Working with Automated Reasoning Tools
– HOL Syntax and Semantics –

Christoph Benzmüller and Geoff Sutcliffe

Department of Computer Science, Saarland University

TPTPSYS'08

SS08, Block Course at Saarland University, Germany

Syntax

Syntax

HOL-Syntax: Simple Types

Simple Types $\mathcal{T}$:	o	(truth values)
	ι	(individuals)
	$(\alpha \longrightarrow \beta)$	(functions from α to β)

HOL-Syntax: Simply Typed λ-Terms

Typed Terms:

X_α	Variables ($\mathcal{V}$)
c_α	Constants & Parameters (Σ & $\mathcal{P}$)
$(\mathbf{F}_{\alpha\longrightarrow\beta}\,\mathbf{B}_\alpha)_\beta$	Application
$(\lambda\mathsf{Y}_\alpha\,\mathbf{A}_\beta)_{\alpha\longrightarrow\beta}$	λ-abstraction

HOL-Syntax: Simply Typed λ-Terms

Typed Terms:

X_α	Variables ($\mathcal{V}$)
c_α	Constants & Parameters (Σ & $\mathcal{P}$)
$(\mathbf{F}_{\alpha\longrightarrow\beta}\,\mathbf{B}_\alpha)_\beta$	Application
$(\lambda\mathsf{Y}_\alpha\,\mathbf{A}_\beta)_{\alpha\longrightarrow\beta}$	λ-abstraction

Equality of Terms:

α-conversion	Changing bound variables	
β-reduction	$((\lambda\mathsf{Y}_\beta\,\mathbf{A}_\alpha)\,\mathbf{B}_\beta) \xrightarrow{\beta} [\mathbf{B}/\mathsf{Y}]\mathbf{A}$	
η-reduction	$(\lambda\mathsf{Y}_\alpha\,(\mathbf{F}_{\alpha\longrightarrow\beta}\,\mathsf{Y})) \xrightarrow{\eta} \mathbf{F}$	$(\mathsf{Y}_\beta \notin \mathbf{Free}(\mathbf{F}))$

HOL-Syntax: Simply Typed λ-Terms

Typed Terms:

X_α	Variables ($\mathcal{V}$)
c_α	Constants & Parameters (Σ & $\mathcal{P}$)
$(\mathbf{F}_{\alpha\longrightarrow\beta}\,\mathbf{B}_\alpha)_\beta$	Application
$(\lambda \mathsf{Y}_\alpha\,\mathbf{A}_\beta)_{\alpha\longrightarrow\beta}$	λ-abstraction

Equality of Terms:

Every term has a unique $\beta\eta$-normal form (up to α-conversion).

HOL: Adding Logical Connectives

- $\top_o$ – true
- $\bot_o$ – false
- $\neg_{o\to o}$ – negation
- $\vee_{o\to o\to o}$ – disjunction
- $\wedge_{o\to o\to o}$ – conjunction
- $\supset_{o\to o\to o}$ – implication
- $\Leftrightarrow_{o\to o\to o}$ – equivalence
- $\forall X_\alpha\text{.} \ldots$ – universal quantification over type α ($\forall$ types α)
- $\exists X_\alpha\text{.} \ldots$ – existential quantification over type α ($\forall$ types α)
- $=_{\alpha\to\alpha\to o}$ – equality at type α ($\forall$ types α)

HOL: Adding Logical Constants to Σ

One minimal choice for signature Σ:

- $\neg_{o \rightarrow o}$ – negation
- $\vee_{o \rightarrow o \rightarrow o}$ – disjunction
- $\Pi_{(\alpha \rightarrow o) \rightarrow o}$ – universal quantification over type α ($\forall$ types α)

HOL: Adding Logical Constants to Σ

One minimal choice for signature Σ:

- $\neg_{o\to o}$ – negation
- $\vee_{o\to o\to o}$ – disjunction
- $\Pi_{(\alpha\to o)\to o}$ – universal quantification over type α (∀ types α)

Use abbreviations for other logical operators

$\mathbf{A} \vee \mathbf{B}$	means	$(\vee\, \mathbf{A}\, \mathbf{B})$
$\mathbf{A} \wedge \mathbf{B}$	means	$\neg(\neg\mathbf{A} \vee \neg\mathbf{B})$
$\mathbf{A} \supset \mathbf{B}$	means	$\neg\mathbf{A} \vee \mathbf{B}$
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	$(\mathbf{A} \supset \mathbf{B}) \wedge (\mathbf{B} \supset \mathbf{A})$
$\forall X_\alpha\, \mathbf{A}$	means	$\Pi(\lambda X_\alpha\, \mathbf{A})$
$\exists X_\alpha\, \mathbf{A}$	means	$\neg\forall X_\alpha\, \neg\mathbf{A}$
$\mathtt{T}$	means	$(\forall X_o\, X \vee \neg X)$
$\mathtt{F}$	means	$\neg\mathtt{T}$

HOL: Adding Logical Constants to Σ

One minimal choice for signature Σ:

- $\neg_{o\rightarrow o}$ – negation
- $\vee_{o\rightarrow o\rightarrow o}$ – disjunction
- $\Pi_{(\alpha\rightarrow o)\rightarrow o}$ – universal quantification over type α ($\forall$ types α)

Use Leibniz-equality to encode equality

$\mathbf{A}_\alpha \doteq \mathbf{B}_\alpha$ means $\forall P_{\alpha\rightarrow o}(P\,\mathbf{A} \supset P\,\mathbf{B})$

resp. $\Pi(\lambda P_{\alpha\rightarrow o}(\neg P\mathbf{A} \vee P\mathbf{B}))$

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\rightarrow\alpha\rightarrow o}$ – equality

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\rightarrow\alpha\rightarrow o}$ – equality

Use abbreviations for other logical operators

$\mathtt{T}$	means
$\mathtt{F}$	means
$\neg\mathbf{A}$	means
$\mathbf{A} \wedge \mathbf{B}$	means
$\mathbf{A} \vee \mathbf{B}$	means
$\mathbf{A} \supset \mathbf{B}$	means
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means
$\forall X\, \mathbf{A}$	means
$\exists X\, \mathbf{A}$	means

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\to\alpha\to o}$ – equality

Use abbreviations for other logical operators

$\mathtt{T}$	means	$(\lambda X_o X) = (\lambda X_o X)$
$\mathtt{F}$	means	
$\neg\mathbf{A}$	means	
$\mathbf{A} \wedge \mathbf{B}$	means	
$\mathbf{A} \vee \mathbf{B}$	means	
$\mathbf{A} \supset \mathbf{B}$	means	
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	
$\forall X\, \mathbf{A}$	means	
$\exists X\, \mathbf{A}$	means	

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\to\alpha\to o}$ – equality

Use abbreviations for other logical operators

$\mathtt{T}$	means	$(\lambda X_o\, X) = (\lambda X_o\, X)$
$\mathtt{F}$	means	$(\lambda X_o\, \mathtt{T}) = (\lambda X_o\, X)$
$\neg\mathbf{A}$	means	
$\mathbf{A} \wedge \mathbf{B}$	means	
$\mathbf{A} \vee \mathbf{B}$	means	
$\mathbf{A} \supset \mathbf{B}$	means	
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	
$\forall X\, \mathbf{A}$	means	
$\exists X\, \mathbf{A}$	means	

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\to\alpha\to o}$ – equality

Use abbreviations for other logical operators

$\mathtt{T}$	means	$(\lambda X_o\, X) = (\lambda X_o\, X)$
$\mathtt{F}$	means	$(\lambda X_o\, \mathtt{T}) = (\lambda X_o\, X)$
$\neg\mathbf{A}$	means	$\mathbf{A} = \mathtt{F}$
$\mathbf{A} \wedge \mathbf{B}$	means	
$\mathbf{A} \vee \mathbf{B}$	means	
$\mathbf{A} \supset \mathbf{B}$	means	
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	
$\forall X\, \mathbf{A}$	means	
$\exists X\, \mathbf{A}$	means	

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\to\alpha\to o}$ – equality

Use abbreviations for other logical operators

$\mathtt{T}$	means	$(\lambda X_o\, X) = (\lambda X_o\, X)$
$\mathtt{F}$	means	$(\lambda X_o\, \mathtt{T}) = (\lambda X_o\, X)$
$\neg \mathbf{A}$	means	$\mathbf{A} = \mathtt{F}$
$\mathbf{A} \wedge \mathbf{B}$	means	$(\lambda F_{o\to o\to o}\,(F\,\mathtt{T}\,\mathtt{T})) = (\lambda F_{o\to o\to o}\,(F\,\mathbf{A}\,\mathbf{B}))$
$\mathbf{A} \vee \mathbf{B}$	means	
$\mathbf{A} \supset \mathbf{B}$	means	
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	
$\forall X\, \mathbf{A}$	means	
$\exists X\, \mathbf{A}$	means	

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\to\alpha\to o}$ – equality

Use abbreviations for other logical operators

$\mathtt{T}$	means	$(\lambda X_o\, X) = (\lambda X_o\, X)$
$\mathtt{F}$	means	$(\lambda X_o\, \mathtt{T}) = (\lambda X_o\, X)$
$\neg\mathbf{A}$	means	$\mathbf{A} = \mathtt{F}$
$\mathbf{A} \wedge \mathbf{B}$	means	$(\lambda F_{o\to o\to o}\,(F\,\mathtt{T}\,\mathtt{T})) = (\lambda F_{o\to o\to o}\,(F\,\mathbf{A}\,\mathbf{B}))$
$\mathbf{A} \vee \mathbf{B}$	means	$\neg(\neg\mathbf{A} \wedge \neg\mathbf{B})$
$\mathbf{A} \supset \mathbf{B}$	means	
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	
$\forall X\,\mathbf{A}$	means	
$\exists X\,\mathbf{A}$	means	

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\rightarrow\alpha\rightarrow o}$ – equality

Use abbreviations for other logical operators

$\mathtt{T}$	means	$(\lambda X_o\, X) = (\lambda X_o\, X)$
$\mathtt{F}$	means	$(\lambda X_o\, \mathtt{T}) = (\lambda X_o\, X)$
$\neg\mathbf{A}$	means	$\mathbf{A} = \mathtt{F}$
$\mathbf{A} \wedge \mathbf{B}$	means	$(\lambda F_{o\rightarrow o\rightarrow o}\, (F\, \mathtt{T}\, \mathtt{T})) = (\lambda F_{o\rightarrow o\rightarrow o}\, (F\, \mathbf{A}\, \mathbf{B}))$
$\mathbf{A} \vee \mathbf{B}$	means	$\neg(\neg\mathbf{A} \wedge \neg\mathbf{B})$
$\mathbf{A} \supset \mathbf{B}$	means	$\neg\mathbf{A} \vee \mathbf{B}$
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	
$\forall X\, \mathbf{A}$	means	
$\exists X\, \mathbf{A}$	means	

