha]A)([N_\alpha/x_\alpha]B)$
4. $[N_\alpha/x_\alpha](\lambda x_\alpha.A_\gamma) = (\lambda x_\alpha A_\gamma)$
5. $[N_\alpha/x_\alpha](\lambda y_\beta.A_\gamma) = (\lambda y_\beta.[N_\alpha/x_\alpha]A_\gamma)$ if $x_\alpha \neq y_\beta \land (y_\beta \notin FV(N_\alpha) \lor x_\alpha \notin FV(A_\gamma))$
6. $[N_\alpha/x_\alpha](\lambda y_\alpha.A_\gamma) = (\lambda z_\beta.[N_\alpha/x_\alpha][z_\beta/y_\beta]A_\gamma)$ if $x_\alpha \neq y_\beta \land$ $(y_\beta \in FV(N_\alpha) \land x_\alpha \in FV(A_\gamma))$ and z is a 'fresh' variable.

Ex.: Substitution

- $[y/x](\lambda y.x)$ — the occurrence of x is free
  $\neq (\lambda y.y)$ — if we replace x with y, the variable y becomes bound.

Ex.: Substitution

- $[y/x](\lambda y.x)$ — the occurrence of x is free
  $\neq (\lambda y.y)$ — if we replace x with y, the variable y becomes bound.
- $[y/x](\lambda y.x)$ — the occurrence of x is free
  $= (\lambda z[y/x][z/y]x)$ — we need a fresh variable
  $= (\lambda z.y)$ — the occurrence of y is free

Ex.: Substitution

- $[y/x](\lambda y.x)$ — the occurrence of x is free
  $\neq (\lambda y.y)$ — if we replace x with y, the variable y becomes bound.
- $[y/x](\lambda y.x)$ — the occurrence of x is free
  $= (\lambda z[y/x][z/y]x)$ — we need a fresh variable
  $= (\lambda z.y)$ — the occurrence of y is free
- Further Examples on Blackboard

Ex.: Substitution

- $[y/x](\lambda y.x)$ — the occurrence of x is free
  $\neq (\lambda y.y)$ — if we replace x with y, the variable y becomes bound.
- $[y/x](\lambda y.x)$ — the occurrence of x is free
  $= (\lambda z[y/x][z/y]x)$ — we need a fresh variable
  $= (\lambda z.y)$ — the occurrence of y is free
- Further Examples on Blackboard
- Claim: $[N/x]A = A$ if $x \notin FV(A)$
  Proof: Induction on A

Def.: α-Conversion

$$[\lambda x.\, M] \rightarrow_{\alpha} [\lambda y.\, [y/x]M]$$

where $y \notin FV(M)$

Def.: α-Conversion

$$[\lambda x.\, M] \rightarrow_\alpha [\lambda y.\, [y/x]M]$$

where $y \notin FV(M)$

$$A =^\alpha B$$

if A can be converted to B by renaming the bound variables. We read $A =_\alpha B$ as A is α-equal to B.

Def.: α-Conversion

$$[\lambda x.\, M] \rightarrow_\alpha [\lambda y.\, [y/x]M]$$

where $y \notin FV(M)$

$$A =^\alpha B$$

if A can be converted to B by renaming the bound variables. We read $A =_\alpha B$ as **A is α-equal to B**.

From now on $(\lambda y.\, y) = (\lambda z.\, z)$, that is, we will say that two terms are simply equal, if they are α-equal. Two terms are equal means that two terms are α-convertable.

Def.: β-Conversion

A β-redex is a term $((\lambda x.\, A)B)$. The β-reduct of this redex is $[B/x]A$.

Def.: β-Conversion

A β-redex is a term $((\lambda x.\, A)B)$. The β-reduct of this redex is $[B/x]A$.

We say $M \rightarrow_\beta N$, ie. β-reduces in 1 step, if

$$\begin{aligned} M &= P[(\lambda x.\, A)B]_p \\ N &= P[[B/x]A]_p \end{aligned}$$

Def.: β-Conversion

A β-redex is a term $((\lambda x. A)B)$. The β-reduct of this redex is $[B/x]A$.

We say $M \rightarrow_\beta N$, ie. β-reduces in 1 step, if

$$\begin{aligned} M &= P[(\lambda x. A)B]_p \\ N &= P[[B/x]A]_p \end{aligned}$$

We say $M \twoheadrightarrow_\beta N$, ie. β-reduces in several steps, if $\exists M^1, \ldots, M^n$ for $n \geq 1$ such that $M = M^1$ and $N = M^n$ and $M^i \rightarrow_\beta M^{i+1}$.

Def.: β-Normal Form

A term is called β-normal if it contains no β-redexes.

Def.: β-Normal Form

A term is called β-normal if it contains no β-redexes.

Any term that does not contain λ-abstractions is β-normal.

Def.: β-Normal Form

A term is called β-normal if it contains no β-redexes.

Any term that does not contain λ-abstractions is β-normal.

A term is called β-head normal if the head term of its outermost application can not be further reduced.

Def.: β-Normal Form

A term is called β-normal if it contains no β-redexes.

Any term that does not contain λ-abstractions is β-normal.

A term is called β-head normal if the head term of its outermost application can not be further reduced.

Any term that does not contain λ-abstractions is β-head normal.

Thm.: Church-Rosser Property for $\twoheadrightarrow_\beta$

$$
\begin{tikzcd}
 & \mathsf{T}_\alpha \arrow[dl, two heads, "\beta"'] \arrow[dr, two heads, "\beta"] & \\
\mathsf{L}_\alpha \arrow[dr, two heads, "\beta"'] & & \mathsf{R}_\alpha \arrow[dl, two heads, "\beta"] \\
 & \mathsf{B}_\alpha &
\end{tikzcd}
$$

Thm.: Church-Rosser Property for $\twoheadrightarrow_\beta$

$$
\begin{tikzcd}
 & T_\alpha \arrow[dl, two heads, "\beta"'] \arrow[dr, two heads, "\beta"] & \\
L_\alpha \arrow[dr, two heads, "\beta"'] & & R_\alpha \arrow[dl, two heads, "\beta"] \\
 & B_\alpha &
\end{tikzcd}
$$

If T_α β-reduces in multiple steps with one strategy to L_α and with another strategy to R_α then there exists a term B_α such that L_α and R_α β-reduce in multiple steps to B_α.

Thm.: Church-Rosser Property for $\twoheadrightarrow_\beta$

$$
\begin{tikzcd}
 & \mathsf{T}_\alpha \arrow[dl, two heads, "\beta"'] \arrow[dr, two heads, "\beta"] & \\
\mathsf{L}_\alpha \arrow[dr, two heads, "\beta"'] & & \mathsf{R}_\alpha \arrow[dl, two heads, "\beta"] \\
 & \mathsf{B}_\alpha &
\end{tikzcd}
$$

If T_α β-reduces in multiple steps with one strategy to L_α and with another strategy to R_α then there exists a term B_α such that L_α and R_α β-reduce in multiple steps to B_α.

Note that B_α is not necessarily in normal form.

Thm.: Church-Rosser Property for $\twoheadrightarrow_\beta$

$$
\begin{tikzcd}
 & \mathsf{T}_\alpha \arrow[dl, two heads, "\beta"'] \arrow[dr, two heads, "\beta"] & \\
\mathsf{L}_\alpha \arrow[dr, two heads, "\beta"'] & & \mathsf{R}_\alpha \arrow[dl, two heads, "\beta"] \\
 & \mathsf{B}_\alpha &
\end{tikzcd}
$$

If T_α β-reduces in multiple steps with one strategy to L_α and with another strategy to R_α then there exists a term B_α such that L_α and R_α β-reduce in multiple steps to B_α.

Note that B_α is not necessarily in normal form.

The Church-Rosser Property for $\twoheadrightarrow_\beta$ holds for Λ and Λ^α.

Ex.: Church-Rosser Property for $\twoheadrightarrow_\beta$

$$
\begin{tikzcd}
 & ((\lambda f.\, f)\, g)((\lambda x.\, x)\, a) \arrow[dl, two heads, "\beta"'] \arrow[dr, two heads, "\beta"] & \\
g((\lambda x.\, x)\, a) \arrow[dr, two heads, "\beta"'] & & ((\lambda f.\, f) g)\, a \arrow[dl, two heads, "\beta"] \\
 & g\, a &
\end{tikzcd}
$$

Termination

Do we always get a β-normal form as we apply β-reduction?

Termination

Do we always get a β-normal form as we apply β-reduction?

Typed Case: Forall A_α there exists a unique (up to α-conversion) β-normal term B such that $A \twoheadrightarrow_\beta B$

Termination

Do we always get a β-normal form as we apply β-reduction?

Typed Case: Forall A_α there exists a unique (up to α-conversion) β-normal term B such that $A \twoheadrightarrow_\beta B$

Untyped Case: Consider the term $\omega = (\lambda x.\, xx)$

$$(\lambda x.\, xx)(\lambda x.\, xx) \rightarrow^1_\beta \omega\omega$$

Def.: η-Conversion

A η-redex is a term of the form $(\lambda x_\beta . F_{\alpha\beta}\, x)$ where $x \notin FV(F)$. The η-reduct of this term is F.

Def.: η-Conversion

A η-redex is a term of the form $(\lambda x_\beta .\, F_{\alpha\beta}\, x)$ where $x \notin FV(F)$. The η-reduct of this term is F.

We say $M \rightarrow_\eta N$, ie. η-reduces in 1 step, if

$$\begin{aligned} M &= P[(\lambda x_\beta .\, F_{\alpha\beta} x)]_p \\ N &= P[F]_p \end{aligned}$$

Def.: η-Conversion

A η-redex is a term of the form $(\lambda x_\beta . F_{\alpha\beta}\, x)$ where $x \notin FV(F)$. The η-reduct of this term is F.

We say $M \rightarrow_\eta N$, ie. η-reduces in 1 step, if

$$\begin{aligned} M &= P[(\lambda x_\beta . F_{\alpha\beta} x)]_p \\ N &= P[F]_p \end{aligned}$$

We say $M \twoheadrightarrow_\eta N$, ie. η-reduces in several steps, if $\exists M^1, \ldots, M^n$ for $n \geq 1$ such that $M = M^1$ and $N = M^n$ and $M^i \rightarrow_\beta M^{i+1}$.

Def.: η-Normal Form

A term is called η-normal if it contains no η-redexes.

Thm.: Church-Rosser Property for $\twoheadrightarrow_\eta$

$$\begin{tikzcd}
 & \mathsf{T}_\alpha \arrow[dl, two heads, "\eta"'] \arrow[dr, two heads, "\eta"] & \\
\mathsf{L}_\alpha \arrow[dr, two heads, "\eta"'] & & \mathsf{R}_\alpha \arrow[dl, two heads, "\eta"] \\
 & \mathsf{B}_\alpha &
\end{tikzcd}$$

Thm.: Church-Rosser Property for $\twoheadrightarrow_\eta$

$$\begin{tikzcd}
& T_\alpha \arrow[dl, two heads, "\eta"'] \arrow[dr, two heads, "\eta"] & \\
L_\alpha \arrow[dr, two heads, "\eta"'] & & R_\alpha \arrow[dl, two heads, "\eta"] \\
& B_\alpha &
\end{tikzcd}$$

If T_α η-reduces in multiple steps with one strategy to L_α and with another strategy to R_α then there exists a term B_α such that L_α and R_α η-reduce in multiple steps to B_α.

Thm.: Church-Rosser Property for $\twoheadrightarrow_\eta$

$$\begin{tikzcd}
 & \mathsf{T}_\alpha \arrow[dl, two heads, "\eta"'] \arrow[dr, two heads, "\eta"] & \\
\mathsf{L}_\alpha \arrow[dr, two heads, "\eta"'] & & \mathsf{R}_\alpha \arrow[dl, two heads, "\eta"] \\
 & \mathsf{B}_\alpha &
\end{tikzcd}$$

If T_α η-reduces in multiple steps with one strategy to L_α and with another strategy to R_α then there exists a term B_α such that L_α and R_α η-reduce in multiple steps to B_α.

The Church-Rosser Property for $\twoheadrightarrow_\eta$ holds for Λ and Λ^α.

Def.: $\beta\eta$-Conversion

$$\longrightarrow_{\beta\eta} := \longrightarrow_{\beta} \cup \longrightarrow_{\eta}$$

Def.: $\beta\eta$-Conversion

$$\longrightarrow_{\beta\eta} := \longrightarrow_{\beta} \cup \longrightarrow_{\eta}$$

If M $\longrightarrow_{\beta\eta}$ N we say M $\beta\eta$-reduces in 1 step to N.

Def.: $\beta\eta$-Conversion

$$\longrightarrow_{\beta\eta} := \longrightarrow_{\beta} \cup \longrightarrow_{\eta}$$

If $M \longrightarrow_{\beta\eta} N$ we say M $\beta\eta$-reduces in 1 step to N.

We say $M \twoheadrightarrow_{\beta\eta} N$, ie. η-reduces in several steps, if $\exists M^1, \ldots, M^n$ for $n \geq 1$ such that $M = M^1$ and $N = M^n$ and $M^i \longrightarrow_{\beta\eta} M^{i+1}$.

Def.: $\beta\eta$-Normal Form

A term is $\beta\eta$-normal if it contains no β-redexes and no η-redexes.

Thm.: Church-Rosser Property for $\twoheadrightarrow_{\beta\eta}$

$$\begin{tikzcd}
 & T_\alpha \arrow[dl, dashed, two heads, "\beta\eta"'] \arrow[dr, dashed, two heads, "\beta\eta"] & \\
L_\alpha \arrow[dr, dashed, two heads, "\beta\eta"'] & & R_\alpha \arrow[dl, dashed, two heads, "\beta\eta"] \\
 & B_\alpha &
\end{tikzcd}$$

Thm.: Church-Rosser Property for $\twoheadrightarrow_{\beta\eta}$

$$\begin{tikzcd} & \mathsf{T}_\alpha \arrow[dl, two heads, "\beta\eta"'] \arrow[dr, two heads, "\beta\eta"] & \\ \mathsf{L}_\alpha \arrow[dr, two heads, "\beta\eta"'] & & \mathsf{R}_\alpha \arrow[dl, two heads, "\beta\eta"] \\ & \mathsf{B}_\alpha & \end{tikzcd}$$

If T_α $\beta\eta$-reduces in multiple steps with one strategy to L_α and with another strategy to R_α then there exists a term B_α such that L_α and R_α $\beta\eta$-reduce in multiple steps to B_α.

Thm.: Church-Rosser Property for $\twoheadrightarrow_{\beta\eta}$

$$\begin{tikzcd}
 & T_\alpha \arrow[dl, two heads, "\beta\eta"'] \arrow[dr, two heads, "\beta\eta"] & \\
L_\alpha \arrow[dr, two heads, "\beta\eta"'] & & R_\alpha \arrow[dl, two heads, "\beta\eta"] \\
 & B_\alpha &
\end{tikzcd}$$

If T_α $\beta\eta$-reduces in multiple steps with one strategy to L_α and with another strategy to R_α then there exists a term B_α such that L_α and R_α $\beta\eta$-reduce in multiple steps to B_α.

The Church-Rosser Property for $\twoheadrightarrow_{\beta\eta}$ holds for Λ and Λ^α.

Thm.: Strong Church-Rosser Property

In Λ^{α} (simply typed λ-calculus) the relations $\twoheadrightarrow_{\beta}$ and $\twoheadrightarrow_{\beta\eta}$ have the strong Church Rosser property:

Thm.: Strong Church-Rosser Property

In Λ^α (simply typed λ-calculus) the relations $\twoheadrightarrow_\beta$ and $\twoheadrightarrow_{\beta\eta}$ have the strong Church Rosser property: for very term A_τ there exists a unique (up to α-renaming) β-normal resp. $\beta\eta$-normal term B_τ such that $A_\tau \twoheadrightarrow_\beta B_\tau$ resp. $A_\tau \twoheadrightarrow_{\beta\eta} B_\tau$.

Def.: Long $\beta\eta$-Normal Form

Let $\mathsf{n} \geq 0$, $\alpha^1, \ldots, \alpha^\mathsf{n} \in \mathcal{T}$, and $\beta \in \{\mathsf{o}, \iota\}$. A term A of type $(\beta, \alpha^\mathsf{n}, \ldots, \alpha^1)$ is in long $\beta\eta$-normal form if it is of form

$$\lambda \mathsf{x}^1_{\alpha^1} \ldots \mathsf{x}^\mathsf{n}_{\alpha^\mathsf{n}}.(\mathsf{h}_{\beta\gamma^\mathsf{m}\ldots\gamma^1} \mathsf{A}^1_{\gamma^1} \ldots \mathsf{A}^\mathsf{m}_{\gamma^\mathsf{m}})$$

for a variable or constant $\mathsf{h}_{\beta\gamma^\mathsf{m}\ldots\gamma^1}$, $\mathsf{m} \geq 0$ and long $\beta\eta$-normal forms $\mathsf{A}^1_{\gamma^1}, \ldots, \mathsf{A}^\mathsf{m}_{\gamma^\mathsf{m}}$.

Def.: Long $\beta\eta$-Normal Form

Let $\mathsf{n} \geq 0$, $\alpha^1, \ldots, \alpha^\mathsf{n} \in \mathcal{T}$, and $\beta \in \{\mathsf{o}, \iota\}$. A term A of type $(\beta, \alpha^\mathsf{n}, \ldots, \alpha^1)$ is in long $\beta\eta$-normal form if it is of form

$$\lambda \mathsf{x}^1_{\alpha^1} \ldots \mathsf{x}^\mathsf{n}_{\alpha^\mathsf{n}}.(\mathsf{h}_{\beta\gamma^\mathsf{m}\ldots\gamma^1} \mathsf{A}^1_{\gamma^1} \ldots \mathsf{A}^\mathsf{m}_{\gamma^\mathsf{m}})$$

for a variable or constant $\mathsf{h}_{\beta\gamma^\mathsf{m}\ldots\gamma^1}$, $\mathsf{m} \geq 0$ and long $\beta\eta$-normal forms $\mathsf{A}^1_{\gamma^1}, \ldots, \mathsf{A}^\mathsf{m}_{\gamma^\mathsf{m}}$. Note that this is an inductive definition; the base case is when $\mathsf{m} = 0$.

Def.: Long $\beta\eta$-Normal Form

Let $\mathsf{n} \geq 0$, $\alpha^1, \ldots, \alpha^\mathsf{n} \in \mathcal{T}$, and $\beta \in \{\mathsf{o}, \iota\}$. A term A of type $(\beta, \alpha^\mathsf{n}, \ldots, \alpha^1)$ is in long $\beta\eta$-normal form if it is of form

$$\lambda \mathsf{x}^1_{\alpha^1} \ldots \mathsf{x}^\mathsf{n}_{\alpha^\mathsf{n}}.(\mathsf{h}_{\beta\gamma^\mathsf{m}\ldots\gamma^1} \mathsf{A}^1_{\gamma^1} \ldots \mathsf{A}^\mathsf{m}_{\gamma^\mathsf{m}})$$

for a variable or constant $\mathsf{h}_{\beta\gamma^\mathsf{m}\ldots\gamma^1}$, $\mathsf{m} \geq 0$ and long $\beta\eta$-normal forms $\mathsf{A}^1_{\gamma^1}, \ldots, \mathsf{A}^\mathsf{m}_{\gamma^\mathsf{m}}$. Note that this is an inductive definition; the base case is when $\mathsf{m} = 0$. Note that if $\lambda\overline{\mathsf{x}^\mathsf{n}}.(\mathsf{h}\overline{\mathsf{A}^\mathsf{m}})$ is in long $\beta\eta$-normal form then $(\mathsf{h}\overline{\mathsf{A}^\mathsf{m}})$ is of base type.

Ex.: Long $\beta\eta$-Normal Form

Consider the $\beta\eta$-normal term $\mathsf{f}_{\iota(\iota\iota)}$.

$$\mathsf{f}_{\iota(\iota\iota)}$$

$$\uparrow^{\eta}$$

$$\lambda \mathsf{w}_{\iota\iota}.\,(\mathsf{f}_{\iota(\iota\iota)}\mathsf{w}_{\iota\iota})$$

$$\uparrow^{\eta}$$

$$\lambda \mathsf{w}_{\iota\iota}.\,(\mathsf{f}(\lambda \mathsf{x}_{\iota}.\,\mathsf{w}_{\iota\iota}\mathsf{x}))$$

Thm.: Long $\beta\eta$-Normal Form

For every term A there is unique long $\beta\eta$-normal form B such that $A =^{\beta\eta} B$.

Rem.: $\beta\eta$-Head Normal Form

Instead of terms in long $\beta\eta$-normal form we often use in practice terms in $\beta\eta$-head normal form.

Rem.: $\beta\eta$-Head Normal Form

Instead of terms in long $\beta\eta$-normal form we often use in practice terms in $\beta\eta$-head normal form. Definition is similar to long $\beta\eta$-normal, but we do not require the embedded terms $A^i_{\gamma^i}$ to be in normal form.

Notation

- $A{\downarrow_\beta}$ is the β-normal form of A.

Notation

- $A{\downarrow_\beta}$ is the β-normal form of A.
- $A{\downarrow_\eta}$ is the η-normal form of A.

Notation

- $A{\downarrow_\beta}$ is the β-normal form of A.
- $A{\downarrow_\eta}$ is the η-normal form of A.
- $A{\downarrow}$ is the $\beta\eta$-normal form of A.

Notation

- $A{\downarrow_\beta}$ is the β-normal form of A.
- $A{\downarrow_\eta}$ is the η-normal form of A.
- $A{\downarrow}$ is the $\beta\eta$-normal form of A.
- $A{\updownarrow}$ is the long $\beta\eta$-normal form of A.

Semantics: Σ-Evaluations

Ex.: An Interesting Applicative Structure

$\mathsf{D}_\alpha := \{\mathsf{A}_\alpha \in \Lambda_\alpha | \mathsf{A} \text{ is closed}\}$.

- Is D_α non-empty for all α?

Ex.: An Interesting Applicative Structure

$\mathsf{D}_\alpha := \{\mathsf{A}_\alpha \in \Lambda_\alpha | \mathsf{A} \text{ is closed}\}$.

- Is D_α non-empty for all α?
- If $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_o \neq \emptyset$, then $\forall \alpha \in \mathcal{T}. \Lambda_\alpha \neq \emptyset$.

Ex.: An Interesting Applicative Structure

$D_\alpha := \{A_\alpha \in \Lambda_\alpha | \, A \text{ is closed}\}$.

- Is D_α non-empty for all α?
- If $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_o \neq \emptyset$, then $\forall \alpha \in \mathcal{T} . \Lambda_\alpha \neq \emptyset$.
- Is $D_{\alpha\beta}$ a set of functions? (ie. $D_{\alpha\beta} \subseteq (D_\alpha)^{D_\beta}$?) — No!

Ex.: An Interesting Applicative Structure

$D_\alpha := \{A_\alpha \in \Lambda_\alpha \mid A \text{ is closed}\}$.

- Is D_α non-empty for all α?
- If $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_o \neq \emptyset$, then $\forall \alpha \in \mathcal{T}. \Lambda_\alpha \neq \emptyset$.
- Is $D_{\alpha\beta}$ a set of functions? (ie. $D_{\alpha\beta} \subseteq (D_\alpha)^{D_\beta}$?) — No!
- Is $(\lambda x_\iota\, x) \in D_{\iota\iota}$? — Yes!

Ex.: An Interesting Applicative Structure

$D_\alpha := \{A_\alpha \in \Lambda_\alpha \mid A \text{ is closed}\}$.

- Is D_α non-empty for all α?
- If $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_o \neq \emptyset$, then $\forall \alpha \in \mathcal{T}. \Lambda_\alpha \neq \emptyset$.
- Is $D_{\alpha\beta}$ a set of functions? (ie. $D_{\alpha\beta} \subseteq (D_\alpha)^{D_\beta}$?) — No!
- Is $(\lambda x_\iota \, x) \in D_{\iota\iota}$? — Yes!
- $D = (D_\alpha)_{\alpha \in \mathcal{T}}$ is not a frame!

Ex.: An Interesting Applicative Structure

$\mathsf{D}_\alpha := \{\mathsf{A}_\alpha \in \Lambda_\alpha \,|\, \mathsf{A} \text{ is closed}\}$.

- Is D_α non-empty for all α?
- If $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_o \neq \emptyset$, then $\forall \alpha \in \mathcal{T}.\, \Lambda_\alpha \neq \emptyset$.
- Is $\mathsf{D}_{\alpha\beta}$ a set of functions? (ie. $\mathsf{D}_{\alpha\beta} \subseteq (\mathsf{D}_\alpha)^{\mathsf{D}_\beta}$?) — No!
- Is $(\lambda \mathsf{x}_\iota\, \mathsf{x}) \in \mathsf{D}_{\iota\iota}$? — Yes!
- $\mathsf{D} = (\mathsf{D}_\alpha)_{\alpha \in \mathcal{T}}$ is not a frame!
- It requires a specific application operator $@ : \mathsf{D}_{\alpha\beta} \times \mathsf{D}_\beta \longrightarrow \mathsf{D}_\alpha$

Ex.: An Interesting Applicative Structure

$D_\alpha := \{A_\alpha \in \Lambda_\alpha | \, A \text{ is closed}\}$.

- Is D_α non-empty for all α?
- If $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_o \neq \emptyset$, then $\forall \alpha \in \mathcal{T} . \Lambda_\alpha \neq \emptyset$.
- Is $D_{\alpha\beta}$ a set of functions? (ie. $D_{\alpha\beta} \subseteq (D_\alpha)^{D_\beta}$?) — No!
- Is $(\lambda x_\iota \, x) \in D_{\iota\iota}$? — Yes!
- $D = (D_\alpha)_{\alpha \in \mathcal{T}}$ is not a frame!
- It requires a specific application operator $@ : D_{\alpha\beta} \times D_\beta \longrightarrow D_\alpha$
- If Λ_α is non-empty for all $\alpha \in \mathcal{T}$, then $< D, @ >$ is an applicative structure.

Ex.: Interpretation of Terms

Syntax   Semantics $< \mathrm{D}, @ >$

$(\lambda \mathrm{x}_{\iota}.\, \mathrm{x})$

Ex.: Interpretation of Terms

Syntax	Semantics	$< \mathsf{D}, @ >$
$(\lambda \mathsf{x}_\iota . \mathsf{x})$	$(\lambda \mathsf{x}_\iota . \mathsf{x})$	

Ex.: Interpretation of Terms

Syntax	Semantics	$< \mathsf{D}, @ >$
$(\lambda \mathsf{x}_\iota . \mathsf{x})$	$(\lambda \mathsf{x}_\iota . \mathsf{x})$	$\in \mathsf{D}_{\iota\iota}$

Ex.: Interpretation of Terms

Syntax	Semantics	$< \mathsf{D}, @ >$
$(\lambda \mathsf{x}_\iota . \mathsf{x})$	$(\lambda \mathsf{x}_\iota . \mathsf{x})$	$\in \mathsf{D}_{\iota\iota}$
y_ι		

Ex.: Interpretation of Terms

Syntax	Semantics	$< D, @ >$
$(\lambda x_\iota . x)$	$(\lambda x_\iota . x)$	$\in D_{\iota\iota}$
y_ι	$\varphi(y)$	

Ex.: Interpretation of Terms

Syntax	Semantics	$< \mathsf{D}, @ >$
$(\lambda \mathsf{x}_\iota . \mathsf{x})$	$(\lambda \mathsf{x}_\iota . \mathsf{x})$	$\in \mathsf{D}_{\iota\iota}$
y_ι	$\varphi(\mathsf{y})$	$\in \mathsf{D}_\iota$

Ex.: Interpretation of Terms

Syntax	Semantics	$< D, @ >$
$(\lambda x_\iota . x)$	$(\lambda x_\iota . x)$	$\in D_{\iota\iota}$
y_ι	$\varphi(y)$	$\in D_\iota$
$a_\iota \in C$		

Ex.: Interpretation of Terms

Syntax	Semantics	$< D, @ >$
$(\lambda x_\iota . x)$	$(\lambda x_\iota . x)$	$\in D_{\iota\iota}$
y_ι	$\varphi(y)$	$\in D_\iota$
$a_\iota \in C$	a	

Ex.: Interpretation of Terms

Syntax	Semantics	$< D, @ >$
$(\lambda x_\iota . x)$	$(\lambda x_\iota . x)$	$\in D_{\iota\iota}$
y_ι	$\varphi(y)$	$\in D_\iota$
$a_\iota \in C$	a	$\in D_\iota$

Ex.: Interpretation of Terms

Syntax	Semantics	$< D, @ >$
$(\lambda x_\iota . x)$	$(\lambda x_\iota . x)$	$\in D_{\iota\iota}$
y_ι	$\varphi(y)$	$\in D_\iota$
$a_\iota \in C$	a	$\in D_\iota$
$(\lambda x_\iota . x) a_\iota$		

Ex.: Interpretation of Terms

Syntax	Semantics	$< D, @ >$
$(\lambda x_\iota . x)$	$(\lambda x_\iota . x)$	$\in D_{\iota\iota}$
y_ι	$\varphi(y)$	$\in D_\iota$
$a_\iota \in C$	a	$\in D_\iota$
$(\lambda x_\iota . x) a_\iota$	$(\lambda x_\iota . x) @ a_\iota$	

Ex.: Interpretation of Terms

Syntax	Semantics	$< D, @ >$
$(\lambda x_\iota . x)$	$(\lambda x_\iota . x)$	$\in D_{\iota\iota}$
y_ι	$\varphi(y)$	$\in D_\iota$
$a_\iota \in C$	a	$\in D_\iota$
$(\lambda x_\iota . x) a_\iota$	$(\lambda x_\iota . x) @ a_\iota$	$\in D_\iota$

Ex.: Interpretation of Terms

Syntax	Semantics	$< D, @ >$
$(\lambda x_\iota . x)$	$(\lambda x_\iota . x)$	$\in D_{\iota\iota}$
y_ι	$\varphi(y)$	$\in D_\iota$
$a_\iota \in C$	a	$\in D_\iota$
$(\lambda x_\iota . x) a_\iota$	$(\lambda x_\iota . x) @ a_\iota$	$\in D_\iota$

Remark: The variable y_ι is a non-closed well-formed formula of type ι. We need an assignment $\varphi_\alpha : V_\alpha \longrightarrow D_\alpha$ to give it a meaning.

Ex.: Interesting Applicative Structures

- Let $\mathsf{D}_\alpha\downarrow_\beta := \{\mathsf{A}_\alpha \in \Lambda_\alpha | \, \mathsf{A} \text{ is closed and } \mathsf{A} \text{ is in } \beta\text{-normal form}\}$

Ex.: Interesting Applicative Structures

- Let $\mathsf{D}_\alpha\downarrow_\beta := \{\mathsf{A}_\alpha \in \Lambda_\alpha \mid \mathsf{A}$ is closed and A is in β-normal form$\}$
- Let $\mathsf{D} := (\mathsf{D}_\alpha\downarrow_\beta)_{\alpha\in\mathcal{T}}$

Ex.: Interesting Applicative Structures

- Let $\mathsf{D}_\alpha \downarrow_\beta := \{\mathsf{A}_\alpha \in \Lambda_\alpha |\, \mathsf{A}$ is closed and A is in β-normal form$\}$
- Let $\mathsf{D} := (\mathsf{D}_\alpha \downarrow_\beta)_{\alpha \in \mathcal{T}}$
- Let $@^\beta_{\gamma\delta} : \mathsf{D}_{\gamma\delta} \times \mathsf{D}_\delta \longrightarrow \mathsf{D}_\gamma$ be defined by

$$\mathsf{F}_{\gamma\delta} @^\beta_{\gamma\delta} \mathsf{G}_\delta = (\mathsf{F}\,\mathsf{G}) \downarrow_\beta$$

for all $\mathsf{F}_{\gamma\delta} \in \mathsf{D}_{\gamma\delta}$ and $\mathsf{G}_\delta \in \mathsf{D}_\delta$.

Ex.: Interesting Applicative Structures

- Let $\mathsf{D}_\alpha \downarrow_\beta := \{\mathsf{A}_\alpha \in \Lambda_\alpha | \, \mathsf{A}$ is closed and A is in β-normal form$\}$
- Let $\mathsf{D} := (\mathsf{D}_\alpha \downarrow_\beta)_{\alpha \in \mathcal{T}}$
- Let $@^\beta_{\gamma\delta} : \mathsf{D}_{\gamma\delta} \times \mathsf{D}_\delta \longrightarrow \mathsf{D}_\gamma$ be defined by

$$\mathsf{F}_{\gamma\delta} @^\beta_{\gamma\delta} \mathsf{G}_\delta = (\mathsf{F}\,\mathsf{G}) \downarrow_\beta$$

  for all $\mathsf{F}_{\gamma\delta} \in \mathsf{D}_{\gamma\delta}$ and $\mathsf{G}_\delta \in \mathsf{D}_\delta$.
- $@^\beta = (@^\beta_{\gamma\delta})_{\gamma\delta \in \mathcal{T}}$

Ex.: Interesting Applicative Structures

- Let $\mathsf{D}_\alpha \downarrow_\beta := \{\mathsf{A}_\alpha \in \Lambda_\alpha | \, \mathsf{A}$ is closed and A is in β-normal form$\}$
- Let $\mathsf{D} := (\mathsf{D}_\alpha \downarrow_\beta)_{\alpha \in \mathcal{T}}$
- Let $@^\beta_{\gamma\delta} : \mathsf{D}_{\gamma\delta} \times \mathsf{D}_\delta \longrightarrow \mathsf{D}_\gamma$ be defined by

$$\mathsf{F}_{\gamma\delta} @^\beta_{\gamma\delta} \mathsf{G}_\delta = (\mathsf{F}\,\mathsf{G}) \downarrow_\beta$$

  for all $\mathsf{F}_{\gamma\delta} \in \mathsf{D}_{\gamma\delta}$ and $\mathsf{G}_\delta \in \mathsf{D}_\delta$.
- $@^\beta = (@^\beta_{\gamma\delta})_{\gamma\delta \in \mathcal{T}}$

Claim: If $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_o \neq \emptyset$ (i.e., at least one constant for each base type is given), then $(\mathsf{D}, @^\beta)$ is an applicative structure.

Ex.: Interesting Applicative Structures

Proof:

- Is $\mathrm{D}_\alpha\downarrow_\beta$ nonempty for all $\alpha \in \mathcal{T}$?

Ex.: Interesting Applicative Structures

Proof:

- Is $\mathrm{D}_\alpha\downarrow_\beta$ nonempty for all $\alpha \in \mathcal{T}$?
- Yes! This follows since $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_\wr \neq \emptyset$.

Ex.: Interesting Applicative Structures

Proof:

- Is $\mathsf{D}_\alpha \downarrow_\beta$ nonempty for all $\alpha \in \mathcal{T}$?
- Yes! This follows since $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_\wr \neq \emptyset$.
- Is $\mathsf{F}_{\gamma\delta} @^{\beta}_{\gamma\delta} \mathsf{G}_\delta \in \mathsf{D}_\gamma \downarrow_\beta$?

Ex.: Interesting Applicative Structures

Proof:

- Is $D_\alpha \downarrow_\beta$ nonempty for all $\alpha \in \mathcal{T}$?
- Yes! This follows since $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_\wr \neq \emptyset$.
- Is $F_{\gamma\delta} @^{\beta}_{\gamma\delta} G_\delta \in D_\gamma \downarrow_\beta$?
- Let's check: $F_{\gamma\delta} @^{\beta}_{\gamma\delta} G_\delta = (F\,G) \downarrow_\beta \in D_\gamma \downarrow_\beta$

Ex.: Interesting Applicative Structures

- Let $\mathrm{D}_\alpha \downarrow_{\beta\eta} := \{\mathrm{A}_\alpha \in \Lambda_\alpha |\ \mathrm{A}$ is closed and A is in $\beta\eta$-normal form$\}$

Ex.: Interesting Applicative Structures

- Let $\mathsf{D}_\alpha\downarrow_{\beta\eta} := \{\mathsf{A}_\alpha \in \Lambda_\alpha |$ A is closed and A is in $\beta\eta$-normal form$\}$
- Let $\mathsf{D} := (\mathsf{D}_\alpha\downarrow_{\beta\eta})_{\alpha\in\mathcal{T}}$

Ex.: Interesting Applicative Structures

- Let $\mathsf{D}_\alpha \downarrow_{\beta\eta} := \{\mathsf{A}_\alpha \in \Lambda_\alpha |$ A is closed and A is in $\beta\eta$-normal form$\}$
- Let $\mathsf{D} := (\mathsf{D}_\alpha \downarrow_{\beta\eta})_{\alpha\in\mathcal{T}}$
- Let $@^{\beta\eta}_{\gamma\delta} : \mathsf{D}_{\gamma\delta} \times \mathsf{D}_\delta \longrightarrow \mathsf{D}_\gamma$ be defined by

$$\mathsf{F}_{\gamma\delta} @^{\beta\eta}_{\gamma\delta} \mathsf{G}_\delta = (\mathsf{F}\,\mathsf{G}) \downarrow$$

for all $\mathsf{F}_{\gamma\delta} \in \mathsf{D}_{\gamma\delta}$ and $\mathsf{G}_\delta \in \mathsf{D}_\delta$.

Ex.: Interesting Applicative Structures

- Let $\mathsf{D}_\alpha\downarrow_{\beta\eta} := \{\mathsf{A}_\alpha \in \Lambda_\alpha \mid \mathsf{A}$ is closed and A is in $\beta\eta$-normal form$\}$
- Let $\mathsf{D} := (\mathsf{D}_\alpha\downarrow_{\beta\eta})_{\alpha\in\mathcal{T}}$
- Let $@^{\beta\eta}_{\gamma\delta} : \mathsf{D}_{\gamma\delta} \times \mathsf{D}_\delta \longrightarrow \mathsf{D}_\gamma$ be defined by

$$\mathsf{F}_{\gamma\delta} @^{\beta\eta}_{\gamma\delta} \mathsf{G}_\delta = (\mathsf{F}\,\mathsf{G})\downarrow$$

  for all $\mathsf{F}_{\gamma\delta} \in \mathsf{D}_{\gamma\delta}$ and $\mathsf{G}_\delta \in \mathsf{D}_\delta$.
- $@^{\beta\eta} = (@^{\beta\eta}_{\gamma\delta})_{\gamma\delta\in\mathcal{T}}$

Ex.: Interesting Applicative Structures

- Let $\mathsf{D}_\alpha\downarrow_{\beta\eta} := \{\mathsf{A}_\alpha \in \Lambda_\alpha \,|\, \mathsf{A}$ is closed and A is in $\beta\eta$-normal form$\}$
- Let $\mathsf{D} := (\mathsf{D}_\alpha\downarrow_{\beta\eta})_{\alpha\in\mathcal{T}}$
- Let $@^{\beta\eta}_{\gamma\delta} : \mathsf{D}_{\gamma\delta} \times \mathsf{D}_\delta \longrightarrow \mathsf{D}_\gamma$ be defined by

$$\mathsf{F}_{\gamma\delta} @^{\beta\eta}_{\gamma\delta} \mathsf{G}_\delta = (\mathsf{F}\,\mathsf{G})\downarrow$$

  for all $\mathsf{F}_{\gamma\delta} \in \mathsf{D}_{\gamma\delta}$ and $\mathsf{G}_\delta \in \mathsf{D}_\delta$.
- $@^{\beta\eta} = (@^{\beta\eta}_{\gamma\delta})_{\gamma\delta\in\mathcal{T}}$

Claim: If $\mathcal{C}_\iota \neq \emptyset$ and $\mathcal{C}_o \neq \emptyset$ (i.e., at least one constant for each base type is given), then $(\mathsf{D}, @^{\beta\eta})$ is an applicative structure.

