

Automated Theorem Proving in Higher-Order Logics

Christoph E. Benzmüller

Special thanks to: Chad E. Brown



<http://www.ags.uni-sb.de/~chris/>

ATPHOL'06

SS06, Lecture Course at TU Darmstadt, Germany



UNIVERSITÄT
DES
SAARLANDES

ATPHOL'06 – p.1

©Benzmüller, 2006



Introduction Meeting

Outline for Today



- Notion of higher-order logic and focus of this lecture

Outline for Today

ATPHOL'06-[1] – p.2



- Notion of higher-order logic and focus of this lecture
- 'Who am I?' and 'Who are You?'



UNIVERSITÄT
DES
SAARLANDES

ATPHOL'06-[1] – p.3

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL'06-[1] – p.3

Outline for Today



- Notion of higher-order logic and focus of this lecture
- 'Who am I?' and 'Who are You?'
- Organizational issues

Outline for Today



- Notion of higher-order logic and focus of this lecture
- 'Who am I?' and 'Who are You?'
- Organizational issues
- Overview on my current research projects (if time permits)

©Benzmüller, 2006



ATPHOL06-[1] – p.3

Outline for Today



- Notion of higher-order logic and focus of this lecture
- 'Who am I?' and 'Who are You?'
- Organizational issues
- Overview on my current research projects (if time permits)
- (Actual lecture starts on Monday!)

Notion of Higher-Order Logic



We are interested in expressive logic language(s):

- quantification over functions and predicate variables

©Benzmüller, 2006



ATPHOL06-[1] – p.3

©Benzmüller, 2006



ATPHOL06-[1] – p.4

We are interested in expressive logic language(s):

- quantification over functions and predicate variables

► $\forall x_{\text{nat}} \forall y_{\text{nat}}. x + y = y + x$ versus

$\exists o_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}}. x \circ y = y \circ x$

We are interested in expressive logic language(s):

- quantification over functions and predicate variables

► $\forall x_{\text{nat}} \forall y_{\text{nat}}. x + y = y + x$ versus

$\exists o_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}}. x \circ y = y \circ x$

► $\lambda x_{\text{nat}}. x \in A \wedge x \in B$ is a witness for $\exists p_{\text{nat} \rightarrow o}. A \cap B \subseteq p$

We are interested in expressive logic language(s):

- quantification over functions and predicate variables

► $\forall x_{\text{nat}} \forall y_{\text{nat}}. x + y = y + x$ versus

$\exists o_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}}. x \circ y = y \circ x$

► $\lambda x_{\text{nat}}. x \in A \wedge x \in B$ is a witness for $\exists p_{\text{nat} \rightarrow o}. A \cap B \subseteq p$

- λ -terms to denote/represent (unnamed) functions, predicates and sets

We are interested in expressive logic language(s):

- quantification over functions and predicate variables

► $\forall x_{\text{nat}} \forall y_{\text{nat}}. x + y = y + x$ versus

$\exists o_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}}. x \circ y = y \circ x$

► $\lambda x_{\text{nat}}. x \in A \wedge x \in B$ is a witness for $\exists p_{\text{nat} \rightarrow o}. A \cap B \subseteq p$

- λ -terms to denote/represent (unnamed) functions, predicates and sets

► a function f on naturals with $x \rightarrow x + 1$ can be represented by the unnamed λ -term $\lambda x_{\text{nat}}. x + 1$

We are interested in expressive logic language(s):

- quantification over functions and predicate variables
 - ▶ $\forall x_{\text{nat}} \forall y_{\text{nat}}. x + y = y + x$ versus
 $\exists o_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}}. x \circ y = y \circ x$
 - ▶ $\lambda x_{\text{nat}}. x \in A \wedge x \in B$ is a witness for $\exists p_{\text{nat} \rightarrow o}. A \cap B \subseteq p$
- λ -terms to denote/represent (unnamed) functions, predicates and sets
 - ▶ a function f on naturals with $x \rightarrow x + 1$ can be represented by the unnamed λ -term $\lambda x_{\text{nat}}. x + 1$
 - ▶ the set of all even naturals can be represented by the unnamed λ -term $\lambda x_{\text{nat}}. \text{even } x$

■ ...

- ...
- predicates and functions over other predicates and functions

- ...
- predicates and functions over other predicates and functions
 - ▶ $\exists o_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}} \forall z_{\text{nat}}. \text{commutative}(o) \wedge (x \circ y) \circ z = x \circ (y \circ z)$

Notion of Higher-Order Logic



- ...
- predicates and functions over other predicates and functions
 - ▶ $\exists \circ_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}} \forall z_{\text{nat}}. \text{commutative}(\circ) \wedge (x \circ y) \circ z = x \circ (y \circ z)$
- types to avoid logical paradoxes and inconsistencies

Notion of Higher-Order Logic



- ...
- predicates and functions over other predicates and functions
 - ▶ $\exists \circ_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}} \forall z_{\text{nat}}. \text{commutative}(\circ) \wedge (x \circ y) \circ z = x \circ (y \circ z)$
- types to avoid logical paradoxes and inconsistencies
 - ▶ “the set of all sets that do not contain themselves”
 - ▶ our choice is: Alonzo Church’s Simple Type Theory / Classical Higher-Order Logic

Notion of Higher-Order Logic



- ...
- predicates and functions over other predicates and functions
 - ▶ $\exists \circ_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}} \forall z_{\text{nat}}. \text{commutative}(\circ) \wedge (x \circ y) \circ z = x \circ (y \circ z)$
- types to avoid logical paradoxes and inconsistencies
 - ▶ “the set of all sets that do not contain themselves”

Notion of Higher-Order Logic



- ...
- predicates and functions over other predicates and functions
 - ▶ $\exists \circ_{\text{nat} \times \text{nat} \rightarrow \text{nat}} \forall x_{\text{nat}} \forall y_{\text{nat}} \forall z_{\text{nat}}. \text{commutative}(\circ) \wedge (x \circ y) \circ z = x \circ (y \circ z)$
- types to avoid logical paradoxes and inconsistencies
 - ▶ “the set of all sets that do not contain themselves”
 - ▶ our choice is: Alonzo Church’s Simple Type Theory / Classical Higher-Order Logic
 - ▶ we will **not** study rich type systems supporting polymorphism or dependent types

Focus of the Lecture



Focus of the Lecture



$$\exists p. p \in A \cap B$$



$$p = \{x \mid x \text{ in } A \text{ and } x \text{ in } B\}$$

Focus of the Lecture



Focus of the Lecture



- Syntax: Simply typed λ -calculus and Church's Simple Type Theory
- Semantics: applicative structures, different model classes (including Henkin semantics), extensionality and equality

- Syntax: Simply typed λ -calculus and Church's Simple Type Theory
- Semantics: applicative structures, different model classes (including Henkin semantics), extensionality and equality
- Abstract consistency, Hintikka sets, and model existence

Focus of the Lecture



Focus of the Lecture



- Syntax: Simply typed λ -calculus and Church's Simple Type Theory
- Semantics: applicative structures, different model classes (including Henkin semantics), extensionality and equality
- Abstract consistency, Hintikka sets, and model existence
- Calculi: natural deduction, sequent calculus, unification, extensional resolution, soundness and completeness, cut-elimination and cut-simulation

- Syntax: Simply typed λ -calculus and Church's Simple Type Theory
- Semantics: applicative structures, different model classes (including Henkin semantics), extensionality and equality
- Abstract consistency, Hintikka sets, and model existence
- Calculi: natural deduction, sequent calculus, unification, extensional resolution, soundness and completeness, cut-elimination and cut-simulation
- The theorem prover LEO

Relevance and Applications



Applications of Higher-Order Theorem Proving

- Formal Methods

Relevance and Applications



Applications of Higher-Order Theorem Proving

- Formal Methods
- Mathematics

Applications of Higher-Order Theorem Proving

- Formal Methods
- Mathematics
- Functional and Logical Programming

Applications of Higher-Order Theorem Proving

- Formal Methods
- Mathematics
- Functional and Logical Programming
- AI

Applications of Higher-Order Theorem Proving

- Formal Methods
- Mathematics
- Functional and Logical Programming
- AI
 - ▶ CYC
 - ▶ ...

Applications of Higher-Order Theorem Proving

- Formal Methods
- Mathematics
- Functional and Logical Programming
- AI
 - ▶ CYC
 - ▶ ...

Applications of Higher-Order Theorem Proving

- Formal Methods
- Mathematics
- Functional and Logical Programming
- AI
 - ▶ CYC
 - ▶ ...
- Computational Linguistics

Applications of Higher-Order Theorem Proving

- Formal Methods
- Mathematics
- Functional and Logical Programming
- AI
 - ▶ CYC
 - ▶ ...
- Computational Linguistics
- ...



'Who am I?' and 'Who are You?'

Who am I?



Who am I? _____



Who am I? _____



- Grown-up in Saarburg and Trier

- Grown-up in Saarburg and Trier

- schools in Saarburg and Trier
- freelancer of daily newspaper in Trier



©Benzmüller, 2006



ATPHOL'06-[1] – p.11

©Benzmüller, 2006



ATPHOL'06-[1] – p.11

Who am I? _____



Who am I? _____



- Grown-up in Saarburg and Trier
 - schools in Saarburg and Trier
 - freelancer of daily newspaper in Trier
- Sport: Middle- and Long-distance Running



- Grown-up in Saarburg and Trier
 - schools in Saarburg and Trier
 - freelancer of daily newspaper in Trier
- Sport: Middle- and Long-distance Running
 - German champion
 - Olympic Center (OSP) Saarbrücken



©Benzmüller, 2006



ATPHOL'06-[1] – p.11

©Benzmüller, 2006



ATPHOL'06-[1] – p.11

Who am I? _____



Who am I? _____



- Grown-up in Saarburg and Trier

- schools in Saarburg and Trier
- freelancer of daily newspaper in Trier



- Sport: Middle- and Long-distance Running

- German champion
- Olympic Center (OSP) Saarbrücken



- Computer Science Study in Saarbrücken (89-95):

- Grown-up in Saarburg and Trier

- schools in Saarburg and Trier
- freelancer of daily newspaper in Trier



- Sport: Middle- and Long-distance Running

- German champion
- Olympic Center (OSP) Saarbrücken



- Computer Science Study in Saarbrücken (89-95):

- aside of professional training at OSP
- 'caught' by Jörg Siekmann's AI lecture
- focus: formal methods, algebraic specification, automated reasoning



©Benzmüller, 2006



ATPHOL'06-[1] – p.11



ATPHOL'06-[1] – p.11

Who am I? _____



Who am I? _____



- PhD at Saarbrücken and CMU, Pittsburgh (95-99)

- PhD at Saarbrücken and CMU, Pittsburgh (95-99)

- mechanization/automation of higher-order logic
- higher-order theorem prover LEO
- semantics of higher-order logic
- Stipend of the Deutsche Studienstiftung
- J. Siekmann, M. Kohlhase, F. Pfenning, P. Andrews



©Benzmüller, 2006



ATPHOL'06-[1] – p.12

©Benzmüller, 2006



ATPHOL'06-[1] – p.12

Who am I? _____



Who am I? _____



- PhD at Saarbrücken and CMU, Pittsburgh (95-99)

- mechanization/automation of higher-order logic
- higher-order theorem prover LEO
- semantics of higher-order logic
- Stipend of the Deutsche Studienstiftung
- J. Siekmann, M. Kohlhase, F. Pfenning, P. Andrews



- PostDoc in UK (1999/2000)

- PhD at Saarbrücken and CMU, Pittsburgh (95-99)

- mechanization/automation of higher-order logic
- higher-order theorem prover LEO
- semantics of higher-order logic
- Stipend of the Deutsche Studienstiftung
- J. Siekmann, M. Kohlhase, F. Pfenning, P. Andrews



- PostDoc in UK (1999/2000)

- Univ. of Birmingham and Univ. of Edinburgh
- EPSRC grant
- focus on agent-oriented theorem proving



©Benzmüller, 2006



ATPHOL'06-[1] – p.12

Who am I? _____



Who am I? _____



- Head of OMEGA group / Dozent at Saarland University (since 2001)



- Head of OMEGA group / Dozent at Saarland University (since 2001)
 - focus on assistance systems for formal methods and mathematics
 - group has approx. 10 full time researchers
 - principal investigator of 2 projects in the SFB 378: Resource-adaptive Cognitive Processes

©Benzmüller, 2006



ATPHOL'06-[1] – p.13

©Benzmüller, 2006



ATPHOL'06-[1] – p.13

Who am I?



- Head of OMEGA group / Dozent at Saarland University (since 2001)
 - focus on assistance systems for formal methods and mathematics
 - group has approx. 10 full time researchers
 - principal investigator of 2 projects in the SFB 378: Resource-adaptive Cognitive Processes
- SFB Project OMEGA:



©Benzmüller, 2006



ATPHOL'06-[1] – p.13

Who am I?



- Head of OMEGA group / Dozent at Saarland University (since 2001)
 - focus on assistance systems for formal methods and mathematics
 - group has approx. 10 full time researchers
 - principal investigator of 2 projects in the SFB 378: Resource-adaptive Cognitive Processes
- SFB Project OMEGA:
 - integrated environment of maths reasoning tools
 - application of artificial intelligence techniques
- SFB Project DIALOG:
 - natural tutorial dialog: student ↔ maths assistance system
 - collaboration with Computational Linguistics



©Benzmüller, 2006



ATPHOL'06-[1] – p.13

Who am I?



- Head of OMEGA group / Dozent at Saarland University (since 2001)
 - focus on assistance systems for formal methods and mathematics
 - group has approx. 10 full time researchers
 - principal investigator of 2 projects in the SFB 378: Resource-adaptive Cognitive Processes
- SFB Project OMEGA:



- integrated environment of maths reasoning tools
- application of artificial intelligence techniques



Who am I?



- Head of OMEGA group / Dozent at Saarland University (since 2001)
 - focus on assistance systems for formal methods and mathematics
 - group has approx. 10 full time researchers
 - principal investigator of 2 projects in the SFB 378: Resource-adaptive Cognitive Processes
- SFB Project OMEGA:



- integrated environment of maths reasoning tools
- application of artificial intelligence techniques



- SFB Project DIALOG:

- natural tutorial dialog: student ↔ maths assistance system
- collaboration with Computational Linguistics



©Benzmüller, 2006



ATPHOL'06-[1] – p.13



Organizational Issues

Before we start ...

We need to discuss and fix:

- Miscellaneous

Before we start ...

We need to discuss and fix:

- Miscellaneous
- Lectures

Before we start ...



We need to discuss and fix:

- Miscellaneous
- Lectures
- Exercises and Tutorials

©Benzmüller, 2006



ATPHOL'06-[1] – p.16

Miscellaneous



- Lecture Webpage:

<http://www.ag.ssb.uni-saarland.de/~chris/ATPHOL-06/>

This webpage will contain important information such as:
dates, announcements, link to slides and exercises, literature,
etc.

Before we start ...



We need to discuss and fix:

- Miscellaneous
- Lectures
- Exercises and Tutorials
- Examination

©Benzmüller, 2006



ATPHOL'06-[1] – p.16

Miscellaneous



- Lecture Webpage:

<http://www.ag.ssb.uni-saarland.de/~chris/ATPHOL-06/>

This webpage will contain important information such as:
dates, announcements, link to slides and exercises, literature,
etc.

- I will handout a registration questionnaire: Please write your name, immatriculation number, semester, etc.

©Benzmüller, 2006



ATPHOL'06-[1] – p.17

©Benzmüller, 2006



ATPHOL'06-[1] – p.17

Miscellaneous



Miscellaneous



- Lecture Webpage:
<http://www.ag.ssb.de/chris/ATPHOL-06/>
This webpage will contain important information such as:
dates, announcements, link to slides and exercises, literature,
etc.
- I will handout a registration questionnaire: Please write your
name, immatriculation number, semester, etc.
- Except for the lecturing dates and times, I will be in
Saarbrücken. To discuss problems etc., please write an e-mail

©Benzmüller, 2006



ATPHOL'06-[1] – p.17

Miscellaneous



Lectures



- Lecture Webpage:
<http://www.ag.ssb.de/chris/ATPHOL-06/>
This webpage will contain important information such as:
dates, announcements, link to slides and exercises, literature,
etc.
- I will handout a registration questionnaire: Please write your
name, immatriculation number, semester, etc.
- Except for the lecturing dates and times, I will be in
Saarbrücken. To discuss problems etc., please write an e-mail
- Organizational support: Markus Aderhold and Prof. Christoph
Walther
- Please note: I have no team that supports this lecture and I am
preparing the lectures in my spare time.

©Benzmüller, 2006



ATPHOL'06-[1] – p.17

©Benzmüller, 2006



ATPHOL'06-[1] – p.18

Lectures



Lectures



- Discussion of the exact start end end time of the lectures; do we want a break?
- Difficulty: I want to prerequisite that you are familiar with propositional and first order logic syntax and semantics (and probably mechanization). If this is not the case please let's discuss this issue now.

- Discussion of the exact start end end time of the lectures; do we want a break?
- Difficulty: I want to prerequisite that you are familiar with propositional and first order logic syntax and semantics (and probably mechanization). If this is not the case please let's discuss this issue now.
- Lectures will be a mixture between beamer presentation and traditional blackboard lecture.

©Benzmüller, 2006



ATPHOL'06-[1] – p.18

©Benzmüller, 2006



ATPHOL'06-[1] – p.18

Lectures



Exercises and Tutorials



- Discussion of the exact start end end time of the lectures; do we want a break?
- Difficulty: I want to prerequisite that you are familiar with propositional and first order logic syntax and semantics (and probably mechanization). If this is not the case please let's discuss this issue now.
- Lectures will be a mixture between beamer presentation and traditional blackboard lecture.

- Theoretical exercises will be announced on the slides.

©Benzmüller, 2006



ATPHOL'06-[1] – p.18

©Benzmüller, 2006



ATPHOL'06-[1] – p.19

- Theoretical exercises will be announced on the slides.
- They will be discussed in the tutorials; the tutorials will be held by myself.

- Theoretical exercises will be announced on the slides.
- They will be discussed in the tutorials; the tutorials will be held by myself.
- The exercises will not be corrected/graded, however, I expect that every student regularly presents a solution of the theoretical exercises on the blackboard; these regular presentations are required for entering the final examination.

Exercises and Tutorials

- Theoretical exercises will be announced on the slides.
- They will be discussed in the tutorials; the tutorials will be held by myself.
- The exercises will not be corrected/graded, however, I expect that every student regularly presents a solution of the theoretical exercises on the blackboard; these regular presentations are required for entering the final examination.
- I am also planning a practical exercise parallel to the lecture: we will form teams of students and each team will implement an algorithm related to the lecture (most likely higher-order (pre-)unification); at the end of the lecture these implementations will be evaluated in a competition.

Examination

- Written or oral exam as part of a masters study or as part of a diploma exam in a diploma study (e.g. with Prof. Walther)

Examination



Examination



- Written or oral exam as part of a masters study or as part of a diploma exam in a diploma study (e.g. with Prof. Walther)
- The entry requirements for the final exam are

- Written or oral exam as part of a masters study or as part of a diploma exam in a diploma study (e.g. with Prof. Walther)
- The entry requirements for the final exam are
 - ▶ regular presentations in tutorials

Examination



Examination



- Written or oral exam as part of a masters study or as part of a diploma exam in a diploma study (e.g. with Prof. Walther)
- The entry requirements for the final exam are
 - ▶ regular presentations in tutorials
 - ▶ passed evaluation of the practical exercise

- Written or oral exam as part of a masters study or as part of a diploma exam in a diploma study (e.g. with Prof. Walther)
- The entry requirements for the final exam are
 - ▶ regular presentations in tutorials
 - ▶ passed evaluation of the practical exercise
- Do we have to fix dates for the evaluation of the practical exercise evaluation/competition and for the final exam?

Examination



Examination



- Written or oral exam as part of a masters study or as part of a diploma exam in a diploma study (e.g. with Prof. Walther)
- The entry requirements for the final exam are
 - ▶ regular presentations in tutorials
 - ▶ passed evaluation of the practical exercise
- Do we have to fix dates for the evaluation of the practical exercise evaluation/competition and for the final exam?
- Any questions?

©Benzmüller, 2006



ATPHOL'06-[1] – p.20

©Benzmüller, 2006



ATPHOL'06-[1] – p.20



Overview on my current
research projects (→ other
slides)

©Benzmüller, 2006



ATPHOL'06-[1] – p.21

©Benzmüller, 2006



ATPHOL'06-[2] – p.22



Some Historical Remarks

- Ancient Greek's (mostly Aristotle): laws of human thought; theory of well-chosen axioms and rules (syllogism)

Examples:

- Modus Ponens:

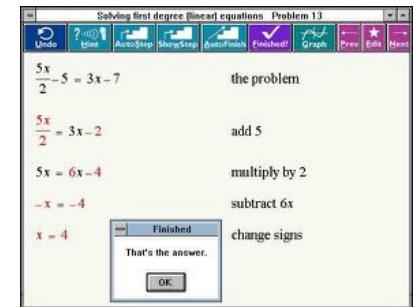
$$\frac{A \Rightarrow B \quad A}{B} \text{ mp}$$

- Modus Tollens:

$$\frac{A \Rightarrow B \quad \neg B}{\neg A} \text{ mt}$$



W. Schickard's mechanical calculator
(1592 - 1635)



MathPert system (Michael Beeson)

History (Cont'd)

- Gottfried Wilhelm Leibniz (1646-1716)

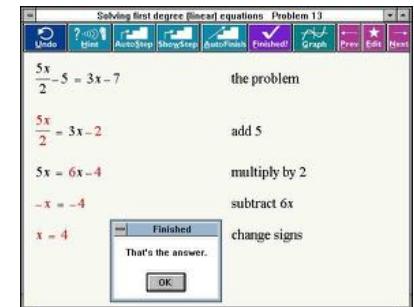
- Dream of formalizing and mechanizing mathematical reasoning (lingua characteristica and calculus ratiocinator)
- Mechanization of simple arithmetical operations, e.g., mechanical calculator capable of multiplication

History (Cont'd)

- Computer algebra systems
 - Roots: Abacus (approx. 500 a.D.)
 - mechanical calculators built from 15. century on
 - modern computer algebra systems are today widely used



W. Schickard's mechanical calculator
(1592 - 1635)



MathPert system (Michael Beeson)

History (Cont'd)

History (Cont'd)

- It took until the end of the 19. century that *modern mathematical logic* was (re-)born:

- George Boole (1815-1864),
- Gottlob Frege (1848-1925),
- Bertrand Russel (1872-1970),
- and many others

stimulated the new and deep interest of many researchers in the field of mathematical logic.

History (Cont'd)



History (Cont'd)



- Frege's *Begriffsschrift* is described by Davis [Davis83] *not only as the direct ancestor of contemporary systems of mathematical logic but also as the ancestor of all formal languages, including computer programming languages.*
- First-order logic:

$$\forall x, y, z. (x + (y + z)) = ((x + y) + z)$$

History (Cont'd)



- In the early 30's results came fast:
 - Kurt Gödel, Jacques Herbrand and Thoralf Skolem proved the *completeness of the (first-order) predicate calculus* in 1930 [Goedel30,Herbrand30,Skolem28]: every valid formula in the language of the predicate calculus is derivable from its axioms.
 - However, Gödel showed in his famous *incompleteness theorems* [Goedel31] that it is impossible to develop a generally complete calculus that mechanizes mathematical reasoning. More precisely, Gödel showed that *as soon as a system is rich enough to encode Peano arithmetic, one can construct sentences that are valid in Peano arithmetic but which are not derivable in the system itself.*

- Hilbert's program at the very beginning of this century [Hilbert04,Hilbert27] aimed at the complete development of modern mathematics in a formal system.

History (Cont'd)



- Development of electronic computers in the 40's and 50's: disappointment gradually gave away to attempts of developing and implementing proof procedures in practice.
- J.A. Robinson: achieved important break-through (in first-order theorem proving) with his resolution approach in 1965 [Robinson65]. The most important improvement of this approach compared to former ones is that in order to prove a theorem it tries to *refute the negated theorem in a goal directed way*, thereby *employing first-order unification as a powerful filter* instead of simply enumerating the Herbrand universe like most earlier methods.
- Remark: *This lecture (amongst others) investigates the problems of applying the resolution idea in higher-order logic.*

©Benzmüller, 2006



ATPHOL'06-[2] – p.31

History HOL



- Higher-order logic: any simply typed logical system that allows quantification over function and predicate variables.
- It was Bertrand Russel [Russell02, Russell03] who first pointed out in 1902 that in connection with the comprehension principles (these principles assure the existence of certain functions) this may allow for *paradoxes*: most prominent example is the *set of all non-self-containing sets*.
- As a possible solution Russel suggested a few years later in [Russell08] a theory of types as a basis for the formalization of mathematics that differentiates between objects and sets (or functions) consisting of these kinds of objects.

©Benzmüller, 2006



ATPHOL'06-[2] – p.33

History (Cont'd)



- Robinson's ideas are still employed in many state of the art theorem provers such as OTTER, EQP (which solved the Robbins Problem), or the superposition based prover SPASS.
- Even tableaux based provers like PROTEIN are rather closely related to the resolution approach and unification became an essential (filtering) tool for the whole field.

©Benzmüller, 2006



ATPHOL'06-[2] – p.34

History HOL (Cont'd)



History HOL (Cont'd)

- Idea was also taken up by Alonzo Church in 1940, who invented the *simply typed λ -calculus* [Church40] in order to prevent such paradoxes in the untyped λ -calculus, which he developed with Schönfinkel and Curry ten years earlier.
- Typed and untyped λ -calculi play an important or even central role in many research fields of modern computer science.
- The avoidance of paradoxes like Russel's paradox is also a main reason why we employ a logic based on Church's simply typed λ -calculus [Church40] — i.e., *classical type theory / classical higher-order logic* — in this lecture.

History HOL (Cont'd)



- Relatively few researchers concentrated on the mechanization of higher-order logic.
- Robinson presents in [Robinson68, Robinson69] a higher-order proof procedure based on the tableaux idea that itself employs many ideas from the calculi given in [Schutte60] and [Takeuti53].
- The most important works to be mentioned are Peter Andrews' investigation of higher-order resolution [Andrews71], Jensen and Pietrowski's approach [JePi72] and especially Gerard Huet's constrained resolution approach [Huet72, Huet73].

History (Cont'd)



- Big challenge for the mechanization of higher-order logic is the undecidability of higher-order unification [Lucchesi72, Huet73, Goldfarb81]
- Andrews' resolution approach still avoids unification (and instead employs an enumeration of the universe)
- Huet's constrained resolution approach [Huet72] solves the problem by encoding the particular unification problems as unification constraints and by delaying the application of higher-order unification until the end of a refutation.
- Huet's approach additionally gains from the higher-order pre-unification algorithm [Huet75] which avoids the guessing aspects of full higher-order unification



Introduction

λ -Calculus: Motivation



Consider the following arithmetical computations

$$(-1)^2 - 1 = 0$$

$$(1)^2 - 1 = 0$$

$$(2)^2 - 1 = 3$$

...

Consider the following arithmetical computations

$$(-1)^2 - 1 = 0$$

$$(1)^2 - 1 = 0$$

$$(2)^2 - 1 = 3$$

...

A more general arithmetic expression for the LHS:

$$x^2 - 1$$

Consider the 0's (Nullstellen) of this function; we can express the existence of two 0's in first-order logic as follows

$$\exists n, m. n^2 - 1 = 0 \wedge m^2 - 1 = 0 \wedge n \neq m$$

Consider the 0's (Nullstellen) of this function; we can express the existence of two 0's in first-order logic as follows

$$\exists n, m. n^2 - 1 = 0 \wedge m^2 - 1 = 0 \wedge n \neq m$$

Now we may want to talk about the existence of a function f with two 0's:

$$(1) \exists f. \exists n, m. f(n) = 0 \wedge f(m) = 0 \wedge n \neq m$$

Consider the 0's (Nullstellen) of this function; we can express the existence of two 0's in first-order logic as follows

$$\exists n, m. n^2 - 1 = 0 \wedge m^2 - 1 = 0 \wedge n \neq m$$

Now we may want to talk about the existence of a function f with two 0's:

$$(1) \quad \exists f. \exists n, m. f(n) = 0 \wedge f(m) = 0 \wedge n \neq m$$

This expression is not a first-order statement; however we want to be able to express such statements. We also want to prove such statements and in a constructive proof we would like to provide witnesses for f and n, m .

In first-order logic we can describe f by the following equation

$$f(x) = x^2 - 1$$

UNIVERSITÄT
DES
SAARLANDES

©Benzmüller, 2006

ATPHOL'06-[2] – p.39

λ -Calculus: Set of λ -expressions

Given a countably infinite set of identifiers, say $a, b, c, \dots, x, y, z, x_1, x_2, \dots$. The set of all λ -expressions can then be described by the following context-free grammar in BNF:

1. $\langle \text{expr} \rangle ::= \langle \text{identifier} \rangle$

In λ -calculus the specified function f can be described (without giving it a name) by the witnessing λ -term

$$[\lambda x. x^2 - 1]$$

and the witnesses for n and m are -1 and 1 .



©Benzmüller, 2006

ATPHOL'06-[2] – p.40

λ -Calculus: Set of λ -expressions

Given a countably infinite set of identifiers, say $a, b, c, \dots, x, y, z, x_1, x_2, \dots$. The set of all λ -expressions can then be described by the following context-free grammar in BNF:

1. $\langle \text{expr} \rangle ::= \langle \text{identifier} \rangle$

2. $\langle \text{expr} \rangle ::= [\lambda \langle \text{identifier} \rangle . \langle \text{expr} \rangle]$

abstraction

Given a countably infinite set of identifiers, say

a, b, c, ..., x, y, z, x1, x2, The set of all λ -expressions can then be described by the following context-free grammar in BNF:

1. $\langle \text{expr} \rangle ::= \langle \text{identifier} \rangle$
2. $\langle \text{expr} \rangle ::= [\lambda \langle \text{identifier} \rangle . \langle \text{expr} \rangle]$ abstraction
3. $\langle \text{expr} \rangle ::= [\langle \text{expr} \rangle \langle \text{expr} \rangle]$ application

λ -Calculus: Conventions

We often omit brackets with the following conventions:

- [F A B] means [[F A] B]. (Application associates to the left.)
- [$\lambda x.\lambda y. B$] means [$\lambda x.[\lambda y. B]$].

λ -Calculus: Conventions

We often omit brackets with the following conventions:

- [F A B] means [[F A] B]. (Application associates to the left.)

λ -Calculus: Conventions

We often omit brackets with the following conventions:

- [F A B] means [[F A] B]. (Application associates to the left.)
- [$\lambda x.\lambda y. B$] means [$\lambda x.[\lambda y. B]$].
- A dot (except possibly after λ <identifier>) stands for a left bracket whose mate is as far to the right as possible without changing the existing bracketing.

Consider now the instantiation of (1) with these witness terms

$$\exists f. \exists n, m. f(n) = 0 \wedge f(m) = 0 \wedge n \neq m$$

Consider now the instantiation of (1) with these witness terms

$$\exists f. \exists n, m. f(n) = 0 \wedge f(m) = 0 \wedge n \neq m$$

$$f \longrightarrow \exists n, m. [[\lambda x. x^2 - 1] n] = 0 \wedge [[\lambda x. x^2 - 1] m] = 0 \wedge n \neq m$$

Consider now the instantiation of (1) with these witness terms

$$\exists f. \exists n, m. f(n) = 0 \wedge f(m) = 0 \wedge n \neq m$$

$$f \longrightarrow \exists n, m. [[\lambda x. x^2 - 1] n] = 0 \wedge [[\lambda x. x^2 - 1] m] = 0 \wedge n \neq m$$

$$n, m \longrightarrow [[\lambda x. x^2 - 1] - 1] = 0 \wedge [[\lambda x. x^2 - 1] 1] = 0 \wedge -1 \neq 1$$

Consider now the instantiation of (1) with these witness terms

$$\exists f. \exists n, m. f(n) = 0 \wedge f(m) = 0 \wedge n \neq m$$

$$f \longrightarrow \exists n, m. [[\lambda x. x^2 - 1] n] = 0 \wedge [[\lambda x. x^2 - 1] m] = 0 \wedge n \neq m$$

$$n, m \longrightarrow [[\lambda x. x^2 - 1] - 1] = 0 \wedge [[\lambda x. x^2 - 1] 1] = 0 \wedge -1 \neq 1$$

Finally we can ‘evaluate’ function applications by so called β -reduction

$$((-1)^2 - 1) = 0 \wedge (1^2 - 1) = 0 \wedge -1 \neq 1$$

λ -Calculus: β -reduction



The β -reduction rule expresses the idea of function application as motivated on the previous slide. Formally it states that

$$[[\lambda x. A] B] \longrightarrow_{\beta} A[x/B]$$

if all free occurrences in B remain free in $A[x/B]$. Here, $A[x/B]$ means the expression E with every free occurrence of x in A replaced with B .

©Benzmüller, 2006



ATPHOL'06-[2] – p.44

λ -Calculus: α -conversion



The names of the bound variables are unimportant:

$$\lambda x. x^2 - 1 \text{ and } \lambda y. y^2 - 1$$

denote the same function.

©Benzmüller, 2006



ATPHOL'06-[2] – p.46

λ -Calculus: Currying



A function of two variables is expressed in lambda calculus as a function of one argument which returns a function of one argument. For instance, the function

$$f(x, y) = x^2 - y$$

is encoded as

$$[\lambda x. \lambda y. x^2 - y]$$

©Benzmüller, 2006



ATPHOL'06-[2] – p.45

λ -Calculus: α -conversion



The names of the bound variables are unimportant:

$$\lambda x. x^2 - 1 \text{ and } \lambda y. y^2 - 1$$

denote the same function.

Formally, the α -conversion rule states that if x and y are variables and A is a λ -expression then

$$[\lambda x. A] \longleftrightarrow_{\alpha} [\lambda y. A[x/y]]$$

if y does not appear freely in A and y is not bound by a λ in A whenever it replaces a x .

©Benzmüller, 2006



ATPHOL'06-[2] – p.46

η -reduction expresses the idea of (functional) extensionality, which in this context is that two functions are the same iff they give the same result for all arguments:

$$[\lambda x.Fx] \longrightarrow_{\eta} F$$

whenever x does not appear free in F .

- We define $\longrightarrow_{\alpha\beta\eta}^*$ as the smallest equivalence relation closed under the reduction rules \longrightarrow_{β} and \longrightarrow_{η} and α -conversion.
(Similarly we may define \longrightarrow_M^* for $M \subset \{\alpha, \beta, \eta\}$)

- We define $\longrightarrow_{\alpha\beta\eta}^*$ as the smallest equivalence relation closed under the reduction rules \longrightarrow_{β} and \longrightarrow_{η} and α -conversion.
(Similarly we may define \longrightarrow_M^* for $M \subset \{\alpha, \beta, \eta\}$)
- We call two λ -terms E and T $\alpha\beta\eta$ -equivalent (or short equivalent) if

$$E \xleftarrow{\longrightarrow_{\alpha\beta\eta}^*} T$$

- We define $\longrightarrow_{\alpha\beta\eta}^*$ as the smallest equivalence relation closed under the reduction rules \longrightarrow_{β} and \longrightarrow_{η} and α -conversion.
(Similarly we may define \longrightarrow_M^* for $M \subset \{\alpha, \beta, \eta\}$)
- We call two λ -terms E and T $\alpha\beta\eta$ -equivalent (or short equivalent) if

$$E \xleftarrow{\longrightarrow_{\alpha\beta\eta}^*} T$$

(Similarly we may define M -equivalence for $M \subset \{\alpha, \beta, \eta\}$)

- A λ -expression is called a β -normal form if it does not allow any β -reduction, i.e., has no subexpression of the form

$$[[\lambda x . A] B]$$

- A λ -expression is called a β -normal form if it does not allow any β -reduction, i.e., has no subexpression of the form

$$[[\lambda x . A] B]$$

- A λ -expression is called a β -normal form if it does not allow any β -reduction, i.e., has no subexpression of the form

$$[[\lambda x . A] B]$$

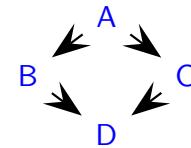
- Not every λ -expression is equivalent to a β -normal form (where $\beta \in \{\beta, \beta\eta\}$)

- A λ -expression is called a η -normal form if it does not allow any η -reduction, i.e., has no subexpression of the form (where x does not occur free in E)

$$[\lambda x . E x]$$

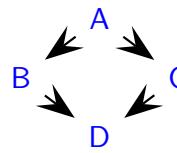
- A λ -expression is called a $\beta\eta$ -normal form if it satisfies both conditions.

- Not every λ -expression is equivalent to a β -normal form (where $\beta \in \{\beta, \beta\eta\}$)
- The Church-Rosser theorem(s) state that if $A \rightarrow^* B$ and $A \rightarrow^* C$, then there is some D such that $B \rightarrow^* D$ and $C \rightarrow^* D$.



- Not every λ -expression is equivalent to a β -normal form (where $\beta \in \{\beta, \beta\eta\}$)
- The Church-Rosser theorem(s) state that if $A \rightarrow^* B$ and $A \rightarrow^* C$, then there is some D such that $B \rightarrow^* D$ and $C \rightarrow^* D$.

- Not every λ -expression is equivalent to a β -normal form (where $\beta \in \{\beta, \beta\eta\}$)
- The Church-Rosser theorem(s) state that if $A \rightarrow^* B$ and $A \rightarrow^* C$, then there is some D such that $B \rightarrow^* D$ and $C \rightarrow^* D$.



- From Church-Rosser it follows that every term has at most one β -normal form (up to α -conversion).

Consider twofold iteration of function $f := [\lambda x.x^2 - 1]$

$$f(f(x)) = (x^2 - 1)^2 - 1 = x^4 - 2x^2$$

Consider twofold iteration of function $f := [\lambda x.x^2 - 1]$

$$f(f(x)) = (x^2 - 1)^2 - 1 = x^4 - 2x^2$$

The following λ -term expresses twofold iteration of a function

$$[\lambda g.\lambda y.g [g y]]$$

Consider twofold iteration of function $f := [\lambda x.x^2 - 1]$

$$f(f(x)) = (x^2 - 1)^2 - 1 = x^4 - 2x^2$$

The following λ -term expresses twofold iteration of a function

$$[\lambda g.\lambda y.g [g y]]$$

Let us apply this λ -term now to our function f

$$\begin{aligned} & [[\lambda g.\lambda y.g [g y]] [\lambda x.x^2 - 1]] \\ \longrightarrow_{\beta} & [\lambda y.[\lambda x.x^2 - 1][[\lambda x.x^2 - 1]y]] \end{aligned}$$

Consider twofold iteration of function $f := [\lambda x.x^2 - 1]$

$$f(f(x)) = (x^2 - 1)^2 - 1 = x^4 - 2x^2$$

The following λ -term expresses twofold iteration of a function

$$[\lambda g.\lambda y.g [g y]]$$

Let us apply this λ -term now to our function f

$$[[\lambda g.\lambda y.g [g y]] [\lambda x.x^2 - 1]]$$

Consider twofold iteration of function $f := [\lambda x.x^2 - 1]$

$$f(f(x)) = (x^2 - 1)^2 - 1 = x^4 - 2x^2$$

The following λ -term expresses twofold iteration of a function

$$[\lambda g.\lambda y.g [g y]]$$

Let us apply this λ -term now to our function f

$$\begin{aligned} & [[\lambda g.\lambda y.g [g y]] [\lambda x.x^2 - 1]] \\ \longrightarrow_{\beta} & [\lambda y.[\lambda x.x^2 - 1][[\lambda x.x^2 - 1]y]] \end{aligned}$$

Consider twofold iteration of function $f := [\lambda x.x^2 - 1]$

$$f(f(x)) = (x^2 - 1)^2 - 1 = x^4 - 2x^2$$

The following λ -term expresses twofold iteration of a function

$$[\lambda g.\lambda y.g [g y]]$$

Let us apply this λ -term now to our function f

$$\begin{aligned} & [[\lambda g.\lambda y.g [g y]] [\lambda x.x^2 - 1]] \\ \longrightarrow_{\beta} & [\lambda y.[\lambda x.x^2 - 1][[\lambda x.x^2 - 1]y]] \\ \longrightarrow_{\beta} & \lambda y.[\lambda x.x^2 - 1] [y^2 - 1] \end{aligned}$$

Consider twofold iteration of function $f := [\lambda x.x^2 - 1]$

$$f(f(x)) = (x^2 - 1)^2 - 1 = x^4 - 2x^2$$

The following λ -term expresses twofold iteration of a function

$$[\lambda g.\lambda y.g [g y]]$$

Let us apply this λ -term now to our function f

$$\begin{aligned} & [[\lambda g.\lambda y.g [g y]] [\lambda x.x^2 - 1]] \\ \longrightarrow_{\beta} & [\lambda y.[\lambda x.x^2 - 1][[\lambda x.x^2 - 1]y]] \\ \longrightarrow_{\beta} & \lambda y.[\lambda x.x^2 - 1] [y^2 - 1] \\ \longrightarrow_{\beta} & [\lambda y.[y^2 - 1]^2 - 1] = \lambda y.y^4 - 2y^2 \end{aligned}$$

λ -Calculus: Church Numerals

We employ iteration to define natural numbers as Church numerals:

$$\bar{0} = [\lambda f.\lambda x.x], \quad \bar{1} = [\lambda f.\lambda x.fx], \quad \bar{2} = [\lambda f.\lambda x.f(fx)], \quad \dots$$

Generally a natural number n is encoded as the Church numeral

$$\bar{n} = [\lambda f.\lambda y.f^n y]$$

where f^n is an abbreviation for $\underbrace{[f [f [f \dots [f}_n y]]]$.

We employ iteration to define natural numbers as Church numerals:

$$\bar{0} = [\lambda f.\lambda x.x], \quad \bar{1} = [\lambda f.\lambda x.fx], \quad \bar{2} = [\lambda f.\lambda x.f(fx)], \quad \dots$$

λ -Calculus: Church Numerals

We employ iteration to define natural numbers as Church numerals:

$$\bar{0} = [\lambda f.\lambda x.x], \quad \bar{1} = [\lambda f.\lambda x.fx], \quad \bar{2} = [\lambda f.\lambda x.f(fx)], \quad \dots$$

Generally a natural number n is encoded as the Church numeral

$$\bar{n} = [\lambda f.\lambda y.f^n y]$$

where f^n is an abbreviation for $\underbrace{[f [f [f \dots [f}_n y]]]$.

Intuitively, the number n in lambda calculus is a function that takes a function f as argument and returns the n -th iterate of f .

We can now define a successor function $\overline{\text{SUCC}}$, which takes a number \overline{n} and returns $\overline{n + 1}$:

$$\overline{\text{SUCC}} = [\lambda n. \lambda f. \lambda x. f[nfx]]$$

We can now define a successor function $\overline{\text{SUCC}}$, which takes a number \overline{n} and returns $\overline{n + 1}$:

$$\overline{\text{SUCC}} = [\lambda n. \lambda f. \lambda x. f[nfx]]$$

Addition is defined as follows:

$$\overline{\text{PLUS}} = [\lambda m. \lambda n. \lambda f. \lambda x. mf[nfx]]$$

We can now define a successor function $\overline{\text{SUCC}}$, which takes a number \overline{n} and returns $\overline{n + 1}$:

$$\overline{\text{SUCC}} = [\lambda n. \lambda f. \lambda x. f[nfx]]$$

Addition is defined as follows:

$$\overline{\text{PLUS}} = [\lambda m. \lambda n. \lambda f. \lambda x. mf[nfx]]$$

Multiplication can then be defined as

$$\overline{\text{MULT}} = \lambda m. \lambda n. m[\overline{\text{PLUS}}\ n]\overline{0},$$

the idea being that multiplying m and n is the same as adding n to 0 m times.

The predecessor function is more difficult:

$$\overline{\text{PRED}} = \lambda n. \lambda f. \lambda x. n[\lambda g. \lambda h. h[gf]] [\lambda u. x] [\lambda u. u]$$

λ -Calculus: Church Numerals



The predecessor function is more difficult:

$$\overline{\text{PRED}} = \lambda n. \lambda f. \lambda x. n[\lambda g. \lambda h. h [g f]] [\lambda u. x] [\lambda u. u]$$

or alternatively

$$\overline{\text{PRED}} = \lambda n. n[\lambda g. \lambda k. [g \overline{1}] [\lambda u. \overline{\text{PLUS}} [g k] \overline{1}] k] [\lambda l. \overline{0}] \overline{0}$$

λ -Calculus: Church Numerals



The predecessor function is more difficult:

$$\overline{\text{PRED}} = \lambda n. \lambda f. \lambda x. n[\lambda g. \lambda h. h [g f]] [\lambda u. x] [\lambda u. u]$$

or alternatively

$$\overline{\text{PRED}} = \lambda n. n[\lambda g. \lambda k. [g \overline{1}] [\lambda u. \overline{\text{PLUS}} [g k] \overline{1}] k] [\lambda l. \overline{0}] \overline{0}$$

Note the trick $[g \overline{1}] [\lambda u. \overline{\text{PLUS}} [g k] \overline{1}] k$ which evaluates to k if $[g \overline{1}]$ is $\overline{0}$ and to $[g k] + \overline{1}$ otherwise.

λ -Calculus: Sets



$$\{x | x^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

λ -Calculus: Sets



$$\{x | x^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

The set A has two elements:

$$\exists A. \exists m, n. m \in A \wedge n \in A \wedge m \neq n$$

$$\{x|x^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

The set A has two elements:

$$\exists A. \exists m, n. m \in A \wedge n \in A \wedge m \neq n$$

In first-order, A can be 'defined' by:

$$[x \in A] \equiv [x^2 - 1 = 0]$$

$$\{x|x^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

The set A has two elements:

$$\exists A. \exists m, n. m \in A \wedge n \in A \wedge m \neq n$$

In first-order, A can be 'defined' by:

$$[x \in A] \equiv [x^2 - 1 = 0]$$

In this expression we talk about 'membership'

Alternatively, we can express the characteristic function of A by the λ -term

$$[\lambda x. [x^2 - 1 = 0]]$$

$$\{x|x^2 - 1 = 0\}$$

$$(\{-1, 1\})$$

The set A has two elements:

$$\exists A. \exists m, n. m \in A \wedge n \in A \wedge m \neq n$$

In first-order, A can be 'defined' by:

$$[x \in A] \equiv [x^2 - 1 = 0]$$

In this expression we talk about 'membership'

$$[\lambda x. x^2 - 1 = 0]$$

$$[\lambda x.x^2 - 1 = 0]$$

The idea is as follows

$[[\lambda x.x^2 - 1 = 0] a]$ evaluates to $a^2 - 1 = 0$

$$[\lambda x.x^2 - 1 = 0]$$

The idea is as follows

$[[\lambda x.x^2 - 1 = 0] a]$ evaluates to $a^2 - 1 = 0$

The expression $a^2 - 1 = 0$ is \top (\top denotes Truth) if a is -1 or 1 .

$$[\lambda x.x^2 - 1 = 0]$$

The idea is as follows

$[[\lambda x.x^2 - 1 = 0] a]$ evaluates to $a^2 - 1 = 0$

The expression $a^2 - 1 = 0$ is \top (\top denotes Truth) if a is -1 or 1 .

Otherwise, $a^2 - 1 = 0$ is \perp (\perp denotes Falsehood)

$$[\lambda x.x^2 - 1 = 0]$$

The idea is as follows

$[[\lambda x.x^2 - 1 = 0] a]$ evaluates to $a^2 - 1 = 0$

The expression $a^2 - 1 = 0$ is \top (\top denotes Truth) if a is -1 or 1 .

Otherwise, $a^2 - 1 = 0$ is \perp (\perp denotes Falsehood)

The characteristic function $[\lambda x.x^2 - 1 = 0]$ provides a witness for

$$\exists P. \exists m, n. [P m] \wedge [P n] \wedge m \neq n$$

λ -Calculus: Sets



For each natural number n there is a Church numeral:

$$\bar{n} = \lambda f. \lambda y. [f^n y]$$

λ -Calculus: Sets



For each natural number n there is a Church numeral:

$$\bar{n} = \lambda f. \lambda y. [f^n y]$$

We can also define the set $\bar{\mathbb{N}}$ of all Church numerals

λ -Calculus: Sets



For each natural number n there is a Church numeral:

$$\bar{n} = \lambda f. \lambda y. [f^n y]$$

We can also define the set $\bar{\mathbb{N}}$ of all Church numerals
 $\bar{\mathbb{N}}$ must satisfy three properties:

λ -Calculus: Sets

For each natural number n there is a Church numeral:

$$\bar{n} = \lambda f. \lambda y. [f^n y]$$

We can also define the set $\bar{\mathbb{N}}$ of all Church numerals
 $\bar{\mathbb{N}}$ must satisfy three properties:

1. $[\bar{\mathbb{N}} \bar{0}]$ ‘ $\bar{0}$ is a Church numeral’

λ -Calculus: Sets



For each natural number n there is a Church numeral:

$$\bar{n} = \lambda f. \lambda y. [f^n y]$$

We can also define the set $\bar{\mathbb{N}}$ of all Church numerals
 $\bar{\mathbb{N}}$ must satisfy three properties:

1. $[\bar{\mathbb{N}} \bar{0}]$ “ $\bar{0}$ is a Church numeral”
2. $\forall x. [\bar{\mathbb{N}} x] \supset [\bar{\mathbb{N}} [\text{SUCC } x]]$ “ $\bar{\mathbb{N}}$ is closed under successor”

λ -Calculus: Sets



For each natural number n there is a Church numeral:

$$\bar{n} = \lambda f. \lambda y. [f^n y]$$

We can also define the set $\bar{\mathbb{N}}$ of all Church numerals
 $\bar{\mathbb{N}}$ must satisfy three properties:

1. $[\bar{\mathbb{N}} \bar{0}]$ “ $\bar{0}$ is a Church numeral”
2. $\forall x. [\bar{\mathbb{N}} x] \supset [\bar{\mathbb{N}} [\text{SUCC } x]]$ “ $\bar{\mathbb{N}}$ is closed under successor”
3. $\forall P. [P \bar{0}] \wedge [\forall x. [P x] \supset [P [\text{SUCC } x]]] \supset [\bar{\mathbb{N}} \subseteq P]$
“ $\bar{\mathbb{N}}$ is the least such set”

λ -Calculus: Sets



For each natural number n there is a Church numeral:

$$\bar{n} = \lambda f. \lambda y. [f^n y]$$

We can also define the set $\bar{\mathbb{N}}$ of all Church numerals
 $\bar{\mathbb{N}}$ must satisfy three properties:

1. $[\bar{\mathbb{N}} \bar{0}]$ “ $\bar{0}$ is a Church numeral”
2. $\forall x. [\bar{\mathbb{N}} x] \supset [\bar{\mathbb{N}} [\text{SUCC } x]]$ “ $\bar{\mathbb{N}}$ is closed under successor”
3. $\forall P. [P \bar{0}] \wedge [\forall x. [P x] \supset [P [\text{SUCC } x]]] \supset [\bar{\mathbb{N}} \subseteq P]$
“ $\bar{\mathbb{N}}$ is the least such set”

Define $\bar{\mathbb{N}}$ to be:

$$\lambda z. \forall P. [[P \bar{0}] \wedge [\forall x. [P x] \supset [P [\text{SUCC } x]]]] \supset [P z]$$

λ -Calculus: Sets



Define $\bar{\mathbb{N}}$ to be:

$$\lambda z. \forall P. [[P \bar{0}] \wedge [\forall x. [P x] \supset [P [\text{SUCC } x]]]] \supset [P z]$$

λ -Calculus: Sets



Define \bar{N} to be:

$$\lambda z. \forall P. [[P \bar{0}] \wedge [\forall x. [P x] \supset [P . \overline{\text{SUCC}} x]]] \supset [P z]$$

This satisfies the three requirements.

λ -Calculus: Sets



Define \bar{N} to be:

$$\lambda z. \forall P. [[P \bar{0}] \wedge [\forall x. [P x] \supset [P . \overline{\text{SUCC}} x]]] \supset [P z]$$

This satisfies the three requirements.

- $[\bar{N} \bar{0}]$ since $[P \bar{0}]$ implies $[P \bar{0}]$
- $\forall x. [\bar{N} x] \supset [\bar{N} [\overline{\text{SUCC}} x]]$ since if $P x$ and P is closed under successor, then $P [\overline{\text{SUCC}} p]$

λ -Calculus: Sets



Define \bar{N} to be:

$$\lambda z. \forall P. [[P \bar{0}] \wedge [\forall x. [P x] \supset [P . \overline{\text{SUCC}} x]]] \supset [P z]$$

This satisfies the three requirements.

- $[\bar{N} \bar{0}]$ since $[P \bar{0}]$ implies $[P \bar{0}]$

λ -Calculus: Sets



λ -Calculus: Sets

Define \bar{N} to be:

$$\lambda z. \forall P. [[P \bar{0}] \wedge [\forall x. [P x] \supset [P . \overline{\text{SUCC}} x]]] \supset [P z]$$

This satisfies the three requirements.