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\rightarrow\alpha\rightarrow o}$ – equality

Use abbreviations for other logical operators

$\mathtt{T}$	means	$(\lambda X_o\, X) = (\lambda X_o\, X)$
$\mathtt{F}$	means	$(\lambda X_o\, \mathtt{T}) = (\lambda X_o\, X)$
$\neg\mathbf{A}$	means	$\mathbf{A} = \mathtt{F}$
$\mathbf{A} \wedge \mathbf{B}$	means	$(\lambda F_{o\rightarrow o\rightarrow o}\, (F\,\mathtt{T}\,\mathtt{T})) = (\lambda F_{o\rightarrow o\rightarrow o}\, (F\,\mathbf{A}\,\mathbf{B}))$
$\mathbf{A} \vee \mathbf{B}$	means	$\neg(\neg\mathbf{A} \wedge \neg\mathbf{B})$
$\mathbf{A} \supset \mathbf{B}$	means	$\neg\mathbf{A} \vee \mathbf{B}$
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	$\mathbf{A} = \mathbf{B}$
$\forall X\, \mathbf{A}$	means	
$\exists X\, \mathbf{A}$	means	

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\rightarrow\alpha\rightarrow o}$ – equality

Use abbreviations for other logical operators

$\mathtt{T}$	means	$(\lambda X_o\, X) = (\lambda X_o\, X)$
$\mathtt{F}$	means	$(\lambda X_o\, \mathtt{T}) = (\lambda X_o\, X)$
$\neg\mathbf{A}$	means	$\mathbf{A} = \mathtt{F}$
$\mathbf{A} \wedge \mathbf{B}$	means	$(\lambda F_{o\rightarrow o\rightarrow o}\, (F\, \mathtt{T}\, \mathtt{T})) = (\lambda F_{o\rightarrow o\rightarrow o}\, (F\, \mathbf{A}\, \mathbf{B}))$
$\mathbf{A} \vee \mathbf{B}$	means	$\neg(\neg\mathbf{A} \wedge \neg\mathbf{B})$
$\mathbf{A} \supset \mathbf{B}$	means	$\neg\mathbf{A} \vee \mathbf{B}$
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	$\mathbf{A} = \mathbf{B}$
$\forall X\, \mathbf{A}$	means	$(\lambda X_o\, \mathbf{A}) = (\lambda X_o\, \mathtt{T})$
$\exists X\, \mathbf{A}$	means	

HOL: Adding Logical Constants to Σ

Another minimal choice for signature Σ:

- $=_{\alpha\to\alpha\to o}$ – equality

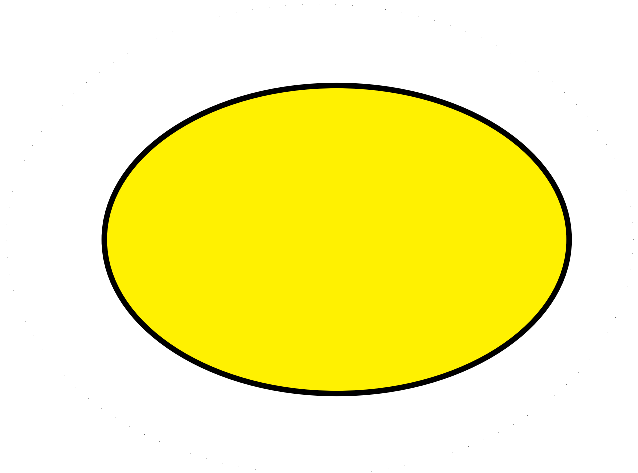
Use abbreviations for other logical operators

$\mathtt{T}$	means	$(\lambda X_o\, X) = (\lambda X_o\, X)$
$\mathtt{F}$	means	$(\lambda X_o\, \mathtt{T}) = (\lambda X_o\, X)$
$\neg\mathbf{A}$	means	$\mathbf{A} = \mathtt{F}$
$\mathbf{A} \wedge \mathbf{B}$	means	$(\lambda F_{o\to o\to o}\,(F\,\mathtt{T}\,\mathtt{T})) = (\lambda F_{o\to o\to o}\,(F\,\mathbf{A}\,\mathbf{B}))$
$\mathbf{A} \vee \mathbf{B}$	means	$\neg(\neg\mathbf{A} \wedge \neg\mathbf{B})$
$\mathbf{A} \supset \mathbf{B}$	means	$\neg\mathbf{A} \vee \mathbf{B}$
$\mathbf{A} \Leftrightarrow \mathbf{B}$	means	$\mathbf{A} = \mathbf{B}$
$\forall X\, \mathbf{A}$	means	$(\lambda X_o\, \mathbf{A}) = (\lambda X_o\, \mathtt{T})$
$\exists X\, \mathbf{A}$	means	$\neg(\forall X_\alpha\, \neg\mathbf{A})$

Semantics

Semantics: Model Classes
(different extensionality
properties)

Model Classes (Extensionality)



Standard Models $\mathfrak{ST}(\Sigma)$

- Idea of Standard Semantics:

$$\iota \longrightarrow \mathcal{D}_\iota \qquad \text{(choose)}$$

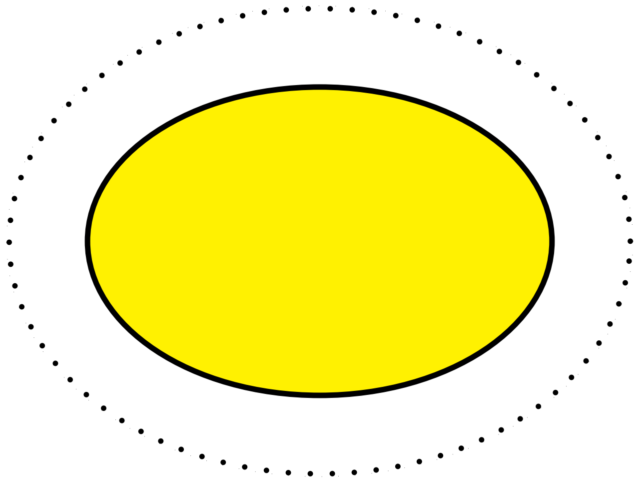
$$o \longrightarrow \mathcal{D}_o = \{\mathtt{T}, \mathtt{F}\} \qquad \text{(fixed)}$$

$$(\alpha \rightarrow \beta) \longrightarrow$$

$$\mathcal{D}_{\alpha \rightarrow \beta} = \mathcal{F}(\mathcal{D}_\alpha, \mathcal{D}_\beta) \qquad \text{(fixed)}$$

(undecidable)

Model Classes (Extensionality)



Standard Models $\mathfrak{ST}(\Sigma)$

- Idea of Standard Semantics:

  $\iota \longrightarrow \mathcal{D}_\iota$ (choose)

  $o \longrightarrow \mathcal{D}_o = \{\mathtt{T}, \mathtt{F}\}$ (fixed)

  $(\alpha \rightarrow \beta) \longrightarrow$

  $\mathcal{D}_{\alpha\rightarrow\beta} = \mathcal{F}(\mathcal{D}_\alpha, \mathcal{D}_\beta)$ (fixed)

  (undecidable)

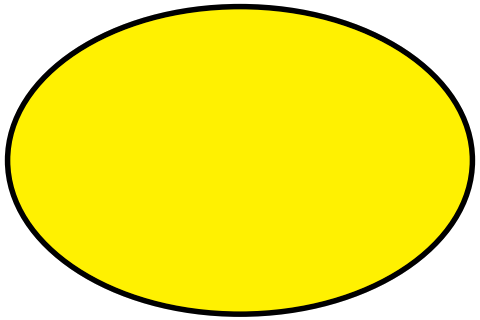
- Henkin's Generalization:

  $\mathcal{D}_{\alpha\rightarrow\beta} \subseteq \mathcal{F}(\mathcal{D}_\alpha, \mathcal{D}_\beta)$ (choose)

  but elements are still functions and Denotatspflicht holds (semi-decidable)

  [Henkin-50]

Model Classes (Extensionality)

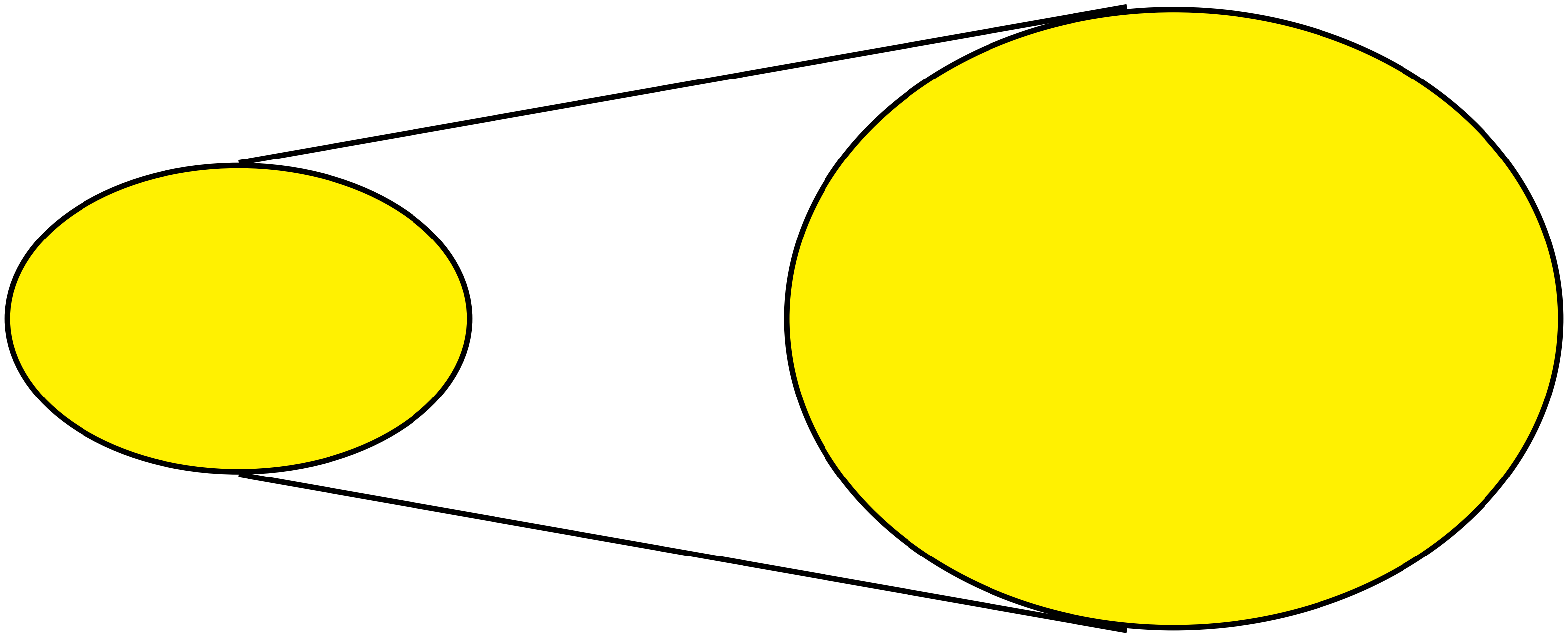


Standard Models $\mathfrak{ST}(\Sigma)$

choose: $\mathcal{D}_\iota$

fixed: $\mathcal{D}_o, \mathcal{D}_{\alpha\to\beta}$, functions

Model Classes (Extensionality)