Ex.: Interesting Applicative Structures

Proof:

- . . . analogous . . .

Def.: Variable Assignment

Let $\mathcal{A} := (\mathcal{D}, @)$ be an applicative structure.

Def.: Variable Assignment

Let $\mathcal{A} := (\mathcal{D}, @)$ be an applicative structure.

A typed function $\varphi\colon \mathcal{V} \longrightarrow \mathcal{D} := (\varphi_\alpha\colon \mathcal{V}_\alpha \longrightarrow \mathcal{D}_\alpha)_{\alpha\in\mathcal{T}}$ is called a variable assignment into $\mathcal{A}$.

Def.: Variable Assignment

Let $\mathcal{A} := (\mathcal{D}, @)$ be an applicative structure.

A typed function $\varphi\colon \mathcal{V} \longrightarrow \mathcal{D} := (\varphi_\alpha\colon \mathcal{V}_\alpha \longrightarrow \mathcal{D}_\alpha)_{\alpha \in \mathcal{T}}$ is called a variable assignment into $\mathcal{A}$.

Given a variable assignment φ, variable X_α, and value $\mathsf{a} \in \mathcal{D}_\alpha$,

Def.: Variable Assignment

Let $\mathcal{A} := (\mathcal{D}, @)$ be an applicative structure.

A typed function $\varphi\colon \mathcal{V} \longrightarrow \mathcal{D} := (\varphi_\alpha\colon \mathcal{V}_\alpha \longrightarrow \mathcal{D}_\alpha)_{\alpha \in \mathcal{T}}$ is called a variable assignment into $\mathcal{A}$.

Given a variable assignment φ, variable X_α, and value $\mathsf{a} \in \mathcal{D}_\alpha$, we use $\varphi, [\mathsf{a}/\mathsf{X}]$ to denote the variable assignment with

Def.: Variable Assignment

Let $\mathcal{A} := (\mathcal{D}, @)$ be an applicative structure.

A typed function $\varphi : \mathcal{V} \longrightarrow \mathcal{D} := (\varphi_\alpha : \mathcal{V}_\alpha \longrightarrow \mathcal{D}_\alpha)_{\alpha \in \mathcal{T}}$ is called a variable assignment into $\mathcal{A}$.

Given a variable assignment φ, variable X_α, and value $\mathsf{a} \in \mathcal{D}_\alpha$, we use $\varphi, [\mathsf{a}/\mathsf{X}]$ to denote the variable assignment with

$$(\varphi, [\mathsf{a}/\mathsf{X}])(\mathsf{X}) = \mathsf{a}$$

Def.: Variable Assignment

Let $\mathcal{A} := (\mathcal{D}, @)$ be an applicative structure.

A typed function $\varphi\colon \mathcal{V} \longrightarrow \mathcal{D} := (\varphi_\alpha\colon \mathcal{V}_\alpha \longrightarrow \mathcal{D}_\alpha)_{\alpha \in \mathcal{T}}$ is called a variable assignment into $\mathcal{A}$.

Given a variable assignment φ, variable X_α, and value $\mathsf{a} \in \mathcal{D}_\alpha$, we use $\varphi, [\mathsf{a}/\mathsf{X}]$ to denote the variable assignment with

$$(\varphi, [\mathsf{a}/\mathsf{X}])(\mathsf{X}) = \mathsf{a}$$

and

$$(\varphi, [\mathsf{a}/\mathsf{X}])(\mathsf{Y}) = \varphi(\mathsf{Y})$$

for variables Y other than X.

Some Assumptions

From now on, we assume the signature $\Sigma_\alpha = (\mathcal{V}, \mathcal{C})$ to be infinite for each type α.

Some Assumptions

From now on, we assume the signature $\Sigma_\alpha = (\mathcal{V}, \mathcal{C})$ to be infinite for each type α. Furthermore, we assume there is a particular cardinal $\aleph_s$ such that Σ_α has cardinality $\aleph_s$ for every type α.

Some Assumptions

From now on, we assume the signature $\Sigma_\alpha = (\mathcal{V}, \mathcal{C})$ to be infinite for each type α. Furthermore, we assume there is a particular cardinal $\aleph_s$ such that Σ_α has cardinality $\aleph_s$ for every type α. Since $\mathcal{V}$ is countable, this implies $\mathit{wff}_\alpha(\Sigma) := \Lambda^\alpha$ and $\mathit{cwff}_\alpha(\Sigma) := \{\mathsf{A} \in \Lambda^\alpha | \mathsf{A}\ \text{closed}\}$ have cardinality $\aleph_s$ for each type α.

Some Assumptions

From now on, we assume the signature $\Sigma_\alpha = (\mathcal{V}, \mathcal{C})$ to be infinite for each type α. Furthermore, we assume there is a particular cardinal $\aleph_s$ such that Σ_α has cardinality $\aleph_s$ for every type α. Since $\mathcal{V}$ is countable, this implies $\mathit{wff}_\alpha(\Sigma) := \Lambda^\alpha$ and $\mathit{cwff}_\alpha(\Sigma) := \{A \in \Lambda^\alpha | A \text{ closed}\}$ have cardinality $\aleph_s$ for each type α. Also, whether or not primitive equality is included in the signature, there can only be finitely many logical constants in Σ_α for each particular type α.

Some Assumptions

From now on, we assume the signature $\Sigma_\alpha = (\mathcal{V}, \mathcal{C})$ to be infinite for each type α. Furthermore, we assume there is a particular cardinal $\aleph_s$ such that Σ_α has cardinality $\aleph_s$ for every type α. Since $\mathcal{V}$ is countable, this implies $\mathit{wff}_\alpha(\Sigma) := \Lambda^\alpha$ and $\mathit{cwff}_\alpha(\Sigma) := \{A \in \Lambda^\alpha | A \text{ closed}\}$ have cardinality $\aleph_s$ for each type α. Also, whether or not primitive equality is included in the signature, there can only be finitely many logical constants in Σ_α for each particular type α. Thus, the cardinality of the set of parameters in Σ_α is also $\aleph_s$. In the countable case, $\aleph_s$ is $\aleph_0$.

Σ-Evaluations

Let Σ be a signature.

Σ-Evaluations

Let Σ be a signature. We build on the notion of applicative structures to define Σ-evaluations, where the evaluation function is assumed to respect application and β-conversion.

Σ-Evaluations

Let Σ be a signature. We build on the notion of applicative structures to define Σ-evaluations, where the evaluation function is assumed to respect application and β-conversion.

In such models, a function is not uniquely determined by its behavior on all possible arguments.

Σ-Evaluations

Let Σ be a signature. We build on the notion of applicative structures to define Σ-evaluations, where the evaluation function is assumed to respect application and β-conversion.

In such models, a function is not uniquely determined by its behavior on all possible arguments.

Such models can be constructed, for example, by labeling for functions (e.g., a green and a red version of a function f) in order to differentiate between them, even though they are functionally equivalent.

Σ-Evaluations

Let $\mathcal{E}\colon \mathcal{F}_{\mathcal{T}}(\mathcal{V};\mathcal{D}) \longrightarrow \mathcal{F}_{\mathcal{T}}(\textit{wff}(\Sigma),\mathcal{D})$ be a total function, where $\mathcal{F}_{\mathcal{T}}(\mathcal{V};\mathcal{D})$ is the set of variable assignments and $\mathcal{F}_{\mathcal{T}}(\textit{wff}(\Sigma),\mathcal{D})$ is the set of typed functions mapping terms into objects in $\mathcal{D}$.

Σ-Evaluations

Let $\mathcal{E}\colon \mathcal{F}_{\mathcal{T}}(\mathcal{V};\mathcal{D}) \longrightarrow \mathcal{F}_{\mathcal{T}}(\mathit{wff}(\Sigma),\mathcal{D})$ be a total function, where $\mathcal{F}_{\mathcal{T}}(\mathcal{V};\mathcal{D})$ is the set of variable assignments and $\mathcal{F}_{\mathcal{T}}(\mathit{wff}(\Sigma),\mathcal{D})$ is the set of typed functions mapping terms into objects in $\mathcal{D}$. We will write the argument of $\mathcal{E}$ as a subscript. So, for each assignment φ, we have a typed function

$$\mathcal{E}_\varphi\colon \mathit{wff}(\Sigma) \longrightarrow \mathcal{D}$$

Σ-Evaluations

Let $\mathcal{E}\colon \mathcal{F}_{\mathcal{T}}(\mathcal{V};\mathcal{D}) \longrightarrow \mathcal{F}_{\mathcal{T}}(\mathit{wff}(\Sigma),\mathcal{D})$ be a total function, where $\mathcal{F}_{\mathcal{T}}(\mathcal{V};\mathcal{D})$ is the set of variable assignments and $\mathcal{F}_{\mathcal{T}}(\mathit{wff}(\Sigma),\mathcal{D})$ is the set of typed functions mapping terms into objects in $\mathcal{D}$. We will write the argument of $\mathcal{E}$ as a subscript. So, for each assignment φ, we have a typed function

$$\mathcal{E}_\varphi\colon \mathit{wff}(\Sigma) \longrightarrow \mathcal{D}$$

What properties shall $\mathcal{E}$ fulfill?

Def.: Evaluation Function

$\mathcal{E}$ is called an evaluation function for an applicative structure $\mathcal{A} = (\mathcal{D}, @)$

Def.: Evaluation Function

$\mathcal{E}$ is called an evaluation function for an applicative structure $\mathcal{A} = (\mathcal{D}, @)$ if for any assignments φ and ψ into $\mathcal{A}$, we have

Def.: Evaluation Function

$\mathcal{E}$ is called an evaluation function for an applicative structure $\mathcal{A} = (\mathcal{D}, @)$ if for any assignments φ and ψ into $\mathcal{A}$, we have

1. $\mathcal{E}_\varphi|_\mathcal{V} = \varphi$

Def.: Evaluation Function

$\mathcal{E}$ is called an evaluation function for an applicative structure $\mathcal{A} = (\mathcal{D}, @)$ if for any assignments φ and ψ into $\mathcal{A}$, we have

1. $\mathcal{E}_\varphi|_\mathcal{V} = \varphi$
2. $\mathcal{E}_\varphi(\mathbf{FA}) = \mathcal{E}_\varphi(\mathbf{F})@\mathcal{E}_\varphi(\mathbf{A})$ for any $\mathbf{F} \in \mathit{wff}_{\alpha\to\beta}(\Sigma)$ and $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$ and types α and β.

Def.: Evaluation Function

$\mathcal{E}$ is called an evaluation function for an applicative structure $\mathcal{A} = (\mathcal{D}, @)$ if for any assignments φ and ψ into $\mathcal{A}$, we have

1. $\mathcal{E}_\varphi|_\mathcal{V} = \varphi$
2. $\mathcal{E}_\varphi(\mathbf{FA}) = \mathcal{E}_\varphi(\mathbf{F})@\mathcal{E}_\varphi(\mathbf{A})$ for any $\mathbf{F} \in \mathit{wff}_{\alpha\rightarrow\beta}(\Sigma)$ and $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$ and types α and β.
3. $\mathcal{E}_\varphi(\mathbf{A}) = \mathcal{E}_\psi(\mathbf{A})$ for any type α and $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$, whenever φ and ψ coincide on $\mathrm{FV}(\mathbf{A})$.

Def.: Evaluation Function

$\mathcal{E}$ is called an evaluation function for an applicative structure $\mathcal{A} = (\mathcal{D}, @)$ if for any assignments φ and ψ into $\mathcal{A}$, we have

1. $\mathcal{E}_\varphi|_\mathcal{V} = \varphi$
2. $\mathcal{E}_\varphi(\mathbf{FA}) = \mathcal{E}_\varphi(\mathbf{F})@\mathcal{E}_\varphi(\mathbf{A})$ for any $\mathbf{F} \in \mathit{wff}_{\alpha\rightarrow\beta}(\Sigma)$ and $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$ and types α and β.
3. $\mathcal{E}_\varphi(\mathbf{A}) = \mathcal{E}_\psi(\mathbf{A})$ for any type α and $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$, whenever φ and ψ coincide on $\mathrm{FV}(\mathbf{A})$.
4. $\mathcal{E}_\varphi(\mathbf{A}) = \mathcal{E}_\varphi(\mathbf{A}{\downarrow_\beta})$ for all $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$.

Def.: Σ-Evaluation

We call $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ a Σ-evaluation if $(\mathcal{D}, @)$ is an applicative structure and $\mathcal{E}$ is an evaluation function for $(\mathcal{D}, @)$. We call $\mathcal{E}_\varphi(\mathbf{A}_\alpha) \in \mathcal{D}_\alpha$ the denotation of $\mathbf{A}_\alpha$ in $\mathcal{J}$ for φ.

Def.: Σ-Evaluation

We call $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ a Σ-evaluation if $(\mathcal{D}, @)$ is an applicative structure and $\mathcal{E}$ is an evaluation function for $(\mathcal{D}, @)$. We call $\mathcal{E}_\varphi(\mathbf{A}_\alpha) \in \mathcal{D}_\alpha$ the denotation of $\mathbf{A}_\alpha$ in $\mathcal{J}$ for φ.

Remark: since $\mathcal{E}$ is a function, the denotation in $\mathcal{J}$ is unique. However, for a given applicative structure $\mathcal{A}$, there may be many possible evaluation functions.

Def.: Σ-Evaluation

We call $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ a Σ-evaluation if $(\mathcal{D}, @)$ is an applicative structure and $\mathcal{E}$ is an evaluation function for $(\mathcal{D}, @)$. We call $\mathcal{E}_\varphi(\mathbf{A}_\alpha) \in \mathcal{D}_\alpha$ the denotation of $\mathbf{A}_\alpha$ in $\mathcal{J}$ for φ.

Remark: since $\mathcal{E}$ is a function, the denotation in $\mathcal{J}$ is unique. However, for a given applicative structure $\mathcal{A}$, there may be many possible evaluation functions.

If $\mathbf{A}$ is a closed formula, then $\mathcal{E}_\varphi(\mathbf{A})$ is independent of φ, since $\mathbf{Free}(\mathbf{A}) = \emptyset$. In these cases we sometimes drop the reference to φ from $\mathcal{E}_\varphi(\mathbf{A})$ and simply write $\mathcal{E}(\mathbf{A})$.

Def.: Functional/Full/Standard Σ-Eval.

We call a Σ-evaluation $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ functional [full, standard] if the applicative structure $(\mathcal{D}, @)$ is functional [full, standard].

Def.: Functional/Full/Standard Σ-Eval.

We call a Σ-evaluation $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ functional [full, standard] if the applicative structure $(\mathcal{D}, @)$ is functional [full, standard].

We say $\mathcal{J}$ is a Σ-evaluation over a frame if $(\mathcal{D}, @)$ is a frame.

What is the Idea?

Σ-evaluations generalize Σ-evaluations over frames, which are the basis for Henkin models, to the non-functional case.

What is the Idea?

Σ-evaluations generalize Σ-evaluations over frames, which are the basis for Henkin models, to the non-functional case.

The existence of an evaluation function that meets the conditions as presented seems to be the weakest situation where one would like to speak of a model.

What is the Idea?

Σ-evaluations generalize Σ-evaluations over frames, which are the basis for Henkin models, to the non-functional case.

The existence of an evaluation function that meets the conditions as presented seems to be the weakest situation where one would like to speak of a model.

We cannot in general assume the evaluation function is uniquely determined by its values on constants as this requires functionality.

What is the Idea?

Σ-evaluations generalize Σ-evaluations over frames, which are the basis for Henkin models, to the non-functional case.

The existence of an evaluation function that meets the conditions as presented seems to be the weakest situation where one would like to speak of a model.

We cannot in general assume the evaluation function is uniquely determined by its values on constants as this requires functionality. Example: two evaluation functions $\mathcal{E}$ and $\mathcal{E}'$ on the same applicative structure may agree on all constants, but give a different value to the term $(\lambda X_\iota.X)$.

Lemma: Σ-Evaluations respect β-Equality

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation and $\mathbf{A} =_\beta \mathbf{B}$. For all assignments φ into $(\mathcal{D}, @)$, we have

.

Lemma: Σ-Evaluations respect β-Equality

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation and $\mathbf{A} =_\beta \mathbf{B}$. For all assignments φ into $(\mathcal{D}, @)$, we have

$$\mathcal{E}_\varphi(\mathbf{A}) = \qquad\qquad\qquad = \mathcal{E}_\varphi(\mathbf{B})$$

.

Lemma: Σ-Evaluations respect β-Equality

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation and $\mathbf{A} =_\beta \mathbf{B}$. For all assignments φ into $(\mathcal{D}, @)$, we have

$$\mathcal{E}_\varphi(\mathbf{A}) = \mathcal{E}_\varphi(\mathbf{A}{\downarrow_\beta}) \quad \mathcal{E}_\varphi(\mathbf{B}{\downarrow_\beta}) = \mathcal{E}_\varphi(\mathbf{B})$$

.

Lemma: Σ-Evaluations respect β-Equality

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation and $\mathbf{A} =_\beta \mathbf{B}$. For all assignments φ into $(\mathcal{D}, @)$, we have

$$\mathcal{E}_\varphi(\mathbf{A}) = \mathcal{E}_\varphi(\mathbf{A}{\downarrow_\beta}) = \mathcal{E}_\varphi(\mathbf{B}{\downarrow_\beta}) = \mathcal{E}_\varphi(\mathbf{B})$$

.

Thm.: Substitution-Value Lemma

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation and φ be an assignment into $\mathcal{J}$.

Thm.: Substitution-Value Lemma

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation and φ be an assignment into $\mathcal{J}$. For any types α and β, variables X_β, and formulae $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$ and $\mathbf{B} \in \mathit{wff}_\beta(\Sigma)$, we have

Thm.: Substitution-Value Lemma

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation and φ be an assignment into $\mathcal{J}$. For any types α and β, variables X_β, and formulae $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$ and $\mathbf{B} \in \mathit{wff}_\beta(\Sigma)$, we have

$$\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\mathbf{A}) = \mathcal{E}_\varphi([\mathbf{B}/\mathsf{X}]\mathbf{A})$$

.

Prf.: Substitution-Value Lemma

Proof:

Prf.: Substitution-Value Lemma

Proof: Using the fact that $\mathcal{E}$ respects β-equality and the other properties of $\mathcal{E}$, we can compute

Prf.: Substitution-Value Lemma

Proof: Using the fact that $\mathcal{E}$ respects β-equality and the other properties of $\mathcal{E}$, we can compute

$$\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathrm{X}]}(\mathbf{A}) \quad =$$

.

Prf.: Substitution-Value Lemma

Proof: Using the fact that $\mathcal{E}$ respects β-equality and the other properties of $\mathcal{E}$, we can compute

$$\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathrm{X}]}(\mathbf{A}) \quad = \quad \mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathrm{X}]}((\lambda \mathrm{X}.\mathbf{A})\mathrm{X})$$

.

Prf.: Substitution-Value Lemma

Proof: Using the fact that $\mathcal{E}$ respects β-equality and the other properties of $\mathcal{E}$, we can compute

$$\begin{aligned}
\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\mathbf{A}) &= \mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}((\lambda \mathsf{X}.\mathbf{A})\mathsf{X}) \\
&= \mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\lambda \mathsf{X}.\mathbf{A})@\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\mathsf{X})
\end{aligned}$$

.

Prf.: Substitution-Value Lemma

Proof: Using the fact that $\mathcal{E}$ respects β-equality and the other properties of $\mathcal{E}$, we can compute

$$\begin{aligned}
\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\mathbf{A}) &= \mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}((\lambda \mathsf{X}\text{.}\mathbf{A})\mathsf{X}) \\
&= \mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\lambda \mathsf{X}\text{.}\mathbf{A})@\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\mathsf{X}) \\
&= \mathcal{E}_\varphi(\lambda \mathsf{X}\text{.}\mathbf{A})@\mathcal{E}_\varphi(\mathbf{B})
\end{aligned}$$

.

Prf.: Substitution-Value Lemma

Proof: Using the fact that $\mathcal{E}$ respects β-equality and the other properties of $\mathcal{E}$, we can compute

$$\begin{aligned}
\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\mathbf{A}) &= \mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}((\lambda \mathsf{X}.\mathbf{A})\mathsf{X}) \\
&= \mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\lambda \mathsf{X}.\mathbf{A})@\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\mathsf{X}) \\
&= \mathcal{E}_\varphi(\lambda \mathsf{X}.\mathbf{A})@\mathcal{E}_\varphi(\mathbf{B}) \\
&= \mathcal{E}_\varphi((\lambda \mathsf{X}.\mathbf{A})\mathbf{B})
\end{aligned}$$

.

Prf.: Substitution-Value Lemma

Proof: Using the fact that $\mathcal{E}$ respects β-equality and the other properties of $\mathcal{E}$, we can compute

$$\begin{aligned}
\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\mathbf{A}) &= \mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}((\lambda \mathsf{X}_{\bullet}\mathbf{A})\mathsf{X}) \\
&= \mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\lambda \mathsf{X}_{\bullet}\mathbf{A})@\mathcal{E}_{\varphi,[\mathcal{E}_\varphi(\mathbf{B})/\mathsf{X}]}(\mathsf{X}) \\
&= \mathcal{E}_\varphi(\lambda \mathsf{X}_{\bullet}\mathbf{A})@\mathcal{E}_\varphi(\mathbf{B}) \\
&= \mathcal{E}_\varphi((\lambda \mathsf{X}_{\bullet}\mathbf{A})\mathbf{B}) \\
&= \mathcal{E}_\varphi([\mathbf{B}/\mathsf{X}]\mathbf{A}).
\end{aligned}$$

Weaker Notions of Functionality

We will consider two weaker notions of functionality. These forms are often discussed in the literature (cf. [HindleySeldin86]).

Weaker Notions of Functionality

We will consider two weaker notions of functionality. These forms are often discussed in the literature (cf. [HindleySeldin86]).

- η-functionality simply means the evaluation respects η-conversion.

Weaker Notions of Functionality

We will consider two weaker notions of functionality. These forms are often discussed in the literature (cf. [HindleySeldin86]).

- η-functionality simply means the evaluation respects η-conversion.
- ξ-functionality means we have functionality (only) with respect to λ-abstractions.

Def.: η-Functional

Let $\mathcal{J} = (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation.

Def.: η-Functional

Let $\mathcal{J} = (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation.
We say $\mathcal{J}$ is η-functional if

Def.: η-Functional

Let $\mathcal{J} = (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation.
We say $\mathcal{J}$ is η-functional if

$$\mathcal{E}_\varphi(\mathbf{A}) = \mathcal{E}_\varphi(\mathbf{A}{\downarrow_{\beta\eta}})$$

for any type α, formula $\mathbf{A} \in \mathit{wff}_\alpha(\Sigma)$, and assignment φ.

Def.: ξ-Functional

Let $\mathcal{J} = (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation.

Def.: ξ-Functional

Let $\mathcal{J} = (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation. We say $\mathcal{J}$ is ξ-functional if

Def.: ξ-Functional

Let $\mathcal{J} = (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation. We say $\mathcal{J}$ is ξ-functional if for all $\alpha, \beta \in \mathcal{T}$, $\mathbf{M}, \mathbf{N} \in wff_\beta(\Sigma)$, assignments φ, and variables X_α,

Def.: ξ-Functional

Let $\mathcal{J} = (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation. We say $\mathcal{J}$ is ξ-functional if for all $\alpha, \beta \in \mathcal{T}$, $\mathbf{M}, \mathbf{N} \in wff_\beta(\Sigma)$, assignments φ, and variables X_α,

$$\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha . \mathbf{M}_\beta) = \mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha . \mathbf{N}_\beta)$$

Def.: ξ-Functional

Let $\mathcal{J} = (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation. We say $\mathcal{J}$ is ξ-functional if for all $\alpha, \beta \in \mathcal{T}$, $\mathbf{M}, \mathbf{N} \in wff_\beta(\Sigma)$, assignments φ, and variables X_α,

$$\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha. \mathbf{M}_\beta) = \mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha. \mathbf{N}_\beta)$$

whenever

$$\mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{M}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{N})$$

for every $\mathsf{a} \in \mathcal{D}_\alpha$.

Lemma: Functionality and η

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a functional Σ-evaluation.

Lemma: Functionality and η

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a functional Σ-evaluation.

1. For any assignment φ into $\mathcal{J}$ and $\mathbf{F} \in \mathit{wff}_{\alpha\to\beta}(\Sigma)$ where $\mathrm{X}_\alpha \notin \mathbf{Free}(\mathbf{F})$, we have

Lemma: Functionality and η

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a functional Σ-evaluation.

1. For any assignment φ into $\mathcal{J}$ and $\mathbf{F} \in \mathit{wff}_{\alpha\to\beta}(\Sigma)$ where $\mathsf{X}_\alpha \notin \mathbf{Free}(\mathbf{F})$, we have

$$\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha \mathbf{.} \mathbf{F}\mathsf{X}) = \mathcal{E}_\varphi(\mathbf{F})$$

Lemma: Functionality and η

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a functional Σ-evaluation.

1. For any assignment φ into $\mathcal{J}$ and $\mathbf{F} \in \textit{wff}_{\alpha \to \beta}(\Sigma)$ where $\mathsf{X}_\alpha \notin \mathbf{Free}(\mathbf{F})$, we have

$$\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha \mathbf{.} \mathbf{F}\mathsf{X}) = \mathcal{E}_\varphi(\mathbf{F})$$

2. If a formula $\mathbf{A}$ η-reduces to $\mathbf{B}$ in one step, then for any assignment φ into $\mathcal{J}$, we have

Lemma: Functionality and η

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a functional Σ-evaluation.

1. For any assignment φ into $\mathcal{J}$ and $\mathbf{F} \in \mathit{wff}_{\alpha\to\beta}(\Sigma)$ where $\mathsf{X}_\alpha \notin \mathbf{Free}(\mathbf{F})$, we have

$$\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha \mathbf{.} \mathbf{F}\mathsf{X}) = \mathcal{E}_\varphi(\mathbf{F})$$

2. If a formula $\mathbf{A}$ η-reduces to $\mathbf{B}$ in one step, then for any assignment φ into $\mathcal{J}$, we have

$$\mathcal{E}_\varphi(\mathbf{A}) = \mathcal{E}_\varphi(\mathbf{B})$$

Lemma: Functionality and η

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a functional Σ-evaluation.

1. For any assignment φ into $\mathcal{J}$ and $\mathbf{F} \in wff_{\alpha\rightarrow\beta}(\Sigma)$ where $\mathsf{X}_\alpha \notin \mathbf{Free}(\mathbf{F})$, we have

$$\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha.\mathbf{F}\mathsf{X}) = \mathcal{E}_\varphi(\mathbf{F})$$

2. If a formula $\mathbf{A}$ η-reduces to $\mathbf{B}$ in one step, then for any assignment φ into $\mathcal{J}$, we have

$$\mathcal{E}_\varphi(\mathbf{A}) = \mathcal{E}_\varphi(\mathbf{B})$$

Proof: Exercise

Lemma: Functionality and η+ξ

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation.

Lemma: Functionality and η+ξ

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation. Then $\mathcal{J}$ is functional iff it is both η-functional and ξ-functional.

Lemma: Functionality and η+ξ

Let $\mathcal{J} := (\mathcal{D}, @, \mathcal{E})$ be a Σ-evaluation. Then $\mathcal{J}$ is functional iff it is both η-functional and ξ-functional.

Proof: Exercise

Logical Constants in Signature

Let $\Sigma := (\mathcal{V}, \mathcal{C})$ be a signature.

Logical Constants in Signature

Let $\Sigma := (\mathcal{V}, \mathcal{C})$ be a signature.

The following logical constants may or may not be in the set $\mathcal{C}$ of constants:

Logical Constants in Signature

Let $\Sigma := (\mathcal{V}, \mathcal{C})$ be a signature.

The following logical constants may or may not be in the set $\mathcal{C}$ of constants:

$$\top_o, \ \bot_o, \ \neg_{oo}, \ \vee_{ooo}, \ \wedge_{ooo}, \ \supset_{ooo}, \ \Leftrightarrow_{ooo}$$

Logical Constants in Signature

Let $\Sigma := (\mathcal{V}, \mathcal{C})$ be a signature.

The following logical constants may or may not be in the set $\mathcal{C}$ of constants:

$$\top_{o},\ \bot_{o},\ \neg_{oo},\ \vee_{ooo},\ \wedge_{ooo},\ \supset_{ooo},\ \Leftrightarrow_{ooo}$$

$$\Pi^{\alpha}_{o(o\alpha)}(\Pi^{\alpha}F_{o\alpha} \sim \forall x_{\alpha}Fx),\ \Sigma^{\alpha}_{o(o\alpha)}(\Sigma^{\alpha}F_{o\alpha} \sim \exists x_{\alpha}Fx)$$

Logical Constants in Signature

Let $\Sigma := (\mathcal{V}, \mathcal{C})$ be a signature.

The following logical constants may or may not be in the set $\mathcal{C}$ of constants:

$$\top_o, \ \bot_o, \ \neg_{oo}, \ \vee_{ooo}, \ \wedge_{ooo}, \ \supset_{ooo}, \ \Leftrightarrow_{ooo}$$

$$\Pi^{\alpha}_{o(o\alpha)}(\Pi^{\alpha}F_{o\alpha} \ \sim \ \forall x_{\alpha}Fx), \ \Sigma^{\alpha}_{o(o\alpha)}(\Sigma^{\alpha}F_{o\alpha} \ \sim \ \exists x_{\alpha}Fx)$$

$$=^{\alpha}_{o\alpha\alpha}$$

Logical Constants in Signature

Let $\Sigma := (\mathcal{V}, \mathcal{C})$ be a signature.

The following logical constants may or may not be in the set $\mathcal{C}$ of constants:

$$\top_{\mathsf{o}},\ \bot_{\mathsf{o}},\ \neg_{\mathsf{oo}},\ \vee_{\mathsf{ooo}},\ \wedge_{\mathsf{ooo}},\ \supset_{\mathsf{ooo}},\ \Leftrightarrow_{\mathsf{ooo}}$$

$$\Pi^{\alpha}_{\mathsf{o}(\mathsf{o}\alpha)}(\Pi^{\alpha}\mathsf{F}_{\mathsf{o}\alpha} \sim \forall \mathsf{x}_{\alpha}\mathsf{F}\mathsf{x}),\ \Sigma^{\alpha}_{\mathsf{o}(\mathsf{o}\alpha)}(\Sigma^{\alpha}\mathsf{F}_{\mathsf{o}\alpha} \sim \exists \mathsf{x}_{\alpha}\mathsf{F}\mathsf{x})$$

$$=^{\alpha}_{\mathsf{o}\alpha\alpha}$$

for all $\alpha \in \mathcal{T}$

Once More: Cantor’s Theorem

For any set A,

$$|A| < |\mathcal{P}(A)|$$

Once More: Cantor's Theorem

For any set A,

$$|\mathsf{A}| < |\mathcal{P}(\mathsf{A})|$$

i.e., $\neg\exists g : \mathsf{A} \longrightarrow \mathcal{P}(\mathsf{A})$ with g surjective.

Once More: Cantor's Theorem

Assume the set A is associated with ι.

Once More: Cantor's Theorem

Assume the set A is associated with ι. Then $\mathcal{P}(A)$ has type $o\iota$, i.e. the type of "sets" (or characteristic functions) over ι.

Once More: Cantor's Theorem

Assume the set A is associated with ι. Then $\mathcal{P}(A)$ has type $o\iota$, i.e. the type of "sets" (or characteristic functions) over ι.

$$D_{o\iota}$$

Once More: Cantor's Theorem

Assume the set A is associated with ι. Then $\mathcal{P}(\mathsf{A})$ has type $\mathsf{o}\iota$, i.e. the type of "sets" (or characteristic functions) over ι.

$$\mathsf{D}_{\mathsf{o}\iota} = \mathsf{D}_{\mathsf{o}}^{\mathsf{D}_{\iota}}$$

Once More: Cantor's Theorem

Assume the set A is associated with ι. Then $\mathcal{P}(\mathsf{A})$ has type $o\iota$, i.e. the type of "sets" (or characteristic functions) over ι.

$$\begin{aligned} \mathsf{D}_{o\iota} &= \mathsf{D}_o^{\mathsf{D}_\iota} \\ &= \{\bot, \top\}^{\mathsf{D}_\iota} \end{aligned}$$

Once More: Cantor's Theorem

Assume the set A is associated with ι. Then $\mathcal{P}(A)$ has type $o\iota$, i.e. the type of "sets" (or characteristic functions) over ι.

$$\begin{aligned} D_{o\iota} &= D_o^{D_\iota} \\ &= \{\bot, \top\}^{D_\iota} \\ &= \{f |\ f : D_\iota \to \{\bot, \top\}\ \} \end{aligned}$$

Once More: Cantor's Theorem

Assume the set A is associated with ι. Then $\mathcal{P}(A)$ has type $o\iota$, i.e. the type of "sets" (or characteristic functions) over ι.

$$\begin{aligned} D_{o\iota} &= D_o^{D_\iota} \\ &= \{\perp, \top\}^{D_\iota} \\ &= \{f \mid f : D_\iota \longrightarrow \{\perp, \top\}\ \} \\ &\cong \{\ X \mid X \subseteq D_\iota\ \} \end{aligned}$$

Once More: Cantor's Theorem

Assume the set A is associated with ι. Then $\mathcal{P}(A)$ has type $o\iota$, i.e. the type of "sets" (or characteristic functions) over ι.

$$
\begin{aligned}
D_{o\iota} &= D_o^{D_\iota} \\
&= \{\bot, \top\}^{D_\iota} \\
&= \{f \mid f : D_\iota \rightarrow \{\bot, \top\}\} \\
&\cong \{X \mid X \subseteq D_\iota\} \\
&= \mathcal{P}(D_\iota)
\end{aligned}
$$

Once More: Cantor's Theorem

We can now formulate Cantor's Theorem with typed terms (as seen before):

Once More: Cantor's Theorem

We can now formulate Cantor's Theorem with typed terms (as seen before):

$$\neg\exists g_{o\iota\iota}\forall f_{o\iota}\exists x_{\iota} : gx = f$$

Once More: Cantor's Theorem

We can now formulate Cantor's Theorem with typed terms (as seen before):

$$\neg\exists g_{o\iota\iota} \forall f_{o\iota} \exists x_\iota : gx = f$$

which is shorthand for:

Once More: Cantor's Theorem

We can now formulate Cantor's Theorem with typed terms (as seen before):

$$\neg \exists g_{o\iota\iota} \forall f_{o\iota} \exists x_\iota : gx = f$$

which is shorthand for:

$$\neg_{oo} \Sigma^{o\iota\iota}_{o(o(o\iota\iota))} \left(\lambda g_{o\iota\iota}. \Pi^{o\iota}_{o(o(o\iota))} \left(\lambda f_{o\iota}. \Sigma^{\iota}_{o(o\iota)} \left(\lambda x_\iota. =^{o\iota}_{o(o\iota)(o\iota)} (gx)\, f\right)\right)\right)$$

Once More: Cantor's Theorem

We can now formulate Cantor's Theorem with typed terms (as seen before):

$$\neg \exists g_{o\iota\iota} \forall f_{o\iota} \exists x_\iota : gx = f$$

which is shorthand for:

$$\neg_{oo} \Sigma^{o\iota\iota}_{o(o(o\iota\iota))} \left(\lambda g_{o\iota\iota} . \Pi^{o\iota}_{o(o(o\iota))} \left(\lambda f_{o\iota} . \Sigma^{\iota}_{o(o\iota)} \left(\lambda x_\iota . =^{o\iota}_{o(o\iota)(o\iota)} (gx)\, f \right) \right) \right)$$

Note: for this term to be in the set $cwff_\alpha(\Sigma)$, the constants $\neg_{oo}$, $\Sigma^{o\iota\iota}_{o(o(o\iota\iota))}$, $\Pi^{o\iota}_{o(o(o\iota))}$, Σ^{ι} and $=^{o\iota}$ have to be in the set $\mathcal{C}$.

Once More: Cantor's Theorem

Proof:

Once More: Cantor's Theorem

Proof: Assume such a function g exists.

Once More: Cantor's Theorem

Proof: Assume such a function g exists.

Let $f = \{x \mid x \notin gx\}$ that is $f = (\lambda x_\iota . \neg gxx)$.

Once More: Cantor's Theorem

Proof: Assume such a function g exists.

Let $f = \{x \mid x \notin gx\}$ that is $f = (\lambda x_\iota . \neg gxx)$.

g is surjective,

Once More: Cantor's Theorem

Proof: Assume such a function g exists.
Let $\mathsf{f} = \{\mathsf{x} \mid \mathsf{x} \notin \mathsf{gx}\}$ that is $\mathsf{f} = (\lambda \mathsf{x}_\iota . \neg \mathsf{gxx})$.
g is surjective, hence

$$(\exists \mathsf{y}_\iota : \mathsf{gy} = [\lambda \mathsf{x} . \neg \mathsf{gxx}])$$

Once More: Cantor's Theorem

Proof: Assume such a function g exists.
Let $f = \{x \mid x \notin gx\}$ that is $f = (\lambda x_\iota . \neg gxx)$.
g is surjective, hence

$$(\exists y_\iota : gy = [\lambda x . \neg gxx])$$

hence

$$(gyy \Leftrightarrow \neg gyy)$$

Once More: Cantor's Theorem

Proof: Assume such a function g exists.
Let $f = \{x \mid x \notin gx\}$ that is $f = (\lambda x_\iota . \neg gxx)$.
g is surjective, hence

$$(\exists y_\iota : gy = [\lambda x . \neg gxx])$$

hence

$$(gyy \Leftrightarrow \neg gyy)$$

Contradiction!

Once More: Cantor's Theorem

Proof: Assume such a function g exists.
Let $f = \{x \mid x \notin gx\}$ that is $f = (\lambda x_\iota . \neg gxx)$.
g is surjective, hence

$$(\exists y_\iota : gy = [\lambda x . \neg gxx])$$

hence

$$(gyy \Leftrightarrow \neg gyy)$$

Contradiction!

Note that the proof uses $\neg$.

Semantics: Σ-Models

Def.: Properties of Logical Constants

Let $(\mathrm{D}, @)$ be an applicative structure and let $\mathtt{v} : \mathcal{D}_{\mathsf{o}} \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$).