- $[\bar{N} \bar{0}]$ since $[P \bar{0}]$ implies $[P \bar{0}]$
- $\forall x. [\bar{N} x] \supset [\bar{N} [\overline{\text{SUCC}} x]]$ since if $P x$ and P is closed under successor, then $P [\overline{\text{SUCC}} p]$
- $\forall P. [P \bar{0}] \wedge [\forall x. [P x] \supset [P . \overline{\text{SUCC}} x]] \supset [\bar{N} \subseteq P]$

\bar{N} is the least such set as the intersection of all such sets P

Define \bar{N} to be:

$$\lambda z. \forall P. [[P \bar{0}] \wedge [\forall x. [P x] \supset [P . \text{SUCC } x]]] \supset [P z]$$

This satisfies the three requirements.

We have used quantification over sets (characteristic functions – the variable P) to define \bar{N} .

Our representation framework is very powerful.

Our representation framework is very powerful.

Actually it is so powerful that it is **inconsistent!**

Our representation framework is very powerful.

Actually it is so powerful that it is **inconsistent!**

Russell's paradox:

Consider the term R :

$$[\lambda x. \neg[x x]]$$

Our representation framework is very powerful.

Actually it is so powerful that it is **inconsistent!**

Russell's paradox:

Consider the term R:

$$[\lambda x. \neg[x x]]$$

As a characteristic function, R represents the set of all sets which do not contain themselves:

$$\{x | x \notin x\}$$

Consider the term R:

$$[\lambda x. \neg[x x]]$$

Now we evaluate the expression E := [R R]

$$[[\lambda x. \neg x x] R]$$

Consider the term R:

$$[\lambda x. \neg[x x]]$$

Consider the term R:

$$[\lambda x. \neg[x x]]$$

Now we evaluate the expression E := [R R]

$$[[\lambda x. \neg x x] R]$$

evaluates to

λ -Calculus: Russell's Paradox



Consider the term R:

$$[\lambda x. \neg[x x]]$$

Now we evaluate the expression E := [R R]

$[[\lambda x. \neg.x x] R]$ evaluates to $\neg[R R]$

λ -Calculus: Russell's Paradox



Consider the term R:

$$[\lambda x. \neg[x x]]$$

Now we evaluate the expression E := [R R]

$[[\lambda x. \neg.x x] R]$ evaluates to $\neg[R R]$

And we evaluate $\neg[R R]$

$$\neg[[\lambda x. \neg.x x] R]$$

λ -Calculus: Russell's Paradox



Consider the term R:

$$[\lambda x. \neg[x x]]$$

Now we evaluate the expression E := [R R]

$[[\lambda x. \neg.x x] R]$ evaluates to $\neg[R R]$

And we evaluate $\neg[R R]$

$\neg[[\lambda x. \neg.x x] R]$ evaluates to

λ -Calculus: Russell's Paradox



Consider the term R:

$$[\lambda x. \neg[x x]]$$

Now we evaluate the expression E := [R R]

$[[\lambda x. \neg.x x] R]$ evaluates to $\neg[R R]$

And we evaluate $\neg[R R]$

$\neg[[\lambda x. \neg.x x] R]$ evaluates to $\neg\neg[R R]$

λ -Calculus: Russell's Paradox



Consider the term R:

$$[\lambda x. \neg[x x]]$$

Now we evaluate the expression E := [R R]

$$[[\lambda x. \neg.x x] R] \text{ evaluates to } \neg[R R]$$

And we evaluate $\neg[R R]$

$$\neg[[\lambda x. \neg.x x] R] \text{ evaluates to } \neg\neg[R R]$$

which is equivalent to [R R]

λ -Calculus: Russell's Paradox



Consider the term R:

$$[\lambda x. \neg[x x]]$$

Now we evaluate the expression E := [R R]

$$[[\lambda x. \neg.x x] R] \text{ evaluates to } \neg[R R]$$

And we evaluate $\neg[R R]$

$$\neg[[\lambda x. \neg.x x] R] \text{ evaluates to } \neg\neg[R R]$$

which is equivalent to [R R]

Thus if E holds we can infer $\neg E$ and vice versa. This is Russell's paradox.

λ -Calculus: Nontermination



Note that the term $[\lambda x. \neg.x x]$ (just as the standard example $[\lambda x. x x]$) does not terminate with respect to β -reduction:

$$[R R] \xrightarrow{\beta} \neg[R R] \xrightarrow{\beta} \neg\neg[R R] \xrightarrow{\beta} \dots$$

Typed λ -Calculus



We can avoid Russell's paradox using simple types.

We can avoid Russell's paradox using simple types.

Simple Types:

- o Base type of propositions

We can avoid Russell's paradox using simple types.

Simple Types:

- o Base type of propositions
- ι Base type of individuals

We can avoid Russell's paradox using simple types.

Simple Types:

- o Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \rightarrow \alpha)$) Type of functions from β to α

We can avoid Russell's paradox using simple types.

Simple Types:

- o Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \rightarrow \alpha)$) Type of functions from β to α

One may include arbitrarily many base types $\iota^1, \dots, \iota^n, \dots$

We can avoid Russell's paradox using simple types.

Simple Types:

- \circ Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \rightarrow \alpha)$) Type of functions from β to α

We often omit parenthesis in types. $(\alpha\beta\gamma)$ means $((\alpha\beta)\gamma)$

We can avoid Russell's paradox using simple types.

Simple Types:

- \circ Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \rightarrow \alpha)$) Type of functions from β to α

We often omit parenthesis in types. $(\alpha\beta\gamma)$ means $((\alpha\beta)\gamma)$

Likewise $(\gamma \rightarrow \beta \rightarrow \alpha)$ means $(\gamma \rightarrow (\beta \rightarrow \alpha))$

We can avoid Russell's paradox using simple types.

Simple Types:

- \circ Base type of propositions
- ι Base type of individuals
- $(\alpha\beta)$ (or $(\beta \rightarrow \alpha)$) Type of functions from β to α

We often omit parenthesis in types. $(\alpha\beta\gamma)$ means $((\alpha\beta)\gamma)$

Likewise $(\gamma \rightarrow \beta \rightarrow \alpha)$ means $(\gamma \rightarrow (\beta \rightarrow \alpha))$

Note that the type $(\alpha\beta\gamma)$ (or $(\gamma \rightarrow \beta \rightarrow \alpha)$) is the type of a (Curried) function of two arguments which returns a value of type α .

- Typed Variables x_α

- Typed Variables x_α
- Typed Constants and Parameters P_α

- Typed Variables x_α
- Typed Constants and Parameters P_α
- Application $[F_{\alpha\beta}B_\beta]_\alpha$ – or $[F_{\beta\rightarrow\alpha}B_\beta]_\alpha$

- Typed Variables x_α
- Typed Constants and Parameters P_α
- Application $[F_{\alpha\beta}B_\beta]_\alpha$ – or $[F_{\beta\rightarrow\alpha}B_\beta]_\alpha$
- λ -abstraction $[\lambda y_\beta. A_\alpha]_{\alpha\beta}$ – or $[\lambda y_\beta. A_\alpha]_{\beta\rightarrow\alpha}$

- Typed Variables x_α
- Typed Constants and Parameters P_α
- Application $[F_{\alpha\beta}B_\beta]_\alpha$ – or $[F_{\beta\rightarrow\alpha}B_\beta]_\alpha$
- λ -abstraction $[\lambda y_\beta. A_\alpha]_{\alpha\beta}$ – or $[\lambda y_\beta. A_\alpha]_{\beta\rightarrow\alpha}$

Examples:

- $[\lambda x_\alpha. x_\alpha]$ term of type $(\alpha\alpha)$ – identity on type α

- Typed Variables x_α
- Typed Constants and Parameters P_α
- Application $[F_{\alpha\beta}B_\beta]_\alpha$ – or $[F_{\beta\rightarrow\alpha}B_\beta]_\alpha$
- λ -abstraction $[\lambda y_\beta. A_\alpha]_{\alpha\beta}$ – or $[\lambda y_\beta. A_\alpha]_{\beta\rightarrow\alpha}$

Examples:

- $[\lambda x_\alpha. x_\alpha]$ term of type $(\alpha\alpha)$ – identity on type α
- $[\lambda y_\beta. x_\alpha]$ term of type $(\alpha\beta)$ – constant x -valued function

Consider the untyped term

$$[\lambda x. x^2 - 1]$$

This is shorthand for

$$[\lambda x. [\text{MINUS} [\text{SQUARE} x] 1]]$$

where MINUS, SQUARE and 1 are constants.

Consider the untyped term

$$[\lambda x. x^2 - 1]$$

Consider the untyped term

$$[\lambda x. x^2 - 1]$$

This is shorthand for

$$[\lambda x. [\text{MINUS} [\text{SQUARE} x] 1]]$$

where MINUS, SQUARE and 1 are constants.
Is there a corresponding typed term?

Typed λ -Calculus: Typed Terms



Consider the untyped term

$$[\lambda x. x^2 - 1]$$

This is shorthand for

$$[\lambda x. [\text{MINUS} [\text{SQUARE} x] 1]]$$

where MINUS, SQUARE and 1 are constants.

Is there a corresponding typed term?

Assume the type of individuals ι corresponds to real numbers.

Typed λ -Calculus: Typed Terms



Consider the untyped term

$$[\lambda x. x^2 - 1]$$

This is shorthand for

$$[\lambda x. [\text{MINUS} [\text{SQUARE} x] 1]]$$

where MINUS, SQUARE and 1 are constants.

Is there a corresponding typed term?

Assume the type of individuals ι corresponds to real numbers.

- x and 1 should be real numbers (type ι)

Typed λ -Calculus: Typed Terms



Consider the untyped term

$$[\lambda x. x^2 - 1]$$

This is shorthand for

$$[\lambda x. [\text{MINUS} [\text{SQUARE} x] 1]]$$

where MINUS, SQUARE and 1 are constants.

Is there a corresponding typed term?

Assume the type of individuals ι corresponds to real numbers.

- x and 1 should be real numbers (type ι)
- SQUARE should take a real number to a real number (type $(\iota\iota)$)

Typed λ -Calculus: Typed Terms



Consider the untyped term

$$[\lambda x. x^2 - 1]$$

This is shorthand for

$$[\lambda x. [\text{MINUS} [\text{SQUARE} x] 1]]$$

where MINUS, SQUARE and 1 are constants.

Is there a corresponding typed term?

Assume the type of individuals ι corresponds to real numbers.

- x and 1 should be real numbers (type ι)
- SQUARE should take a real number to a real number (type $(\iota\iota)$)
- MINUS should take two real numbers to a real number (type $(\iota\iota\iota)$)

Typed λ -Calculus: Typed Terms



Consider the untyped term

$$[\lambda x. x^2 - 1]$$

This is shorthand for

$$[\lambda x. [\text{MINUS} [\text{SQUARE} x] 1]]$$

where MINUS, SQUARE and 1 are constants.

Is there a corresponding typed term?

Assume the type of individuals ι corresponds to real numbers.

Typed Term:

$$[\lambda x_\iota. [\text{MINUS}_{\iota\iota} [\text{SQUARE}_\iota x_\iota] 1_\iota]]$$

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL'06-[2] – p.64

Typed λ -Calculus: Typed Terms



Consider the untyped term

$$[\lambda x. [x^2 - 1 = 0]]$$

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL'06-[2] – p.65

Typed λ -Calculus: Typed Terms



Consider the untyped term

$$[\lambda x. x^2 - 1]$$

This is shorthand for

$$[\lambda x. [\text{MINUS} [\text{SQUARE} x] 1]]$$

where MINUS, SQUARE and 1 are constants.

Is there a corresponding typed term?

Assume the type of individuals ι corresponds to real numbers.

Typed Term:

$$[\lambda x_\iota. [\text{MINUS}_{\iota\iota} [\text{SQUARE}_\iota x_\iota] 1_\iota]]$$

This term has type $(\iota\iota)$.

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL'06-[2] – p.64

Typed λ -Calculus: Typed Terms



Consider the untyped term

$$[\lambda x. [x^2 - 1 = 0]]$$

This is shorthand for

$$[\lambda x. [= [\text{MINUS} [\text{SQUARE} x] 1] 0]]$$

where =, MINUS, SQUARE, 0 and 1 are constants.

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL'06-[2] – p.65

Consider the untyped term

$$[\lambda x. [x^2 - 1 = 0]]$$

This is shorthand for

$$[\lambda x. [= [\text{MINUS} [\text{SQUARE} x] 1] 0]]$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.

- Already know types of MINUS, SQUARE and 1.

Consider the untyped term

$$[\lambda x. [x^2 - 1 = 0]]$$

This is shorthand for

$$[\lambda x. [= [\text{MINUS} [\text{SQUARE} x] 1] 0]]$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.

- Already know types of MINUS, SQUARE and 1.
- 0 should be a real number (type ν)
- $=$ takes two real numbers and returns a truth value (type $(\text{o} \nu \nu)$)

Consider the untyped term

$$[\lambda x. [x^2 - 1 = 0]]$$

This is shorthand for

$$[\lambda x. [= [\text{MINUS} [\text{SQUARE} x] 1] 0]]$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.

- Already know types of MINUS, SQUARE and 1.
- 0 should be a real number (type ν)

Consider the untyped term

$$[\lambda x. [x^2 - 1 = 0]]$$

This is shorthand for

$$[\lambda x. [= [\text{MINUS} [\text{SQUARE} x] 1] 0]]$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.

Typed Term:

$$[\lambda x_\nu. [=_{\text{o} \nu} [\text{MINUS}_\nu [\text{SQUARE}_\nu x_\nu] 1_\nu] 0_\nu]]$$

Consider the untyped term

$$[\lambda x. [x^2 - 1 = 0]]$$

This is shorthand for

$$[\lambda x. [= [\text{MINUS} [\text{SQUARE} x] 1] 0]]$$

where $=$, MINUS, SQUARE, 0 and 1 are constants.

Typed Term:

$$[\lambda x_i. [=_{\text{o}\iota} [\text{MINUS}_{\iota\iota} [\text{SQUARE}_{\iota\iota} x_i] 1_i] 0_i]$$

This term has type $(\text{o}\iota)$.

Typed λ -Calculus: Assigning Types

The basis for such an algorithm is the following deduction system:

General algorithm for assigning types to terms (when this is possible) – see Hindley97.

Typed λ -Calculus: Assigning Types

The basis for such an algorithm is the following deduction system:

$$\frac{C : \alpha \in \Gamma \quad C \text{ variable, parameter or constant}}{\Gamma \vdash_{\text{TA}} C : \alpha} \text{Hyp}$$

Typed λ -Calculus: Assigning Types



The basis for such an algorithm is the following deduction system:

$$\frac{C : \alpha \in \Gamma \quad C \text{ variable, parameter or constant}}{\Gamma \vdash_{TA} C : \alpha} \text{ Hyp}$$

$$\frac{\Gamma, y : \beta \vdash_{TA} A : \alpha}{\Gamma \vdash_{TA} [\lambda y. A] : \alpha\beta} \text{ Lam}$$

Typed λ -Calculus: Assigning Types



The basis for such an algorithm is the following deduction system:

$$\frac{C : \alpha \in \Gamma \quad C \text{ variable, parameter or constant}}{\Gamma \vdash_{TA} C : \alpha} \text{ Hyp}$$

$$\frac{\Gamma, y : \beta \vdash_{TA} A : \alpha}{\Gamma \vdash_{TA} [\lambda y. A] : \alpha\beta} \text{ Lam}$$

$$\frac{\Gamma \vdash_{TA} F : \alpha\beta \quad \Gamma \vdash_{TA} B : \beta}{\Gamma \vdash_{TA} [FB] : \alpha} \text{ App}$$

Typed λ -Calculus: Assigning Types



The basis for such an algorithm is the following deduction system:

$$\frac{C : \alpha \in \Gamma \quad C \text{ variable, parameter or constant}}{\Gamma \vdash_{TA} C : \alpha} \text{ Hyp}$$

$$\frac{\Gamma, y : \beta \vdash_{TA} A : \alpha}{\Gamma \vdash_{TA} [\lambda y. A] : \alpha\beta} \text{ Lam}$$

$$\frac{\Gamma \vdash_{TA} F : \alpha\beta \quad \Gamma \vdash_{TA} B : \beta}{\Gamma \vdash_{TA} [FB] : \alpha} \text{ App}$$

We can assign the type α to a term A in context Γ whenever we can derive

$$\Gamma \vdash_{TA} A : \alpha$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \text{SQUARE}x]$

Goal: Find a type α such that

$$\text{SQUARE} : (\underline{\alpha}) \vdash_{TA} [\lambda x. \text{SQUARE}x] : \alpha$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. [\text{SQUARE}x]]$

Goal: Find a type α such that

$\text{SQUARE} : (\underline{\alpha}) \vdash_{\text{TA}} [\lambda x. [\text{SQUARE}x]] : \alpha$

⋮
 $\text{SQUARE} : (\underline{\alpha}) \vdash_{\text{TA}} [\lambda x. [\text{SQUARE}x]] : \alpha$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. [\text{SQUARE}x]]$

Goal: Find a type α such that

$\text{SQUARE} : (\underline{\alpha}) \vdash_{\text{TA}} [\lambda x. [\text{SQUARE}x]] : \alpha$

α is $(\gamma\beta)$

$$\frac{\text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} [\text{SQUARE}x] : \gamma}{\text{SQUARE} : (\underline{\alpha}) \vdash_{\text{TA}} [\lambda x. [\text{SQUARE}x]] : \gamma\beta} \text{ Lam}$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. [\text{SQUARE}x]]$

Goal: Find a type α such that

$\text{SQUARE} : (\underline{\alpha}) \vdash_{\text{TA}} [\lambda x. [\text{SQUARE}x]] : \alpha$

$$\frac{\text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} \text{SQUARE} : (\gamma\delta) \quad \text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} x : \delta}{\text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} [\text{SQUARE}x] : \gamma} \text{ App}$$

$$\frac{\text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} [\text{SQUARE}x] : \gamma}{\text{SQUARE} : (\underline{\alpha}) \vdash_{\text{TA}} [\lambda x. [\text{SQUARE}x]] : \gamma\beta} \text{ Lam}$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. [\text{SQUARE}x]]$

Goal: Find a type α such that

$\text{SQUARE} : (\underline{\alpha}) \vdash_{\text{TA}} [\lambda x. [\text{SQUARE}x]] : \alpha$

γ and δ are both $\underline{\alpha}$

$$\frac{\text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} \text{SQUARE} : (\underline{\alpha}) \quad \text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} x : \underline{\alpha}}{\text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} [\text{SQUARE}x] : \underline{\alpha}} \text{ Hyp}$$

$$\frac{\text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} [\text{SQUARE}x] : \underline{\alpha}}{\text{SQUARE} : (\underline{\alpha}) \vdash_{\text{TA}} [\lambda x. [\text{SQUARE}x]] : \underline{\alpha}\beta} \text{ App}$$

$$\frac{\text{SQUARE} : (\underline{\alpha}), x : \beta \vdash_{\text{TA}} [\text{SQUARE}x] : \underline{\alpha}}{\text{SQUARE} : (\underline{\alpha}) \vdash_{\text{TA}} [\lambda x. [\text{SQUARE}x]] : \underline{\alpha}\beta} \text{ Lam}$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. [SQUARE]x]$

Goal: Find a type α such that

$SQUARE : (\nu) \vdash_{TA} [\lambda x. [SQUARE]x] : \alpha$

β is ν

$$\frac{\text{SQUARE} : (\nu), x : \nu \vdash_{TA} \text{SQUARE} : (\nu) \quad \text{Hyp} \quad \text{SQUARE} : (\nu), x : \nu \vdash_{TA} x : \nu \quad \text{Hyp}}{\frac{\text{SQUARE} : (\nu), x : \nu \vdash_{TA} [\text{SQUARE}]x : \nu \quad \text{Lam}}{\text{SQUARE} : (\nu) \vdash_{TA} [\lambda x. [SQUARE]x] : \nu}}$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. [SQUARE]x]$

Goal: Find a type α such that

$SQUARE : (\nu) \vdash_{TA} [\lambda x. [SQUARE]x] : \alpha$

β is ν

$$\frac{\text{SQUARE} : (\nu), x : \nu \vdash_{TA} \text{SQUARE} : (\nu) \quad \text{Hyp} \quad \text{SQUARE} : (\nu), x : \nu \vdash_{TA} x : \nu \quad \text{Hyp}}{\frac{\text{SQUARE} : (\nu), x : \nu \vdash_{TA} [\text{SQUARE}]x : \nu \quad \text{Lam}}{\text{SQUARE} : (\nu) \vdash_{TA} [\lambda x. [SQUARE]x] : \nu}}$$

So $[\lambda x. [SQUARE]x]$ can be assigned the type (ν) in context
 $\text{SQUARE} : (\nu)$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. [SQUARE]x]$

Goal: Find a type α such that

$SQUARE : (\nu) \vdash_{TA} [\lambda x. [SQUARE]x] : \alpha$

β is ν

$$\frac{\text{SQUARE} : (\nu), x : \nu \vdash_{TA} \text{SQUARE} : (\nu) \quad \text{Hyp} \quad \text{SQUARE} : (\nu), x : \nu \vdash_{TA} x : \nu \quad \text{Hyp}}{\frac{\text{SQUARE} : (\nu), x : \nu \vdash_{TA} [\text{SQUARE}]x : \nu \quad \text{Lam}}{\text{SQUARE} : (\nu) \vdash_{TA} [\lambda x. [SQUARE]x] : \nu}}$$

So $[\lambda x. [SQUARE]x]$ can be assigned the type (ν) in context
 $\text{SQUARE} : (\nu)$

Corresponding Typed Term: $[\lambda x_\nu. [SQUARE]_{\nu} x_\nu]$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (\alpha\alpha) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$

$$\neg : (oo) \vdash_{TA} \dots [\lambda x. \neg [xx]] : \alpha$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$
 α is $(\gamma\beta)$

$$\frac{\neg : (oo), x : \beta \vdash_{TA} [\neg [xx]] : \gamma}{\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \gamma\beta} \text{ Lam}$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$

$$\frac{\neg : (oo), x : \beta \vdash_{TA} \neg : (\gamma\delta) \quad \neg : (oo), x : \beta \vdash_{TA} [xx] : \delta}{\neg : (oo), x : \beta \vdash_{TA} [\neg [xx]] : \gamma} \text{ App}$$

$$\frac{\neg : (oo), x : \beta \vdash_{TA} [\neg [xx]] : \gamma}{\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \gamma\beta} \text{ Lam}$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$
 γ and δ are both \circ

$$\frac{\neg : (oo), x : \beta \vdash_{TA} \neg : (oo) \quad \neg : (oo), x : \beta \vdash_{TA} [xx] : \circ}{\neg : (oo), x : \beta \vdash_{TA} [\neg [xx]] : \circ} \text{ Hyp}$$

$$\frac{\neg : (oo), x : \beta \vdash_{TA} [\neg [xx]] : \circ}{\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \circ\beta} \text{ App}$$

$$\frac{\neg : (oo), x : \beta \vdash_{TA} [\neg [xx]] : \circ}{\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \circ\beta} \text{ Lam}$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$

$$\neg : (oo), x : \beta \vdash_{TA} [xx] : o$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$

$$\frac{\neg : (oo), x : \beta \vdash_{TA} x : (o\epsilon) \quad \neg : (oo), x : \beta \vdash_{TA} x : \epsilon}{\neg : (oo), x : \beta \vdash_{TA} [xx] : o} \text{App}$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$

β is $(o\epsilon)$

$$\frac{\neg : (oo), x : (o\epsilon) \vdash_{TA} x : (o\epsilon) \quad \neg : (oo), x : (o\epsilon) \vdash_{TA} x : \epsilon}{\neg : (oo), x : (o\epsilon) \vdash_{TA} [xx] : o} \text{App}$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$

Only remaining subgoal:

$$\neg : (oo), x : (o\epsilon) \vdash_{TA} x : \epsilon$$

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$

Only remaining subgoal:

$$\neg : (oo), x : (\omega\epsilon) \vdash_{TA} x : \epsilon$$

This goal cannot be solved since $(\omega\epsilon)$ cannot equal ϵ .

Typed λ -Calculus: Assigning Types



Untyped Term: $[\lambda x. \neg [xx]]$

Goal: Find a type α such that $\neg : (oo) \vdash_{TA} [\lambda x. \neg [xx]] : \alpha$

Only remaining subgoal:

$$\neg : (oo), x : (\omega\epsilon) \vdash_{TA} x : \epsilon$$

This goal cannot be solved since $(\omega\epsilon)$ cannot equal ϵ .

Hence $[\lambda x. \neg [xx]]$ cannot be typed – avoiding Russell's Paradox.

Typed λ -Calculus: $\beta\eta$



β -reduction:

$$[[\lambda y_\beta . A_\alpha] B_\beta] \xrightarrow{\beta} A_\alpha[y_\beta/B_\beta]$$

Typed λ -Calculus: $\beta\eta$



β -reduction:

$$[[\lambda y_\beta . A_\alpha] B_\beta] \xrightarrow{\beta} A_\alpha[y_\beta/B_\beta]$$

η -reduction:

$$[\lambda y_\beta . F_{\alpha\beta} y_\beta] \xrightarrow{\eta} F_{\alpha\beta}$$

β -reduction:

$$[[\lambda y_\beta . A_\alpha] B_\beta] \longrightarrow_\beta A_\alpha[y_\beta/B_\beta]$$

η -reduction:

$$[\lambda y_\beta . F_{\alpha\beta} y_\beta] \longrightarrow_\eta F_{\alpha\beta}$$

Facts:

- $\beta\eta$ -normalization terminates for typed terms.

β -reduction:

$$[[\lambda y_\beta . A_\alpha] B_\beta] \longrightarrow_\beta A_\alpha[y_\beta/B_\beta]$$

η -reduction:

$$[\lambda y_\beta . F_{\alpha\beta} y_\beta] \longrightarrow_\eta F_{\alpha\beta}$$

Facts:

- $\beta\eta$ -normalization terminates for typed terms.
- Every typed term has a unique $\beta\eta$ -normal form.



Introduction (Contd.)

Typed λ -Calculus: Logical Constants

We gain expressive power by combining typed λ -calculus with logical constants.

We gain expressive power by combining typed λ -calculus with logical constants.

\top_o – true

\perp_o – false

\neg_{oo} – negation

\vee_{ooo} – disjunction

\wedge_{ooo} – conjunction

\supset_{ooo} – implication

\equiv_{ooo} – equivalence

We gain expressive power by combining typed λ -calculus with logical constants.

$=_{o\alpha\alpha}^{\alpha}$ – equality at type α

$\Pi_{o(o\alpha)}^{\alpha}$ – universal quantification over type α

$\Sigma_{o(o\alpha)}^{\alpha}$ – existential quantification over type α

Intuition: $[\Sigma^{\alpha} . \lambda x_{\alpha} . C_o]$ is to true iff $\{x_{\alpha} | C\}$ is nonempty.

Church's Classical Type Theory: HOL

HOL: Abbreviations

$[A_o \vee B_o]$ means $[\vee_{ooo} A_o B_o]$

$[A_o \wedge B_o]$ means $[\wedge_{ooo} A_o B_o]$

$[A_o \supset B_o]$ means $[\supset_{ooo} A_o B_o]$

$[A_o \equiv B_o]$ means $[\equiv_{ooo} A_o B_o]$

$[A_{\alpha} =^{\alpha} B_{\alpha}]$ means $[=_{o\alpha\alpha}^{\alpha} A_{\alpha} B_{\alpha}]$

$[\forall x_{\alpha}. A_o]$ means $[\Pi_{o(o\alpha)}^{\alpha} . \lambda x_{\alpha}. A_o]$.

$[\exists x_{\alpha}. A_o]$ means $[\Sigma_{o(o\alpha)}^{\alpha} . \lambda x_{\alpha}. A_o]$.

HOL: Expressing Properties

$[\lambda x_{\iota}. x^2 - 1]$

$$[\lambda x_\iota. x^2 - 1]$$

$$[\lambda x_\iota. [\text{MINUS}_{\iota\iota} [\text{SQUARE}_\iota x] 1_\iota]]_\iota$$

$$[\lambda x_\iota. x^2 - 1]$$

Term of type \circ expressing existence of an f with two roots:

$$[\exists f_\iota. \exists n_\iota. \exists m_\iota. [[f n] =^\iota 0_\iota] \wedge [[f m] =^\iota 0_\iota] \wedge \neg[n =^\iota m]]_\circ$$

$$[\lambda x_\iota. x^2 - 1]$$

Term of type \circ expressing existence of an f with two roots:

$$[\underbrace{\exists f_\iota. \exists n_\iota. \exists m_\iota. [[f n] =^\iota 0_\iota] \wedge [[f m] =^\iota 0_\iota]}_{\sum^\iota \lambda f_\iota} \wedge \neg[n =^\iota m]]_\circ$$

$$[\lambda x_\iota. x^2 - 1]$$

Term of type \circ expressing existence of an f with two roots:

$$[\exists f_\iota. \exists n_\iota. \exists m_\iota. \underbrace{[[f n] =^\iota 0_\iota]}_{[=^\iota [f n] 0]} \wedge \underbrace{[[f m] =^\iota 0_\iota]}_{[=^\iota [f m] 0]} \wedge \neg[n =^\iota m]]_\circ$$

$$[\lambda x_\iota. x^2 - 1]$$

$$[\lambda x_\iota. [x^2 - 1] = 0]$$

$$[\lambda x_\iota. x^2 - 1]$$

$$[\lambda x_\iota. [x^2 - 1] = 0]$$

$$[\lambda x_\iota. [=^\iota [\text{MINUS}_{\iota\iota} [\text{SQUARE}_\iota x] 1_\iota] 0_\iota]]_{\circ\iota}$$

$$[\lambda x_\iota. x^2 - 1]$$

$$[\lambda x_\iota. [x^2 - 1] = 0]$$

Term of type \circ expressing existence of a set (characteristic function) P with two elements

$$[\exists P_\circ. \exists m_\iota. \exists n_\iota. [P m] \wedge [P n] \wedge \neg[m = n]]_\circ$$

Suppose ι corresponds to real numbers.

Given constants: $<_{\circ\iota\iota}$, ABS_ι , $\text{MINUS}_{\iota\iota}$

We can give the usual $\epsilon - \delta$ definition of limits.

HOL: Expressing Properties



Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{ou}(\iota)}$:

HOL: Expressing Properties



Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{ou}(\iota)}$:

$$\begin{aligned} [\lambda f_{\iota\iota}. \lambda a_{\iota\iota}. \lambda L_{\iota\iota}. \forall \epsilon_{\iota\iota}. [\epsilon > 0] \supset . \exists \delta_{\iota\iota}. [\delta > 0] \\ \wedge . \forall x_{\iota\iota}. [|x - a| < \delta] \supset [|f x] - L | < \epsilon] \end{aligned}$$

HOL: Expressing Properties



Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{ou}(\iota)}$:

$$\begin{aligned} [\lambda f_{\iota\iota}. \lambda a_{\iota\iota}. \lambda L_{\iota\iota}. \forall \epsilon_{\iota\iota}. \overbrace{[\epsilon > 0]}^{[< 0 \epsilon]} \supset . \exists \delta_{\iota\iota}. [\delta > 0] \\ \wedge . \forall x_{\iota\iota}. [|x - a| < \delta] \supset [|f x] - L | < \epsilon] \end{aligned}$$

HOL: Expressing Properties



Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{ou}(\iota)}$:

$$\begin{aligned} [\lambda f_{\iota\iota}. \lambda a_{\iota\iota}. \lambda L_{\iota\iota}. \forall \epsilon_{\iota\iota}. [\epsilon > 0] \supset . \exists \delta_{\iota\iota}. \overbrace{[\delta > 0]}^{[< 0 \delta]} \\ \wedge . \forall x_{\iota\iota}. [|x - a| < \delta] \supset [|f x] - L | < \epsilon] \end{aligned}$$

HOL: Expressing Properties



Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{ou}(\iota)}$:

$$[\lambda f_{\iota u} \cdot \lambda a_{\iota} \cdot \lambda L_{\iota} \cdot \forall \epsilon_{\iota} [\epsilon > 0] \supset \exists \delta_{\iota} [\delta > 0] \\ \wedge \forall x_{\iota} [|\underbrace{x - a}_{\text{[MINUS } x a]}| < \delta] \supset |[f x] - L| < \epsilon]$$

HOL: Expressing Properties



Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{ou}(\iota)}$:

$$[\lambda f_{\iota u} \cdot \lambda a_{\iota} \cdot \lambda L_{\iota} \cdot \forall \epsilon_{\iota} [\epsilon > 0] \supset \exists \delta_{\iota} [\delta > 0] \\ \wedge \forall x_{\iota} [|\underbrace{x - a}_{\text{[MINUS } x a]}| < \delta] \supset |[f x] - L| < \epsilon]$$

HOL: Expressing Properties



Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{ou}(\iota)}$:

$$[\lambda f_{\iota u} \cdot \lambda a_{\iota} \cdot \lambda L_{\iota} \cdot \forall \epsilon_{\iota} [\epsilon > 0] \supset \exists \delta_{\iota} [\delta > 0] \\ \wedge \forall x_{\iota} [|\underbrace{x - a}_{\text{[ABS } \text{MINUS } x a]}| < \delta] \supset |[f x] - L| < \epsilon]$$

HOL: Expressing Properties



Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{ou}(\iota)}$:

$$[\lambda f_{\iota u} \cdot \lambda a_{\iota} \cdot \lambda L_{\iota} \cdot \forall \epsilon_{\iota} [\epsilon > 0] \supset \exists \delta_{\iota} [\delta > 0] \\ \wedge \forall x_{\iota} [|\underbrace{x - a}_{\text{[ABS } \text{MINUS } x a]}| < \delta] \supset |[f x] - L| < \epsilon]$$

Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{o}\iota\iota(\iota\iota)}$:

$$[\lambda f_{\iota\iota}. \lambda a_{\iota\iota}. \lambda L_{\iota\iota}. \forall \epsilon_{\iota\iota}. [\epsilon > 0] \supset . \exists \delta_{\iota\iota}. [\delta > 0] \\ \wedge . \forall x_{\iota\iota}. [|x - a| < \delta] \supset [|f x] - L| < \epsilon]]$$

Suppose ι corresponds to real numbers.

Given constants: $<_{\text{uu}}$, ABS_{uu} , MINUS_{uu}

We can give the usual $\epsilon - \delta$ definition of limits.

$\text{LIM}_{\text{o}\iota\iota(\iota\iota)}$:

$$[\lambda f_{\iota\iota}. \lambda a_{\iota\iota}. \lambda L_{\iota\iota}. \forall \epsilon_{\iota\iota}. [\epsilon > 0] \supset . \exists \delta_{\iota\iota}. [\delta > 0] \\ \wedge . \forall x_{\iota\iota}. [|x - a| < \delta] \supset [|f x] - L| < \epsilon]]$$

Similarly can define continuity, differentiation, etc.

HOL: Prefix Polymorphism

Some definitions are naturally expressed using type variables:

HOL: Prefix Polymorphism

Some definitions are naturally expressed using type variables:

Consider the notion of subset:

For each type α we can define $\subseteq_{\text{o}(\text{o}\alpha)(\text{o}\alpha)}$ to be:

$$\lambda X_{\text{o}\alpha}. \lambda Y_{\text{o}\alpha}. [\forall z_{\alpha}. [X z] \supset [Y z]]$$

HOL: Prefix Polymorphism



Some definitions are naturally expressed using type variables:

Consider the notion of subset:

For each type α we can define $\subseteq_{\circ(\alpha)(\alpha)}$ to be:

$$\lambda X_{\circ\alpha}. \lambda Y_{\circ\alpha}. [\forall z_{\alpha}. [X z] \supset [Y z]]$$

We can think of α as a type variable and $\subseteq_{\circ(\alpha)(\alpha)}$ to be polymorphic.



©Benzmüller, 2006

ATPHOL'06-[3] – p.75

HOL: Prefix Polymorphism



Some definitions are naturally expressed using type variables:

Consider the notion of subset:

For each type α we can define $\subseteq_{\circ(\alpha)(\alpha)}$ to be:

$$\lambda X_{\circ\alpha}. \lambda Y_{\circ\alpha}. [\forall z_{\alpha}. [X z] \supset [Y z]]$$

We can think of α as a type variable and $\subseteq_{\circ(\alpha)(\alpha)}$ to be polymorphic. In any particular occurrence of $\subseteq_{\circ(\alpha)(\alpha)}$ we should be able to instantiate the type variable α .

Example: (using infix notation)

$$[\lambda U_{\circ\iota}. [U \subseteq_{\circ(\circ\iota)(\circ\iota)} X_{\circ\iota}]] \subseteq_{\circ(\circ(\circ\iota))(\circ(\circ\iota))} [\lambda U_{\circ\iota}. [U \subseteq_{\circ(\circ\iota)(\circ\iota)} Y_{\circ\iota}]]$$



©Benzmüller, 2006

ATPHOL'06-[3] – p.75

HOL: Prefix Polymorphism



Some definitions are naturally expressed using type variables:

Consider the notion of subset:

For each type α we can define $\subseteq_{\circ(\alpha)(\alpha)}$ to be:

$$\lambda X_{\circ\alpha}. \lambda Y_{\circ\alpha}. [\forall z_{\alpha}. [X z] \supset [Y z]]$$

We can think of α as a type variable and $\subseteq_{\circ(\alpha)(\alpha)}$ to be polymorphic. In any particular occurrence of $\subseteq_{\circ(\alpha)(\alpha)}$ we should be able to instantiate the type variable α .



©Benzmüller, 2006

ATPHOL'06-[3] – p.75

HOL: Cantor's Theorem



There is no surjection from a set A onto the power set $\mathcal{P}(A)$ of A.



ATPHOL'06-[3] – p.76

HOL: Cantor's Theorem



There is no surjection from a set A onto the power set $\mathcal{P}(A)$ of A.

- Suppose A corresponds to type $\underline{\iota}$.

HOL: Cantor's Theorem



There is no surjection from a set A onto the power set $\mathcal{P}(A)$ of A.

- Suppose A corresponds to type $\underline{\iota}$.
- Then $\mathcal{P}(A)$ corresponds to type $(\text{o}\underline{\iota})$.

HOL: Cantor's Theorem

©Benzmüller, 2006



ATPHOL'06-[3] – p.76



There is no surjection from a set A onto the power set $\mathcal{P}(A)$ of A.

- Suppose A corresponds to type $\underline{\iota}$.
- Then $\mathcal{P}(A)$ corresponds to type $(\text{o}\underline{\iota})$.

HOL: Standard Higher-Order Model



ATPHOL'06-[3] – p.76



\mathcal{D}_ι (individuals)

$$\neg \exists g_{\text{o}\underline{\iota}}. \forall f_{\text{o}\underline{\iota}}. \exists x_\iota. g x =^{\text{o}\underline{\iota}} f$$

©Benzmüller, 2006



ATPHOL'06-[3] – p.76

©Benzmüller, 2006



ATPHOL'06-[3] – p.77

HOL: Standard Higher-Order Model



HOL: Standard Higher-Order Model



$\mathcal{P}(\mathcal{D}_\iota)$ (all sets)

\mathcal{D}_ι (individuals)

(all sets of sets)

$\mathcal{P}(\mathcal{P}(\mathcal{D}_\iota))$

$\mathcal{P}(\mathcal{D}_\iota)$ (all sets)

\mathcal{D}_ι (individuals)

HOL: Standard Higher-Order Model



HOL: Henkin-Style Model



$\mathcal{P}(\mathcal{P}(\mathcal{D}_\iota))$

$\mathcal{P}(\mathcal{D}_\iota)$

\mathcal{D}_ι

$\mathcal{D}_{\text{o}\iota} \subseteq \mathcal{P}(\mathcal{D}_\iota)$ (some sets)

\mathcal{D}_ι (individuals)

(some sets of sets)

$$\mathcal{D}_{o(o\iota)} \subseteq \mathcal{P}(\mathcal{D}_{o\iota})$$

$$\mathcal{D}_{o\iota} \subseteq \mathcal{P}(\mathcal{D}_\iota) \quad (\text{some sets})$$

$$\mathcal{D}_\iota \quad (\text{individuals})$$

$$\vdots$$

$$\mathcal{D}_{o(o\iota)} \subseteq \mathcal{P}(\mathcal{D}_{o\iota})$$

$$\mathcal{D}_{o\iota} \subseteq \mathcal{P}(\mathcal{D}_\iota)$$

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



Types, Frames, and
Applicative Structures

Def.: Types

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

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

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

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

Def.: Types



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

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

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

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

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:

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

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

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

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 $\mathbb{N} = \{0, 1, 2, \dots\}$.

- $0 \in \mathbb{N}$

Ex.: Freely Generated



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

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

Ex.: Freely Generated



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

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

Ex.: Freely Generated



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

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

Ex.: Freely Generated



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

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

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

Ex.: Freely Generated



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

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

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

Contrast \mathbb{N} to $\mathbb{Z} = \{\dots, -1, 0, 1, \dots\}$.

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":

- $\circ \neq \iota$

Ex.: Freely Generated



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

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

Ex.: Freely Generated



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

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

Ex.: Freely Generated



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

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

Ex.: Types



Ex.: Types

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

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

Ex.: Types



Ex.: Types

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

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

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



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

Is $(o\iota\iota)$ also a type? – no

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

Def.: Functions



Let A, B be sets.

Def.: Functions



Let A, B be sets.

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

Def.: Functions



Let A, B be sets.

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

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

Def.: Functions

©Benzmüller, 2006



ATPHOL'06-[3] – p.84



Let A, B be sets.

$f : B \rightarrow A$: 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

ATPHOL'06-[3] – p.84



©Benzmüller, 2006



Let A, B be sets.

$f : B \rightarrow A$: 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 $|A^B| = m^n = |A|^{|B|}$.

©Benzmüller, 2006



ATPHOL'06-[3] – p.84

©Benzmüller, 2006



ATPHOL'06-[3] – p.84

Def.: Functions



Let A, B be sets.

$f : B \rightarrow A$: 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.

$f : B \rightarrow A$: 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:

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

Def.: Functions



Let A, B be sets.

$f : B \rightarrow A$: 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:

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



Def.: Functions



Let A, B be sets.

$f : B \rightarrow A$: 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:

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

Def.: Functions

Let A, B be sets.

$f : B \rightarrow A$: 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:

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

Ex.: Sets of Functions

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

$|F| = ?$

Ex.: Sets of Functions

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

$|F| = ?$

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



Ex.: Sets of Labelled Functions



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

$|F| = ?$

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



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

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

Ex.: Sets of Labelled Functions



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

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

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

Def.: Frames



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

Def.: Frames



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

$$\forall \alpha, \beta \in \mathcal{T} : D_{\alpha\beta} \subseteq D_\alpha^{D_\beta}$$

Def.: Frames



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

$$\forall \alpha, \beta \in \mathcal{T} : D_{\alpha\beta} \subseteq D_\alpha^{D_\beta}$$

A Frame is called **standard** if

Def.: Frames



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

$$\forall \alpha, \beta \in \mathcal{T} : D_{\alpha\beta} \subseteq D_\alpha^{D_\beta}$$

A Frame is called **standard** if

$$D_{\alpha\beta} = D_\alpha^{D_\beta} \quad \forall \alpha, \beta \in \mathcal{T}$$

Ex.: Frames



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

Ex.: Frames



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

$$D_i = \{1\}$$

©Benzmüller, 2006



ATPHOL'06-[3] – p.88

Ex.: Frames



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

$$D_i = \{1\}$$

$$D_{\alpha\beta} = D_\alpha^{D_\beta}$$

Ex.: Frames



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

$$D_i = \{1\}$$

$$D_{\alpha\beta} = D_\alpha^{D_\beta}$$

D: the standard frame with $D_o = \{\perp, \top\}$, $D_i = \{1\}$

©Benzmüller, 2006



ATPHOL'06-[3] – p.88

©Benzmüller, 2006



ATPHOL'06-[3] – p.88

Ex.: Frames (Contd.)



Consider the set $D_{o(\iota\iota)}((o(\iota\iota)))$. Is the set empty?

Ex.: Frames (Contd.)



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

Ex.: Frames (Contd.)



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

Claim: $\forall \alpha \in \mathcal{T} : D_\alpha \neq \emptyset$.

Ex.: Frames (Contd.)



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

Claim: $\forall \alpha \in \mathcal{T} : D_\alpha \neq \emptyset$.

Proof: induction on type.

Ex.: Frames (Contd.)



Consider the set $D_{o(u)((o(i\alpha)))}$. Is the set empty? **—no!**

Claim: $\forall \alpha \in T : D_\alpha \neq \emptyset$.

Proof: induction on type.

- **Base:** $D_o = \{\perp, \top\} \neq \emptyset, D_i = \{1\} \neq \emptyset$.

Ex.: Frames (Contd.)



Consider the set $D_{o(u)((o(i\alpha)))}$. Is the set empty? **—no!**

Claim: $\forall \alpha \in T : D_\alpha \neq \emptyset$.

Proof: induction on type.

- **Base:** $D_o = \{\perp, \top\} \neq \emptyset, D_i = \{1\} \neq \emptyset$.
- **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 $D_{o(u)((o(i\alpha)))}$. Is the set empty? **—no!**

Claim: $\forall \alpha \in T : D_\alpha \neq \emptyset$.

Proof: induction on type.

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

Ex.: Frames (Contd.)



Consider the set $D_{o(u)((o(i\alpha)))}$. Is the set empty? **—no!**

Claim: $\forall \alpha \in T : D_\alpha \neq \emptyset$.

Proof: induction on type.

- **Base:** $D_o = \{\perp, \top\} \neq \emptyset, D_i = \{1\} \neq \emptyset$.
- **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 $D_{o(u)((o(i_0)))}$. Is the set empty? —no!

Claim: $\forall \alpha \in \mathcal{T} : D_\alpha \neq \emptyset$.

Proof: induction on type.

- Base: $D_o = \{\perp, \top\} \neq \emptyset, D_i = \{1\} \neq \emptyset$.
- 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

$\langle D, @ \rangle$

where

Def.: Typed Applicative Structure



A (typed) applicative structure is a tupel

$\langle D, @ \rangle$

where

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

Def.: Typed Applicative Structure



A (typed) applicative structure is a tupel

$$\langle D, @ \rangle$$

where

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

Def.: Typed Applicative Structure



A (typed) applicative structure is a tupel

$$\langle D, @ \rangle$$

where

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

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

Rem.: Currying



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

Rem.: Currying

The application operator \textcircled{a} 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 $f@\langle a^1, \dots, a^n \rangle := (\dots (f@a^1) \dots @a^n)$ (“Currying”).

Let D be a frame.

Interesting Properties

Let D be a frame.

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

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 $\langle D, \textcircled{a} \rangle$ be an applicative structure. Consider the property:

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 $\langle D, @ \rangle$. We say that $\langle D, @ \rangle$ is functional if

Def.: Functional Applicative Structures

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

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

Def.: Full Applicative Structures



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

Def.: Full Applicative Structures



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

Def.: Full Applicative Structures



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

$$\forall \alpha, \beta \quad \forall h : D_\beta \rightarrow D_\alpha \quad \exists f \in D_{\alpha\beta} \forall b \in D_\beta : f @ b = h(b)$$

Def.: Standard Applicative Structures



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

Def.: Standard Applicative Structures



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

Def.: Standard Applicative Structures



An applicative structure $\mathcal{A} := \langle D, @ \rangle$ is called **standard** if it is a frame structure (i.e. $@$ is function application) where 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 $D_\alpha = \{1\} \quad \forall \alpha$

Example: Full Functional Appl. Structure



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

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

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

Def.: Homomorphic Appl. Structures



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

Def.: Homomorphic Appl. Structures



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

©Benzmüller, 2006



ATPHOL'06-[3] – p.98

Def.: Homomorphic Appl. Structures



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

- $\kappa_\alpha : D_\alpha^1 \rightarrow D_\alpha^2 \quad \forall \alpha \in \mathcal{T}$

Def.: Homomorphic Appl. Structures



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

- $\kappa_\alpha : D_\alpha^1 \rightarrow D_\alpha^2 \quad \forall \alpha \in \mathcal{T}$
- $\forall \alpha, \beta \in \mathcal{T}, \quad \forall f \in D_{\alpha\beta}^1, \quad \forall b \in D_\beta^1:$

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

©Benzmüller, 2006



ATPHOL'06-[3] – p.98

©Benzmüller, 2006



ATPHOL'06-[3] – p.98

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:

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 $\langle D^2, @^2 \rangle$

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 -CalculusDef.: Untyped λ -CalculusLet $\Sigma = (\mathcal{V}, \mathcal{C})$ be a signature where

- \mathcal{V} –countably infinite set of variables

Def.: Untyped λ -CalculusLet $\Sigma = (\mathcal{V}, \mathcal{C})$ 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 $\Lambda = \text{wff}_\Sigma(\Sigma)$ 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 $\Lambda = \text{wff}_\Sigma(\Sigma)$ to be the smallest set s.t:

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

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

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



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

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

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

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

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

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

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 $\Lambda^\alpha = \text{wff}_\Sigma(\Sigma)_\alpha = \bigcup_{\alpha \in \mathcal{T}} \Lambda_\alpha$ 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 $\Lambda^\alpha = \text{wff}_\Sigma(\Sigma)_\alpha = \bigcup_{\alpha \in \mathcal{T}} \Lambda_\alpha$ 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 $c_\alpha \in \Lambda_\alpha$
- $A_{\alpha\beta} \in \Lambda_{\alpha\beta}, B_\beta \in \Lambda_\beta$ then $(A B) \in \Lambda_\alpha$

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 $\Lambda^\alpha = \text{wff}_\Sigma(\Sigma)_\alpha = \bigcup_{\alpha \in \mathcal{T}} \Lambda_\alpha$ 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 $c_\alpha \in \Lambda_\alpha$
- $A_{\alpha\beta} \in \Lambda_{\alpha\beta}, B_\beta \in \Lambda_\beta$ then $(A B) \in \Lambda_\alpha$
- $x_\alpha \in \mathcal{V}_\alpha, A_\beta \in \Lambda_\beta$ then $(\lambda x_\alpha. A_\beta)_{\beta\alpha} \in \Lambda_{\beta\alpha}$

- brackets may be avoided: $A B C \rightsquigarrow ((A B) C)$

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

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

Notational Conventions



Notational Conventions



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

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

©Benzmüller, 2006



ATPHOL06-[3] – p.103

©Benzmüller, 2006



ATPHOL06-[3] – p.103

Notational Conventions



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

Def.: Positions in λ -Terms



Consider the following term:

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

©Benzmüller, 2006



ATPHOL06-[3] – p.103

©Benzmüller, 2006



ATPHOL06-[3] – p.103

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

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

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

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

... 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 $T := ((\lambda x.x)((\lambda y.\textcolor{red}{y})(\lambda z.z)))$

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

$A[B]_p$

Example:

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

Def.: Replacement at Position



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

$A[B]_p$

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

$[A/x] : \Lambda \rightarrow \Lambda$ (untyped)

Def.: Substitution



Substitution is a map

$[A/x] : \Lambda \rightarrow \Lambda$ (untyped)
 $[A_\alpha/x_\alpha] : \Lambda_\alpha \rightarrow \Lambda_\alpha$ (typed)

Def.: Substitution



Substitution is a map

$[A/x] : \Lambda \rightarrow \Lambda$ (untyped)
 $[A_\alpha/x_\alpha] : \Lambda_\alpha \rightarrow \Lambda_\alpha$ (typed)

and is defined as follows:

Def.: Substitution



Substitution is a map

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

and is defined as follows:

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

Def.: Substitution



Substitution is a map

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

and is defined as follows:

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

2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \wedge a_\beta \in V_\beta \cup C_\beta$

©Benzmüller, 2006



ATPHOL06-[4] – p.110

Def.: Substitution



Substitution is a map

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

and is defined as follows:

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

2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \wedge a_\beta \in V_\beta \cup C_\beta$

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

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

and is defined as follows:

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

2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \wedge a_\beta \in V_\beta \cup C_\beta$

3. $[N_\alpha/x_\alpha](A_{\alpha\alpha}B_\beta) = ([N_\alpha/x_\alpha]A)([N_\alpha/x_\alpha]B)$

4. $[N_\alpha/x_\alpha](\lambda x_\alpha.A_\gamma) = (\lambda x_\alpha A_\gamma)$

©Benzmüller, 2006



ATPHOL06-[4] – p.110

©Benzmüller, 2006



ATPHOL06-[4] – p.110

Def.: Substitution



Substitution is a map

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

and is defined as follows:

1. $[N_\alpha/x_\alpha]x_\alpha = N_\alpha$
2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \wedge a_\beta \in V_\beta \cup C_\beta$
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

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

and is defined as follows:

1. $[N_\alpha/x_\alpha]x_\alpha = N_\alpha$
2. $[N_\alpha/x_\alpha]a_\beta = a_\beta$ if $a_\beta \neq x_\alpha \wedge a_\beta \in V_\beta \cup C_\beta$
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
 $x_\alpha \neq y_\beta \wedge (y_\beta \notin FV(N_\alpha) \vee x_\alpha \notin FV(A_\gamma))$
6. $[N_\alpha/x_\alpha](\lambda y_\alpha.A_\gamma) = (\lambda z_\beta.[N_\alpha/x_\alpha][z_\beta/y_\beta]A_\gamma)$ if $x_\alpha \neq y_\beta \wedge$
 $(y_\beta \in FV(N_\alpha) \wedge x_\alpha \in FV(A_\gamma))$ 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 -convertible.