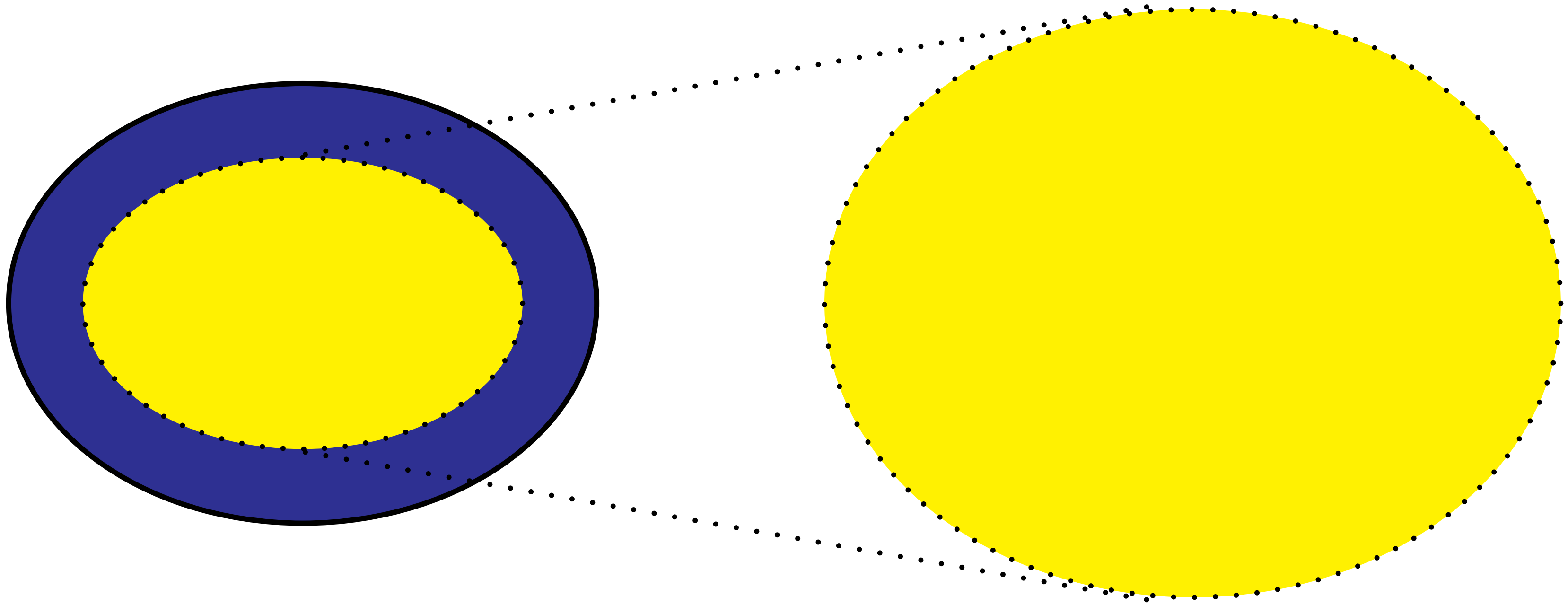
Standard Models $\mathfrak{ST}(\Sigma)$

Formulas valid in $\mathfrak{ST}(\Sigma)$

choose: $\mathcal{D}_\iota$

fixed: $\mathcal{D}_o, \mathcal{D}_{\alpha\rightarrow\beta}$, functions

Model Classes (Extensionality)



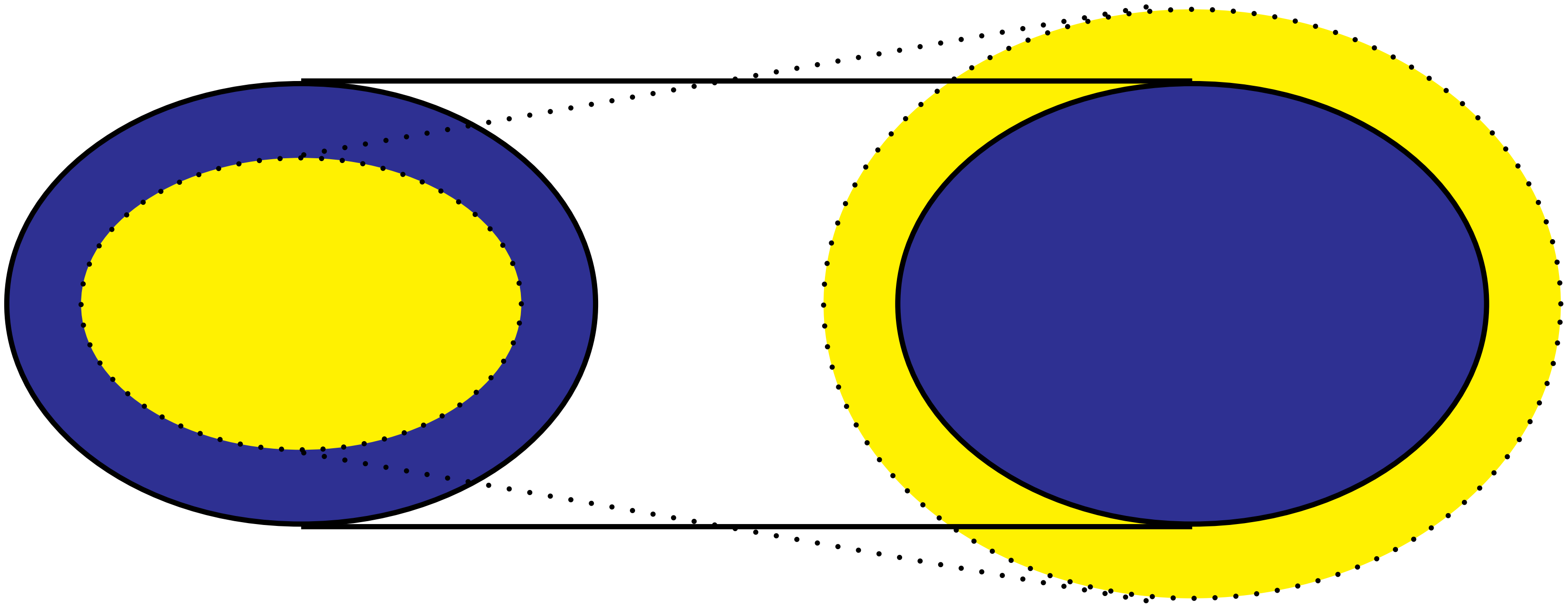
Henkin Models $\mathfrak{H}(\Sigma) = \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$

Formulas valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$?

choose: $\mathcal{D}_\iota, \mathcal{D}_{\alpha\rightarrow\beta}$

fixed: $\mathcal{D}_o$, functions

Model Classes (Extensionality)



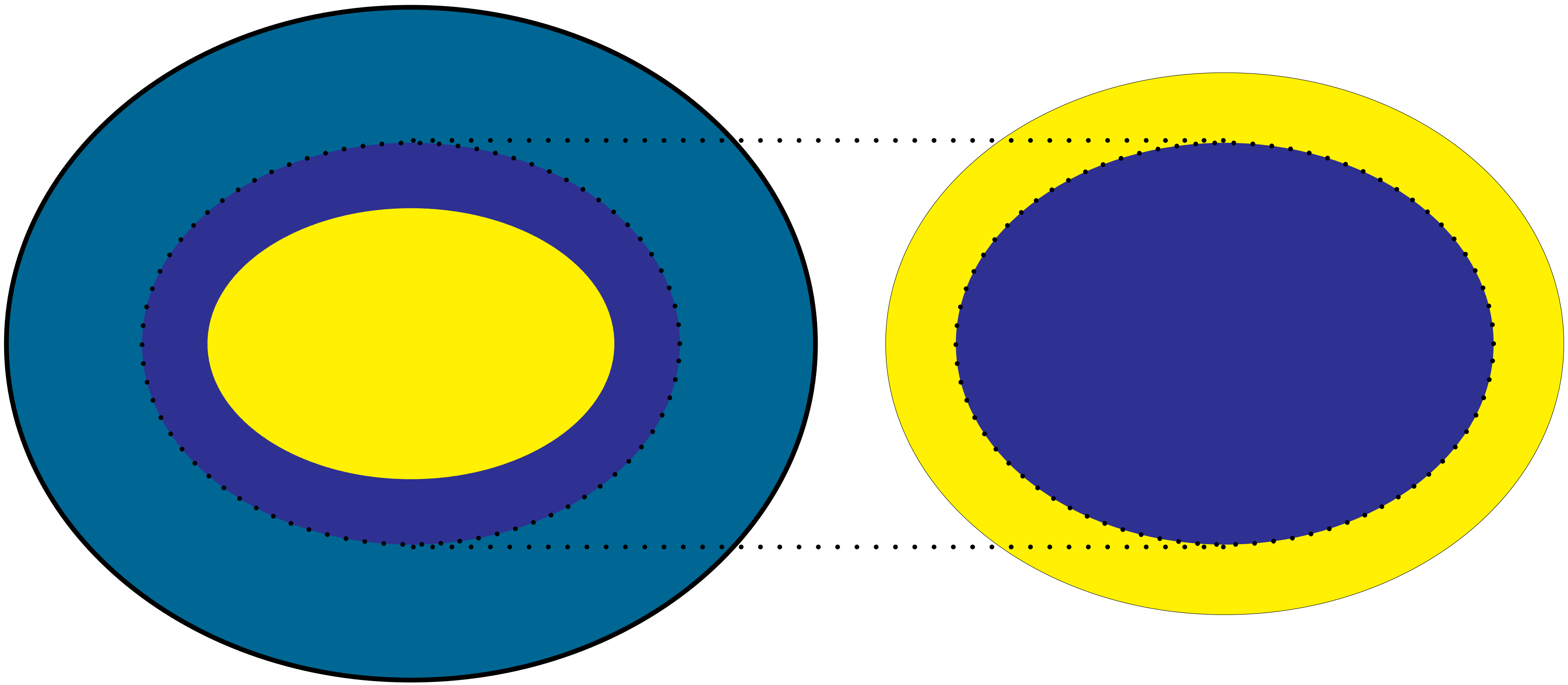
Henkin Models $\mathfrak{H}(\Sigma) = \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$