Def.: Properties of Logical Constants

Let $(\mathsf{D}, @)$ be an applicative structure and let $\mathsf{v} : \mathcal{D}_{\mathsf{o}} \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $\mathsf{a} \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_{\mathsf{c}}(()\mathsf{a})$ with respect to v:

Def.: Properties of Logical Constants

Let $(\mathcal{D}, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c((\,)a)$ with respect to v:

c		

Def.: Properties of Logical Constants

Let $(\mathsf{D}, @)$ be an applicative structure and let $\mathsf{v} : \mathcal{D}_{\mathsf{o}} \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $\mathsf{a} \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_{\mathsf{c}}(()\mathsf{a})$ with respect to v:

c	β	

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when

Def.: Properties of Logical Constants

Let $(\mathcal{D}, @)$ be an applicative structure and let $v : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$		

Def.: Properties of Logical Constants

Let $(\mathsf{D}, @)$ be an applicative structure and let $\mathsf{v} : \mathcal{D}_o \rightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $\mathsf{a} \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_\mathsf{c}(()\mathsf{a})$ with respect to v:

c	β	$\mathfrak{L}_\mathsf{c}(()\mathsf{a})$ holds when
$\top$	o	

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$		

Def.: Properties of Logical Constants

Let $(\mathcal{D}, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$		

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\perp$	o	$v(a) = \mathtt{F}$
$\neg$	oo	

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$	oo	$v(a@b) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$	oo	$v(a@b) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$	oo	$v(a@b) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$	oo	$v(a@b) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$	oo	$v(a@b) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$		

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$	oo	$v(a@b) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$	oo	$v(a@b) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$			

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$	oo	$v(a@b) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when
$\top$	o	$v(a) = \mathtt{T}$
$\bot$	o	$v(a) = \mathtt{F}$
$\neg$	oo	$v(a@b) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$ iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \to \{T, F\}$ be a function (for given $T \neq F$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = T$	
$\bot$	o	$v(a) = F$	
$\neg$	oo	$v(a@b) = T$	iff $v(b) = F$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = T$	iff $v(b) = T$ or $v(c) = T$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = T$	iff $v(b) = T$ and $v(c) = T$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = T$	iff $v(b) = F$ or $v(c) = T$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$			

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v: \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$	ooo		

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = v(c)$ $\forall b, c \in \mathcal{D}_o$

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = v(c)$ $\forall b, c \in \mathcal{D}_o$
$=^\alpha$			

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = v(c)$ $\forall b, c \in \mathcal{D}_o$
$=^\alpha$	$o\alpha\alpha$		

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v: \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = v(c)$ $\forall b, c \in \mathcal{D}_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = \mathtt{T}$	iff $b = c$ $\forall b, c \in \mathcal{D}_o$

Def.: Properties of Logical Constants

Let $(\mathsf{D}, @)$ be an applicative structure and let $\mathsf{v} : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $\mathsf{a} \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_\mathsf{c}(()\mathsf{a})$ with respect to v:

c	β	$\mathfrak{L}_\mathsf{c}(()\mathsf{a})$ holds when	
$\top$	o	$\mathsf{v}(\mathsf{a}) = \mathtt{T}$	
$\bot$	o	$\mathsf{v}(\mathsf{a}) = \mathtt{F}$	
$\neg$	oo	$\mathsf{v}(\mathsf{a}@\mathsf{b}) = \mathtt{T}$	iff $\mathsf{v}(\mathsf{b}) = \mathtt{F}$ $\forall \mathsf{b} \in \mathcal{D}_o$
$\vee$	ooo	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$	iff $\mathsf{v}(\mathsf{b}) = \mathtt{T}$ or $\mathsf{v}(\mathsf{c}) = \mathtt{T}$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
$\wedge$	ooo	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$	iff $\mathsf{v}(\mathsf{b}) = \mathtt{T}$ and $\mathsf{v}(\mathsf{c}) = \mathtt{T}$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
$\supset$	ooo	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$	iff $\mathsf{v}(\mathsf{b}) = \mathtt{F}$ or $\mathsf{v}(\mathsf{c}) = \mathtt{T}$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$	iff $\mathsf{v}(\mathsf{b}) = \mathsf{v}(\mathsf{c})$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
$=^\alpha$	$o\alpha\alpha$	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$	iff $\mathsf{b} = \mathsf{c}$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
Π^α			

Def.: Properties of Logical Constants

Let $(\mathsf{D}, @)$ be an applicative structure and let $\mathsf{v} : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $\mathsf{a} \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_\mathsf{c}(()\mathsf{a})$ with respect to v:

c	β	$\mathfrak{L}_\mathsf{c}(()\mathsf{a})$ holds when
$\top$	o	$\mathsf{v}(\mathsf{a}) = \mathtt{T}$
$\bot$	o	$\mathsf{v}(\mathsf{a}) = \mathtt{F}$
$\neg$	oo	$\mathsf{v}(\mathsf{a}@\mathsf{b}) = \mathtt{T}$ iff $\mathsf{v}(\mathsf{b}) = \mathtt{F}$ $\forall \mathsf{b} \in \mathcal{D}_o$
$\vee$	ooo	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$ iff $\mathsf{v}(\mathsf{b}) = \mathtt{T}$ or $\mathsf{v}(\mathsf{c}) = \mathtt{T}$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
$\wedge$	ooo	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$ iff $\mathsf{v}(\mathsf{b}) = \mathtt{T}$ and $\mathsf{v}(\mathsf{c}) = \mathtt{T}$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
$\supset$	ooo	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$ iff $\mathsf{v}(\mathsf{b}) = \mathtt{F}$ or $\mathsf{v}(\mathsf{c}) = \mathtt{T}$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$ iff $\mathsf{v}(\mathsf{b}) = \mathsf{v}(\mathsf{c})$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
$=^\alpha$	$o\alpha\alpha$	$\mathsf{v}(\mathsf{a}@\mathsf{b}@\mathsf{c}) = \mathtt{T}$ iff $\mathsf{b} = \mathsf{c}$ $\forall \mathsf{b}, \mathsf{c} \in \mathcal{D}_o$
Π^α	$o(o\alpha)$	

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = v(c)$ $\forall b, c \in \mathcal{D}_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = \mathtt{T}$	iff $b = c$ $\forall b, c \in \mathcal{D}_o$
Π^α	$o(o\alpha)$	$v(a@f) = \mathtt{T}$	iff $\forall b \in \mathcal{D}_\alpha : v(f@b) = \mathtt{T}$ $\forall f \in \mathcal{D}_{o\alpha}$

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \rightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = v(c)$ $\forall b, c \in \mathcal{D}_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = \mathtt{T}$	iff $b = c$ $\forall b, c \in \mathcal{D}_o$
Π^α	$o(o\alpha)$	$v(a@f) = \mathtt{T}$	iff $\forall b \in \mathcal{D}_\alpha :\ v(f@b) = \mathtt{T}$ $\forall f \in \mathcal{D}_{o\alpha}$
Σ^α			

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = v(c)$ $\forall b, c \in \mathcal{D}_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = \mathtt{T}$	iff $b = c$ $\forall b, c \in \mathcal{D}_o$
Π^α	$o(o\alpha)$	$v(a@f) = \mathtt{T}$	iff $\forall b \in \mathcal{D}_\alpha : v(f@b) = \mathtt{T}$ $\forall f \in \mathcal{D}_{o\alpha}$
Σ^α	$o(o\alpha)$		

Def.: Properties of Logical Constants

Let $(D, @)$ be an applicative structure and let $v : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a function (for given $\mathtt{T} \neq \mathtt{F}$). For each logical constant c_β and for $a \in \mathcal{D}_\beta$, we define the proposition $\mathfrak{L}_c(()a)$ with respect to v:

c	β	$\mathfrak{L}_c(()a)$ holds when	
$\top$	o	$v(a) = \mathtt{T}$	
$\bot$	o	$v(a) = \mathtt{F}$	
$\neg$	oo	$v(a@b) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ $\forall b \in \mathcal{D}_o$
$\vee$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\wedge$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{T}$ and $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\supset$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = \mathtt{F}$ or $v(c) = \mathtt{T}$ $\forall b, c \in \mathcal{D}_o$
$\Leftrightarrow$	ooo	$v(a@b@c) = \mathtt{T}$	iff $v(b) = v(c)$ $\forall b, c \in \mathcal{D}_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = \mathtt{T}$	iff $b = c$ $\forall b, c \in \mathcal{D}_o$
Π^α	$o(o\alpha)$	$v(a@f) = \mathtt{T}$	iff $\forall b \in \mathcal{D}_\alpha : v(f@b) = \mathtt{T}$ $\forall f \in \mathcal{D}_{o\alpha}$
Σ^α	$o(o\alpha)$	$v(a@f) = \mathtt{T}$	iff $\exists b \in \mathcal{D}_\alpha : v(f@b) = \mathtt{T}$ $\forall f \in \mathcal{D}_{o\alpha}$

Def.: Σ-Valuation

Let $\mathcal{J} := (\mathrm{D}, @, \mathcal{E})$ be a Σ-evaluation and $\mathsf{v} : \mathcal{D}_{\mathsf{o}} \longrightarrow \{\mathtt{T}, \mathtt{F}\}$.

Def.: Σ-Valuation

Let $\mathcal{J} := (\mathrm{D}, @, \mathcal{E})$ be a Σ-evaluation and $\mathsf{v} : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$. We say

v is a Σ-valuation w.r.t $\mathcal{J}$ if

Def.: Σ-Valuation

Let $\mathcal{J} := (\mathsf{D}, @, \mathcal{E})$ be a Σ-evaluation and $\mathsf{v} : \mathcal{D}_{\mathsf{o}} \longrightarrow \{\mathtt{T}, \mathtt{F}\}$. We say v is a Σ-valuation w.r.t $\mathcal{J}$ if $\mathfrak{L}_{\mathsf{c}}((\mathcal{E}(\mathsf{c})))$ holds w.r.t v for each logical constant $\mathsf{c} \in \Sigma$.

Def.: Σ-Model

Let $\mathcal{J} := (\mathrm{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \rightarrow \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

Def.: Σ-Model

Let $\mathcal{J} := (\mathsf{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

We say $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ is a Σ-model.

Def.: Σ-Model

Let $\mathcal{J} := (\mathsf{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

We say $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ is a Σ-model.

If $(\mathsf{D}, @, \mathcal{E})$ is functional (full, standard), we say $\mathcal{M}$ is functional (full, standard).

Def.: Σ-Model

Let $\mathcal{J} := (\mathsf{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

We say $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ is a Σ-model.

If $(\mathsf{D}, @, \mathcal{E})$ is functional (full, standard), we say $\mathcal{M}$ is functional (full, standard).

If $(\mathsf{D}, @, \mathcal{E})$ is η-functional, we say $\mathcal{M}$ is η-functional.

Def.: Σ-Model

Let $\mathcal{J} := (\mathsf{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \rightarrow \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

We say $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ is a Σ-model.

If $(\mathsf{D}, @, \mathcal{E})$ is functional (full, standard), we say $\mathcal{M}$ is functional (full, standard).

If $(\mathsf{D}, @, \mathcal{E})$ is η-functional, we say $\mathcal{M}$ is η-functional.

If $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional, we say $\mathcal{M}$ is ξ-functional.

Some Conventions: Equality

Some important conventions:

- $=$ denotes primitive equality

Some Conventions: Equality

Some important conventions:

- $=$ denotes primitive equality
- $\doteq$ denotes Leibniz equality: $\mathbf{A}_\alpha \doteq^\alpha \mathbf{B}_\alpha := \forall \mathsf{P}_{o\alpha} . (\mathsf{P}\mathbf{A}) \Rightarrow (\mathsf{P}\mathbf{B})$

Some Conventions: Equality

Some important conventions:

- $=$ denotes primitive equality
- $\doteq$ denotes Leibniz equality: $\mathbf{A}_\alpha \doteq^\alpha \mathbf{B}_\alpha := \forall \mathrm{P}_{o\alpha}\text{.}(\mathrm{P}\mathbf{A}) \Rightarrow (\mathrm{P}\mathbf{B})$
- $\ddot{=}$ ...other definition of equality (e.g., see [Andrews02])

Some Conventions: Equality

Some important conventions:

- $=$ denotes primitive equality
- $\doteq$ denotes Leibniz equality: $\mathbf{A}_\alpha \doteq^\alpha \mathbf{B}_\alpha := \forall \mathrm{P}_{o\alpha}\text{.}(\mathrm{P}\mathbf{A}) \Rightarrow (\mathrm{P}\mathbf{B})$
- $\ddot{=}$ . . . other definition of equality (e.g., see [Andrews02])

We use $\stackrel{*}{=}$ in the following to refer to any of the above

Def.: Properties $\mathfrak{f}, \mathfrak{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

Def.: Properties $\mathfrak{f}, \mathfrak{b}, \eta, \xi$

Let $\mathcal{M} = (\mathcal{D}, @, \mathcal{E}, v)$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathcal{D}, @, \mathcal{E})$ is η-functional)

Def.: Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)

Def.: Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)
- f if M is functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is functional)

Def.: Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)
- f if M is functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is functional)
- b if v is injective.

Def.: Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)
- f if M is functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is functional)
- b if v is injective.

  Note: In the [JSC04]-paper, b is defined as $\mathsf{D}_o = \{\mathtt{T}, \mathtt{F}\}$, but here we are using the injectivity criterion, because we are varying the signature. If the signature is too sparse, we could have a D_o with two elements which both valuate via v to $\mathtt{T}$. Another ill case would be D_o with just one element.

Def.: Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)
- f if M is functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is functional)
- b if v is injective.
- q if for all $\alpha \in \mathcal{T}$ there is some $\mathsf{q} \in \mathsf{D}_{o\alpha\alpha}$ such that $\mathfrak{L}_{=^\alpha}(\mathsf{q})$.

Def.: Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)
- f if M is functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is functional)
- b if v is injective.
- q if for all $\alpha \in \mathcal{T}$ there is some $\mathsf{q} \in \mathsf{D}_{o\alpha\alpha}$ such that $\mathfrak{L}_{=^\alpha}(\mathsf{q})$.

  Note: This basically says that for each type α the identity relation over α is already present in the model. If we require $=_{o\alpha\alpha} \in \mathcal{C}$ with $\mathfrak{L}_{=^\alpha}(\mathcal{E}_\varphi(=_{o\alpha\alpha}))$, then this property is automatically ensured, but not for weaker signatures. See [Andrew71] for a detailed discussion of property q. Andrews constructs a Henkin model where Leibniz equality $\doteq$ does not evaluate to the intended identity relation. This is resolved by property q.

Lemma: Surjective v

Let $\mathcal{C}$ be a signature and $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model.

Lemma: Surjective v

Let $\mathcal{C}$ be a signature and $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model.
If $\mathtt{T}, \mathtt{F} \in \mathcal{C}$ or $\neg \in \mathcal{C}$ then v is surjective.

Lemma: Surjective v

Let $\mathcal{C}$ be a signature and $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model.
If $\mathtt{T}, \mathtt{F} \in \mathcal{C}$ or $\neg \in \mathcal{C}$ then v is surjective.

Proof: Exercise.

Thm.: Property $\mathfrak{b}$

Let $\mathcal{C}$ be a signature and $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model.

Thm.: Property $\mathfrak{b}$

Let $\mathcal{C}$ be a signature and $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model.
Suppose $\mathtt{T}, \mathtt{F} \in \mathcal{C}$ or $\neg \in \mathcal{C}$.

Thm.: Property $\mathfrak{b}$

Let $\mathcal{C}$ be a signature and $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model.
Suppose $\mathtt{T}, \mathtt{F} \in \mathcal{C}$ or $\neg \in \mathcal{C}$.
Then $\mathcal{M}$ satisfies property $\mathfrak{b}$ iff $|\mathsf{D}_{\mathsf{o}}| = 2$.

Thm.: Property $\mathfrak{b}$

Let $\mathcal{C}$ be a signature and $\mathcal{M} = (\mathrm{D}, @, \mathcal{E}, \mathrm{v})$ be a $\mathcal{C}$-model.
Suppose $\mathtt{T}, \mathtt{F} \in \mathcal{C}$ or $\neg \in \mathcal{C}$.
Then $\mathcal{M}$ satisfies property $\mathfrak{b}$ iff $|\mathrm{D}_o| = 2$.

Proof: Exercise.

Semantics: HOL-CUBE

Def. (Reminder): Σ-Model

Let $\mathcal{J} := (\mathrm{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

Def. (Reminder): Σ-Model

Let $\mathcal{J} := (\mathsf{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

We say $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ is a Σ-model.

Def. (Reminder): Σ-Model

Let $\mathcal{J} := (\mathsf{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

We say $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ is a Σ-model.

If $(\mathsf{D}, @, \mathcal{E})$ is functional (full, standard), we say $\mathcal{M}$ is functional (full, standard).

Def. (Reminder): Σ-Model

Let $\mathcal{J} := (\mathsf{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \to \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

We say $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ is a Σ-model.

If $(\mathsf{D}, @, \mathcal{E})$ is functional (full, standard), we say $\mathcal{M}$ is functional (full, standard).

If $(\mathsf{D}, @, \mathcal{E})$ is η-functional, we say $\mathcal{M}$ is η-functional.

Def. (Reminder): Σ-Model

Let $\mathcal{J} := (\mathsf{D}, @, \mathcal{E})$ be a Σ-evaluation and let $\mathsf{v} : \mathcal{D}_o \rightarrow \{\mathtt{T}, \mathtt{F}\}$ be a Σ-valuation w.r.t $\mathcal{J}$

We say $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ is a Σ-model.

If $(\mathsf{D}, @, \mathcal{E})$ is functional (full, standard), we say $\mathcal{M}$ is functional (full, standard).

If $(\mathsf{D}, @, \mathcal{E})$ is η-functional, we say $\mathcal{M}$ is η-functional.

If $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional, we say $\mathcal{M}$ is ξ-functional.

Def. (Reminder): Properties $\mathfrak{f}, \mathfrak{b}, \eta, \xi$

Let $\mathcal{M} = (\mathcal{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, $\mathcal{M}$ has property

Def. (Reminder): Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)

Def. (Reminder): Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)

Def. (Reminder): Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)
- f if M is functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is functional)

Def. (Reminder): Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)
- f if M is functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is functional)
- b if v is injective.

Def. (Reminder): Properties $\mathsf{f}, \mathsf{b}, \eta, \xi$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a $\mathcal{C}$-model. We say, M has property

- η if M is η-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is η-functional)
- ξ if M is ξ-functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is ξ-functional)
- f if M is functional (respectively $(\mathsf{D}, @, \mathcal{E})$ is functional)
- b if v is injective.
- q if for all $\alpha \in \mathcal{T}$ there is some $\mathsf{q} \in \mathsf{D}_{o\alpha\alpha}$ such that $\mathfrak{L}_{=^\alpha}(\mathsf{q})$.

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$.

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$. We obtain a hierarchy of subclasses of $\mathfrak{M}_\beta(\Sigma)$ by adding the properties $\xi, \eta, \mathsf{f}, \mathsf{b}$.

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$. We obtain a hierarchy of subclasses of $\mathfrak{M}_\beta(\Sigma)$ by adding the properties $\xi, \eta, \mathsf{f}, \mathsf{b}$. Thus we obtain

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$. We obtain a hierarchy of subclasses of $\mathfrak{M}_\beta(\Sigma)$ by adding the properties $\xi, \eta, \mathsf{f}, \mathsf{b}$. Thus we obtain

- $\mathfrak{M}_{\beta\eta}(\Sigma)$

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$. We obtain a hierarchy of subclasses of $\mathfrak{M}_\beta(\Sigma)$ by adding the properties $\xi, \eta, \mathsf{f}, \mathsf{b}$. Thus we obtain

- $\mathfrak{M}_{\beta\eta}(\Sigma)$
- $\mathfrak{M}_{\beta\xi}(\Sigma)$

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$. We obtain a hierarchy of subclasses of $\mathfrak{M}_\beta(\Sigma)$ by adding the properties $\xi, \eta, \mathsf{f}, \mathsf{b}$. Thus we obtain

- $\mathfrak{M}_{\beta\eta}(\Sigma)$
- $\mathfrak{M}_{\beta\xi}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma)$

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$. We obtain a hierarchy of subclasses of $\mathfrak{M}_\beta(\Sigma)$ by adding the properties $\xi, \eta, \mathsf{f}, \mathsf{b}$. Thus we obtain

- $\mathfrak{M}_{\beta\eta}(\Sigma)$
- $\mathfrak{M}_{\beta\xi}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{b}}(\Sigma)$

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$. We obtain a hierarchy of subclasses of $\mathfrak{M}_\beta(\Sigma)$ by adding the properties $\xi, \eta, \mathfrak{f}, \mathfrak{b}$. Thus we obtain

- $\mathfrak{M}_{\beta\eta}(\Sigma)$
- $\mathfrak{M}_{\beta\xi}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{b}}(\Sigma)$
- $\mathfrak{M}_{\beta\eta\mathfrak{b}}(\Sigma)$

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$. We obtain a hierarchy of subclasses of $\mathfrak{M}_\beta(\Sigma)$ by adding the properties $\xi, \eta, \mathsf{f}, \mathsf{b}$. Thus we obtain

- $\mathfrak{M}_{\beta\eta}(\Sigma)$
- $\mathfrak{M}_{\beta\xi}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{b}}(\Sigma)$
- $\mathfrak{M}_{\beta\eta\mathfrak{b}}(\Sigma)$
- $\mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma)$

Def. (Reminder): Different Model Classes

We denote the class of $\mathcal{C}$-models by $\mathfrak{M}_\beta(\Sigma)$. We obtain a hierarchy of subclasses of $\mathfrak{M}_\beta(\Sigma)$ by adding the properties $\xi, \eta, \mathfrak{f}, \mathfrak{b}$. Thus we obtain

- $\mathfrak{M}_{\beta\eta}(\Sigma)$
- $\mathfrak{M}_{\beta\xi}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{b}}(\Sigma)$
- $\mathfrak{M}_{\beta\eta\mathfrak{b}}(\Sigma)$
- $\mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma)$
- $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$

Def.: Satisfies, models, and $\models$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a Σ-model and let φ be an assignment into $\mathcal{M}$.

Def.: Satisfies, models, and $\models$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a Σ-model and let φ be an assignment into $\mathcal{M}$.

We say φ satisfies a formula $\mathbf{A} \in \mathit{wff}_{\mathsf{o}}(\Sigma)$ in $\mathcal{M}$ (we write $\mathcal{M} \models_{\varphi} \mathbf{A}$) if $v(\mathcal{E}_{\varphi}(\mathbf{A})) = \mathtt{T}$.

Def.: Satisfies, models, and $\models$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a Σ-model and let φ be an assignment into $\mathcal{M}$.

We say φ satisfies a formula $\mathbf{A} \in \mathit{wff}_o(\Sigma)$ in $\mathcal{M}$ (we write $\mathcal{M} \models_\varphi \mathbf{A}$) if $\upsilon(\mathcal{E}_\varphi(\mathbf{A})) = \mathtt{T}$.

We say that $\mathbf{A}$ is valid in $\mathcal{M}$ (and write $\mathcal{M} \models \mathbf{A}$) if $\mathcal{M} \models_\varphi \mathbf{A}$ for all assignments φ.

Def.: Satisfies, models, and $\models$

Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a Σ-model and let φ be an assignment into $\mathcal{M}$.

We say φ satisfies a formula $\mathbf{A} \in \mathit{wff}_{\mathrm{o}}(\Sigma)$ in $\mathcal{M}$ (we write $\mathcal{M} \models_{\varphi} \mathbf{A}$) if $\upsilon(\mathcal{E}_{\varphi}(\mathbf{A})) = \mathtt{T}$.

We say that $\mathbf{A}$ is valid in $\mathcal{M}$ (and write $\mathcal{M} \models \mathbf{A}$) if $\mathcal{M} \models_{\varphi} \mathbf{A}$ for all assignments φ. When $\mathbf{A} \in \mathit{cwff}_{\mathrm{o}}(\Sigma)$, we drop the reference to the assignment and use the notation $\mathcal{M} \models \mathbf{A}$.

Def.: Satisfies, models, and $\models$

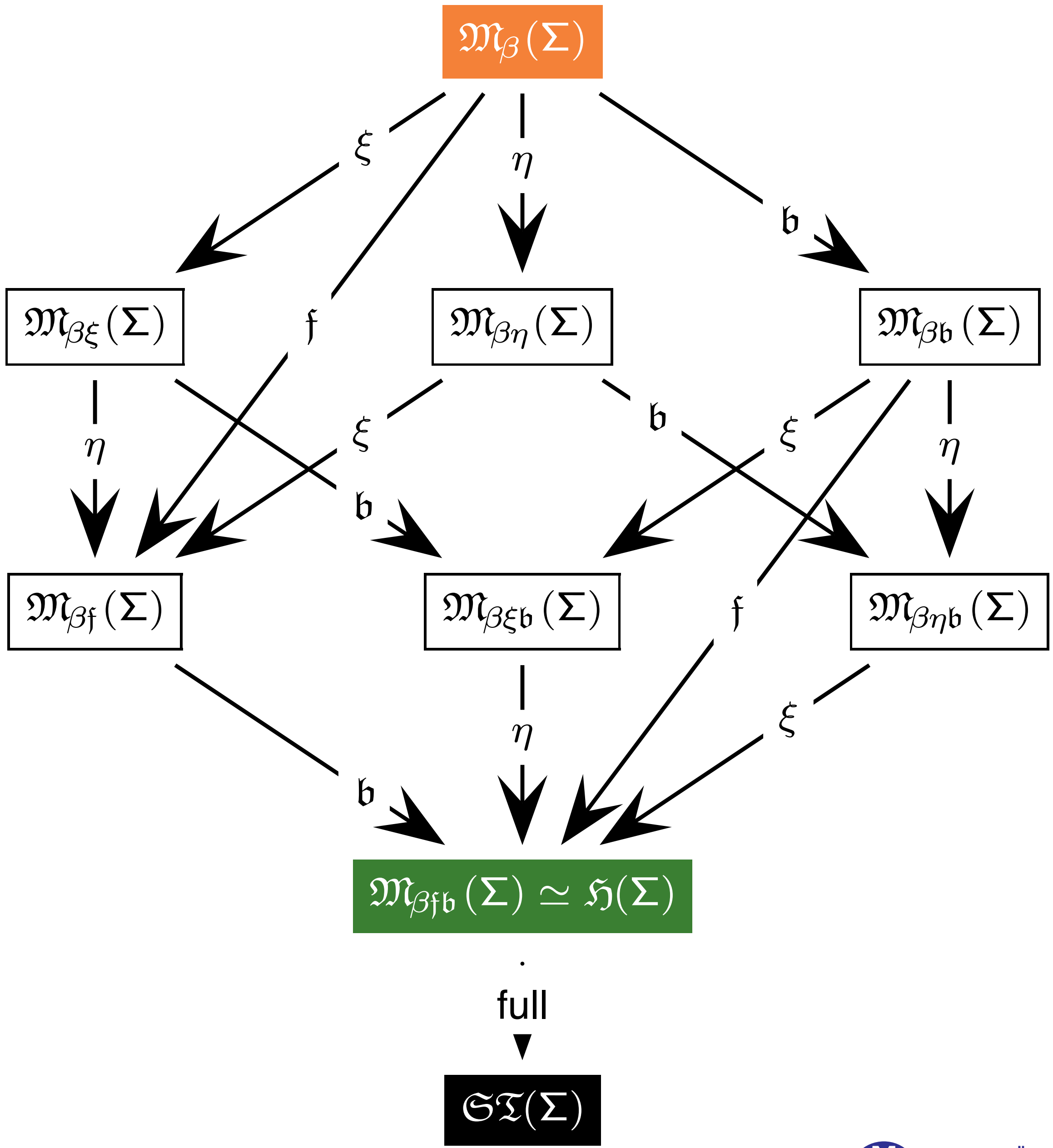
Let $\mathcal{M} = (\mathsf{D}, @, \mathcal{E}, \mathsf{v})$ be a Σ-model and let φ be an assignment into $\mathcal{M}$.

We say φ satisfies a formula $\mathbf{A} \in \mathit{wff}_o(\Sigma)$ in $\mathcal{M}$ (we write $\mathcal{M} \models_\varphi \mathbf{A}$) if $\upsilon(\mathcal{E}_\varphi(\mathbf{A})) = \mathtt{T}$.

We say that $\mathbf{A}$ is valid in $\mathcal{M}$ (and write $\mathcal{M} \models \mathbf{A}$) if $\mathcal{M} \models_\varphi \mathbf{A}$ for all assignments φ. When $\mathbf{A} \in \mathit{cwff}_o(\Sigma)$, we drop the reference to the assignment and use the notation $\mathcal{M} \models \mathbf{A}$.

Finally, we say that $\mathcal{M}$ is a Σ-model for a set $\Phi \subseteq \mathit{cwff}_o(\Sigma)$ (we write $\mathcal{M} \models \Phi$) if $\mathcal{M} \models \mathbf{A}$ for all $\mathbf{A} \in \Phi$.

Semantics: HOL-CUBE



Landscape of HOL model classes

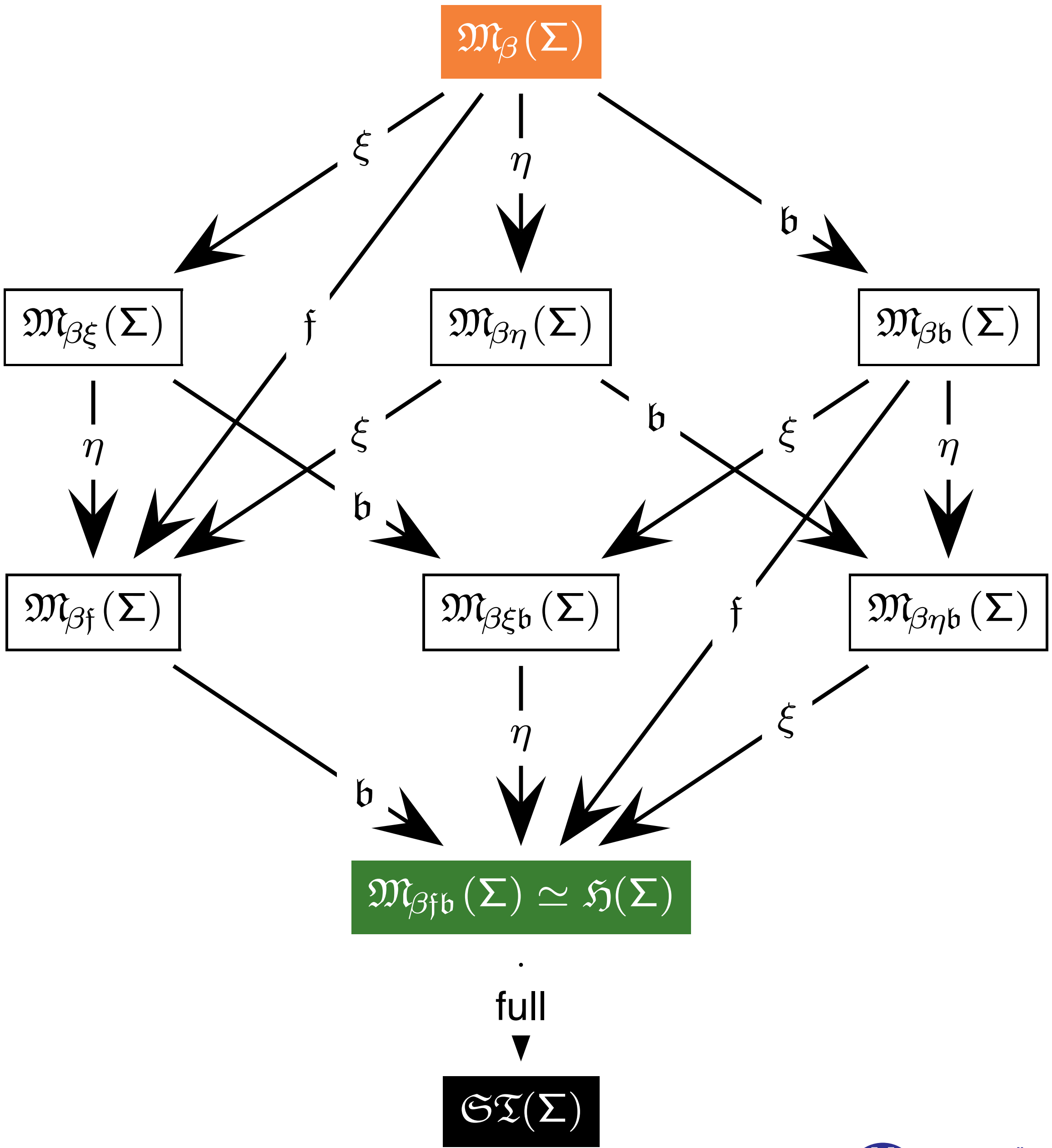
[Kohlhase-PhD-94]

[Benzmüller-PhD-99]

[Brown-PhD-04]

[Benzm.BrownKohlhase-JSL-04]

Semantics: HOL-CUBE



Landscape of HOL model classes

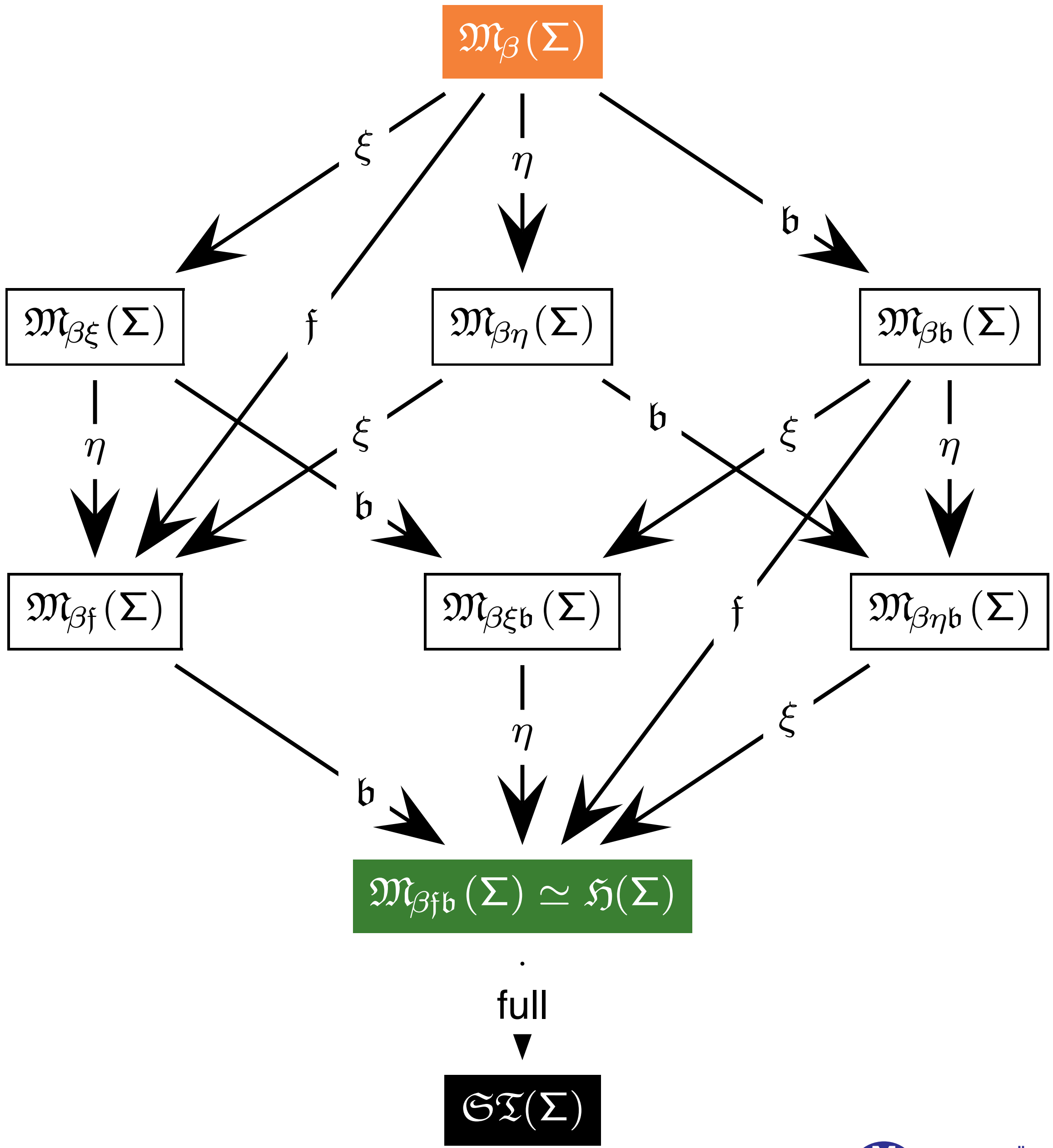
[Kohlhase-PhD-94]

[Benzmüller-PhD-99]

[Brown-PhD-04]

[Benzm.BrownKohlhase-JSL-04]

$\mathfrak{M}_{\beta}(\Sigma)$ model class for Σ-fragment of elementary type theory



Landscape of HOL model classes

[Kohlhase-PhD-94]

[Benzmüller-PhD-99]

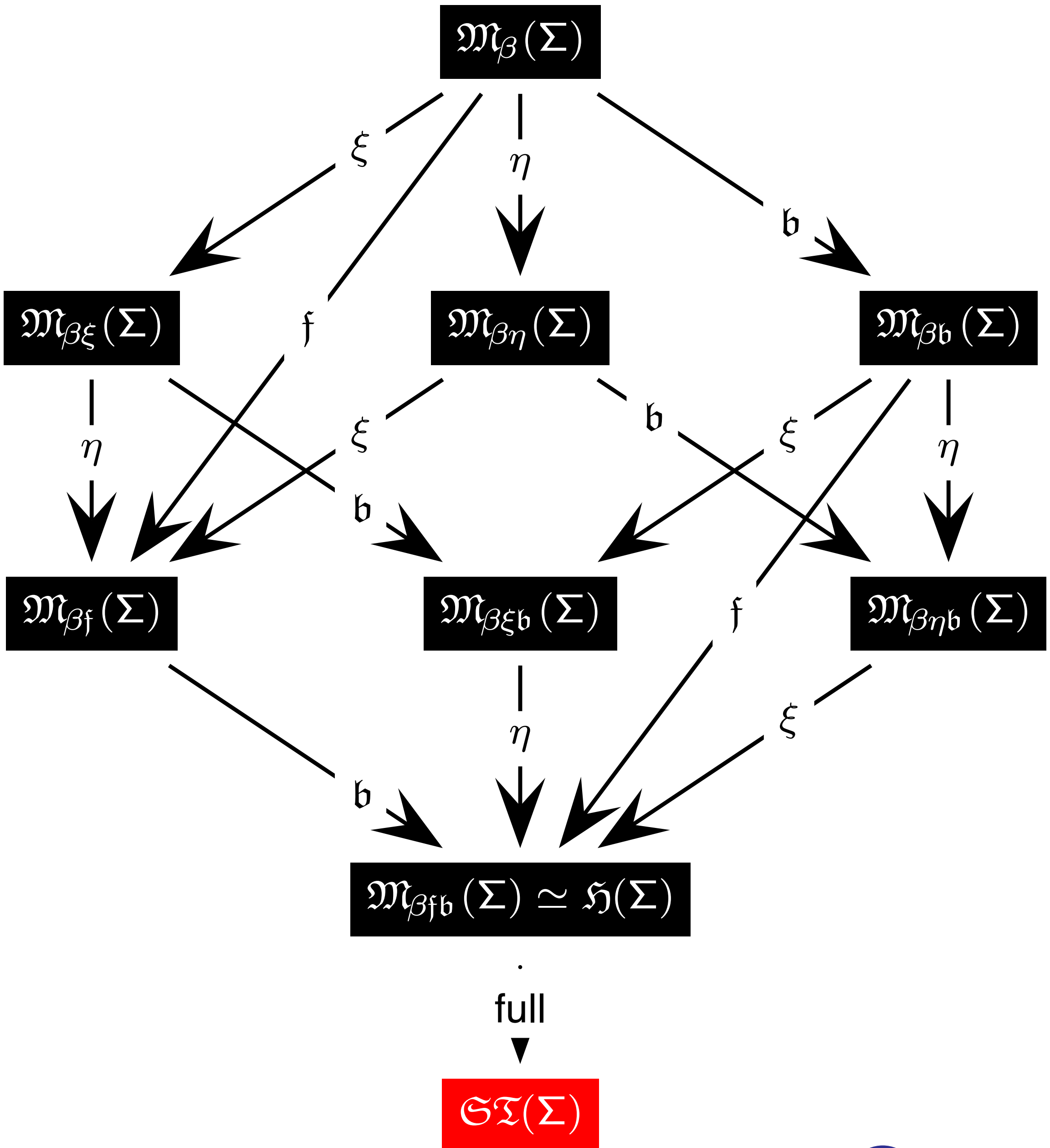
[Brown-PhD-04]

[Benzm.BrownKohlhase-JSL-04]

$\mathfrak{M}_{\beta}(\Sigma)$ model class for Σ-fragment of elementary type theory

$\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$ model class for Σ-fragment of extensional type theory (Henkin models)

Semantics: HOL-CUBE

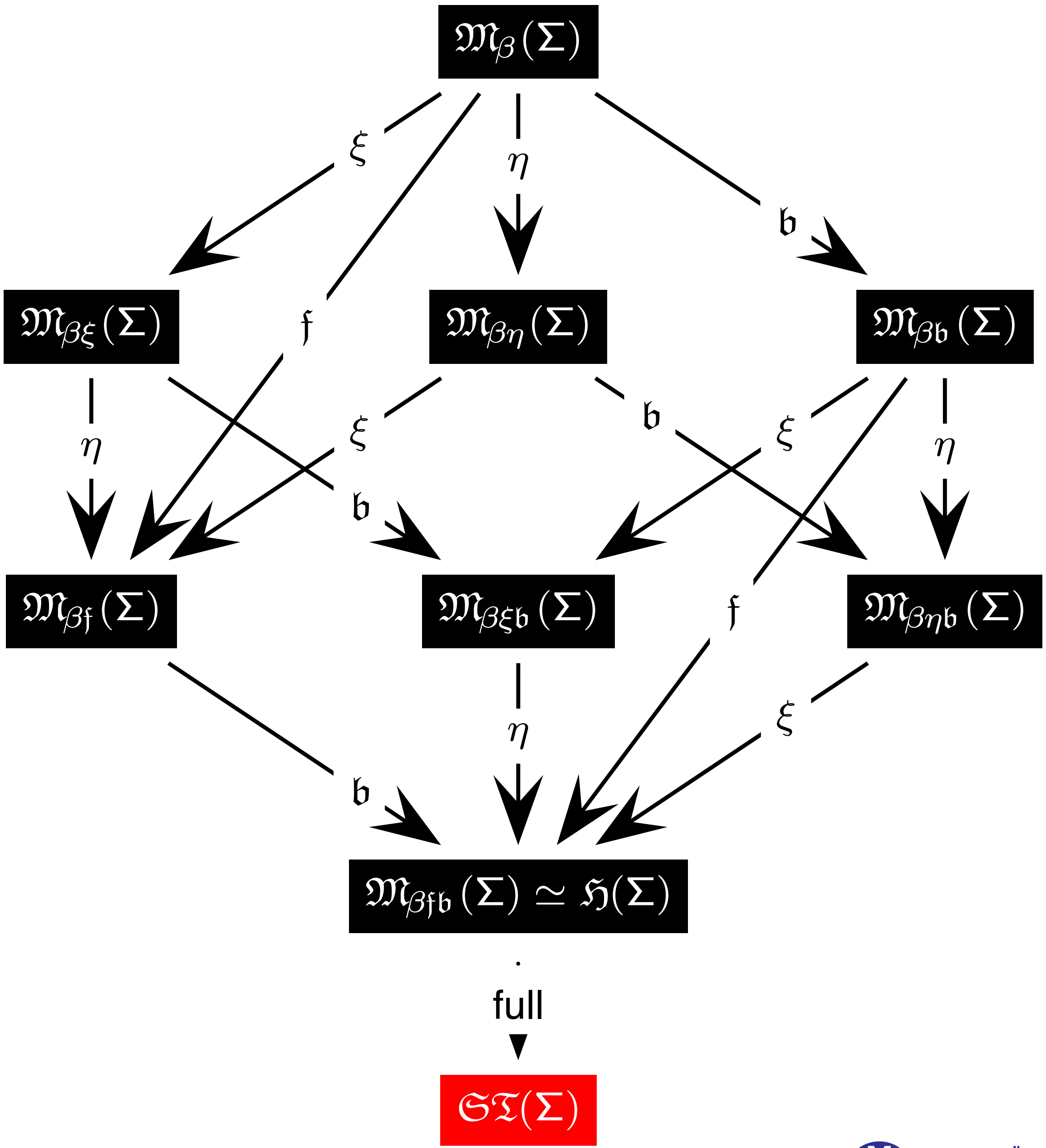


β: models support β-equality

$\mathfrak{q}$: models provide identity relations

$\forall\alpha : \text{id} \in \mathcal{D}_{\alpha\to\alpha\to o}$

Semantics: HOL-CUBE



β: models support β-equality

$\mathfrak{q}$: models provide identity relations

$\forall\alpha : \mathsf{id} \in \mathcal{D}_{\alpha\to\alpha\to o}$

- [Andrews72]: without property $\mathfrak{q}$ Leibniz equality $\doteq$ not necessarily evaluates to identity relation even in Henkin semantics ($\mathfrak{H}(\Sigma)$)

Standard Models and Henkin Models

Leon Henkin generalized the class of admissible domains for functional types.