©Benzmüller, 2006



ATPHOL06-[4] – p.112

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 $[B/x]A$.

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

©Benzmüller, 2006



ATPHOL06-[4] – p.113

©Benzmüller, 2006



ATPHOL06-[4] – p.113

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



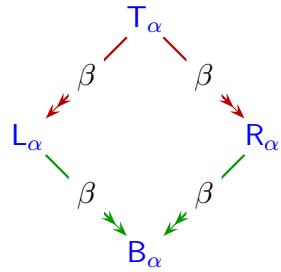
A term is called β -normal if it contains no β -redexes.

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

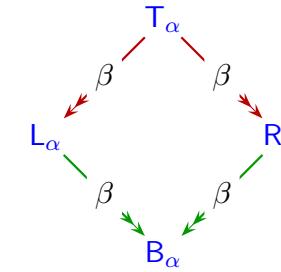
A term is called β -head normal if the head term of its outermost application can not be further reduced.

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

Thm.: Church-Rosser Property for \rightarrow_{β}

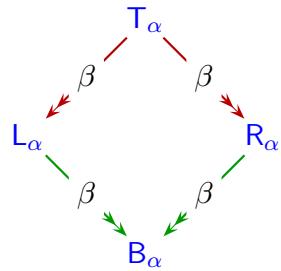


Thm.: Church-Rosser Property for \rightarrow_{β}



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 .

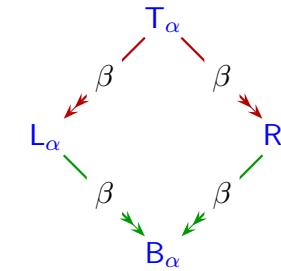
Thm.: Church-Rosser Property for \rightarrow_{β}



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 \rightarrow_{β}

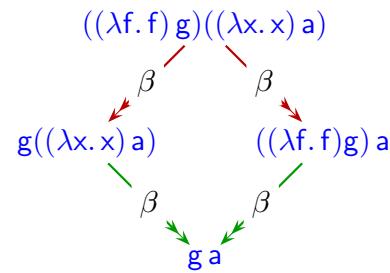


If T_α β -reduces in multiple steps with one strategy to L_α and with another strategy to R_α then there exists a term B_α such that L_α and R_α β -reduce in multiple steps to B_α .

Note that B_α is not necessarily in normal form.

The Church-Rosser Property for \rightarrow_{β} holds for Λ and Λ^α .

Ex.: Church-Rosser Property for \rightarrow_{β}



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: For all A_α there exists a unique (up to α -conversion) β -normal term B such that $A \rightarrow_{\beta} B$

Termination



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

Typed Case: For all A_α there exists a unique (up to α -conversion) β -normal term B such that $A \rightarrow_{\beta} B$

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

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

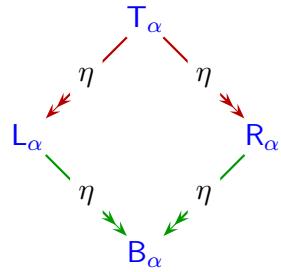
We say $M \rightarrowtail_\eta N$, ie. **η -reduces in several steps**, if $\exists M^1, \dots, M^n$ for $n \geq 1$ such that $M = M^1$ and $N = M^n$ and $M^i \rightarrow_\beta M^{i+1}$.

Def.: η -Normal Form

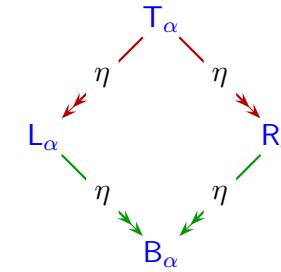


A term is called **η -normal** if it contains no η -redexes.

Thm.: Church-Rosser Property for \rightarrow_{η}

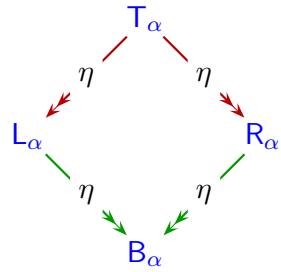


Thm.: Church-Rosser Property for \rightarrow_{η}



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 \rightarrow_{η}



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 \rightarrow_{η} holds for Λ and Λ^α .

Def.: $\beta\eta$ -Conversion



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

Def.: $\beta\eta$ -Conversion



Def.: $\beta\eta$ -Conversion



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

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

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

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

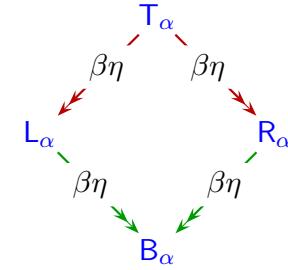
We say $M \twoheadrightarrow_{\beta\eta} N$, ie. η -reduces in several steps, if $\exists M^1, \dots, M^n$ for $n \geq 1$ such that $M = M^1$ and $N = M^n$ and $M^i \rightarrow_{\beta\eta} M^{i+1}$.

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

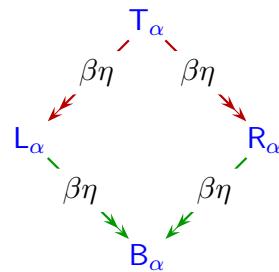


A term is $\beta\eta$ -normal if it contains no β -redexes and no η -redexes.

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

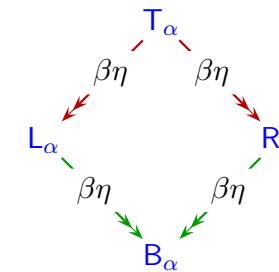


Thm.: Church-Rosser Property for $\rightarrow_{\beta\eta}$



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 $\rightarrow_{\beta\eta}$



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 $\rightarrow_{\beta\eta}$ holds for Λ and Λ^α .

Thm.: Strong Church-Rosser Property



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

Thm.: Strong Church-Rosser Property



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

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



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

$$\lambda x_{\alpha^1}^1 \dots x_{\alpha^n}^n. (h_{\beta\gamma^m \dots \gamma^1} A_{\gamma^1}^1 \dots A_{\gamma^m}^m)$$

for a variable or constant $h_{\beta\gamma^m \dots \gamma^1}$, $m \geq 0$ and long $\beta\eta$ -normal forms $A_{\gamma^1}^1, \dots, A_{\gamma^m}^m$.

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



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

$$\lambda x_{\alpha^1}^1 \dots x_{\alpha^n}^n. (h_{\beta\gamma^m \dots \gamma^1} A_{\gamma^1}^1 \dots A_{\gamma^m}^m)$$

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

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



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

$$\lambda x_{\alpha^1}^1 \dots x_{\alpha^n}^n. (h_{\beta\gamma^m \dots \gamma^1} A_{\gamma^1}^1 \dots A_{\gamma^m}^m)$$

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

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



Consider the $\beta\eta$ -normal term $f_{\iota(u)}$.

$$\begin{aligned} f_{\iota(u)} \\ \uparrow \eta \\ \lambda w_{\iota u}. (f_{\iota(u)} w_{\iota u}) \\ \uparrow \eta \\ \lambda w_{\iota u}. (f(\lambda x_{\iota}. w_{\iota u} x)) \end{aligned}$$

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

©Benzmüller, 2006



ATPHOL06-[4] – p.127



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 $A_{\gamma_i}^i$ to be in normal form.

Notation

ATPHOL06-[4] – p.128



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

©Benzmüller, 2006



ATPHOL06-[4] – p.128

©Benzmüller, 2006



ATPHOL06-[4] – p.129

Notation



Notation

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

- $A \downarrow_\beta$ 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 .
- $A \uparrow$ is the long $\beta\eta$ -normal form of A .



Semantics: Σ -Evaluations

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 ?

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 $C_i \neq \emptyset$ and $C_o \neq \emptyset$, 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 $C_i \neq \emptyset$ and $C_o \neq \emptyset$, then $\forall \alpha \in \mathcal{T}. \Lambda_\alpha \neq \emptyset$.

Ex.: An Interesting Applicative Structure



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

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

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 $C_i \neq \emptyset$ and $C_o \neq \emptyset$, then $\forall \alpha \in T. \Lambda_\alpha \neq \emptyset$.
- Is $D_{\alpha\beta}$ a set of functions? (ie. $D_{\alpha\beta} \subseteq (D_\alpha)^{D_\beta}$) —No!
- Is $(\lambda x_i x) \in D_{i,i}$? —Yes!
- $D = (D_\alpha)_{\alpha \in T}$ is not a frame!
- It requires a specific application operator $@ : D_{\alpha\beta} \times D_\beta \rightarrow D_\alpha$

Ex.: An Interesting Applicative Structure



©Benzmüller, 2006

ATPHOL06-[5] – p.131



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

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

Ex.: Interpretation of Terms



©Benzmüller, 2006

ATPHOL06-[5] – p.131

Syntax Semantics $\langle D, @ \rangle$
 $(\lambda x_i x)$

©Benzmüller, 2006



ATPHOL06-[5] – p.131

©Benzmüller, 2006



ATPHOL06-[5] – p.132

Ex.: Interpretation of Terms



Syntax Semantics $\langle D, @ \rangle$
 $(\lambda x_i.x)$ $(\lambda x_i.x)$

Ex.: Interpretation of Terms

Syntax Semantics $\langle D, @ \rangle$
 $(\lambda x_i.x)$ $(\lambda x_i.x) \in D_{\text{u}}$

Ex.: Interpretation of Terms



Syntax Semantics $\langle D, @ \rangle$
 $(\lambda x_i.x)$ $(\lambda x_i.x) \in D_{\text{u}}$
 y_i

Ex.: Interpretation of Terms

Syntax Semantics $\langle D, @ \rangle$
 $(\lambda x_i.x)$ $(\lambda x_i.x) \in D_{\text{u}}$
 y_i $\varphi(y)$

Ex.: Interpretation of Terms



Syntax	Semantics	$\langle D, @ \rangle$
$(\lambda x_t.x)$	$(\lambda x_t.x)$	$\in D_u$
y_t	$\varphi(y)$	$\in D_t$

Ex.: Interpretation of Terms



Syntax	Semantics	$\langle D, @ \rangle$
$(\lambda x_t.x)$	$(\lambda x_t.x)$	$\in D_u$
y_t	$\varphi(y)$	$\in D_t$
$a_t \in C$		

Ex.: Interpretation of Terms



Syntax	Semantics	$\langle D, @ \rangle$
$(\lambda x_t.x)$	$(\lambda x_t.x)$	$\in D_u$
y_t	$\varphi(y)$	$\in D_t$
$a_t \in C$	a	

Ex.: Interpretation of Terms



Syntax	Semantics	$\langle D, @ \rangle$
$(\lambda x_t.x)$	$(\lambda x_t.x)$	$\in D_u$
y_t	$\varphi(y)$	$\in D_t$
$a_t \in C$	a	$\in D_t$

Ex.: Interpretation of Terms



Syntax	Semantics	$\langle D, @ \rangle$
$(\lambda x_t. x)$	$(\lambda x_t. x)$	$\in D_u$
y_t	$\varphi(y)$	$\in D_t$
$a_t \in C$	a	$\in D_t$
$(\lambda x_t. x)a_t$		

Ex.: Interpretation of Terms



Syntax	Semantics	$\langle D, @ \rangle$
$(\lambda x_t. x)$	$(\lambda x_t. x)$	$\in D_u$
y_t	$\varphi(y)$	$\in D_t$
$a_t \in C$	a	$\in D_t$
$(\lambda x_t. x)a_t$	$(\lambda x_t. x)@a_t$	

Ex.: Interpretation of Terms



Syntax	Semantics	$\langle D, @ \rangle$
$(\lambda x_t. x)$	$(\lambda x_t. x)$	$\in D_u$
y_t	$\varphi(y)$	$\in D_t$
$a_t \in C$	a	$\in D_t$
$(\lambda x_t. x)a_t$	$(\lambda x_t. x)@a_t$	$\in D_t$

Ex.: Interpretation of Terms



Syntax	Semantics	$\langle D, @ \rangle$
$(\lambda x_t. x)$	$(\lambda x_t. x)$	$\in D_u$
y_t	$\varphi(y)$	$\in D_t$
$a_t \in C$	a	$\in D_t$
$(\lambda x_t. x)a_t$	$(\lambda x_t. x)@a_t$	$\in D_t$

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

Ex.: Interesting Applicative Structures



Ex.: Interesting Applicative Structures



- Let $D_\alpha \downarrow_\beta := \{A_\alpha \in \Lambda_\alpha \mid A \text{ is closed and } A \text{ is in } \beta\text{-normal form}\}$

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

Ex.: Interesting Applicative Structures



Ex.: Interesting Applicative Structures



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

$$F_{\gamma\delta} @_{\gamma\delta}^\beta G_\delta = (FG) \downarrow_\beta$$

for all $F_{\gamma\delta} \in D_{\gamma\delta}$ and $G_\delta \in D_\delta$.

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

$$F_{\gamma\delta} @_{\gamma\delta}^\beta G_\delta = (FG) \downarrow_\beta$$

for all $F_{\gamma\delta} \in D_{\gamma\delta}$ and $G_\delta \in D_\delta$.

$$@^\beta = (@_{\gamma\delta}^\beta)_{\gamma\delta \in T}$$

Ex.: Interesting Applicative Structures



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

$$F_{\gamma\delta} @^\beta G_\delta = (FG) \downarrow_\beta$$

for all $F_{\gamma\delta} \in D_{\gamma\delta}$ and $G_\delta \in D_\delta$.

- $@^\beta = (@^\beta_{\gamma\delta})_{\gamma\delta \in T}$

Claim: If $C_i \neq \emptyset$ and $C_o \neq \emptyset$ (i.e., at least one constant for each base type is given), then $(D, @^\beta)$ is an applicative structure.

Proof:

- Is $D_\alpha \downarrow_\beta$ nonempty for all $\alpha \in T$?

Ex.: Interesting Applicative Structures



Proof:

- Is $D_\alpha \downarrow_\beta$ nonempty for all $\alpha \in T$?
- Yes! This follows since $C_i \neq \emptyset$ and $C_o \neq \emptyset$.

Ex.: Interesting Applicative Structures



Proof:

- Is $D_\alpha \downarrow_\beta$ nonempty for all $\alpha \in T$?

Proof:

- Is $D_\alpha \downarrow_\beta$ nonempty for all $\alpha \in T$?
- Yes! This follows since $C_i \neq \emptyset$ and $C_o \neq \emptyset$.
- Is $F_{\gamma\delta} @^\beta G_\delta \in D_\gamma \downarrow_\beta$?

Ex.: Interesting Applicative Structures



Proof:

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

Ex.: Interesting Applicative Structures



- Let $D_\alpha \downarrow_{\beta\eta} := \{A_\alpha \in \Lambda_\alpha \mid A \text{ is closed and } A \text{ is in } \beta\eta\text{-normal form}\}$

Ex.: Interesting Applicative Structures



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

Ex.: Interesting Applicative Structures



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

$$F_{\gamma\delta} @_{\gamma\delta}^{\beta\eta} G_\delta = (FG) \downarrow$$

for all $F_{\gamma\delta} \in D_{\gamma\delta}$ and $G_\delta \in D_\delta$.

Ex.: Interesting Applicative Structures



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

$$F_{\gamma\delta} @_{\gamma\delta}^{\beta\eta} G_\delta = (FG) \downarrow$$

for all $F_{\gamma\delta} \in D_{\gamma\delta}$ and $G_\delta \in D_\delta$.

- $@^{\beta\eta} = (@_{\gamma\delta}^{\beta\eta})_{\gamma\delta \in T}$

Ex.: Interesting Applicative Structures



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

$$F_{\gamma\delta} @_{\gamma\delta}^{\beta\eta} G_\delta = (FG) \downarrow$$

for all $F_{\gamma\delta} \in D_{\gamma\delta}$ and $G_\delta \in D_\delta$.

- $@^{\beta\eta} = (@_{\gamma\delta}^{\beta\eta})_{\gamma\delta \in T}$

Claim: If $\mathcal{C}_i \neq \emptyset$ and $\mathcal{C}_o \neq \emptyset$ (i.e., at least one constant for each base type is given), then $(D, @^{\beta\eta})$ 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 $\varphi: \mathcal{V} \longrightarrow \mathcal{D} := (\varphi_\alpha: \mathcal{V}_\alpha \longrightarrow \mathcal{D}_\alpha)_{\alpha \in T}$ is called a **variable assignment** into \mathcal{A} .

Def.: Variable Assignment



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

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

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

Def.: Variable Assignment



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

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

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

Def.: Variable Assignment



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

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

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

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

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

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

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

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

and

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

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 . Furthermore, we assume there is a particular cardinal \aleph_s such that Σ_α has cardinality \aleph_s for every type α .

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 . Since \mathcal{V} is countable, this implies $wff_\alpha(\Sigma) := \Lambda^\alpha$ and $cwff_\alpha(\Sigma) := \{A \in \Lambda^\alpha \mid A \text{ closed}\}$ 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 $wff_\alpha(\Sigma) := \Lambda^\alpha$ and $cwff_\alpha(\Sigma) := \{A \in \Lambda^\alpha \mid A \text{ closed}\}$ 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 $wff_\alpha(\Sigma) := \Lambda^\alpha$ and $cwff_\alpha(\Sigma) := \{A \in \Lambda^\alpha \mid A \text{ closed}\}$ 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 $\mathcal{E}: \mathcal{F}_T(\mathcal{V}; \mathcal{D}) \longrightarrow \mathcal{F}_T(wff(\Sigma), \mathcal{D})$ be a total function, where $\mathcal{F}_T(\mathcal{V}; \mathcal{D})$ is the set of variable assignments and $\mathcal{F}_T(wff(\Sigma), \mathcal{D})$ is the set of typed functions mapping terms into objects in \mathcal{D} .

Σ -Evaluations



Let $\mathcal{E}: \mathcal{F}_T(\mathcal{V}; \mathcal{D}) \longrightarrow \mathcal{F}_T(wff(\Sigma), \mathcal{D})$ be a total function, where $\mathcal{F}_T(\mathcal{V}; \mathcal{D})$ is the set of variable assignments and $\mathcal{F}_T(wff(\Sigma), \mathcal{D})$ 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

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

Let $\mathcal{E}: \mathcal{F}_T(\mathcal{V}; \mathcal{D}) \longrightarrow \mathcal{F}_T(wff(\Sigma), \mathcal{D})$ be a total function, where $\mathcal{F}_T(\mathcal{V}; \mathcal{D})$ is the set of variable assignments and $\mathcal{F}_T(wff(\Sigma), \mathcal{D})$ 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

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

What properties shall \mathcal{E} fulfill?

Def.: Evaluation Function

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

\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

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

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

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}_α 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}_α 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}_α 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 $\text{Free}(\mathbf{A}) = \emptyset$. 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.

©Benzmüller, 2006



ATPHOL06-[5] – p.144

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 $(\lambda X, x)$.

©Benzmüller, 2006



ATPHOL06-[5] – p.144

©Benzmüller, 2006



ATPHOL06-[5] – p.144

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

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

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 $A \in wff_\alpha(\Sigma)$ and $B \in wff_\beta(\Sigma)$, 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 $A \in wff_\alpha(\Sigma)$ and $B \in wff_\beta(\Sigma)$, we have

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

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

©Benzmüller, 2006



ATPHOL06-[5] – p.147

Prf.: Substitution-Value Lemma



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

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

©Benzmüller, 2006



ATPHOL06-[5] – p.147

Prf.: Substitution-Value Lemma



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

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

©Benzmüller, 2006



ATPHOL06-[5] – p.147

Prf.: Substitution-Value Lemma



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

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

©Benzmüller, 2006



ATPHOL06-[5] – p.147

Prf.: Substitution-Value Lemma



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

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

.

Prf.: Substitution-Value Lemma



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

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

.

Prf.: Substitution-Value Lemma



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

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

Weaker Notions of Functionality



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

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.

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

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

for any type α , formula $\mathbf{A} \in wff_\alpha(\Sigma)$, 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



Def.: ξ -Functional



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

$$\mathcal{E}_{\varphi}(\lambda X_{\alpha}. M_{\beta}) = \mathcal{E}_{\varphi}(\lambda X_{\alpha}. N_{\beta})$$

Def.: ξ -Functional



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

$$\mathcal{E}_{\varphi}(\lambda X_{\alpha}. M_{\beta}) = \mathcal{E}_{\varphi}(\lambda X_{\alpha}. N_{\beta})$$

whenever

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

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

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 $F \in wff_{\alpha \rightarrow \beta}(\Sigma)$ where $X_{\alpha} \notin \text{Free}(F)$, 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 $\mathbf{F} \in \text{wff}_{\alpha \rightarrow \beta}(\Sigma)$ where $X_\alpha \notin \text{Free}(\mathbf{F})$, we have

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

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 $\mathbf{F} \in \text{wff}_{\alpha \rightarrow \beta}(\Sigma)$ where $X_\alpha \notin \text{Free}(\mathbf{F})$, we have

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



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

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

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

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

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



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

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

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

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 $\eta+\xi$



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

Lemma: Functionality and $\eta+\xi$



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 $\eta+\xi$



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:

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

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:

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

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

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:

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

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

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

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:

$\top_o, \perp_o, \neg_{oo}, \vee_{ooo}, \wedge_{ooo}, \exists_{ooo}, \Leftrightarrow_{ooo}$

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

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

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

Once More: Cantor's Theorem

For any set A ,

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

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

Once More: Cantor's Theorem

For any set A ,

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

Once More: Cantor's Theorem

Assume the set A is associated with ν .

Once More: Cantor's Theorem



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

D_{o_ι}

Once More: Cantor's Theorem



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

$$D_{\text{o}_\iota} = D_{\text{o}}^{D_\iota}$$

Once More: Cantor's Theorem



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

$$\begin{aligned} D_{\text{o}_\iota} &= D_{\text{o}}^{D_\iota} \\ &= \{\perp, \top\}^{D_\iota} \end{aligned}$$

Once More: Cantor's Theorem



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

$$\begin{aligned} D_{o_\iota} &= D_o^{D_\iota} \\ &= \{\perp, \top\}^{D_\iota} \\ &= \{f \mid f : D_\iota \rightarrow \{\perp, \top\}\} \end{aligned}$$

Once More: Cantor's Theorem



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

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

Once More: Cantor's Theorem



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

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

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):

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

Once More: Cantor's Theorem



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

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

which is shorthand for:

Once More: Cantor's Theorem



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

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

which is shorthand for:

$$\neg_{oo} \Sigma_{o(o(o\omega))}^{o\omega} \left(\lambda g_{o\omega}. \Pi_{o(o(o\iota))}^{o\iota} \left(\lambda f_{o\iota}. \Sigma_{o(o\iota)}^\iota \left(\lambda x_\iota. =_{o(o\iota)(o\iota)}^{o\iota} (gx) f \right) \right) \right)$$

Once More: Cantor's Theorem



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

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

which is shorthand for:

$$\neg_{oo} \Sigma_{o(o(o\omega))}^{o\omega} \left(\lambda g_{o\omega}. \Pi_{o(o(o\iota))}^{o\iota} \left(\lambda f_{o\iota}. \Sigma_{o(o\iota)}^\iota \left(\lambda x_\iota. =_{o(o\iota)(o\iota)}^{o\iota} (gx) f \right) \right) \right)$$

Note: for this term to be in the set $cwff_\alpha(\Sigma)$, the constants \neg_{oo} , $\Sigma_{o(o(o\omega))}^{o\omega}$, $\Pi_{o(o(o\iota))}^{o\iota}$, Σ^ι and $=_{o(o\iota)(o\iota)}^{o\iota}$ 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 $f = (\lambda x_t. \neg g x x)$.

Once More: Cantor's Theorem



Proof: Assume such a function g exists.

Let $f = \{x \mid x \notin gx\}$ that is $f = (\lambda x_t. \neg g x x)$.
 g is surjective,

Once More: Cantor's Theorem



Proof: Assume such a function g exists.

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

g is surjective, hence

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

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[5] – p.157

Once More: Cantor's Theorem



Proof: Assume such a function g exists.

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

g is surjective, hence

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

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 $f = (\lambda x_t. \neg gxx)$.

g is surjective, hence

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

hence

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

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[5] – p.157

Once More: Cantor's Theorem



Once More: Cantor's Theorem



Proof: Assume such a function g exists.

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

g is surjective, hence

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

hence

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

Contradiction!

Note that the proof uses \neg .

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[5] – p.157

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[5] – p.157



Semantics: Σ -Models

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants



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

Def.: Properties of Logical Constants

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

Def.: Properties of Logical Constants

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

Def.: Properties of Logical Constants

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

Def.: Properties of Logical Constants

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

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a @ b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a @ b @ c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a @ b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a @ b @ c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge		

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a @ b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a @ b @ c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a @ b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a @ b @ c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = F \text{ or } v(c) = T \quad \forall b, c \in D_o$

Def.: Properties of Logical Constants

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

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

Def.: Properties of Logical Constants

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

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

Def.: Properties of Logical Constants

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

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

Def.: Properties of Logical Constants

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

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

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = F \text{ or } v(c) = T \quad \forall b, c \in D_o$
\leftrightarrow	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = v(c) \quad \forall b, c \in D_o$
$=^\alpha$	$o\alpha\alpha$	

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = F \text{ or } v(c) = T \quad \forall b, c \in D_o$
\leftrightarrow	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = v(c) \quad \forall b, c \in D_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = T \quad \text{iff } b = c \quad \forall b, c \in D_o$
Π^α		

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = F \text{ or } v(c) = T \quad \forall b, c \in D_o$
\leftrightarrow	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = v(c) \quad \forall b, c \in D_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = T \quad \text{iff } b = c \quad \forall b, c \in D_o$

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = F \text{ or } v(c) = T \quad \forall b, c \in D_o$
\leftrightarrow	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = v(c) \quad \forall b, c \in D_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = T \quad \text{iff } b = c \quad \forall b, c \in D_o$
Π^α	$o(\alpha)$	

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = F \text{ or } v(c) = T \quad \forall b, c \in D_o$
\Leftrightarrow	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = v(c) \quad \forall b, c \in D_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = T \quad \text{iff } b = c \quad \forall b, c \in D_o$
Π^α	$o(o\alpha)$	$v(a@f) = T \quad \text{iff } \forall b \in D_\alpha : v(f@b) = T \quad \forall f \in D_{o\alpha}$
Σ^α	$o(o\alpha)$	

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = F \text{ or } v(c) = T \quad \forall b, c \in D_o$
\Leftrightarrow	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = v(c) \quad \forall b, c \in D_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = T \quad \text{iff } b = c \quad \forall b, c \in D_o$
Π^α	$o(o\alpha)$	$v(a@f) = T \quad \text{iff } \forall b \in D_\alpha : v(f@b) = T \quad \forall f \in D_{o\alpha}$
Σ^α	$o(o\alpha)$	

Def.: Properties of Logical Constants

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

c	β	$\mathcal{L}_c((a))$ holds when
\top	o	$v(a) = T$
\perp	o	$v(a) = F$
\neg	oo	$v(a@b) = T \quad \text{iff } v(b) = F \quad \forall b \in D_o$
\vee	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ or } v(c) = T \quad \forall b, c \in D_o$
\wedge	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = T \text{ and } v(c) = T \quad \forall b, c \in D_o$
\supset	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = F \text{ or } v(c) = T \quad \forall b, c \in D_o$
\Leftrightarrow	ooo	$v(a@b@c) = T \quad \text{iff } v(b) = v(c) \quad \forall b, c \in D_o$
$=^\alpha$	$o\alpha\alpha$	$v(a@b@c) = T \quad \text{iff } b = c \quad \forall b, c \in D_o$
Π^α	$o(o\alpha)$	$v(a@f) = T \quad \text{iff } \forall b \in D_\alpha : v(f@b) = T \quad \forall f \in D_{o\alpha}$
Σ^α	$o(o\alpha)$	

Def.: Σ -Valuation



Let $\mathcal{J} := (D, @, \mathcal{E})$ be a Σ -evaluation and $v : D_o \rightarrow \{T, F\}$.

Def.: Σ -Valuation



Let $\mathcal{J} := (D, @, \mathcal{E})$ be a Σ -evaluation and $v : D_o \rightarrow \{T, F\}$. We say v is a Σ -valuation w.r.t \mathcal{J} if



Def.: Σ -Valuation



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

Def.: Σ -Model



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



Def.: Σ -Model



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

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

Def.: Σ -Model



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

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

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

Def.: Σ -Model



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

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

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

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

Def.: Σ -Model



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

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

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

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

If $(D, @, \mathcal{E})$ 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**: $A_\alpha \doteq^\alpha B_\alpha := \forall P_{\alpha\alpha^*}(PA) \Rightarrow (PB)$

Some Conventions: Equality



Some important conventions:

- = denotes **primitive equality**
- \doteq denotes **Leibniz equality**: $A_\alpha \doteq^\alpha B_\alpha := \forall P_{\alpha\alpha^*}(PA) \Rightarrow (PB)$
- \equiv ... other definition of equality (e.g., see [Andrews02])

Some Conventions: Equality



Some important conventions:

- = denotes **primitive equality**
- \doteq denotes **Leibniz equality**: $A_\alpha \doteq^\alpha B_\alpha := \forall P_{\alpha\alpha^*}(PA) \Rightarrow (PB)$
- \equiv ... other definition of equality (e.g., see [Andrews02])

We use \equiv^* in the following to refer to **any** of the above

Def.: Properties f, b, η, ξ



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

Def.: Properties f, b, η, ξ



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

η if M is η -functional (respectively $(D, @, E)$ is η -functional)

Def.: Properties f, b, η, ξ



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

- η if M is η -functional (respectively $(D, @, E)$ is η -functional)
- ξ if M is ξ -functional (respectively $(D, @, E)$ is ξ -functional)

Def.: Properties f, b, η, ξ



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

- η if M is η -functional (respectively $(D, @, E)$ is η -functional)
- ξ if M is ξ -functional (respectively $(D, @, E)$ is ξ -functional)
- f if M is functional (respectively $(D, @, E)$ is functional)

Def.: Properties f, b, η, ξ



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

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

Def.: Properties f, b, η, ξ



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

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

Note: In the [JSC04]-paper, b is defined as $D_o = \{\top, \perp\}$, 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 evaluate via \mathbf{v} to \top . Another ill case would be D_o with just one element.

Def.: Properties f, b, η, ξ



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

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

Def.: Properties f, b, η, ξ



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

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

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 $\mathcal{L}_{=^\alpha}(\mathcal{E}_\varphi(=_{o\alpha\alpha}))$, 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 $\mathcal{M} = (D, @, \mathcal{E}, v)$ be a \mathcal{C} -model.

Lemma: Surjective v



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

©Benzmüller, 2006



ATPHOL06-[6] – p.164

Lemma: Surjective v



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

Proof: Exercise.

©Benzmüller, 2006



ATPHOL06-[6] – p.164

Thm.: Property b



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

©Benzmüller, 2006



ATPHOL06-[6] – p.165

Thm.: Property b



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

Thm.: Property b



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

Thm.: Property b



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

Proof: Exercise.



Semantics: HOL-CUBE

Def. (Reminder): Σ -Model



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

Def. (Reminder): Σ -Model



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

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

Def. (Reminder): Σ -Model



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

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

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

Def. (Reminder): Σ -Model



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

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

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

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

Def. (Reminder): Σ -Model



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

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

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

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

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

Def. (Reminder): Properties f, b, η, ξ



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

Def. (Reminder): Properties f, b, η, ξ



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

Def. (Reminder): Properties f, b, η, ξ



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

Def. (Reminder): Properties f, b, η, ξ



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

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

Def. (Reminder): Properties f, b, η, ξ



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

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

Def. (Reminder): Properties f, b, η, ξ



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

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

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 ξ, η, f, 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 ξ, η, f, 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 ξ, η, f, 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 ξ, η, f, 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 ξ, η, f, b . Thus we obtain

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

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 ξ, η, f, b . Thus we obtain

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

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 ξ, η, f, b . Thus we obtain

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

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 ξ, η, f, b . Thus we obtain

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

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 ξ, η, f, b . Thus we obtain

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

Def.: Satisfies, models, and \models



Let $\mathcal{M} = (D, @, \mathcal{E}, v)$ be a Σ -model and let φ be an assignment into \mathcal{M} .

We say φ satisfies a formula $A \in wff_o(\Sigma)$ in \mathcal{M} (we write $\mathcal{M} \models_\varphi A$) if $v(\mathcal{E}_\varphi(A)) = T$.

Def.: Satisfies, models, and \models



Let $\mathcal{M} = (D, @, \mathcal{E}, v)$ be a Σ -model and let φ be an assignment into \mathcal{M} .

Def.: Satisfies, models, and \models



Let $\mathcal{M} = (D, @, \mathcal{E}, v)$ be a Σ -model and let φ be an assignment into \mathcal{M} .

We say φ satisfies a formula $A \in wff_o(\Sigma)$ in \mathcal{M} (we write $\mathcal{M} \models_\varphi A$) if $v(\mathcal{E}_\varphi(A)) = T$.

We say that A is valid in \mathcal{M} (and write $\mathcal{M} \models A$) if $\mathcal{M} \models_\varphi A$ for all assignments φ .

Let $\mathcal{M} = (D, @, \mathcal{E}, v)$ be a Σ -model and let φ be an assignment into \mathcal{M} .

We say φ satisfies a formula $A \in wff_o(\Sigma)$ in \mathcal{M} (we write $\mathcal{M} \models_{\varphi} A$) if $v(\mathcal{E}_{\varphi}(A)) = T$.

We say that A is valid in \mathcal{M} (and write $\mathcal{M} \models A$) if $\mathcal{M} \models_{\varphi} A$ for all assignments φ . When $A \in cwff_o(\Sigma)$, we drop the reference to the assignment and use the notation $\mathcal{M} \models A$.

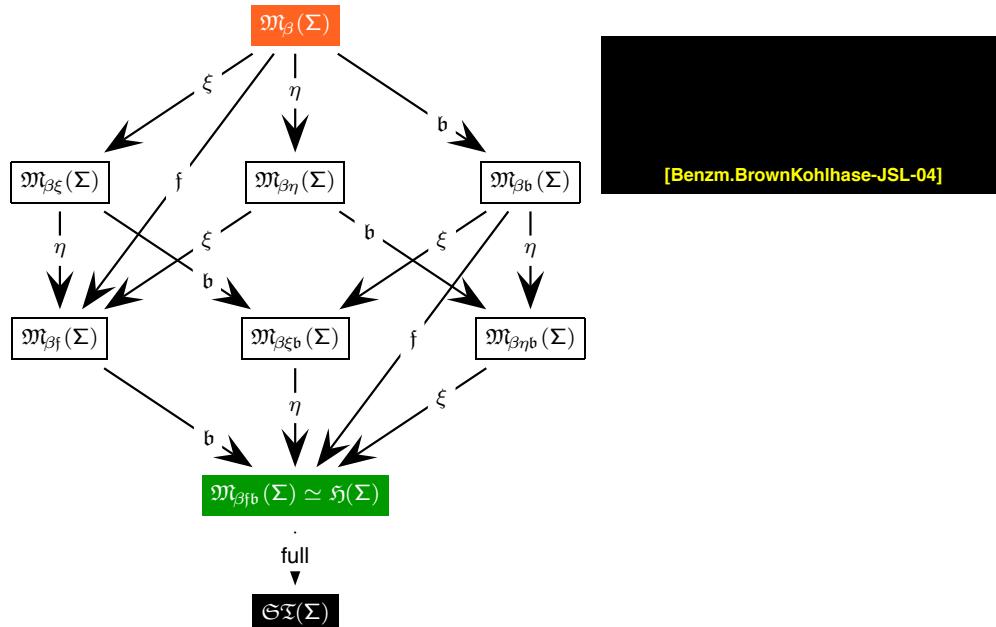
Let $\mathcal{M} = (D, @, \mathcal{E}, v)$ be a Σ -model and let φ be an assignment into \mathcal{M} .

We say φ satisfies a formula $A \in wff_o(\Sigma)$ in \mathcal{M} (we write $\mathcal{M} \models_{\varphi} A$) if $v(\mathcal{E}_{\varphi}(A)) = T$.

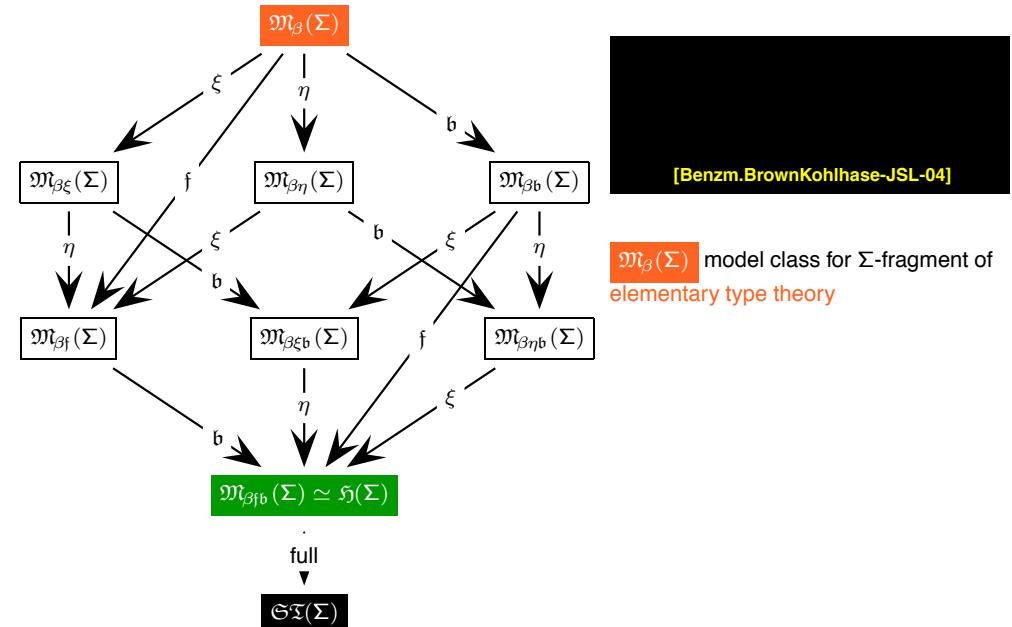
We say that A is valid in \mathcal{M} (and write $\mathcal{M} \models A$) if $\mathcal{M} \models_{\varphi} A$ for all assignments φ . When $A \in cwff_o(\Sigma)$, we drop the reference to the assignment and use the notation $\mathcal{M} \models A$.

Finally, we say that \mathcal{M} is a Σ -model for a set $\Phi \subseteq cwff_o(\Sigma)$ (we write $\mathcal{M} \models \Phi$) if $\mathcal{M} \models A$ for all $A \in \Phi$.

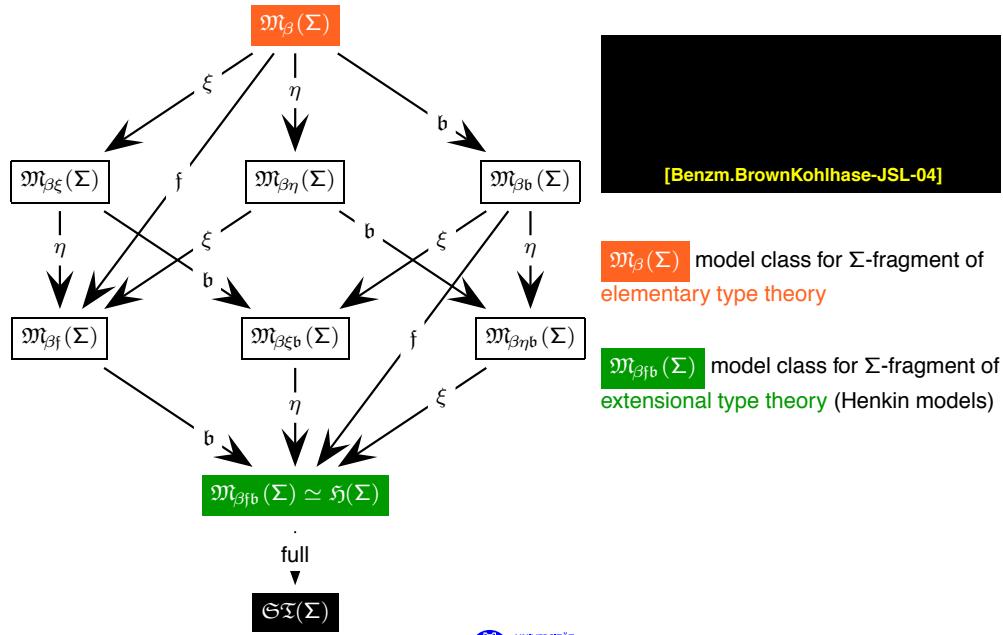
Semantics: HOL-CUBE



Semantics: HOL-CUBE

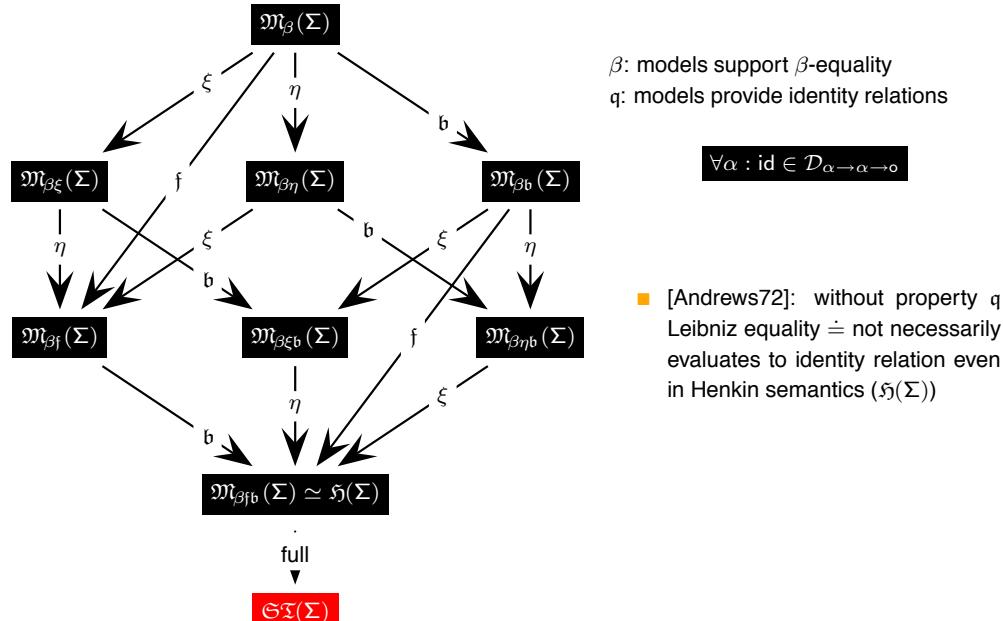


Semantics: HOL-CUBE



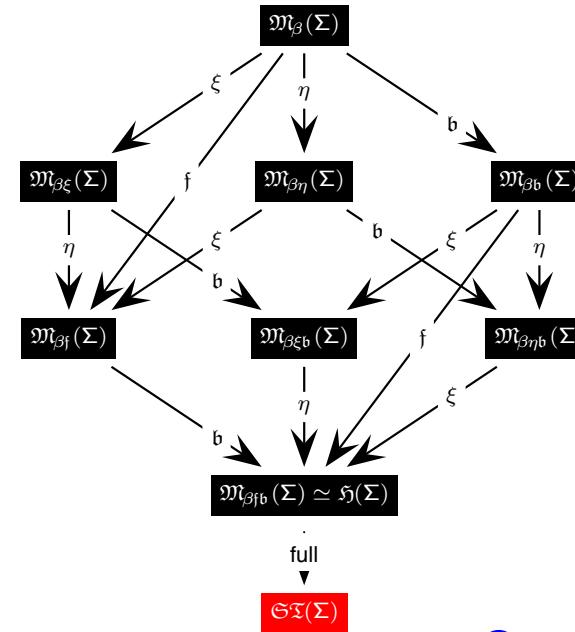
©Benzmüller, 2006

Semantics: HOL-CUBE



©Benzmüller, 2006

Semantics: HOL-CUBE



©Benzmüller, 2006

Standard Models and Henkin Models

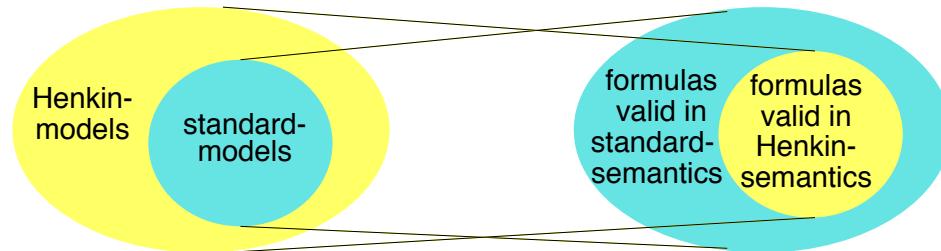
Leon Henkin generalized the class of admissible domains for functional types.

- [Andrews72]: without property η Leibniz equality \doteq not necessarily evaluates to identity relation even in Henkin semantics ($\mathfrak{H}(\Sigma)$)

©Benzmüller, 2006

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) 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).



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) 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).

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].

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].

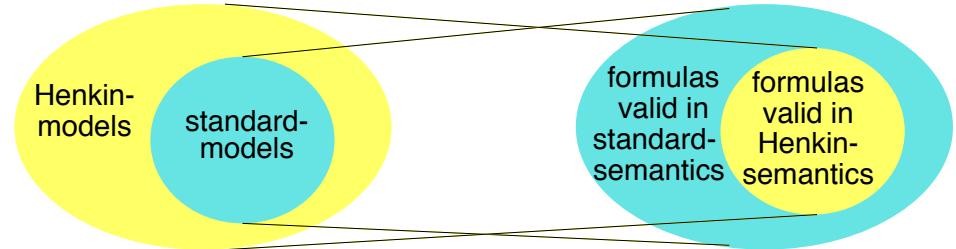
Note that even though we can consider model classes with richer and richer function spaces, **we can never reach standard models where function spaces are full while maintaining complete (recursively axiomatizable) calculi**.

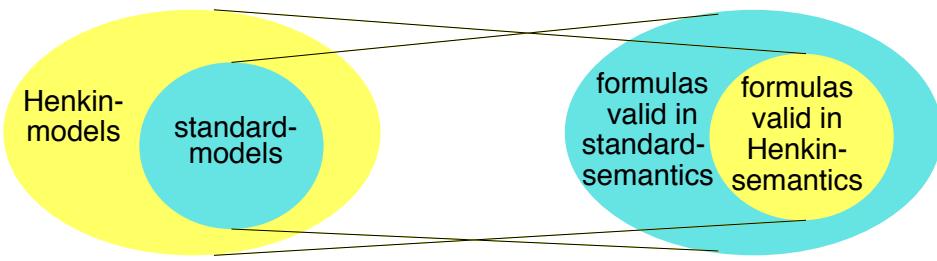
The generalization to Henkin models restricts the set of valid formulae sufficiently so that **all of them can be proven** by a Hilbert-style calculus [Henkin50].

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





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.
- $I := \lambda X.X$ and $L := \lambda X.\text{rev}(\text{rev}(X))$, 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.
- $I := \lambda X.X$ and $L := \lambda X.\text{rev}(\text{rev}(X))$, where rev is the self-inverse function.
- The identity function has constant complexity, the function rev is linear in the length of its argument.

How do we account for models without functional extensionality?

- We have generalized the notion of domains at function types and evaluation functions.

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.

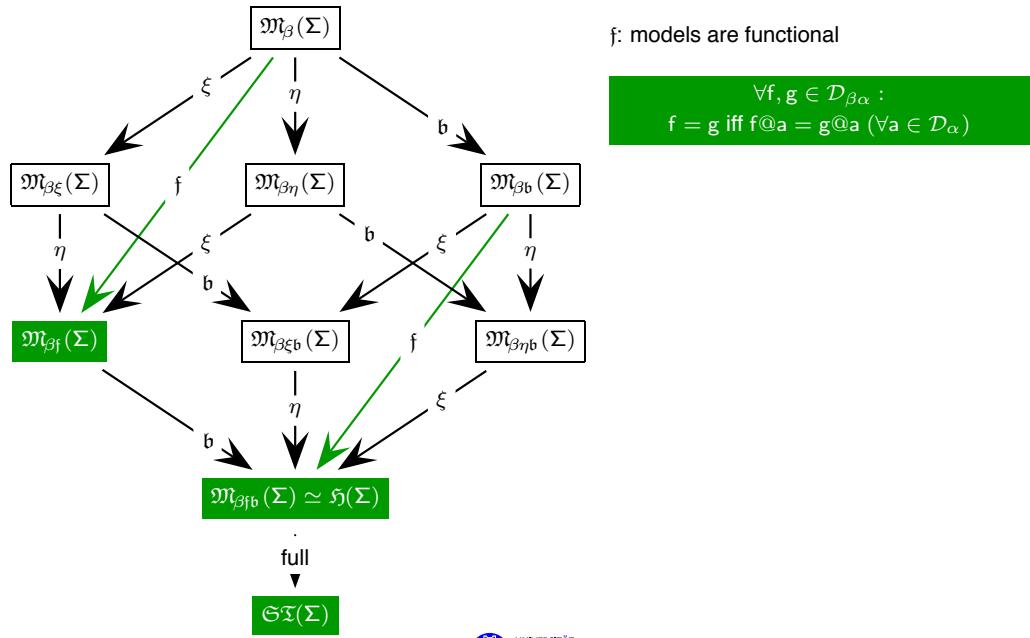
Models without Functional Extensionality

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.

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.



Models without η - or ξ -Functionality

Motivation: in standard literature functional extensionality is often discussed in terms of

- ### ■ ξ -functionality

Models without η - or ξ -Functionality

Motivation: in standard literature functional extensionality is often discussed in terms of

Models without η - or ξ -Functionality

Motivation: in standard literature functional extensionality is often discussed in terms of

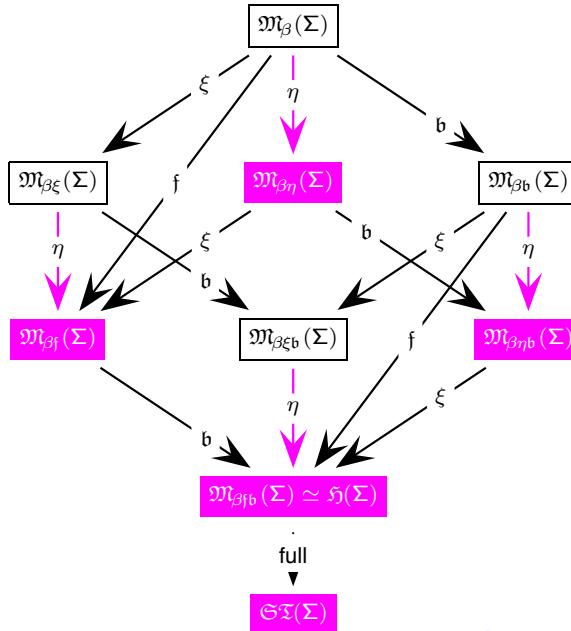
- ξ -functionality
 - η -functionality

Models without η - or ξ -Functionality

Motivation: in standard literature functional extensionality is often discussed in terms of

- ξ -functionality
- η -functionality
- Therefore, we integrated these two cases in our landscape.