Formulas valid in $\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma)$

choose: $\mathcal{D}_\iota, \mathcal{D}_{\alpha\to\beta}$

fixed: $\mathcal{D}_o$, functions

Model Classes (Extensionality)



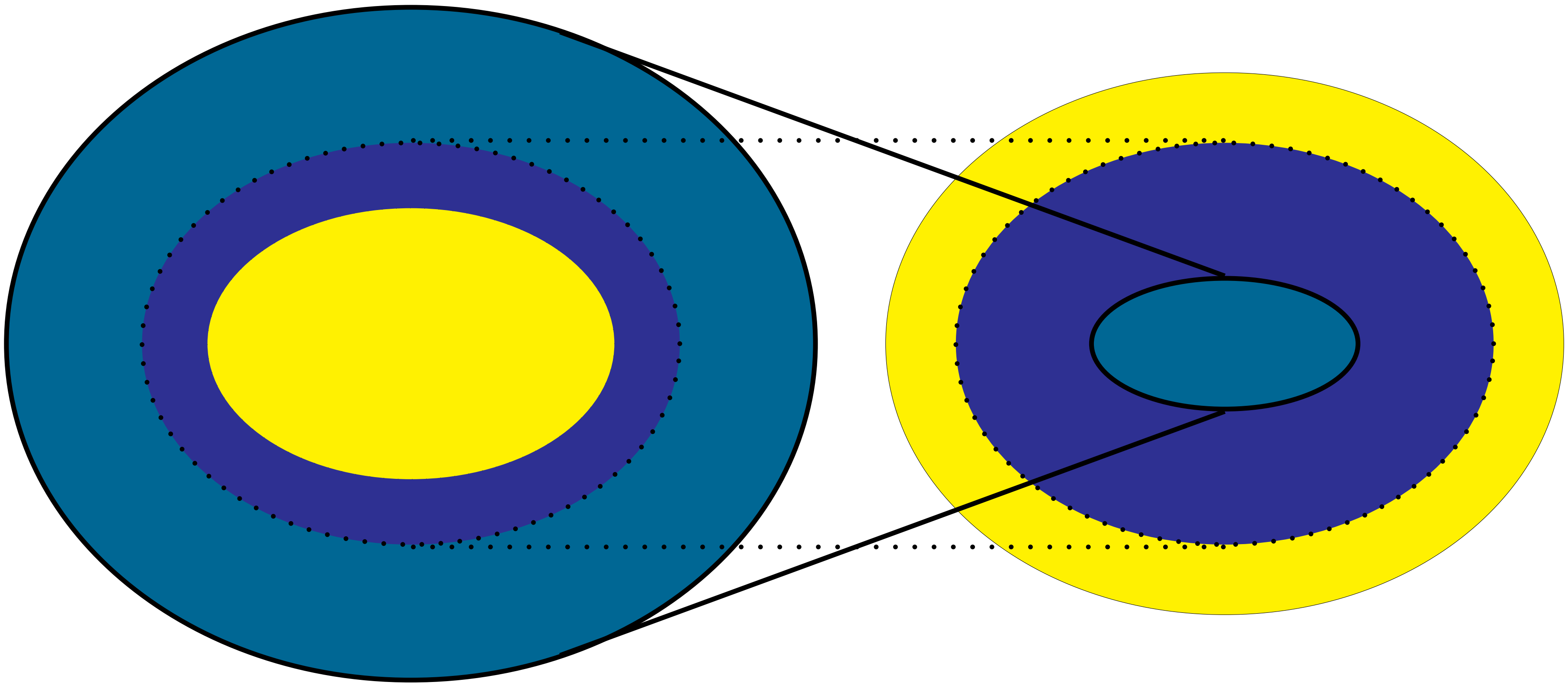
Non-Extensional Models $\mathfrak{M}_\beta(\Sigma)$

Formulas valid in $\mathfrak{M}_\beta(\Sigma)$?

choose: $\mathcal{D}_\iota, \mathcal{D}_{\alpha\to\beta}$, also non−functions, $\mathcal{D}_o$

fixed:

Model Classes (Extensionality)



Non-Extensional Models $\mathfrak{M}_\beta(\Sigma)$

Formulas valid in $\mathfrak{M}_\beta(\Sigma)$?

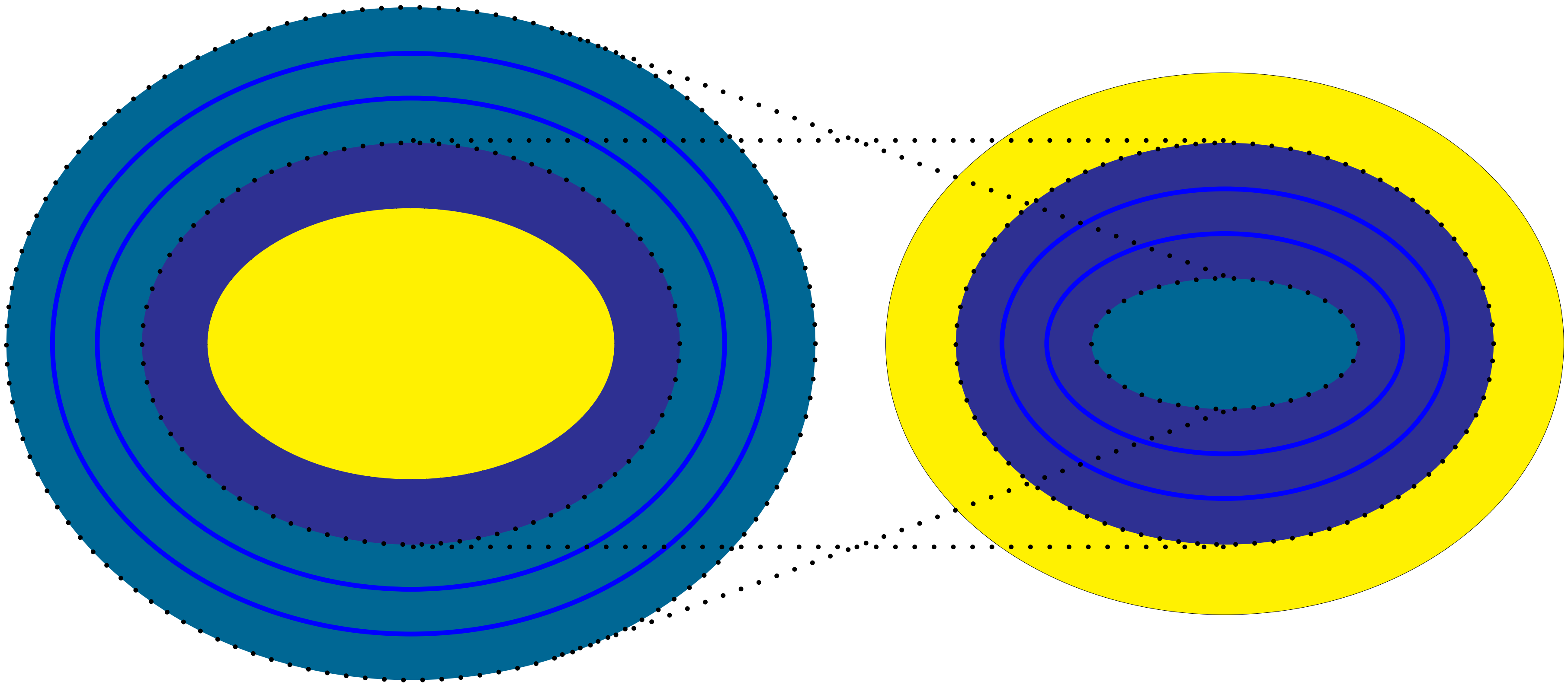
choose: $\mathcal{D}_\iota, \mathcal{D}_{\alpha\to\beta}$, also non−functions, $\mathcal{D}_o$

fixed:

Ex.: $\forall X. \forall Y. X \vee Y \Leftrightarrow Y \vee X$

vs. $\vee \doteq \lambda X. \lambda Y. Y \vee X$

Model Classes (Extensionality)



We additionally studied different model classes with 'varying degrees of extensionality'

$$\forall X.\forall Y.X \vee Y \Leftrightarrow Y \vee X \qquad \forall X.\forall Y.X \vee Y \doteq Y \vee X$$

$$\lambda X.\lambda Y.X \vee Y \doteq \lambda X.\lambda Y.Y \vee X \qquad \vee \doteq \lambda X.\lambda Y.Y \vee X$$

Model Classes (Extensionality)

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

$\mathfrak{b}$: Boolean extensionality, $\mathcal{D}_o = \{\mathtt{T}, \mathtt{F}\}$

$\mathfrak{f}(=\eta+\xi)$: functional extensionality

η: η-functional

ξ: ξ-functionality

$\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma)$

full

$\mathfrak{ST}(\Sigma)$

$\mathfrak{M}_\beta(\Sigma)$ non-extensional Σ-models

$\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma)$ Henkin models

Model Classes (Extensionality)

$$
\begin{tikzcd}
\mathfrak{M}_{\beta}(\Sigma) \arrow[dr, "\mathfrak{b}"] \arrow[dd, dotted, "{\mathfrak{b}, \mathfrak{f}(= \eta + \xi)}"'] & \\
& \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dl, "\mathfrak{f}"] \\
\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, dotted, "\text{full}"] & \\
\mathfrak{ST}(\Sigma) &
\end{tikzcd}
$$

Motivation for
Models without Functional Extensionality

- modeling programs:
  $p_1 \neq p_2$ even if $p_1@a = p_2@a$ for every $a \in \mathcal{D}_\alpha$
- consider, e.g., run-time complexity:
  $p_1 \leftarrow \lambda X.1$
  and
  $p_2 \leftarrow \lambda X.1 + (X+1)^2 - (X^2 + 2X + 1)$

Model Classes (Extensionality)

$$
\begin{tikzcd}
\mathfrak{M}_\beta(\Sigma) \arrow[dr, "\mathfrak{b}"] \arrow[dd, dotted, "{\mathfrak{b}, \mathfrak{f}(= \eta + \xi)}"'] & \\
& \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dl, "\mathfrak{f}"] \\
\mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, dotted, "\text{full}"] & \\
\mathfrak{ST}(\Sigma) &
\end{tikzcd}
$$

Motivation for
Models without Boolean Extensionality?