Standard Models and Henkin Models

Leon Henkin generalized the class of admissible domains for functional types.

Instead of requiring $\mathcal{D}_{\alpha\beta}$ (and thus in particular, $\mathcal{D}_{o\iota}$) to be the full set of functions (predicates), it is sufficient to require that $\mathcal{D}_{\alpha\beta}$ has enough members that any well-formed formula can be evaluated (in other words, the domains of function types are rich enough to satisfy comprehension).

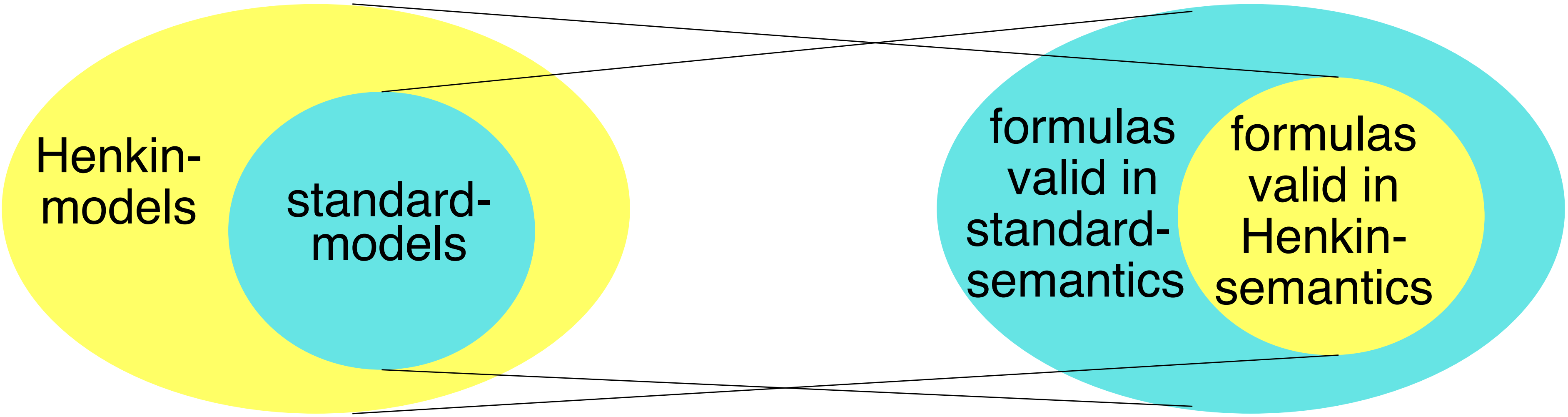
Standard Models and Henkin Models

Leon Henkin generalized the class of admissible domains for functional types.

Instead of requiring $\mathcal{D}_{\alpha\beta}$ (and thus in particular, $\mathcal{D}_{o\iota}$) to be the full set of functions (predicates), it is sufficient to require that $\mathcal{D}_{\alpha\beta}$ has enough members that any well-formed formula can be evaluated (in other words, the domains of function types are rich enough to satisfy comprehension).

Note that with this generalized notion of a model, there are fewer formulae that are valid in all models (intuitively, for any given formula there are more possibilities for counter-models).

Standard Models and Henkin Models



Standard Models and Henkin Models

The generalization to Henkin models restricts the set of valid formulae sufficiently so that all of them can be proven by a Hilbert-style calculus [Henkin50].

Standard Models and Henkin Models

The generalization to Henkin models restricts the set of valid formulae sufficiently so that all of them can be proven by a Hilbert-style calculus [Henkin50].

Of course our HOL-CUBE is not complete here; we can axiomatically require the existence of particular (classes of) functions, e.g., by assuming the description or choice operators.

Standard Models and Henkin Models

The generalization to Henkin models restricts the set of valid formulae sufficiently so that all of them can be proven by a Hilbert-style calculus [Henkin50].

Of course our HOL-CUBE is not complete here; we can axiomatically require the existence of particular (classes of) functions, e.g., by assuming the description or choice operators.

We will not pursue this here; for a detailed discussion of the semantic issues raised by the presence of these logical constants see [Andrews72].

Standard Models and Henkin Models

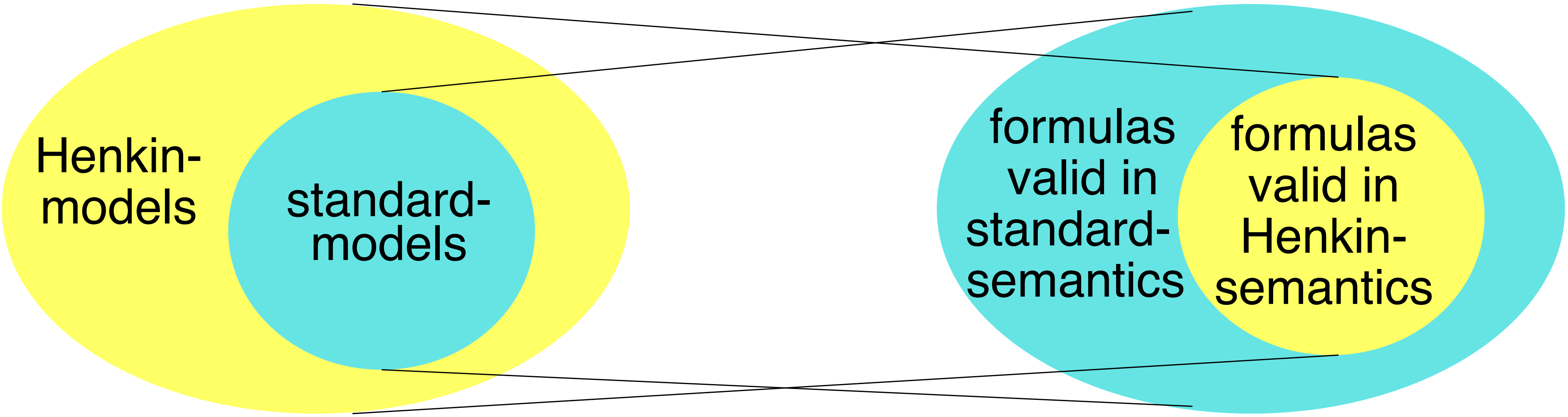
The generalization to Henkin models restricts the set of valid formulae sufficiently so that all of them can be proven by a Hilbert-style calculus [Henkin50].

Of course our HOL-CUBE is not complete here; we can axiomatically require the existence of particular (classes of) functions, e.g., by assuming the description or choice operators.

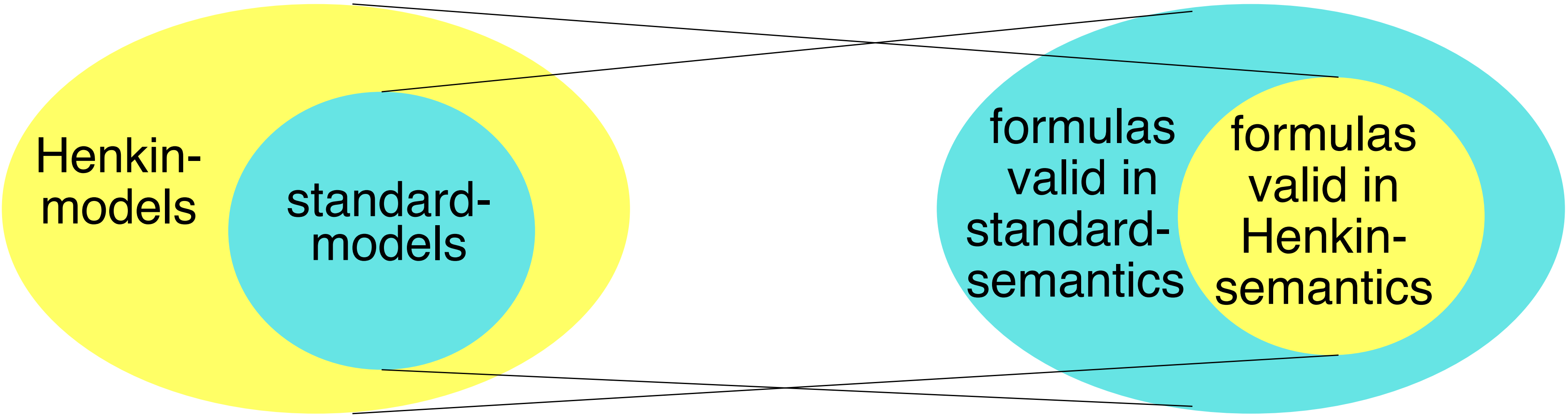
We will not pursue this here; for a detailed discussion of the semantic issues raised by the presence of these logical constants see [Andrews72].

Note that even though we can consider model classes with richer and richer function spaces, we can never reach standard models where function spaces are full while maintaining complete (recursively axiomatizable) calculi.

Standard Models and Henkin Models



Standard Models and Henkin Models



What has been our motivation for further generalization of Henkin semantics with respect to Boolean and functional extensionality?

Models without Functional Extensionality

Motivation: modeling programs as (higher-order) functions

- We might be interested in intensional properties like run-time complexity.

Models without Functional Extensionality

Motivation: modeling programs as (higher-order) functions

- We might be interested in intensional properties like run-time complexity.
- $\mathbf{I} := \lambda X.X$ and $\mathbf{L} := \lambda X.\mathsf{rev}(\mathsf{rev}(X))$, where rev is the self-inverse function.

Models without Functional Extensionality

Motivation: modeling programs as (higher-order) functions

- We might be interested in intensional properties like run-time complexity.
- $\mathbf{I} := \lambda X.X$ and $\mathbf{L} := \lambda X.\mathsf{rev}(\mathsf{rev}(X))$, where rev is the self-inverse function.
- The identity function has constant complexity, the function rev is linear in the length of its argument.

Models without Functional Extensionality

How do we account for models without functional extensionality?

- We have generalized the notion of domains at function types and evaluation functions.

Models without Functional Extensionality

How do we account for models without functional extensionality?

- We have generalized the notion of domains at function types and evaluation functions.
- The usual construction already uses sets of (extensional) functions for the domains of function type and the property of functionality to construct values for λ-terms.

Models without Functional Extensionality

How do we account for models without functional extensionality?

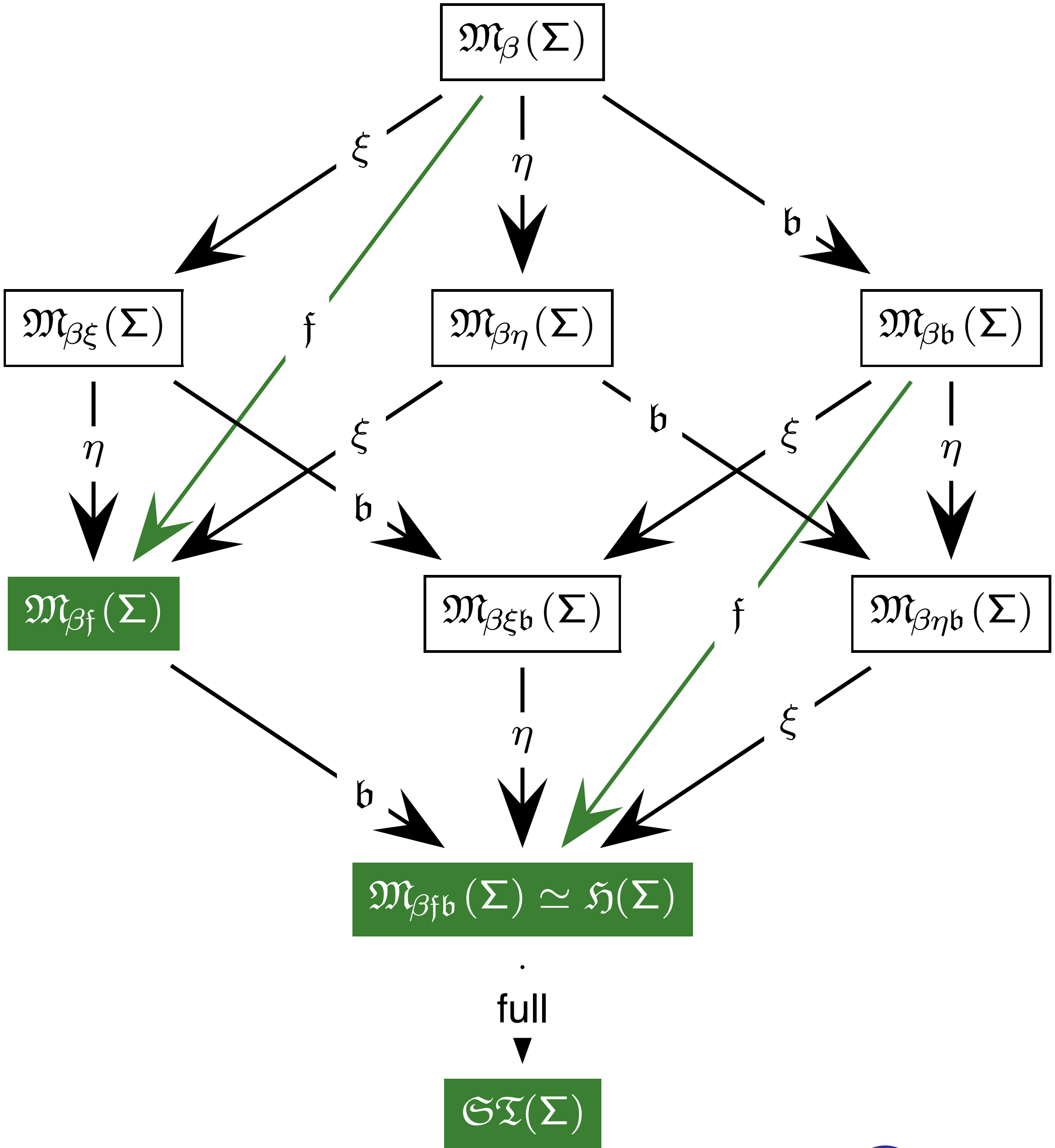
- We have generalized the notion of domains at function types and evaluation functions.
- The usual construction already uses sets of (extensional) functions for the domains of function type and the property of functionality to construct values for λ-terms.
- We build on the notion of applicative structures to define Σ-evaluations, where the evaluation function is assumed to respect application and β-conversion.

Models without Functional Extensionality

How do we account for models without functional extensionality?

- We have generalized the notion of domains at function types and evaluation functions.
- The usual construction already uses sets of (extensional) functions for the domains of function type and the property of functionality to construct values for λ-terms.
- We build on the notion of applicative structures to define Σ-evaluations, where the evaluation function is assumed to respect application and β-conversion.
- In such models, a function is not uniquely determined by its behavior on all possible arguments.

Semantics: HOL-CUBE



$\mathfrak{f}$: models are functional

$$\forall f, g \in \mathcal{D}_{\beta\alpha} :$$
$$f = g \text{ iff } f@a = g@a \; (\forall a \in \mathcal{D}_{\alpha})$$

Models without η- or ξ-Functionality

Motivation: in standard literature functional extensionality is often is discussed in terms of

Models without η- or ξ-Functionality

Motivation: in standard literature functional extensionality is often is discussed in terms of

- ξ-functionality

Models without η- or ξ-Functionality

Motivation: in standard literature functional extensionality is often is discussed in terms of

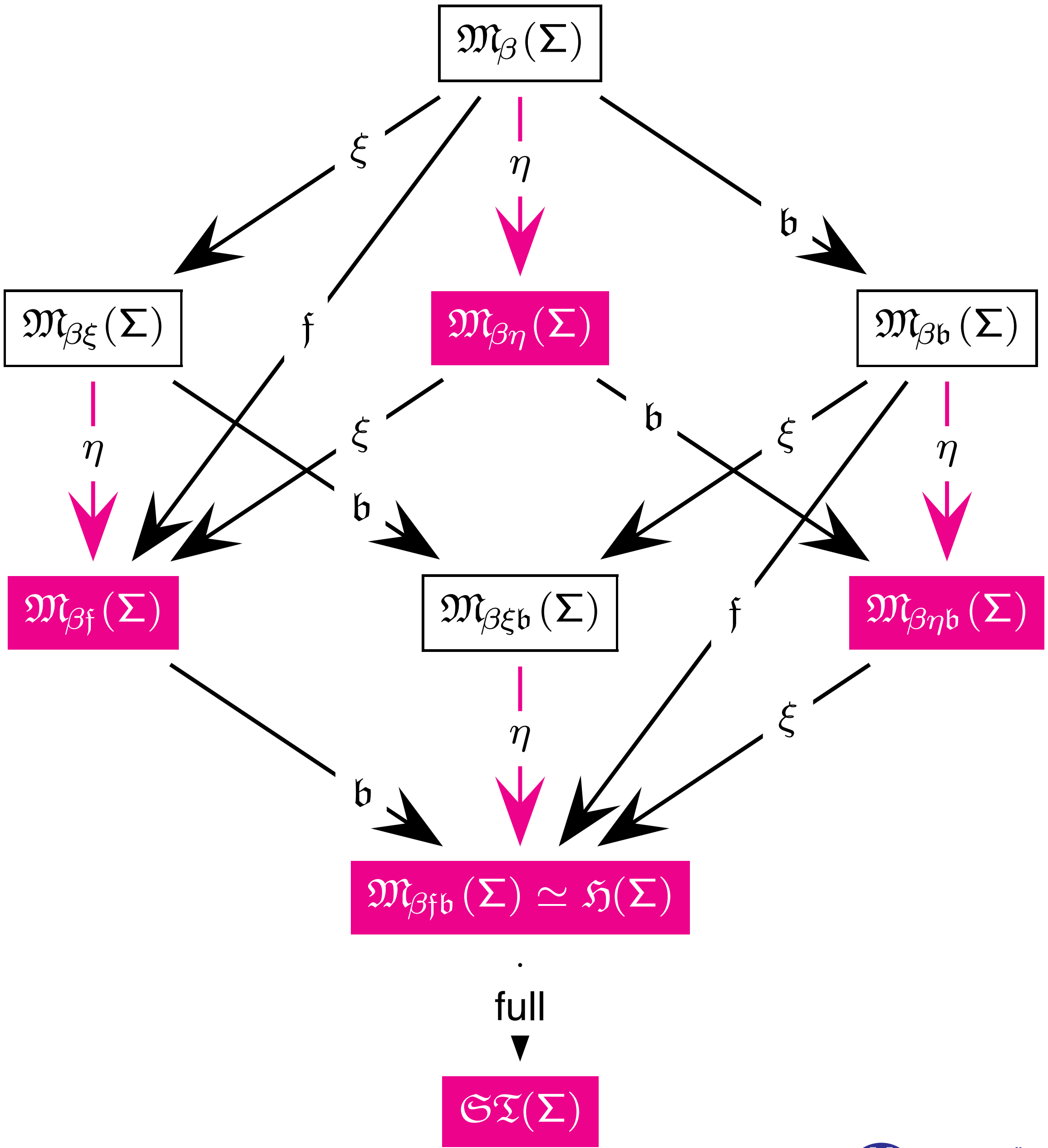
- ξ-functionality
- η-functionality

Models without η- or ξ-Functionality

Motivation: in standard literature functional extensionality is often is discussed in terms of

- ξ-functionality
- η-functionality
- Therefore, we integrated these two cases in our landscape.

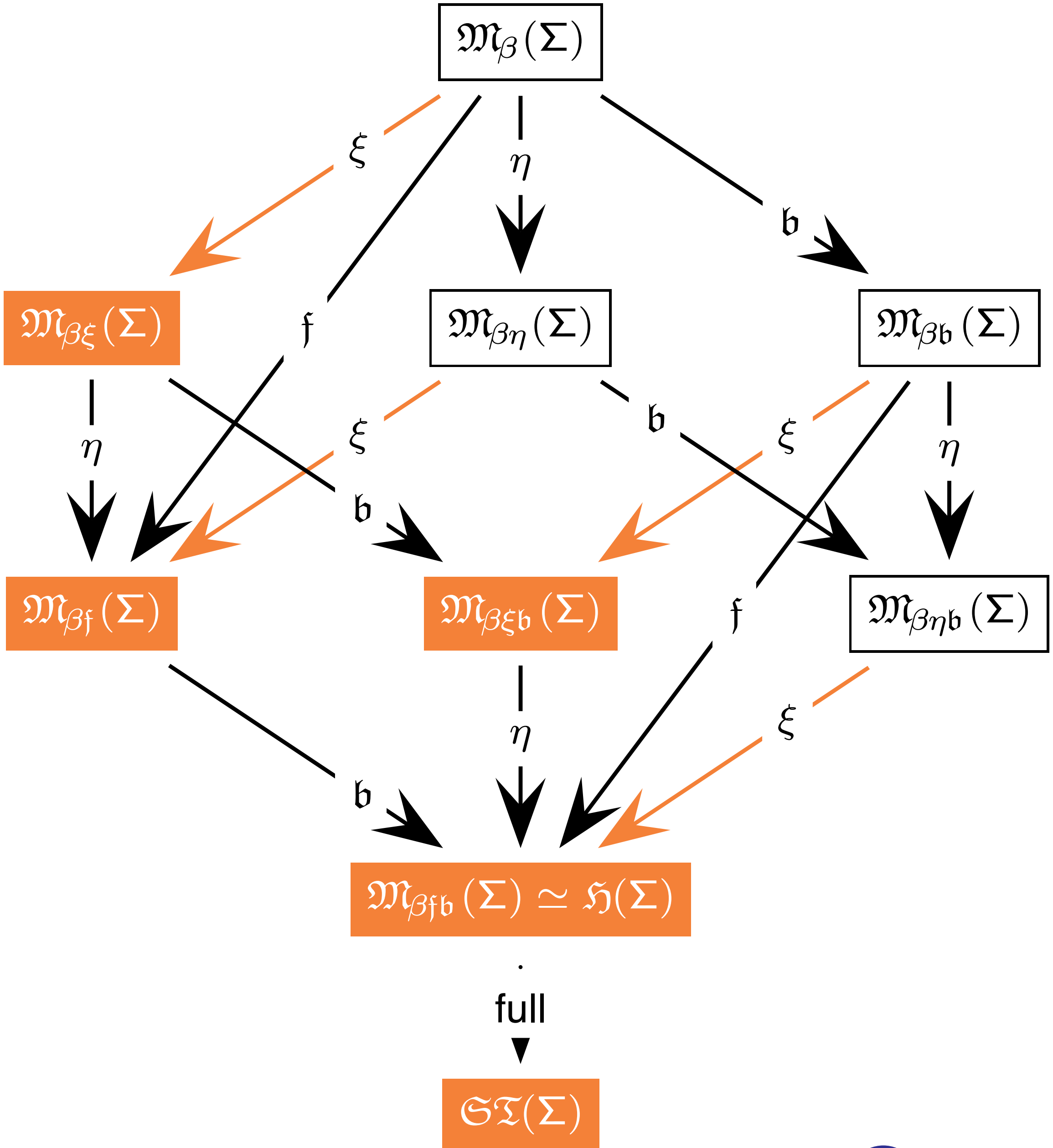
Semantics: HOL-CUBE



η: models are η-functional

$$\mathcal{E}_\varphi(\mathsf{A}) = \mathcal{E}_\varphi(\mathsf{A}\downarrow_{\beta\eta})$$

Semantics: HOL-CUBE



ξ: models are ξ-functional

$\mathcal{E}_\varphi(\lambda X_\alpha \mathbf{.} M_\beta) = \mathcal{E}_\varphi(\lambda X_\alpha \mathbf{.} N_\beta)$ iff
$\mathcal{E}_{\varphi,[a/X]}(M) = \mathcal{E}_{\varphi,[a/X]}(N)$ $(\forall a \in \mathcal{D}_\alpha)$

Models without Boolean Extensionality

Motivation: Semantics of natural language

Models without Boolean Extensionality

Motivation: Semantics of natural language

- We may not want to commit to a logic where the sentence
  “John believes that Phil is a woodchuck”

Models without Boolean Extensionality

Motivation: Semantics of natural language

- We may not want to commit to a logic where the sentence "John believes that Phil is a woodchuck" automatically entails "John believes that Phil is a groundhog"

Models without Boolean Extensionality

Motivation: Semantics of natural language

- We may not want to commit to a logic where the sentence "John believes that Phil is a woodchuck" automatically entails "John believes that Phil is a groundhog" since John might not know that "woodchuck" is just another word for "groundhog".

Models without Boolean Extensionality

Motivation: Semantics of natural language

- We may not want to commit to a logic where the sentence "John believes that Phil is a woodchuck" automatically entails "John believes that Phil is a groundhog" since John might not know that "woodchuck" is just another word for "groundhog".
- However, Boolean extensionality does just that: whenever two propositions are equivalent, they must be equal, and can be substituted for each other.

Models without Boolean Extensionality

Motivation: Semantics of natural language

- We may not want to commit to a logic where the sentence "John believes that Phil is a woodchuck" automatically entails "John believes that Phil is a groundhog" since John might not know that "woodchuck" is just another word for "groundhog".
- However, Boolean extensionality does just that: whenever two propositions are equivalent, they must be equal, and can be substituted for each other.
- Another example: $\mathsf{obvious}(\mathbf{O})$ and $\mathsf{obvious}(\mathbf{F})$ where $\mathbf{O} := 2 + 2 = 4$ and $\mathbf{F} := \forall n > 2.x^n + y^n = z^n \Rightarrow x = y = z = 0$ should not be equivalent, even if their arguments are.

Models without Boolean Extensionality

Motivation: Semantics of natural language

- We may not want to commit to a logic where the sentence "John believes that Phil is a woodchuck" automatically entails "John believes that Phil is a groundhog" since John might not know that "woodchuck" is just another word for "groundhog".
- However, Boolean extensionality does just that: whenever two propositions are equivalent, they must be equal, and can be substituted for each other.
- Another example: $\mathsf{obvious}(\mathbf{O})$ and $\mathsf{obvious}(\mathbf{F})$ where $\mathbf{O} := 2 + 2 = 4$ and $\mathbf{F} := \forall n > 2.x^n + y^n = z^n \Rightarrow x = y = z = 0$ should not be equivalent, even if their arguments are.
- Such phenomena have been studied under the heading of "hyper-intensional semantics" in theoretical semantics.

Models without Boolean Extensionality

How do we account for models without Boolean extensionality?

- We have weakened the assumption that $\mathcal{D}_o = \{\mathtt{T}, \mathtt{F}\}$, since this entails that the values of $\mathbf{O}$ and $\mathbf{F}$ are identical.

Models without Boolean Extensionality

How do we account for models without Boolean extensionality?

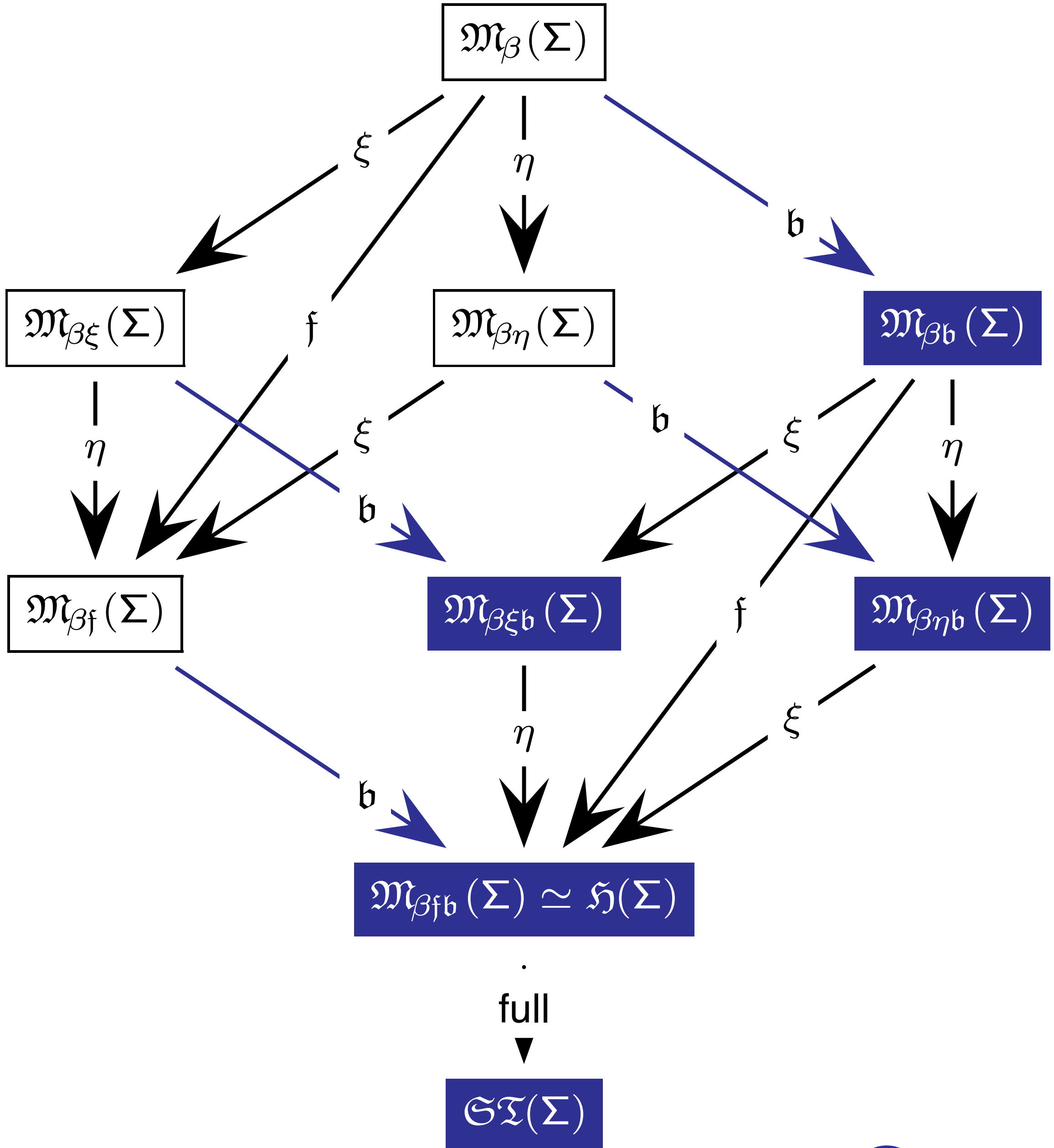
- We have weakened the assumption that $\mathcal{D}_o = \{\mathtt{T}, \mathtt{F}\}$, since this entails that the values of $\mathbf{O}$ and $\mathbf{F}$ are identical.
- In our Σ-models without property $\mathfrak{b}$ we only insist that there is a division of the truth values into “good” and “bad” ones, which we express by insisting on the existence of a valuation υ of $\mathcal{D}_o$, i.e., a function $\upsilon\colon \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ that is coordinated with the interpretations of the logical constants $\neg$, $\vee$, and Π^α (for each type α).

Models without Boolean Extensionality

How do we account for models without Boolean extensionality?

- We have weakened the assumption that $\mathcal{D}_o = \{\mathtt{T}, \mathtt{F}\}$, since this entails that the values of $\mathbf{O}$ and $\mathbf{F}$ are identical.
- In our Σ-models without property $\mathfrak{b}$ we only insist that there is a division of the truth values into "good" and "bad" ones, which we express by insisting on the existence of a valuation υ of $\mathcal{D}_o$, i.e., a function $\upsilon : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ that is coordinated with the interpretations of the logical constants $\neg$, $\vee$, and Π^α (for each type α).
- Notion of validity: we call a sentence $\mathbf{A}$ valid in such a model if $\upsilon(\mathsf{a}) = \mathtt{T}$, where $\mathsf{a} \in \mathcal{D}_o$ is the denotation of the sentence $\mathbf{A}$.

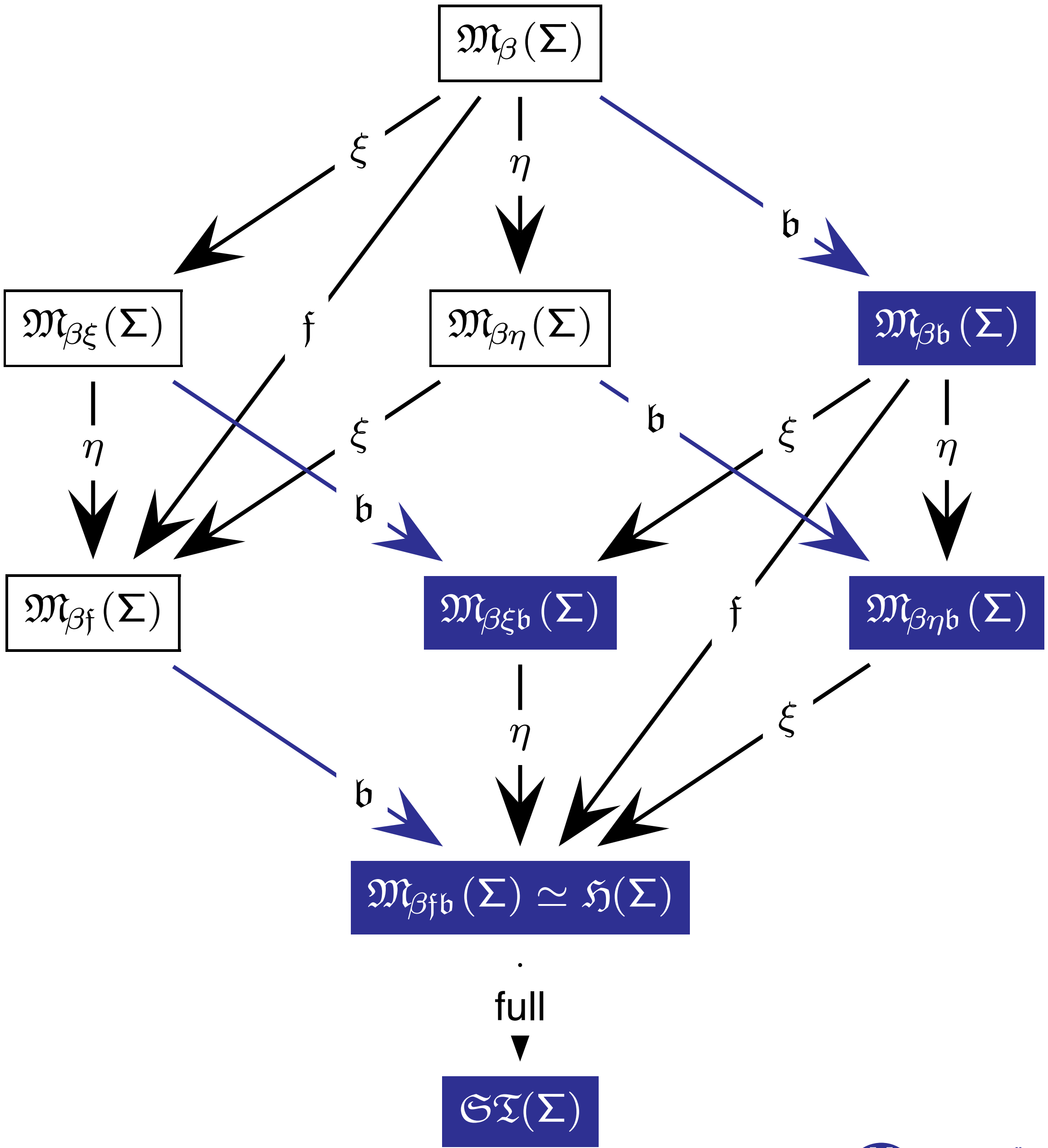
Semantics: HOL-CUBE



$\mathfrak{b}$: models are Boolean extensional

v is injective

Semantics: HOL-CUBE



$\mathfrak{b}$: models are Boolean extensional

v is injective

If Σ contains sufficiently many logical constants:

$\mathcal{D}_o = \{\bot, \top\}$

Semantics and Theorem Proving: Test Problems for Theorem Provers

Test Problems for Theorem Provers

- Test problems for FOL theorem provers

Test Problems for Theorem Provers

- Test problems for FOL theorem provers
  - [McCharenOverbeekWos76], [WilsonMinker79], [Pelletier86], etc.