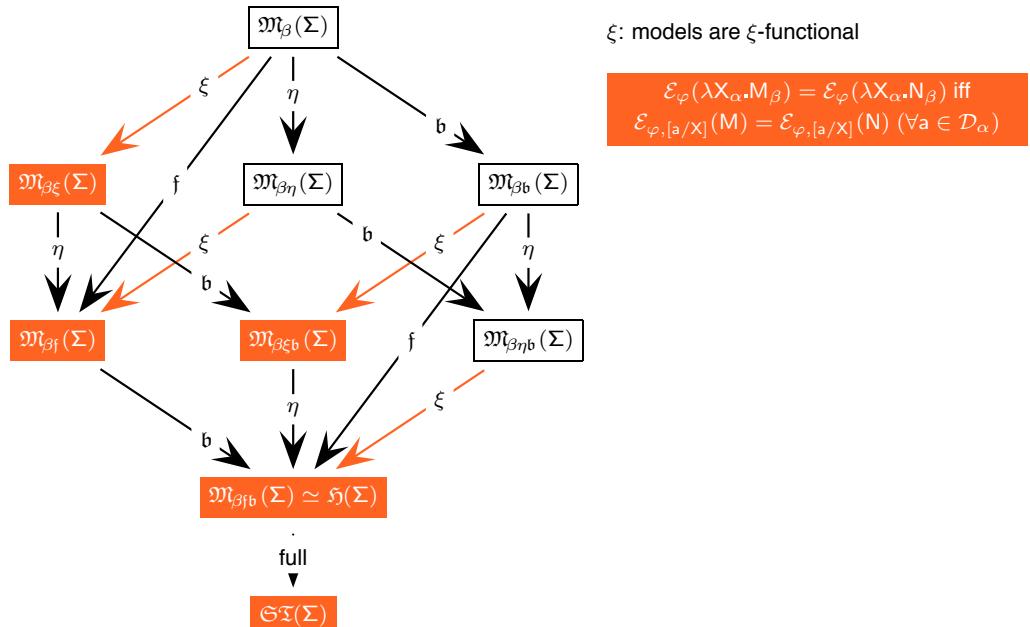
Semantics: HOL-CUBE



η : models are η -functional

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

Semantics: HOL-CUBE



ξ : models are ξ -functional

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

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” 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”

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: obvious(O) and obvious(F) where $O := 2 + 2 = 4$ and $F := \forall n > 2x^n + y^n = z^n \Rightarrow x = y = z = 0$ should not be equivalent, even if their arguments are.

©Benzmüller, 2006



ATPHOL06-[7] – p.183

Models without Boolean Extensionality



How do we account for models without Boolean extensionality?

- We have weakened the assumption that $\mathcal{D}_o = \{\text{T}, \text{F}\}$, since this entails that the values of O and F are identical.

©Benzmüller, 2006



ATPHOL06-[7] – p.184

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: obvious(O) and obvious(F) where $O := 2 + 2 = 4$ and $F := \forall n > 2x^n + y^n = z^n \Rightarrow x = y = z = 0$ should not be equivalent, even if their arguments are.
- Such phenomena have been studied under the heading of “hyper-intensional semantics” in theoretical semantics.

©Benzmüller, 2006



ATPHOL06-[7] – p.183

Models without Boolean Extensionality



How do we account for models without Boolean extensionality?

- We have weakened the assumption that $\mathcal{D}_o = \{\text{T}, \text{F}\}$, since this entails that the values of O and F are identical.
- In our Σ -models without property 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 v of \mathcal{D}_o , i.e., a function $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ that is coordinated with the interpretations of the logical constants \neg , \vee , and Π^α (for each type α).

©Benzmüller, 2006



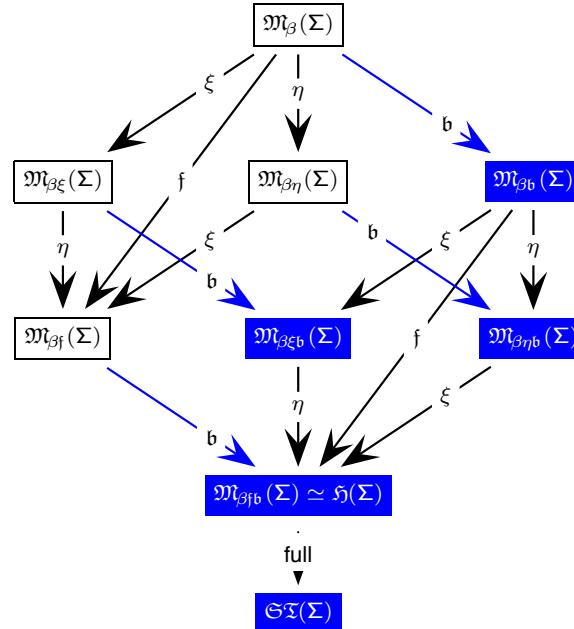
ATPHOL06-[7] – p.184

Models without Boolean Extensionality

How do we account for models without Boolean extensionality?

- We have weakened the assumption that $\mathcal{D}_o = \{\text{T}, \text{F}\}$, since this entails that the values of **O** and **F** are identical.
- In our **Σ -models** without property **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 **v** of \mathcal{D}_o , i.e., a function $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ 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 **A** valid in such a model if $v(a) = \text{T}$, where $a \in \mathcal{D}_o$ is the denotation of the sentence **A**.

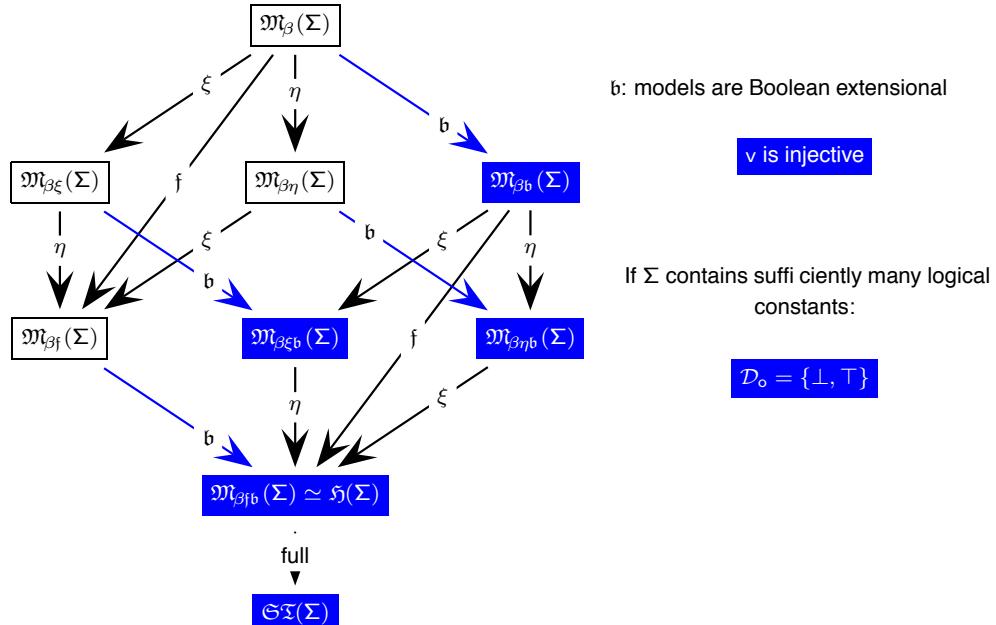
Semantics: HOL-CUBE



b: models are Boolean extensional

v is injective

Semantics: HOL-CUBE



b: models are Boolean extensional

v is injective

If Σ contains sufficiently many logical constants:

$$\mathcal{D}_o = \{\perp, \top\}$$



Semantics and Theorem
Proving: Test Problems for
Theorem Provers

- Test problems for FOL 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 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 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 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 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 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!!!
- Examples are simple

- Examples are simple
 - ▶ highlight the essence of some semantical or technical point

- Examples are simple
 - ▶ highlight the essence of some semantical or technical point
 - ▶ easy to understand and easy to encode

- 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 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

- 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]

- 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

- 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

Remark: Signature

Unless stated otherwise we assume on the following slides that our signature Σ contains the following logical connectives:

$$\{\top, \perp, \neg, \wedge, \vee, \supset, \Leftrightarrow\} \cup \{\Pi^\alpha, \Sigma^\alpha, =^\alpha\}$$

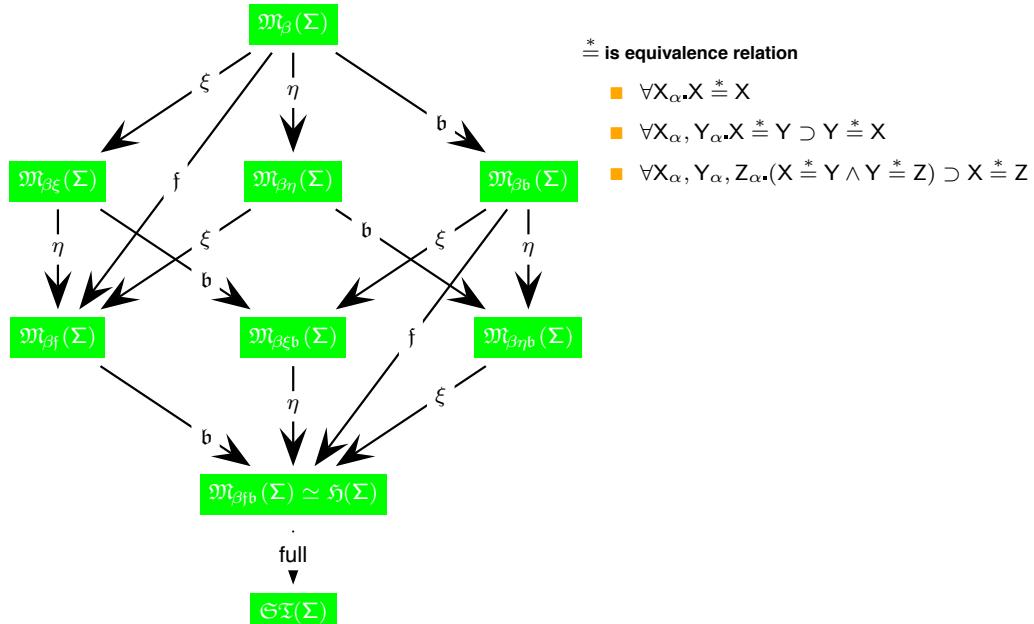
(less logical connectives are possible)

- 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

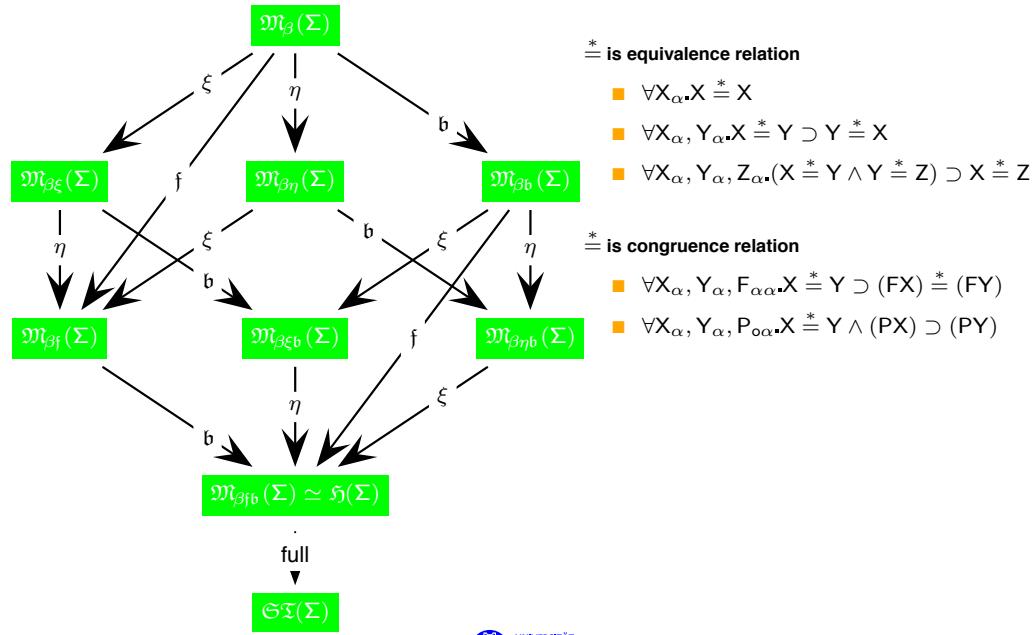
- Examples are structured
 - ▶ quick indicators for completeness and soundness wrt to HOL model classes from [Benzm.BrownKohlhase-JSL-04]
 - ▶ shall precede formal soundness / completeness analysis
 - ▶ many are collected from experience with LEO and TPS

- (Some more challenging examples are also added in [TPHOLS-05])

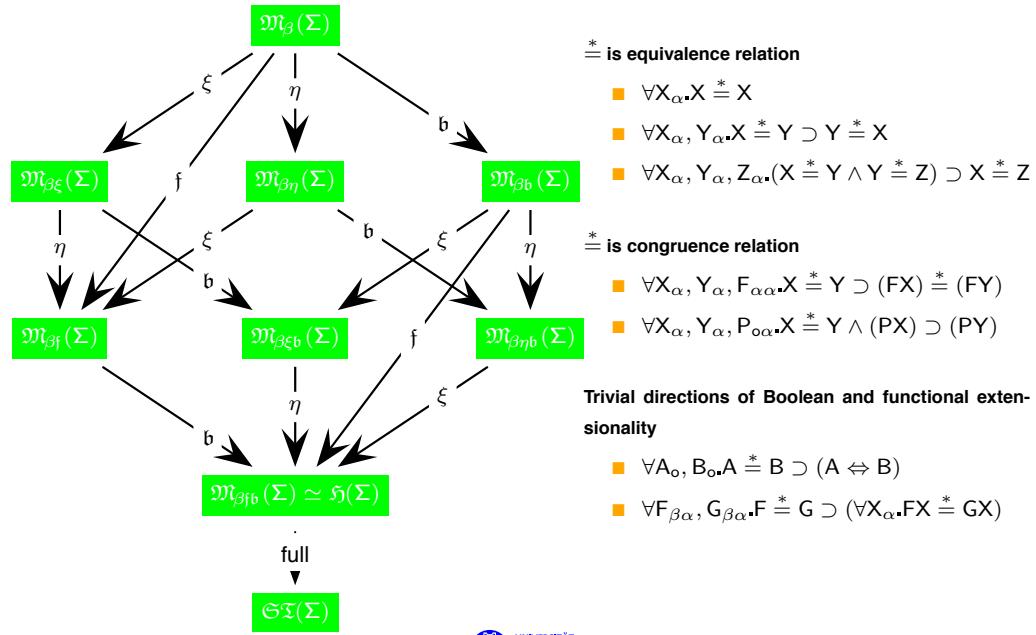
HOL-Problems: β



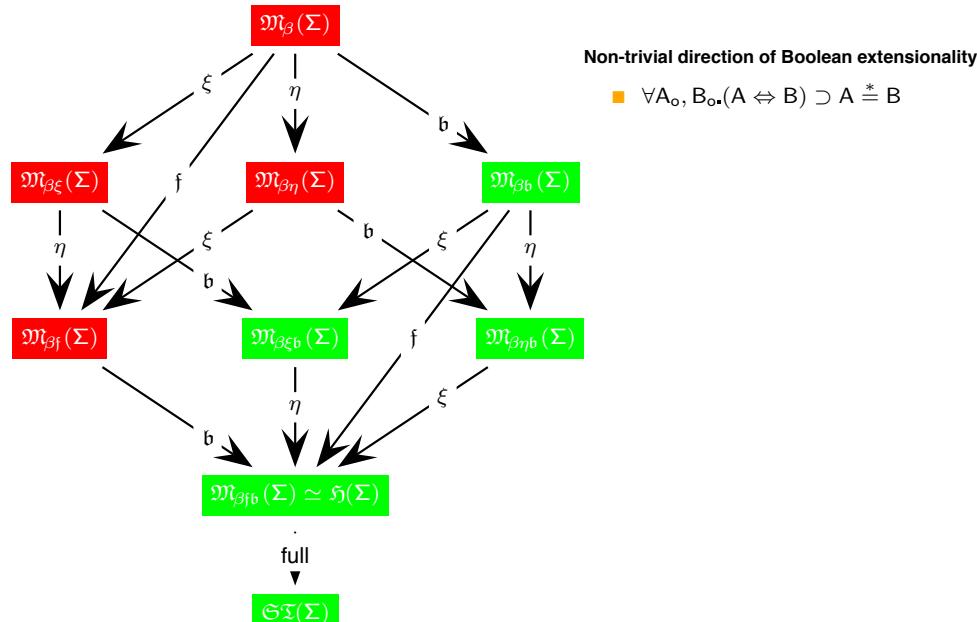
HOL-Problems: β



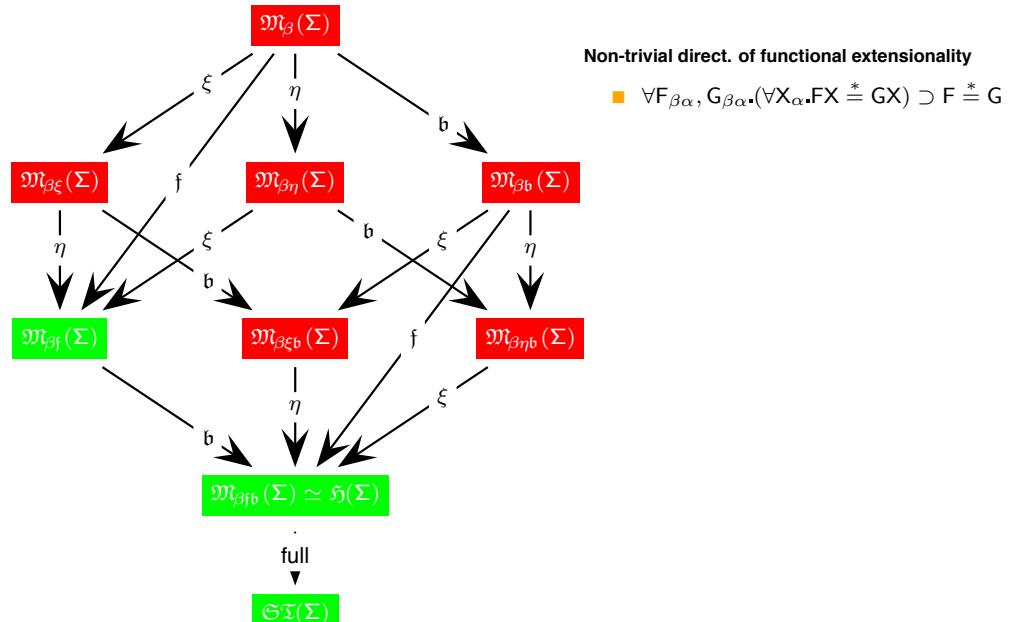
HOL-Problems: β



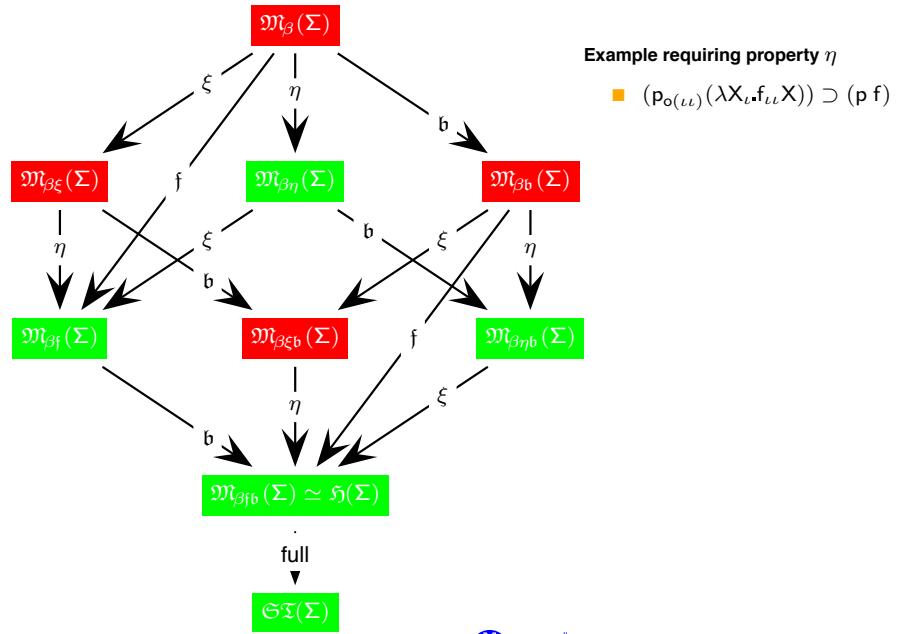
HOL-Problems: b



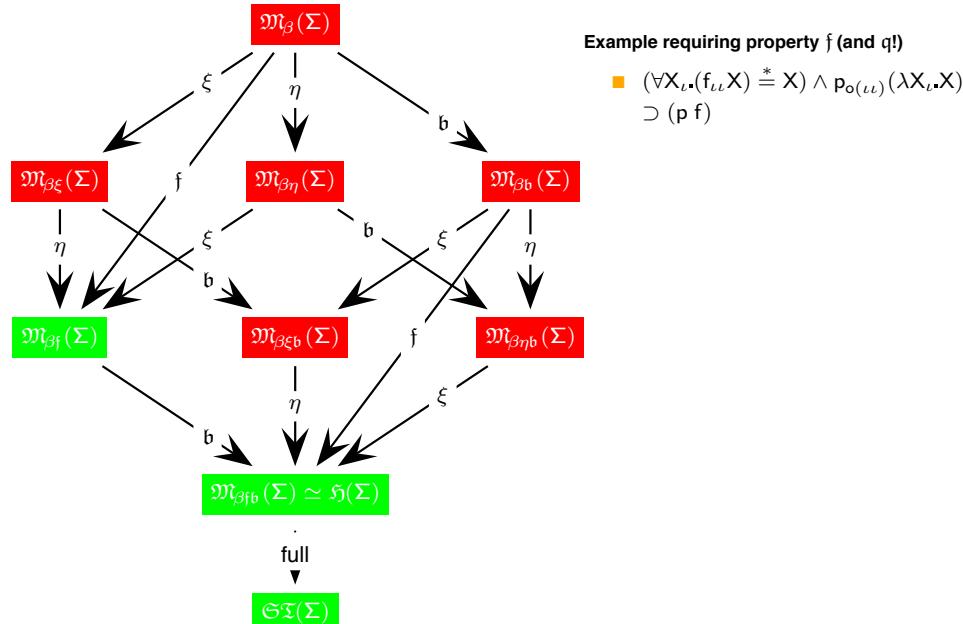
HOL-Problems: f



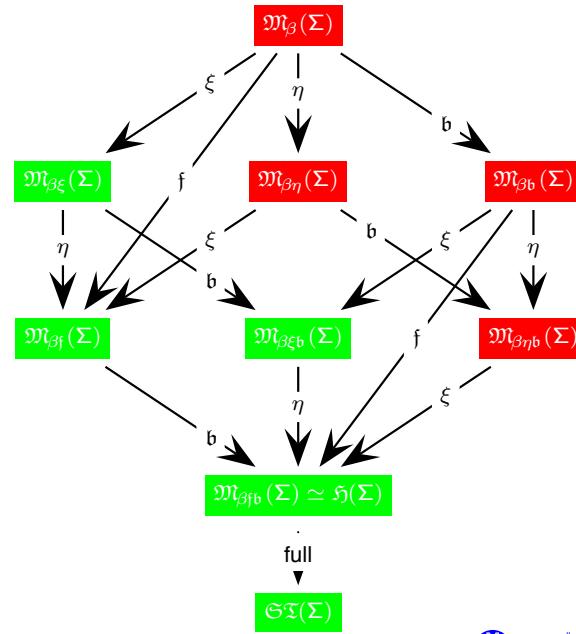
HOL-Problems: η



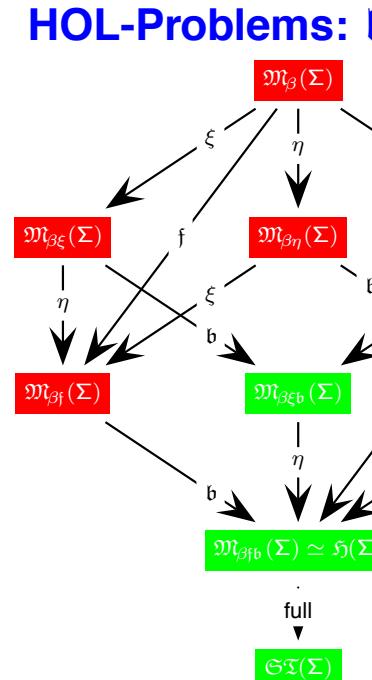
HOL-Problems: f



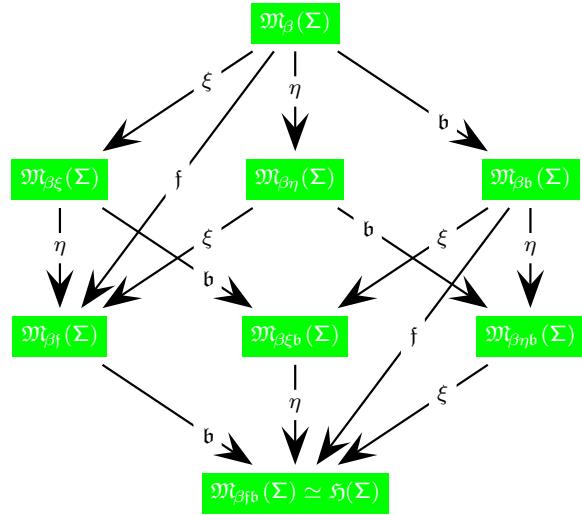
HOL-Problems: ξ



HOL-Problems: b



HOL-Problems: Other Examples



Playing with DeMorgan's Law:

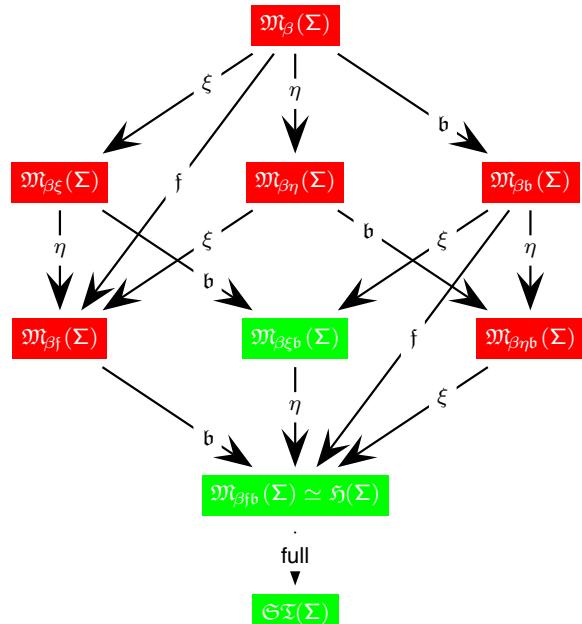
- $\forall X, Y. X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$

'Ok' for all model classes

full
▼

$\mathfrak{S}\mathfrak{T}(\Sigma)$

HOL-Problems: DeMorgan's Law



Playing with DeMorgan's Law:

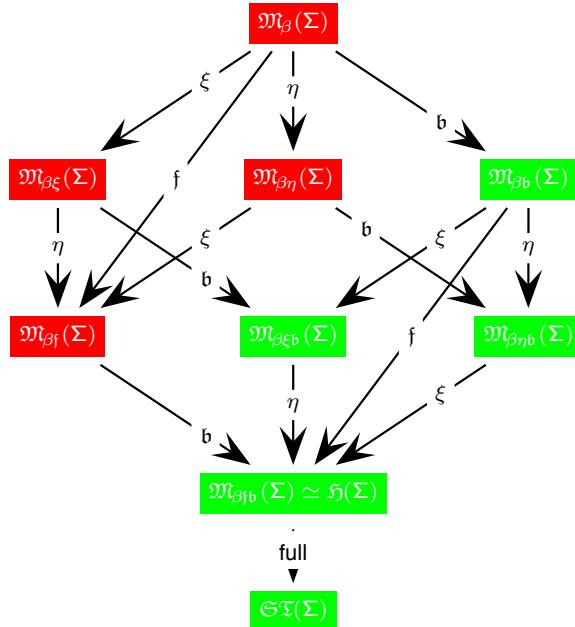
- $\forall X, Y. X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$
- $\forall X, Y. X \wedge Y \stackrel{*}{=} \neg(\neg X \vee \neg Y)$
- $(\lambda U \lambda V. U \wedge V) \stackrel{*}{=} (\lambda X \lambda Y. \neg(\neg X \vee \neg Y))$

requires \$b\$ and \$\xi\$

full
▼

$\mathfrak{S}\mathfrak{T}(\Sigma)$

HOL-Problems: DeMorgan's Law



Playing with DeMorgan's Law:

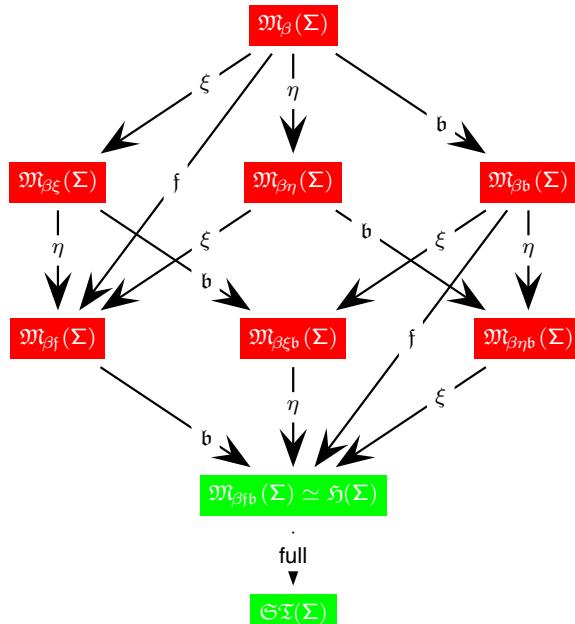
- $\forall X, Y. X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$
- $\forall X, Y. X \wedge Y \stackrel{*}{=} \neg(\neg X \vee \neg Y)$

requires \$b\$

full
▼

$\mathfrak{S}\mathfrak{T}(\Sigma)$

HOL-Problems: DeMorgan's Law



Playing with DeMorgan's Law:

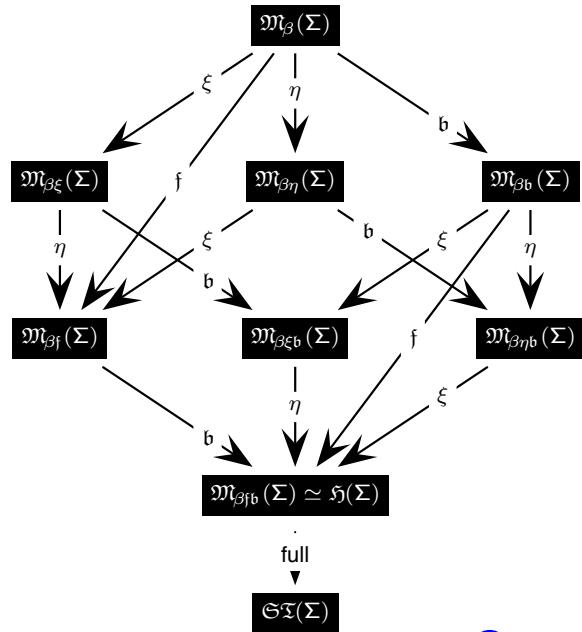
- $\forall X, Y. X \wedge Y \Leftrightarrow \neg(\neg X \vee \neg Y)$
- $\forall X, Y. X \wedge Y \stackrel{*}{=} \neg(\neg X \vee \neg Y)$
- $(\lambda U \lambda V. U \wedge V) \stackrel{*}{=} (\lambda X \lambda Y. \neg(\neg X \vee \neg Y))$
- $\wedge \stackrel{*}{=} (\lambda X \lambda Y. \neg(\neg X \vee \neg Y))$

requires \$b\$ and \$\xi\$

full
▼

$\mathfrak{S}\mathfrak{T}(\Sigma)$

HOL-Problems: Set Comprehension

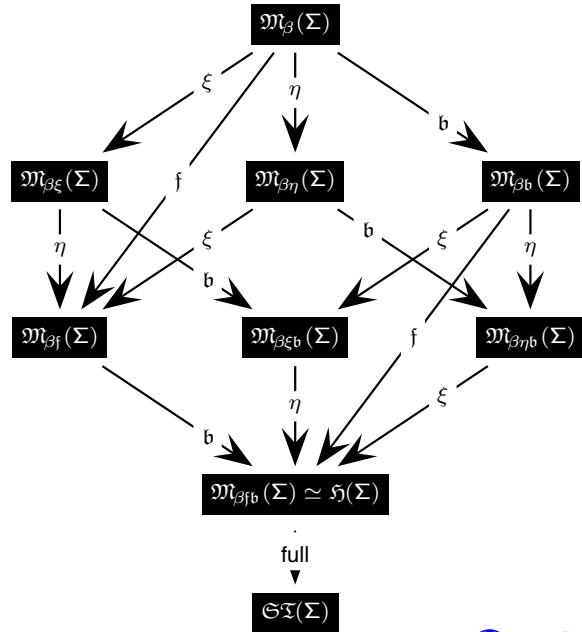


Set comprehension

- 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
- depend on logical constants in Σ

full
▼

HOL-Problems: Set Comprehension



Set comprehension

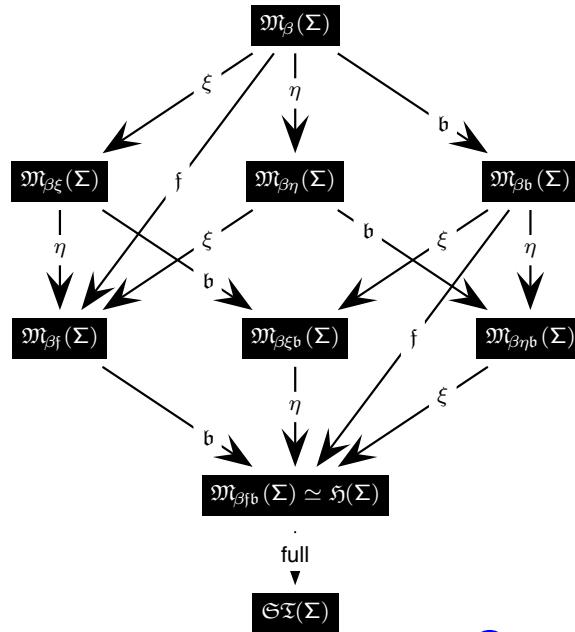
- big challenge for automation
- [Benzm.BrownKohlhase-Draft-05] set instantiations can be used to simulate cut-rule if one of the following axioms is given: comprehension, induction, extensionality, choice, description
- depend on logical constants in Σ

On the following slides emphasis on:

- signature Σ varying
- no property q assumed
- External vs. internal logical constants**
- if $\neg \notin \Sigma$:
 - refers to 'external' symbol
 - $\mathcal{M} \models \neg A$ means $\mathcal{M} \not\models A$

full
▼

HOL-Problems: Set Comprehension



Set comprehension

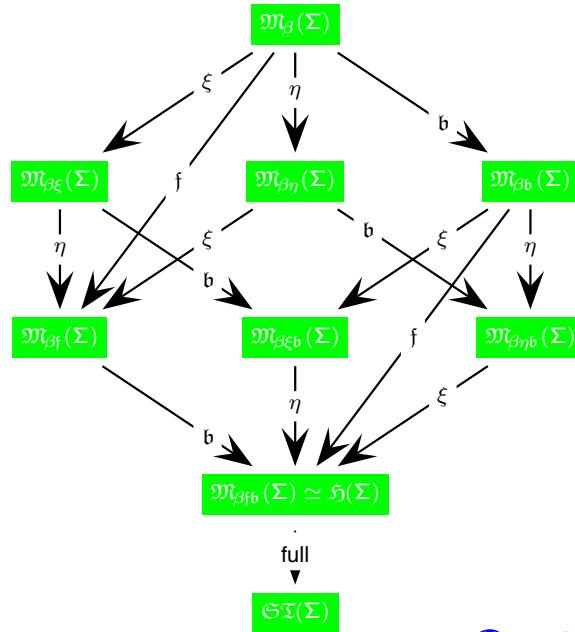
- big challenge for automation
- [Benzm.BrownKohlhase-Draft-05] set instantiations can be used to simulate cut-rule if one of the following axioms is given: comprehension, induction, extensionality, choice, description
- depend on logical constants in Σ

On the following slides emphasis on:

- signature Σ varying
- no property q assumed

full
▼

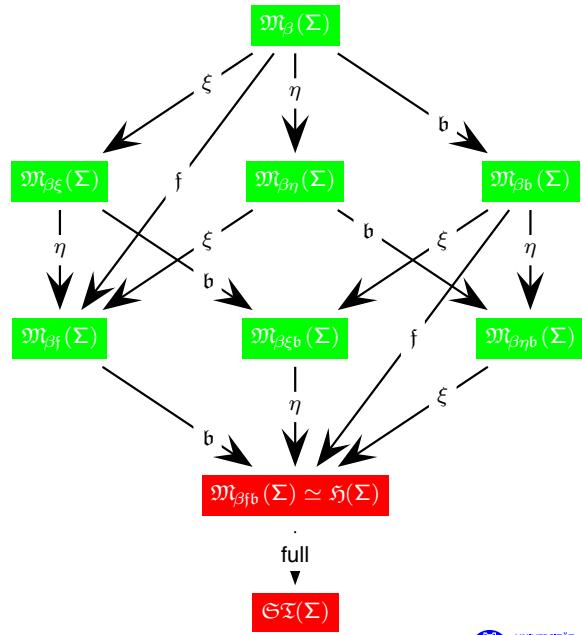
HOL-Problems: Set Comprehension



Set comprehension

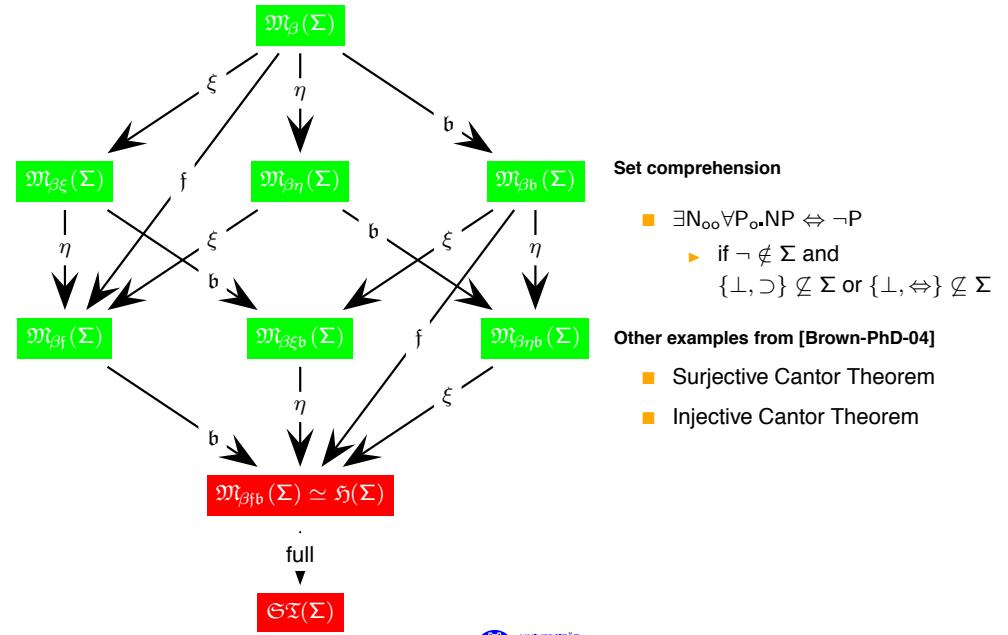
- $\exists N_{\infty} \forall P_0.NP \Leftrightarrow \neg P$
 - if $\neg \in \Sigma$ or $\{\perp, \supset\} \subseteq \Sigma$ or $\{\perp, \Leftrightarrow\} \subseteq \Sigma$
 - e.g.: $N_{\infty} \leftarrow \lambda X_{\infty}.\neg X$
e.g.: $N_{\infty} \leftarrow \lambda X_{\infty}.X \supset \perp$

full
▼



Set comprehension

- $\exists N_o \forall P_o. NP \Leftrightarrow \neg P$
- ▶ if $\neg \notin \Sigma$ and $\{\perp, \supset\} \not\subseteq \Sigma$ or $\{\perp, \Leftrightarrow\} \not\subseteq \Sigma$



Set comprehension

- $\exists N_o \forall P_o. NP \Leftrightarrow \neg P$
- ▶ if $\neg \notin \Sigma$ and $\{\perp, \supset\} \not\subseteq \Sigma$ or $\{\perp, \Leftrightarrow\} \not\subseteq \Sigma$

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.

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 $\mathcal{D}_o := \{\text{T}, \text{F}\}$ and $\mathcal{D}_t := \{*\}$.
- Easy to define an evaluation function \mathcal{E} for this frame by induction on terms, using functions to interpret λ -abstractions.

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\text{T}, \text{F}\}$ and $\mathcal{D}_t := \{*\}$.

Ex.: Singleton Model

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\text{T}, \text{F}\}$ and $\mathcal{D}_t := \{*\}$.
- Easy to define an evaluation function \mathcal{E} for this frame by induction on terms, using functions to interpret λ -abstractions.
- The identity function $v: \mathcal{D}_o \longrightarrow \{\text{T}, \text{F}\}$ 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 $\mathcal{D}_o := \{\text{T}, \text{F}\}$ and $\mathcal{D}_i := \{*\}$.
- Easy to define an evaluation function \mathcal{E} for this frame by induction on terms, using functions to interpret λ -abstractions.
- The identity function $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ is a valuation, assuming the logical constants are interpreted in the standard way.
- Thus, $\mathcal{M}^{\beta\text{fb}} := (\mathcal{D}, @, \mathcal{E}, v)$ defines a Σ -model.

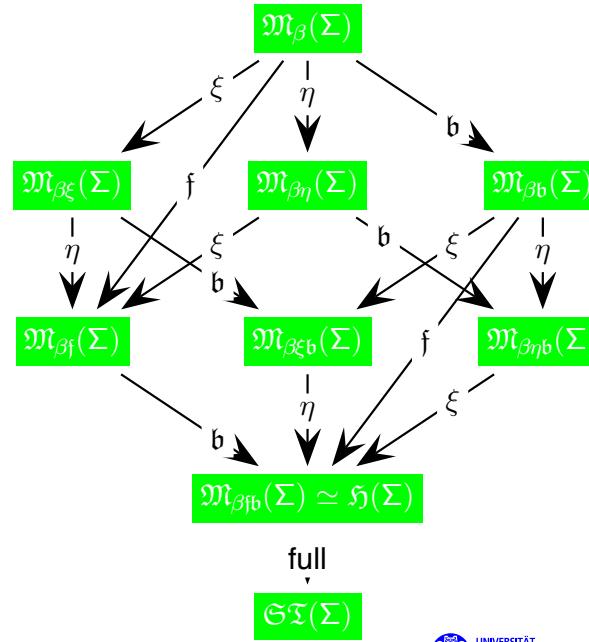
Ex.: Singleton Model

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\text{T}, \text{F}\}$ and $\mathcal{D}_i := \{*\}$.
- Easy to define an evaluation function \mathcal{E} for this frame by induction on terms, using functions to interpret λ -abstractions.
- The identity function $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ is a valuation, assuming the logical constants are interpreted in the standard way.
- Thus, $\mathcal{M}^{\beta\text{fb}} := (\mathcal{D}, @, \mathcal{E}, v)$ defines a Σ -model.
- This model satisfies properties b, f (hence η and ξ) and q (since the frame is full).

Ex.: Singleton Model

- We choose $(\mathcal{D}, @)$ as the full frame with $\mathcal{D}_o := \{\text{T}, \text{F}\}$ and $\mathcal{D}_i := \{*\}$.
- Easy to define an evaluation function \mathcal{E} for this frame by induction on terms, using functions to interpret λ -abstractions.
- The identity function $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ is a valuation, assuming the logical constants are interpreted in the standard way.
- Thus, $\mathcal{M}^{\beta\text{fb}} := (\mathcal{D}, @, \mathcal{E}, v)$ defines a Σ -model.
- This model satisfies properties b, f (hence η and ξ) and q (since the frame is full).
- So, $\mathcal{M}^{\beta\text{fb}} \in \mathfrak{ST}(\Sigma) \subseteq \mathfrak{H}(\Sigma) \subseteq \mathfrak{M}_{\beta\text{fb}}(\Sigma) \subseteq \dots$

Ex.: Singleton Model



Ex.: Model without Boolean Extensionality



- Assume Σ contains only the connectives \neg, \vee, Π^α ; 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, Π^α ; 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 $\mathcal{D}_t = \{0, 1\}$.
- 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
a	a	a	a
b	a	a	a
c	a	a	c

$\mathcal{E}(\vee)$	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, Π^α ; 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 $\mathcal{D}_t = \{0, 1\}$.

Ex.: Model without Boolean Extensionality



Ex.: Model without Boolean Extensionality



- Assume Σ contains only the connectives \neg, \vee, Π^α ; 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 $\mathcal{D}_t = \{0, 1\}$.
- 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
a	a	a	a
b	a	a	a
c	a	a	c

$\mathcal{E}(\vee)$	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



Ex.: Model without Boolean Extensionality



- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines \mathcal{E} on all formulae.

- 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



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 $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ be defined by $v(a) := \text{T}$, $v(b) := \text{T}$ and $v(c) := \text{F}$.

- 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 $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ be defined by $v(a) := \text{T}$, $v(b) := \text{T}$ and $v(c) := \text{F}$.
- Easy to check that $\mathcal{M}^{\beta f} := (\mathcal{D}, @, \mathcal{E}, v)$ 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 $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ be defined by $v(a) := \text{T}$, $v(b) := \text{T}$ and $v(c) := \text{F}$.
- Easy to check that $\mathcal{M}^{\beta f} := (\mathcal{D}, @, \mathcal{E}, v)$ is indeed a Σ -model.
- Since $\mathcal{M}^{\beta f}$ is a model over a frame it satisfies property 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 $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ be defined by $v(a) := \text{T}$, $v(b) := \text{T}$ and $v(c) := \text{F}$.
- Easy to check that $\mathcal{M}^{\beta f} := (\mathcal{D}, @, \mathcal{E}, v)$ is indeed a Σ -model.
- Since $\mathcal{M}^{\beta f}$ is a model over a frame it satisfies property f .
- Since this frame is full, we know property q holds.

Ex.: Model without Boolean Extensionality



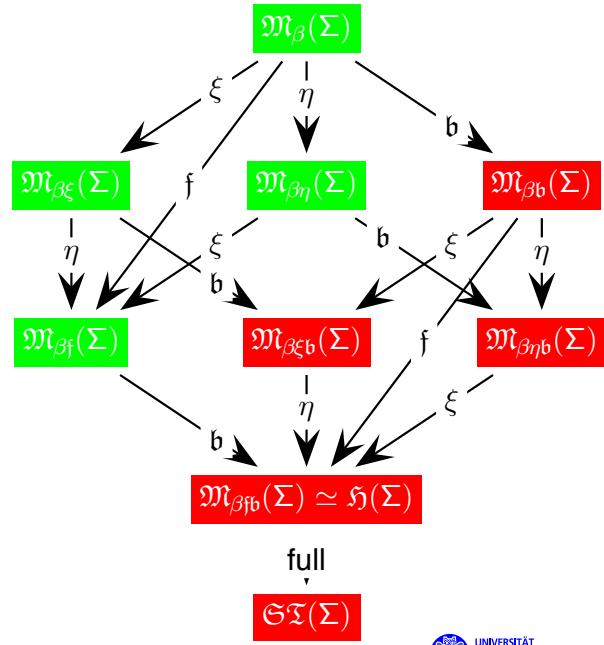
- Since $(\mathcal{D}, @)$ is a frame, hence functional, this uniquely determines \mathcal{E} on all formulae.
- Since the frame is full, we are guaranteed that there will be enough functions to interpret λ -abstractions.
- Let $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ be defined by $v(a) := \text{T}$, $v(b) := \text{T}$ and $v(c) := \text{F}$.
- Easy to check that $\mathcal{M}^{\beta f} := (\mathcal{D}, @, \mathcal{E}, v)$ is indeed a Σ -model.
- Since $\mathcal{M}^{\beta f}$ is a model over a frame it satisfies property f .
- Since this frame is full, we know property q holds.
- Clearly property 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 $v: \mathcal{D}_o \rightarrow \{\text{T}, \text{F}\}$ be defined by $v(a) := \text{T}$, $v(b) := \text{T}$ and $v(c) := \text{F}$.
- Easy to check that $\mathcal{M}^{\beta f} := (\mathcal{D}, @, \mathcal{E}, v)$ is indeed a Σ -model.
- Since $\mathcal{M}^{\beta f}$ is a model over a frame it satisfies property f .
- Since this frame is full, we know property q holds.
- Clearly property b fails.
- So, $\mathcal{M}^{\beta f} \in \mathfrak{M}_{\beta f}(\Sigma) \setminus \mathfrak{M}_{\beta b}(\Sigma)$.

Ex.: Model without Boolean Extensionality



©Benzmüller, 2006



ATPHOL06-[7] – p.210

Ex.: Groundhogs and Woodchucks



- Let $\mathcal{M}^{\beta f}$ be given as above and suppose $\text{woodchuck}_{\iota \rightarrow o}$, $\text{groundhog}_{\iota \rightarrow o}$, john_ι , and phil_ι are in the signature Σ .

Ex.: Model without Boolean Extensionality



In the previous model one can easily verify, if $d := \mathcal{E}_\varphi(D_o)$ and $e := \mathcal{E}_\varphi(E_o)$, then the values $\mathcal{E}_\varphi(D \wedge E)$, $\mathcal{E}_\varphi(D \Rightarrow E)$, and $\mathcal{E}_\varphi(D \Leftrightarrow E)$ are given by the following tables:

$\mathcal{E}(D \wedge E)$	e:	$\mathcal{E}(D \Rightarrow E)$	e:	$\mathcal{E}(D \Leftrightarrow E)$	e:
d: a	a b c	d: a	a b c	d: a	a b c
b	a a c	b	a a c	b	a a c
c	c c c	c	a a a	c	c c a

Now we show that one can properly model the **woodchuck/groundhog** example.



ATPHOL06-[7] – p.211

Ex.: Groundhogs and Woodchucks



- Let $\mathcal{M}^{\beta f}$ be given as above and suppose $\text{woodchuck}_{\iota \rightarrow o}$, $\text{groundhog}_{\iota \rightarrow o}$, john_ι , and phil_ι are in the signature Σ .
- Let $\mathcal{E}(\text{phil}) := 0$ and $\mathcal{E}(\text{john}) := 1$.



ATPHOL06-[7] – p.212

©Benzmüller, 2006



ATPHOL06-[7] – p.212

©Benzmüller, 2006

Ex.: Groundhogs and Woodchucks



- Let $\mathcal{M}^{\beta f}$ be given as above and suppose $\text{woodchuck}_{\iota \rightarrow o}$, $\text{groundhog}_{\iota \rightarrow o}$, john_ι , and phil_ι are in the signature Σ .
- Let $\mathcal{E}(\text{phil}) := 0$ and $\mathcal{E}(\text{john}) := 1$.
- Let $\mathcal{E}(\text{woodchuck})$ be the function $w \in \mathcal{D}_{\iota \rightarrow o}$ with $w(0) = b$ and $w(1) = c$.

Ex.: Groundhogs and Woodchucks



- Let $\mathcal{M}^{\beta f}$ be given as above and suppose $\text{woodchuck}_{\iota \rightarrow o}$, $\text{groundhog}_{\iota \rightarrow o}$, john_ι , and phil_ι are in the signature Σ .
- Let $\mathcal{E}(\text{phil}) := 0$ and $\mathcal{E}(\text{john}) := 1$.
- Let $\mathcal{E}(\text{woodchuck})$ be the function $w \in \mathcal{D}_{\iota \rightarrow o}$ with $w(0) = b$ and $w(1) = c$.
- Let $\mathcal{E}(\text{groundhog})$ be the function $g \in \mathcal{D}_{\iota \rightarrow o}$ with $g(0) = a$ and $g(1) = c$.

Ex.: Groundhogs and Woodchucks



- Let $\mathcal{M}^{\beta f}$ be given as above and suppose $\text{woodchuck}_{\iota \rightarrow o}$, $\text{groundhog}_{\iota \rightarrow o}$, john_ι , and phil_ι are in the signature Σ .
- Let $\mathcal{E}(\text{phil}) := 0$ and $\mathcal{E}(\text{john}) := 1$.
- Let $\mathcal{E}(\text{woodchuck})$ be the function $w \in \mathcal{D}_{\iota \rightarrow o}$ with $w(0) = b$ and $w(1) = c$.
- Let $\mathcal{E}(\text{groundhog})$ be the function $g \in \mathcal{D}_{\iota \rightarrow o}$ with $g(0) = a$ and $g(1) = c$.
- One can show that the sentence $\forall X_\iota (\text{woodchuck } X) \Leftrightarrow (\text{groundhog } X)$ is valid.

Ex.: Groundhogs and Woodchucks



- Let $\mathcal{M}^{\beta f}$ be given as above and suppose $\text{woodchuck}_{\iota \rightarrow o}$, $\text{groundhog}_{\iota \rightarrow o}$, john_ι , and phil_ι are in the signature Σ .
- Let $\mathcal{E}(\text{phil}) := 0$ and $\mathcal{E}(\text{john}) := 1$.
- Let $\mathcal{E}(\text{woodchuck})$ be the function $w \in \mathcal{D}_{\iota \rightarrow o}$ with $w(0) = b$ and $w(1) = c$.
- Let $\mathcal{E}(\text{groundhog})$ be the function $g \in \mathcal{D}_{\iota \rightarrow o}$ with $g(0) = a$ and $g(1) = c$.
- One can show that the sentence $\forall X_\iota (\text{woodchuck } X) \Leftrightarrow (\text{groundhog } X)$ is valid.
- Also, $\mathcal{E}(\text{woodchuck } \text{phil}) = b$ and $\mathcal{E}(\text{groundhog } \text{phil}) = a$, so the propositions $(\text{woodchuck } \text{phil})$ and $(\text{groundhog } \text{phil})$ are valid.