- modeling of intensional concepts like 'knowledge', 'believe', etc.
- example:
  $\mathbf{O} := 2 + 2 = 4$
  $\mathbf{F} := \forall x, y, z, n > 2.x^n + y^n = z^n \Rightarrow x = y = z = 0$
  We want to model:
  $\mathbf{O} \Leftrightarrow \mathbf{F}$ but
  $\text{john\_knows}(\mathbf{F}) \not\Leftrightarrow \text{john\_knows}(\mathbf{O})$
- if we have $\mathcal{D}_o = \{\mathtt{T}, \mathtt{F}\}$ then
  $\mathbf{O} \Leftrightarrow \mathbf{F}$ implies $\mathbf{O} = \mathbf{F}$
  which also enforces
  $\text{john\_knows}(\mathbf{F}) \Leftrightarrow \text{john\_knows}(\mathbf{O})$

Model Classes (Extensionality)

$$
\begin{tikzcd}
 & \mathfrak{M}_{\beta}(\Sigma) \arrow[dl, "\mathfrak{f}"'] \arrow[dr, "\mathfrak{b}"] \arrow[dd, dotted, "{\mathfrak{b}, \mathfrak{f}(= \eta + \xi)}"] & \\
\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \arrow[dr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dl, "\mathfrak{f}"'] \\
 & \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, dotted, "\text{full}"] & \\
 & \mathfrak{S}\mathfrak{T}(\Sigma) &
\end{tikzcd}
$$

Models without η

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

Model Classes (Extensionality)

$$
\begin{tikzcd}
& \mathfrak{M}_{\beta}(\Sigma) \arrow[dl, "\mathfrak{f}"'] \arrow[dr, "\mathfrak{b}"] \arrow[dd, dotted, "{\mathfrak{b},\mathfrak{f}(=\eta+\xi)}"] & \\
\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \arrow[dr, "\mathfrak{b}"'] & & \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dl, "\mathfrak{f}"] \\
& \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, dotted, "\text{full}"] & \\
& \mathfrak{S}\mathfrak{T}(\Sigma) &
\end{tikzcd}
$$

Models without η

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

Model Classes (Extensionality)

$$
\begin{tikzcd}
 & \mathfrak{M}_{\beta}(\Sigma) \arrow[dl, "\xi"'] \arrow[ddl, dotted, "\mathfrak{f}"] \arrow[dr, "\mathfrak{b}"] & \\
\mathfrak{M}_{\beta\xi}(\Sigma) \arrow[d, "\eta"] & & \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[ddl, "\mathfrak{f}"] \\
\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \arrow[dr, "\mathfrak{b}"'] & & \\
 & \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, dotted, "\text{full}"] & \\
 & \mathfrak{ST}(\Sigma) &
\end{tikzcd}
$$

Models without ξ

$$\mathcal{E}_{\varphi}(\lambda X_{\alpha}.M_{\beta}) = \mathcal{E}_{\varphi}(\lambda X_{\alpha}.N_{\beta}) \text{ iff}$$
$$\mathcal{E}_{\varphi,[a/X]}(M) = \mathcal{E}_{\varphi,[a/X]}(N) \; (\forall a \in \mathcal{D}_{\alpha})$$

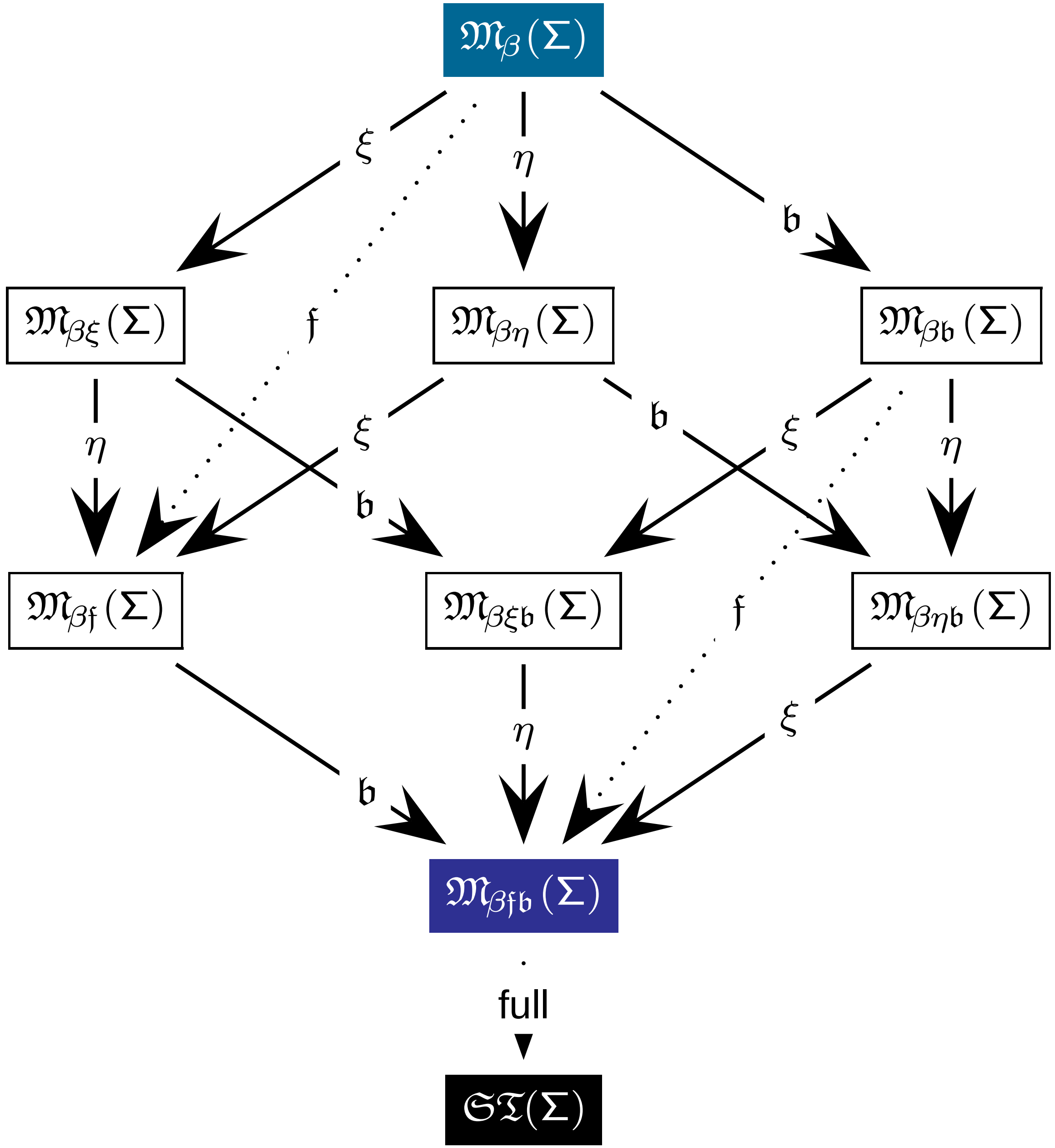
Model Classes (Extensionality)

$$
\begin{tikzcd}
 & & \mathfrak{M}_{\beta}(\Sigma) \arrow[dll, "\xi"'] \arrow[ddll, dotted, "\mathfrak{f}"] \arrow[drr, "\mathfrak{b}"] & & \\
\mathfrak{M}_{\beta\xi}(\Sigma) \arrow[d, "\eta"] & & & & \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dll, "\xi"'] \arrow[ddll, dotted, "\mathfrak{f}"] \\
\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma) \arrow[d, "\eta"] & & \\
 & & \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, dotted, "\text{full}"] & & \\
 & & \mathfrak{ST}(\Sigma) & &
\end{tikzcd}
$$

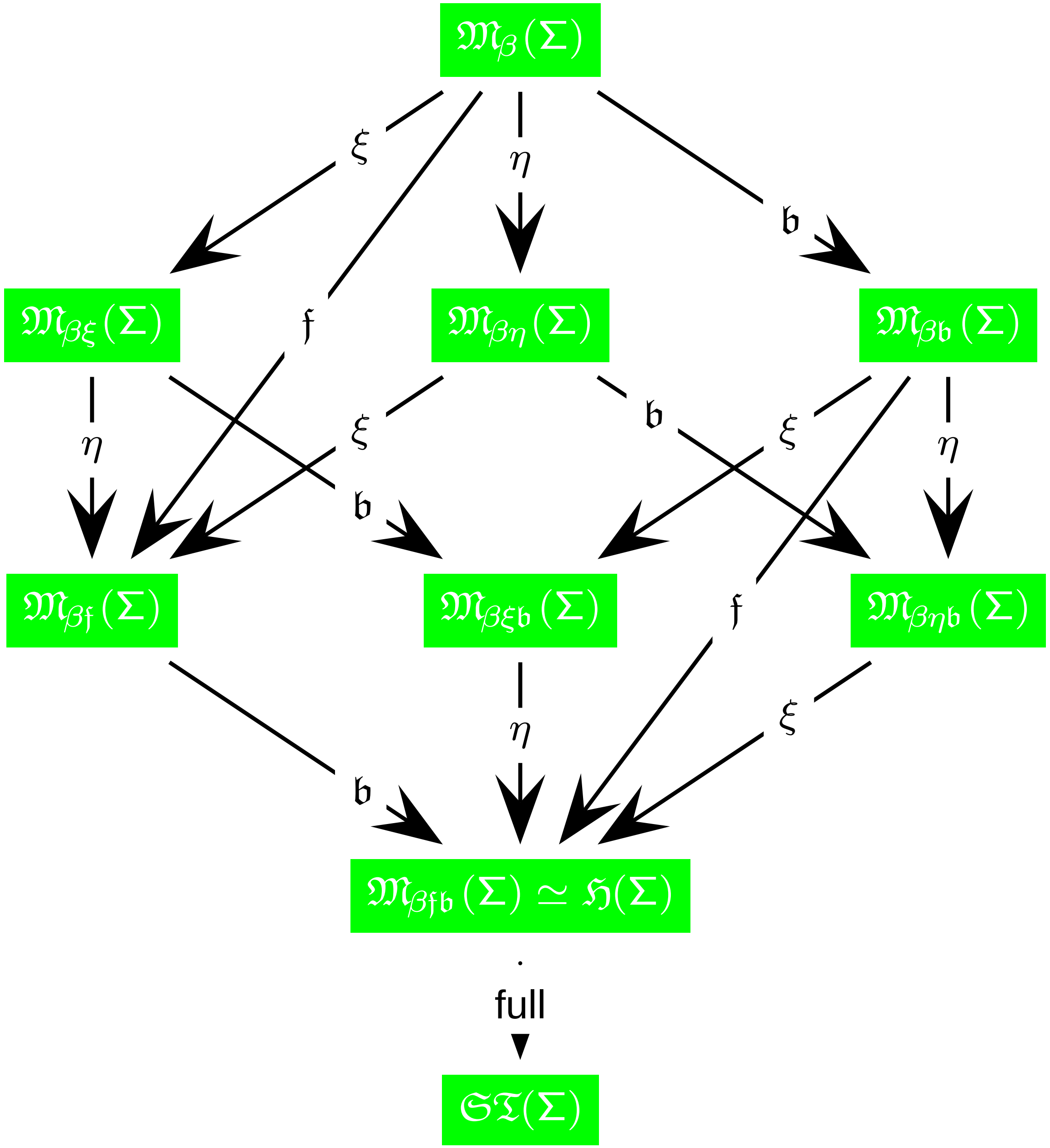
Model Classes (Extensionality)