Test Problems for Theorem Provers

- Test problems for FOL theorem provers
  - [McCharenOverbeekWos76], [WilsonMinker79], [Pelletier86], etc.
  - TPTP [PelletierSutcliffeSuttner02]

Test Problems for Theorem Provers

- Test problems for FOL theorem provers
  - [McCharenOverbeekWos76], [WilsonMinker79], [Pelletier86], etc.
  - TPTP [PelletierSutcliffeSuttner02]
  - significantly fostered the development of FOL ATPs

Test Problems for Theorem Provers

- Test problems for FOL theorem provers
  - [McCharenOverbeekWos76], [WilsonMinker79], [Pelletier86], etc.
  - TPTP [PelletierSutcliffeSuttner02]
  - significantly fostered the development of FOL ATPs
- Test problems for HOL theorem provers

Test Problems for Theorem Provers

- Test problems for FOL theorem provers
  - [McCharenOverbeekWos76], [WilsonMinker79], [Pelletier86], etc.
  - TPTP [PelletierSutcliffeSuttner02]
  - significantly fostered the development of FOL ATPs
- Test problems for HOL theorem provers
  - common library missing

Test Problems for Theorem Provers

- Test problems for FOL theorem provers
  - [McCharenOverbeekWos76], [WilsonMinker79], [Pelletier86], etc.
  - TPTP [PelletierSutcliffeSuttner02]
  - significantly fostered the development of FOL ATPs
- Test problems for HOL theorem provers
  - common library missing
- Following slides: example problems from our paper [TPHOLS-05]

Test Problems for Theorem Provers

- Test problems for FOL theorem provers
  - [McCharenOverbeekWos76], [WilsonMinker79], [Pelletier86], etc.
  - TPTP [PelletierSutcliffeSuttner02]
  - significantly fostered the development of FOL ATPs
- Test problems for HOL theorem provers
  - common library missing
- Following slides: example problems from our paper [TPHOLS-05]
- Are we proposing challenging HOL benchmark problems?

Test Problems for Theorem Provers

- Test problems for FOL theorem provers
  - [McCharenOverbeekWos76], [WilsonMinker79], [Pelletier86], etc.
  - TPTP [PelletierSutcliffeSuttner02]
  - significantly fostered the development of FOL ATPs
- Test problems for HOL theorem provers
  - common library missing
- Following slides: example problems from our paper [TPHOLS-05]
- Are we proposing challenging HOL benchmark problems?
  - No!!!

Test Problems for Theorem Provers

- Examples are simple

Test Problems for Theorem Provers

- Examples are simple
  - highlight the essence of some semantical or technical point

Test Problems for Theorem Provers

- Examples are simple
  - highlight the essence of some semantical or technical point
  - easy to understand and easy to encode

Test Problems for Theorem Provers

- Examples are simple
  - highlight the essence of some semantical or technical point
  - easy to understand and easy to encode
  - relevant for both: automated and interactive TP

Test Problems for Theorem Provers

- Examples are simple
  - highlight the essence of some semantical or technical point
  - easy to understand and easy to encode
  - relevant for both: automated and interactive TP
- Examples are structured

Test Problems for Theorem Provers

- Examples are simple
  - highlight the essence of some semantical or technical point
  - easy to understand and easy to encode
  - relevant for both: automated and interactive TP
- Examples are structured
  - quick indicators for completeness and soundness wrt to HOL model classes from [Benzm.BrownKohlhase-JSL-04]

Test Problems for Theorem Provers

- Examples are simple
  - highlight the essence of some semantical or technical point
  - easy to understand and easy to encode
  - relevant for both: automated and interactive TP
- Examples are structured
  - quick indicators for completeness and soundness wrt to HOL model classes from [Benzm.BrownKohlhase-JSL-04]
  - shall precede formal soundness / completeness analysis

Test Problems for Theorem Provers

- Examples are simple
  - highlight the essence of some semantical or technical point
  - easy to understand and easy to encode
  - relevant for both: automated and interactive TP
- Examples are structured
  - quick indicators for completeness and soundness wrt to HOL model classes from [Benzm.BrownKohlhase-JSL-04]
  - shall precede formal soundness / completeness analysis
  - many are collected from experience with LEO and TPS

Test Problems for Theorem Provers

- Examples are simple
  - highlight the essence of some semantical or technical point
  - easy to understand and easy to encode
  - relevant for both: automated and interactive TP
- Examples are structured
  - quick indicators for completeness and soundness wrt to HOL model classes from [Benzm.BrownKohlhase-JSL-04]
  - shall precede formal soundness / completeness analysis
  - many are collected from experience with LEO and TPS
- (Some more challenging examples are also added in [TPHOLS-05])

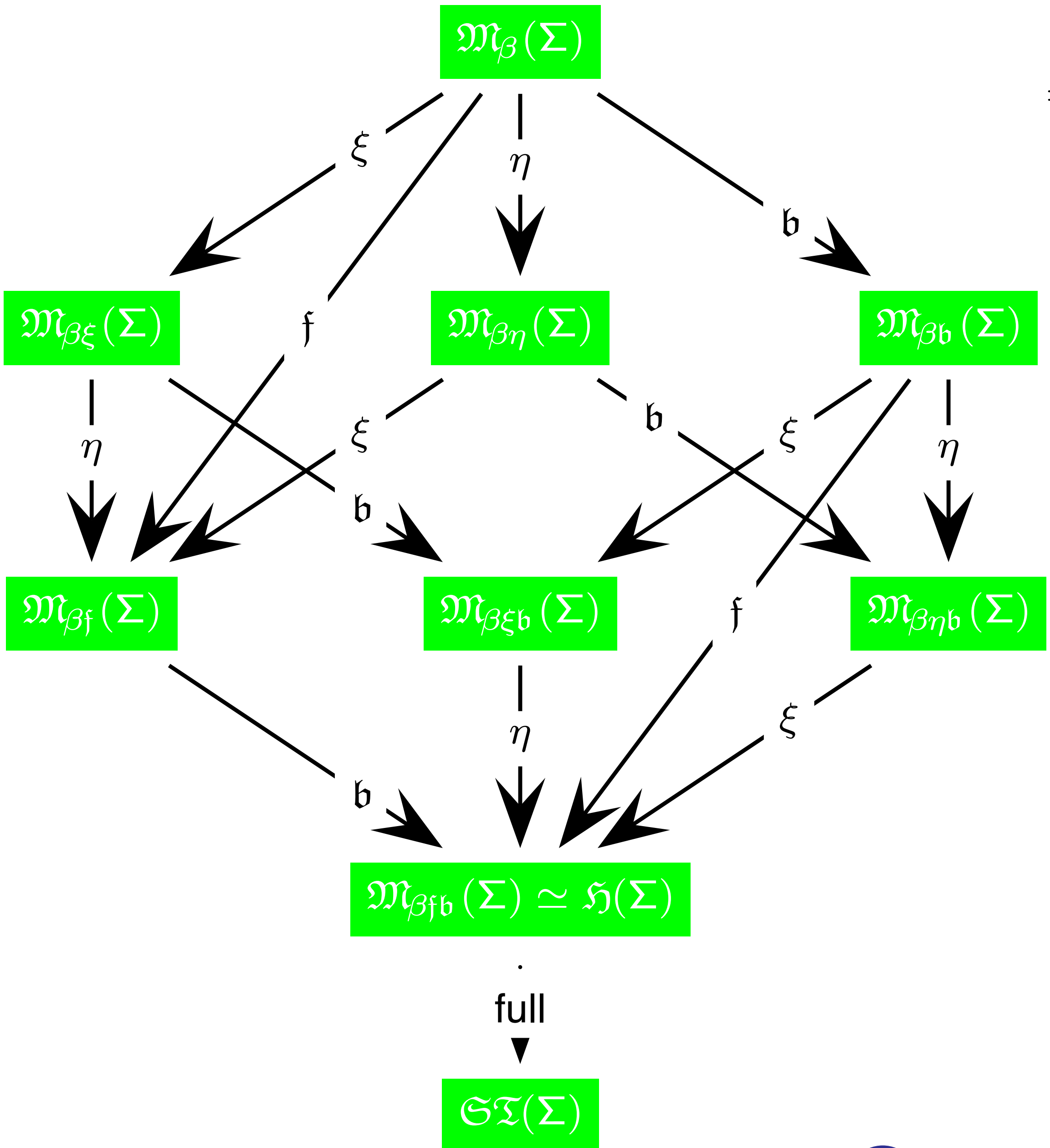
Remark: Signature

Unless stated otherwise we assume on the following slides that our signature Σ contains the following logical connectives:

$$\{\top, \bot, \neg, \wedge, \vee, \supset, \Leftrightarrow\} \cup \{\Pi^\alpha, \Sigma^\alpha, =^\alpha\}$$

(less logical connectives are possible)

HOL-Problems: β



$\stackrel{*}{\doteq}$ is equivalence relation

- $\forall X_\alpha. X \stackrel{*}{\doteq} X$
- $\forall X_\alpha, Y_\alpha. X \stackrel{*}{\doteq} Y \supset Y \stackrel{*}{\doteq} X$
- $\forall X_\alpha, Y_\alpha, Z_\alpha. (X \stackrel{*}{\doteq} Y \wedge Y \stackrel{*}{\doteq} Z) \supset X \stackrel{*}{\doteq} Z$

HOL-Problems: β

$$
\begin{tikzcd}
 & & \mathfrak{M}_{\beta}(\Sigma) \arrow[dll, "\xi"'] \arrow[ddll, "\mathfrak{f}"] \arrow[d, "\eta"] \arrow[drr, "\mathfrak{b}"] & & \\
\mathfrak{M}_{\beta\xi}(\Sigma) \arrow[d, "\eta"] \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\eta}(\Sigma) \arrow[dll, "\xi"'] \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dll, "\xi"'] \arrow[ddll, "\mathfrak{f}"] \arrow[d, "\eta"] \\
\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma) \arrow[d, "\eta"] & & \mathfrak{M}_{\beta\eta\mathfrak{b}}(\Sigma) \arrow[dll, "\xi"'] \\
 & & \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, dotted, "\text{full}"] & & \\
 & & \mathfrak{ST}(\Sigma) & &
\end{tikzcd}
$$

$\stackrel{*}{\doteq}$ is equivalence relation

- $\forall X_\alpha . X \stackrel{*}{\doteq} X$
- $\forall X_\alpha, Y_\alpha . X \stackrel{*}{\doteq} Y \supset Y \stackrel{*}{\doteq} X$
- $\forall X_\alpha, Y_\alpha, Z_\alpha . (X \stackrel{*}{\doteq} Y \wedge Y \stackrel{*}{\doteq} Z) \supset X \stackrel{*}{\doteq} Z$

$\stackrel{*}{\doteq}$ is congruence relation

- $\forall X_\alpha, Y_\alpha, F_{\alpha\alpha} . X \stackrel{*}{\doteq} Y \supset (FX) \stackrel{*}{\doteq} (FY)$
- $\forall X_\alpha, Y_\alpha, P_{o\alpha} . X \stackrel{*}{\doteq} Y \wedge (PX) \supset (PY)$

HOL-Problems: β

$$\begin{tikzcd}
 & & \mathfrak{M}_{\beta}(\Sigma) \arrow[dll, "\xi"'] \arrow[ddll, "\mathfrak{f}"] \arrow[d, "\eta"] \arrow[drr, "\mathfrak{b}"] & & \\
\mathfrak{M}_{\beta\xi}(\Sigma) \arrow[d, "\eta"] \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\eta}(\Sigma) \arrow[dll, "\xi"'] \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dll, "\xi"'] \arrow[ddll, "\mathfrak{f}"] \arrow[d, "\eta"] \\
\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma) \arrow[d, "\eta"] & & \mathfrak{M}_{\beta\eta\mathfrak{b}}(\Sigma) \arrow[dll, "\xi"'] \\
 & & \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, "\text{full}"] & & \\
 & & \mathfrak{ST}(\Sigma) & &
\end{tikzcd}$$

$\stackrel{*}{\doteq}$ **is equivalence relation**

- $\forall X_\alpha . X \stackrel{*}{\doteq} X$
- $\forall X_\alpha, Y_\alpha . X \stackrel{*}{\doteq} Y \supset Y \stackrel{*}{\doteq} X$
- $\forall X_\alpha, Y_\alpha, Z_\alpha . (X \stackrel{*}{\doteq} Y \wedge Y \stackrel{*}{\doteq} Z) \supset X \stackrel{*}{\doteq} Z$

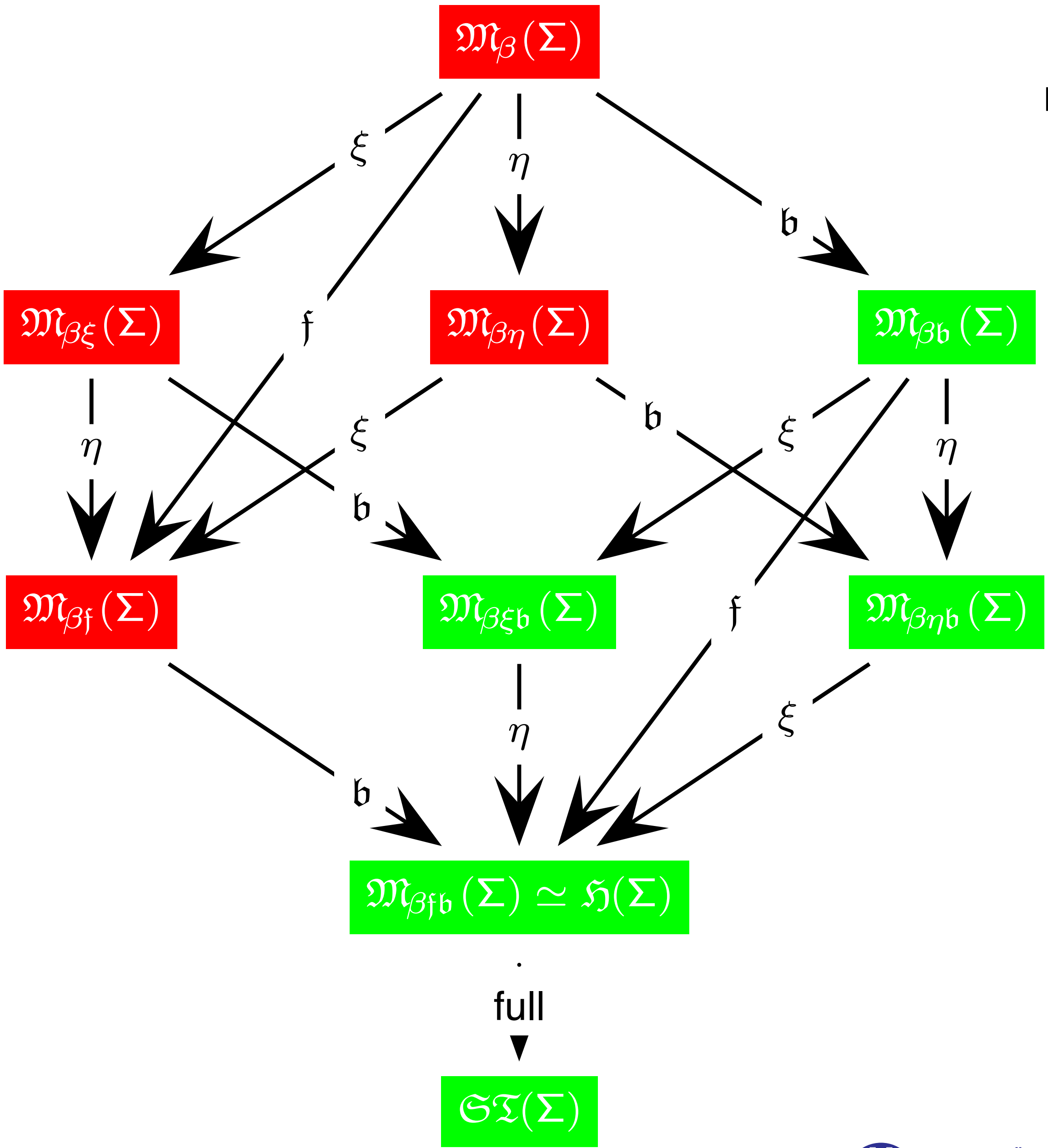
$\stackrel{*}{\doteq}$ **is congruence relation**

- $\forall X_\alpha, Y_\alpha, F_{\alpha\alpha} . X \stackrel{*}{\doteq} Y \supset (FX) \stackrel{*}{\doteq} (FY)$
- $\forall X_\alpha, Y_\alpha, P_{o\alpha} . X \stackrel{*}{\doteq} Y \wedge (PX) \supset (PY)$

Trivial directions of Boolean and functional extensionality

- $\forall A_o, B_o . A \stackrel{*}{\doteq} B \supset (A \Leftrightarrow B)$
- $\forall F_{\beta\alpha}, G_{\beta\alpha} . F \stackrel{*}{\doteq} G \supset (\forall X_\alpha . FX \stackrel{*}{\doteq} GX)$

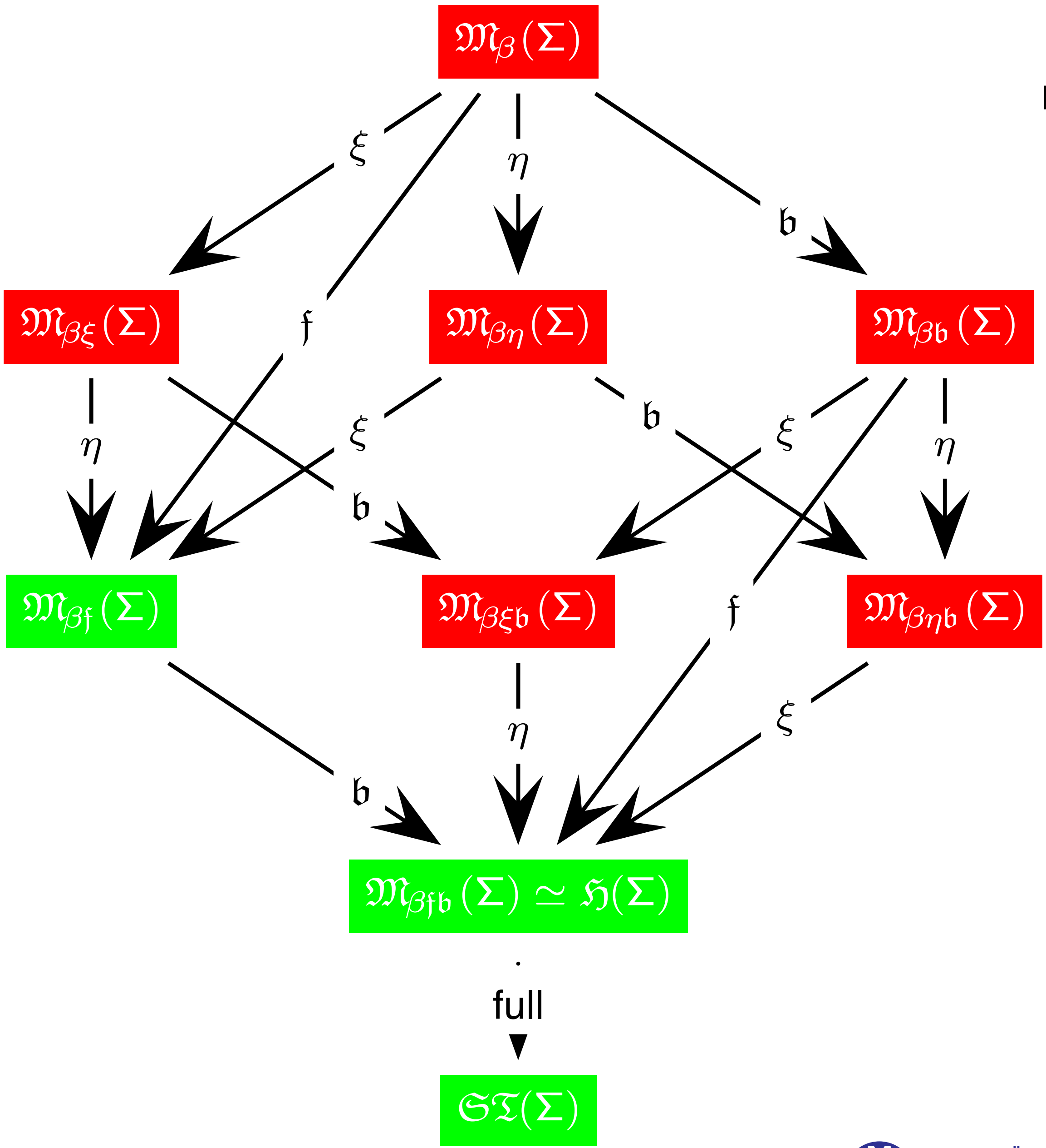
HOL-Problems: $\mathfrak{b}$



Non-trivial direction of Boolean extensionality

- $\forall A_o, B_o. (A \Leftrightarrow B) \supset A \stackrel{*}{\doteq} B$

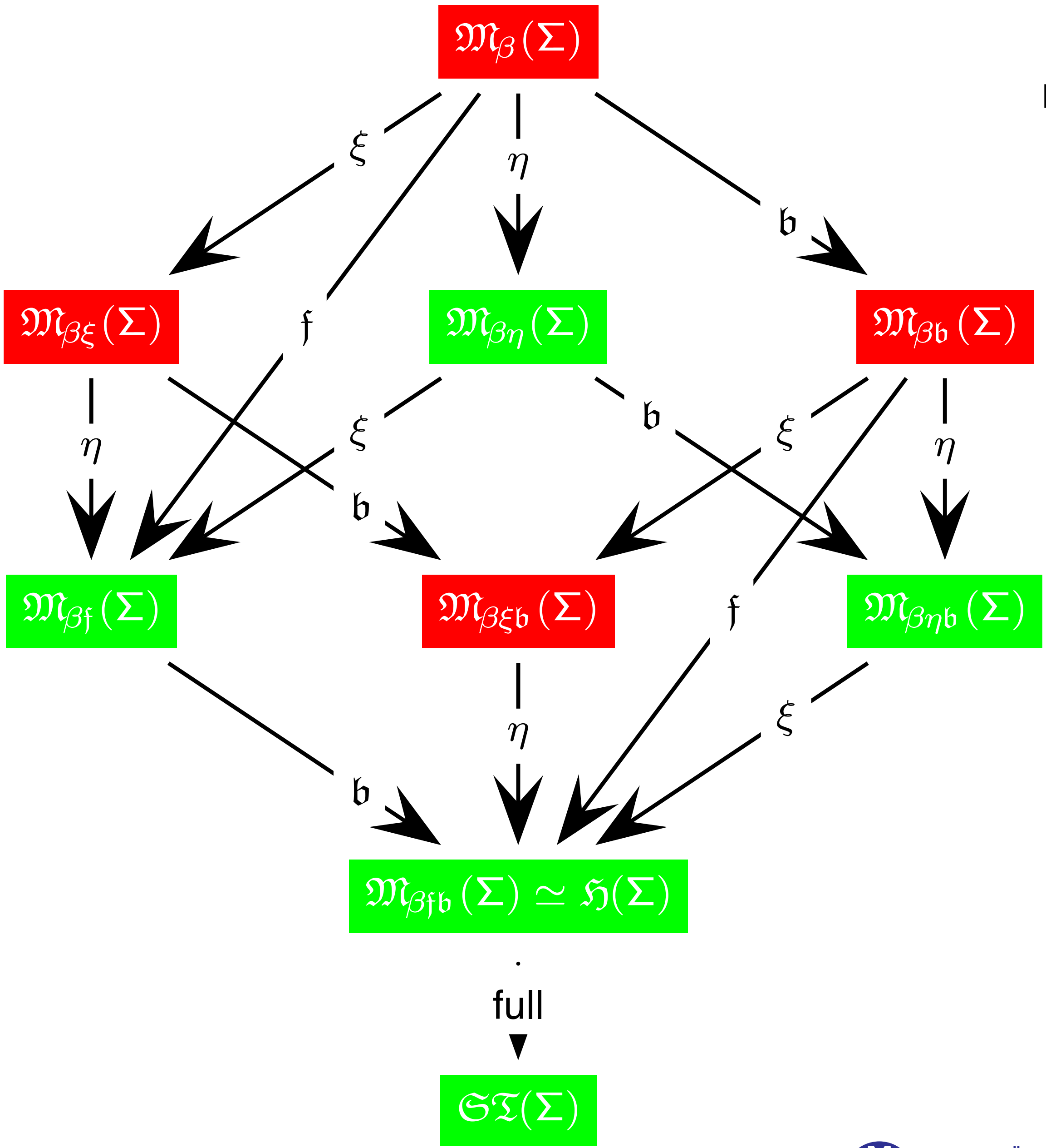
HOL-Problems: $\mathfrak{f}$



Non-trivial direct. of functional extensionality

- $\forall F_{\beta\alpha}, G_{\beta\alpha}.(\forall X_{\alpha}.FX \stackrel{*}{\doteq} GX) \supset F \stackrel{*}{\doteq} G$

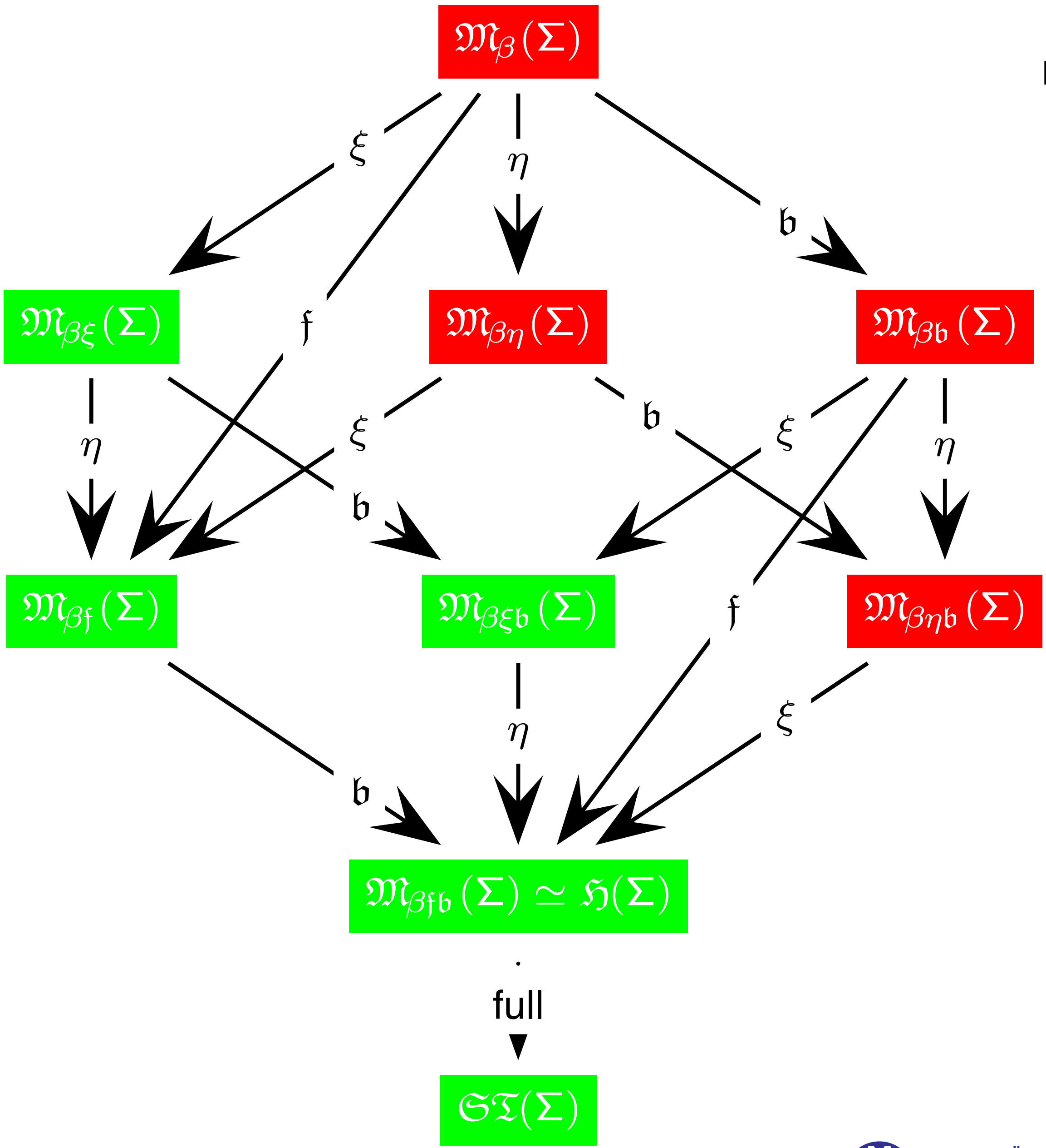
HOL-Problems: η



Example requiring property η

- $(p_{o(\iota\iota)}(\lambda X_\iota . f_{\iota\iota} X)) \supset (p\ f)$

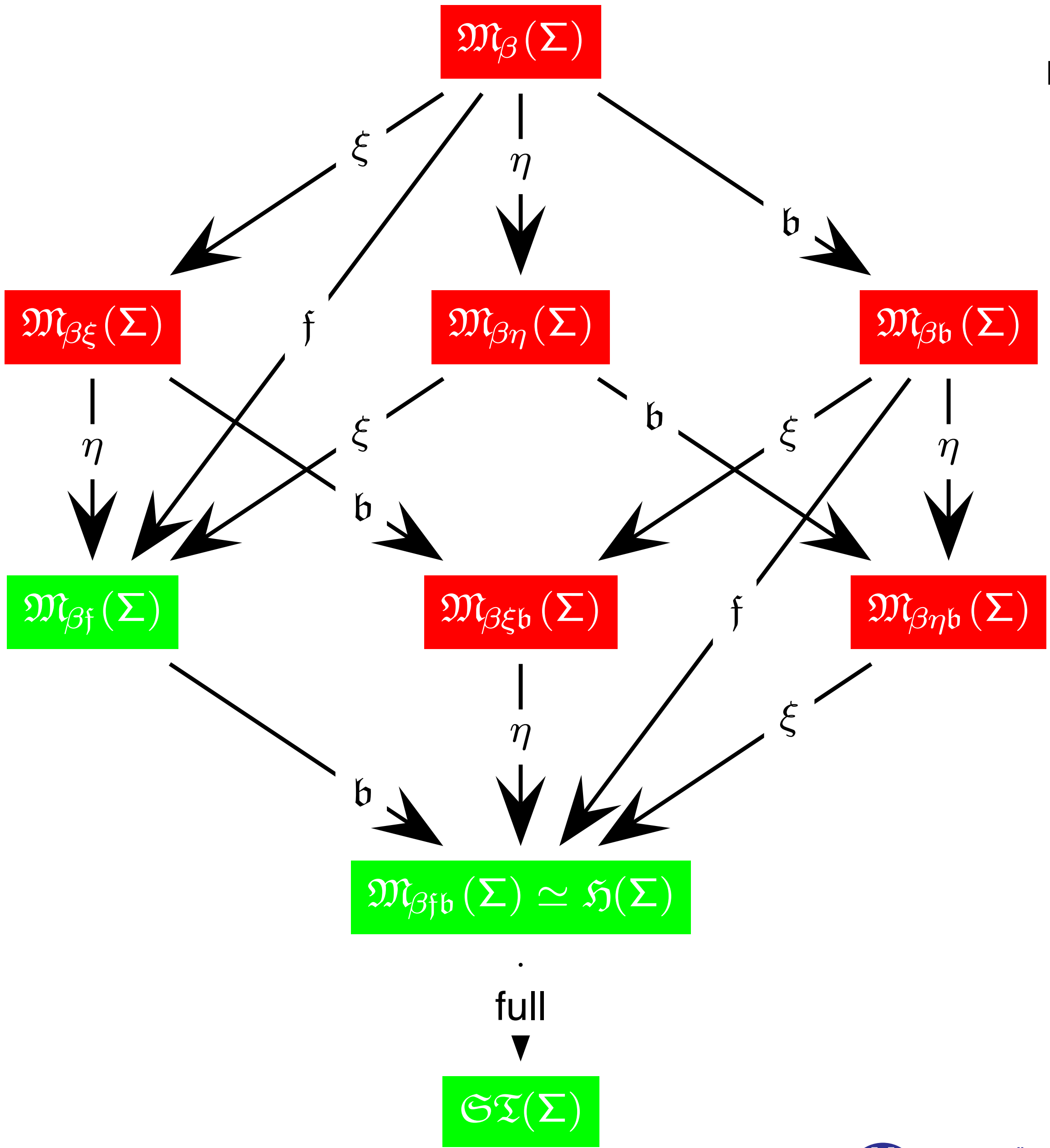
HOL-Problems: ξ



Example requiring property ξ (and q!)

- $(\forall X_\iota.(f_{\iota\iota}X) \stackrel{*}{\doteq} X) \wedge p_{o(\iota\iota)}(\lambda X_\iota.X)$
  $\supset p(\lambda X_\iota.fX)$

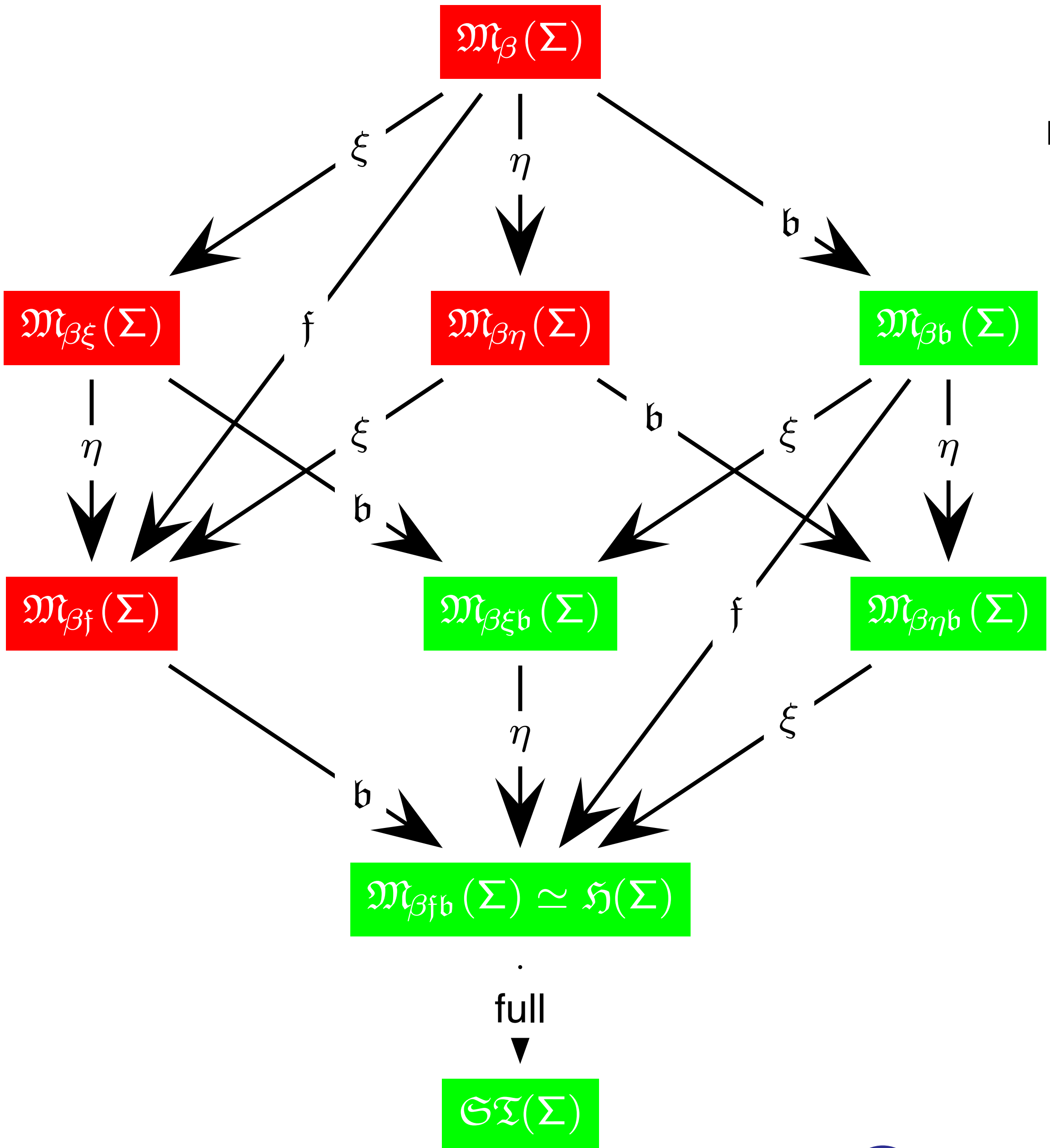
HOL-Problems: $\mathfrak{f}$



Example requiring property $\mathfrak{f}$ (and $\mathfrak{q}$!)