Ex.: Groundhogs and Woodchucks



Ex.: Groundhogs and Woodchucks



- Suppose $\text{believe}_{\iota \rightarrow o \rightarrow o} \in \Sigma$ and $\mathcal{E}(\text{believe})$ is the (Curried) function $\text{bel} \in \mathcal{D}_{\iota \rightarrow o \rightarrow o}$ such that $\text{bel}(1)(b) = b$ and $\text{bel}(1)(a) = \text{bel}(1)(c) = \text{bel}(0)(a) = \text{bel}(0)(b) = \text{bel}(0)(c) = c$.

- Suppose $\text{believe}_{\iota \rightarrow o \rightarrow o} \in \Sigma$ and $\mathcal{E}(\text{believe})$ is the (Curried) function $\text{bel} \in \mathcal{D}_{\iota \rightarrow o \rightarrow o}$ such that $\text{bel}(1)(b) = b$ and $\text{bel}(1)(a) = \text{bel}(1)(c) = \text{bel}(0)(a) = \text{bel}(0)(b) = \text{bel}(0)(c) = c$.
- Intuitively, John believes propositions with value **b**, but not those with value **a** or **c**.

©Benzmüller, 2006



ATPHOL06-[7] – p.213

©Benzmüller, 2006



ATPHOL06-[7] – p.213

Ex.: Groundhogs and Woodchucks



Generalizing the Previous Model



- Suppose $\text{believe}_{\iota \rightarrow o \rightarrow o} \in \Sigma$ and $\mathcal{E}(\text{believe})$ is the (Curried) function $\text{bel} \in \mathcal{D}_{\iota \rightarrow o \rightarrow o}$ such that $\text{bel}(1)(b) = b$ and $\text{bel}(1)(a) = \text{bel}(1)(c) = \text{bel}(0)(a) = \text{bel}(0)(b) = \text{bel}(0)(c) = c$.
- Intuitively, John believes propositions with value **b**, but not those with value **a** or **c**.
- So, $\text{believes john(woodchuck phil)}$ is valid, while $\text{believes john(groundhog phil)}$ is not.

As we have seen, Boolean extensionality fails when one has more than two values in \mathcal{D}_o .

©Benzmüller, 2006



ATPHOL06-[7] – p.213

©Benzmüller, 2006



ATPHOL06-[7] – p.214

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 $\mathcal{D}_o := \{F\} \cup \mathcal{B}$, where \mathcal{B} is any set with $T \in \mathcal{B}$ and $F \notin \mathcal{B}$.

©Benzmüller, 2006



ATPHOL06-[7] – p.214

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 $\mathcal{D}_o := \{F\} \cup \mathcal{B}$, where \mathcal{B} is any set with $T \in \mathcal{B}$ and $F \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{T\}$. In this way, we can easily construct models for the case with property b and the case without property b simultaneously.

©Benzmüller, 2006



ATPHOL06-[7] – p.214

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 $\mathcal{D}_o := \{F\} \cup \mathcal{B}$, where \mathcal{B} is any set with $T \in \mathcal{B}$ and $F \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{T\}$.

©Benzmüller, 2006



ATPHOL06-[7] – p.214

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 $\mathcal{D}_o := \{F\} \cup \mathcal{B}$, where \mathcal{B} is any set with $T \in \mathcal{B}$ and $F \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{T\}$. In this way, we can easily construct models for the case with property b and the case without property b simultaneously. We will use this idea to parameterize the remaining model constructions by \mathcal{B} .

©Benzmüller, 2006



ATPHOL06-[7] – p.214

As we have seen, Boolean extensionality fails when one has more than two values in \mathcal{D}_o . We can generalize the construction defining $\mathcal{D}_o := \{F\} \cup \mathcal{B}$, where \mathcal{B} is any set with $T \in \mathcal{B}$ and $F \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{T\}$. In this way, we can easily construct models for the case with property b and the case without property 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.

As we have seen, Boolean extensionality fails when one has more than two values in \mathcal{D}_o . We can generalize the construction defining $\mathcal{D}_o := \{F\} \cup \mathcal{B}$, where \mathcal{B} is any set with $T \in \mathcal{B}$ and $F \notin \mathcal{B}$. The model will satisfy Boolean extensionality iff $\mathcal{B} = \{T\}$. In this way, we can easily construct models for the case with property b and the case without property 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 v -pre-images of T and 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 $T \in \mathcal{B}$ and $F \notin \mathcal{B}$

©Benzmüller, 2006



ATPHOL06-[8] – p.216

Ex.: Models without Funct. Extensionality



- Idea: attach distinguishing labels to functions without changing their applicative behavior
- Let \mathcal{B} be any set with $T \in \mathcal{B}$ and $F \notin \mathcal{B}$
- Let $\mathcal{D}_o := \{F\} \cup \mathcal{B}$ and $\mathcal{D}_i := \{*\}$
- For each function type $\beta\alpha$, let

$$\mathcal{D}_{\beta\alpha} := \{(i, f) \mid i \in \{0, 1\} \text{ and } f: \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta\}$$

©Benzmüller, 2006



ATPHOL06-[8] – p.216

Ex.: Models without Funct. Extensionality



- Idea: attach distinguishing labels to functions without changing their applicative behavior
- Let \mathcal{B} be any set with $T \in \mathcal{B}$ and $F \notin \mathcal{B}$
- Let $\mathcal{D}_o := \{F\} \cup \mathcal{B}$ and $\mathcal{D}_i := \{*\}$

©Benzmüller, 2006



ATPHOL06-[8] – p.216

Ex.: Models without Funct. Extensionality



- Idea: attach distinguishing labels to functions without changing their applicative behavior
- Let \mathcal{B} be any set with $T \in \mathcal{B}$ and $F \notin \mathcal{B}$
- Let $\mathcal{D}_o := \{F\} \cup \mathcal{B}$ and $\mathcal{D}_i := \{*\}$
- For each function type $\beta\alpha$, let

$$\mathcal{D}_{\beta\alpha} := \{(i, f) \mid i \in \{0, 1\} \text{ and } f: \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta\}$$

- We define application by

$$(i, f)@a := f(a)$$

whenever $(i, f) \in \mathcal{D}_{\beta\alpha}$ and $a \in \mathcal{D}_\alpha$

©Benzmüller, 2006



ATPHOL06-[8] – p.216

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
 - $\mathcal{E}(\neg) := (0, n)$ where $n(b) := F$ for every $b \in \mathcal{B}$ and $n(F) := T$

Ex.: Models without η and \mathfrak{f}



- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms

Ex.: Models without η and \mathfrak{f}



- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms
 - $\mathcal{E}(\neg) := (0, n)$ where $n(b) := F$ for every $b \in \mathcal{B}$ and $n(F) := T$
 - $\mathcal{E}(v) := (0, d)$ where
 - $d(b) := (0, k^T)$ for every $b \in \mathcal{B}$ and
 - $d(F) := (0, id)$(k^T is the constant T function)
(id is the identity function from \mathcal{D}_o to \mathcal{D}_o)

Ex.: Models without η and f



- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms
 - ▶ $\mathcal{E}(\neg) := (0, n)$ where $n(b) := F$ for every $b \in \mathcal{B}$ and $n(F) := T$
 - ▶ $\mathcal{E}(V) := (0, d)$ where
 - $d(b) := (0, k^T)$ for every $b \in \mathcal{B}$ and
 - $d(F) := (0, id)$
(k^T is the constant 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 $(i, f) \in \mathcal{D}_{o\alpha}$, $\pi^\alpha((i, f)) := T$ if $f(a) \in \mathcal{B}$ for all $a \in \mathcal{D}_\alpha$ and $\pi^\alpha(i, f) := F$ otherwise

Ex.: Models without η and f



- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms
 - ▶ $\mathcal{E}(\neg) := (0, n)$ where $n(b) := F$ for every $b \in \mathcal{B}$ and $n(F) := T$
 - ▶ $\mathcal{E}(V) := (0, d)$ where
 - $d(b) := (0, k^T)$ for every $b \in \mathcal{B}$ and
 - $d(F) := (0, id)$
(k^T is the constant 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 $(i, f) \in \mathcal{D}_{o\alpha}$, $\pi^\alpha((i, f)) := T$ if $f(a) \in \mathcal{B}$ for all $a \in \mathcal{D}_\alpha$ and $\pi^\alpha(i, f) := F$ otherwise
 - ▶ $q^\alpha := (0, q^\alpha) \in \mathcal{D}_{o\alpha\alpha}$ where $q^\alpha(a) := (0, s^a)$ and $s^a(b) := T$ if $a = b$ and $s^a(b) := F$ otherwise

Ex.: Models without η and f



Ex.: Models without η and f

- Easy to check that $(\mathcal{D}, @)$ is an applicative structure:
- Evaluation function defined by induction on terms
 - ▶ $\mathcal{E}(\neg) := (0, n)$ where $n(b) := F$ for every $b \in \mathcal{B}$ and $n(F) := T$
 - ▶ $\mathcal{E}(V) := (0, d)$ where
 - $d(b) := (0, k^T)$ for every $b \in \mathcal{B}$ and
 - $d(F) := (0, id)$
(k^T is the constant 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 $(i, f) \in \mathcal{D}_{o\alpha}$, $\pi^\alpha((i, f)) := T$ if $f(a) \in \mathcal{B}$ for all $a \in \mathcal{D}_\alpha$ and $\pi^\alpha(i, f) := F$ otherwise
 - ▶ $q^\alpha := (0, q^\alpha) \in \mathcal{D}_{o\alpha\alpha}$ where $q^\alpha(a) := (0, s^a)$ and $s^a(b) := T$ if $a = b$ and $s^a(b) := F$ otherwise
 - ▶ $\mathcal{E}(w) \in \mathcal{D}_\alpha$ arbitrary for parameters $w \in \Sigma_\alpha$.



Ex.: Models without η and f



Ex.: Models without η and \mathfrak{f}



- For variables, we define $\mathcal{E}_\varphi(X) := \varphi(X)$

Ex.: Models without η and \mathfrak{f}



- For variables, we define $\mathcal{E}_\varphi(X) := \varphi(X)$
- For application, we define $\mathcal{E}_\varphi(FA) := \mathcal{E}_\varphi(F)@\mathcal{E}_\varphi(A)$

Ex.: Models without η and \mathfrak{f}



- For variables, we define $\mathcal{E}_\varphi(X) := \varphi(X)$
- For application, we define $\mathcal{E}_\varphi(FA) := \mathcal{E}_\varphi(F)@\mathcal{E}_\varphi(A)$
- For λ -abstractions, we define $\mathcal{E}_\varphi(\lambda X_\alpha.B_\beta) := (0, f)$ where $f: D_\alpha \longrightarrow D_\beta$ is the function such that $f(a) = \mathcal{E}_{\varphi,[a/X]}(B)$ for all $a \in D_\alpha$

Ex.: Models without η and \mathfrak{f}



- For variables, we define $\mathcal{E}_\varphi(X) := \varphi(X)$
- For application, we define $\mathcal{E}_\varphi(FA) := \mathcal{E}_\varphi(F)@\mathcal{E}_\varphi(A)$
- For λ -abstractions, we define $\mathcal{E}_\varphi(\lambda X_\alpha.B_\beta) := (0, f)$ where $f: D_\alpha \longrightarrow D_\beta$ is the function such that $f(a) = \mathcal{E}_{\varphi,[a/X]}(B)$ for all $a \in D_\alpha$
- 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 $\mathcal{E}_\varphi(X) := \varphi(X)$
- ▶ For application, we define $\mathcal{E}_\varphi(FA) := \mathcal{E}_\varphi(F) @ \mathcal{E}_\varphi(A)$
- ▶ For λ -abstractions, we define $\mathcal{E}_\varphi(\lambda X_\alpha.B_\beta) := (0, f)$ where $f: \mathcal{D}_\alpha \rightarrow \mathcal{D}_\beta$ is the function such that $f(a) = \mathcal{E}_{\varphi,[a/X]}(B)$ for all $a \in \mathcal{D}_\alpha$
- With some work (which we omit), one can show that this \mathcal{E} is an evaluation function
- Taking v to be the function such that $v(b) := T$ for every $b \in \mathcal{B}$ and $v(F) := F$, one can easily show that this is a valuation

Ex.: Models without η and \mathfrak{f}



- ▶ For variables, we define $\mathcal{E}_\varphi(X) := \varphi(X)$
- ▶ For application, we define $\mathcal{E}_\varphi(FA) := \mathcal{E}_\varphi(F) @ \mathcal{E}_\varphi(A)$
- ▶ For λ -abstractions, we define $\mathcal{E}_\varphi(\lambda X_\alpha.B_\beta) := (0, f)$ where $f: \mathcal{D}_\alpha \rightarrow \mathcal{D}_\beta$ is the function such that $f(a) = \mathcal{E}_{\varphi,[a/X]}(B)$ for all $a \in \mathcal{D}_\alpha$
- With some work (which we omit), one can show that this \mathcal{E} is an evaluation function
- Taking v to be the function such that $v(b) := T$ for every $b \in \mathcal{B}$ and $v(F) := F$, one can easily show that this is a valuation
- Hence, $\mathcal{M}^{\mathcal{B}} := (\mathcal{D}, @, \mathcal{E}, v)$ is a Σ -model

Ex.: Models without η and \mathfrak{f}



- The objects $q^\alpha := (0, q^\alpha)$ witness property q for $\mathcal{M}^{\mathcal{B}}$

Ex.: Models without η and \mathfrak{f}



- The objects $q^\alpha := (0, q^\alpha)$ witness property q for $\mathcal{M}^{\mathcal{B}}$
- The objects $(1, q^\alpha)$ also witness property q (so, in the non-functional case such witnesses are not unique)

Ex.: Models without η and f



Ex.: Models without η and f



- The objects $q^\alpha := (0, q^\alpha)$ witness property q for \mathcal{M}^B
- The objects $(1, q^\alpha)$ also witness property q (so, in the non-functional case such witnesses are not unique)
- Hence, $\mathcal{M}^B := (\mathcal{D}, @, \mathcal{E}, v)$ is a Σ -model with property q

- Property f fails for \mathcal{M}^B , since the applicative structure $(\mathcal{D}, @)$ is not functional:

©Benzmüller, 2006



ATPHOL06-[8] – p.219

©Benzmüller, 2006



ATPHOL06-[8] – p.220

Ex.: Models without η and f



Ex.: Models without η and f



- Property f fails for \mathcal{M}^B , since the applicative structure $(\mathcal{D}, @)$ is not functional:
 - Consider $u: \mathcal{D}_v \longrightarrow \mathcal{D}_v$.

- Property f fails for \mathcal{M}^B , since the applicative structure $(\mathcal{D}, @)$ is not functional:
 - Consider $u: \mathcal{D}_v \longrightarrow \mathcal{D}_v$.
 - For both $(0, u), (1, u) \in \mathcal{D}_u$ we have

$$(i, u) @ * = *$$

although $(0, u) \neq (1, u)$

©Benzmüller, 2006



ATPHOL06-[8] – p.220

©Benzmüller, 2006



ATPHOL06-[8] – p.220

Ex.: Models without η and \mathfrak{f}



Ex.: Models without η and \mathfrak{f}



- Does η hold?

- Does η hold?
- No!

Ex.: Models without η and \mathfrak{f}



- Does η hold?
- No!
- Compute, for example, $\mathcal{E}(\lambda F_{\beta\alpha}.F)$ and $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_{\alpha}.FX)$

Ex.: Models without η and \mathfrak{f}



- Does η hold?
- No!
- Compute, for example, $\mathcal{E}(\lambda F_{\beta\alpha}.F)$ and $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_{\alpha}.FX)$
 - $\mathcal{E}(\lambda F_{\beta\alpha}.F) = (0, \text{id})$ where id is the identity function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$

Ex.: Models without η and ξ



- Does η hold?
- No!
- Compute, for example, $\mathcal{E}(\lambda F_{\beta\alpha}.F)$ and $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_{\alpha}.FX)$
 - ▶ $\mathcal{E}(\lambda F_{\beta\alpha}.F) = (0, \text{id})$ where id is the identity function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$
 - ▶ $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_{\alpha}.FX) = (0, p)$ where p is the function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$ such that $p((i, f)) = (0, f)$ for each $f: \mathcal{D}_{\alpha} \rightarrow \mathcal{D}_{\beta}$

Ex.: Models without η and ξ



- Does η hold?
- No!
- Compute, for example, $\mathcal{E}(\lambda F_{\beta\alpha}.F)$ and $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_{\alpha}.FX)$
 - ▶ $\mathcal{E}(\lambda F_{\beta\alpha}.F) = (0, \text{id})$ where id is the identity function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$
 - ▶ $\mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_{\alpha}.FX) = (0, p)$ where p is the function from $\mathcal{D}_{\beta\alpha}$ to $\mathcal{D}_{\beta\alpha}$ such that $p((i, f)) = (0, f)$ for each $f: \mathcal{D}_{\alpha} \rightarrow \mathcal{D}_{\beta}$
- Hence $\mathcal{E}(\lambda F_{\beta\alpha}.F) \neq \mathcal{E}(\lambda F_{\beta\alpha}.\lambda X_{\alpha}.FX)$

Ex.: Models without η and ξ



- Does ξ hold?

Ex.: Models without η and ξ



- Does ξ hold?
- Yes!

Ex.: Models without η and f



Ex.: Models without η and f



- Does ξ hold?
- Yes!
- If

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

for every $a \in D_\alpha$, then

$$\mathcal{E}_\varphi(\lambda X_\alpha.M) = (0, f) = \mathcal{E}_\varphi(\lambda X.N)$$

where $f(a) = \mathcal{E}_{\varphi,[a/X]}(M) = \mathcal{E}_{\varphi,[a/X]}(N)$ for every $a \in D_\alpha$.

Ex.: Models without η and f



Ex.: Models without η and f



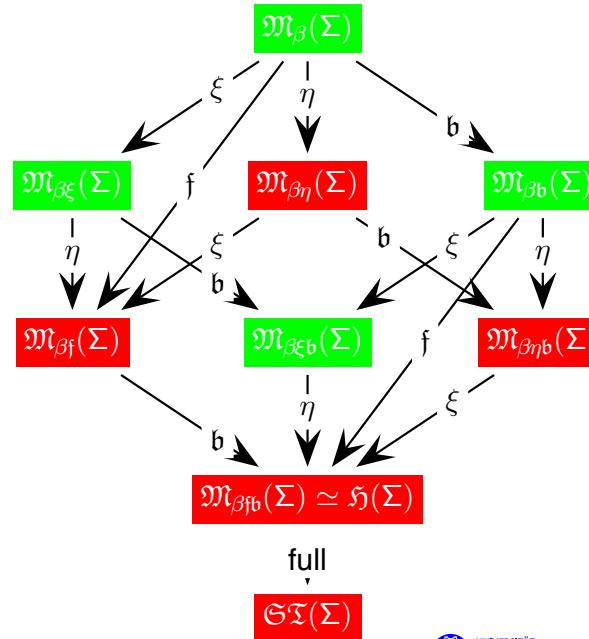
- If $B = \{T\}$, then the model $\mathcal{M}^{\beta\xi b} := \mathcal{M}^{\{T\}}$ satisfies property b .
- So, we know $\mathcal{M}^{\beta\xi b} \in \mathfrak{M}_{\beta\xi b}(\Sigma) \setminus \mathfrak{M}_{\beta\mathfrak{f}b}(\Sigma)$.

- If $B = \{T\}$, then the model $\mathcal{M}^{\beta\xi b} := \mathcal{M}^{\{T\}}$ satisfies property b .
- So, we know $\mathcal{M}^{\beta\xi b} \in \mathfrak{M}_{\beta\xi b}(\Sigma) \setminus \mathfrak{M}_{\beta\mathfrak{f}b}(\Sigma)$.
- On the other hand, if b is any value with $b \notin \{T, F\}$, and $B = \{T, b\}$, then the model $\mathcal{M}^{\beta\xi} := \mathcal{M}^{\{T, b\}}$ does not satisfy property b .

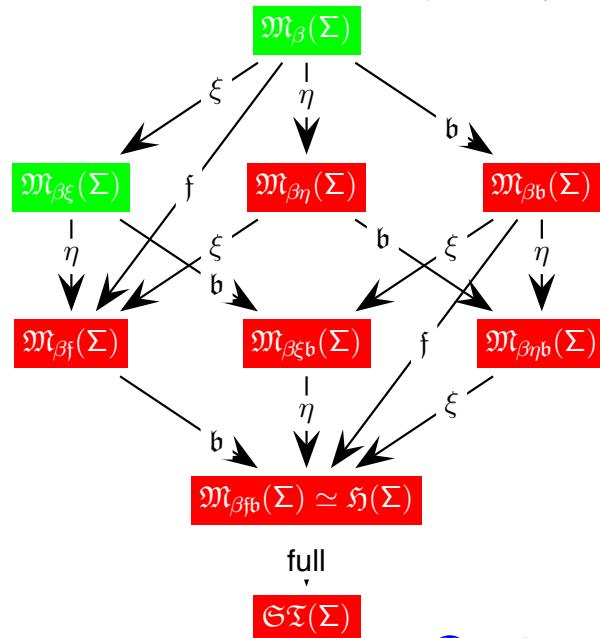
Ex.: Models without η and f

- If $B = \{T\}$, then the model $\mathcal{M}^{\beta\xi b} := \mathcal{M}^{\{T\}}$ satisfies property b .
- So, we know $\mathcal{M}^{\beta\xi b} \in \mathfrak{M}_{\beta\xi b}(\Sigma) \setminus \mathfrak{M}_{\beta b}(\Sigma)$.
- On the other hand, if b is any value with $b \notin \{T, F\}$, and $B = \{T, b\}$, then the model $\mathcal{M}^{\beta\xi} := \mathcal{M}^{\{T, b\}}$ does not satisfy property b .
- In this case, we know $\mathcal{M}^{\beta\xi} \in \mathfrak{M}_{\beta\xi}(\Sigma) \setminus (\mathfrak{M}_{\beta f}(\Sigma) \cup \mathfrak{M}_{\beta\xi b}(\Sigma))$.

Ex.: Models without η and f



Ex.: Models without η and f



Ex.: Models without η and f

- Let \mathcal{M}^B be the Σ -model $(\mathcal{D}, @, \mathcal{E}, v)$ as constructed before

Ex.: Models without η and \mathfrak{f}



- Let \mathcal{M}^B be the Σ -model $(\mathcal{D}, @, \mathcal{E}, v)$ as constructed before
- Define an alternative evaluation function \mathcal{E}' by induction:

Ex.: Models without η and \mathfrak{f}



- Let \mathcal{M}^B be the Σ -model $(\mathcal{D}, @, \mathcal{E}, v)$ as constructed before
- Define an alternative evaluation function \mathcal{E}' by induction:
 - For all $w \in \Sigma$, let $\mathcal{E}'(w) := \mathcal{E}(w)$

Ex.: Models without η and \mathfrak{f}



- Let \mathcal{M}^B be the Σ -model $(\mathcal{D}, @, \mathcal{E}, v)$ as constructed before
- Define an alternative evaluation function \mathcal{E}' by induction:
 - For all $w \in \Sigma$, let $\mathcal{E}'(w) := \mathcal{E}(w)$
 - For variables we define $\mathcal{E}'_\varphi(X) := \varphi(X)$

Ex.: Models without η and \mathfrak{f}



- Let \mathcal{M}^B be the Σ -model $(\mathcal{D}, @, \mathcal{E}, v)$ as constructed before
- Define an alternative evaluation function \mathcal{E}' by induction:
 - For all $w \in \Sigma$, let $\mathcal{E}'(w) := \mathcal{E}(w)$
 - For variables we define $\mathcal{E}'_\varphi(X) := \varphi(X)$
 - We must define $\mathcal{E}'_\varphi(\mathbf{F}\mathbf{A}) := \mathcal{E}'_\varphi(\mathbf{F}) @ \mathcal{E}'_\varphi(\mathbf{A})$

Ex.: Models without η and f



- Let \mathcal{M}^B be the Σ -model $(\mathcal{D}, @, \mathcal{E}, v)$ as constructed before
 - Define an alternative evaluation function \mathcal{E}' by induction:
 - For all $w \in \Sigma$, let $\mathcal{E}'(w) := \mathcal{E}(w)$
 - For variables we define $\mathcal{E}'_\varphi(X) := \varphi(X)$
 - We must define $\mathcal{E}'_\varphi(F A) := \mathcal{E}'_\varphi(F) @ \mathcal{E}'_\varphi(A)$
 - We choose $\mathcal{E}'_\varphi(\lambda X_\alpha B_\beta) := (f, f)$ where $f: \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$ is the function such that $f(a) = \mathcal{E}_{\varphi, [a/X]}(B)$ for all $a \in \mathcal{D}_\alpha$

Ex.: Models without η and f



- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ -model $(\mathcal{D}, @, \mathcal{E}, v)$ as constructed before
 - Define an alternative evaluation function \mathcal{E}' by induction:
 - For all $w \in \Sigma$, let $\mathcal{E}'(w) := \mathcal{E}(w)$
 - For variables we define $\mathcal{E}'_{\varphi}(X) := \varphi(X)$
 - We must define $\mathcal{E}'_{\varphi}(F A) := \mathcal{E}'_{\varphi}(F) @ \mathcal{E}'_{\varphi}(A)$
 - We choose $\mathcal{E}'_{\varphi}(\lambda X_{\alpha}. B_{\beta}) := (1, f)$ where $f: \mathcal{D}_{\alpha} \longrightarrow \mathcal{D}_{\beta}$ is the function such that $f(a) = \mathcal{E}_{\varphi, [a/X]}(B)$ for all $a \in \mathcal{D}_{\alpha}$
 - \mathcal{E} and \mathcal{E}' agree on all constants, they are different though:

$$\mathcal{E}(\lambda X_\nu.X) = (0, \text{id}) \neq (1, \text{id}) = \mathcal{E}'(\lambda X_\nu.X)$$

where $\text{id} : \mathcal{D}_I \rightarrow \mathcal{D}_I$ is the identity function

Ex.: Models without η and f



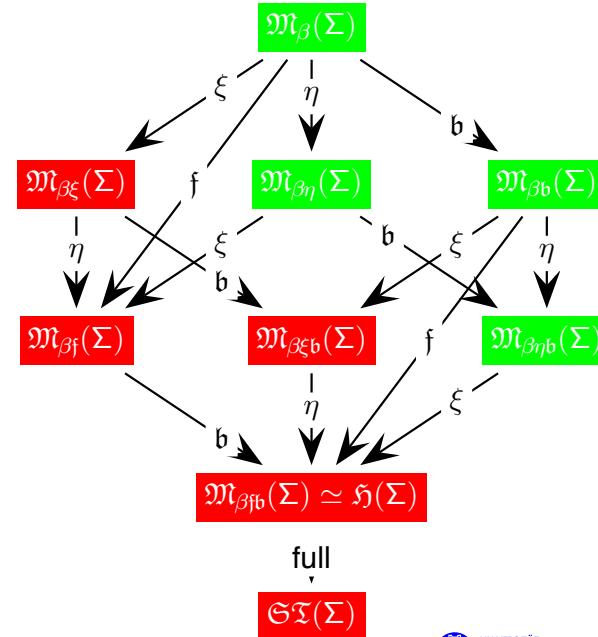
- Let $\mathcal{M}^{\mathcal{B}}$ be the Σ -model $(\mathcal{D}, @, \mathcal{E}, v)$ as constructed before
 - Define an alternative evaluation function \mathcal{E}' by induction:
 - For all $w \in \Sigma$, let $\mathcal{E}'(w) := \mathcal{E}(w)$
 - For variables we define $\mathcal{E}'_\varphi(X) := \varphi(X)$
 - We must define $\mathcal{E}'_\varphi(\mathbf{F}\mathbf{A}) := \mathcal{E}'_\varphi(\mathbf{F}) @ \mathcal{E}'_\varphi(\mathbf{A})$
 - We choose $\mathcal{E}'_\varphi(\lambda X_\alpha . B_\beta) := (1, f)$ where $f: \mathcal{D}_\alpha \longrightarrow \mathcal{D}_\beta$ is the function such that $f(a) = \mathcal{E}_{\varphi, [a/X]}(B)$ for all $a \in \mathcal{D}_\alpha$
 - \mathcal{E} and \mathcal{E}' agree on all constants, they are different though:

$$\mathcal{E}(\lambda X_\nu.X) = (0, \text{id}) \neq (1, \text{id}) = \mathcal{E}'(\lambda X_\nu.X)$$

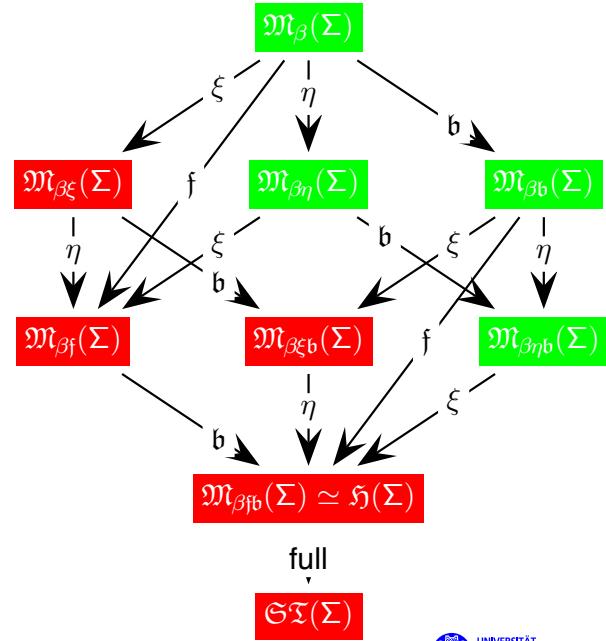
where $\text{id} : \mathcal{D}_t \rightarrow \mathcal{D}_t$ is the identity function

- Thus, in non-functional models evaluation functions are not uniquely determined by their values on constants

Ex.: Models without ξ



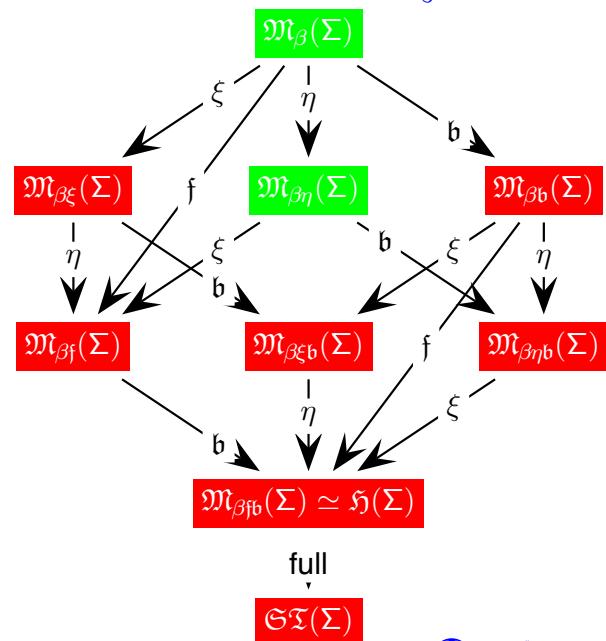
Ex.: Models without ξ



Not here!

See [JSL-04]

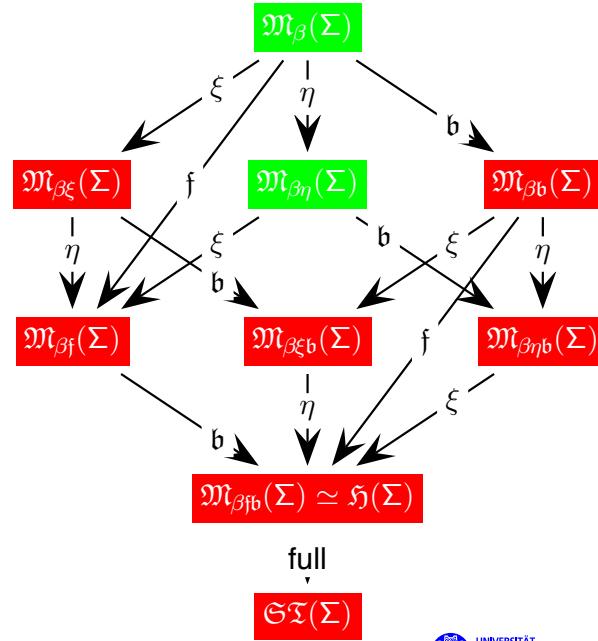
Ex.: Models without ξ



Not here!

See [JSL-04]

Ex.: Models without ξ



©Benzmüller, 2006

ATPHOL06-[8] – p.228



Calculi: First-Order Natural
Deduction and Sequent
Calculus

From Natural Deduction to Sequent Calculus and Back

From Natural Deduction to Sequent Calculus and Back

Remark: We first illustrate the correspondence between natural deduction and sequent calculus in first-order logic. Later we will present natural deduction calculi for HOL. More precisely we will present one sound and complete calculus for each class in our landscape of semantics as presented before.

Reading

- F. Pfenning: Automated Theorem Proving, Course at Carnegie Mellon University. Draft. 1999.
- A.S. Troelstra and H. Schwichtenberg: Basic Proof Theory. Cambridge. 2nd Edition 2000.
- John Byrnes: Proof Search and Normal Forms in Natural Deduction. PhD Thesis. Carnegie Mellon University. 1999.
- ...many more books on Proof Theory ...

Natural Deduction: Motivation

- Frege, Russel, Hilbert: Predicate calculus and type theory as formal basis for mathematics

- Frege, Russel, Hilbert: Predicate calculus and type theory as formal basis for mathematics
- Gentzen: Natural deduction (ND) as intuitive formulation of predicate calculus; introduction and elimination rules for each logical connective

- Frege, Russel, Hilbert: Predicate calculus and type theory as formal basis for mathematics

- Gentzen: Natural deduction (ND) as intuitive formulation of predicate calculus; introduction and elimination rules for each logical connective

The formalization of logical deduction, especially as it has been developed by Frege, Russel, and Hilbert, is rather far removed from the forms of deduction used in practice in mathematical proofs. . . In contrast I intended first to set up a formal system which comes as close as possible to actual reasoning. The result was a calculus of natural deduction (NJ for intuitionist, NK for classical predicate logic).

[Gentzen: Investigations into logical deduction]

Sequent Calculus: Motivation

Sequent Calculus: Motivation

- Gentzen had a pure technical motivation for sequent calculus:

- Gentzen had a pure technical motivation for sequent calculus:
 - ▶ same theorems as natural deduction

- Gentzen had a pure technical motivation for sequent calculus:
 - ▶ same theorems as natural deduction
 - ▶ prove of the Hauptsatz (all sequent proofs can be found with a simple strategy)

- Gentzen had a pure technical motivation for sequent calculus:
 - ▶ same theorems as natural deduction
 - ▶ prove of the Hauptsatz (all sequent proofs can be found with a simple strategy)
 - ▶ corollary: consistency of formal system(s)

Sequent Calculus: Motivation

- Gentzen had a pure technical motivation for sequent calculus:
 - ▶ same theorems as natural deduction
 - ▶ prove of the Hauptsatz (all sequent proofs can be found with a simple strategy)
 - ▶ corollary: consistency of formal system(s)

The Hauptsatz says that every purely logical proof can be reduced to a definite, though not unique, normal form. Perhaps we may express the essential properties of such a normal proof by saying: it is not round-about. . . .

In order to be able to prove the Hauptsatz in a convenient form, I had to provide a logical calculus especially for the purpose. For this the natural calculus proved unsuitable.

[Gentzen: Investigations into logical deduction]

Sequent Calculus: Introduction

- Sequent calculus exposes many details of fine structure of proofs in a very clear manner. Therefore it is well suited to serve as a basic representation formalism for many automation oriented search procedures

- Sequent calculus exposes many details of fine structure of proofs in a very clear manner. Therefore it is well suited to serve as a basic representation formalism for many automation oriented search procedures
 - ▶ Backward: tableaux, connection methods, matrix methods, some forms of resolution

- Sequent calculus exposes many details of fine structure of proofs in a very clear manner. Therefore it is well suited to serve as a basic representation formalism for many automation oriented search procedures
 - ▶ Backward: tableaux, connection methods, matrix methods, some forms of resolution
 - ▶ Forward: classical resolution, inverse method
- Don't be afraid of the many variants of sequent calculi.

- Sequent calculus exposes many details of fine structure of proofs in a very clear manner. Therefore it is well suited to serve as a basic representation formalism for many automation oriented search procedures
 - ▶ Backward: tableaux, connection methods, matrix methods, some forms of resolution
 - ▶ Forward: classical resolution, inverse method

Natural Deduction



Natural Deduction



Natural deduction rules operate on proof trees.

Example:

Natural deduction rules operate on proof trees.

Example:

$$\begin{array}{c} D_1 \quad D_2 \\ \hline A \quad B \\ \hline A \wedge B \end{array} \wedge I \quad \begin{array}{c} D_1 \\ \hline A \wedge B \end{array} \wedge E_l \quad \begin{array}{c} D_1 \\ \hline A \wedge B \end{array} \wedge E_r$$

► Conjunction:

Natural Deduction



Natural deduction rules operate on proof trees.

Example:

► Conjunction:

$$\begin{array}{c} D_1 \quad D_2 \\ \hline A \quad B \\ \hline A \wedge B \end{array} \wedge I \quad \begin{array}{c} D_1 \\ \hline A \wedge B \end{array} \wedge E_l \quad \begin{array}{c} D_1 \\ \hline B \\ \hline A \wedge B \end{array} \wedge E_r$$

Natural Deduction



Natural deduction rules operate on proof trees.

Example:

► Conjunction:

$$\begin{array}{c} D_1 \quad D_2 \\ \hline A \quad B \\ \hline A \wedge B \end{array} \wedge I \quad \begin{array}{c} D_1 \\ \hline A \wedge B \end{array} \wedge E_l \quad \begin{array}{c} D_1 \\ \hline B \\ \hline A \wedge B \end{array} \wedge E_r$$

The presentation on the next slides treats the proof tree aspects implicit.

Example:

The presentation on the next slides treats the proof tree aspects implicit.

Example:

► Conjunction:

$$\begin{array}{c} A \quad B \\ \hline A \wedge B \end{array} \wedge I \quad \begin{array}{c} D_1 \\ \hline A \wedge B \end{array} \wedge E_l \quad \begin{array}{c} D_1 \\ \hline B \\ \hline A \wedge B \end{array} \wedge E_r$$

Natural Deduction Rules Ia



Natural Deduction Rules Ia



■ Conjunction:

$$\frac{A \quad B}{A \wedge B} \wedge I \quad \frac{A \wedge B}{A} \wedge E_I \quad \frac{A \wedge B}{B} \wedge E_r$$

■ Conjunction:

$$\frac{A \quad B}{A \wedge B} \wedge I \quad \frac{A \wedge B}{A} \wedge E_I \quad \frac{A \wedge B}{B} \wedge E_r$$

$$[A]_1 \quad [B]_2$$

$$\text{■ Disjunction: } \frac{A}{A \vee B} \vee I_l \quad \frac{B}{A \vee B} \vee I_r \quad \frac{A \vee B \quad C}{C} \frac{C}{C} \vee E^{1,2}$$

Natural Deduction Rules Ia



■ Conjunction:

$$\frac{A \quad B}{A \wedge B} \wedge I \quad \frac{A \wedge B}{A} \wedge E_I \quad \frac{A \wedge B}{B} \wedge E_r$$

$$[A]_1 \quad [B]_2$$

■ Disjunction:

$$\frac{A}{A \vee B} \vee I_l \quad \frac{B}{A \vee B} \vee I_r \quad \frac{A \vee B \quad C}{C} \frac{C}{C} \vee E^{1,2}$$

$$[A]_1$$

■ Implication:

$$\frac{B}{A \Rightarrow B} \Rightarrow I^1 \quad \frac{A \Rightarrow B \quad A}{B} \Rightarrow E$$

Natural Deduction Rules Ia



■ Conjunction:

$$\frac{A \quad B}{A \wedge B} \wedge I \quad \frac{A \wedge B}{A} \wedge E_I \quad \frac{A \wedge B}{B} \wedge E_r$$

$$[A]_1 \quad [B]_2$$

$$\text{■ Disjunction: } \frac{A}{A \vee B} \vee I_l \quad \frac{B}{A \vee B} \vee I_r \quad \frac{A \vee B \quad C}{C} \frac{C}{C} \vee E^{1,2}$$

$$[A]_1$$

■ Implication:

$$\frac{B}{A \Rightarrow B} \Rightarrow I^1 \quad \frac{A \Rightarrow B \quad A}{B} \Rightarrow E$$

$$\perp \top I \quad \frac{}{C} \perp E$$

■ Negation:

$$\frac{[A]_1 \vdots \perp}{\neg A} \neg I^1 \quad \frac{\neg A \quad A}{\perp} \neg E$$

■ Negation:

■ Universal Quantif.:

$$\frac{[A]_1 \vdots \perp}{\neg A} \neg I^1 \quad \frac{\neg A \quad A}{\perp} \neg E$$

$$\frac{A[x/P^*] \quad \forall I}{\forall x.A} \quad \frac{\forall x.A \quad A[x/T]}{A[x/T]} \forall E$$

(*: parameter P must be new in context)

■ Negation:

$$\frac{[A]_1 \vdots \perp}{\neg A} \neg I^1 \quad \frac{\neg A \quad A}{\perp} \neg E$$

■ Universal Quantif.:

$$\frac{A[x/P^*] \quad \forall I}{\forall x.A} \quad \frac{\forall x.A \quad A[x/T]}{A[x/T]} \forall E$$

(*: parameter P must be new in context)

■ Existential Quantif.:

$$\frac{A[x/T] \quad \exists I}{\exists x.A} \quad \frac{\exists x.A \quad C}{C} \exists E$$

(*: parameter P must be new in context)

■ For classical logic choose one of the following

Natural Deduction Rules IIIa



Natural Deduction Rules IIIa



- For classical logic choose one of the following

► Excluded Middle

$$\frac{}{A \vee \neg A} XM$$

- For classical logic choose one of the following

► Excluded Middle

$$\frac{}{A \vee \neg A} XM$$

► Double Negation

$$\frac{\neg\neg A}{A} \neg\neg C$$

Natural Deduction Rules IIIa



Natural Deduction



- For classical logic choose one of the following

► Excluded Middle

$$\frac{}{A \vee \neg A} XM$$

- Structural properties

► Double Negation

$$\frac{\neg\neg A}{A} \neg\neg C$$

► Proof by Contradiction

$$\begin{array}{c} [\neg A] \\ \vdots \\ \bot \\ \hline A \end{array} \perp_c$$

- Structural properties

- ▶ Exchange

hypotheses order is irrelevant

- Structural properties

- ▶ Exchange

hypotheses order is irrelevant

- ▶ Weakening

hypothesis need not be used

- Structural properties

- ▶ Exchange

hypotheses order is irrelevant

- ▶ Weakening

hypothesis need not be used

- ▶ Contraction

hypotheses can be used more than once

$$\frac{\frac{[A]_1 \quad [A]_2}{A \wedge A} \wedge I}{A \Rightarrow (A \wedge A)} \Rightarrow I^2$$

or

$$\frac{\frac{[A]_1 \quad [A]_1}{A \wedge A} \wedge I}{A \Rightarrow (A \wedge A)} \Rightarrow I^2$$

$$\frac{\frac{[A \wedge B]_1 \quad E_r}{B} \wedge I}{\frac{\frac{[A \wedge B]_1}{A} \wedge E_l \quad \frac{C \vee A}{C \vee A} \vee I_r}{B \wedge (C \vee A)} \wedge I}{(A \wedge B) \Rightarrow (B \wedge (C \vee A))} \Rightarrow I^1$$

- FO-Soundness of ND: Let F be a first-order formula such that there is a ND proof of F . Then F is valid. $(\vdash F \Rightarrow \models F)$
 (Proof: Standard textbooks)

- FO-Soundness of ND: Let F be a first-order formula such that there is a ND proof of F . Then F is valid. $(\vdash F \Rightarrow \models F)$
 (Proof: Standard textbooks)
- FO-Completeness of ND: Let F be a valid first-order formula then there is a ND proof of F $(\models F \Rightarrow \vdash F)$.
 (Proof: Standard textbooks)

Idea: Localizing hypotheses; explicit representation of the available assumptions for each formula occurrence in a ND proof:

$$\Gamma \vdash A$$

Γ is a multiset of the (uncanceled) assumptions on which formula A depends. Γ is called context.

Idea: Localizing hypotheses; explicit representation of the available assumptions for each formula occurrence in a ND proof:

$$\Gamma \vdash A$$

Γ is a multiset of the (uncanceled) assumptions on which formula A depends. Γ is called context.

Example proof in context notation:

$$\frac{\overline{A_1 \vdash A} \quad \overline{A_2 \vdash A}}{\overline{A_1, A_2 \vdash A \wedge A}} \wedge I$$

$$\frac{\overline{A_1 \vdash A \Rightarrow (A \wedge A)}}{\vdash A \Rightarrow (A \Rightarrow (A \wedge A))} \Rightarrow I_2$$

$$\frac{}{\vdash A \Rightarrow (A \Rightarrow (A \wedge A))} \Rightarrow I_1$$

Another Idea: Consider sets of assumptions instead of multisets.

$$\Gamma \vdash A$$

Γ is now a set of (uncanceled) assumptions on which formula A depends.

Another Idea: Consider sets of assumptions instead of multisets.

$$\Gamma \vdash A$$

Γ is now a set of (uncanceled) assumptions on which formula A depends.

Example proof:

$$\frac{\overline{A \vdash A} \quad \overline{A \vdash A} \quad \wedge I}{\overline{A \vdash A \wedge A} \quad \Rightarrow I} \frac{}{A \vdash A \Rightarrow (A \wedge A)} \Rightarrow I$$

$$\vdash A \Rightarrow (A \Rightarrow (A \wedge A)) \Rightarrow I$$

Structural properties to ensure

Structural properties to ensure

- ▶ Exchange (hypotheses order is irrelevant)

$$\frac{\Gamma, B, A \vdash C}{\Gamma, A, B \vdash C}$$

Structural properties to ensure

- ▶ Exchange (hypotheses order is irrelevant)

$$\frac{\Gamma, B, A \vdash C}{\Gamma, A, B \vdash C}$$

- ▶ Weakening (hypothesis need not be used)

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}$$

Structural properties to ensure

- ▶ Exchange (hypotheses order is irrelevant)

$$\frac{\Gamma, B, A \vdash C}{\Gamma, A, B \vdash C}$$

- ▶ Weakening (hypothesis need not be used)

$$\frac{\Gamma \vdash C}{\Gamma, A \vdash C}$$

- ▶ Contraction (hypotheses can be used more than once)

$$\frac{\Gamma, A, A \vdash C}{\Gamma, A \vdash C}$$

Natural Deduction Rules Ib

- Hypotheses:

$$\overline{\Gamma, A, \Delta \vdash A}$$

Natural Deduction Rules Ib

- Hypotheses:

$$\overline{\Gamma, A, \Delta \vdash A}$$

- Conjunction:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge I \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge E_l \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge E_r$$

■ Hypotheses:

$$\frac{}{\Gamma, A, \Delta \vdash A}$$

■ Conjunction:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge I \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge E_l \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge E_r$$

■ Disjunction:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \vee B} \vee I_l \quad \frac{\Gamma \vdash B \quad \Gamma \vdash A}{\Gamma \vdash A \vee B} \vee I_r$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \vee E_r$$

■ Hypotheses:

$$\frac{}{\Gamma, A, \Delta \vdash A}$$

■ Conjunction:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \wedge B} \wedge I \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A} \wedge E_l \quad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B} \wedge E_r$$

■ Disjunction:

$$\frac{\Gamma \vdash A \quad \Gamma \vdash B}{\Gamma \vdash A \vee B} \vee I_l \quad \frac{\Gamma \vdash B \quad \Gamma \vdash A}{\Gamma \vdash A \vee B} \vee I_r$$

$$\frac{\Gamma \vdash A \vee B \quad \Gamma, A \vdash C \quad \Gamma, B \vdash C}{\Gamma \vdash C} \vee E_r$$

■ Implication:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B} \Rightarrow I \quad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B} \Rightarrow E$$

■ Truth and Falsehood:

$$\frac{}{\Gamma \vdash T} TI \quad \frac{\Gamma \vdash \perp}{\Gamma \vdash C} \perp E$$

■ Truth and Falsehood:

$$\frac{}{\Gamma \vdash T} TI \quad \frac{\Gamma \vdash \perp}{\Gamma \vdash C} \perp E$$

■ Negation:

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg I \quad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \perp} \neg E$$

- Truth and Falsehood:

$$\frac{}{\Gamma \vdash \top} \top I \quad \frac{\Gamma \vdash \perp}{\Gamma \vdash C} \perp E$$

- Negation:

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg I \quad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \perp} \neg E$$

- Universal Quantif.:

$$\frac{\Gamma \vdash A[x/P^*]}{\Gamma \vdash \forall x.A} \forall I \quad \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[x/T]} \forall E$$

(*: parameter P must be new in context)

- Truth and Falsehood:

$$\frac{}{\Gamma \vdash \top} \top I \quad \frac{\Gamma \vdash \perp}{\Gamma \vdash C} \perp E$$

- Negation:

$$\frac{\Gamma, A \vdash \perp}{\Gamma \vdash \neg A} \neg I \quad \frac{\Gamma \vdash \neg A \quad \Gamma \vdash A}{\Gamma \vdash \perp} \neg E$$

- Universal Quantif.:

$$\frac{\Gamma \vdash A[x/P^*]}{\Gamma \vdash \forall x.A} \forall I \quad \frac{\Gamma \vdash \forall x.A}{\Gamma \vdash A[x/T]} \forall E$$

(*: parameter P must be new in context)

- Existential Quantif.:

$$\frac{\Gamma \vdash A[x/T]}{\Gamma \vdash \exists x.A} \exists I \quad \frac{\Gamma \vdash \exists x.A \quad \Gamma, A[x/P^*] \vdash C}{\Gamma \vdash C} \exists E$$

(*: parameter P must be new in context)

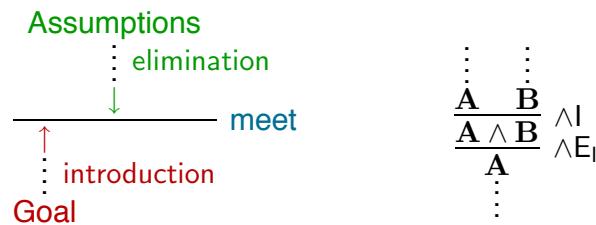
For classical logic add:

- Proof by Contradiction:

$$\frac{\Gamma, \neg A \vdash \perp}{\Gamma \vdash A} \perp_c$$

- Idea (Prawitz, Sieg & Scheines, Byrnes & Sieg): Detour free proofs: strictly use introduction rules bottom up (from proposed theorem to hypothesis) and elimination rules top down (from assumptions to proposed theorem). When they meet in the middle we have found a [proof in normal form](#).

- Idea (Prawitz, Sieg & Scheines, Byrnes & Sieg): Detour free proofs: strictly use introduction rules bottom up (from proposed theorem to hypothesis) and elimination rules top down (from assumptions to proposed theorem). When they meet in the middle we have found a **proof in normal form**.