$$
\begin{tikzcd}[column sep=large, row sep=large]
 & \mathfrak{M}_{\beta}(\Sigma) \arrow[dl, "\xi"'] \arrow[d, "\eta"] \arrow[dr, "\mathfrak{b}"] \arrow[ddl, dotted, "\mathfrak{f}"] & \\
\mathfrak{M}_{\beta\xi}(\Sigma) \arrow[d, "\eta"] & \mathfrak{M}_{\beta\eta}(\Sigma) \arrow[dl, "\xi"'] & \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dl, "\xi"'] \arrow[d, "\eta"] \arrow[ddl, dotted, "\mathfrak{f}"] \\
\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \arrow[dr, "\mathfrak{b}"] & \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma) \arrow[d, "\eta"] & \mathfrak{M}_{\beta\eta\mathfrak{b}}(\Sigma) \arrow[dl, "\xi"'] \\
 & \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) \simeq \mathfrak{H}(\Sigma) \arrow[d, dotted, "\text{full}"] & \\
 & \mathfrak{ST}(\Sigma) &
\end{tikzcd}
$$

Model Classes (Extensionality)



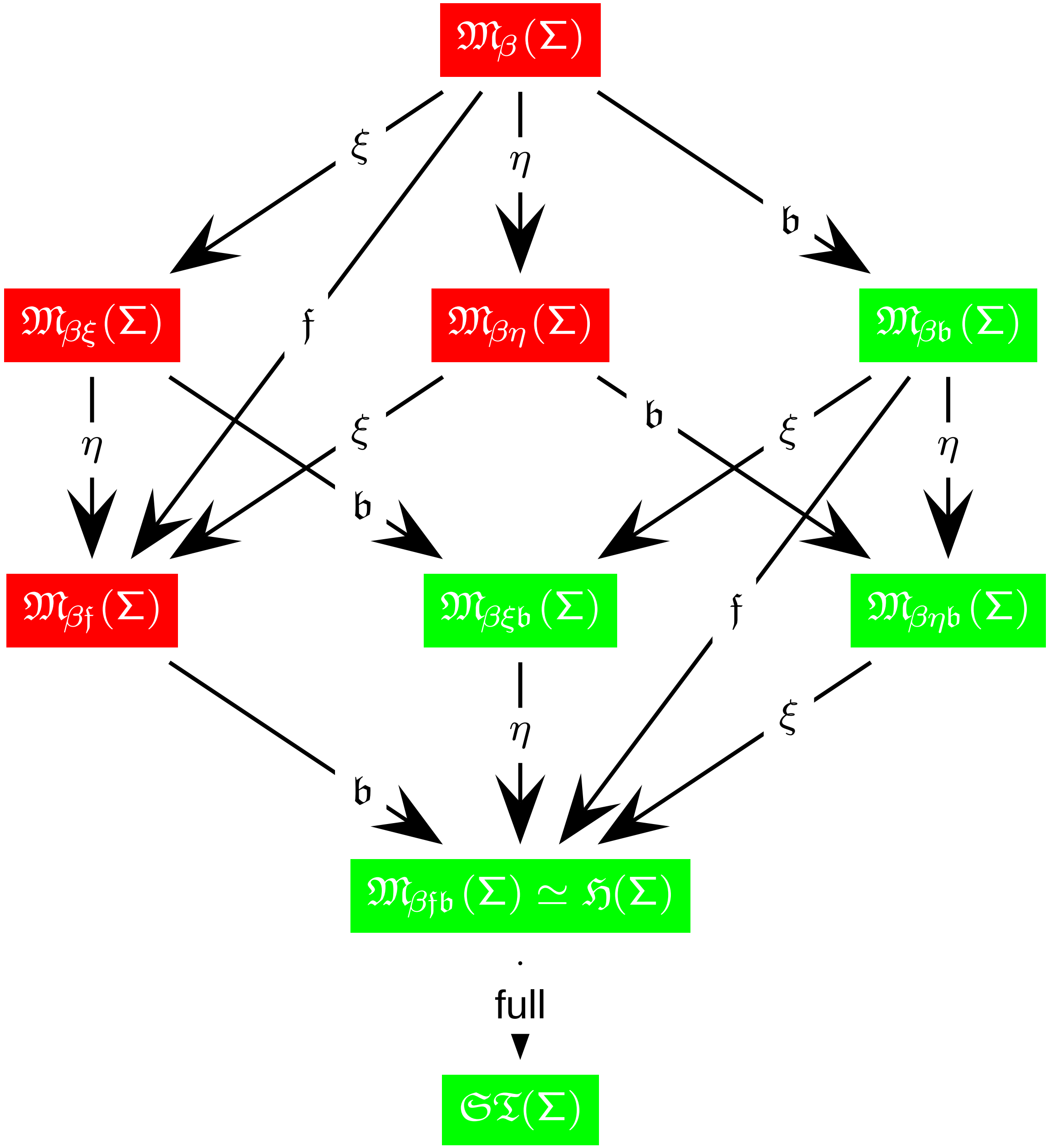
Model Classes (Extensionality)



- $\forall X. \forall Y. X \vee Y \Leftrightarrow Y \vee X$

valid for all model classes

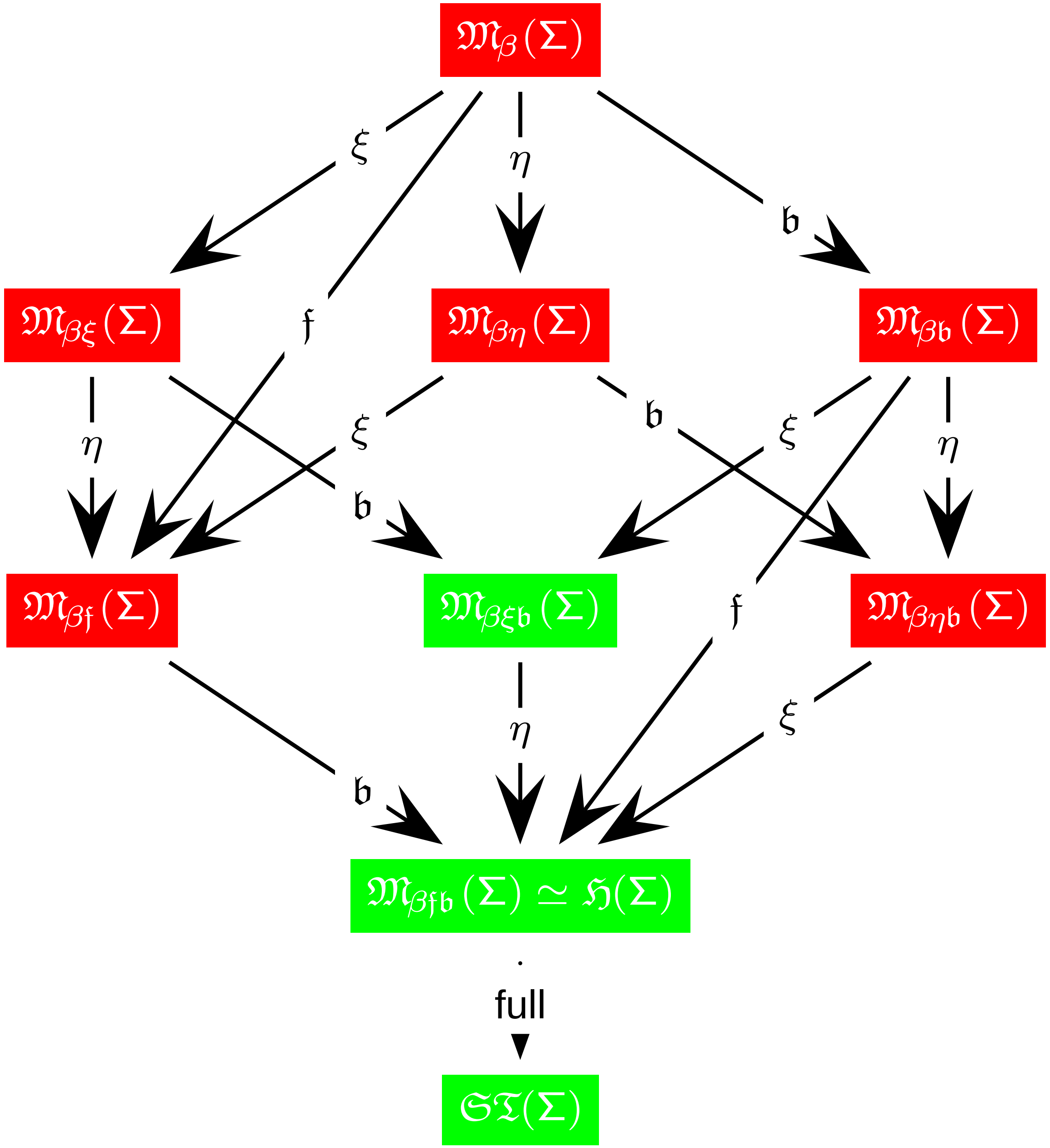
Model Classes (Extensionality)