- $(\forall X_\iota.(f_{\iota\iota} X) \stackrel{*}{\doteq} X) \wedge p_{o(\iota\iota)}(\lambda X_\iota.X) \supset (p\ f)$

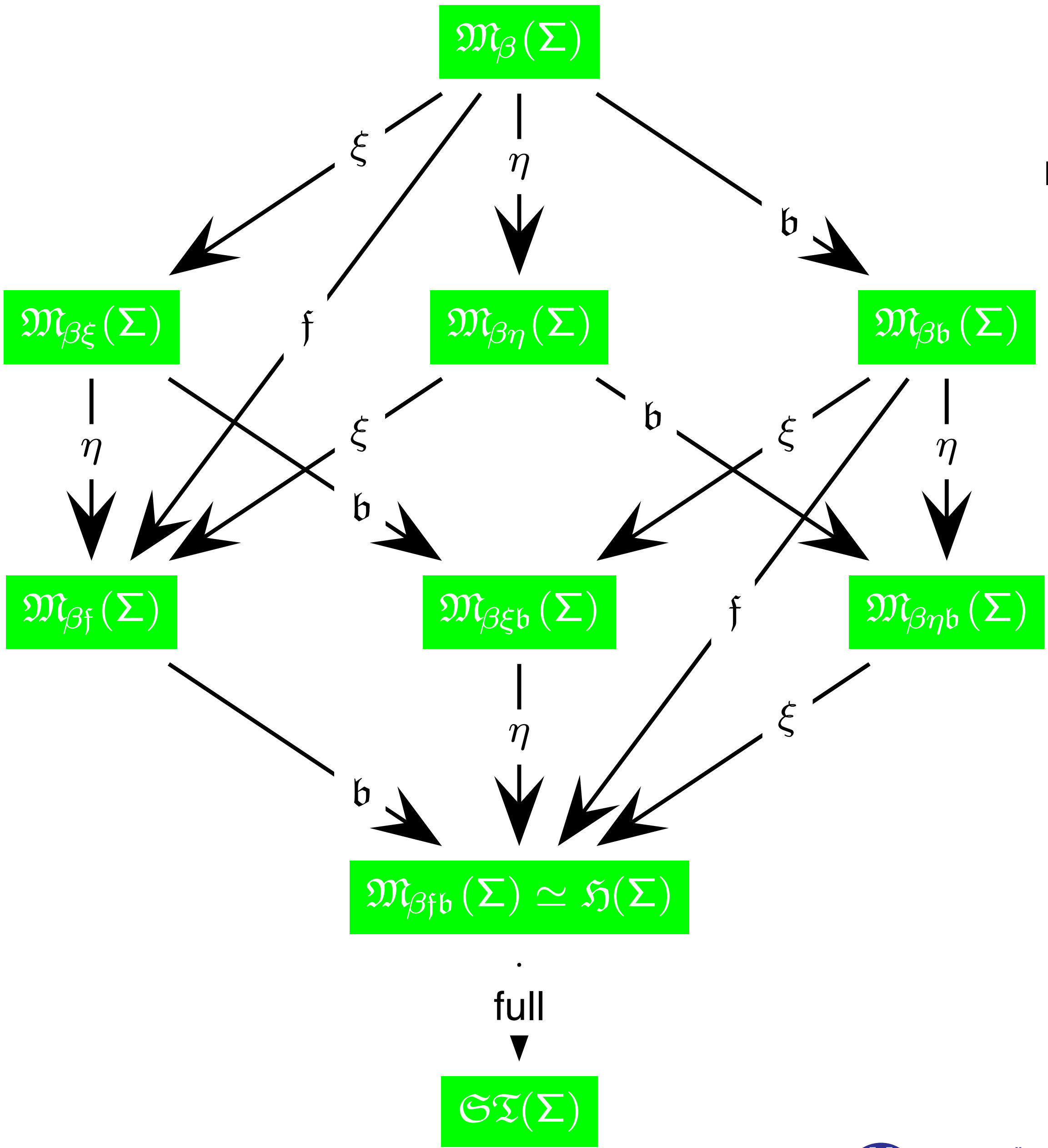
HOL-Problems: $\mathfrak{b}$



Examples requiring property $\mathfrak{b}$

- $(p_{oo}\ a_o) \wedge (p\ b_o) \Rightarrow (p\ (a \wedge b))$
- $\neg(a \stackrel{*}{\doteq} \neg a)$ (in particular $\neg(a = \neg a)$)
- $(h_{\iota o}((h \top) \stackrel{*}{\doteq} (h \bot))) \stackrel{*}{\doteq} (h \bot)$

HOL-Problems: Other Examples

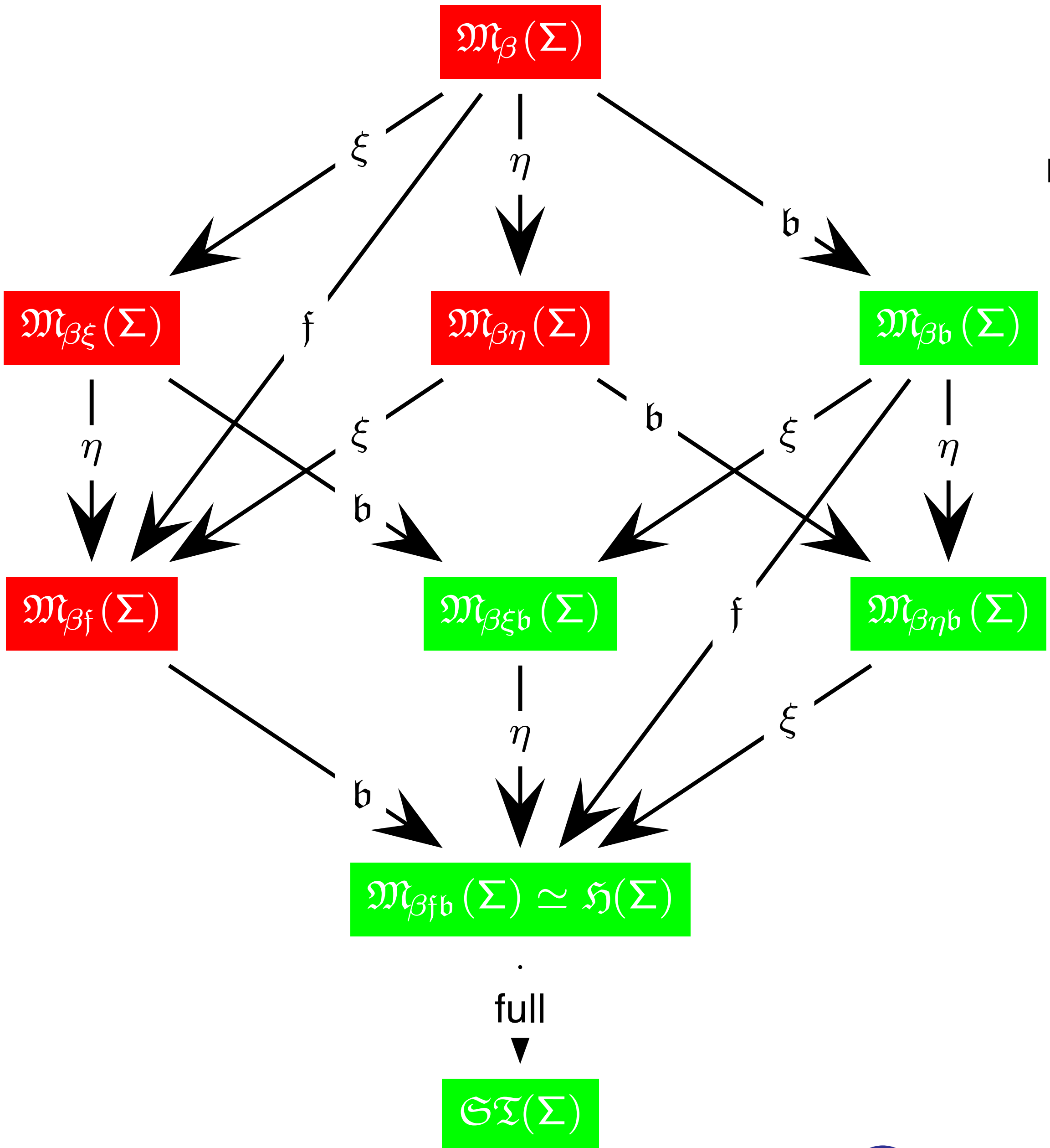


Playing with DeMorgan's Law:

- $\forall X, Y \boldsymbol{.} X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$

'Ok' for all model classes

HOL-Problems: DeMorgan's Law

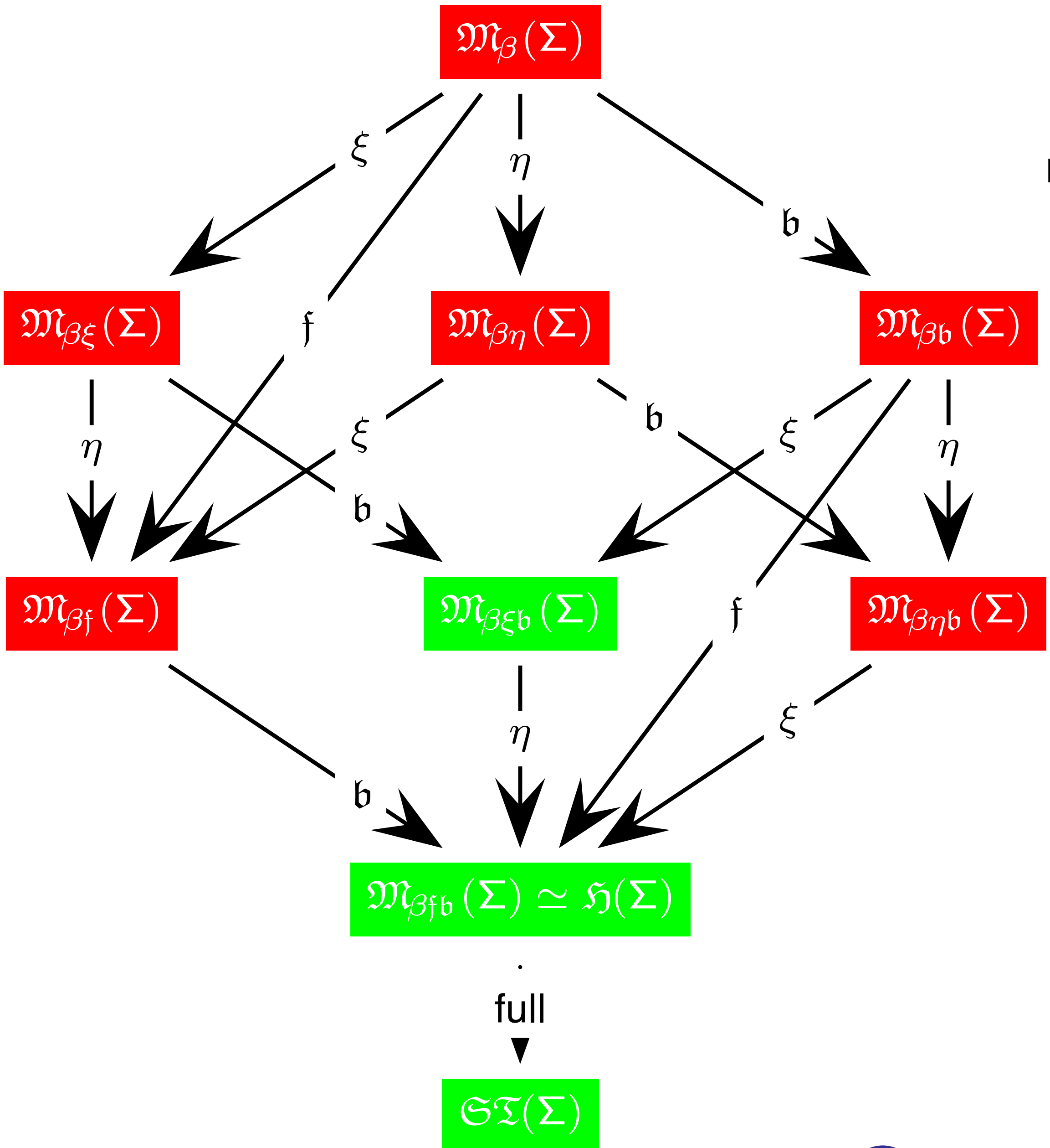


Playing with DeMorgan's Law:

- $\forall X, Y \boldsymbol{.} X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$
- $\forall X, Y \boldsymbol{.} X \wedge Y \stackrel{*}{=} \neg(\neg X \vee \neg Y)$

requires $\mathfrak{b}$

HOL-Problems: DeMorgan's Law

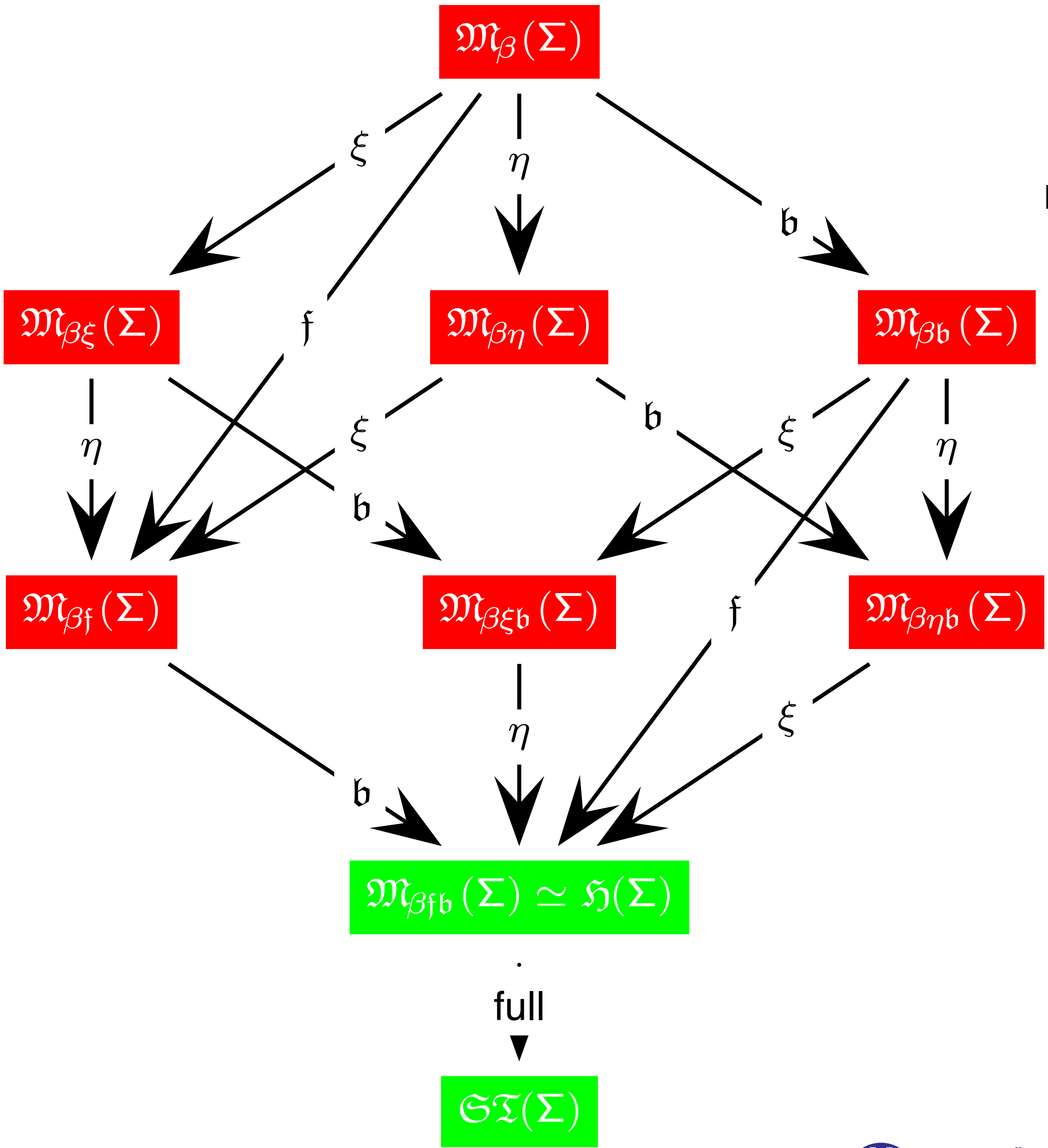


Playing with DeMorgan's Law:

- $\forall X, Y.X \land Y \Leftrightarrow \neg(\neg X \lor \neg Y)$
- $\forall X, Y.X \land Y \stackrel{*}{\doteq} \neg(\neg X \lor \neg Y)$
- $(\lambda U \lambda V.U \land V) \stackrel{*}{\doteq} (\lambda X \lambda Y.\neg(\neg X \lor \neg Y))$

requires $\mathfrak{b}$ and ξ

HOL-Problems: DeMorgan's Law

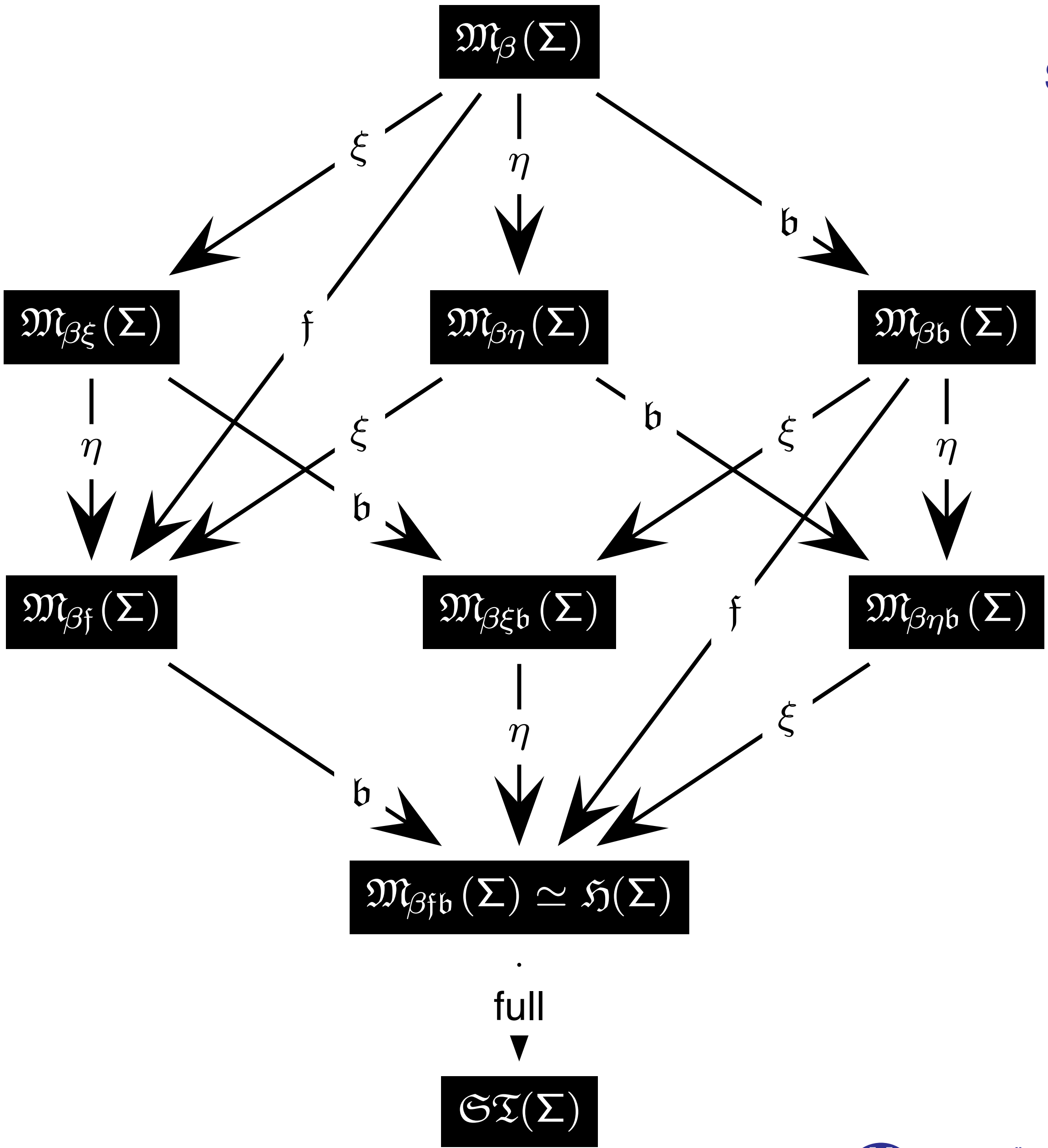


Playing with DeMorgan's Law:

- $\forall X, Y. X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$
- $\forall X, Y. X \wedge Y \stackrel{*}{\doteq} \neg(\neg X \vee \neg Y)$
- $(\lambda U \lambda V. U \wedge V) \stackrel{*}{\doteq} (\lambda X \lambda Y. \neg(\neg X \vee \neg Y))$
- $\wedge \stackrel{*}{\doteq} (\lambda X \lambda Y. \neg(\neg X \vee \neg Y))$

requires $\mathfrak{b}$ and $\mathfrak{f}$

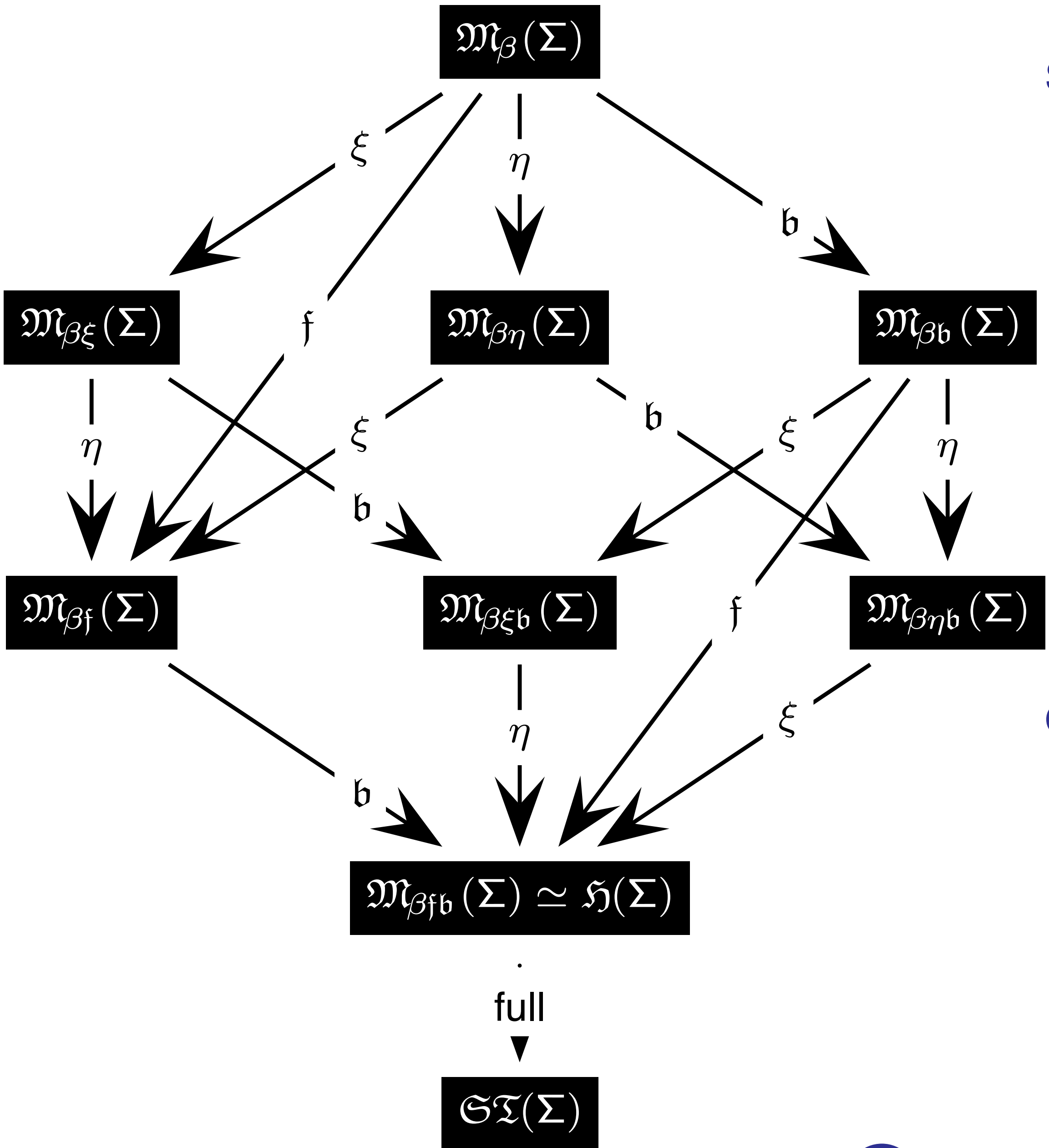
HOL-Problems: Set Comprehension



Set comprehension

- big challenge for automation
- [Benzm.BrownKohlhase-Draft-05] set instantiations can be used to simulate cut-rule if one of the following axioms is given: comprehension, induction, extensionality, choice, description
- dependend on logical constants in Σ

HOL-Problems: Set Comprehension



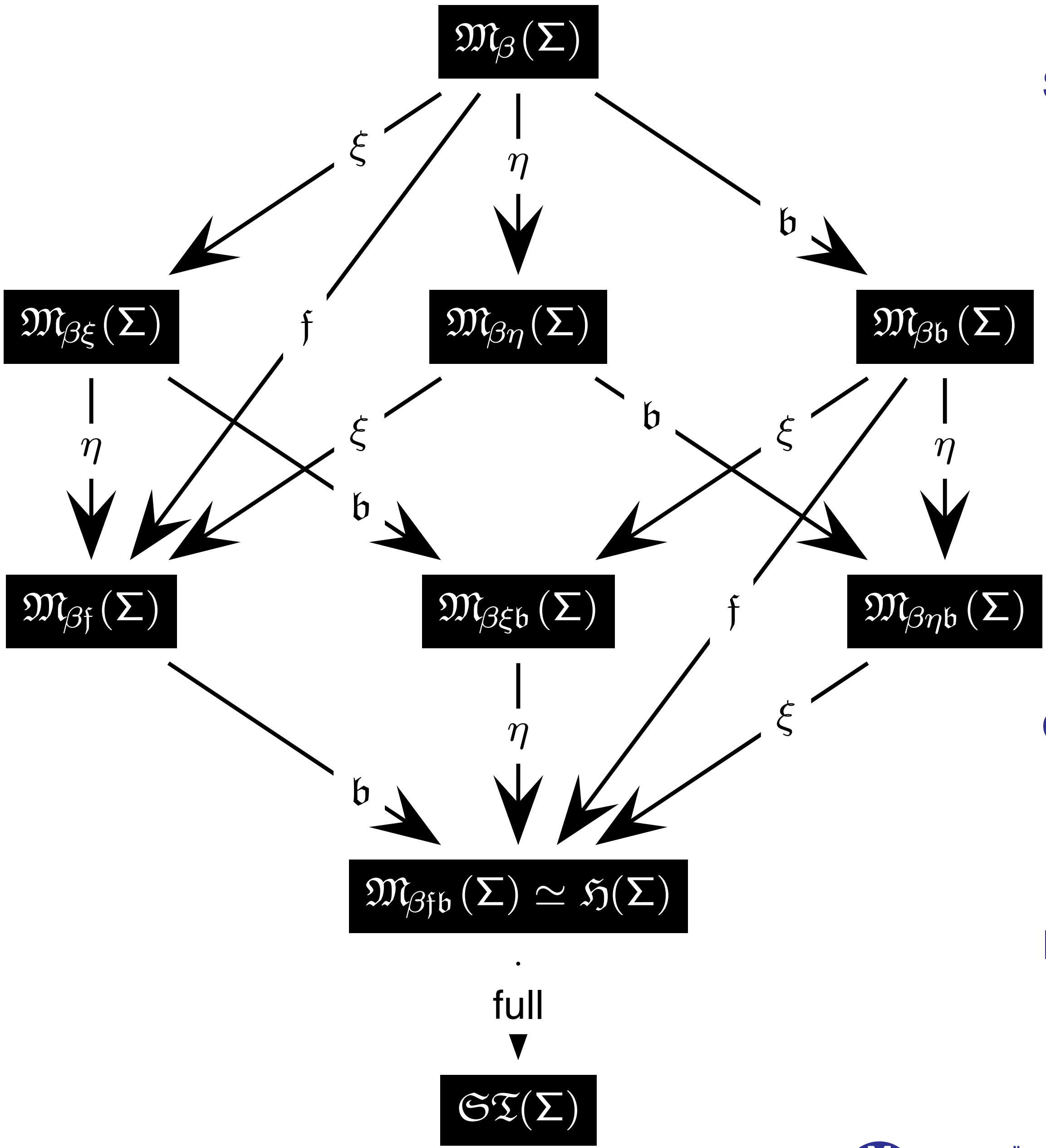
Set comprehension

- big challenge for automation
- [Benzm.BrownKohlhase-Draft-05] set instantiations can be used to simulate cut-rule if one of the following axioms is given: comprehension, induction, extensionality, choice, description
- dependend on logical constants in Σ

On the following slides emphasis on:

- signature Σ varying
- no property $\mathfrak{q}$ assumed

HOL-Problems: Set Comprehension



Set comprehension

- big challenge for automation
- [Benzm.BrownKohlhase-Draft-05] set instantiations can be used to simulate cut-rule if one of the following axioms is given: comprehension, induction, extensionality, choice, description
- dependend on logical constants in Σ

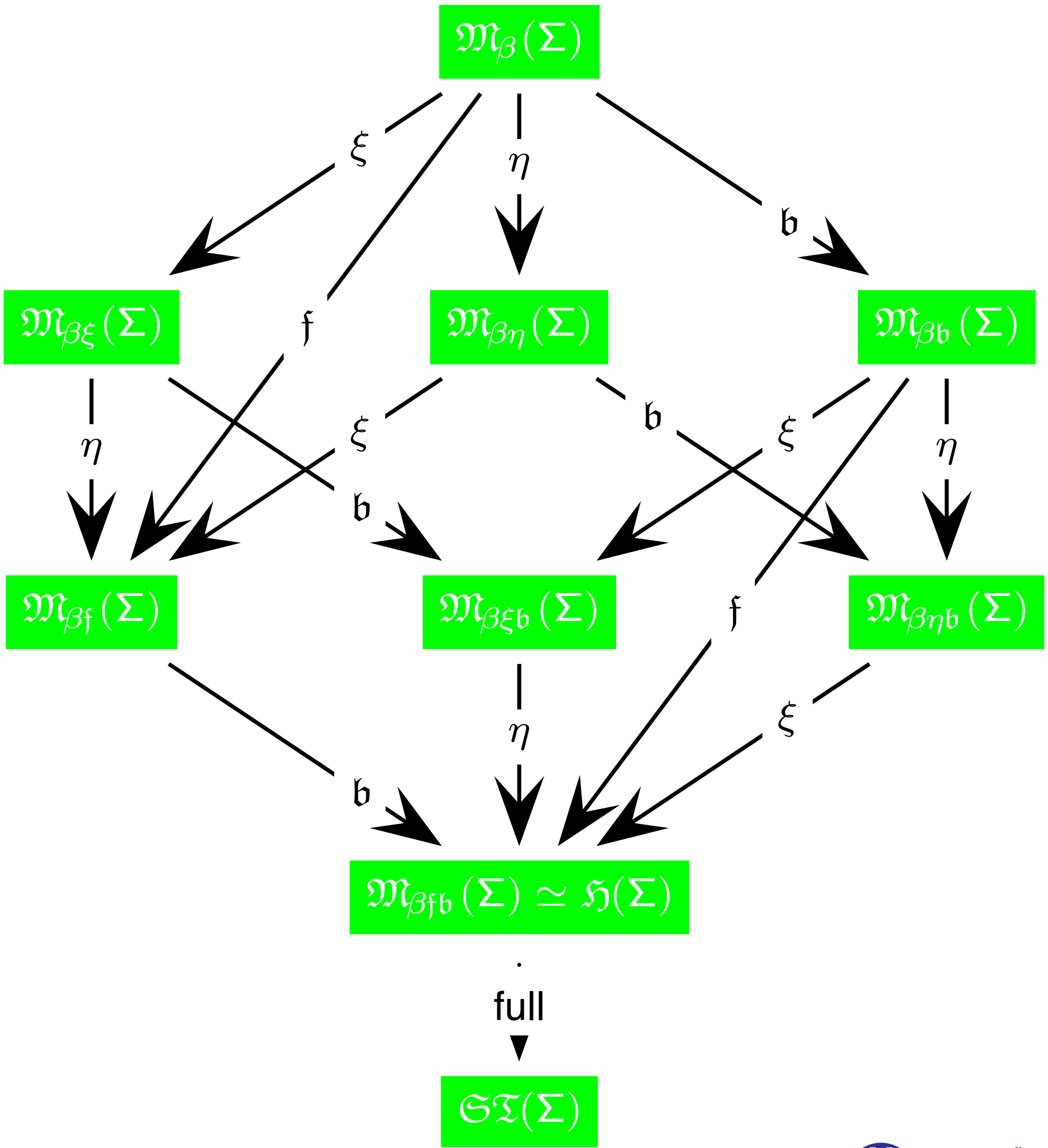
On the following slides emphasis on:

- signature Σ varying
- no property $\mathfrak{q}$ assumed

External vs. internal logical constants

- if $\neg \notin \Sigma$:
  $\neg$ refers to 'external' symbol
  $\mathcal{M} \models \neg A$ means $\mathcal{M} \not\models A$

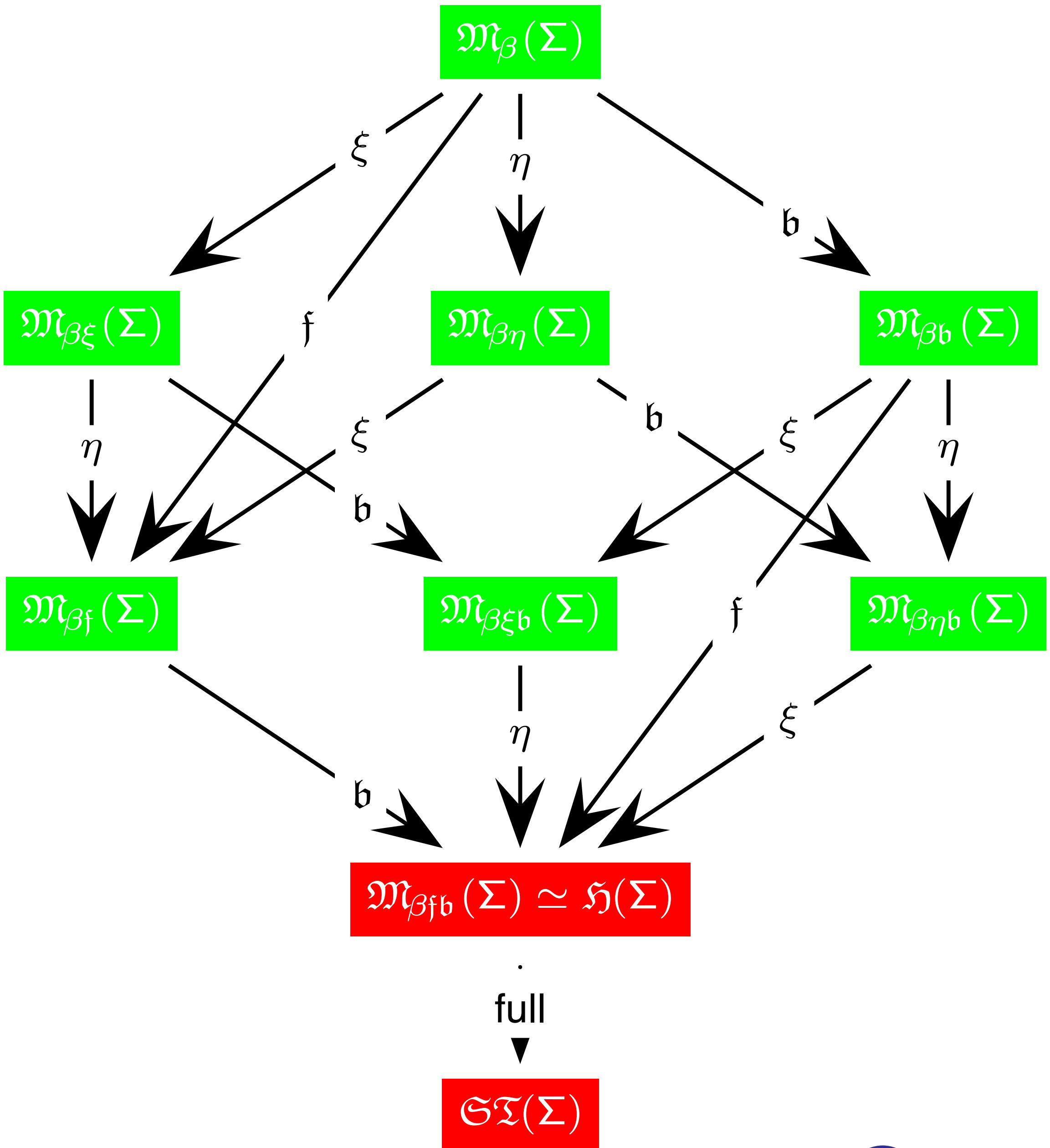
HOL-Problems: Set Comprehension



Set comprehension

- $\exists N_{oo} \forall P_o . NP \Leftrightarrow \neg P$
  - if $\neg \in \Sigma$ or $\{\bot, \supset\} \subseteq \Sigma$ or $\{\bot, \Leftrightarrow\} \subseteq \Sigma$
  - e.g.: $N_{oo} \longleftarrow \lambda X_o . \neg X$
    e.g.: $N_{oo} \longleftarrow \lambda X_o . X \supset \bot$

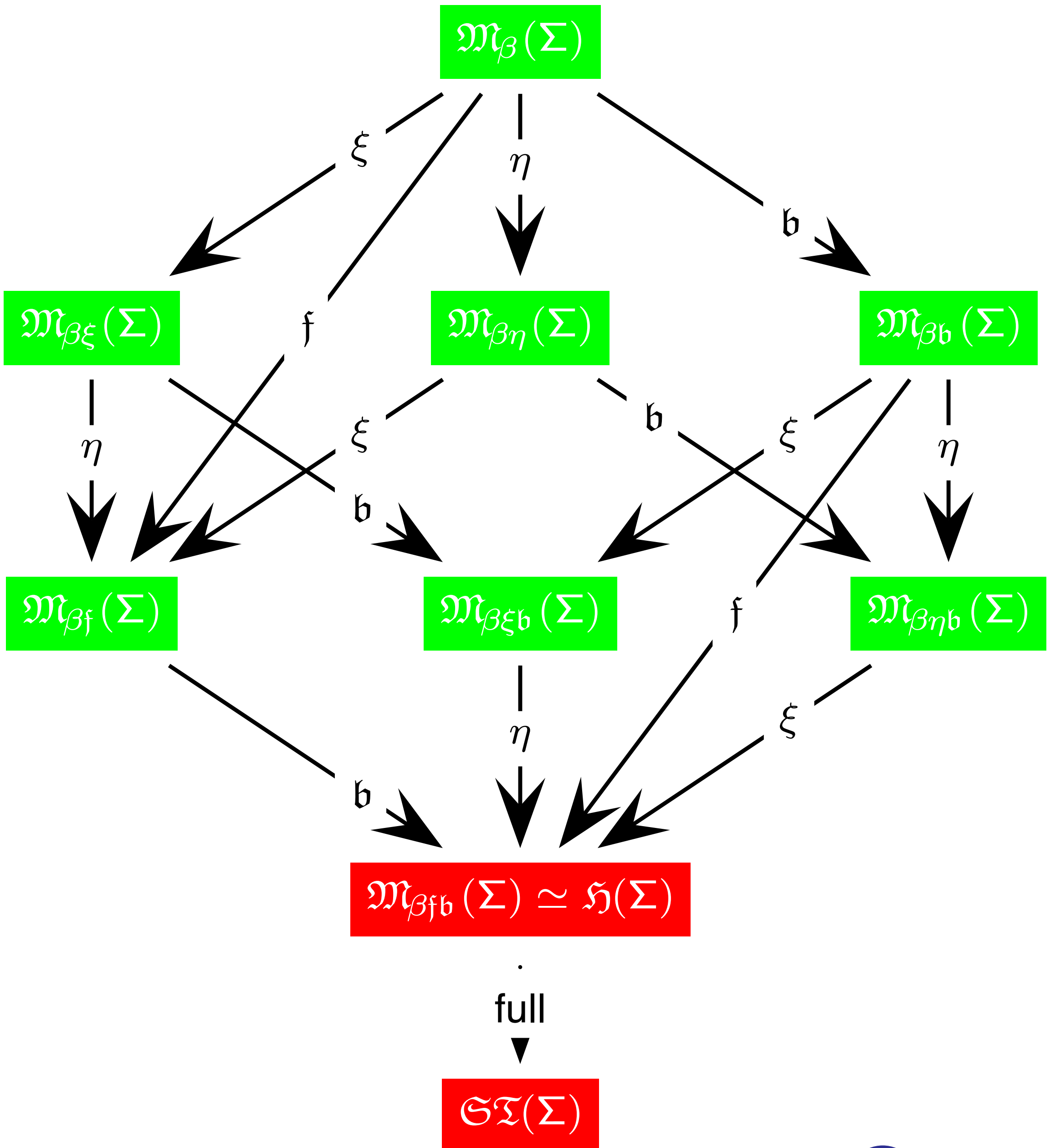
HOL-Problems: Set Comprehension



Set comprehension

- $\exists N_{oo} \forall P_o . N P \Leftrightarrow \neg P$
  - if $\neg \notin \Sigma$ and $\{\bot, \supset\} \not\subseteq \Sigma$ or $\{\bot, \Leftrightarrow\} \not\subseteq \Sigma$

HOL-Problems: Set Comprehension



Set comprehension

- $\exists N_{oo} \forall P_o . NP \Leftrightarrow \neg P$
  - if $\neg \notin \Sigma$ and $\{\bot, \supset\} \not\subseteq \Sigma$ or $\{\bot, \Leftrightarrow\} \not\subseteq \Sigma$

Other examples from [Brown-PhD-04]

- Surjective Cantor Theorem
- Injective Cantor Theorem

Semantics: Examples of Σ-Models

Examples of Σ-Models

We now sketch the construction of models in the model classes $\mathfrak{M}_*(\Sigma)$ to demonstrate concretely how properties for Boolean, strong and weak functional extensionality can fail.