Intercalating Natural Deductions

Intercalating Natural Deductions

- New annotations:
 - $A \uparrow$: A is obtained by an introduction derivation

- New annotations:
 - $A \uparrow$: A is obtained by an introduction derivation
 - $A \downarrow$: A is extracted from a hypothesis by an elimination derivation

- New annotations:

- ▶ $A \uparrow$: A is obtained by an introduction derivation
- ▶ $A \downarrow$: A is extracted from a hypothesis by an elimination derivation

- Example:

$$\frac{\Gamma, A \vdash_{\text{IC}} B \uparrow}{\Gamma \vdash_{\text{IC}} A \Rightarrow B \uparrow} \Rightarrow I \quad \frac{\Gamma \vdash_{\text{IC}} A \Rightarrow B \downarrow \quad \Gamma \vdash_{\text{IC}} A \uparrow}{\Gamma \vdash_{\text{IC}} B \uparrow} \Rightarrow E$$

ND Intercalation Rules I

- Hypotheses: $\Gamma, A, \Delta \vdash_{\text{IC}} A \downarrow$
- Conjunction:

$$\frac{\Gamma \vdash_{\text{IC}} A \uparrow \quad \Gamma \vdash_{\text{IC}} B \uparrow}{\Gamma \vdash_{\text{IC}} A \wedge B \uparrow} \wedge I \quad \frac{\Gamma \vdash_{\text{IC}} A \wedge B \downarrow}{\Gamma \vdash_{\text{IC}} A \downarrow} \wedge E_I \quad \frac{\Gamma \vdash_{\text{IC}} A \wedge B \downarrow}{\Gamma \vdash_{\text{IC}} B \downarrow} \wedge E_r$$

ND Intercalation Rules I

- Hypotheses:
- Conjunction:

$$\frac{\Gamma \vdash_{\text{IC}} A \uparrow \quad \Gamma \vdash_{\text{IC}} B \uparrow}{\Gamma \vdash_{\text{IC}} A \wedge B \uparrow} \wedge I \quad \frac{\Gamma \vdash_{\text{IC}} A \wedge B \downarrow}{\Gamma \vdash_{\text{IC}} A \downarrow} \wedge E_I \quad \frac{\Gamma \vdash_{\text{IC}} A \wedge B \downarrow}{\Gamma \vdash_{\text{IC}} B \downarrow} \wedge E_r$$

- Disjunction:

$$\frac{\Gamma \vdash_{\text{IC}} A \uparrow}{\Gamma \vdash_{\text{IC}} A \vee B \uparrow} \vee I \quad \frac{\Gamma \vdash_{\text{IC}} B \uparrow}{\Gamma \vdash_{\text{IC}} A \vee B \uparrow} \vee I_r$$

$$\frac{\Gamma \vdash_{\text{IC}} A \vee B \downarrow \quad \Gamma, A \vdash_{\text{IC}} C \uparrow \quad \Gamma, B \vdash_{\text{IC}} C \uparrow}{\Gamma \vdash_{\text{IC}} C \uparrow} \vee E$$

ND Intercalation Rules I



ND Intercalation Rules II



- Hypotheses:

$$\Gamma, A, \Delta \vdash_{\text{IC}} A \downarrow$$

- Conjunction:

$$\frac{\Gamma \vdash_{\text{IC}} A \uparrow \quad \Gamma \vdash_{\text{IC}} B \uparrow}{\Gamma \vdash_{\text{IC}} A \wedge B \uparrow} \wedge I \quad \frac{\Gamma \vdash_{\text{IC}} A \wedge B \downarrow}{\Gamma \vdash_{\text{IC}} A \downarrow} \wedge E_l \quad \frac{\Gamma \vdash_{\text{IC}} A \wedge B \downarrow}{\Gamma \vdash_{\text{IC}} B \downarrow} \wedge E_r$$

- Disjunction:

$$\frac{\Gamma \vdash_{\text{IC}} A \uparrow}{\Gamma \vdash_{\text{IC}} A \vee B \uparrow} \vee I_l \quad \frac{\Gamma \vdash_{\text{IC}} B \uparrow}{\Gamma \vdash_{\text{IC}} A \vee B \uparrow} \vee I_r$$

$$\frac{\Gamma \vdash_{\text{IC}} A \vee B \downarrow \quad \Gamma, A \vdash_{\text{IC}} C \uparrow \quad \Gamma, B \vdash_{\text{IC}} C \uparrow}{\Gamma \vdash_{\text{IC}} C \uparrow} \vee E$$

- Implication:

$$\frac{\Gamma, A \vdash_{\text{IC}} B \uparrow}{\Gamma \vdash_{\text{IC}} A \Rightarrow B \uparrow} \Rightarrow I \quad \frac{\Gamma \vdash_{\text{IC}} A \Rightarrow B \downarrow \quad \Gamma \vdash_{\text{IC}} A \uparrow}{\Gamma \vdash_{\text{IC}} B \uparrow} \Rightarrow E$$



©Benzmüller, 2006

ATPHOL06-[9] – p.250



ND Intercalation Rules II



- Truth and Falsehood:

$$\frac{\Gamma \vdash_{\text{IC}} \top \uparrow}{\Gamma \vdash_{\text{IC}} \bot \uparrow} \top I \quad \frac{\Gamma \vdash_{\text{IC}} \bot \downarrow}{\Gamma \vdash_{\text{IC}} C \uparrow} \bot E$$

- Negation:

$$\frac{\Gamma, A \vdash_{\text{IC}} \bot \uparrow}{\Gamma \vdash_{\text{IC}} \neg A \uparrow} \neg I \quad \frac{\Gamma \vdash_{\text{IC}} \neg A \downarrow \quad \Gamma \vdash_{\text{IC}} A \uparrow}{\Gamma \vdash_{\text{IC}} \bot \uparrow} \neg E$$

ND Intercalation Rules II



- Truth and Falsehood:

$$\frac{\Gamma \vdash_{\text{IC}} \top \uparrow}{\Gamma \vdash_{\text{IC}} \bot \uparrow} \top I \quad \frac{\Gamma \vdash_{\text{IC}} \bot \downarrow}{\Gamma \vdash_{\text{IC}} C \uparrow} \bot E$$

- Negation:

$$\frac{\Gamma, A \vdash_{\text{IC}} \bot \uparrow}{\Gamma \vdash_{\text{IC}} \neg A \uparrow} \neg I \quad \frac{\Gamma \vdash_{\text{IC}} \neg A \downarrow \quad \Gamma \vdash_{\text{IC}} A \uparrow}{\Gamma \vdash_{\text{IC}} \bot \uparrow} \neg E$$

- Universal Quantif.:

$$\frac{\Gamma \vdash_{\text{IC}} A[x/P^*] \uparrow}{\Gamma \vdash_{\text{IC}} \forall x.A \uparrow} \forall I \quad \frac{\Gamma \vdash_{\text{IC}} \forall x.A \downarrow}{\Gamma \vdash_{\text{IC}} A[x/T] \downarrow} \forall E$$

(*: parameter P must be new in context)



©Benzmüller, 2006

ATPHOL06-[9] – p.251



ATPHOL06-[9] – p.251

- Truth and Falsehood:

$$\frac{}{\Gamma \vdash_{\text{IC}} \top \uparrow} \top I \quad \frac{\Gamma \vdash_{\text{IC}} \perp \downarrow}{\Gamma \vdash_{\text{IC}} C \uparrow} \perp E$$

- Negation:

$$\frac{\Gamma, A \vdash_{\text{IC}} \perp \uparrow}{\Gamma \vdash_{\text{IC}} \neg A \uparrow} \neg I \quad \frac{\Gamma \vdash_{\text{IC}} \neg A \downarrow \quad \Gamma \vdash_{\text{IC}} A \uparrow}{\Gamma \vdash_{\text{IC}} \perp \uparrow} \neg E$$

- Universal Quantif.:

$$\frac{\Gamma \vdash_{\text{IC}} A[x/P^*] \uparrow}{\Gamma \vdash_{\text{IC}} \forall x.A \uparrow} \forall I \quad \frac{\Gamma \vdash_{\text{IC}} \forall x.A \downarrow}{\Gamma \vdash_{\text{IC}} A[x/T] \downarrow} \forall E$$

(*: parameter P must be new in context)

- Existential Quantif.:

$$\frac{\Gamma \vdash_{\text{IC}} A[x/T] \uparrow}{\Gamma \vdash_{\text{IC}} \exists x.A \uparrow} \exists I \quad \frac{\Gamma \vdash_{\text{IC}} \exists x.A \downarrow \quad \Gamma, A[x/P^*] \vdash_{\text{IC}} C \uparrow}{\Gamma \vdash C \uparrow} \exists E$$

(*: parameter P must be new in context)

Intercalation and ND

- Normal form proofs

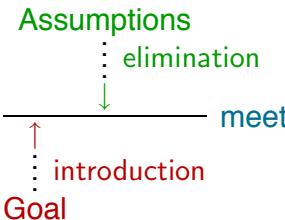
For classical logic add:

- Proof by Contradiction:

$$\frac{\Gamma, \neg A \vdash_{\text{IC}} \perp \uparrow}{\Gamma \vdash_{\text{IC}} A \uparrow} \perp_c$$

Intercalation and ND

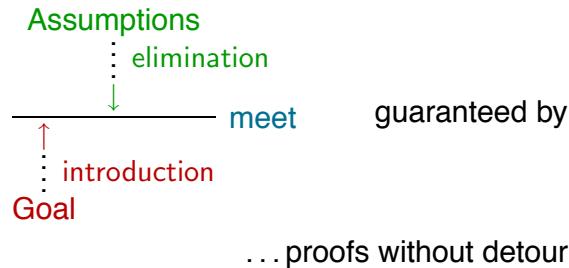
- Normal form proofs



guaranteed by

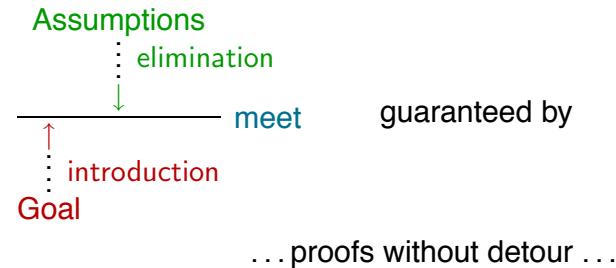
$$\frac{\Gamma \vdash_{\text{IC}} A \downarrow}{\Gamma \vdash_{\text{IC}} A \uparrow} \text{meet}$$

- Normal form proofs



$$\frac{\Gamma \vdash_{\text{IC}} A \downarrow}{\Gamma \vdash_{\text{IC}} A \uparrow} \text{meet}$$

- Normal form proofs



$$\frac{\Gamma \vdash_{\text{IC}} A \downarrow}{\Gamma \vdash_{\text{IC}} A \uparrow} \text{meet}$$

- To model all ND proofs add

$$\frac{\Gamma \vdash_{\text{IC}} A \uparrow}{\Gamma \vdash_{\text{IC}} A \downarrow} \text{roundabout}$$

Example Proofs

- In normal form

$$\frac{\frac{\frac{\frac{M \wedge Q \vdash_{\text{IC}} M \wedge Q \downarrow}{M \wedge Q \vdash_{\text{IC}} Q \downarrow} \wedge E_r}{M \wedge Q \vdash_{\text{IC}} Q \uparrow} \text{meet}}{\frac{M \wedge Q \vdash_{\text{IC}} Q \uparrow}{M \wedge Q \vdash_{\text{IC}} Q \vee S \uparrow} \vee I_l}}{M \wedge Q \vdash_{\text{IC}} (M \wedge Q) \Rightarrow (Q \vee S) \uparrow} \Rightarrow I$$

Example Proofs

- In normal form

$$\frac{\frac{\frac{\frac{M \wedge Q \vdash_{\text{IC}} M \wedge Q \downarrow}{M \wedge Q \vdash_{\text{IC}} Q \downarrow} \wedge E_r}{M \wedge Q \vdash_{\text{IC}} Q \uparrow} \text{meet}}{\frac{M \wedge Q \vdash_{\text{IC}} Q \uparrow}{M \wedge Q \vdash_{\text{IC}} Q \vee S \uparrow} \vee I_l}}{M \wedge Q \vdash_{\text{IC}} (M \wedge Q) \Rightarrow (Q \vee S) \uparrow} \Rightarrow I$$

- With detour

$$\frac{\frac{\frac{\frac{M \wedge Q \vdash_{\text{IC}} Q \uparrow \quad M \wedge Q \vdash_{\text{IC}} M \uparrow}{M \wedge Q \vdash_{\text{IC}} Q \wedge M \uparrow} \wedge I}{M \wedge Q \vdash_{\text{IC}} Q \wedge M \downarrow} \text{roundabout}}{\frac{M \wedge Q \vdash_{\text{IC}} Q \downarrow}{M \wedge Q \vdash_{\text{IC}} Q \uparrow} \wedge E_l}}{M \wedge Q \vdash_{\text{IC}} (M \wedge Q) \Rightarrow (Q \vee S) \uparrow} \Rightarrow I$$

Let \vdash_{IC} denote the intercalation calculus with rule **roundabout** and \vdash_{IC} the calculus without this rule.

Let \vdash_{IC} denote the intercalation calculus with rule **roundabout** and \vdash_{IC} the calculus without this rule.

- ▶ Theorem 1 (Soundness of $\Gamma \vdash_{\text{IC}} A \uparrow$ relative to \vdash): If $\Gamma \vdash_{\text{IC}} A \uparrow$ then $\Gamma \vdash A$.

Let \vdash_{IC} denote the intercalation calculus with rule **roundabout** and \vdash_{IC} the calculus without this rule.

- ▶ Theorem 1 (Soundness of $\Gamma \vdash_{\text{IC}} A \uparrow$ relative to \vdash): If $\Gamma \vdash_{\text{IC}} A \uparrow$ then $\Gamma \vdash A$.
- ▶ Theorem 2 (Completeness of $\Gamma \vdash_{\text{IC}} A \uparrow$ relative to \vdash): If $\Gamma \vdash A$ then $\Gamma \vdash_{\text{IC}} A \uparrow$.

Let \vdash_{IC} denote the intercalation calculus with rule **roundabout** and \vdash_{IC} the calculus without this rule.

- ▶ Theorem 1 (Soundness of $\Gamma \vdash_{\text{IC}} A \uparrow$ relative to \vdash): If $\Gamma \vdash_{\text{IC}} A \uparrow$ then $\Gamma \vdash A$.
- ▶ Theorem 2 (Completeness of $\Gamma \vdash_{\text{IC}} A \uparrow$ relative to \vdash): If $\Gamma \vdash A$ then $\Gamma \vdash_{\text{IC}} A \uparrow$.
- ▶ Is normal form proof search also complete?:
If $\Gamma \vdash_{\text{IC}} A \uparrow$ then $\Gamma \vdash_{\text{IC}} A \uparrow$?

We will investigate this question within the sequent calculus.

Let \vdash_{IC} denote the intercalation calculus with rule roundabout and \vdash_{IC} the calculus without this rule.

- ▶ Theorem 1 (Soundness of \vdash_{IC} relative to \vdash): If $\Gamma \vdash_{\text{IC}} A \uparrow$ then $\Gamma \vdash A$.
- ▶ Theorem 2 (Completeness of \vdash_{IC} relative to \vdash): If $\Gamma \vdash A$ then $\Gamma \vdash_{\text{IC}} A \uparrow$.
- ▶ Is normal form proof search also complete?

$$\text{If } \Gamma \vdash_{\text{IC}} A \uparrow \text{ then } \Gamma \vdash_{\text{IC}} A \uparrow ?$$

We will investigate this question within the sequent calculus.

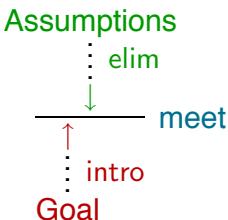
Normal form ND proofs

Sequent proofs

From ND to Sequent Calculus

Normal form ND proofs

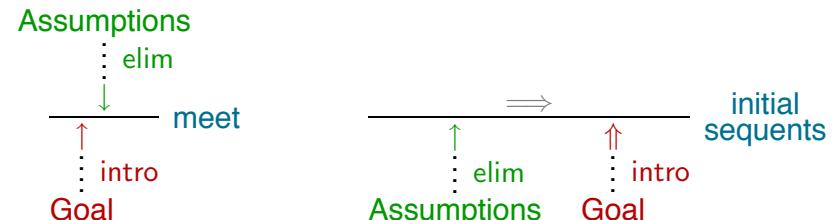
Sequent proofs



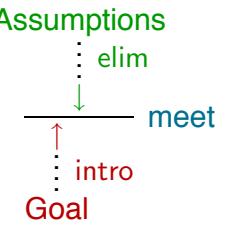
From ND to Sequent Calculus

Normal form ND proofs

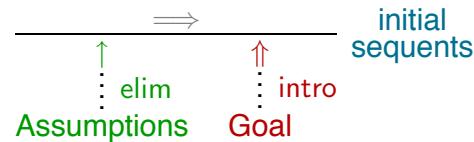
Sequent proofs



Normal form ND proofs



Sequent proofs



Initial Sequents:

$$\frac{}{\Gamma, A \Rightarrow \Delta, A} \text{ init} \quad (A \text{ atomic})$$

Sequents pair $\langle\Gamma, \Delta\rangle$ of finite lists, multisets, or sets of formulas

Notation: $\Gamma \Rightarrow \Delta$ Γ conjunctiv and Δ disjunctive

Intuitive: a kind of implication, Δ “follows from” Γ

Sequent Calculus Rules I

Initial Sequents: $\frac{}{\Gamma, A \Rightarrow \Delta, A} \text{ init} \quad (A \text{ atomic})$

Conjunction:

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta} \wedge L \quad \frac{\Gamma \Rightarrow \Delta, A \quad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B} \wedge R$$

Sequent Calculus Rules I

Initial Sequents: $\frac{}{\Gamma, A \Rightarrow \Delta, A} \text{ init} \quad (A \text{ atomic})$

Conjunction:

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta} \wedge L \quad \frac{\Gamma \Rightarrow \Delta, A \quad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B} \wedge R$$

Implication

$$\frac{\Gamma \Rightarrow \Delta, A \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \Rightarrow B \Rightarrow \Delta} \Rightarrow L \quad \frac{\Gamma, A \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \Rightarrow B} \Rightarrow R$$

Sequent Calculus Rules I



Sequent Calculus Rules II



■ Initial Sequents: $\frac{}{\Gamma, A \Rightarrow \Delta, A} \text{ init}$ (A atomic)

■ Conjunction:

$$\frac{\Gamma, A, B \Rightarrow \Delta}{\Gamma, A \wedge B \Rightarrow \Delta} \wedge L \quad \frac{\Gamma \Rightarrow \Delta, A \quad \Gamma \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \wedge B} \wedge R$$

■ Implication

$$\frac{\Gamma \Rightarrow \Delta, A \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \Rightarrow B \Rightarrow \Delta} \Rightarrow L \quad \frac{\Gamma, A \Rightarrow \Delta, B}{\Gamma \Rightarrow \Delta, A \Rightarrow B} \Rightarrow R$$

■ Truth and Falsehood

$$\frac{}{\Gamma, \perp \Rightarrow \Delta} \perp L \quad \frac{}{\Gamma \Rightarrow \Delta, \top} \top R$$

Sequent Calculus Rules II



■ Negation: $\frac{\Gamma \Rightarrow \Delta, A}{\Gamma, \neg A \Rightarrow \Delta} \neg L \quad \frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A} \neg R$

■ Disjunction:

$$\frac{\Gamma \Rightarrow \Delta, A, B}{\Gamma \Rightarrow \Delta, A \vee B} \vee R \quad \frac{\Gamma, A \Rightarrow \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \vee B \Rightarrow \Delta} \vee L$$

Sequent Calculus Rules II



■ Negation: $\frac{\Gamma \Rightarrow \Delta, A}{\Gamma, \neg A \Rightarrow \Delta} \neg L \quad \frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A} \neg R$

■ Disjunction:

$$\frac{\Gamma \Rightarrow \Delta, A, B}{\Gamma \Rightarrow \Delta, A \vee B} \vee R \quad \frac{\Gamma, A \Rightarrow \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \vee B \Rightarrow \Delta} \vee L$$

■ Universal Quantification:

$$\frac{\Gamma, \forall x.A, A[x/T] \Rightarrow \Delta}{\Gamma, \forall x.A \Rightarrow \Delta} \forall L \quad \frac{\Gamma \Rightarrow \Delta, A[x/P^*]}{\Gamma \Rightarrow \Delta, \forall x.A} \forall R$$

(*: parameter P must be new in context)

■ Negation:

$$\frac{\Gamma \Rightarrow \Delta, A}{\Gamma, \neg A \Rightarrow \Delta} \neg L \quad \frac{\Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta, \neg A} \neg R$$

■ Disjunction:

$$\frac{\Gamma \Rightarrow \Delta, A, B}{\Gamma \Rightarrow \Delta, A \vee B} \vee R \quad \frac{\Gamma, A \Rightarrow \Delta \quad \Gamma, B \Rightarrow \Delta}{\Gamma, A \vee B \Rightarrow \Delta} \vee L$$

■ Universal Quantification:

$$\frac{\Gamma, \forall x.A, A[x/T] \Rightarrow \Delta}{\Gamma, \forall x.A \Rightarrow \Delta} \forall L \quad \frac{\Gamma \Rightarrow \Delta, A[x/P^*]}{\Gamma \Rightarrow \Delta, \forall x.A} \forall R$$

(*: parameter P must be new in context)

■ Existential Quantification:

$$\frac{\Gamma, A[x/P^*] \Rightarrow \Delta}{\Gamma, \exists x.A \Rightarrow \Delta} \exists L \quad \frac{\Gamma \Rightarrow \Delta, \exists x.A, A[x/T]}{\Gamma \Rightarrow \Delta, \exists x.A} \exists R$$

(*: parameter P must be new in context)



$$\frac{\overline{A, B \Rightarrow B} \text{ init} \quad \overline{A, B \Rightarrow C, A} \text{ init}}{\overline{A \wedge B \Rightarrow B} \wedge L \quad \overline{A \wedge B \Rightarrow C \vee A} \vee R} \frac{}{A \wedge B \Rightarrow B \wedge (C \vee A) \wedge R} \\ \Rightarrow (A \wedge B) \Rightarrow B \wedge (C \vee A) \Rightarrow R$$

Sequent Calculus: Cut-rule

- To map natural deductions (in \vdash and \vdash_{ic}) to sequent calculus derivations we add the so called cut-rule:

$$\frac{\Gamma \Rightarrow \Delta, A \quad \Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta} \text{Cut}$$

Sequent Calculus: Cut-rule

- To map natural deductions (in \vdash and \vdash_{ic}) to sequent calculus derivations we add the so called cut-rule:



- To map natural deductions (in \vdash and \vdash_{IC}) to sequent calculus derivations we add the so called cut-rule:

$$\frac{\Gamma \Rightarrow \Delta, A \quad \Gamma, A \Rightarrow \Delta}{\Gamma \Rightarrow \Delta} \text{Cut}$$

- The question whether normal form proof search (\vdash_{IC}) is complete corresponds to the question whether the cut-rule can be eliminated (is *admissible*) in sequent calculus.

Sequent Calculus

Let \Rightarrow^+ denote the sequent calculus with cut-rule and \Rightarrow the sequent calculus without the cut-rule.

Theorem 3 (Soundness of \Rightarrow relative to \vdash_{IC} and \vdash_{IC}^+)

- (a) If $\Gamma \Rightarrow C$ then $\vdash_{\text{IC}} C \uparrow$.
- (b) If $\Gamma \Rightarrow^+ C$ then $\vdash_{\text{IC}}^+ C \uparrow$.

Sequent Calculus

Let \Rightarrow^+ denote the sequent calculus with cut-rule and \Rightarrow the sequent calculus without the cut-rule.

Theorem 3 (Soundness of \Rightarrow relative to \vdash_{IC} and \vdash_{IC}^+)

- (a) If $\Gamma \Rightarrow C$ then $\vdash_{\text{IC}} C \uparrow$.
- (b) If $\Gamma \Rightarrow^+ C$ then $\vdash_{\text{IC}}^+ C \uparrow$.

Theorem 4 (Completeness of \Rightarrow relative to \vdash_{IC} and \vdash_{IC}^+)

- (a) If $\vdash_{\text{IC}} C \uparrow$ then $\Gamma \Rightarrow C$.
- (b) If $\vdash_{\text{IC}}^+ C \uparrow$ then $\Gamma \Rightarrow^+ C$.

Gentzen's Hauptsatz



Gentzen's Hauptsatz



Theorem 5 (Cut-Elimination): Cut-elimination holds for the sequent calculus. In other words: The cut rule is *admissible* in the sequent calculus.

Theorem 5 (Cut-Elimination): Cut-elimination holds for the sequent calculus. In other words: The cut rule is *admissible* in the sequent calculus.

$$\text{If } \Gamma \Rightarrow^+ \Delta \text{ then } \Gamma \Rightarrow \Delta$$

Gentzen's Hauptsatz



Theorem 5 (Cut-Elimination): Cut-elimination holds for the sequent calculus. In other words: The cut rule is *admissible* in the sequent calculus.

$$\text{If } \Gamma \Rightarrow^+ \Delta \text{ then } \Gamma \Rightarrow \Delta$$

Proof non-trivial; main means: nested inductions and case distinctions over rule applications

Gentzen's Hauptsatz



Theorem 5 (Cut-Elimination): Cut-elimination holds for the sequent calculus. In other words: The cut rule is *admissible* in the sequent calculus.

$$\text{If } \Gamma \Rightarrow^+ \Delta \text{ then } \Gamma \Rightarrow \Delta$$

Proof non-trivial; main means: nested inductions and case distinctions over rule applications

This result qualifies the sequent calculus as suitable for automating proof search.

Applications of Cut-Elimination



Theorem (Normalization for ND):

$$\text{If } \Gamma \vdash C \text{ then } \Gamma \vdash_{\text{IC}} C \uparrow.$$

Applications of Cut-Elimination



Theorem (Normalization for ND):

$$\text{If } \Gamma \vdash C \text{ then } \Gamma \vdash_{\text{IC}} C \uparrow.$$

Proof sketch:

Applications of Cut-Elimination



Theorem (Normalization for ND):

$$\text{If } \Gamma \vdash C \text{ then } \Gamma \vdash_{\text{IC}} C \uparrow.$$

Proof sketch:

- ▶ Assume $\Gamma \vdash C$.

Applications of Cut-Elimination



Theorem (Normalization for ND):

$$\text{If } \Gamma \vdash C \text{ then } \Gamma \vdash_{\text{IC}} C \uparrow.$$

Proof sketch:

- ▶ Assume $\Gamma \vdash C$.
- ▶ Then $\Gamma \vdash^{\pm}_{\text{IC}} C \uparrow$ by completeness of \vdash^{\pm}_{IC} .

Theorem (Normalization for ND):

$$\text{If } \Gamma \vdash C \text{ then } \Gamma \vdash_{\text{IC}} C \uparrow.$$

Proof sketch:

- ▶ Assume $\Gamma \vdash C$.
- ▶ Then $\Gamma \vdash_{\text{IC}}^{\pm} C \uparrow$ by completeness of \vdash_{IC}^{\pm} .
- ▶ Then $\Gamma \Rightarrow^{+} C$ by completeness of \Rightarrow^{+} .

Theorem (Normalization for ND):

$$\text{If } \Gamma \vdash C \text{ then } \Gamma \vdash_{\text{IC}} C \uparrow.$$

Proof sketch:

- ▶ Assume $\Gamma \vdash C$.
- ▶ Then $\Gamma \vdash_{\text{IC}}^{\pm} C \uparrow$ by completeness of \vdash_{IC}^{\pm} .
- ▶ Then $\Gamma \Rightarrow^{+} C$ by completeness of \Rightarrow^{+} .
- ▶ Then $\Gamma \Rightarrow C$ by cut-elimination.

Applications of Cut-Elimination

Theorem (Normalization for ND):

$$\text{If } \Gamma \vdash C \text{ then } \Gamma \vdash_{\text{IC}} C \uparrow.$$

Proof sketch:

- ▶ Assume $\Gamma \vdash C$.
- ▶ Then $\Gamma \vdash_{\text{IC}}^{\pm} C \uparrow$ by completeness of \vdash_{IC}^{\pm} .
- ▶ Then $\Gamma \Rightarrow^{+} C$ by completeness of \Rightarrow^{+} .
- ▶ Then $\Gamma \Rightarrow C$ by cut-elimination.
- ▶ Then $\Gamma \vdash_{\text{IC}} C \uparrow$ by soundness of \Rightarrow .

What have we done?

Natural Deduction		
\vdash (with detours)		
	→	

What have we done? _____

Natural Deduction	Intercalation	
\vdash (with detours)	\vdash_{ic} (with roundabout)	
→ → →	→ → →	

What have we done? _____

Natural Deduction	Intercalation	Sequent Calculus
\vdash (with detours)	\vdash_{ic} (with roundabout)	\Rightarrow^+ (with cut)
→ → →	→ → →	→ → ↓

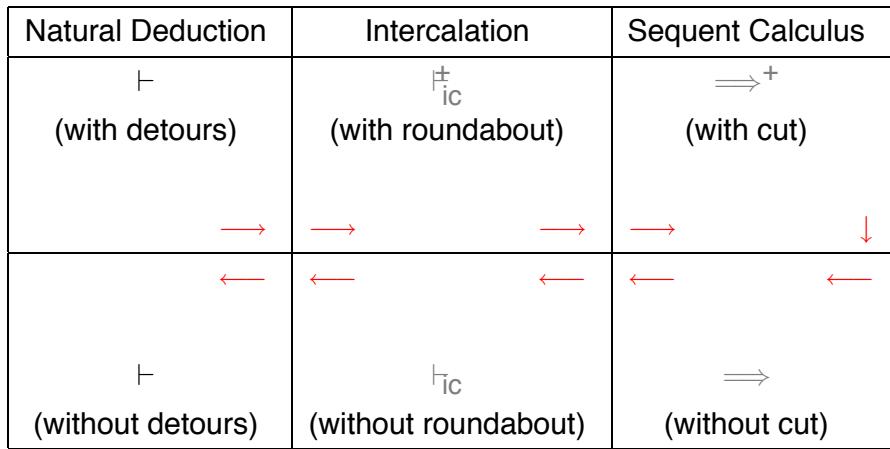
What have we done? _____

Natural Deduction	Intercalation	Sequent Calculus
\vdash (with detours)	\vdash_{ic} (with roundabout)	\Rightarrow^+ (with cut)
→ → →	→ → →	→ → ↓

What have we done? _____

Natural Deduction	Intercalation	Sequent Calculus
\vdash (with detours)	\vdash_{ic} (with roundabout)	\Rightarrow^+ (with cut)
→ → →	→ → →	→ → ↓

Natural Deduction	Intercalation	Sequent Calculus
\vdash (with detours)	\vdash_{ic} (without roundabout)	\Rightarrow (without cut)
← ←	← ←	← ←



Theorem (Consistency of ND): There is no natural deduction derivation $\vdash \perp$.

Applications of Cut-Elimination

Theorem (Consistency of ND): There is no natural deduction derivation $\vdash \perp$.

Proof sketch:

Applications of Cut-Elimination

Theorem (Consistency of ND): There is no natural deduction derivation $\vdash \perp$.

Proof sketch:

- ▶ Assume there is a proof of $\vdash \perp$.

Theorem (Consistency of ND): There is no natural deduction derivation $\vdash \perp$.

Proof sketch:

- ▶ Assume there is a proof of $\vdash \perp$.
- ▶ Then $\Rightarrow^+ \perp$ by completeness of \Rightarrow^+ and \Vdash_{IC}^+ .

Theorem (Consistency of ND): There is no natural deduction derivation $\vdash \perp$.

Proof sketch:

- ▶ Assume there is a proof of $\vdash \perp$.
- ▶ Then $\Rightarrow^+ \perp$ by completeness of \Rightarrow^+ and \Vdash_{IC}^+ .
- ▶ Then $\Rightarrow \perp$ by cut-elimination.

Applications of Cut-Elimination

Theorem (Consistency of ND): There is no natural deduction derivation $\vdash \perp$.

Proof sketch:

- ▶ Assume there is a proof of $\vdash \perp$.
- ▶ Then $\Rightarrow^+ \perp$ by completeness of \Rightarrow^+ and \Vdash_{IC}^+ .
- ▶ Then $\Rightarrow \perp$ by cut-elimination.
- ▶ But $\Rightarrow \perp$ cannot be the conclusion of any sequent rule.

Summary

- We have illustrated the connection of

Summary



Summary



- We have illustrated the connection of
 - ▶ natural deduction and sequent calculus

- We have illustrated the connection of
 - ▶ natural deduction and sequent calculus
 - ▶ normal form natural deductions and cut-free sequent calculus.

Summary



Summary



- We have illustrated the connection of
 - ▶ natural deduction and sequent calculus
 - ▶ normal form natural deductions and cut-free sequent calculus.
- Fact: Sequent calculus often employed as meta-theory for specialized proof search calculi and strategies.

- We have illustrated the connection of
 - ▶ natural deduction and sequent calculus
 - ▶ normal form natural deductions and cut-free sequent calculus.
- Fact: Sequent calculus often employed as meta-theory for specialized proof search calculi and strategies.
- Question: Can these calculi and strategies be transformed to natural deduction proof search?



Calculi: Higher-Order Natural
Deduction

ND Calculi for HOL

Some conventions for this part:

- signature Σ contains only the logical constants \neg, \vee, Π^α unless stated otherwise.
- $\Phi * A := \Phi \cup \{A\}$

ND Calculi for HOL

Some conventions for this part:

- signature Σ contains only the logical constants \neg, \vee, Π^α unless stated otherwise.
- $\Phi * A := \Phi \cup \{A\}$
- context representation of ND calculi

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\frac{\mathbf{A} \in \Phi}{\Phi \Vdash \mathbf{A}} \mathfrak{N}\mathfrak{K}(Hyp)$$



©Benzmüller, 2006

ATPHOL'06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\frac{\mathbf{A} \in \Phi}{\Phi \Vdash \mathbf{A}} \mathfrak{N}\mathfrak{K}(Hyp)$$

$$\frac{\mathbf{A} =_\beta \mathbf{B} \quad \Phi \Vdash \mathbf{A}}{\Phi \Vdash \mathbf{B}} \mathfrak{N}\mathfrak{K}(\beta)$$



©Benzmüller, 2006

ATPHOL'06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\frac{\mathbf{A} \in \Phi}{\Phi \Vdash \mathbf{A}} \mathfrak{N}\mathfrak{K}(Hyp)$$

$$\frac{\mathbf{A} =_\beta \mathbf{B} \quad \Phi \Vdash \mathbf{A}}{\Phi \Vdash \mathbf{B}} \mathfrak{N}\mathfrak{K}(\beta)$$

$$\frac{\Phi * \mathbf{A} \Vdash F_o}{\Phi \Vdash \neg \mathbf{A}} \mathfrak{N}\mathfrak{K}(\neg I)$$



©Benzmüller, 2006

ATPHOL'06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\frac{\mathbf{A} \in \Phi}{\Phi \Vdash \mathbf{A}} \mathfrak{N}\mathfrak{K}(Hyp)$$

$$\frac{\mathbf{A} =_\beta \mathbf{B} \quad \Phi \Vdash \mathbf{A}}{\Phi \Vdash \mathbf{B}} \mathfrak{N}\mathfrak{K}(\beta)$$

$$\frac{\Phi * \mathbf{A} \Vdash F_o}{\Phi \Vdash \neg \mathbf{A}} \mathfrak{N}\mathfrak{K}(\neg I)$$

$$\frac{\Phi \Vdash \neg \mathbf{A} \quad \Phi \Vdash \mathbf{A}}{\Phi \Vdash C} \mathfrak{N}\mathfrak{K}(\neg E)$$



©Benzmüller, 2006

ATPHOL'06-[10] – p.269

ATPHOL'06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\begin{array}{c}
 \frac{A \in \Phi}{\Phi \Vdash A} \mathfrak{N}\mathfrak{K}(Hyp) \quad \frac{A =_\beta B \quad \Phi \Vdash A}{\Phi \Vdash B} \mathfrak{N}\mathfrak{K}(\beta) \\
 \frac{\Phi * A \Vdash F_o}{\Phi \Vdash \neg A} \mathfrak{N}\mathfrak{K}(\neg I) \quad \frac{\Phi \Vdash \neg A \quad \Phi \Vdash A}{\Phi \Vdash C} \mathfrak{N}\mathfrak{K}(\neg E) \\
 \frac{\Phi \Vdash A}{\Phi \Vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_L) \quad \frac{}{\Phi \Vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_R)
 \end{array}$$



©Benzmüller, 2006

ATPHOL06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\begin{array}{c}
 \frac{A \in \Phi}{\Phi \Vdash A} \mathfrak{N}\mathfrak{K}(Hyp) \quad \frac{A =_\beta B \quad \Phi \Vdash A}{\Phi \Vdash B} \mathfrak{N}\mathfrak{K}(\beta) \\
 \frac{\Phi * A \Vdash F_o}{\Phi \Vdash \neg A} \mathfrak{N}\mathfrak{K}(\neg I) \quad \frac{\Phi \Vdash \neg A \quad \Phi \Vdash A}{\Phi \Vdash C} \mathfrak{N}\mathfrak{K}(\neg E) \\
 \frac{\Phi \Vdash A}{\Phi \Vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_L) \quad \frac{\Phi \Vdash B}{\Phi \Vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_R)
 \end{array}$$



©Benzmüller, 2006

ATPHOL06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\begin{array}{c}
 \frac{A \in \Phi}{\Phi \Vdash A} \mathfrak{N}\mathfrak{K}(Hyp) \quad \frac{A =_\beta B \quad \Phi \Vdash A}{\Phi \Vdash B} \mathfrak{N}\mathfrak{K}(\beta) \\
 \frac{\Phi * A \Vdash F_o}{\Phi \Vdash \neg A} \mathfrak{N}\mathfrak{K}(\neg I) \quad \frac{\Phi \Vdash \neg A \quad \Phi \Vdash A}{\Phi \Vdash C} \mathfrak{N}\mathfrak{K}(\neg E) \\
 \frac{\Phi \Vdash A}{\Phi \Vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_L) \quad \frac{\Phi \Vdash B}{\Phi \Vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_R) \\
 \frac{\Phi \Vdash A \vee B \quad \Phi * A \Vdash C \quad \Phi * B \Vdash C}{\Phi \Vdash C} \mathfrak{N}\mathfrak{K}(\vee E)
 \end{array}$$



©Benzmüller, 2006

ATPHOL06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\begin{array}{c}
 \frac{A \in \Phi}{\Phi \Vdash A} \mathfrak{N}\mathfrak{K}(Hyp) \quad \frac{A =_\beta B \quad \Phi \Vdash A}{\Phi \Vdash B} \mathfrak{N}\mathfrak{K}(\beta) \\
 \frac{\Phi * A \Vdash F_o}{\Phi \Vdash \neg A} \mathfrak{N}\mathfrak{K}(\neg I) \quad \frac{\Phi \Vdash \neg A \quad \Phi \Vdash A}{\Phi \Vdash C} \mathfrak{N}\mathfrak{K}(\neg E) \\
 \frac{\Phi \Vdash A}{\Phi \Vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_L) \quad \frac{\Phi \Vdash B}{\Phi \Vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_R) \\
 \frac{\Phi \Vdash A \vee B \quad \Phi * A \Vdash C \quad \Phi * B \Vdash C}{\Phi \Vdash C} \mathfrak{N}\mathfrak{K}(\vee E) \\
 \frac{\Phi \Vdash Gw_\alpha \quad w \text{ new parameter}}{\Phi \Vdash \Pi^\alpha G} \mathfrak{N}\mathfrak{K}(\Pi)^w
 \end{array}$$



©Benzmüller, 2006

ATPHOL06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\begin{array}{c}
 \frac{A \in \Phi}{\Phi \vdash A} \mathfrak{N}\mathfrak{K}(Hyp) \quad \frac{A =_\beta B \quad \Phi \vdash A}{\Phi \vdash B} \mathfrak{N}\mathfrak{K}(\beta) \\
 \\
 \frac{\Phi * A \vdash F_o}{\Phi \vdash \neg A} \mathfrak{N}\mathfrak{K}(\neg I) \quad \frac{\Phi \vdash \neg A \quad \Phi \vdash A}{\Phi \vdash C} \mathfrak{N}\mathfrak{K}(\neg E) \\
 \\
 \frac{\Phi \vdash A}{\Phi \vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_L) \quad \frac{\Phi \vdash B}{\Phi \vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_R) \\
 \\
 \frac{\Phi \vdash A \vee B \quad \Phi * A \vdash C \quad \Phi * B \vdash C}{\Phi \vdash C} \mathfrak{N}\mathfrak{K}(\vee E) \\
 \\
 \frac{\Phi \vdash Gw_\alpha \quad w \text{ new parameter}}{\Phi \vdash \Pi^\alpha G} \mathfrak{N}\mathfrak{K}(\Pi)^w \\
 \\
 \frac{\Phi \vdash \Pi^\alpha G}{\Phi \vdash GA} \mathfrak{N}\mathfrak{K}(\Pi E)
 \end{array}$$

©Benzmüller, 2006



ATPHOL'06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\frac{\Phi \vdash A \wedge B}{\Phi \vdash A} \mathfrak{N}\mathfrak{K}(\wedge E_L)$$

©Benzmüller, 2006



ATPHOL'06-[10] – p.270

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$

$$\begin{array}{c}
 \frac{A \in \Phi}{\Phi \vdash A} \mathfrak{N}\mathfrak{K}(Hyp) \quad \frac{A =_\beta B \quad \Phi \vdash A}{\Phi \vdash B} \mathfrak{N}\mathfrak{K}(\beta) \\
 \\
 \frac{\Phi * A \vdash F_o}{\Phi \vdash \neg A} \mathfrak{N}\mathfrak{K}(\neg I) \quad \frac{\Phi \vdash \neg A \quad \Phi \vdash A}{\Phi \vdash C} \mathfrak{N}\mathfrak{K}(\neg E) \\
 \\
 \frac{\Phi \vdash A}{\Phi \vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_L) \quad \frac{\Phi \vdash B}{\Phi \vdash A \vee B} \mathfrak{N}\mathfrak{K}(\vee I_R) \\
 \\
 \frac{\Phi \vdash A \vee B \quad \Phi * A \vdash C \quad \Phi * B \vdash C}{\Phi \vdash C} \mathfrak{N}\mathfrak{K}(\vee E) \\
 \\
 \frac{\Phi \vdash Gw_\alpha \quad w \text{ new parameter}}{\Phi \vdash \Pi^\alpha G} \mathfrak{N}\mathfrak{K}(\Pi)^w \\
 \\
 \frac{\Phi \vdash \Pi^\alpha G}{\Phi \vdash GA} \mathfrak{N}\mathfrak{K}(\Pi E) \quad \frac{\Phi * \neg A \vdash F_o}{\Phi \vdash A} \mathfrak{N}\mathfrak{K}(Contr)
 \end{array}$$

©Benzmüller, 2006



ATPHOL'06-[10] – p.269

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\frac{\Phi \vdash A \wedge B}{\Phi \vdash A} \mathfrak{N}\mathfrak{K}(\wedge E_L) \quad \frac{\Phi \vdash A \wedge B}{\Phi \vdash B} \mathfrak{N}\mathfrak{K}(\wedge E_R)$$

©Benzmüller, 2006



ATPHOL'06-[10] – p.270

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\frac{\Phi \models A \wedge B}{\Phi \models A} \mathfrak{N}\mathfrak{K}(\wedge E_L) \quad \frac{\Phi \models A \wedge B}{\Phi \models B} \mathfrak{N}\mathfrak{K}(\wedge E_R) \quad \frac{\Phi \models A \quad \Phi \models B}{\Phi \models A \wedge B} \mathfrak{N}\mathfrak{K}(\wedge I)$$

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\frac{\Phi \models A \wedge B}{\Phi \models A} \mathfrak{N}\mathfrak{K}(\wedge E_L) \quad \frac{\Phi \models A \wedge B}{\Phi \models B} \mathfrak{N}\mathfrak{K}(\wedge E_R) \quad \frac{\Phi \models A \quad \Phi \models B}{\Phi \models A \wedge B} \mathfrak{N}\mathfrak{K}(\wedge I)$$

$$\frac{\Phi \models A \Rightarrow B \quad \Phi \models A}{\Phi \models B} \mathfrak{N}\mathfrak{K}(\Rightarrow E)$$

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\frac{\Phi \models A \wedge B}{\Phi \models A} \mathfrak{N}\mathfrak{K}(\wedge E_L) \quad \frac{\Phi \models A \wedge B}{\Phi \models B} \mathfrak{N}\mathfrak{K}(\wedge E_R) \quad \frac{\Phi \models A \quad \Phi \models B}{\Phi \models A \wedge B} \mathfrak{N}\mathfrak{K}(\wedge I)$$

$$\frac{\Phi \models A \Rightarrow B \quad \Phi \models A}{\Phi \models B} \mathfrak{N}\mathfrak{K}(\Rightarrow E) \quad \frac{\Phi, A \models B}{\Phi \models A \Rightarrow B} \mathfrak{N}\mathfrak{K}(\Rightarrow I)$$

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\frac{\Phi \models A \wedge B}{\Phi \models A} \mathfrak{N}\mathfrak{K}(\wedge E_L) \quad \frac{\Phi \models A \wedge B}{\Phi \models B} \mathfrak{N}\mathfrak{K}(\wedge E_R) \quad \frac{\Phi \models A \quad \Phi \models B}{\Phi \models A \wedge B} \mathfrak{N}\mathfrak{K}(\wedge I)$$

$$\frac{\Phi \models A \Rightarrow B \quad \Phi \models A}{\Phi \models B} \mathfrak{N}\mathfrak{K}(\Rightarrow E) \quad \frac{\Phi, A \models B}{\Phi \models A \Rightarrow B} \mathfrak{N}\mathfrak{K}(\Rightarrow I)$$

$$\frac{\Phi \models \mathbf{GT}_\alpha}{\Phi \models \Sigma^\alpha \mathbf{G}} \mathfrak{N}\mathfrak{K}(\Sigma I)$$

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\begin{array}{c}
 \frac{\Phi \models A \wedge B \quad \mathfrak{N}\mathfrak{K}(\wedge E_L)}{\Phi \models A} \quad \frac{\Phi \models A \wedge B \quad \mathfrak{N}\mathfrak{K}(\wedge E_R)}{\Phi \models B} \quad \frac{\Phi \models A \quad \Phi \models B \quad \mathfrak{N}\mathfrak{K}(\wedge I)}{\Phi \models A \wedge B} \\
 \\
 \frac{\Phi \models A \Rightarrow B \quad \Phi \models A \quad \mathfrak{N}\mathfrak{K}(\Rightarrow E)}{\Phi \models B} \quad \frac{\Phi, A \models B \quad \mathfrak{N}\mathfrak{K}(\Rightarrow I)}{\Phi \models A \Rightarrow B} \\
 \\
 \frac{\Phi \models GT_\alpha \quad \mathfrak{N}\mathfrak{K}(\Sigma I)}{\Phi \models \Sigma^\alpha G} \quad \frac{\Phi \models \Sigma^\alpha G \quad \Phi * Gw_\alpha \vdash C \quad w \text{ new parameter}}{\Phi \vdash C} \quad \mathfrak{N}\mathfrak{K}(\Sigma E)
 \end{array}$$

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\begin{array}{c}
 \frac{\Phi \models A \wedge B \quad \mathfrak{N}\mathfrak{K}(\wedge E_L)}{\Phi \models A} \quad \frac{\Phi \models A \wedge B \quad \mathfrak{N}\mathfrak{K}(\wedge E_R)}{\Phi \models B} \quad \frac{\Phi \models A \quad \Phi \models B \quad \mathfrak{N}\mathfrak{K}(\wedge I)}{\Phi \models A \wedge B} \\
 \\
 \frac{\Phi \models A \Rightarrow B \quad \Phi \models A \quad \mathfrak{N}\mathfrak{K}(\Rightarrow E)}{\Phi \models B} \quad \frac{\Phi, A \models B \quad \mathfrak{N}\mathfrak{K}(\Rightarrow I)}{\Phi \models A \Rightarrow B} \\
 \\
 \frac{\Phi \models GT_\alpha \quad \mathfrak{N}\mathfrak{K}(\Sigma I)}{\Phi \models \Sigma^\alpha G} \quad \frac{\Phi \models \Sigma^\alpha G \quad \Phi * Gw_\alpha \vdash C \quad w \text{ new parameter}}{\Phi \vdash C} \quad \mathfrak{N}\mathfrak{K}(\Sigma E) \\
 \\
 \frac{\Phi \vdash T =^\alpha W \quad \Phi \vdash A[T]}{\Phi \vdash A[W]} \quad \mathfrak{N}\mathfrak{K}(= Subst) \\
 \\
 \frac{}{\Phi \vdash A = A} \quad \mathfrak{N}\mathfrak{K}(= Refl)
 \end{array}$$

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\begin{array}{c}
 \frac{\Phi \models A \wedge B \quad \mathfrak{N}\mathfrak{K}(\wedge E_L)}{\Phi \models A} \quad \frac{\Phi \models A \wedge B \quad \mathfrak{N}\mathfrak{K}(\wedge E_R)}{\Phi \models B} \quad \frac{\Phi \models A \quad \Phi \models B \quad \mathfrak{N}\mathfrak{K}(\wedge I)}{\Phi \models A \wedge B} \\
 \\
 \frac{\Phi \models A \Rightarrow B \quad \Phi \models A \quad \mathfrak{N}\mathfrak{K}(\Rightarrow E)}{\Phi \models B} \quad \frac{\Phi, A \models B \quad \mathfrak{N}\mathfrak{K}(\Rightarrow I)}{\Phi \models A \Rightarrow B} \\
 \\
 \frac{\Phi \models GT_\alpha \quad \mathfrak{N}\mathfrak{K}(\Sigma I)}{\Phi \models \Sigma^\alpha G} \quad \frac{\Phi \models \Sigma^\alpha G \quad \Phi * Gw_\alpha \vdash C \quad w \text{ new parameter}}{\Phi \vdash C} \quad \mathfrak{N}\mathfrak{K}(\Sigma E) \\
 \\
 \frac{\Phi \vdash T =^\alpha W \quad \Phi \vdash A[T]}{\Phi \vdash A[W]} \quad \mathfrak{N}\mathfrak{K}(= Subst) \quad \frac{}{\Phi \vdash A = A} \quad \mathfrak{N}\mathfrak{K}(= Refl)
 \end{array}$$

ND Calculi for HOL



Inference rules for $\mathfrak{N}\mathfrak{K}_\beta$ (for richer signatures)

$$\begin{array}{c}
 \frac{\Phi \models A \wedge B \quad \mathfrak{N}\mathfrak{K}(\wedge E_L)}{\Phi \models A} \quad \frac{\Phi \models A \wedge B \quad \mathfrak{N}\mathfrak{K}(\wedge E_R)}{\Phi \models B} \quad \frac{\Phi \models A \quad \Phi \models B \quad \mathfrak{N}\mathfrak{K}(\wedge I)}{\Phi \models A \wedge B} \\
 \\
 \frac{\Phi \models A \Rightarrow B \quad \Phi \models A \quad \mathfrak{N}\mathfrak{K}(\Rightarrow E)}{\Phi \models B} \quad \frac{\Phi, A \models B \quad \mathfrak{N}\mathfrak{K}(\Rightarrow I)}{\Phi \models A \Rightarrow B} \\
 \\
 \frac{\Phi \models GT_\alpha \quad \mathfrak{N}\mathfrak{K}(\Sigma I)}{\Phi \models \Sigma^\alpha G} \quad \frac{\Phi \models \Sigma^\alpha G \quad \Phi * Gw_\alpha \vdash C \quad w \text{ new parameter}}{\Phi \vdash C} \quad \mathfrak{N}\mathfrak{K}(\Sigma E) \\
 \\
 \frac{\Phi \vdash T =^\alpha W \quad \Phi \vdash A[T]}{\Phi \vdash A[W]} \quad \mathfrak{N}\mathfrak{K}(= Subst) \quad \frac{}{\Phi \vdash A = A} \quad \mathfrak{N}\mathfrak{K}(= Refl)
 \end{array}$$

Alternative: Define logical constants $\wedge, \Rightarrow, \Sigma$, etc. in terms of \neg, \vee, Π as usual and strictly use Leibniz equality instead of primitive equality; then the above rules are not needed.