- $\forall X.\forall Y.X \lor Y \Leftrightarrow Y \lor X$
- $\forall X.\forall Y.X \lor Y \doteq Y \lor X$

validity requires $\mathfrak{b}$

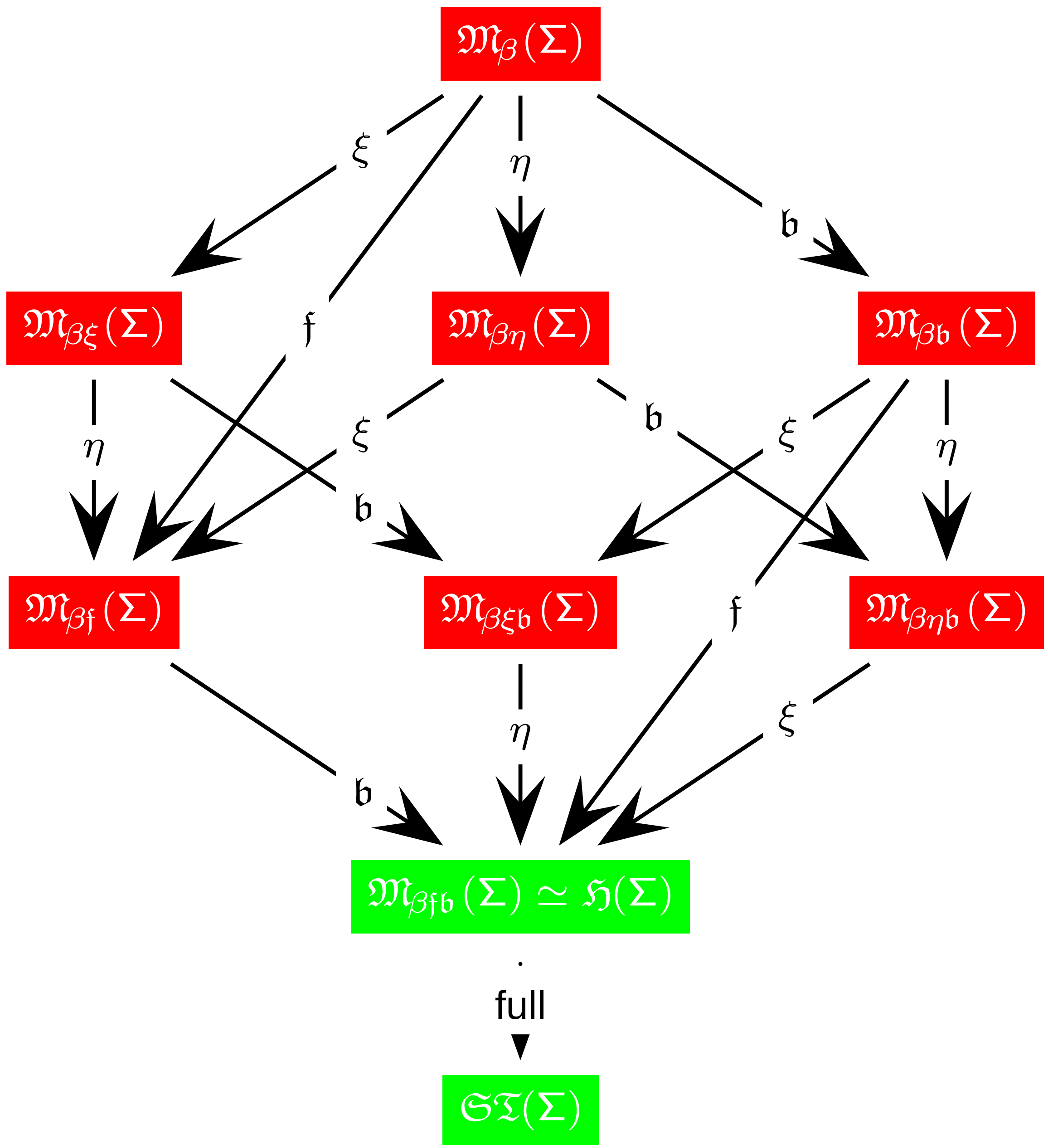
Model Classes (Extensionality)



- $\forall X.\forall Y.X \vee Y \Leftrightarrow Y \vee X$
- $\forall X.\forall Y.X \vee Y \doteq Y \vee X$
- $\lambda X.\lambda Y.X \vee Y \doteq \lambda X.\lambda Y.Y \vee X$

validity requires $\mathfrak{b}$ and ξ

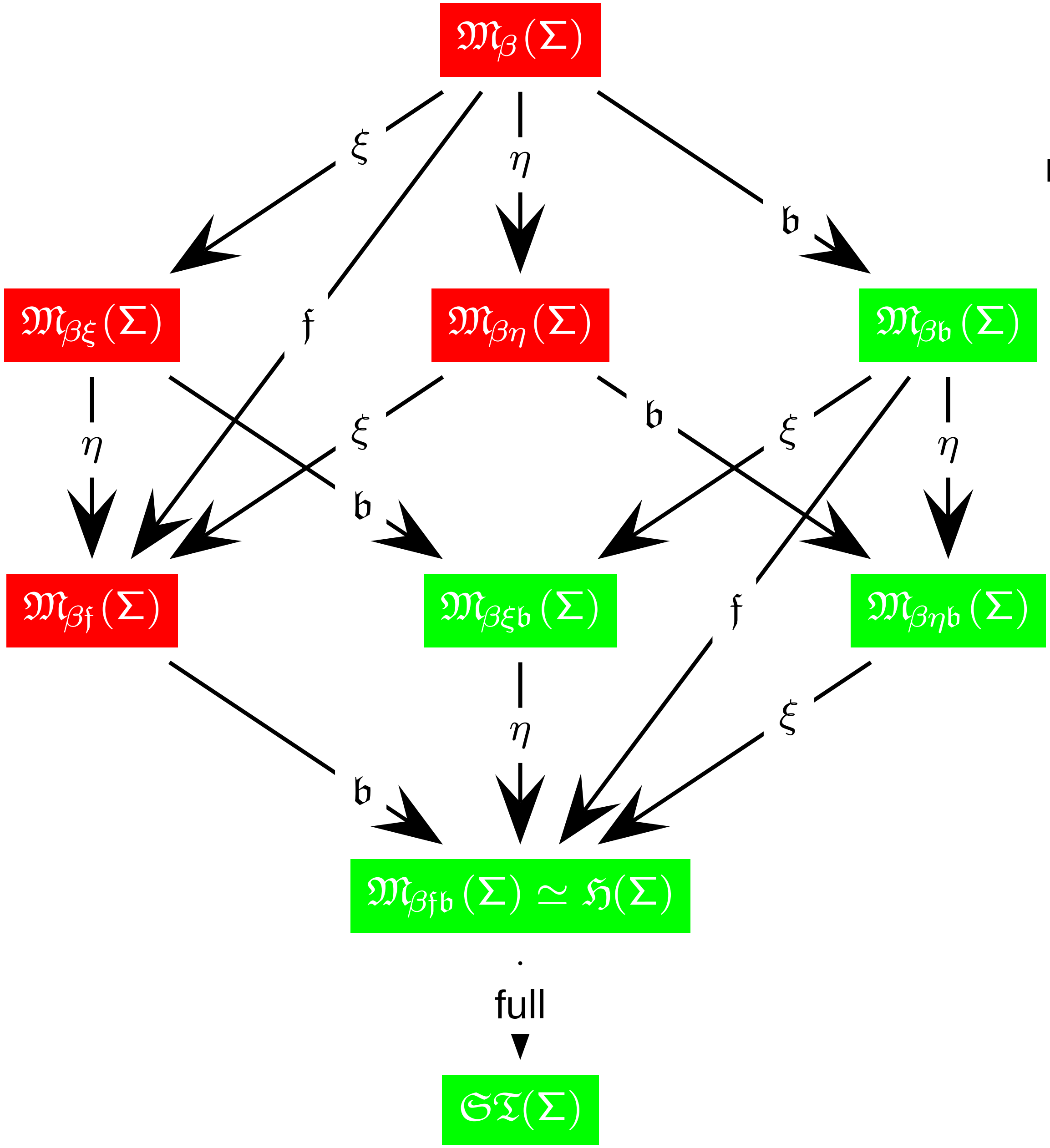
Model Classes (Extensionality)



- $\forall X.\forall Y.X \vee Y \Leftrightarrow Y \vee X$
- $\forall X.\forall Y.X \vee Y \doteq Y \vee X$
- $\lambda X.\lambda Y.X \vee Y \doteq \lambda X.\lambda Y.Y \vee X$
- $\vee \doteq \lambda X.\lambda Y.Y \vee X$

validity requires $\mathfrak{b}$ and $\mathfrak{f}$

Useful: Test Problems for TPs



Examples requiring property $\mathfrak{b}$

- $(p\ a_o) \land (p\ b_o) \Rightarrow (p\ (a \land b))$
- $(h_{o\to\iota}((h\top) \doteq (h\bot))) \doteq (h\bot)$

Semantics - Calculi - Abstract Consistency

Semantics:

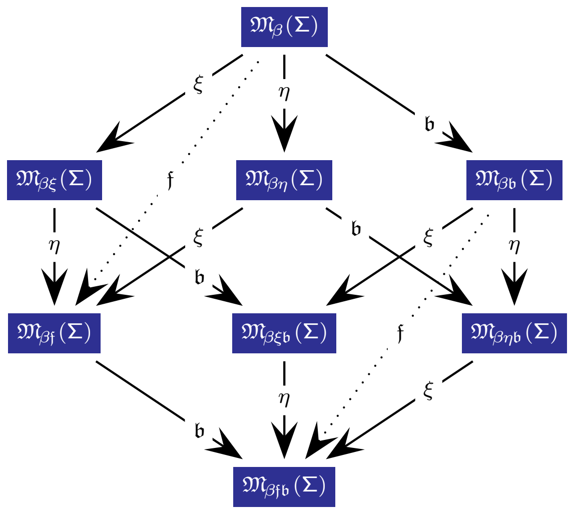
Model Classes (Extensionality)

$$
\begin{tikzcd}
 & & \mathfrak{M}_{\beta}(\Sigma) \arrow[dll, "\xi"'] \arrow[d, "\eta"] \arrow[drr, "\mathfrak{b}"] \arrow[ddll, dotted, "\mathfrak{f}"] & & \\
\mathfrak{M}_{\beta\xi}(\Sigma) \arrow[d, "\eta"'] \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\eta}(\Sigma) \arrow[dll, "\xi"'] \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\mathfrak{b}}(\Sigma) \arrow[dll, "\xi"'] \arrow[d, "\eta"] \arrow[ddll, dotted, "\mathfrak{f}"] \\
\mathfrak{M}_{\beta\mathfrak{f}}(\Sigma) \arrow[drr, "\mathfrak{b}"] & & \mathfrak{M}_{\beta\xi\mathfrak{b}}(\Sigma) \arrow[d, "\eta"] & & \mathfrak{M}_{\beta\eta\mathfrak{b}}(\Sigma) \arrow[dll, "\xi"'] \\
 & & \mathfrak{M}_{\beta\mathfrak{f}\mathfrak{b}}(\Sigma) & &
\end{tikzcd}
$$