Examples of Σ-Models

We now sketch the construction of models in the model classes $\mathfrak{M}_*(\Sigma)$ to demonstrate concretely how properties for Boolean, strong and weak functional extensionality can fail.

We need this to show that the inclusions of the model classes in our landscape are proper, and we indeed need all of them.

Ex.: Singleton Model

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\mathtt{T}, \mathtt{F}\}$ and $\mathcal{D}_\iota := \{*\}$.

Ex.: Singleton Model

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\mathtt{T}, \mathtt{F}\}$ and $\mathcal{D}_\iota := \{*\}$.
- Easy to define an evaluation function $\mathcal{E}$ for this frame by induction on terms, using functions to interpret λ-abstractions.

Ex.: Singleton Model

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\mathtt{T}, \mathtt{F}\}$ and $\mathcal{D}_\iota := \{*\}$.
- Easy to define an evaluation function $\mathcal{E}$ for this frame by induction on terms, using functions to interpret λ-abstractions.
- The identity function $\upsilon: \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ is a valuation, assuming the logical constants are interpreted in the standard way.

Ex.: Singleton Model

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\mathtt{T}, \mathtt{F}\}$ and $\mathcal{D}_\iota := \{*\}$.
- Easy to define an evaluation function $\mathcal{E}$ for this frame by induction on terms, using functions to interpret λ-abstractions.
- The identity function $\upsilon : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ is a valuation, assuming the logical constants are interpreted in the standard way.
- Thus, $\mathcal{M}^{\beta\mathfrak{fb}} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ defines a Σ-model.

Ex.: Singleton Model

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\mathtt{T}, \mathtt{F}\}$ and $\mathcal{D}_\iota := \{*\}$.
- Easy to define an evaluation function $\mathcal{E}$ for this frame by induction on terms, using functions to interpret λ-abstractions.
- The identity function $\upsilon\colon \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ is a valuation, assuming the logical constants are interpreted in the standard way.
- Thus, $\mathcal{M}^{\beta\mathfrak{f}\mathfrak{b}} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ defines a Σ-model.
- This model satisfies properties $\mathfrak{b}$, $\mathfrak{f}$ (hence η and ξ) and $\mathfrak{q}$ (since the frame is full).

Ex.: Singleton Model

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\mathtt{T}, \mathtt{F}\}$ and $\mathcal{D}_\iota := \{*\}$.
- Easy to define an evaluation function $\mathcal{E}$ for this frame by induction on terms, using functions to interpret λ-abstractions.
- The identity function $\upsilon\colon \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ is a valuation, assuming the logical constants are interpreted in the standard way.
- Thus, $\mathcal{M}^{\beta\mathfrak{fb}} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ defines a Σ-model.
- This model satisfies properties $\mathfrak{b}$, $\mathfrak{f}$ (hence η and ξ) and $\mathfrak{q}$ (since the frame is full).
- So, $\mathcal{M}^{\beta\mathfrak{fb}} \in \mathfrak{ST}(\Sigma) \subseteq \mathfrak{H}(\Sigma) \subseteq \mathfrak{M}_{\beta\mathfrak{fb}}(\Sigma) \subseteq \ldots$.

Ex.: Singleton Model

$$
\begin{tikzcd}
 & \mathfrak{M}_{\beta}(\Sigma) \arrow[dl, "\xi"'] \arrow[d, "\eta"] \arrow[dr, "\mathfrak{b}"] \arrow[ddl, "\mathfrak{f}"] & \\
\mathfrak{M}_{\beta\xi}(\Sigma) \arrow[d, "\eta"] \arrow[dr, "\mathfrak{b}"] & \mathfrak{M}_{\beta\eta}(\Sigma) \arrow[dl, "\xi"'] \arrow[dr, "\mathfrak{b}"] & \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dl, "\xi"'] \arrow[d, "\eta"] \arrow[ddl, "\mathfrak{f}"] \\
\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \arrow[dr, "\mathfrak{b}"] & \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma) \arrow[d, "\eta"] & \mathfrak{M}_{\beta\eta\mathfrak{b}}(\Sigma) \arrow[dl, "\xi"'] \\
 & \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, "\text{full}"] & \\
 & \mathfrak{S}\mathfrak{T}(\Sigma) &
\end{tikzcd}
$$

Ex.: Model without Boolean Extensionality

- Assume Σ contains only the connectives $\neg, \vee, \Pi^{\alpha}$; other connectives defined as usual, e.g., $\forall X, Y. X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$.

Ex.: Model without Boolean Extensionality

- Assume Σ contains only the connectives $\neg, \vee, \Pi^\alpha$; other connectives defined as usual, e.g., $\forall X, Y. X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$.
- Choose $(\mathcal{D}, @)$ as full frame with $\mathcal{D}_o = \{a, b, c\}$ and $\mathcal{D}_\iota = \{0, 1\}$.

Ex.: Model without Boolean Extensionality

- Assume Σ contains only the connectives $\neg, \vee, \Pi^\alpha$; other connectives defined as usual, e.g., $\forall X, Y. X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$.
- Choose $(\mathcal{D}, @)$ as full frame with $\mathcal{D}_o = \{a, b, c\}$ and $\mathcal{D}_\iota = \{0, 1\}$.
- We define evaluation function $\mathcal{E}$ for this frame by defining $\mathcal{E}(\neg)$, $\mathcal{E}(\vee)$, and $\mathcal{E}(\Pi^\alpha)$:

$\mathcal{E}(\neg)$	a	b	c
	c	c	a

$\mathcal{E}(\vee)$	a	b	c
a	a	a	a
b	a	a	a
c	a	a	c

$$\mathcal{E}(\Pi^\alpha)@f = \begin{cases} a, & \text{if } f@g \in \{a, b\} \text{ for all } g \in \mathcal{D}_\alpha \\ c, & \text{if } f@g = c \text{ for some } g \in \mathcal{D}_\alpha \end{cases}$$

Ex.: Model without Boolean Extensionality

- Assume Σ contains only the connectives $\neg, \vee, \Pi^\alpha$; other connectives defined as usual, e.g., $\forall X, Y. X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$.
- Choose $(\mathcal{D}, @)$ as full frame with $\mathcal{D}_o = \{a, b, c\}$ and $\mathcal{D}_\iota = \{0, 1\}$.
- We define evaluation function $\mathcal{E}$ for this frame by defining $\mathcal{E}(\neg)$, $\mathcal{E}(\vee)$, and $\mathcal{E}(\Pi^\alpha)$:

$\mathcal{E}(\neg)$	a	b	c
	c	c	a

$\mathcal{E}(\vee)$	a	b	c
a	a	a	a
b	a	a	a
c	a	a	c

$$\mathcal{E}(\Pi^\alpha)@f = \begin{cases} a, & \text{if } f@g \in \{a, b\} \text{ for all } g \in \mathcal{D}_\alpha \\ c, & \text{if } f@g = c \text{ for some } g \in \mathcal{D}_\alpha \end{cases}$$

- We can choose $\mathcal{E}(w)$ to be arbitrary for parameters $w \in \Sigma$.

Ex.: Model without Boolean Extensionality

- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines $\mathcal{E}$ on all formulae.

Ex.: Model without Boolean Extensionality

- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines $\mathcal{E}$ on all formulae.
- Since the frame is full, we are guaranteed that there will be enough functions to interpret λ-abstractions.

Ex.: Model without Boolean Extensionality

- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines $\mathcal{E}$ on all formulae.
- Since the frame is full, we are guaranteed that there will be enough functions to interpret λ-abstractions.
- Let $\upsilon : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be defined by $\upsilon(\mathsf{a}) := \mathtt{T}$, $\upsilon(\mathsf{b}) := \mathtt{T}$ and $\upsilon(\mathsf{c}) := \mathtt{F}$.

Ex.: Model without Boolean Extensionality

- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines $\mathcal{E}$ on all formulae.
- Since the frame is full, we are guaranteed that there will be enough functions to interpret λ-abstractions.
- Let $\upsilon: \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be defined by $\upsilon(\mathsf{a}) := \mathtt{T}$, $\upsilon(\mathsf{b}) := \mathtt{T}$ and $\upsilon(\mathsf{c}) := \mathtt{F}$.
- Easy to check that $\mathcal{M}^{\beta\mathfrak{f}} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ is indeed a Σ-model.

Ex.: Model without Boolean Extensionality

- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines $\mathcal{E}$ on all formulae.
- Since the frame is full, we are guaranteed that there will be enough functions to interpret λ-abstractions.
- Let $\upsilon : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be defined by $\upsilon(\mathsf{a}) := \mathtt{T}$, $\upsilon(\mathsf{b}) := \mathtt{T}$ and $\upsilon(\mathsf{c}) := \mathtt{F}$.
- Easy to check that $\mathcal{M}^{\beta\mathfrak{f}} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ is indeed a Σ-model.
- Since $\mathcal{M}^{\beta\mathfrak{f}}$ is a model over a frame it satisfies property $\mathfrak{f}$.

Ex.: Model without Boolean Extensionality

- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines $\mathcal{E}$ on all formulae.
- Since the frame is full, we are guaranteed that there will be enough functions to interpret λ-abstractions.
- Let $\upsilon \colon \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be defined by $\upsilon(\mathsf{a}) := \mathtt{T}$, $\upsilon(\mathsf{b}) := \mathtt{T}$ and $\upsilon(\mathsf{c}) := \mathtt{F}$.
- Easy to check that $\mathcal{M}^{\beta\mathfrak{f}} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ is indeed a Σ-model.
- Since $\mathcal{M}^{\beta\mathfrak{f}}$ is a model over a frame it satisfies property $\mathfrak{f}$.
- Since this frame is full, we know property $\mathfrak{q}$ holds.

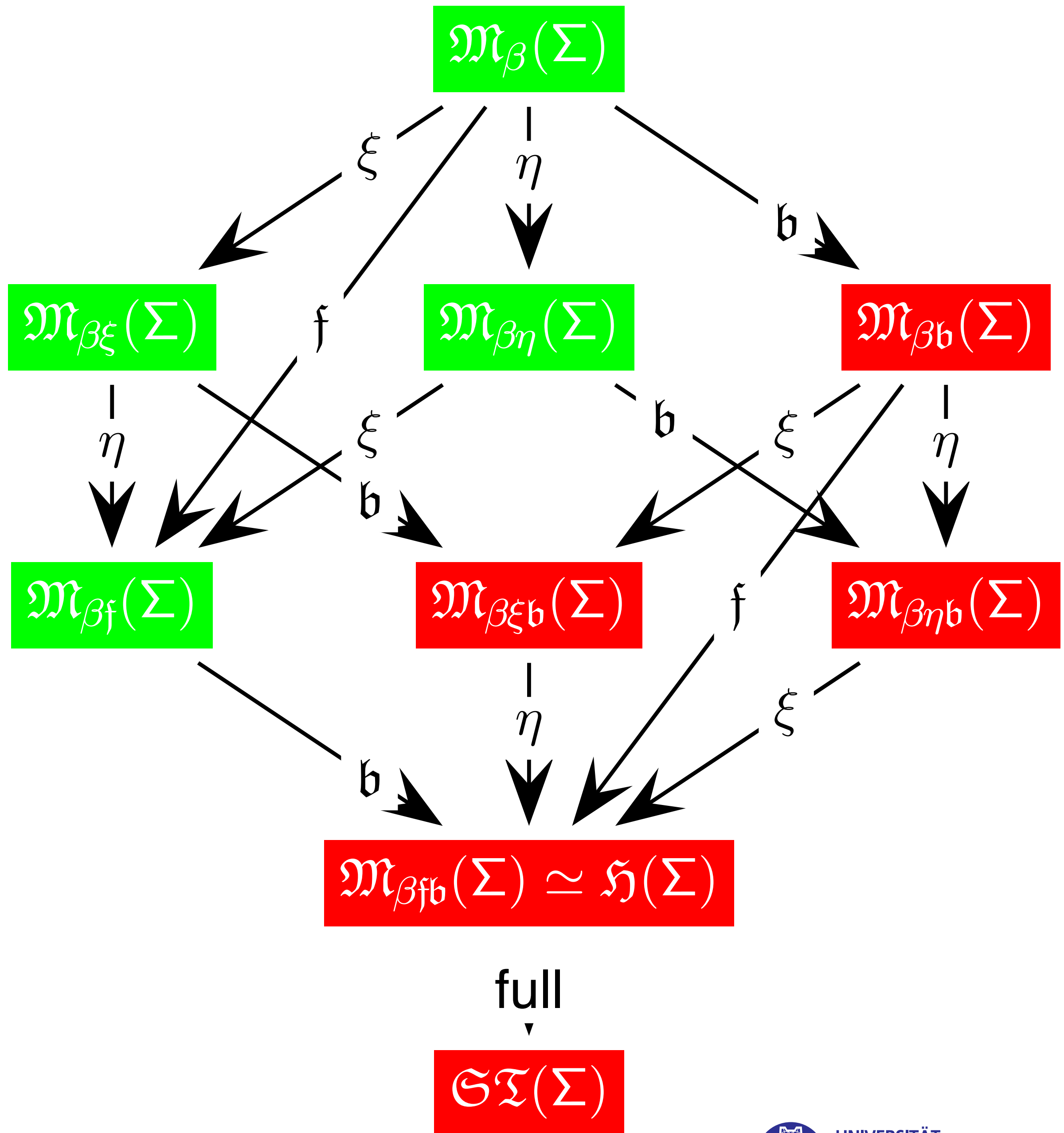
Ex.: Model without Boolean Extensionality

- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines $\mathcal{E}$ on all formulae.
- Since the frame is full, we are guaranteed that there will be enough functions to interpret λ-abstractions.
- Let $\upsilon : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be defined by $\upsilon(\mathsf{a}) := \mathtt{T}$, $\upsilon(\mathsf{b}) := \mathtt{T}$ and $\upsilon(\mathsf{c}) := \mathtt{F}$.
- Easy to check that $\mathcal{M}^{\beta\mathfrak{f}} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ is indeed a Σ-model.
- Since $\mathcal{M}^{\beta\mathfrak{f}}$ is a model over a frame it satisfies property $\mathfrak{f}$.
- Since this frame is full, we know property $\mathfrak{q}$ holds.
- Clearly property $\mathfrak{b}$ fails.

Ex.: Model without Boolean Extensionality

- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines $\mathcal{E}$ on all formulae.
- Since the frame is full, we are guaranteed that there will be enough functions to interpret λ-abstractions.
- Let $\upsilon : \mathcal{D}_o \longrightarrow \{\mathtt{T}, \mathtt{F}\}$ be defined by $\upsilon(\mathsf{a}) := \mathtt{T}$, $\upsilon(\mathsf{b}) := \mathtt{T}$ and $\upsilon(\mathsf{c}) := \mathtt{F}$.
- Easy to check that $\mathcal{M}^{\beta\mathfrak{f}} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ is indeed a Σ-model.
- Since $\mathcal{M}^{\beta\mathfrak{f}}$ is a model over a frame it satisfies property $\mathfrak{f}$.
- Since this frame is full, we know property $\mathfrak{q}$ holds.
- Clearly property $\mathfrak{b}$ fails.
- So, $\mathcal{M}^{\beta\mathfrak{f}} \in \mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \setminus \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$.

Ex.: Model without Boolean Extensionality



Ex.: Model without Boolean Extensionality

In the previous model one can easily verify, if $\mathsf{d} := \mathcal{E}_\varphi(\mathbf{D}_o)$ and $\mathsf{e} := \mathcal{E}_\varphi(\mathbf{E}_o)$, then the values $\mathcal{E}_\varphi(\mathbf{D} \wedge \mathbf{E})$, $\mathcal{E}_\varphi(\mathbf{D} \Rightarrow \mathbf{E})$, and $\mathcal{E}_\varphi(\mathbf{D} \Leftrightarrow \mathbf{E})$ are given by the following tables:

		e:		
$\mathcal{E}(\mathbf{D} \wedge \mathbf{E})$		a	b	c
d:	a	a	a	c
	b	a	a	c
	c	c	c	c

		e:		
$\mathcal{E}(\mathbf{D} \Rightarrow \mathbf{E})$		a	b	c
d:	a	a	a	c
	b	a	a	c
	c	a	a	a

		e:		
$\mathcal{E}(\mathbf{D} \Leftrightarrow \mathbf{E})$		a	b	c
d:	a	a	a	c
	b	a	a	c
	c	c	c	a

Now we show that one can properly model the woodchuck/groundhog example.

Ex.: Groundhogs and Woodchucks

- Let $\mathcal{M}^{\beta\mathfrak{f}}$ be given as above and suppose $\mathsf{woodchuck}_{\iota\to o}$, $\mathsf{groundhog}_{\iota\to o}$, john_{ι}, and phil_{ι} are in the signature Σ.

Ex.: Groundhogs and Woodchucks

- Let $\mathcal{M}^{\beta\mathfrak{f}}$ be given as above and suppose $\mathsf{woodchuck}_{\iota\rightarrow o}$, $\mathsf{groundhog}_{\iota\rightarrow o}$, john_{ι}, and phil_{ι} are in the signature Σ.
- Let $\mathcal{E}(\mathsf{phil}) := 0$ and $\mathcal{E}(\mathsf{john}) := 1$.

Ex.: Groundhogs and Woodchucks

- Let $\mathcal{M}^{\beta\mathfrak{f}}$ be given as above and suppose $\mathsf{woodchuck}_{\iota\to o}$, $\mathsf{groundhog}_{\iota\to o}$, john_ι, and phil_ι are in the signature Σ.
- Let $\mathcal{E}(\mathsf{phil}) := 0$ and $\mathcal{E}(\mathsf{john}) := 1$.
- Let $\mathcal{E}(\mathsf{woodchuck})$ be the function $\mathsf{w} \in \mathcal{D}_{\iota\to o}$ with $\mathsf{w}(0) = \mathsf{b}$ and $\mathsf{w}(1) = \mathsf{c}$.

Ex.: Groundhogs and Woodchucks

- Let $\mathcal{M}^{\beta\mathfrak{f}}$ be given as above and suppose $\mathsf{woodchuck}_{\iota\rightarrow o}$, $\mathsf{groundhog}_{\iota\rightarrow o}$, john_{ι}, and phil_{ι} are in the signature Σ.
- Let $\mathcal{E}(\mathsf{phil}) := 0$ and $\mathcal{E}(\mathsf{john}) := 1$.
- Let $\mathcal{E}(\mathsf{woodchuck})$ be the function $\mathsf{w} \in \mathcal{D}_{\iota\rightarrow o}$ with $\mathsf{w}(0) = \mathsf{b}$ and $\mathsf{w}(1) = \mathsf{c}$.
- Let $\mathcal{E}(\mathsf{groundhog})$ be the function $\mathsf{g} \in \mathcal{D}_{\iota\rightarrow o}$ with $\mathsf{g}(0) = \mathsf{a}$ and $\mathsf{g}(1) = \mathsf{c}$.

Ex.: Groundhogs and Woodchucks

- Let $\mathcal{M}^{\beta\mathfrak{f}}$ be given as above and suppose $\mathsf{woodchuck}_{\iota\to o}$, $\mathsf{groundhog}_{\iota\to o}$, john_{ι}, and phil_{ι} are in the signature Σ.
- Let $\mathcal{E}(\mathsf{phil}) := 0$ and $\mathcal{E}(\mathsf{john}) := 1$.
- Let $\mathcal{E}(\mathsf{woodchuck})$ be the function $\mathsf{w} \in \mathcal{D}_{\iota\to o}$ with $\mathsf{w}(0) = \mathsf{b}$ and $\mathsf{w}(1) = \mathsf{c}$.
- Let $\mathcal{E}(\mathsf{groundhog})$ be the function $\mathsf{g} \in \mathcal{D}_{\iota\to o}$ with $\mathsf{g}(0) = \mathsf{a}$ and $\mathsf{g}(1) = \mathsf{c}$.
- One can show that the sentence $\forall X_{\iota}.(\mathsf{woodchuck}\ X) \Leftrightarrow (\mathsf{groundhog}\ X)$ is valid.

Ex.: Groundhogs and Woodchucks

- Let $\mathcal{M}^{\beta\mathfrak{f}}$ be given as above and suppose $\mathsf{woodchuck}_{\iota\rightarrow o}$, $\mathsf{groundhog}_{\iota\rightarrow o}$, john_{ι}, and phil_{ι} are in the signature Σ.
- Let $\mathcal{E}(\mathsf{phil}) := 0$ and $\mathcal{E}(\mathsf{john}) := 1$.
- Let $\mathcal{E}(\mathsf{woodchuck})$ be the function $\mathsf{w} \in \mathcal{D}_{\iota\rightarrow o}$ with $\mathsf{w}(0) = \mathsf{b}$ and $\mathsf{w}(1) = \mathsf{c}$.
- Let $\mathcal{E}(\mathsf{groundhog})$ be the function $\mathsf{g} \in \mathcal{D}_{\iota\rightarrow o}$ with $\mathsf{g}(0) = \mathsf{a}$ and $\mathsf{g}(1) = \mathsf{c}$.
- One can show that the sentence $\forall X_{\iota}.(\mathsf{woodchuck}\ X) \Leftrightarrow (\mathsf{groundhog}\ X)$ is valid.
- Also, $\mathcal{E}(\mathsf{woodchuck}\ \mathsf{phil}) = \mathsf{b}$ and $\mathcal{E}(\mathsf{groundhog}\ \mathsf{phil}) = \mathsf{a}$, so the propositions $(\mathsf{woodchuck}\ \mathsf{phil})$ and $(\mathsf{groundhog}\ \mathsf{phil})$ are valid.

Ex.: Groundhogs and Woodchucks

- Suppose $\mathsf{believe}_{\iota\longrightarrow o\longrightarrow o} \in \Sigma$ and $\mathcal{E}(\mathsf{believe})$ is the (Curried) function $\mathsf{bel} \in \mathcal{D}_{\iota\longrightarrow o\longrightarrow o}$ such that $\mathsf{bel}(1)(\mathsf{b}) = \mathsf{b}$ and $\mathsf{bel}(1)(\mathsf{a}) = \mathsf{bel}(1)(\mathsf{c}) = \mathsf{bel}(0)(\mathsf{a}) = \mathsf{bel}(0)(\mathsf{b}) = \mathsf{bel}(0)(\mathsf{c}) = \mathsf{c}$.

Ex.: Groundhogs and Woodchucks

- Suppose $\mathsf{believe}_{\iota\to o\to o} \in \Sigma$ and $\mathcal{E}(\mathsf{believe})$ is the (Curried) function $\mathsf{bel} \in \mathcal{D}_{\iota\to o\to o}$ such that $\mathsf{bel}(1)(\mathsf{b}) = \mathsf{b}$ and $\mathsf{bel}(1)(\mathsf{a}) = \mathsf{bel}(1)(\mathsf{c}) = \mathsf{bel}(0)(\mathsf{a}) = \mathsf{bel}(0)(\mathsf{b}) = \mathsf{bel}(0)(\mathsf{c}) = \mathsf{c}$.
- Intuitively, John believes propositions with value b, but not those with value a or c.

Ex.: Groundhogs and Woodchucks

- Suppose $\mathsf{believe}_{\iota\to o\to o} \in \Sigma$ and $\mathcal{E}(\mathsf{believe})$ is the (Curried) function $\mathsf{bel} \in \mathcal{D}_{\iota\to o\to o}$ such that $\mathsf{bel}(1)(\mathsf{b}) = \mathsf{b}$ and $\mathsf{bel}(1)(\mathsf{a}) = \mathsf{bel}(1)(\mathsf{c}) = \mathsf{bel}(0)(\mathsf{a}) = \mathsf{bel}(0)(\mathsf{b}) = \mathsf{bel}(0)(\mathsf{c}) = \mathsf{c}$.
- Intuitively, John believes propositions with value b, but not those with value a or c.
- So, $\mathsf{believes\ john(woodchuck\ phil)}$ is valid, while $\mathsf{believes\ john(groundhog\ phil)}$ is not.

Generalizing the Previous Model

As we have seen, Boolean extensionality fails when one has more than two values in $\mathcal{D}_o$.

Generalizing the Previous Model

As we have seen, Boolean extensionality fails when one has more than two values in $\mathcal{D}_o$. We can generalize the construction defining $\mathcal{D}_o := \{\mathtt{F}\} \cup \mathcal{B}$, where $\mathcal{B}$ is any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$.

Generalizing the Previous Model

As we have seen, Boolean extensionality fails when one has more than two values in $\mathcal{D}_o$. We can generalize the construction defining $\mathcal{D}_o := \{\mathtt{F}\} \cup \mathcal{B}$, where $\mathcal{B}$ is any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{\mathtt{T}\}$.

Generalizing the Previous Model

As we have seen, Boolean extensionality fails when one has more than two values in $\mathcal{D}_o$. We can generalize the construction defining $\mathcal{D}_o := \{\mathtt{F}\} \cup \mathcal{B}$, where $\mathcal{B}$ is any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{\mathtt{T}\}$. In this way, we can easily construct models for the case with property $\mathfrak{b}$ and the case without property $\mathfrak{b}$ simultaneously.

Generalizing the Previous Model

As we have seen, Boolean extensionality fails when one has more than two values in $\mathcal{D}_o$. We can generalize the construction defining $\mathcal{D}_o := \{\mathtt{F}\} \cup \mathcal{B}$, where $\mathcal{B}$ is any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{\mathtt{T}\}$. In this way, we can easily construct models for the case with property $\mathfrak{b}$ and the case without property $\mathfrak{b}$ simultaneously. We will use this idea to parameterize the remaining model constructions by $\mathcal{B}$.

Generalizing the Previous Model

As we have seen, Boolean extensionality fails when one has more than two values in $\mathcal{D}_o$. We can generalize the construction defining $\mathcal{D}_o := \{\mathtt{F}\} \cup \mathcal{B}$, where $\mathcal{B}$ is any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{\mathtt{T}\}$. In this way, we can easily construct models for the case with property $\mathfrak{b}$ and the case without property $\mathfrak{b}$ simultaneously. We will use this idea to parameterize the remaining model constructions by $\mathcal{B}$.

These semantic constructions are similar to those in multi-valued logics.

Generalizing the Previous Model

As we have seen, Boolean extensionality fails when one has more than two values in $\mathcal{D}_o$. We can generalize the construction defining $\mathcal{D}_o := \{\mathtt{F}\} \cup \mathcal{B}$, where $\mathcal{B}$ is any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{\mathtt{T}\}$. In this way, we can easily construct models for the case with property $\mathfrak{b}$ and the case without property $\mathfrak{b}$ simultaneously. We will use this idea to parameterize the remaining model constructions by $\mathcal{B}$.

These semantic constructions are similar to those in multi-valued logics. In contrast to these logics where the logical connectives are adapted to talk about multiple truth values, in our setting we are mainly interested in multiple truth values as diverse υ-pre-images of $\mathtt{T}$ and $\mathtt{F}$.

Semantics: Examples of Σ-Models (Contd.)

Ex.: Models without Funct. Extensionality

- Idea: attach distinguishing labels to functions without changing their applicative behavior

Ex.: Models without Funct. Extensionality

- Idea: attach distinguishing labels to functions without changing their applicative behavior
- Let $\mathcal{B}$ be any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$

Ex.: Models without Funct. Extensionality

- Idea: attach distinguishing labels to functions without changing their applicative behavior
- Let $\mathcal{B}$ be any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$
- Let $\mathcal{D}_o := \{\mathtt{F}\} \cup \mathcal{B}$ and $\mathcal{D}_\iota := \{*\}$

Ex.: Models without Funct. Extensionality

- Idea: attach distinguishing labels to functions without changing their applicative behavior
- Let $\mathcal{B}$ be any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$
- Let $\mathcal{D}_o := \{\mathtt{F}\} \cup \mathcal{B}$ and $\mathcal{D}_\iota := \{*\}$
- For each function type $\beta\alpha$, let

$$\mathcal{D}_{\beta\alpha} := \{(\mathsf{i}, \mathsf{f}) \mid \mathsf{i} \in \{0, 1\} \text{ and } \mathsf{f} \colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta\}$$

Ex.: Models without Funct. Extensionality

- Idea: attach distinguishing labels to functions without changing their applicative behavior
- Let $\mathcal{B}$ be any set with $\mathtt{T} \in \mathcal{B}$ and $\mathtt{F} \notin \mathcal{B}$
- Let $\mathcal{D}_o := \{\mathtt{F}\} \cup \mathcal{B}$ and $\mathcal{D}_\iota := \{*\}$
- For each function type $\beta\alpha$, let

$$\mathcal{D}_{\beta\alpha} := \{(\mathsf{i}, \mathsf{f}) \mid \mathsf{i} \in \{0, 1\} \text{ and } \mathsf{f} \colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta\}$$

- We define application by

$$(\mathsf{i}, \mathsf{f})@\mathsf{a} := \mathsf{f}(\mathsf{a})$$

whenever $(\mathsf{i}, \mathsf{f}) \in \mathcal{D}_{\beta\alpha}$ and $\mathsf{a} \in \mathcal{D}_\alpha$

Ex.: Models without η and $\mathfrak{f}$

- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:

Ex.: Models without η and $\mathfrak{f}$

- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms

Ex.: Models without η and $\mathfrak{f}$

- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms
  - $\mathcal{E}(\neg) := (0, \mathsf{n})$ where $\mathsf{n}(\mathsf{b}) := \mathtt{F}$ for every $\mathsf{b} \in \mathcal{B}$ and $\mathsf{n}(\mathtt{F}) := \mathtt{T}$

Ex.: Models without η and $\mathfrak{f}$

- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms
  - $\mathcal{E}(\neg) := (0, \mathsf{n})$ where $\mathsf{n}(\mathsf{b}) := \mathtt{F}$ for every $\mathsf{b} \in \mathcal{B}$ and $\mathsf{n}(\mathtt{F}) := \mathtt{T}$
  - $\mathcal{E}(\vee) := (0, \mathsf{d})$ where
    $\mathsf{d}(\mathsf{b}) := (0, \mathsf{k}^{\mathtt{T}})$ for every $\mathsf{b} \in \mathcal{B}$ and
    $\mathsf{d}(\mathtt{F}) := (0, \mathsf{id})$
    ($\mathsf{k}^{\mathtt{T}}$ is the constant $\mathtt{T}$ function)
    (id is the identity function from $\mathcal{D}_o$ to $\mathcal{D}_o$)

Ex.: Models without η and $\mathfrak{f}$

- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms
  - $\mathcal{E}(\neg) := (0, \mathsf{n})$ where $\mathsf{n}(\mathsf{b}) := \mathtt{F}$ for every $\mathsf{b} \in \mathcal{B}$ and $\mathsf{n}(\mathtt{F}) := \mathtt{T}$
  - $\mathcal{E}(\vee) := (0, \mathsf{d})$ where
    $\mathsf{d}(\mathsf{b}) := (0, \mathsf{k}^{\mathtt{T}})$ for every $\mathsf{b} \in \mathcal{B}$ and
    $\mathsf{d}(\mathtt{F}) := (0, \mathsf{id})$
    ($\mathsf{k}^{\mathtt{T}}$ is the constant $\mathtt{T}$ function)
    (id is the identity function from $\mathcal{D}_{\mathsf{o}}$ to $\mathcal{D}_{\mathsf{o}}$)
  - $\mathcal{E}(\Pi^{\alpha}) := (0, \pi^{\alpha})$ where for each $(\mathsf{i}, \mathsf{f}) \in \mathcal{D}_{\mathsf{o}\alpha}$, $\pi^{\alpha}((\mathsf{i}, \mathsf{f})) := \mathtt{T}$ if $\mathsf{f}(\mathsf{a}) \in \mathcal{B}$ for all $\mathsf{a} \in \mathcal{D}_{\alpha}$ and $\pi^{\alpha}(\mathsf{i}, \mathsf{f}) := \mathtt{F}$ otherwise

Ex.: Models without η and $\mathfrak{f}$

- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms
  - $\mathcal{E}(\neg) := (0, \mathsf{n})$ where $\mathsf{n}(\mathsf{b}) := \mathtt{F}$ for every $\mathsf{b} \in \mathcal{B}$ and $\mathsf{n}(\mathtt{F}) := \mathtt{T}$
  - $\mathcal{E}(\vee) := (0, \mathsf{d})$ where
    $\mathsf{d}(\mathsf{b}) := (0, \mathsf{k}^{\mathtt{T}})$ for every $\mathsf{b} \in \mathcal{B}$ and
    $\mathsf{d}(\mathtt{F}) := (0, \mathsf{id})$
    ($\mathsf{k}^{\mathtt{T}}$ is the constant $\mathtt{T}$ function)
    (id is the identity function from $\mathcal{D}_{\mathsf{o}}$ to $\mathcal{D}_{\mathsf{o}}$)
  - $\mathcal{E}(\Pi^{\alpha}) := (0, \pi^{\alpha})$ where for each $(\mathsf{i}, \mathsf{f}) \in \mathcal{D}_{\mathsf{o}\alpha}$, $\pi^{\alpha}((\mathsf{i}, \mathsf{f})) := \mathtt{T}$ if $\mathsf{f}(\mathsf{a}) \in \mathcal{B}$ for all $\mathsf{a} \in \mathcal{D}_{\alpha}$ and $\pi^{\alpha}(\mathsf{i}, \mathsf{f}) := \mathtt{F}$ otherwise
  - $\mathsf{q}^{\alpha} := (0, \mathsf{q}^{\alpha}) \in \mathcal{D}_{\mathsf{o}\alpha\alpha}$ where $\mathsf{q}^{\alpha}(\mathsf{a}) := (0, \mathsf{s}^{\mathsf{a}})$ and $\mathsf{s}^{\mathsf{a}}(\mathsf{b}) := \mathtt{T}$ if $\mathsf{a} = \mathsf{b}$ and $\mathsf{s}^{\mathsf{a}}(\mathsf{b}) := \mathtt{F}$ otherwise

Ex.: Models without η and $\mathfrak{f}$

- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms
  - $\mathcal{E}(\neg) := (0, \mathsf{n})$ where $\mathsf{n}(\mathsf{b}) := \mathtt{F}$ for every $\mathsf{b} \in \mathcal{B}$ and $\mathsf{n}(\mathtt{F}) := \mathtt{T}$
  - $\mathcal{E}(\vee) := (0, \mathsf{d})$ where
    $\mathsf{d}(\mathsf{b}) := (0, \mathsf{k}^{\mathtt{T}})$ for every $\mathsf{b} \in \mathcal{B}$ and
    $\mathsf{d}(\mathtt{F}) := (0, \mathsf{id})$
    ($\mathsf{k}^{\mathtt{T}}$ is the constant $\mathtt{T}$ function)
    (id is the identity function from $\mathcal{D}_o$ to $\mathcal{D}_o$)
  - $\mathcal{E}(\Pi^\alpha) := (0, \pi^\alpha)$ where for each $(\mathsf{i}, \mathsf{f}) \in \mathcal{D}_{o\alpha}$, $\pi^\alpha((\mathsf{i}, \mathsf{f})) := \mathtt{T}$ if $\mathsf{f}(\mathsf{a}) \in \mathcal{B}$ for all $\mathsf{a} \in \mathcal{D}_\alpha$ and $\pi^\alpha(\mathsf{i}, \mathsf{f}) := \mathtt{F}$ otherwise
  - $\mathsf{q}^\alpha := (0, \mathsf{q}^\alpha) \in \mathcal{D}_{o\alpha\alpha}$ where $\mathsf{q}^\alpha(\mathsf{a}) := (0, \mathsf{s}^\mathsf{a})$ and $\mathsf{s}^\mathsf{a}(\mathsf{b}) := \mathtt{T}$ if $\mathsf{a} = \mathsf{b}$ and $\mathsf{s}^\mathsf{a}(\mathsf{b}) := \mathtt{F}$ otherwise
  - $\mathcal{E}(\mathsf{w}) \in \mathcal{D}_\alpha$ arbitrary for parameters $\mathsf{w} \in \Sigma_\alpha$.