Inference rules for extensionality (rules for $\xi, \eta, \mathfrak{f}, \mathfrak{b}$)

$$\frac{\mathbf{A} \stackrel{\beta\eta}{=} \mathbf{B} \quad \Phi \vdash \mathbf{A}}{\Phi \vdash \mathbf{B}} \mathfrak{N}(\eta)$$

Inference rules for extensionality (rules for $\xi, \eta, \mathfrak{f}, \mathfrak{b}$)

$$\frac{\mathbf{A} \stackrel{\beta\eta}{=} \mathbf{B} \quad \Phi \vdash \mathbf{A} \quad \Phi \vdash \forall x_\alpha.M \stackrel{\beta}{=} N}{\Phi \vdash (\lambda x_\alpha.M) \stackrel{\beta\alpha}{=} (\lambda x_\alpha.N)} \mathfrak{N}(\xi)$$

Inference rules for extensionality (rules for $\xi, \eta, \mathfrak{f}, \mathfrak{b}$)

$$\frac{\mathbf{A} \stackrel{\beta\eta}{=} \mathbf{B} \quad \Phi \vdash \mathbf{A}}{\Phi \vdash \mathbf{B}} \mathfrak{N}(\eta) \quad \frac{\Phi \vdash \forall x_\alpha.M \stackrel{\beta}{=} N}{\Phi \vdash (\lambda x_\alpha.M) \stackrel{\beta\alpha}{=} (\lambda x_\alpha.N)} \mathfrak{N}(\xi)$$

$$\frac{\Phi \vdash \forall x_\alpha.Gx \stackrel{\beta}{=} Hx}{\Phi \vdash G \stackrel{\beta\alpha}{=} H} \mathfrak{N}(\mathfrak{f})$$

Inference rules for extensionality (rules for $\xi, \eta, \mathfrak{f}, \mathfrak{b}$)

$$\frac{\mathbf{A} \stackrel{\beta\eta}{=} \mathbf{B} \quad \Phi \vdash \mathbf{A}}{\Phi \vdash \mathbf{B}} \mathfrak{N}(\eta) \quad \frac{\Phi \vdash \forall x_\alpha.M \stackrel{\beta}{=} N}{\Phi \vdash (\lambda x_\alpha.M) \stackrel{\beta\alpha}{=} (\lambda x_\alpha.N)} \mathfrak{N}(\xi)$$

$$\frac{\Phi \vdash \forall x_\alpha.Gx \stackrel{\beta}{=} Hx}{\Phi \vdash G \stackrel{\beta\alpha}{=} H} \mathfrak{N}(\mathfrak{f})$$

$$\frac{\Phi * A \vdash B \quad \Phi * B \vdash A}{\Phi \vdash A \stackrel{\circ}{=} B} \mathfrak{N}(\mathfrak{b})$$

Inference rules for extensionality (rules for $\xi, \eta, \mathfrak{f}, \mathfrak{b}$)

$$\begin{array}{c}
 \frac{A \stackrel{\beta\eta}{=} B \quad \Phi \vdash A}{\Phi \vdash B} \mathfrak{N}(\eta) \quad \frac{\Phi \vdash \forall x_\alpha.M \stackrel{\beta}{=} N}{\Phi \vdash (\lambda x_\alpha.M) \stackrel{\beta\alpha}{=} (\lambda x_\alpha.N)} \mathfrak{N}(\xi) \\
 \\
 \frac{\Phi \vdash \forall x_\alpha.Gx \stackrel{\beta}{=} Hx}{\Phi \vdash G \stackrel{\beta\alpha}{=} H} \mathfrak{N}(f) \\
 \\
 \frac{\Phi * A \vdash B \quad \Phi * B \vdash A}{\Phi \vdash A \stackrel{\circ}{=} B} \mathfrak{N}(b)
 \end{array}$$

In case of a primitive notion of equality we define respective extensionality rules also for $=$.

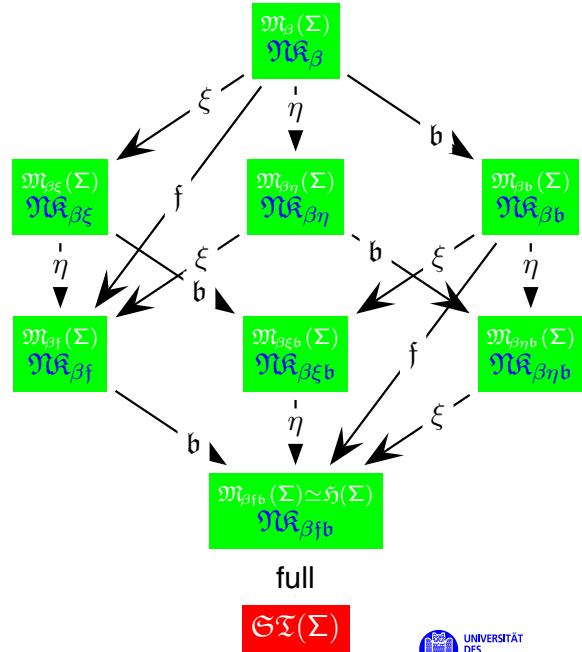
■ The Calculi \mathfrak{N}_*

■ The Calculi \mathfrak{N}_*

- ▶ The calculus \mathfrak{N}_β consists of the inference rules for \mathfrak{N}_β for the provability judgment \Vdash between sets of sentences Φ and sentences A . (We write $\Vdash A$ for $\emptyset \Vdash A$.) The rule $\mathfrak{N}(\beta)$ incorporates β -equality into \Vdash . The others characterize ‘the semantics of the connectives and quantifiers.

■ The Calculi \mathfrak{N}_*

- ▶ The calculus \mathfrak{N}_β consists of the inference rules for \mathfrak{N}_β for the provability judgment \Vdash between sets of sentences Φ and sentences A . (We write $\Vdash A$ for $\emptyset \Vdash A$.) The rule $\mathfrak{N}(\beta)$ incorporates β -equality into \Vdash . The others characterize ‘the semantics of the connectives and quantifiers.
- ▶ For $* \in \{\beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ we obtain the calculus \mathfrak{N}_* by adding the respective extensionality rules when specified in $*$.



- Note that $\mathfrak{N}\mathfrak{R}_\beta$ and $\mathfrak{N}\mathfrak{R}_{\beta f b}$ correspond to the extremes of the model classes in our landscape of model classes. For example, $\mathfrak{N}\mathfrak{R}_{\beta f b}$ will be proven sound and complete for Henkin models, and $\mathfrak{N}\mathfrak{R}_\beta$ will be proven sound and complete for $\mathfrak{M}_\beta(\Sigma)$.
- Standard models do not admit (recursively axiomatizable) calculi that are sound and complete.

- Note that $\mathfrak{N}\mathfrak{R}_\beta$ and $\mathfrak{N}\mathfrak{R}_{\beta f b}$ correspond to the extremes of the model classes in our landscape of model classes. For example, $\mathfrak{N}\mathfrak{R}_{\beta f b}$ will be proven sound and complete for Henkin models, and $\mathfrak{N}\mathfrak{R}_\beta$ will be proven sound and complete for $\mathfrak{M}_\beta(\Sigma)$.

■ (Soundness for \mathfrak{N}_*)

\mathfrak{N}_* is sound for $\mathfrak{M}_*(\Sigma)$ for $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$.

That is, if $\Phi \vdash_{\mathfrak{N}_*} C$ is derivable, then $\mathcal{M} \models C$ for all models $\mathcal{M} = (\mathcal{D}, @, \mathcal{E}, v)$ in $\mathfrak{M}_*(\Sigma)$ such that $\mathcal{M} \models \Phi$.

Proof: ... exercise ...

■ (Soundness for \mathfrak{N}_*)

\mathfrak{N}_* is sound for $\mathfrak{M}_*(\Sigma)$ for $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$.

That is, if $\Phi \vdash_{\mathfrak{N}_*} C$ is derivable, then $\mathcal{M} \models C$ for all models $\mathcal{M} = (\mathcal{D}, @, \mathcal{E}, v)$ in $\mathfrak{M}_*(\Sigma)$ such that $\mathcal{M} \models \Phi$.

Proof: ... exercise ...

■ (Completeness for \mathfrak{N}_*)

Let Φ be a sufficiently Σ -pure set of sentences, A be a sentence, and $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. If A is valid in all models $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ that satisfy Φ , then $\Phi \vdash_{\mathfrak{N}_*} A$.

Proof: ... will follow ...

■ (Soundness for \mathfrak{N}_*)

\mathfrak{N}_* is sound for $\mathfrak{M}_*(\Sigma)$ for $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$.

That is, if $\Phi \vdash_{\mathfrak{N}_*} C$ is derivable, then $\mathcal{M} \models C$ for all models

$\mathcal{M} = (\mathcal{D}, @, \mathcal{E}, v)$ in $\mathfrak{M}_*(\Sigma)$ such that $\mathcal{M} \models \Phi$.

Proof: ... exercise ...

■ (Completeness for \mathfrak{N}_*)

Let Φ be a sufficiently Σ -pure set of sentences, A be a sentence, and $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. If A is valid in all models $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ that satisfy Φ , then $\Phi \vdash_{\mathfrak{N}_*} A$.

Proof: ... will follow ...

Derivation of $\neg(p \vee \neg p) \vdash (p \vee \neg p)$

$$\frac{\frac{\frac{\neg(p \vee \neg p), p \vdash \neg(p \vee \neg p)}{\neg(p \vee \neg p), p \vdash \neg(p \vee \neg p)} \mathfrak{N}(Hyp)}{\neg(p \vee \neg p), p \vdash (p \vee \neg p)} \mathfrak{N}(\vee I_L)}{\neg(p \vee \neg p) \vdash (p \vee \neg p)} \mathfrak{N}(\neg E)}$$

$$\frac{\frac{\frac{\neg(p \vee \neg p), p \vdash F_o}{\neg(p \vee \neg p) \vdash \neg p} \mathfrak{N}(\neg I)}{\neg(p \vee \neg p) \vdash (p \vee \neg p)} \mathfrak{N}(\vee I_R)}{\neg(p \vee \neg p) \vdash (p \vee \neg p)} \mathfrak{N}(\vee I_R)}$$

Completeness (of $\mathcal{N}\mathcal{R}_*$)

Completeness (of $\mathcal{N}\mathcal{R}_*$)

- How to prove completeness?
- Completeness can be proven rather easily for propositional logic calculi.

Completeness (of $\mathcal{N}\mathcal{R}_*$)

- How to prove completeness?
- Completeness can be proven rather easily for propositional logic calculi.
- For first-order and especially higher-order logic completeness proofs become increasingly difficult and technical.

- How to prove completeness?
- Completeness can be proven rather easily for propositional logic calculi.
- For first-order and especially higher-order logic completeness proofs become increasingly difficult and technical.
- Here we will introduce a **strong proof tool** that uniformly supports completeness proofs (and many other things): **abstract consistency**.

- How to prove completeness?
- Completeness can be proven rather easily for propositional logic calculi.
- For first-order and especially higher-order logic completeness proofs become increasingly difficult and technical.
- Here we will introduce a **strong proof tool** that uniformly supports completeness proofs (and many other things): **abstract consistency**.
- This proof tool is based on a strong theorem which connects syntax and semantics: **model existence theorem**.



Abstract Consistency

Abstract Consistency: History

- Technique was developed for first-order logic by Jaakko Hintikka and Raymond Smullyan [Hintikka55,Smullyan63,Smullyan68]. It is well explained in Fitting's textbook [Fitting96].

- Technique was developed for first-order logic by Jaakko Hintikka and Raymond Smullyan [[Hintikka55](#),[Smullyan63](#),[Smullyan68](#)]. It is well explained in Fitting's textbook [[Fitting96](#)].
- The technique has been (partly) extended to higher-order logic by Peter Andrews' in [[Andrews71](#)]; Peter Andrews only achieves a generalization for his rather weak semantical v-complexes (corresponding to our $\mathfrak{M}_\beta(\Sigma)$) and not, for instance, for Henkin Semantics. This extension is well explained in Peter Andrews's textbook [[Andrews02](#)].

- Technique was developed for first-order logic by Jaakko Hintikka and Raymond Smullyan [[Hintikka55](#),[Smullyan63](#),[Smullyan68](#)]. It is well explained in Fitting's textbook [[Fitting96](#)].
- The technique has been (partly) extended to higher-order logic by Peter Andrews' in [[Andrews71](#)]; Peter Andrews only achieves a generalization for his rather weak semantical v-complexes (corresponding to our $\mathfrak{M}_\beta(\Sigma)$) and not, for instance, for Henkin Semantics. This extension is well explained in Peter Andrews's textbook [[Andrews02](#)].
- The technique has been extended to our landscape of HOL model classes in [[Benzmueller-PhD-99](#),[JSL04](#)].

Abstract Consistency: Idea

- A **model existence theorem** for a logical system (i.e., a logical language L together with a consequence relation \models) is a theorem of the form:

Abstract Consistency: Idea

- A **model existence theorem** for a logical system (i.e., a logical language L together with a consequence relation \models) is a theorem of the form:

If a set of sentences Φ of L is a member of an (saturated) abstract consistency class Γ , then there exists a model for Φ .

Abstract Consistency: Idea



- Employing the model existence theorem we can prove completeness of a calculus C (i.e., the derivability rel. \vdash_C) by

Abstract Consistency: Idea



- Employing the model existence theorem we can prove completeness of a calculus C (i.e., the derivability rel. \vdash_C) by proving that the class Γ of sets of sentences Φ that are C -consistent (i.e., cannot be refuted in C) is an (saturated) abstract consistency class.

©Benzmüller, 2006



ATPHOL'06-[11] – p.282

©Benzmüller, 2006



ATPHOL'06-[11] – p.282

Abstract Consistency: Idea



- Employing the model existence theorem we can prove completeness of a calculus C (i.e., the derivability rel. \vdash_C) by proving that the class Γ of sets of sentences Φ that are C -consistent (i.e., cannot be refuted in C) is an (saturated) abstract consistency class.
- Why does this work?

Abstract Consistency: Idea



- Employing the model existence theorem we can prove completeness of a calculus C (i.e., the derivability rel. \vdash_C) by proving that the class Γ of sets of sentences Φ that are C -consistent (i.e., cannot be refuted in C) is an (saturated) abstract consistency class.
- Why does this work?
 - The model existence theorem tells us that C -consistent sets of sentences are satisfiable.

©Benzmüller, 2006



ATPHOL'06-[11] – p.282

©Benzmüller, 2006



ATPHOL'06-[11] – p.282

Abstract Consistency: Idea



- Employing the model existence theorem we can prove completeness of a calculus C (i.e., the derivability rel. \vdash_C) by proving that the class Γ of sets of sentences Φ that are C -consistent (i.e., cannot be refuted in C) is an (saturated) abstract consistency class.
- Why does this work?
 - ▶ The model existence theorem tells us that C -consistent sets of sentences are satisfiable.
 - ▶ Now we assume that a sentence A is valid, so $\neg A$ does not have a model and is therefore C -inconsistent.

Abstract Consistency: Idea



- Employing the model existence theorem we can prove completeness of a calculus C (i.e., the derivability rel. \vdash_C) by proving that the class Γ of sets of sentences Φ that are C -consistent (i.e., cannot be refuted in C) is an (saturated) abstract consistency class.
- Why does this work?
 - ▶ The model existence theorem tells us that C -consistent sets of sentences are satisfiable.
 - ▶ Now we assume that a sentence A is valid, so $\neg A$ does not have a model and is therefore C -inconsistent.
 - ▶ Hence, $\neg A$ is refutable in C .
 - ▶ This shows refutation completeness of C .

Abstract Consistency: Idea



- Employing the model existence theorem we can prove completeness of a calculus C (i.e., the derivability rel. \vdash_C) by proving that the class Γ of sets of sentences Φ that are C -consistent (i.e., cannot be refuted in C) is an (saturated) abstract consistency class.
- Why does this work?
 - ▶ The model existence theorem tells us that C -consistent sets of sentences are satisfiable.
 - ▶ Now we assume that a sentence A is valid, so $\neg A$ does not have a model and is therefore C -inconsistent.
 - ▶ Hence, $\neg A$ is refutable in C .

Abstract Consistency: Idea



- Employing the model existence theorem we can prove completeness of a calculus C (i.e., the derivability rel. \vdash_C) by proving that the class Γ of sets of sentences Φ that are C -consistent (i.e., cannot be refuted in C) is an (saturated) abstract consistency class.
- Why does this work?
 - ▶ The model existence theorem tells us that C -consistent sets of sentences are satisfiable.
 - ▶ Now we assume that a sentence A is valid, so $\neg A$ does not have a model and is therefore C -inconsistent.
 - ▶ Hence, $\neg A$ is refutable in C .
 - ▶ This shows refutation completeness of C .
 - ▶ For many calculi C , this also shows A is provable, thus establishing completeness of C .

Def.: Closed under Subsets / Compact



Let C be a class of sets then C is called **closed under subset** if for all sets S and T it holds that

Def.: Closed under Subsets / Compact



Let C be a class of sets then C is called **closed under subset** if for all sets S and T it holds that

from $S \subseteq T$ and $T \in C$ it follows that $S \in C$.



Let C be a class of sets then C is called **closed under subset** if for all sets S and T it holds that

from $S \subseteq T$ and $T \in C$ it follows that $S \in C$.

Let C be a class of sets. C is called **compact or of finite character** if and only if for every set S holds:



Let C be a class of sets then C is called **closed under subset** if for all sets S and T it holds that

from $S \subseteq T$ and $T \in C$ it follows that $S \in C$.

Let C be a class of sets. C is called **compact or of finite character** if and only if for every set S holds:

$S \in C$ if and only if every finite subset of S is a member of C .

Ex.: Closed under Subsets / Compact



- not closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$

Ex.: Closed under Subsets / Compact



- not closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$
- closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), C\}, \{\neg A, C\}, \{\neg(A \vee B)\}, \{\neg A\}, \{C\}, \{\}\}$

Ex.: Closed under Subsets / Compact



- not closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$
- closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), C\}, \{\neg A, C\}, \{\neg(A \vee B)\}, \{\neg A\}, \{C\}, \{\}\}$
- We define two classes of sets

Ex.: Closed under Subsets / Compact



- not closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$
- closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), C\}, \{\neg A, C\}, \{\neg(A \vee B)\}, \{\neg A\}, \{C\}, \{\}\}$
- We define two classes of sets
 - $C := \{\varphi \mid \varphi \text{ is finite subset of } \mathbb{N}\}$

Ex.: Closed under Subsets / Compact



- not closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$
- closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), C\}, \{\neg A, C\}, \{\neg(A \vee B)\}, \{\neg A\}, \{C\}, \{\}\}$
- We define two classes of sets
 - ▶ $C := \{\varphi \mid \varphi \text{ is finite subset of } \mathbb{N}\}$
 - ▶ $D := 2^{\mathbb{N}}$

Ex.: Closed under Subsets / Compact



- not closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$
- closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), C\}, \{\neg A, C\}, \{\neg(A \vee B)\}, \{\neg A\}, \{C\}, \{\}\}$
- We define two classes of sets
 - ▶ $C := \{\varphi \mid \varphi \text{ is finite subset of } \mathbb{N}\}$
 - ▶ $D := 2^{\mathbb{N}}$
 - ▶ C is closed under subsets but **not** compact.

Ex.: Closed under Subsets / Compact



- not closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg A\}\}$
- closed under subsets: $\{\{\neg(A \vee B), \neg A, C\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), C\}, \{\neg A, C\}, \{\neg(A \vee B)\}, \{\neg A\}, \{C\}, \{\}\}$
- We define two classes of sets
 - ▶ $C := \{\varphi \mid \varphi \text{ is finite subset of } \mathbb{N}\}$
 - ▶ $D := 2^{\mathbb{N}}$
 - ▶ C is closed under subsets but **not** compact.
 - ▶ D is closed under subsets **and** compact.

Lemma: Closed under Subsets / Compact



Lemma:

If C is compact then C is closed under subsets.

Lemma: Closed under Subsets / Compact



Lemma:

If C is compact then C is closed under subsets.

Proof:

©Benzmüller, 2006



ATPHOL06-[11] – p.285

Lemma: Closed under Subsets / Compact



Lemma:

If C is compact then C is closed under subsets.

Proof:

Let $T \in C$ and $S \subseteq T$.

We have to show that $S \in C$.

©Benzmüller, 2006



ATPHOL06-[11] – p.285

Lemma: Closed under Subsets / Compact



Lemma:

If C is compact then C is closed under subsets.

Proof:

Let $T \in C$ and $S \subseteq T$.

©Benzmüller, 2006



ATPHOL06-[11] – p.285

Lemma: Closed under Subsets / Compact



Lemma:

If C is compact then C is closed under subsets.

Proof:

Let $T \in C$ and $S \subseteq T$.

We have to show that $S \in C$.

Every finite subset A of S is also a finite subset of T .

©Benzmüller, 2006



ATPHOL06-[11] – p.285

Lemma: Closed under Subsets / Compact



Lemma:

If C is compact then C is closed under subsets.

Proof:

Let $T \in C$ and $S \subseteq T$.

We have to show that $S \in C$.

Every finite subset A of S is also a finite subset of T .

Since C is compact and $T \in C$ we get that all $A \in C$.

Lemma: Closed under Subsets / Compact



Lemma:

If C is compact then C is closed under subsets.

Proof:

Let $T \in C$ and $S \subseteq T$.

We have to show that $S \in C$.

Every finite subset A of S is also a finite subset of T .

Since C is compact and $T \in C$ we get that all $A \in C$.

Thus, $S \in C$ by compactness.

Def.: Sufficiently Σ -Pure



We introduce a technical side-condition that ensures that we always have enough witness constants.

Def.: Sufficiently Σ -Pure



We introduce a technical side-condition that ensures that we always have enough witness constants.

Let Σ be a signature and Φ be a set of Σ -sentences. Φ is called **sufficiently Σ -pure** if for each type α there is a set $\mathcal{P}_\alpha \subseteq \Sigma_\alpha$ of parameters with equal cardinality to $wff_\alpha(\Sigma)$, such that the elements of \mathcal{P}_α do not occur in the sentences of Φ .

We introduce a technical side-condition that ensures that we always have enough witness constants.

Let Σ be a signature and Φ be a set of Σ -sentences. Φ is called **sufficiently Σ -pure** if for each type α there is a set $\mathcal{P}_\alpha \subseteq \Sigma_\alpha$ of parameters with equal cardinality to $wff_\alpha(\Sigma)$, such that the elements of \mathcal{P}_α do not occur in the sentences of Φ .

This can be obtained in practice by enriching the signature with spurious parameters.

Remember the conventions for this part of the lecture:

- signature Σ contains only the logical constants \neg, \vee, Π^α unless stated otherwise

Abstract Consistency: Conventions

Remember the conventions for this part of the lecture:

- signature Σ contains only the logical constants \neg, \vee, Π^α unless stated otherwise
- as a matter of convenience we will write $\varphi * A$ for $\varphi \cup \{A\}$.

Def.: Abstract Consistency Properties

Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $A, B \in cwf_\alpha(\Sigma)$, $F \in cwf_{\alpha \rightarrow \beta}(\Sigma)$ are arbitrary):

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $\mathbf{A}, \mathbf{B} \in \text{cwff}_o(\Sigma)$, $\mathbf{F} \in \text{cwff}_{\alpha \rightarrow o}(\Sigma)$ are arbitrary):

∇_c If \mathbf{A} is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $\mathbf{A}, \mathbf{B} \in \text{cwff}_o(\Sigma)$, $\mathbf{F} \in \text{cwff}_{\alpha \rightarrow o}(\Sigma)$ are arbitrary):

∇_c If \mathbf{A} is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.

∇_{\neg} If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$.

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $\mathbf{A}, \mathbf{B} \in \text{cwff}_o(\Sigma)$, $\mathbf{F} \in \text{cwff}_{\alpha \rightarrow o}(\Sigma)$ are arbitrary):

∇_c If \mathbf{A} is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.

∇_{\neg} If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$.

∇_β If $\mathbf{A} =_\beta \mathbf{B}$ and $\mathbf{A} \in \Phi$, then $\Phi * \mathbf{B} \in \Gamma_\Sigma$.

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $\mathbf{A}, \mathbf{B} \in \text{cwff}_o(\Sigma)$, $\mathbf{F} \in \text{cwff}_{\alpha \rightarrow o}(\Sigma)$ are arbitrary):

∇_c If \mathbf{A} is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.

∇_{\neg} If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$.

∇_β If $\mathbf{A} =_\beta \mathbf{B}$ and $\mathbf{A} \in \Phi$, then $\Phi * \mathbf{B} \in \Gamma_\Sigma$.

∇_V If $\mathbf{A} \vee \mathbf{B} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$ or $\Phi * \mathbf{B} \in \Gamma_\Sigma$.

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $\mathbf{A}, \mathbf{B} \in \text{cwff}_o(\Sigma)$, $\mathbf{F} \in \text{cwff}_{\alpha \rightarrow o}(\Sigma)$ are arbitrary):

- ∇_c If \mathbf{A} is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.
- ∇_{\neg} If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$.
- ∇_β If $\mathbf{A} =_\beta \mathbf{B}$ and $\mathbf{A} \in \Phi$, then $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
- ∇_V If $\mathbf{A} \vee \mathbf{B} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$ or $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
- ∇_\wedge If $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$, then $\Phi * \neg\mathbf{A} * \neg\mathbf{B} \in \Gamma_\Sigma$.

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $\mathbf{A}, \mathbf{B} \in \text{cwff}_o(\Sigma)$, $\mathbf{F} \in \text{cwff}_{\alpha \rightarrow o}(\Sigma)$ are arbitrary):

- ∇_c If \mathbf{A} is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.
- ∇_{\neg} If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$.
- ∇_β If $\mathbf{A} =_\beta \mathbf{B}$ and $\mathbf{A} \in \Phi$, then $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
- ∇_V If $\mathbf{A} \vee \mathbf{B} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$ or $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
- ∇_\wedge If $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$, then $\Phi * \neg\mathbf{A} * \neg\mathbf{B} \in \Gamma_\Sigma$.
- ∇_W If $\Pi^\alpha \mathbf{F} \in \Phi$, then $\Phi * \mathbf{F} \mathbf{W} \in \Gamma_\Sigma$ for each $\mathbf{W} \in \text{cwff}_\alpha(\Sigma)$.

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $\mathbf{A}, \mathbf{B} \in \text{cwff}_o(\Sigma)$, $\mathbf{F} \in \text{cwff}_{\alpha \rightarrow o}(\Sigma)$ are arbitrary):

- ∇_c If \mathbf{A} is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.
- ∇_{\neg} If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$.
- ∇_β If $\mathbf{A} =_\beta \mathbf{B}$ and $\mathbf{A} \in \Phi$, then $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
- ∇_V If $\mathbf{A} \vee \mathbf{B} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$ or $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
- ∇_\wedge If $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$, then $\Phi * \neg\mathbf{A} * \neg\mathbf{B} \in \Gamma_\Sigma$.
- ∇_W If $\Pi^\alpha \mathbf{F} \in \Phi$, then $\Phi * \mathbf{F} \mathbf{W} \in \Gamma_\Sigma$ for each $\mathbf{W} \in \text{cwff}_\alpha(\Sigma)$.
- ∇_\exists If $\neg\Pi^\alpha \mathbf{F} \in \Phi$, then $\Phi * \neg(\mathbf{F} \mathbf{w}) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ .

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $\mathbf{A}, \mathbf{B} \in \text{cwff}_o(\Sigma)$, $\mathbf{F} \in \text{cwff}_{\alpha \rightarrow o}(\Sigma)$ are arbitrary):

- ∇_c If \mathbf{A} is atomic, then $\mathbf{A} \notin \Phi$ or $\neg\mathbf{A} \notin \Phi$.
- ∇_{\neg} If $\neg\neg\mathbf{A} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$.
- ∇_β If $\mathbf{A} =_\beta \mathbf{B}$ and $\mathbf{A} \in \Phi$, then $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
- ∇_V If $\mathbf{A} \vee \mathbf{B} \in \Phi$, then $\Phi * \mathbf{A} \in \Gamma_\Sigma$ or $\Phi * \mathbf{B} \in \Gamma_\Sigma$.
- ∇_\wedge If $\neg(\mathbf{A} \vee \mathbf{B}) \in \Phi$, then $\Phi * \neg\mathbf{A} * \neg\mathbf{B} \in \Gamma_\Sigma$.
- ∇_W If $\Pi^\alpha \mathbf{F} \in \Phi$, then $\Phi * \mathbf{F} \mathbf{W} \in \Gamma_\Sigma$ for each $\mathbf{W} \in \text{cwff}_\alpha(\Sigma)$.
- ∇_\exists If $\neg\Pi^\alpha \mathbf{F} \in \Phi$, then $\Phi * \neg(\mathbf{F} \mathbf{w}) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ .

(These properties are going back to Hintikka, Smullyan, and Andrews)

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $A, B \in \text{cwff}_o(\Sigma)$, $G, H, (\lambda X_\alpha.M), (\lambda X_\alpha.N) \in \text{cwff}_{\alpha \rightarrow \beta}(\Sigma)$ are arbitrary):

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $A, B \in \text{cwff}_o(\Sigma)$, $G, H, (\lambda X_\alpha.M), (\lambda X_\alpha.N) \in \text{cwff}_{\alpha \rightarrow \beta}(\Sigma)$ are arbitrary):

∇_b If $\neg(A \doteq^\circ B) \in \Phi$, then $\Phi * A * \neg B \in \Gamma_\Sigma$ or $\Phi * \neg A * B \in \Gamma_\Sigma$.



Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $A, B \in \text{cwff}_o(\Sigma)$, $G, H, (\lambda X_\alpha.M), (\lambda X_\alpha.N) \in \text{cwff}_{\alpha \rightarrow \beta}(\Sigma)$ are arbitrary):

∇_b If $\neg(A \doteq^\circ B) \in \Phi$, then $\Phi * A * \neg B \in \Gamma_\Sigma$ or $\Phi * \neg A * B \in \Gamma_\Sigma$.

∇_η If $A \stackrel{\beta\eta}{=} B$ and $A \in \Phi$, then $\Phi * B \in \Gamma_\Sigma$.

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $A, B \in \text{cwff}_o(\Sigma)$, $G, H, (\lambda X_\alpha.M), (\lambda X_\alpha.N) \in \text{cwff}_{\alpha \rightarrow \beta}(\Sigma)$ are arbitrary):

∇_b If $\neg(A \doteq^\circ B) \in \Phi$, then $\Phi * A * \neg B \in \Gamma_\Sigma$ or $\Phi * \neg A * B \in \Gamma_\Sigma$.

∇_η If $A \stackrel{\beta\eta}{=} B$ and $A \in \Phi$, then $\Phi * B \in \Gamma_\Sigma$.

∇_ξ If $\neg(\lambda X_\alpha.M \doteq^{\alpha \rightarrow \beta} \lambda X_\alpha.N) \in \Phi$, then $\Phi * \neg([w/X]M \doteq^\beta [w/X]N) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ .



Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $A, B \in \text{cwff}_o(\Sigma)$, $G, H, (\lambda X_\alpha.M), (\lambda X_\alpha.N) \in \text{cwff}_{\alpha \rightarrow \beta}(\Sigma)$ are arbitrary):

- ∇_b If $\neg(A \doteq^\circ B) \in \Phi$, then $\Phi * A * \neg B \in \Gamma_\Sigma$ or $\Phi * \neg A * B \in \Gamma_\Sigma$.
- ∇_η If $A \stackrel{\beta\eta}{=} B$ and $A \in \Phi$, then $\Phi * B \in \Gamma_\Sigma$.
- ∇_ξ If $\neg(\lambda X_\alpha.M \doteq^{\alpha \rightarrow \beta} \lambda X_\alpha.N) \in \Phi$, then $\Phi * \neg([w/X]M \doteq^\beta [w/X]N) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ .
- ∇_f If $\neg(G \doteq^{\alpha \rightarrow \beta} H) \in \Phi$, then $\Phi * \neg(Gw \doteq^\beta Hw) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ .

Def.: Abstract Consistency Properties



Let Γ_Σ be a class of sets of Σ -sentences. We define (where $\Phi \in \Gamma_\Sigma$, $\alpha, \beta \in \mathcal{T}$, $A, B \in \text{cwff}_o(\Sigma)$, $G, H, (\lambda X_\alpha.M), (\lambda X_\alpha.N) \in \text{cwff}_{\alpha \rightarrow \beta}(\Sigma)$ are arbitrary):

- ∇_b If $\neg(A \doteq^\circ B) \in \Phi$, then $\Phi * A * \neg B \in \Gamma_\Sigma$ or $\Phi * \neg A * B \in \Gamma_\Sigma$.
- ∇_η If $A \stackrel{\beta\eta}{=} B$ and $A \in \Phi$, then $\Phi * B \in \Gamma_\Sigma$.
- ∇_ξ If $\neg(\lambda X_\alpha.M \doteq^{\alpha \rightarrow \beta} \lambda X_\alpha.N) \in \Phi$, then $\Phi * \neg([w/X]M \doteq^\beta [w/X]N) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ .
- ∇_f If $\neg(G \doteq^{\alpha \rightarrow \beta} H) \in \Phi$, then $\Phi * \neg(Gw \doteq^\beta Hw) \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma_\alpha$ which does not occur in any sentence of Φ .

(These properties are new in [Benzmüller-PhD-99, JSL04])

Def.: Abstract Consistency Classes



Let Σ be a signature and Γ_Σ be a class of sets of Σ -sentences that is closed under subsets.

Def.: Abstract Consistency Classes



Let Σ be a signature and Γ_Σ be a class of sets of Σ -sentences that is closed under subsets.

If ∇_c , ∇_\neg , ∇_β , ∇_V , ∇_\wedge , ∇_\forall and ∇_\exists are valid for Γ_Σ , then Γ_Σ is called an **abstract consistency class** for Σ -models.

Def.: Abstract Consistency Classes

Let Σ be a signature and Γ_Σ be a class of sets of Σ -sentences that is closed under subsets.

If $\nabla_c, \nabla_{\neg}, \nabla_\beta, \nabla_V, \nabla_\wedge, \nabla_\forall$ and ∇_\exists are valid for Γ_Σ , then Γ_Σ is called an **abstract consistency class** for Σ -models.

We will denote the collection of abstract consistency classes by \mathfrak{Acc}_β .

Def.: Abstract Consistency Classes

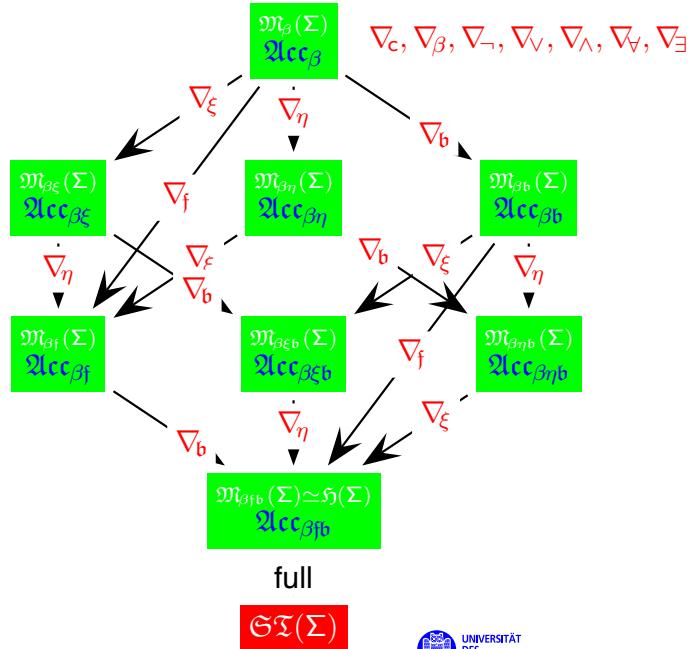
Let Σ be a signature and Γ_Σ be a class of sets of Σ -sentences that is closed under subsets.

If $\nabla_c, \nabla_{\neg}, \nabla_\beta, \nabla_V, \nabla_\wedge, \nabla_\forall$ and ∇_\exists are valid for Γ_Σ , then Γ_Σ is called an **abstract consistency class** for Σ -models.

We will denote the collection of abstract consistency classes by \mathfrak{Acc}_β .

Similarly, we introduce the following collections of specialized abstract consistency classes (with primitive equality): $\mathfrak{Acc}_{\beta\eta}, \mathfrak{Acc}_{\beta\xi}, \mathfrak{Acc}_{\beta f}, \mathfrak{Acc}_{\beta b}, \mathfrak{Acc}_{\beta\eta b}, \mathfrak{Acc}_{\beta\xi b}, \mathfrak{Acc}_{\beta f b}$, where we indicate by indices which additional properties from $\{\nabla_\eta, \nabla_\xi, \nabla_f, \nabla_b\}$ are required.

Abstract Consistency Classes



Ex.: Abstract Consistency Class

- not an abstract consistency class:
 $\{\{\neg(A \vee B), \neg A\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\}\}$

Ex.: Abstract Consistency Class



- not an abstract consistency class:
 $\{\{\neg(A \vee B), \neg A\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\}\}$
- still not:
 $\{\{\neg(A \vee B), \neg A, \neg B\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\}\}$

Ex.: Abstract Consistency Class



- not an abstract consistency class:
 $\{\{\neg(A \vee B), \neg A\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\}\}$
- still not:
 $\{\{\neg(A \vee B), \neg A, \neg B\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\}\}$
- how about this one:
 $\Gamma := \{\{\neg(A \vee B), \neg A, \neg B\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\neg B\}, \{\}\}$

Ex.: Abstract Consistency Class



- not an abstract consistency class:
 $\{\{\neg(A \vee B), \neg A\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\}\}$
- still not:
 $\{\{\neg(A \vee B), \neg A, \neg B\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\}\}$
- how about this one:
 $\Gamma := \{\{\neg(A \vee B), \neg A, \neg B\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\neg B\}, \{\}\}$
- and how about this:
 $\Gamma_0 := \Gamma$
 $\Phi \in \Gamma_i \wedge A \in \Phi \wedge B =_{\beta\eta} A \wedge B \neq A \wedge (\Phi * B) \notin \Gamma_i \longrightarrow$
 $\Gamma_{i+1} := \text{close-under-subsets}(\Gamma_i * (\Phi * B))$
 $\Gamma^* := \Gamma_\infty$

Rem.: Possible Generalization



The work presented here is based on the choice of the primitive logical connectives \neg , \vee and Π^α .

Rem.: Possible Generalization



The work presented here is based on the choice of the primitive logical connectives \neg , \vee and Π^α . A means to generalize the framework over the concrete choice of logical primitives is provided by the uniform notation approach as, for instance, given in [Fitting96].

©Benzmüller, 2006



ATPHOL06-[11] – p.293

Rem.: Possible Generalization



The work presented here is based on the choice of the primitive logical connectives \neg , \vee and Π^α . A means to generalize the framework over the concrete choice of logical primitives is provided by the uniform notation approach as, for instance, given in [Fitting96]. This can be done in straightforward manner: ∇_\wedge becomes an α -property, ∇_\vee becomes a β -property, ∇_\forall becomes a γ -property, and ∇_\exists becomes a δ -property. Thus they will have the following form:

©Benzmüller, 2006



ATPHOL06-[11] – p.293

Rem.: Possible Generalization



The work presented here is based on the choice of the primitive logical connectives \neg , \vee and Π^α . A means to generalize the framework over the concrete choice of logical primitives is provided by the uniform notation approach as, for instance, given in [Fitting96]. This can be done in straightforward manner: ∇_\wedge becomes an α -property, ∇_\vee becomes a β -property, ∇_\forall becomes a γ -property, and ∇_\exists becomes a δ -property.

©Benzmüller, 2006



ATPHOL06-[11] – p.293

Rem.: Possible Generalization



The work presented here is based on the choice of the primitive logical connectives \neg , \vee and Π^α . A means to generalize the framework over the concrete choice of logical primitives is provided by the uniform notation approach as, for instance, given in [Fitting96]. This can be done in straightforward manner: ∇_\wedge becomes an α -property, ∇_\vee becomes a β -property, ∇_\forall becomes a γ -property, and ∇_\exists becomes a δ -property. Thus they will have the following form:

α -case If $\alpha \in \Phi$, then $\Phi * \alpha_1 * \alpha_2 \in \Gamma_\Sigma$.

©Benzmüller, 2006



ATPHOL06-[11] – p.293

Rem.: Possible Generalization



The work presented here is based on the choice of the primitive logical connectives \neg , \vee and Π^α . A means to generalize the framework over the concrete choice of logical primitives is provided by the uniform notation approach as, for instance, given in [Fitting96]. This can be done in straightforward manner: ∇_\wedge becomes an α -property, ∇_\vee becomes a β -property, ∇_\forall becomes a γ -property, and ∇_\exists becomes a δ -property. Thus they will have the following form:

α -case If $\alpha \in \Phi$, then $\Phi * \alpha_1 * \alpha_2 \in \Gamma_\Sigma$.

β -case If $\beta \in \Phi$, then $\Phi * \beta_1 \in \Gamma_\Sigma$ or $\Phi * \beta_2 \in \Gamma_\Sigma$.

Rem.: Possible Generalization



The work presented here is based on the choice of the primitive logical connectives \neg , \vee and Π^α . A means to generalize the framework over the concrete choice of logical primitives is provided by the uniform notation approach as, for instance, given in [Fitting96]. This can be done in straightforward manner: ∇_\wedge becomes an α -property, ∇_\vee becomes a β -property, ∇_\forall becomes a γ -property, and ∇_\exists becomes a δ -property. Thus they will have the following form:

α -case If $\alpha \in \Phi$, then $\Phi * \alpha_1 * \alpha_2 \in \Gamma_\Sigma$.

β -case If $\beta \in \Phi$, then $\Phi * \beta_1 \in \Gamma_\Sigma$ or $\Phi * \beta_2 \in \Gamma_\Sigma$.

γ -case If $\gamma \in \Phi$, then $\Phi * \gamma W \in \Gamma_\Sigma$ for each $W \in \text{cwf}_\alpha(\Sigma)$.

©Benzmüller, 2006



ATPHOL06-[11] – p.293

Rem.: Possible Generalization



The work presented here is based on the choice of the primitive logical connectives \neg , \vee and Π^α . A means to generalize the framework over the concrete choice of logical primitives is provided by the uniform notation approach as, for instance, given in [Fitting96]. This can be done in straightforward manner: ∇_\wedge becomes an α -property, ∇_\vee becomes a β -property, ∇_\forall becomes a γ -property, and ∇_\exists becomes a δ -property. Thus they will have the following form:

α -case If $\alpha \in \Phi$, then $\Phi * \alpha_1 * \alpha_2 \in \Gamma_\Sigma$.

β -case If $\beta \in \Phi$, then $\Phi * \beta_1 \in \Gamma_\Sigma$ or $\Phi * \beta_2 \in \Gamma_\Sigma$.

γ -case If $\gamma \in \Phi$, then $\Phi * \gamma W \in \Gamma_\Sigma$ for each $W \in \text{cwf}_\alpha(\Sigma)$.

δ -case If $\delta \in \Phi$, then $\Phi * \delta w \in \Gamma_\Sigma$ for any parameter $w_\alpha \in \Sigma$ which does not occur in any sentence of Φ .

©Benzmüller, 2006



ATPHOL06-[11] – p.293

©Benzmüller, 2006



ATPHOL06-[11] – p.293

Def.: Saturated



Consider the following property (where $\Phi \in \Gamma_\Sigma$, $A \in \text{cwf}_o(\Sigma)$):



ATPHOL06-[11] – p.294

Def.: Saturated



Consider the following property (where $\Phi \in \Gamma_\Sigma$, $A \in \text{cwff}_o(\Sigma)$):

∇_{sat} Either $\Phi * A \in \Gamma_\Sigma$ or $\Phi * \neg A \in \Gamma_\Sigma$.

Def.: Saturated



Consider the following property (where $\Phi \in \Gamma_\Sigma$, $A \in \text{cwff}_o(\Sigma)$):

∇_{sat} Either $\Phi * A \in \Gamma_\Sigma$ or $\Phi * \neg A \in \Gamma_\Sigma$.

We call an abstract consistency class Γ_Σ **atomically saturated** if ∇_{sat} holds for all atomic A .

Def.: Saturated



Consider the following property (where $\Phi \in \Gamma_\Sigma$, $A \in \text{cwff}_o(\Sigma)$):

∇_{sat} Either $\Phi * A \in \Gamma_\Sigma$ or $\Phi * \neg A \in \Gamma_\Sigma$.

We call an abstract consistency class Γ_Σ **atomically saturated** if ∇_{sat} holds for all atomic A .

We call an abstract consistency class Γ_Σ **saturated** if ∇_{sat} holds for all A .

Ex.: Saturated



■ consider Γ (and Γ^*) from before:

$\{\{\neg(A \vee B), \neg A, \neg B\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), \neg B\}, \{\neg A, \neg B\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\neg B\}, \{\}\}$

Ex.: Saturated



- consider Γ (and Γ^*) from before:
 $\{\{\neg(A \vee B), \neg A, \neg B\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), \neg B\}, \{\neg A, \neg B\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\neg B\}, \{\}\}$
- Γ (and Γ^*) is atomically saturated in case our signature contains no further constants besides A_o and B_o and the logical connectives.

Ex.: Saturated



- consider Γ (and Γ^*) from before:
 $\{\{\neg(A \vee B), \neg A, \neg B\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), \neg B\}, \{\neg A, \neg B\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\neg B\}, \{\}\}$
- Γ (and Γ^*) is atomically saturated in case our signature contains no further constants besides A_o and B_o and the logical connectives.
- if there is another symbol C_o in the signature, then Γ (and Γ^*) is not atomically saturated anymore

©Benzmüller, 2006



ATPHOL06-[11] – p.295

Ex.: Saturated



- consider Γ (and Γ^*) from before:
 $\{\{\neg(A \vee B), \neg A, \neg B\}, \{\neg(A \vee B), \neg A\}, \{\neg(A \vee B), \neg B\}, \{\neg A, \neg B\}, \{\neg(A \vee B)\}, \{\neg A\}, \{\neg B\}, \{\}\}$
- Γ (and Γ^*) is atomically saturated in case our signature contains no further constants besides A_o and B_o and the logical connectives.
- if there is another symbol C_o in the signature, then Γ (and Γ^*) is not atomically saturated anymore
- Γ (and Γ^*) is not saturated: for instance, it does not provide information on the formulas $(\neg A \vee B) \vee A$ and $\Pi^o(\lambda X_o.X)$

Thm.: Model Existence Theorem



Let Γ_Σ be a saturated abstract consistency class and let $\Phi \in \Gamma_\Sigma$ be a sufficiently Σ -pure set of sentences.

For all $* \in \{\beta, \beta\eta, \beta\xi, \beta\f, \beta\b, \beta\eta\b, \beta\xi\b, \beta\f\b\}$ we have:

©Benzmüller, 2006



ATPHOL06-[11] – p.295

©Benzmüller, 2006



ATPHOL06-[11] – p.296

Thm.: Model Existence Theorem



Let Γ_Σ be a saturated abstract consistency class and let $\Phi \in \Gamma_\Sigma$ be a sufficiently Σ -pure set of sentences.

For all $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ we have:

If Γ_Σ is an \mathfrak{Acc}_* , then there exists a model $\mathcal{M} \in \mathfrak{M}_*$ that satisfies Φ .

Thm.: Model Existence Theorem



Let Γ_Σ be a saturated abstract consistency class and let $\Phi \in \Gamma_\Sigma$ be a sufficiently Σ -pure set of sentences.

For all $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ we have:

If Γ_Σ is an \mathfrak{Acc}_* , then there exists a model $\mathcal{M} \in \mathfrak{M}_*$ that satisfies Φ .

Furthermore, each domain of \mathcal{M} has cardinality at most \aleph_0 .

Thm.: Model Existence Theorem



Let Γ_Σ be a saturated abstract consistency class and let $\Phi \in \Gamma_\Sigma$ be a sufficiently Σ -pure set of sentences.

For all $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$ we have:

If Γ_Σ is an \mathfrak{Acc}_* , then there exists a model $\mathcal{M} \in \mathfrak{M}_*$ that satisfies Φ .

Furthermore, each domain of \mathcal{M} has cardinality at most \aleph_0 .

Proof: ... we are not yet ready for this ...

Thm.: Model Existence for Henkin Models



Let Γ_Σ be a saturated abstract consistency class in $\mathfrak{Acc}_{\beta\mathfrak{f}\mathfrak{b}}$ and let $\Phi \in \Gamma_\Sigma$ be a sufficiently Σ -pure set of sentences.

Thm.: Model Existence for Henkin Models



Thm.: Model Existence for Henkin Models



Let Γ_Σ be a saturated abstract consistency class in $\mathfrak{Acc}_{\beta\text{fb}}$ and let $\Phi \in \Gamma_\Sigma$ be a sufficiently Σ -pure set of sentences.

Then there is a Henkin Model that satisfies Φ .

Thm.: Model Existence for Henkin Models



Let Γ_Σ be a saturated abstract consistency class in $\mathfrak{Acc}_{\beta\text{fb}}$ and let $\Phi \in \Gamma_\Sigma$ be a sufficiently Σ -pure set of sentences.

Then there is a Henkin Model that satisfies Φ .

Furthermore, each domain of the model has cardinality at most \aleph_s .

©Benzmüller, 2006



ATPHOL06-[11] – p.297

Thm.: Model Existence for Henkin Models



Let Γ_Σ be a saturated abstract consistency class in $\mathfrak{Acc}_{\beta\text{fb}}$ and let $\Phi \in \Gamma_\Sigma$ be a sufficiently Σ -pure set of sentences.

Then there is a Henkin Model that satisfies Φ .

Furthermore, each domain of the model has cardinality at most \aleph_s .

Proof: ... we are not yet ready for this ...



ATPHOL06-[11] – p.297



Completeness of \mathfrak{N}_* via
Abstract Consistency

©Benzmüller, 2006



ATPHOL06-[11] – p.297

©Benzmüller, 2006



ATPHOL06-[11] – p.298

Def.: \mathfrak{N}_* -Consistent/Inconsistent



A set of sentences Φ is \mathfrak{N}_* -inconsistent if $\Phi \models_{\mathfrak{N}_*} F_o$, and \mathfrak{N}_* -consistent otherwise.

Lemma: Saturated \mathfrak{Acc}_*



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}_*\text{-consistent}\}$ is a saturated \mathfrak{Acc}_* .

Lemma: Saturated \mathfrak{Acc}_*



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}_*\text{-consistent}\}$ is a saturated \mathfrak{Acc}_* .

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an \mathfrak{N}_* -consistent set is \mathfrak{N}_* -consistent.

Lemma: Saturated \mathfrak{Acc}_*



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}_*\text{-consistent}\}$ is a saturated \mathfrak{Acc}_* .

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an \mathfrak{N}_* -consistent set is \mathfrak{N}_* -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}\mathfrak{R}_*\text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an $\mathfrak{N}\mathfrak{R}_*$ -consistent set is $\mathfrak{N}\mathfrak{R}_*$ -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

©Benzmüller, 2006



ATPHOL06-[11] – p.300

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}\mathfrak{R}_*\text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an $\mathfrak{N}\mathfrak{R}_*$ -consistent set is $\mathfrak{N}\mathfrak{R}_*$ -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \vdash \mathbf{F}_o$ by $\mathfrak{N}\mathfrak{R}(\text{Hyp})$ and $\mathfrak{N}\mathfrak{R}(\neg E)$.

©Benzmüller, 2006



ATPHOL06-[11] – p.300

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}\mathfrak{R}_*\text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an $\mathfrak{N}\mathfrak{R}_*$ -consistent set is $\mathfrak{N}\mathfrak{R}_*$ -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$.

©Benzmüller, 2006



ATPHOL06-[11] – p.300

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}\mathfrak{R}_*\text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an $\mathfrak{N}\mathfrak{R}_*$ -consistent set is $\mathfrak{N}\mathfrak{R}_*$ -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \vdash \mathbf{F}_o$ by $\mathfrak{N}\mathfrak{R}(\text{Hyp})$ and $\mathfrak{N}\mathfrak{R}(\neg E)$. Hence Φ is $\mathfrak{N}\mathfrak{R}_*$ -inconsistent.

©Benzmüller, 2006



ATPHOL06-[11] – p.300

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}\mathfrak{R}_*\text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an $\mathfrak{N}\mathfrak{R}_*$ -consistent set is $\mathfrak{N}\mathfrak{R}_*$ -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \vdash F_o$ by $\mathfrak{N}\mathfrak{R}(\text{Hyp})$ and $\mathfrak{N}\mathfrak{R}(\neg E)$.
Hence Φ is $\mathfrak{N}\mathfrak{R}_*$ -inconsistent.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}\mathfrak{R}_*\text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an $\mathfrak{N}\mathfrak{R}_*$ -consistent set is $\mathfrak{N}\mathfrak{R}_*$ -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \vdash F_o$ by $\mathfrak{N}\mathfrak{R}(\text{Hyp})$ and $\mathfrak{N}\mathfrak{R}(\neg E)$.
Hence Φ is $\mathfrak{N}\mathfrak{R}_*$ -inconsistent.

∇_β Let $\mathbf{A} \in \Phi$, $\mathbf{A} =_\beta \mathbf{B}$ and $\Phi * \mathbf{B}$ be $\mathfrak{N}\mathfrak{R}_*$ -inconsistent.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}\mathfrak{R}_*\text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an $\mathfrak{N}\mathfrak{R}_*$ -consistent set is $\mathfrak{N}\mathfrak{R}_*$ -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \vdash F_o$ by $\mathfrak{N}\mathfrak{R}(\text{Hyp})$ and $\mathfrak{N}\mathfrak{R}(\neg E)$.
Hence Φ is $\mathfrak{N}\mathfrak{R}_*$ -inconsistent.

∇_β Let $\mathbf{A} \in \Phi$, $\mathbf{A} =_\beta \mathbf{B}$ and $\Phi * \mathbf{B}$ be $\mathfrak{N}\mathfrak{R}_*$ -inconsistent. That is,
 $\Phi * \mathbf{B} \vdash F_o$.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}\mathfrak{R}_*\text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an $\mathfrak{N}\mathfrak{R}_*$ -consistent set is $\mathfrak{N}\mathfrak{R}_*$ -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \vdash F_o$ by $\mathfrak{N}\mathfrak{R}(\text{Hyp})$ and $\mathfrak{N}\mathfrak{R}(\neg E)$.
Hence Φ is $\mathfrak{N}\mathfrak{R}_*$ -inconsistent.