Semantics - Calculi - Abstract Consistency

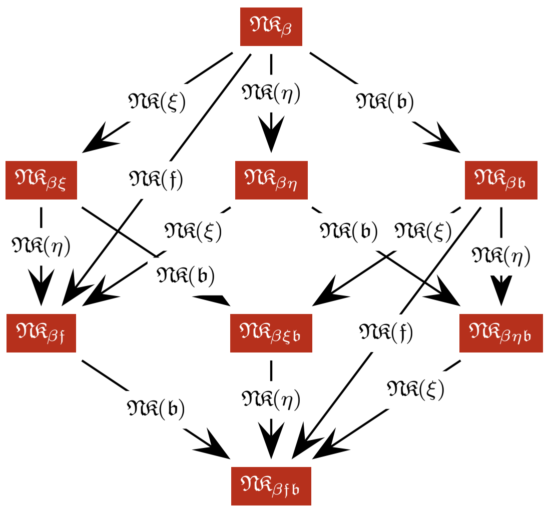
Semantics:

Model Classes (Extensionality)



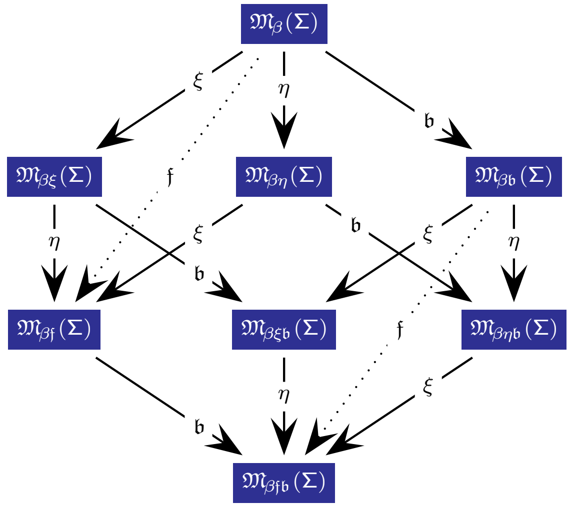
Reference Calculi:

ND (and others)

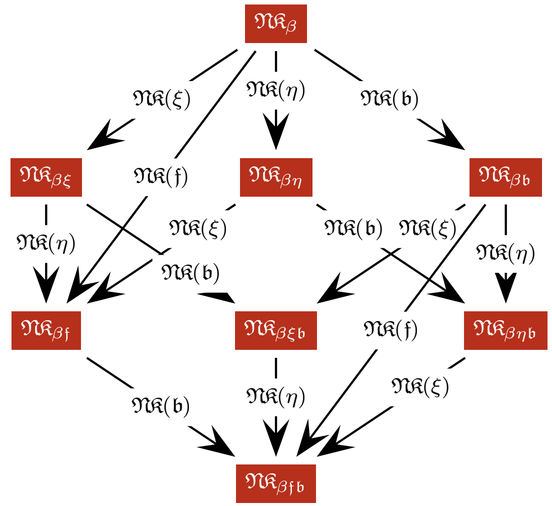


Semantics - Calculi - Abstract Consistency

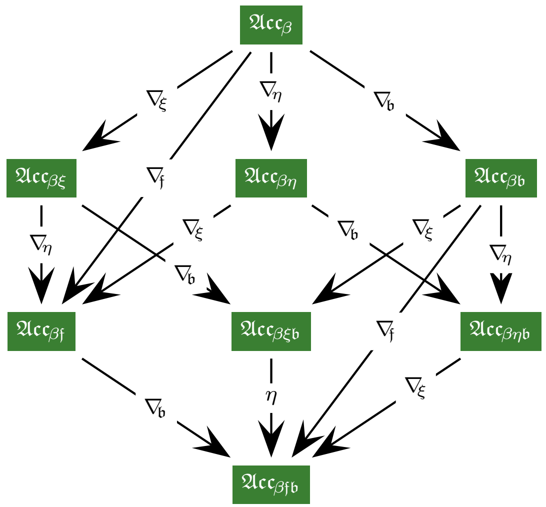
Semantics:
Model Classes (Extensionality)



Reference Calculi:
ND (and others)



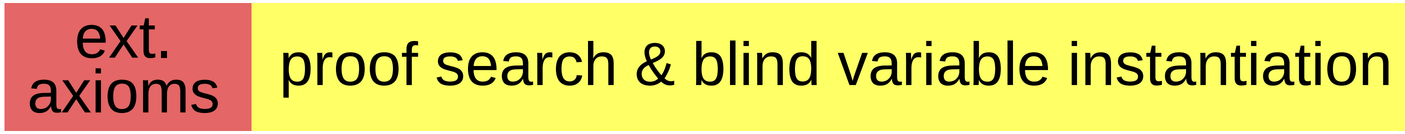
Abstract Consistency / Unifying Principle:
Extensions of Smullyan-63 and Andrews-71



Extensional Resolution

Extensional HO Resolution $\mathcal{ER}$

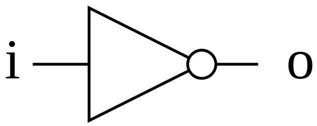
- [Andrews-71] Higher-order resolution (without unification)

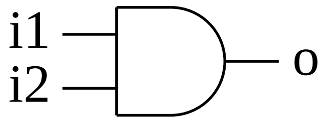


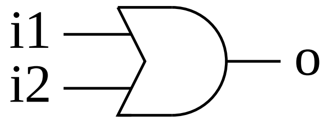
- [Huet-73/75] Higher-order constrained resolution

- [Benzmüller-99] Extensional higher-order resolution

- Some Basic Devices







$\mathsf{NOT}(i, o) =$ $(o = \neg i)$	$\mathsf{AND}(i_1, i_2, o) =$ $(o = (i_1 \wedge i_2))$	$\mathsf{OR}(i_1, i_2, o) =$ $(o = (i_1 \vee i_2))$
$\mathsf{NOT}'(i, o) =$ $(\forall t. o(t) = \neg i(t))$	$\mathsf{AND}'(i_1, i_2, o) =$ $(\forall t. o(t) = (i_1(t) \wedge i_2(t)))$	$\mathsf{OR}'(i_1, i_2, o) =$ $(\forall t. o(t) = (i_1(t) \vee i_2(t)))$

HOL Application: Hardware Verification

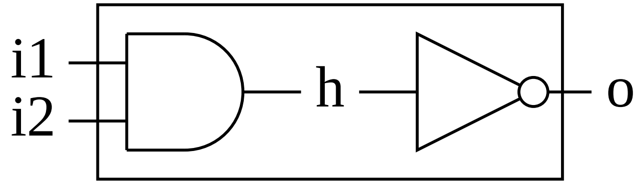
- Specification of NAND Device

$$\mathsf{NAND{-}SPEC}(i_1, i_2, o) =$$
$$(o = \neg(i_1 \wedge i_2))$$

$$\mathsf{NAND{-}SPEC}'(i_1, i_2, o) =$$
$$(\forall t. o(t) = \neg(i_1(t) \wedge i_2(t)))$$

HOL Application: Hardware Verification

- Implementation of NAND Device



$$\mathsf{NAND{-}IMP}(i_1, i_2, o) =$$
$$\exists h_o. \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o)$$

$$\mathsf{NAND{-}IMP}'(i_1, i_2, o) =$$
$$\exists h_{\iota \to o}. \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o)$$

HOL Application: Hardware Verification

- Implementation is correct

$$\mathsf{NAND{-}IMP}(\mathsf{i}_1, \mathsf{i}_2, \mathsf{o}) \Rightarrow \mathsf{NAND{-}SPEC}(\mathsf{i}_1, \mathsf{i}_2, \mathsf{o})$$
$$\mathsf{NAND{-}IMP}'(\mathsf{i}_1, \mathsf{i}_2, \mathsf{o}) \Rightarrow \mathsf{NAND{-}SPEC}'(\mathsf{i}_1, \mathsf{i}_2, \mathsf{o})$$

HOL Application: Hardware Verification

- Implementation is correct

$$\mathsf{NAND{-}IMP}(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}(i_1, i_2, o)$$
$$\mathsf{NAND{-}IMP}'(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}'(i_1, i_2, o)$$

- Definition expansion

HOL Application: Hardware Verification

- Implementation is correct

$$\mathsf{NAND{-}IMP}(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}(i_1, i_2, o)$$
$$\mathsf{NAND{-}IMP}'(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}'(i_1, i_2, o)$$

- Definition expansion

$$(o = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_o . \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o))$$

HOL Application: Hardware Verification

- Implementation is correct

$$\mathsf{NAND{-}IMP}(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}(i_1, i_2, o)$$

$$\mathsf{NAND{-}IMP}'(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}'(i_1, i_2, o)$$

- Definition expansion

$$(o = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_o . \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o))$$

$$(o = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_o . (h = (i_1 \wedge i_2)) \wedge (o = \neg h))$$

HOL Application: Hardware Verification

- Implementation is correct

$$\mathsf{NAND{-}IMP}(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}(i_1, i_2, o)$$
$$\mathsf{NAND{-}IMP}'(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}'(i_1, i_2, o)$$

- Definition expansion

$$(o = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_o\blacksquare \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o))$$
$$(o = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_o\blacksquare (h = (i_1 \wedge i_2)) \wedge (o = \neg h))$$

$$(out = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_{\iota \to o}\blacksquare \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o))$$

HOL Application: Hardware Verification

- Implementation is correct

$$\mathsf{NAND{-}IMP}(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}(i_1, i_2, o)$$
$$\mathsf{NAND{-}IMP}'(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}'(i_1, i_2, o)$$

- Definition expansion

$$(o = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_o\centerdot \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o))$$
$$(o = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_o\centerdot (h = (i_1 \wedge i_2)) \wedge (o = \neg h))$$

$$(out = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_{\iota \rightarrow o}\centerdot \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o))$$
$$(out = \neg(i_1 \wedge i_2)) \Rightarrow$$
$$(\exists h_{\iota \rightarrow o}\centerdot (\forall t_i\centerdot (h(t) = (i_1(t) \wedge i_2(t)))) \wedge (\forall t_i\centerdot (o(t) = \neg h(t))))$$

HOL Application: Hardware Verification

- Implementation is correct

$$\mathsf{NAND{-}IMP}(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}(i_1, i_2, o)$$
$$\mathsf{NAND{-}IMP}'(i_1, i_2, o) \Rightarrow \mathsf{NAND{-}SPEC}'(i_1, i_2, o)$$

- Definition expansion

$$(o = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_o. \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o))$$
$$(o = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_o. (h = (i_1 \wedge i_2)) \wedge (o = \neg h))$$

$$(out = \neg(i_1 \wedge i_2)) \Rightarrow (\exists h_{\iota \to o}. \mathsf{AND}(i_1, i_2, h) \wedge \mathsf{NOT}(h, o))$$
$$(out = \neg(i_1 \wedge i_2)) \Rightarrow$$
$$(\exists h_{\iota \to o}. (\forall t_i. (h(t) = (i_1(t) \wedge i_2(t)))) \wedge (\forall t_i. (o(t) = \neg h(t))))$$

- LEO-II's proofs: approx. 240ms