Ex.: Models without η and $\mathfrak{f}$

Ex.: Models without η and $\mathfrak{f}$

- For variables, we define $\mathcal{E}_\varphi(\mathsf{X}) := \varphi(\mathsf{X})$

Ex.: Models without η and $\mathfrak{f}$

- For variables, we define $\mathcal{E}_\varphi(\mathsf{X}) := \varphi(\mathsf{X})$
- For application, we define $\mathcal{E}_\varphi(\mathbf{FA}) := \mathcal{E}_\varphi(\mathbf{F})@\mathcal{E}_\varphi(\mathbf{A})$

Ex.: Models without η and $\mathfrak{f}$

- For variables, we define $\mathcal{E}_\varphi(\mathsf{X}) := \varphi(\mathsf{X})$
- For application, we define $\mathcal{E}_\varphi(\mathbf{FA}) := \mathcal{E}_\varphi(\mathbf{F})@\mathcal{E}_\varphi(\mathbf{A})$
- For λ-abstractions, we define $\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha \mathbf{.} \mathbf{B}_\beta) := (0, \mathsf{f})$ where $\mathsf{f}\colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$ is the function such that $\mathsf{f}(\mathsf{a}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{B})$ for all $\mathsf{a} \in \mathcal{D}_\alpha$

Ex.: Models without η and $\mathfrak{f}$

- For variables, we define $\mathcal{E}_\varphi(\mathsf{X}) := \varphi(\mathsf{X})$
- For application, we define $\mathcal{E}_\varphi(\mathbf{FA}) := \mathcal{E}_\varphi(\mathbf{F})@\mathcal{E}_\varphi(\mathbf{A})$
- For λ-abstractions, we define $\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha . \mathbf{B}_\beta) := (0, \mathsf{f})$ where $\mathsf{f}\colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$ is the function such that $\mathsf{f}(\mathsf{a}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{B})$ for all $\mathsf{a} \in \mathcal{D}_\alpha$

- With some work (which we omit), one can show that this $\mathcal{E}$ is an evaluation function

Ex.: Models without η and $\mathfrak{f}$

- For variables, we define $\mathcal{E}_\varphi(\mathsf{X}) := \varphi(\mathsf{X})$
- For application, we define $\mathcal{E}_\varphi(\mathbf{FA}) := \mathcal{E}_\varphi(\mathbf{F})@\mathcal{E}_\varphi(\mathbf{A})$
- For λ-abstractions, we define $\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha\mathbf{.}\mathbf{B}_\beta) := (0, \mathsf{f})$ where $\mathsf{f} \colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$ is the function such that $\mathsf{f}(\mathsf{a}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{B})$ for all $\mathsf{a} \in \mathcal{D}_\alpha$

- With some work (which we omit), one can show that this $\mathcal{E}$ is an evaluation function
- Taking υ to be the function such that $\upsilon(\mathsf{b}) := \mathtt{T}$ for every $\mathsf{b} \in \mathcal{B}$ and $\upsilon(\mathtt{F}) := \mathtt{F}$, one can easily show that this is a valuation

Ex.: Models without η and $\mathfrak{f}$

- For variables, we define $\mathcal{E}_\varphi(\mathsf{X}) := \varphi(\mathsf{X})$
- For application, we define $\mathcal{E}_\varphi(\mathbf{FA}) := \mathcal{E}_\varphi(\mathbf{F})@\mathcal{E}_\varphi(\mathbf{A})$
- For λ-abstractions, we define $\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha \mathbf{.} \mathbf{B}_\beta) := (0, \mathsf{f})$ where $\mathsf{f} \colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$ is the function such that $\mathsf{f}(\mathsf{a}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{B})$ for all $\mathsf{a} \in \mathcal{D}_\alpha$

- With some work (which we omit), one can show that this $\mathcal{E}$ is an evaluation function
- Taking υ to be the function such that $\upsilon(\mathsf{b}) := \mathtt{T}$ for every $\mathsf{b} \in \mathcal{B}$ and $\upsilon(\mathtt{F}) := \mathtt{F}$, one can easily show that this is a valuation
- Hence, $\mathcal{M}^{\mathcal{B}} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ is a Σ-model

Ex.: Models without η and $\mathfrak{f}$

- The objects $\mathsf{q}^{\alpha} := (0, \mathsf{q}^{\alpha})$ witness property $\mathfrak{q}$ for $\mathcal{M}^{\mathcal{B}}$

Ex.: Models without η and $\mathfrak{f}$

- The objects $\mathsf{q}^{\alpha} := (0, \mathsf{q}^{\alpha})$ witness property $\mathfrak{q}$ for $\mathcal{M}^{\mathcal{B}}$
- The objects $(1, \mathsf{q}^{\alpha})$ also witness property $\mathfrak{q}$ (so, in the non-functional case such witnesses are not unique)

Ex.: Models without η and $\mathfrak{f}$

- The objects $\mathsf{q}^\alpha := (0, \mathsf{q}^\alpha)$ witness property $\mathfrak{q}$ for $\mathcal{M}^\mathcal{B}$
- The objects $(1, \mathsf{q}^\alpha)$ also witness property $\mathfrak{q}$ (so, in the non-functional case such witnesses are not unique)
- Hence, $\mathcal{M}^\mathcal{B} := (\mathcal{D}, @, \mathcal{E}, \upsilon)$ is a Σ-model with property $\mathfrak{q}$

Ex.: Models without η and $\mathfrak{f}$

- Property $\mathfrak{f}$ fails for $\mathcal{M}^{\mathcal{B}}$, since the applicative structure $(\mathcal{D}, @)$ is not functional:

Ex.: Models without η and $\mathfrak{f}$

- Property $\mathfrak{f}$ fails for $\mathcal{M}^{\mathcal{B}}$, since the applicative structure $(\mathcal{D}, @)$ is not functional:
  - Consider $\mathtt{u}\colon \mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$.

Ex.: Models without η and $\mathfrak{f}$

- Property $\mathfrak{f}$ fails for $\mathcal{M}^{\mathcal{B}}$, since the applicative structure $(\mathcal{D}, @)$ is not functional:
  - Consider $\mathrm{u}\colon \mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$.
  - For both $(0, \mathrm{u}), (1, \mathrm{u}) \in \mathcal{D}_{\iota\iota}$ we have

$$(\mathsf{i}, \mathrm{u})@* = *$$

  although $(0, \mathrm{u}) \neq (1, \mathrm{u})$

Ex.: Models without η and $\mathfrak{f}$

- Does η hold?

Ex.: Models without η and $\mathfrak{f}$

- Does η hold?
- No!

Ex.: Models without η and $\mathfrak{f}$

- Does η hold?
- No!
- Compute, for example, $\mathcal{E}(\lambda \mathsf{F}_{\beta\alpha}\textbf{.}\mathsf{F})$ and $\mathcal{E}(\lambda \mathsf{F}_{\beta\alpha}\textbf{.}\lambda \mathsf{X}_{\alpha}\textbf{.}\mathsf{F}\mathsf{X})$

Ex.: Models without η and $\mathfrak{f}$

- Does η hold?
- No!
- Compute, for example, $\mathcal{E}(\lambda \mathsf{F}_{\beta\alpha} \mathbf{.} \mathsf{F})$ and $\mathcal{E}(\lambda \mathsf{F}_{\beta\alpha} \mathbf{.} \lambda \mathsf{X}_{\alpha} \mathbf{.} \mathsf{FX})$
  - $\mathcal{E}(\lambda \mathsf{F}_{\beta\alpha} \mathbf{.} \mathsf{F}) = (0, \mathsf{id})$ where id is the identity function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$

Ex.: Models without η and $\mathfrak{f}$

- Does η hold?
- No!
- Compute, for example, $\mathcal{E}(\lambda F_{\beta\alpha}.F)$ and $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_{\alpha}.FX)$
  - $\mathcal{E}(\lambda F_{\beta\alpha}.F) = (0, \mathsf{id})$ where id is the identity function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$
  - $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_{\alpha}.FX) = (0, \mathsf{p})$ where p is the function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$ such that $\mathsf{p}((\mathsf{i}, \mathsf{f})) = (0, \mathsf{f})$ for each $\mathsf{f}\colon \mathcal{D}_{\alpha} \longrightarrow \mathcal{D}_{\beta}$

Ex.: Models without η and $\mathfrak{f}$

- Does η hold?
- No!
- Compute, for example, $\mathcal{E}(\lambda F_{\beta\alpha}.F)$ and $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_\alpha.FX)$
  - $\mathcal{E}(\lambda F_{\beta\alpha}.F) = (0, \mathsf{id})$ where id is the identity function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$
  - $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_\alpha.FX) = (0, \mathsf{p})$ where p is the function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$ such that $\mathsf{p}((\mathsf{i}, \mathsf{f})) = (0, \mathsf{f})$ for each $\mathsf{f}\colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$
- Hence $\mathcal{E}(\lambda F_{\beta\alpha}.F) \neq \mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_\alpha.FX)$

Ex.: Models without η and $\mathfrak{f}$

- Does ξ hold?

Ex.: Models without η and $\mathfrak{f}$

- Does ξ hold?
- Yes!

Ex.: Models without η and $\mathfrak{f}$

- Does ξ hold?
- Yes!
- If

$$\mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{M}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{N})$$

for every $\mathsf{a} \in \mathcal{D}_\alpha$, then

$$\mathcal{E}_\varphi(\lambda \mathsf{X}_\alpha . \mathbf{M}) = (0, \mathsf{f}) = \mathcal{E}_\varphi(\lambda \mathsf{X} . \mathbf{N})$$

where $\mathsf{f}(\mathsf{a}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{M}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{N})$ for every $\mathsf{a} \in \mathcal{D}_\alpha$.

Ex.: Models without η and $\mathfrak{f}$

- If $\mathcal{B} = \{\mathtt{T}\}$, then the model $\mathcal{M}^{\beta\xi\mathfrak{b}} := \mathcal{M}^{\{\mathtt{T}\}}$ satisfies property $\mathfrak{b}$.

Ex.: Models without η and $\mathfrak{f}$

- If $\mathcal{B} = \{\mathtt{T}\}$, then the model $\mathcal{M}^{\beta\xi\mathfrak{b}} := \mathcal{M}^{\{\mathtt{T}\}}$ satisfies property $\mathfrak{b}$.
- So, we know $\mathcal{M}^{\beta\xi\mathfrak{b}} \in \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma) \setminus \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$.

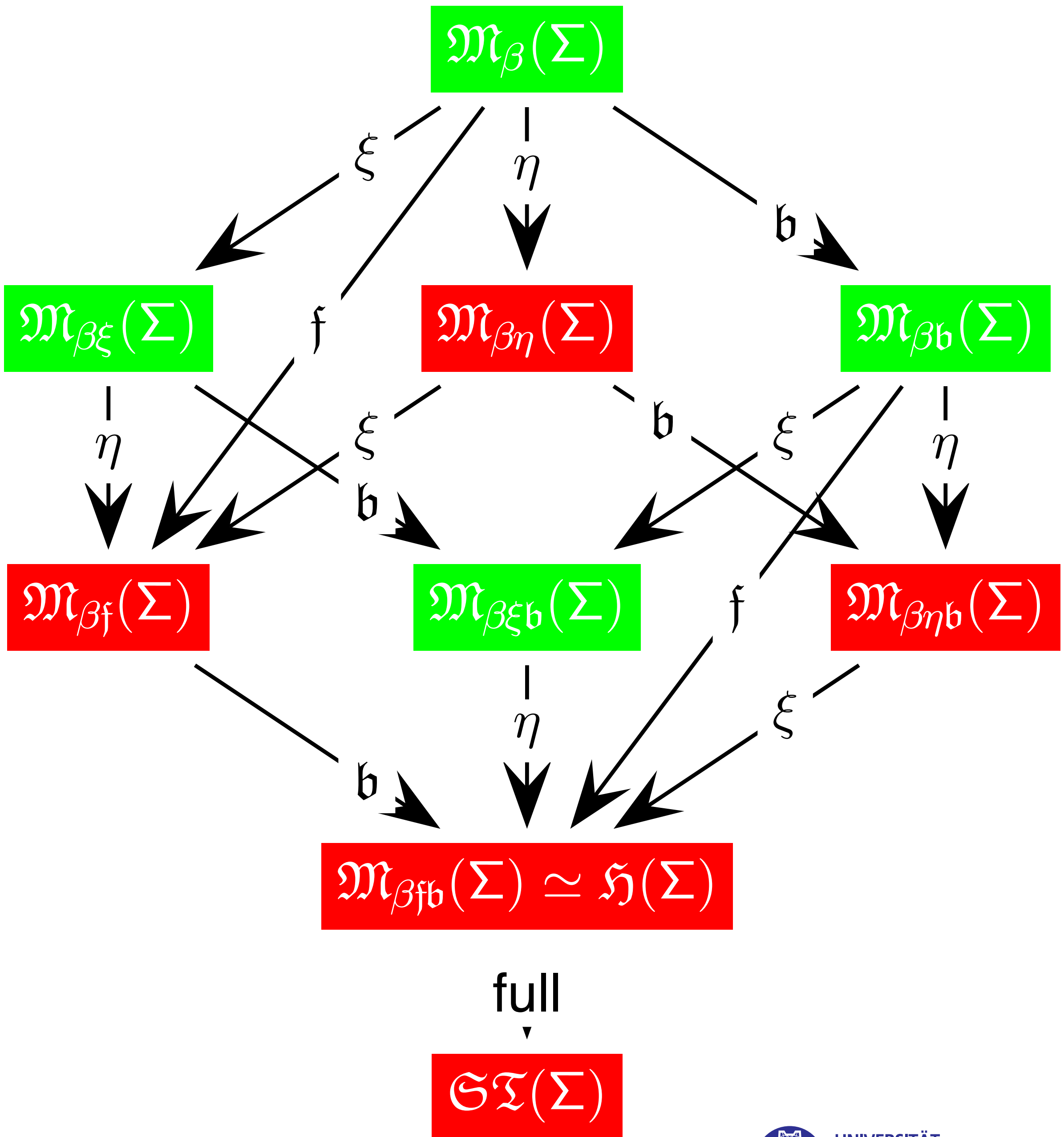
Ex.: Models without η and $\mathfrak{f}$

- If $\mathcal{B} = \{\mathtt{T}\}$, then the model $\mathcal{M}^{\beta\xi\mathfrak{b}} := \mathcal{M}^{\{\mathtt{T}\}}$ satisfies property $\mathfrak{b}$.
- So, we know $\mathcal{M}^{\beta\xi\mathfrak{b}} \in \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma) \setminus \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$.
- On the other hand, if b is any value with $\mathsf{b} \notin \{\mathtt{T}, \mathtt{F}\}$, and $\mathcal{B} = \{\mathtt{T}, \mathsf{b}\}$, then the model $\mathcal{M}^{\beta\xi} := \mathcal{M}^{\{\mathtt{T},\mathsf{b}\}}$ does not satisfy property $\mathfrak{b}$.

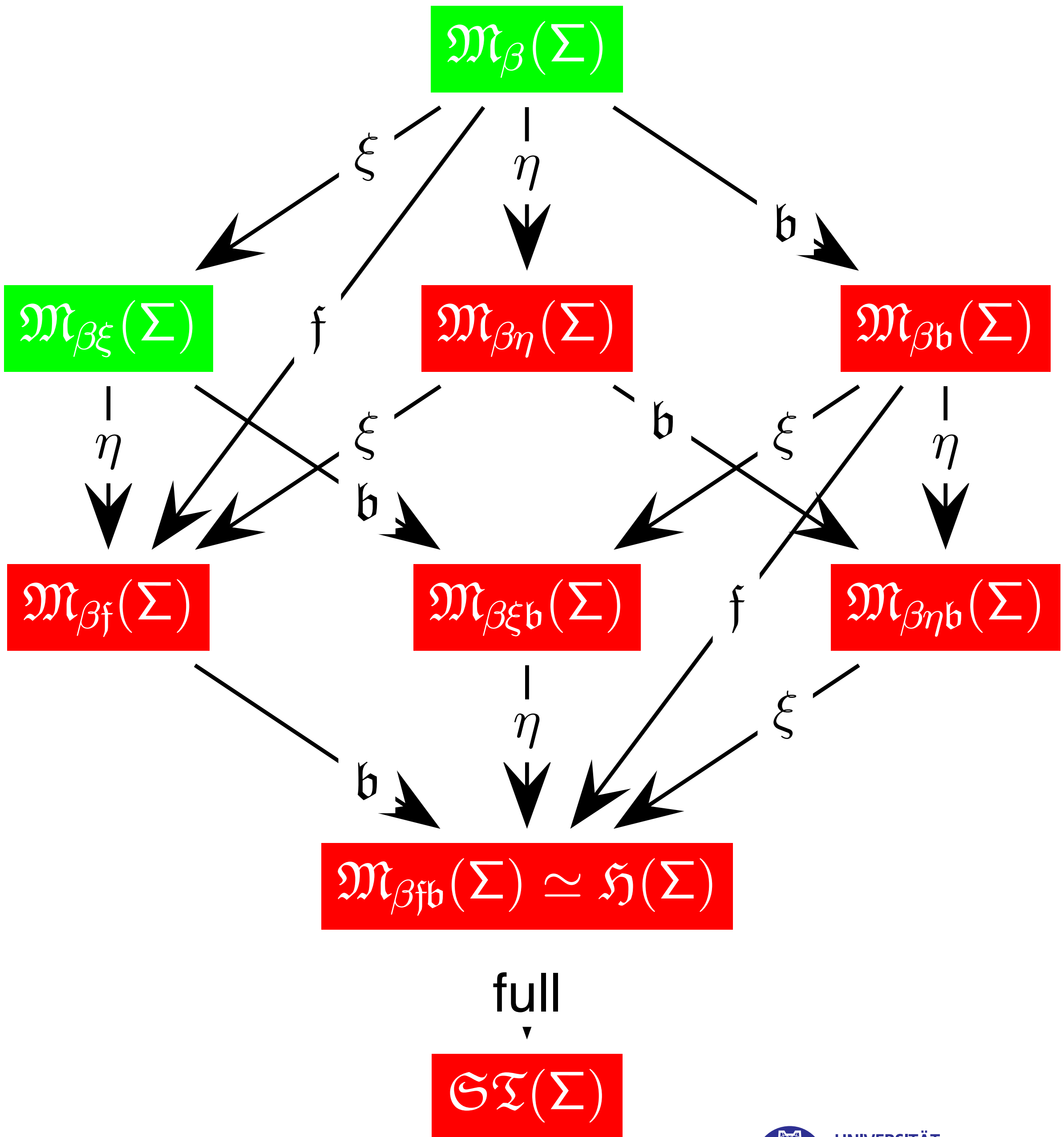
Ex.: Models without η and $\mathfrak{f}$

- If $\mathcal{B} = \{\mathtt{T}\}$, then the model $\mathcal{M}^{\beta\xi\mathfrak{b}} := \mathcal{M}^{\{\mathtt{T}\}}$ satisfies property $\mathfrak{b}$.
- So, we know $\mathcal{M}^{\beta\xi\mathfrak{b}} \in \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma) \setminus \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$.
- On the other hand, if b is any value with $\mathsf{b} \notin \{\mathtt{T}, \mathtt{F}\}$, and $\mathcal{B} = \{\mathtt{T}, \mathsf{b}\}$, then the model $\mathcal{M}^{\beta\xi} := \mathcal{M}^{\{\mathtt{T}, \mathsf{b}\}}$ does not satisfy property $\mathfrak{b}$.
- In this case, we know $\mathcal{M}^{\beta\xi} \in \mathfrak{M}_{\beta\xi}(\Sigma) \setminus (\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \cup \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma))$.

Ex.: Models without η and $\mathfrak{f}$



Ex.: Models without η and $\mathfrak{f}$



Ex.: Models without η and $\mathfrak{f}$

- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ-model $(\mathcal{D}, @, \mathcal{E}, \upsilon)$ as constructed before

Ex.: Models without η and $\mathfrak{f}$

- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ-model $(\mathcal{D}, @, \mathcal{E}, \upsilon)$ as constructed before
- Define an alternative evaluation function $\mathcal{E}'$ by induction:

Ex.: Models without η and $\mathfrak{f}$

- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ-model $(\mathcal{D}, @, \mathcal{E}, \upsilon)$ as constructed before
- Define an alternative evaluation function $\mathcal{E}'$ by induction:
  - For all $\mathrm{w} \in \Sigma$, let $\mathcal{E}'(\mathrm{w}) := \mathcal{E}(\mathrm{w})$

Ex.: Models without η and $\mathfrak{f}$

- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ-model $(\mathcal{D}, @, \mathcal{E}, \upsilon)$ as constructed before
- Define an alternative evaluation function $\mathcal{E}'$ by induction:
  - For all $\mathrm{w} \in \Sigma$, let $\mathcal{E}'(\mathrm{w}) := \mathcal{E}(\mathrm{w})$
  - For variables we define $\mathcal{E}'_{\varphi}(\mathrm{X}) := \varphi(\mathrm{X})$

Ex.: Models without η and $\mathfrak{f}$

- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ-model $(\mathcal{D}, @, \mathcal{E}, \upsilon)$ as constructed before
- Define an alternative evaluation function $\mathcal{E}'$ by induction:
  - For all $\mathrm{w} \in \Sigma$, let $\mathcal{E}'(\mathrm{w}) := \mathcal{E}(\mathrm{w})$
  - For variables we define $\mathcal{E}'_\varphi(\mathrm{X}) := \varphi(\mathrm{X})$
  - We must define $\mathcal{E}'_\varphi(\mathbf{FA}) := \mathcal{E}'_\varphi(\mathbf{F})@\mathcal{E}'_\varphi(\mathbf{A})$

Ex.: Models without η and $\mathfrak{f}$

- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ-model $(\mathcal{D}, @, \mathcal{E}, \upsilon)$ as constructed before
- Define an alternative evaluation function $\mathcal{E}'$ by induction:
  - For all $\mathsf{w} \in \Sigma$, let $\mathcal{E}'(\mathsf{w}) := \mathcal{E}(\mathsf{w})$
  - For variables we define $\mathcal{E}'_\varphi(\mathsf{X}) := \varphi(\mathsf{X})$
  - We must define $\mathcal{E}'_\varphi(\mathbf{FA}) := \mathcal{E}'_\varphi(\mathbf{F})@\mathcal{E}'_\varphi(\mathbf{A})$
  - We choose $\mathcal{E}'_\varphi(\lambda \mathsf{X}_\alpha \mathbf{.} \mathbf{B}_\beta) := (1, \mathsf{f})$ where $\mathsf{f}\colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$ is the function such that $\mathsf{f}(\mathsf{a}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{B})$ for all $\mathsf{a} \in \mathcal{D}_\alpha$

Ex.: Models without η and $\mathfrak{f}$

- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ-model $(\mathcal{D}, @, \mathcal{E}, \upsilon)$ as constructed before
- Define an alternative evaluation function $\mathcal{E}'$ by induction:
  - For all $\mathsf{w} \in \Sigma$, let $\mathcal{E}'(\mathsf{w}) := \mathcal{E}(\mathsf{w})$
  - For variables we define $\mathcal{E}'_\varphi(\mathsf{X}) := \varphi(\mathsf{X})$
  - We must define $\mathcal{E}'_\varphi(\mathbf{FA}) := \mathcal{E}'_\varphi(\mathbf{F})@\mathcal{E}'_\varphi(\mathbf{A})$
  - We choose $\mathcal{E}'_\varphi(\lambda \mathsf{X}_\alpha \mathbf{.} \mathbf{B}_\beta) := (1, \mathsf{f})$ where $\mathsf{f}\colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$ is the function such that $\mathsf{f}(\mathsf{a}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{B})$ for all $\mathsf{a} \in \mathcal{D}_\alpha$
- $\mathcal{E}$ and $\mathcal{E}'$ agree on all constants, they are different though:

$$\mathcal{E}(\lambda \mathsf{X}_\iota \mathbf{.} \mathsf{X}) = (0, \mathsf{id}) \neq (1, \mathsf{id}) = \mathcal{E}'(\lambda \mathsf{X}_\iota \mathbf{.} \mathsf{X})$$

where $\mathsf{id} : \mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$ is the identity function

Ex.: Models without η and $\mathfrak{f}$

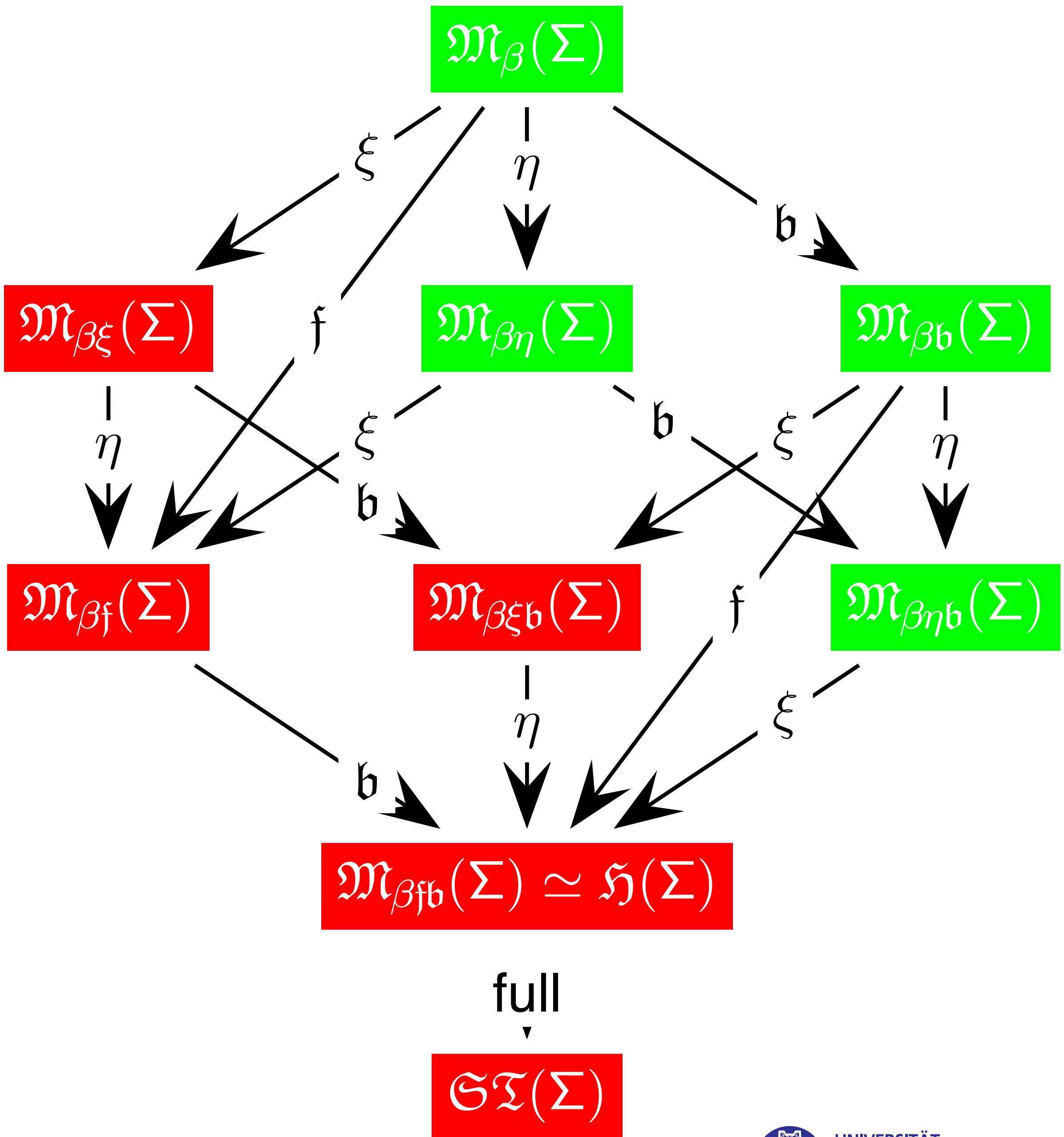
- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ-model $(\mathcal{D}, @, \mathcal{E}, \upsilon)$ as constructed before
- Define an alternative evaluation function $\mathcal{E}'$ by induction:
  - For all $\mathsf{w} \in \Sigma$, let $\mathcal{E}'(\mathsf{w}) := \mathcal{E}(\mathsf{w})$
  - For variables we define $\mathcal{E}'_\varphi(\mathsf{X}) := \varphi(\mathsf{X})$
  - We must define $\mathcal{E}'_\varphi(\mathbf{FA}) := \mathcal{E}'_\varphi(\mathbf{F})@\mathcal{E}'_\varphi(\mathbf{A})$
  - We choose $\mathcal{E}'_\varphi(\lambda \mathsf{X}_\alpha \mathbf{.} \mathbf{B}_\beta) := (1, \mathsf{f})$ where $\mathsf{f}\colon \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$ is the function such that $\mathsf{f}(\mathsf{a}) = \mathcal{E}_{\varphi,[\mathsf{a}/\mathsf{X}]}(\mathbf{B})$ for all $\mathsf{a} \in \mathcal{D}_\alpha$
- $\mathcal{E}$ and $\mathcal{E}'$ agree on all constants, they are different though:

$$\mathcal{E}(\lambda \mathsf{X}_\iota \mathbf{.} \mathsf{X}) = (0, \mathsf{id}) \neq (1, \mathsf{id}) = \mathcal{E}'(\lambda \mathsf{X}_\iota \mathbf{.} \mathsf{X})$$

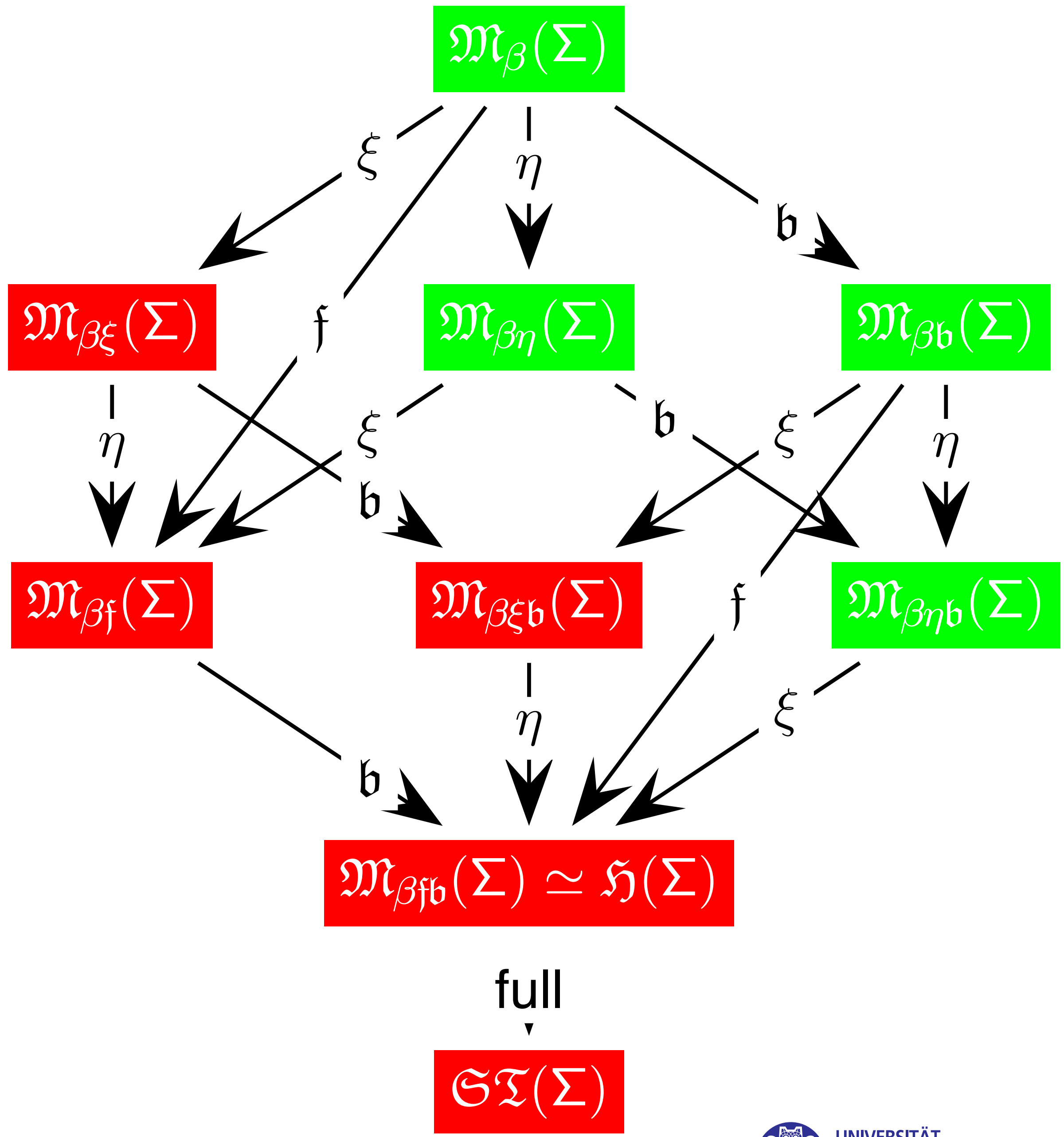
where $\mathsf{id} : \mathcal{D}_\iota \longrightarrow \mathcal{D}_\iota$ is the identity function

- Thus, in non-functional models evaluation functions are not uniquely determined by their values on constants

Ex.: Models without ξ



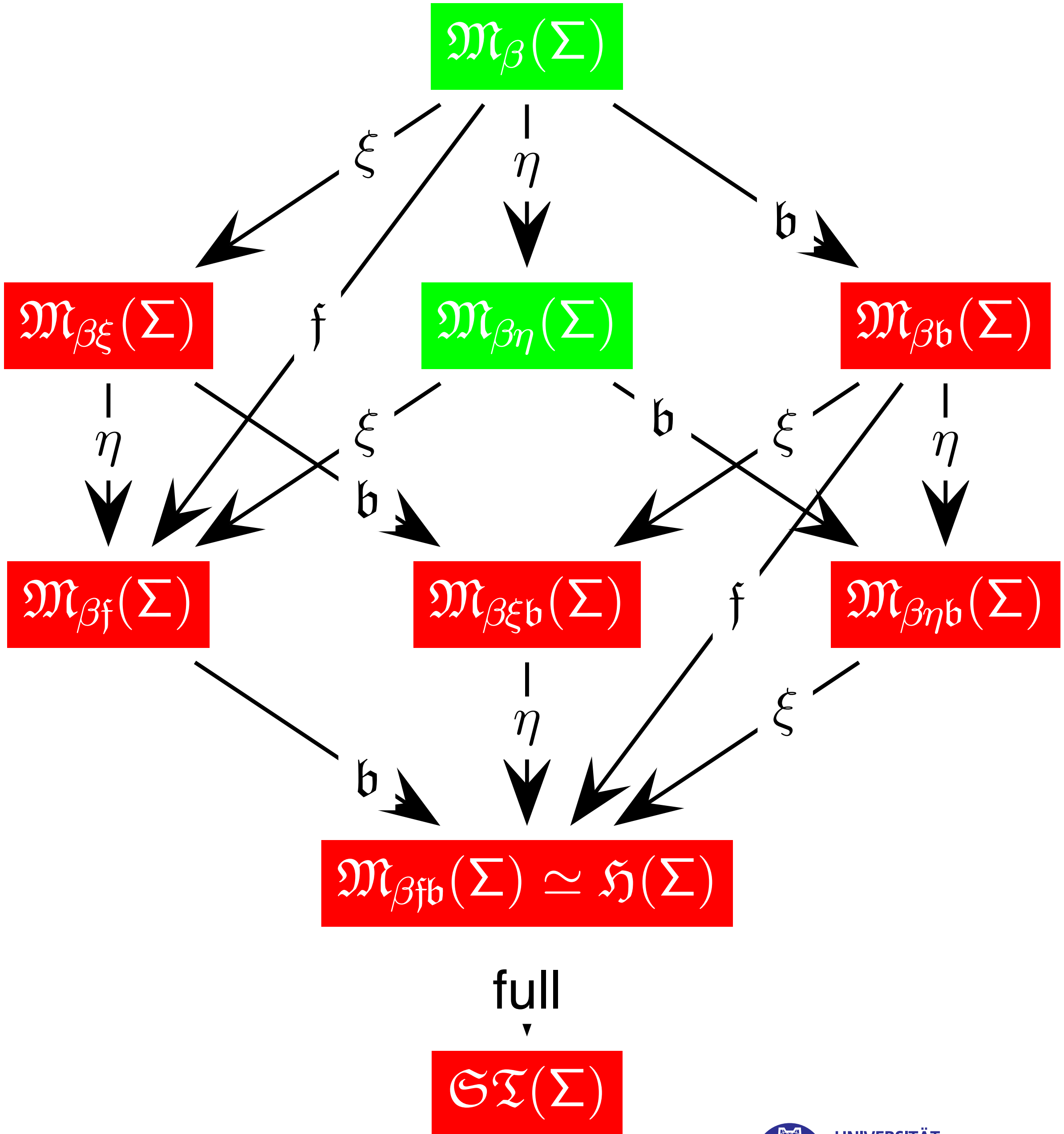
Ex.: Models without ξ



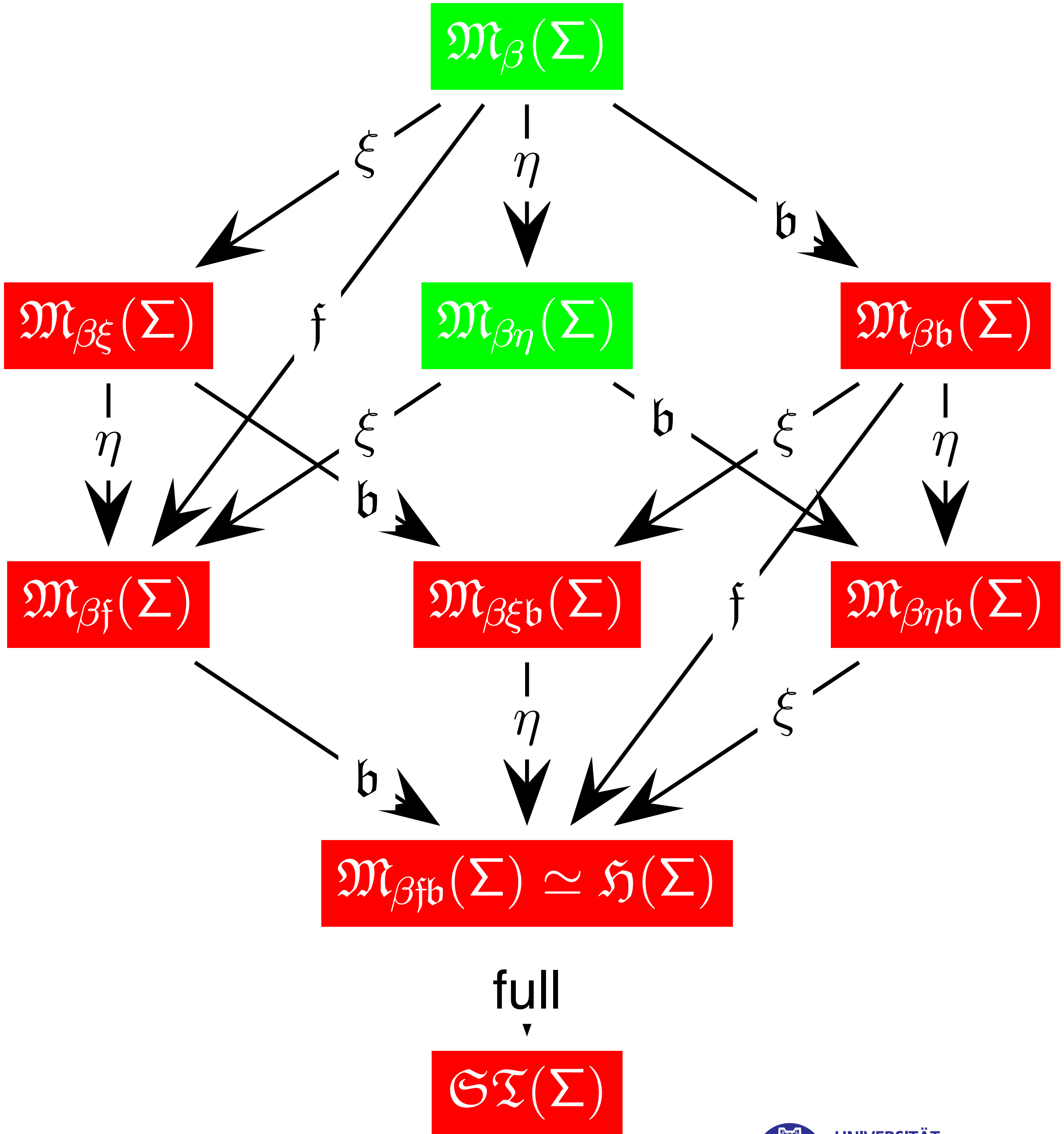
Not here!

See [JSL-04]

Ex.: Models without ξ



Ex.: Models without ξ



Not here!

See [JSL-04]