∇_β Let $\mathbf{A} \in \Phi$, $\mathbf{A} =_\beta \mathbf{B}$ and $\Phi * \mathbf{B}$ be $\mathfrak{N}\mathfrak{R}_*$ -inconsistent. That is,
 $\Phi * \mathbf{B} \vdash F_o$. By $\mathfrak{N}\mathfrak{R}(\neg I)$, we know $\Phi \vdash \neg\mathbf{B}$.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}_* \text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an \mathfrak{N}_* -consistent set is \mathfrak{N}_* -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

- ∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \models F_o$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\neg E)$. Hence Φ is \mathfrak{N}_* -inconsistent.
- ∇_β Let $\mathbf{A} \in \Phi$, $\mathbf{A} =_\beta \mathbf{B}$ and $\Phi * \mathbf{B}$ be \mathfrak{N}_* -inconsistent. That is, $\Phi * \mathbf{B} \models F_o$. By $\mathfrak{N}(\neg I)$, we know $\Phi \models \neg\mathbf{B}$. Since $\mathbf{A} \in \Phi$, we know $\Phi \models \mathbf{B}$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\beta)$.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}_* \text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an \mathfrak{N}_* -consistent set is \mathfrak{N}_* -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

- ∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \models F_o$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\neg E)$. Hence Φ is \mathfrak{N}_* -inconsistent.
- ∇_β Let $\mathbf{A} \in \Phi$, $\mathbf{A} =_\beta \mathbf{B}$ and $\Phi * \mathbf{B}$ be \mathfrak{N}_* -inconsistent. That is, $\Phi * \mathbf{B} \models F_o$. By $\mathfrak{N}(\neg I)$, we know $\Phi \models \neg\mathbf{B}$. Since $\mathbf{A} \in \Phi$, we know $\Phi \models \mathbf{B}$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\beta)$. Using $\mathfrak{N}(\neg E)$, we know $\Phi \models F_o$.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}_* \text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an \mathfrak{N}_* -consistent set is \mathfrak{N}_* -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

- ∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \models F_o$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\neg E)$. Hence Φ is \mathfrak{N}_* -inconsistent.
- ∇_β Let $\mathbf{A} \in \Phi$, $\mathbf{A} =_\beta \mathbf{B}$ and $\Phi * \mathbf{B}$ be \mathfrak{N}_* -inconsistent. That is, $\Phi * \mathbf{B} \models F_o$. By $\mathfrak{N}(\neg I)$, we know $\Phi \models \neg\mathbf{B}$. Since $\mathbf{A} \in \Phi$, we know $\Phi \models \mathbf{B}$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\beta)$. Using $\mathfrak{N}(\neg E)$, we know $\Phi \models F_o$.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}_* \text{-consistent}\}$ is a saturated $\mathfrak{A}\text{cc}_*$.

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an \mathfrak{N}_* -consistent set is \mathfrak{N}_* -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

- ∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \models F_o$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\neg E)$. Hence Φ is \mathfrak{N}_* -inconsistent.
- ∇_β Let $\mathbf{A} \in \Phi$, $\mathbf{A} =_\beta \mathbf{B}$ and $\Phi * \mathbf{B}$ be \mathfrak{N}_* -inconsistent. That is, $\Phi * \mathbf{B} \models F_o$. By $\mathfrak{N}(\neg I)$, we know $\Phi \models \neg\mathbf{B}$. Since $\mathbf{A} \in \Phi$, we know $\Phi \models \mathbf{B}$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\beta)$. Using $\mathfrak{N}(\neg E)$, we know $\Phi \models F_o$.
- ∇_\neg Suppose $\neg\neg\mathbf{A} \in \Phi$ and $\Phi * \mathbf{A}$ is \mathfrak{N}_* -inconsistent.

Lemma: Saturated \mathfrak{Acc}_*



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}_* \text{-consistent}\}$ is a saturated \mathfrak{Acc}_* .

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an \mathfrak{N}_* -consistent set is \mathfrak{N}_* -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \models F_o$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\neg E)$. Hence Φ is \mathfrak{N}_* -inconsistent.

∇_β Let $\mathbf{A} \in \Phi$, $\mathbf{A} =_\beta \mathbf{B}$ and $\Phi * \mathbf{B}$ be \mathfrak{N}_* -inconsistent. That is, $\Phi * \mathbf{B} \models F_o$. By $\mathfrak{N}(\neg I)$, we know $\Phi \models \neg\mathbf{B}$. Since $\mathbf{A} \in \Phi$, we know $\Phi \models \mathbf{B}$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\beta)$. Using $\mathfrak{N}(\neg E)$, we know $\Phi \models F_o$.

∇ Suppose $\neg\neg\mathbf{A} \in \Phi$ and $\Phi * \mathbf{A}$ is \mathfrak{N}_* -inconsistent. From $\Phi * \mathbf{A} \models F_o$ and $\mathfrak{N}(\neg I)$, we have $\Phi \models \neg\mathbf{A}$.



Lemma: Saturated \mathfrak{Acc}_*



Lemma: Saturated \mathfrak{Acc}_*



The class $\Gamma_\Sigma^* := \{\Phi \subseteq \text{cwff}_o(\Sigma) \mid \Phi \text{ is } \mathfrak{N}_* \text{-consistent}\}$ is a saturated \mathfrak{Acc}_* .

Proof: Obviously Γ_Σ^* is closed under subsets, since any subset of an \mathfrak{N}_* -consistent set is \mathfrak{N}_* -consistent. We now check the remaining conditions. We prove all the properties by proving their contrapositive.

∇_c Suppose $\mathbf{A}, \neg\mathbf{A} \in \Phi$. We have $\Phi \models F_o$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\neg E)$. Hence Φ is \mathfrak{N}_* -inconsistent.

∇_β Let $\mathbf{A} \in \Phi$, $\mathbf{A} =_\beta \mathbf{B}$ and $\Phi * \mathbf{B}$ be \mathfrak{N}_* -inconsistent. That is, $\Phi * \mathbf{B} \models F_o$. By $\mathfrak{N}(\neg I)$, we know $\Phi \models \neg\mathbf{B}$. Since $\mathbf{A} \in \Phi$, we know $\Phi \models \mathbf{B}$ by $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\beta)$. Using $\mathfrak{N}(\neg E)$, we know $\Phi \models F_o$.

∇ Suppose $\neg\neg\mathbf{A} \in \Phi$ and $\Phi * \mathbf{A}$ is \mathfrak{N}_* -inconsistent. From $\Phi * \mathbf{A} \models F_o$ and $\mathfrak{N}(\neg I)$, we have $\Phi \models \neg\mathbf{A}$. Since $\neg\neg\mathbf{A} \in \Phi$, we can apply $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\neg E)$ to obtain $\Phi \models F_o$.



Lemma: Saturated \mathfrak{Acc}_*



∇_v Suppose $(\mathbf{A} \vee \mathbf{B}) \in \Phi$ and both $\Phi * \mathbf{A}$ and $\Phi * \mathbf{B}$ are \mathfrak{N}_* -inconsistent.



Lemma: Saturated Acc_{*}



▽ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.

Lemma: Saturated Acc_{*}



▽ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.



Lemma: Saturated Acc_{*}

▽ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.

▽ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent.



Lemma: Saturated Acc_{*}

▽ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.

▽ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By
 $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$.

Lemma: Saturated \mathfrak{Acc}_*



- ∇_V Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ∇_L Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By
 $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$
with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$.

Lemma: Saturated \mathfrak{Acc}_*



- ∇_V Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ∇_L Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By
 $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$
with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and
 $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$.

Lemma: Saturated \mathfrak{Acc}_*



- ∇_V Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ∇_L Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By
 $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$
with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and
 $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ
is \mathfrak{N}_* -inconsistent.

Lemma: Saturated \mathfrak{Acc}_*



- ∇_V Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ∇_L Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By
 $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$
with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and
 $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ
is \mathfrak{N}_* -inconsistent.

Lemma: Saturated \mathfrak{Acc}_*



- ∇ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ∇ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.
- ∇ Suppose $(\Pi^\alpha G) \in \Phi$ and $\Phi * (GA)$ is \mathfrak{N}_* -inconsistent.

Lemma: Saturated \mathfrak{Acc}_*



- ∇ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ∇ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.
- ∇ Suppose $(\Pi^\alpha G) \in \Phi$ and $\Phi * (GA)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(\neg I)$, $\Phi \Vdash \neg(GA)$.

Lemma: Saturated \mathfrak{Acc}_*

- ∇ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ∇ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.
- ∇ Suppose $(\Pi^\alpha G) \in \Phi$ and $\Phi * (GA)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(\neg I)$, $\Phi \Vdash \neg(GA)$. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\Pi E)$, $\Phi \Vdash GA$.

Lemma: Saturated \mathfrak{Acc}_*



- ∇ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent.
By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ∇ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.
- ∇ Suppose $(\Pi^\alpha G) \in \Phi$ and $\Phi * (GA)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(\neg I)$, $\Phi \Vdash \neg(GA)$. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\Pi E)$, $\Phi \Vdash GA$. Finally, $\mathfrak{N}(\neg E)$ implies $\Phi \Vdash F_\circ$.

Lemma: Saturated \mathfrak{Acc}_*



- ▽ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ▽ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.
- ▽ Suppose $(\Pi^\alpha G) \in \Phi$ and $\Phi * (GA)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(\neg I)$, $\Phi \Vdash \neg(GA)$. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\Pi E)$, $\Phi \Vdash GA$. Finally, $\mathfrak{N}(\neg E)$ implies $\Phi \Vdash F_\circ$.
- ▽ Suppose $\neg(\Pi^\alpha G) \in \Phi$, w_α is a parameter which does not occur in Φ , and $\Phi * \neg(Gw)$ is \mathfrak{N}_* -inconsistent.

Lemma: Saturated \mathfrak{Acc}_*



- ▽ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ▽ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.
- ▽ Suppose $(\Pi^\alpha G) \in \Phi$ and $\Phi * (GA)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(\neg I)$, $\Phi \Vdash \neg(GA)$. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\Pi E)$, $\Phi \Vdash GA$. Finally, $\mathfrak{N}(\neg E)$ implies $\Phi \Vdash F_\circ$.
- ▽ Suppose $\neg(\Pi^\alpha G) \in \Phi$, w_α is a parameter which does not occur in Φ , and $\Phi * \neg(Gw)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$, $\Phi \Vdash Gw$.

Lemma: Saturated \mathfrak{Acc}_*

©Benzmüller, 2006



ATPHOL06-[11] – p.301



Lemma: Saturated \mathfrak{Acc}_*

- ▽ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ▽ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.
- ▽ Suppose $(\Pi^\alpha G) \in \Phi$ and $\Phi * (GA)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(\neg I)$, $\Phi \Vdash \neg(GA)$. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\Pi E)$, $\Phi \Vdash GA$. Finally, $\mathfrak{N}(\neg E)$ implies $\Phi \Vdash F_\circ$.
- ▽ Suppose $\neg(\Pi^\alpha G) \in \Phi$, w_α is a parameter which does not occur in Φ , and $\Phi * \neg(Gw)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$, $\Phi \Vdash Gw$. By $\mathfrak{N}(\Pi)^w$, $\Phi \Vdash (\Pi^\alpha G)$. Using $\mathfrak{N}(\neg E)$ with $\neg(\Pi^\alpha G) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.

Lemma: Saturated \mathfrak{Acc}_*

©Benzmüller, 2006



ATPHOL06-[11] – p.301



- ▽ Suppose $(A \vee B) \in \Phi$ and both $\Phi * A$ and $\Phi * B$ are \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\vee E)$, we have $\Phi \Vdash F_\circ$.
- ▽ Suppose $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_R)$, we have $\Phi, \neg A \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, we have $\Phi, \neg A \Vdash F_\circ$. By $\mathfrak{N}(Contr)$ and $\mathfrak{N}(\vee I_L)$, we have $\Phi \Vdash A \vee B$. Using $\mathfrak{N}(\neg E)$ with $\neg(A \vee B) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.
- ▽ Suppose $(\Pi^\alpha G) \in \Phi$ and $\Phi * (GA)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(\neg I)$, $\Phi \Vdash \neg(GA)$. By $\mathfrak{N}(Hyp)$ and $\mathfrak{N}(\Pi E)$, $\Phi \Vdash GA$. Finally, $\mathfrak{N}(\neg E)$ implies $\Phi \Vdash F_\circ$.
- ▽ Suppose $\neg(\Pi^\alpha G) \in \Phi$, w_α is a parameter which does not occur in Φ , and $\Phi * \neg(Gw)$ is \mathfrak{N}_* -inconsistent. By $\mathfrak{N}(Contr)$, $\Phi \Vdash Gw$. By $\mathfrak{N}(\Pi)^w$, $\Phi \Vdash (\Pi^\alpha G)$. Using $\mathfrak{N}(\neg E)$ with $\neg(\Pi^\alpha G) \in \Phi$, Φ is \mathfrak{N}_* -inconsistent.



ATPHOL06-[11] – p.301

Lemma: Saturated Acc_{*}



Lemma: Saturated Acc_{*}



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{M}_* -inconsistent.

Lemma: Saturated Acc_{*}



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{M}_* -inconsistent. We show that Φ is \mathfrak{M}_* -inconsistent.

Lemma: Saturated Acc_{*}



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{M}_* -inconsistent. We show that Φ is \mathfrak{M}_* -inconsistent. Using $\mathfrak{M}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$.

Lemma: Saturated \mathfrak{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Lemma: Saturated \mathfrak{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathfrak{Acc}_β .

Lemma: Saturated \mathfrak{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathfrak{Acc}_β .

Now let us check the conditions for the additional properties η , ξ , f , and b .

Lemma: Saturated \mathfrak{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathfrak{Acc}_β .

Now let us check the conditions for the additional properties η , ξ , f , and b .

Lemma: Saturated \mathcal{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathcal{Acc}_β .

Now let us check the conditions for the additional properties η , ξ , f , and b .

∇_η If $*$ includes η , then the proof proceeds as in ∇_β above, but with the rule $\mathfrak{N}(\eta)$.

Lemma: Saturated \mathcal{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathcal{Acc}_β .

Now let us check the conditions for the additional properties η , ξ , f , and b .

∇_η If $*$ includes η , then the proof proceeds as in ∇_β above, but with the rule $\mathfrak{N}(\eta)$.

∇_ξ Suppose $*$ includes ξ , $\neg(\lambda X.M \doteq^{\alpha \rightarrow \beta} \lambda X.N) \in \Phi$, and $\Phi * \neg([w/X]M \doteq^\beta [w/X]N)$ is \mathfrak{N}_* -inconsistent for some parameter w_α which does not occur in any sentence of Φ .

Lemma: Saturated \mathcal{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathcal{Acc}_β .

Now let us check the conditions for the additional properties η , ξ , f , and b .

∇_η If $*$ includes η , then the proof proceeds as in ∇_β above, but with the rule $\mathfrak{N}(\eta)$.

Lemma: Saturated \mathcal{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathcal{Acc}_β .

Now let us check the conditions for the additional properties η , ξ , f , and b .

∇_η If $*$ includes η , then the proof proceeds as in ∇_β above, but with the rule $\mathfrak{N}(\eta)$.

∇_ξ Suppose $*$ includes ξ , $\neg(\lambda X.M \doteq^{\alpha \rightarrow \beta} \lambda X.N) \in \Phi$, and $\Phi * \neg([w/X]M \doteq^\beta [w/X]N)$ is \mathfrak{N}_* -inconsistent for some parameter w_α which does not occur in any sentence of Φ . By $\mathfrak{N}(\text{Contr})$, we have $\Phi \Vdash ([w/X]M \doteq^\beta [w/X]N)$. By $\mathfrak{N}(\beta)$, we have $\Phi \Vdash ((\lambda X.M \doteq^\beta N)w)$.

Lemma: Saturated \mathcal{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathcal{Acc}_β .

Now let us check the conditions for the additional properties η , ξ , f , and b .

∇_η If $*$ includes η , then the proof proceeds as in ∇_β above, but with the rule $\mathfrak{N}(\eta)$.

∇_ξ Suppose $*$ includes ξ , $\neg(\lambda X.M \doteq^{\alpha \rightarrow \beta} \lambda X.N) \in \Phi$, and $\Phi * \neg([w/X]M \doteq^\beta [w/X]N)$ is \mathfrak{N}_* -inconsistent for some parameter w_α which does not occur in any sentence of Φ . By $\mathfrak{N}(\text{Contr})$, we have $\Phi \Vdash ([w/X]M \doteq^\beta [w/X]N)$. By $\mathfrak{N}(\beta)$, we have $\Phi \Vdash ((\lambda X.M \doteq^\beta N)w)$. By $\mathfrak{N}(III)$, $\Phi \Vdash (\forall X.M \doteq^\beta N)$. By $\mathfrak{N}(\xi)$, $\Phi \Vdash (\lambda X.M \doteq^{\alpha \rightarrow \beta} \lambda X.N)$.

Lemma: Saturated \mathcal{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathcal{Acc}_β .

Now let us check the conditions for the additional properties η , ξ , f , and b .

∇_η If $*$ includes η , then the proof proceeds as in ∇_β above, but with the rule $\mathfrak{N}(\eta)$.

∇_ξ Suppose $*$ includes ξ , $\neg(\lambda X.M \doteq^{\alpha \rightarrow \beta} \lambda X.N) \in \Phi$, and $\Phi * \neg([w/X]M \doteq^\beta [w/X]N)$ is \mathfrak{N}_* -inconsistent for some parameter w_α which does not occur in any sentence of Φ . By $\mathfrak{N}(\text{Contr})$, we have $\Phi \Vdash ([w/X]M \doteq^\beta [w/X]N)$. By $\mathfrak{N}(\beta)$, we have $\Phi \Vdash ((\lambda X.M \doteq^\beta N)w)$. By $\mathfrak{N}(III)$, $\Phi \Vdash (\forall X.M \doteq^\beta N)$.

Lemma: Saturated \mathcal{Acc}_*



∇_{sat} Let $\Phi * A$ and $\Phi * \neg A$ be \mathfrak{N}_* -inconsistent. We show that Φ is \mathfrak{N}_* -inconsistent. Using $\mathfrak{N}(\neg I)$, we know $\Phi \Vdash \neg A$ and $\Phi \Vdash \neg\neg A$. By $\mathfrak{N}(\neg E)$, we have $\Phi \Vdash F_\circ$.

Thus we have shown that Γ_Σ^β is saturated and in \mathcal{Acc}_β .

Now let us check the conditions for the additional properties η , ξ , f , and b .

∇_η If $*$ includes η , then the proof proceeds as in ∇_β above, but with the rule $\mathfrak{N}(\eta)$.

∇_ξ Suppose $*$ includes ξ , $\neg(\lambda X.M \doteq^{\alpha \rightarrow \beta} \lambda X.N) \in \Phi$, and $\Phi * \neg([w/X]M \doteq^\beta [w/X]N)$ is \mathfrak{N}_* -inconsistent for some parameter w_α which does not occur in any sentence of Φ . By $\mathfrak{N}(\text{Contr})$, we have $\Phi \Vdash ([w/X]M \doteq^\beta [w/X]N)$. By $\mathfrak{N}(\beta)$, we have $\Phi \Vdash ((\lambda X.M \doteq^\beta N)w)$. By $\mathfrak{N}(III)$, $\Phi \Vdash (\forall X.M \doteq^\beta N)$. By $\mathfrak{N}(\xi)$, $\Phi \Vdash (\lambda X.M \doteq^{\alpha \rightarrow \beta} \lambda X.N)$. By $\mathfrak{N}(\neg E)$, Φ is \mathfrak{N}_* -inconsistent.

Lemma: Saturated \mathfrak{Acc}_*



- ∇_f This case is analogous to the previous one, generalizing $\lambda X.M \doteq \lambda X.N$ to arbitrary $G \doteq H$ and using the extensionality rule $\mathfrak{N}(f)$ instead of $\mathfrak{N}(\xi)$.

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[11] – p.303

Lemma: Saturated \mathfrak{Acc}_*



- ∇_f This case is analogous to the previous one, generalizing $\lambda X.M \doteq \lambda X.N$ to arbitrary $G \doteq H$ and using the extensionality rule $\mathfrak{N}(f)$ instead of $\mathfrak{N}(\xi)$.
- ∇_b Suppose $*$ includes b .

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[11] – p.303

Lemma: Saturated \mathfrak{Acc}_*



- ∇_f This case is analogous to the previous one, generalizing $\lambda X.M \doteq \lambda X.N$ to arbitrary $G \doteq H$ and using the extensionality rule $\mathfrak{N}(f)$ instead of $\mathfrak{N}(\xi)$.

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[11] – p.303

Lemma: Saturated \mathfrak{Acc}_*



- ∇_f This case is analogous to the previous one, generalizing $\lambda X.M \doteq \lambda X.N$ to arbitrary $G \doteq H$ and using the extensionality rule $\mathfrak{N}(f)$ instead of $\mathfrak{N}(\xi)$.
- ∇_b Suppose $*$ includes b . Assume that $\neg(A \doteq^{\circ} B) \in \Phi$ but both $\Phi * \neg A * B \notin \Gamma_{\Sigma}^*$ and $\Phi * A * \neg B \notin \Gamma_{\Sigma}^*$.

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[11] – p.303

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



- ∇_f This case is analogous to the previous one, generalizing $\lambda X.M \doteq \lambda X.N$ to arbitrary $G \doteq H$ and using the extensionality rule $\mathfrak{M}(f)$ instead of $\mathfrak{M}(\xi)$.
- ∇_b Suppose $*$ includes b . Assume that $\neg(A \doteq^{\circ} B) \in \Phi$ but both $\Phi * \neg A * B \notin \Gamma_{\Sigma}^*$ and $\Phi * A * \neg B \notin \Gamma_{\Sigma}^*$. So both are \mathfrak{M}_* -inconsistent and we have $\Phi * A \Vdash B$ and $\Phi * B \Vdash A$ by $\mathfrak{M}(Contr)$.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



- ∇_f This case is analogous to the previous one, generalizing $\lambda X.M \doteq \lambda X.N$ to arbitrary $G \doteq H$ and using the extensionality rule $\mathfrak{M}(f)$ instead of $\mathfrak{M}(\xi)$.
- ∇_b Suppose $*$ includes b . Assume that $\neg(A \doteq^{\circ} B) \in \Phi$ but both $\Phi * \neg A * B \notin \Gamma_{\Sigma}^*$ and $\Phi * A * \neg B \notin \Gamma_{\Sigma}^*$. So both are \mathfrak{M}_* -inconsistent and we have $\Phi * A \Vdash B$ and $\Phi * B \Vdash A$ by $\mathfrak{M}(Contr)$. By $\mathfrak{M}(b)$, we have $\Phi \Vdash (A \doteq^{\circ} B)$.

Lemma: Saturated $\mathfrak{A}\text{cc}_*$



- ∇_f This case is analogous to the previous one, generalizing $\lambda X.M \doteq \lambda X.N$ to arbitrary $G \doteq H$ and using the extensionality rule $\mathfrak{M}(f)$ instead of $\mathfrak{M}(\xi)$.
- ∇_b Suppose $*$ includes b . Assume that $\neg(A \doteq^{\circ} B) \in \Phi$ but both $\Phi * \neg A * B \notin \Gamma_{\Sigma}^*$ and $\Phi * A * \neg B \notin \Gamma_{\Sigma}^*$. So both are \mathfrak{M}_* -inconsistent and we have $\Phi * A \Vdash B$ and $\Phi * B \Vdash A$ by $\mathfrak{M}(Contr)$. By $\mathfrak{M}(b)$, we have $\Phi \Vdash (A \doteq^{\circ} B)$. Since $\neg(A \doteq^{\circ} B) \in \Phi$, Φ is \mathfrak{M}_* -inconsistent.

Thm.: Henkin's Theorem for \mathfrak{M}_*



Let $* \in \{\beta, \beta\eta, \beta\xi, \beta f, \beta b, \beta\eta b, \beta\xi b, \beta f b\}$. Every sufficiently Σ -pure \mathfrak{M}_* -consistent set of sentences has an $\mathfrak{M}_*(\Sigma)$ -model.

Proof:

Thm.: Henkin's Theorem for \mathfrak{M}_*



Let $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. Every sufficiently Σ -pure \mathfrak{M}_* -consistent set of sentences has an $\mathfrak{M}_*(\Sigma)$ -model.

Proof: Let Φ be a sufficiently Σ -pure \mathfrak{M}_* -consistent set of sentences.

Thm.: Henkin's Theorem for \mathfrak{M}_*



Let $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. Every sufficiently Σ -pure \mathfrak{M}_* -consistent set of sentences has an $\mathfrak{M}_*(\Sigma)$ -model.

Proof: Let Φ be a sufficiently Σ -pure \mathfrak{M}_* -consistent set of sentences. By the previous lemma we know that the class of sets of \mathfrak{M}_* -consistent sentences constitute a saturated \mathfrak{Acc}_* ,



Thm.: Henkin's Theorem for \mathfrak{M}_*



Let $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. Every sufficiently Σ -pure \mathfrak{M}_* -consistent set of sentences has an $\mathfrak{M}_*(\Sigma)$ -model.

Proof: Let Φ be a sufficiently Σ -pure \mathfrak{M}_* -consistent set of sentences. By the previous lemma we know that the class of sets of \mathfrak{M}_* -consistent sentences constitute a saturated \mathfrak{Acc}_* , thus the Model Existence Theorem guarantees an $\mathfrak{M}_*(\Sigma)$ model for Φ .

Thm.: Completeness Theorem for \mathfrak{M}_*



Let Φ be a sufficiently Σ -pure set of sentences, \mathbf{A} be a sentence, and $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. If \mathbf{A} is valid in all models $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ that satisfy Φ , then $\Phi \vdash_{\mathfrak{M}_*} \mathbf{A}$.

Proof:



Thm.: Completeness Theorem for \mathfrak{M}_*



Let Φ be a sufficiently Σ -pure set of sentences, A be a sentence, and $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. If A is valid in all models $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ that satisfy Φ , then $\Phi \vdash_{\mathfrak{M}_*} A$.

Proof: Let A be given such that A is valid in all $\mathfrak{M}_*(\Sigma)$ models that satisfy Φ .

Thm.: Completeness Theorem for \mathfrak{M}_*



Let Φ be a sufficiently Σ -pure set of sentences, A be a sentence, and $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. If A is valid in all models $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ that satisfy Φ , then $\Phi \vdash_{\mathfrak{M}_*} A$.

Proof: Let A be given such that A is valid in all $\mathfrak{M}_*(\Sigma)$ models that satisfy Φ . So, $\Phi * \neg A$ is unsatisfiable in $\mathfrak{M}_*(\Sigma)$. Since only finitely many constants occur in $\neg A$, $\Phi * \neg A$ is sufficiently Σ -pure.

Thm.: Completeness Theorem for \mathfrak{M}_*



Let Φ be a sufficiently Σ -pure set of sentences, A be a sentence, and $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. If A is valid in all models $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ that satisfy Φ , then $\Phi \vdash_{\mathfrak{M}_*} A$.

Proof: Let A be given such that A is valid in all $\mathfrak{M}_*(\Sigma)$ models that satisfy Φ . So, $\Phi * \neg A$ is unsatisfiable in $\mathfrak{M}_*(\Sigma)$. Since only finitely many constants occur in $\neg A$, $\Phi * \neg A$ is sufficiently Σ -pure. So, $\Phi * \neg A$ must be \mathfrak{M}_* -inconsistent by Henkin's theorem above.

Thm.: Completeness Theorem for \mathfrak{M}_*



Let Φ be a sufficiently Σ -pure set of sentences, A be a sentence, and $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. If A is valid in all models $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ that satisfy Φ , then $\Phi \vdash_{\mathfrak{M}_*} A$.

Proof: Let A be given such that A is valid in all $\mathfrak{M}_*(\Sigma)$ models that satisfy Φ . So, $\Phi * \neg A$ is unsatisfiable in $\mathfrak{M}_*(\Sigma)$. Since only finitely many constants occur in $\neg A$, $\Phi * \neg A$ is sufficiently Σ -pure. So, $\Phi * \neg A$ must be \mathfrak{M}_* -inconsistent by Henkin's theorem above. Thus, $\Phi \vdash_{\mathfrak{M}_*} A$ by $\mathfrak{M}_*(\text{Contr})$.

Compactness



We can use the completeness theorems obtained so far to prove a compactness theorem for our semantics:

Let Φ be a sufficiently Σ -pure set of sentences and
 $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. Φ has an $\mathfrak{M}_*(\Sigma)$ -model
iff every finite subset of Φ has an $\mathfrak{M}_*(\Sigma)$ -model.

Proof:

Compactness



We can use the completeness theorems obtained so far to prove a compactness theorem for our semantics:

Let Φ be a sufficiently Σ -pure set of sentences and
 $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. Φ has an $\mathfrak{M}_*(\Sigma)$ -model
iff every finite subset of Φ has an $\mathfrak{M}_*(\Sigma)$ -model.

Proof: If Φ has no $\mathfrak{M}_*(\Sigma)$ -model, then by the previous Henkin Theorem Φ is \mathfrak{N}_* -inconsistent.

Compactness

©Benzmüller, 2006



ATPHOL06-[11] – p.306



We can use the completeness theorems obtained so far to prove a compactness theorem for our semantics:

Let Φ be a sufficiently Σ -pure set of sentences and
 $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. Φ has an $\mathfrak{M}_*(\Sigma)$ -model
iff every finite subset of Φ has an $\mathfrak{M}_*(\Sigma)$ -model.

Proof: If Φ has no $\mathfrak{M}_*(\Sigma)$ -model, then by the previous Henkin Theorem Φ is \mathfrak{N}_* -inconsistent. Since every \mathfrak{N}_* -proof is finite, this means some finite subset Ψ of Φ is \mathfrak{N}_* -inconsistent.

Compactness



ATPHOL06-[11] – p.306



We can use the completeness theorems obtained so far to prove a compactness theorem for our semantics:

Let Φ be a sufficiently Σ -pure set of sentences and
 $* \in \{\beta, \beta\eta, \beta\xi, \beta\mathfrak{f}, \beta\mathfrak{b}, \beta\eta\mathfrak{b}, \beta\xi\mathfrak{b}, \beta\mathfrak{f}\mathfrak{b}\}$. Φ has an $\mathfrak{M}_*(\Sigma)$ -model
iff every finite subset of Φ has an $\mathfrak{M}_*(\Sigma)$ -model.

Proof: If Φ has no $\mathfrak{M}_*(\Sigma)$ -model, then by the previous Henkin Theorem Φ is \mathfrak{N}_* -inconsistent. Since every \mathfrak{N}_* -proof is finite, this means some finite subset Ψ of Φ is \mathfrak{N}_* -inconsistent. Hence, Ψ has no $\mathfrak{M}_*(\Sigma)$ -model.

©Benzmüller, 2006



ATPHOL06-[11] – p.306

©Benzmüller, 2006



ATPHOL06-[11] – p.306

- only logical constants: $\neg_{o \rightarrow o}$, $\vee_{o \rightarrow o \rightarrow o}$, and $\Pi_{(\alpha \rightarrow o) \rightarrow o}$



Approaches to Higher-Order Resolution

Preliminaries and Notation

- only logical constants: $\neg_{o \rightarrow o}$, $\vee_{o \rightarrow o \rightarrow o}$, and $\Pi_{(\alpha \rightarrow o) \rightarrow o}$
- other logical operators can be defined (e.g.,
 $A \wedge B := \neg(\neg A \vee \neg B)$, $\forall X_\alpha.P X := \Pi_{((\alpha \rightarrow o) \rightarrow o)}(\lambda X_\alpha.P X)$, and
 $\exists X_\alpha.P X := \neg\forall X_\alpha.\neg(P X))$

Preliminaries and Notation

- only logical constants: $\neg_{o \rightarrow o}$, $\vee_{o \rightarrow o \rightarrow o}$, and $\Pi_{(\alpha \rightarrow o) \rightarrow o}$
- other logical operators can be defined (e.g.,
 $A \wedge B := \neg(\neg A \vee \neg B)$, $\forall X_\alpha.P X := \Pi_{((\alpha \rightarrow o) \rightarrow o)}(\lambda X_\alpha.P X)$, and
 $\exists X_\alpha.P X := \neg\forall X_\alpha.\neg(P X))$
- variables are printed as upper-case (e.g., X_α), constants as lower-case letters (e.g., c_α), and arbitrary terms appear as bold capital letters (e.g., T_α)

Preliminaries and Notation



- only logical constants: $\neg_{o \rightarrow o}$, $\vee_{o \rightarrow o \rightarrow o}$, and $\Pi_{(\alpha \rightarrow o) \rightarrow o}$
- other logical operators can be defined (e.g.,
 $A \wedge B := \neg(\neg A \vee \neg B)$, $\forall X_\alpha.P X := \Pi_{((\alpha \rightarrow o) \rightarrow o)}(\lambda X_\alpha.P X)$, and
 $\exists X_\alpha.P X := \neg\forall X_\alpha.\neg(P X))$
- variables are printed as upper-case (e.g., X_α), constants as lower-case letters (e.g., c_α), and arbitrary terms appear as bold capital letters (e.g., T_α)
- we abbreviate function applications by $h_{\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta} \overline{U_{\alpha_n}^n}$, which stands for $(\dots (h_{\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta} U_{\alpha_1}^1) \dots U_{\alpha_n}^n)$.

Preliminaries and Notation



- only logical constants: $\neg_{o \rightarrow o}$, $\vee_{o \rightarrow o \rightarrow o}$, and $\Pi_{(\alpha \rightarrow o) \rightarrow o}$
- other logical operators can be defined (e.g.,
 $A \wedge B := \neg(\neg A \vee \neg B)$, $\forall X_\alpha.P X := \Pi_{((\alpha \rightarrow o) \rightarrow o)}(\lambda X_\alpha.P X)$, and
 $\exists X_\alpha.P X := \neg\forall X_\alpha.\neg(P X))$
- variables are printed as upper-case (e.g., X_α), constants as lower-case letters (e.g., c_α), and arbitrary terms appear as bold capital letters (e.g., T_α)
- we abbreviate function applications by $h_{\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta} \overline{U_{\alpha_n}^n}$, which stands for $(\dots (h_{\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta} U_{\alpha_1}^1) \dots U_{\alpha_n}^n)$.
- α -, β -, η -, $\beta\eta$ -conversion and the definition of β -normal,
 $\beta\eta$ -normal, long $\beta\eta$ -normal, and head-normal form defined as usual (see [Barendregt84])
- substitutions defined as usual

Preliminaries and Notation



- only logical constants: $\neg_{o \rightarrow o}$, $\vee_{o \rightarrow o \rightarrow o}$, and $\Pi_{(\alpha \rightarrow o) \rightarrow o}$
- other logical operators can be defined (e.g.,
 $A \wedge B := \neg(\neg A \vee \neg B)$, $\forall X_\alpha.P X := \Pi_{((\alpha \rightarrow o) \rightarrow o)}(\lambda X_\alpha.P X)$, and
 $\exists X_\alpha.P X := \neg\forall X_\alpha.\neg(P X))$
- variables are printed as upper-case (e.g., X_α), constants as lower-case letters (e.g., c_α), and arbitrary terms appear as bold capital letters (e.g., T_α)
- we abbreviate function applications by $h_{\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta} \overline{U_{\alpha_n}^n}$, which stands for $(\dots (h_{\alpha_1 \rightarrow \dots \rightarrow \alpha_n \rightarrow \beta} U_{\alpha_1}^1) \dots U_{\alpha_n}^n)$.
- α -, β -, η -, $\beta\eta$ -conversion and the definition of β -normal,
 $\beta\eta$ -normal, long $\beta\eta$ -normal, and head-normal form defined as usual (see [Barendregt84])

Preliminaries and Notation



- substitutions are represented as $[T_1/X_1, \dots, T_n/X_n]$ where the X_i specify the variables to be replaced by the terms T_i . The application of a substitution σ to a term (resp. literal or clause) C is printed C_σ

Preliminaries and Notation



- substitutions are represented as $[T_1/X_1, \dots, T_n/X_n]$ where the X_i specify the variables to be replaced by the terms T_i . The application of a substitution σ to a term (resp. literal or clause) C is printed C_σ
- a **resolution calculus R** provides a set of rules $\{r_n \mid 0 < n \leq i\}$ defined on clauses

Preliminaries and Notation



- substitutions are represented as $[T_1/X_1, \dots, T_n/X_n]$ where the X_i specify the variables to be replaced by the terms T_i . The application of a substitution σ to a term (resp. literal or clause) C is printed C_σ
- a **resolution calculus R** provides a set of rules $\{r_n \mid 0 < n \leq i\}$ defined on clauses
- we write $\Phi \vdash^{r_n} C$ ($C' \vdash^{r_n} C$) iff clause C is the result of a **one step application** of rule $r_n \in R$ to premise clauses $C'_i \in \Phi$ (to C' respectively)

©Benzmüller, 2006



ATPHOL06-[12] – p.309

Preliminaries and Notation



- substitutions are represented as $[T_1/X_1, \dots, T_n/X_n]$ where the X_i specify the variables to be replaced by the terms T_i . The application of a substitution σ to a term (resp. literal or clause) C is printed C_σ
- a **resolution calculus R** provides a set of rules $\{r_n \mid 0 < n \leq i\}$ defined on clauses
- we write $\Phi \vdash^{r_n} C$ ($C' \vdash^{r_n} C$) iff clause C is the result of a **one step application** of rule $r_n \in R$ to premise clauses $C'_i \in \Phi$ (to C' respectively)
- multiple step derivations in calculus R are abbreviated by $\Phi_1 \vdash_R \Phi_k$ (or $C_1 \vdash_R C_k$)

Def.: General Bindings



Let $\alpha := (\overline{\beta^l} \rightarrow \gamma)$ and let h be a constant or variable of type $(\overline{\delta_m} \rightarrow \gamma)$ in Σ ,

©Benzmüller, 2006



ATPHOL06-[12] – p.309

©Benzmüller, 2006



ATPHOL06-[12] – p.310

Def.: General Bindings



Let $\alpha := (\overline{\beta^l} \rightarrow \gamma)$ and let h be a constant or variable of type $(\overline{\delta_m} \rightarrow \gamma)$ in Σ , then

$$G := \lambda \overline{X_{\beta^l}}. h \overline{V^m}$$

($m \geq 0$) is called a **partial binding of type α and head h** (see also [SnGa89,Snyder91]),

Def.: General Bindings



Let $\alpha := (\overline{\beta^l} \rightarrow \gamma)$ and let h be a constant or variable of type $(\overline{\delta_m} \rightarrow \gamma)$ in Σ , then

$$G := \lambda \overline{X_{\beta^l}}. h \overline{V^m}$$

($m \geq 0$) is called a **partial binding of type α and head h** (see also [SnGa89,Snyder91]), if $V^i = H^i \overline{X_{\beta^l}}$ and the H^i are new variables of types $\overline{\beta^l} \rightarrow \delta^i$.

Partial bindings, where the head is a bound variable $X_{\beta_j}^j$ are called **projection bindings** (we write them as G_α^j) and **imitation bindings** (written G_α^h) otherwise.

Def.: General Bindings



Let $\alpha := (\overline{\beta^l} \rightarrow \gamma)$ and let h be a constant or variable of type $(\overline{\delta_m} \rightarrow \gamma)$ in Σ , then

$$G := \lambda \overline{X_{\beta^l}}. h \overline{V^m}$$

($m \geq 0$) is called a **partial binding of type α and head h** (see also [SnGa89,Snyder91]), if $V^i = H^i \overline{X_{\beta^l}}$ and the H^i are new variables of types $\overline{\beta^l} \rightarrow \delta^i$.

Def.: General Bindings



Let $\alpha := (\overline{\beta^l} \rightarrow \gamma)$ and let h be a constant or variable of type $(\overline{\delta_m} \rightarrow \gamma)$ in Σ , then

$$G := \lambda \overline{X_{\beta^l}}. h \overline{V^m}$$

($m \geq 0$) is called a **partial binding of type α and head h** (see also [SnGa89,Snyder91]), if $V^i = H^i \overline{X_{\beta^l}}$ and the H^i are new variables of types $\overline{\beta^l} \rightarrow \delta^i$.

Partial bindings, where the head is a bound variable $X_{\beta_j}^j$ are called **projection bindings** (we write them as G_α^j) and **imitation bindings** (written G_α^h) otherwise.

Since we need both imitation and projection bindings for higher-order unification, we collect them in the set of **general bindings for h and α** ($\mathcal{AB}_\alpha^h := \{G_\alpha^h\} \cup \{G_\alpha^j \mid j \leq l\}$).

Def.: General Bindings



Let $\alpha := (\beta^l \rightarrow \gamma)$ and let h be a constant or variable of type $(\delta_m \rightarrow \gamma)$ in Σ , then

$$G := \lambda \overline{X}_{\beta}^l. h \overline{V}^m$$

($m \geq 0$) is called a **partial binding of type α and head h** (see also [SnGa89, Snyder91]), if $V^i = H^i \overline{X}_{\beta}^l$ and the H^i are new variables of types $\beta^l \rightarrow \delta^i$.

Partial bindings, where the head is a bound variable $X_{\beta_j}^j$ are called **projection bindings** (we write them as G_{α}^j) and **imitation bindings** (written G_{α}^h) otherwise.

Since we need both imitation and projection bindings for higher-order unification, we collect them in the set of **general bindings for h and α** ($\mathcal{AB}_{\alpha}^h := \{G_{\alpha}^h\} \cup \{G_{\alpha}^j \mid j \leq l\}$).

Def.: Literals



- literals, e.g., $[A]^{\mu}$, consist of a **literal atom A** and a **polarity $\mu \in \{T, F\}$**

Def.: Literals



- literals, e.g., $[A]^{\mu}$, consist of a **literal atom A** and a **polarity $\mu \in \{T, F\}$**
- we distinguish between **proper literals** and **pre-literals**: the (normalised) atom of a pre-literal has a logical constant at head position, whereas this must not be the case for proper literals

Def.: Literals



- literals, e.g., $[A]^{\mu}$, consist of a **literal atom A** and a **polarity $\mu \in \{T, F\}$**
- we distinguish between **proper literals** and **pre-literals**: the (normalised) atom of a pre-literal has a logical constant at head position, whereas this must not be the case for proper literals
- for instance, $[A \vee B]^T$ is a pre-literal and $[p_0 \rightarrow_0 (A \vee B)]^T$ is a proper literal

Def.: Literals



Def.: Unification Constraints



- literals, e.g., $[A]^{\mu}$, consist of a **literal atom** A and a **polarity** $\mu \in \{T, F\}$
- we distinguish between **proper literals** and **pre-literals**: the (normalised) atom of a pre-literal has a logical constant at head position, whereas this must not be the case for proper literals
- for instance, $[A \vee B]^T$ is a pre-literal and $[p_{o \rightarrow o} (A \vee B)]^T$ is a proper literal
- a literal is called **flexible** if its atom contains a variable at head position

©Benzmüller, 2006



ATPHOL06-[12] – p.311

©Benzmüller, 2006



ATPHOL06-[12] – p.312

Def.: Unification Constraints



Def.: Unification Constraints



- a unification problem between two terms T^1 and T^2 (between n terms T^1, \dots, T^n) generated during the refutation process is called an **unification constraint**
- it is represented as $[T^1 \neq? T^2]$ (resp. $[\neq? (T^1, \dots, T^n)]$)

©Benzmüller, 2006



ATPHOL06-[12] – p.312

©Benzmüller, 2006



ATPHOL06-[12] – p.312

- a unification problem between two terms T^1 and T^2 (between n terms T^1, \dots, T^n) generated during the refutation process is called an **unification constraint**

Def.: Unification Constraints



- a unification problem between two terms T^1 and T^2 (between n terms T^1, \dots, T^n) generated during the refutation process is called an **unification constraint**
- it is represented as $[T^1 \neq? T^2]$ (resp. $\neq? (T^1, \dots, T^n)$)
- a unification constraint is called a **flex-flex pair** if both unification terms have **flexible** heads, i.e. variables at head position
- a unification constraint is called a **flex-rigid pair** if one unification term has a **flexible** head, i.e. variable at head position

©Benzmüller, 2006



ATPHOL06-[12] – p.312

Def.: Clauses

- clauses consist of disjunctions of literals or unification constraints



Def.: Clauses



- clauses consist of disjunctions of literals or unification constraints
- the unification constraints specify conditions under which the other literals are valid

©Benzmüller, 2006



ATPHOL06-[12] – p.313

Def.: Clauses

- clauses consist of disjunctions of literals or unification constraints
- the unification constraints specify conditions under which the other literals are valid
- for instance, the clause $[p_{\alpha \rightarrow \beta \rightarrow o} T_\alpha^1 T_\beta^2]^T \vee [T_\alpha^1 \neq? S_\alpha^1] \vee [T_\beta^2 \neq? S_\beta^2]$ can be read as: **if T^1 is unifiable with S^1 and T^2 with S^2 then $(p T^1 T^2)$ holds**



ATPHOL06-[12] – p.313

Def.: Clauses

- clauses consist of disjunctions of literals or unification constraints
- the unification constraints specify conditions under which the other literals are valid
- for instance, the clause
 $[p_{\alpha \rightarrow \beta \rightarrow o} T_\alpha^1 T_\beta^2]^T \vee [T_\alpha^1 \neq? S_\alpha^1] \vee [T_\beta^2 \neq? S_\beta^2]$ can be read as: if T^1 is unifiable with S^1 and T^2 with S^2 then $(p T^1 T^2)$ holds
- we implicitly treat the disjunction operator \vee in clauses as commutative and associative

Def.: Clauses

- clauses consist of disjunctions of literals or unification constraints
- the unification constraints specify conditions under which the other literals are valid
- for instance, the clause
 $[p_{\alpha \rightarrow \beta \rightarrow o} T_\alpha^1 T_\beta^2]^T \vee [T_\alpha^1 \neq? S_\alpha^1] \vee [T_\beta^2 \neq? S_\beta^2]$ can be read as: if T^1 is unifiable with S^1 and T^2 with S^2 then $(p T^1 T^2)$ holds
- we implicitly treat the disjunction operator \vee in clauses as commutative and associative
- additionally we presuppose commutativity of $\neq?$ and implicitly identify any two α -equal constraints or literals.
- furthermore we assume that any two clauses have disjoint sets of free variables, i.e. for each freshly generated clause we choose new free variables

Def.: Clauses

- clauses consist of disjunctions of literals or unification constraints
- the unification constraints specify conditions under which the other literals are valid
- for instance, the clause
 $[p_{\alpha \rightarrow \beta \rightarrow o} T_\alpha^1 T_\beta^2]^T \vee [T_\alpha^1 \neq? S_\alpha^1] \vee [T_\beta^2 \neq? S_\beta^2]$ can be read as: if T^1 is unifiable with S^1 and T^2 with S^2 then $(p T^1 T^2)$ holds
- we implicitly treat the disjunction operator \vee in clauses as commutative and associative
- additionally we presuppose commutativity of $\neq?$ and implicitly identify any two α -equal constraints or literals.

Def.: Clauses (contd.)

- if a clause contains at least one pre-literal we call it a **pre-clause**, otherwise a **proper clause**

- if a clause contains at least one pre-literal we call it a **pre-clause**, otherwise a **proper clause**
- a clause is called **empty**, denoted by \square , if it consists only of (possibly none) **flex-flex** pairs.

- an important aspect of clause normalisation is **Skolemisation**

Rem.: Skolemisation

- an important aspect of clause normalisation is **Skolemisation**
- we employ Miller's sound adaptation of traditional first-order Skolemisation [Miller:pihol83], which associates with each Skolem function the **minimum number of arguments** the Skolem function has to be applied to

Rem.: Skolemisation

- an important aspect of clause normalisation is **Skolemisation**
- we employ Miller's sound adaptation of traditional first-order Skolemisation [Miller:pihol83], which associates with each Skolem function the **minimum number of arguments** the Skolem function has to be applied to
- higher-order Skolemisation becomes sound, if any Skolem function f^n only occurs in a Skolem term, i.e., a formula $S = f^n \bar{A}^n$, where none of the A^i contains a bound variable

Rem.: Skolemisation



- an important aspect of clause normalisation is **Skolemisation**
- we employ Miller's sound adaptation of traditional first-order Skolemisation [Miller:pihol83], which associates with each Skolem function the **minimum number of arguments** the Skolem function has to be applied to
- higher-order Skolemisation becomes sound, if any Skolem function f^n only occurs in a Skolem term, i.e., a formula $S = f^n \bar{A}^n$, where none of the A^i contains a bound variable
- thus, the Skolem terms only serve as descriptions of the existential witnesses and never appear as functions proper

©Benzmüller, 2006



ATPHOL06-[12] – p.315

Rem.: Skolemisation

- an important aspect of clause normalisation is **Skolemisation**
- we employ Miller's sound adaptation of traditional first-order Skolemisation [Miller:pihol83], which associates with each Skolem function the **minimum number of arguments** the Skolem function has to be applied to
- higher-order Skolemisation becomes sound, if any Skolem function f^n only occurs in a Skolem term, i.e., a formula $S = f^n \bar{A}^n$, where none of the A^i contains a bound variable
- thus, the Skolem terms only serve as descriptions of the existential witnesses and never appear as functions proper
- without this additional restriction the calculi do not really become unsound, but one can prove an instance of the axiom of choice ([Andrews73]), which we want to treat as an optional axiom for the resolution calculi presented here

©Benzmüller, 2006



ATPHOL06-[12] – p.315



Approaches to Higher-Order
Resolution: \mathcal{R}

©Benzmüller, 2006



ATPHOL06-[13] – p.316

©Benzmüller, 2006



ATPHOL06-[13] – p.317



Andrews' Higher-Order Resolution \mathcal{R}



We present and discuss Andrews' higher-order resolution calculus [Andrews71] in our uniform notation; we call this calculus \mathcal{R}

λ -Conversion

- Andrews' provides two rules for α -conversion and β -reduction
- he does not provide a rule for η -conversion: consequently η -equality of two terms (e.g., $f_{\nu \rightarrow \nu} \doteq \lambda X.f X$) cannot be proven in this approach without employing the functional extensionality axiom of appropriate type

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[13] – p.317

Andrews' Higher-Order Resolution \mathcal{R}



Clause Normalisation

- \mathcal{R} introduces only four rules belonging to clause normalisation: negation elimination, conjunction elimination, existential elimination, and universal elimination

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[13] – p.318

Andrews' Higher-Order Resolution \mathcal{R}



We present and discuss Andrews' higher-order resolution calculus [Andrews71] in our uniform notation; we call this calculus \mathcal{R}

λ -Conversion

- Andrews' provides two rules for α -conversion and β -reduction
- he does not provide a rule for η -conversion: consequently η -equality of two terms (e.g., $f_{\nu \rightarrow \nu} \doteq \lambda X.f X$) cannot be proven in this approach without employing the functional extensionality axiom of appropriate type
- we omit explicit rules for α - and β -convertibility and instead treat them implicitly, i.e. we assume that the presented rules operate on input and generate output in β -normal form and we automatically identify terms which differ only with respect to the names of bound variables

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[13] – p.317

Andrews' Higher-Order Resolution \mathcal{R}



Clause Normalisation

- \mathcal{R} introduces only four rules belonging to clause normalisation: negation elimination, conjunction elimination, existential elimination, and universal elimination
- as our presentation of clauses in contrast to [Andrews71] explicitly mentions the polarities of clauses and brackets the literal atoms we need additional structural rules, e.g., the rule \vee^T

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[13] – p.318

Clause Normalisation

- \mathcal{R} introduces only four rules belonging to clause normalisation: negation elimination, conjunction elimination, existential elimination, and universal elimination
- as our presentation of clauses in contrast to [Andrews71] explicitly mentions the polarities of clauses and brackets the literal atoms we need additional structural rules, e.g., the rule \vee^T

- negation elimination:

$$\frac{C \vee [\neg A]^T}{C \vee [A]^F} \neg^T \quad \frac{C \vee [\neg A]^F}{C \vee [A]^T} \neg^F$$

Clause Normalisation (contd.)

- existential/universal elimination:

$$\frac{C \vee [\Pi^\alpha A]^T}{C \vee [A X_\alpha]^T} \Pi^T \quad \frac{C \vee [\Pi^\alpha A]^F}{C \vee [A s_\alpha]^F} \Pi^F$$

Clause Normalisation

- \mathcal{R} introduces only four rules belonging to clause normalisation: negation elimination, conjunction elimination, existential elimination, and universal elimination
- as our presentation of clauses in contrast to [Andrews71] explicitly mentions the polarities of clauses and brackets the literal atoms we need additional structural rules, e.g., the rule \vee^T

- negation elimination:

$$\frac{C \vee [\neg A]^T}{C \vee [A]^F} \neg^T \quad \frac{C \vee [\neg A]^F}{C \vee [A]^T} \neg^F$$

- conjunction/disjunction elimination:

$$\frac{C \vee [A \vee B]^T}{C \vee [A]^T \vee [B]^T} \vee^T \quad \frac{C \vee [A \vee B]^F}{C \vee [A]^F} \vee^F \quad \frac{C \vee [A \vee B]^F}{C \vee [B]^F} \vee^F$$

Clause Normalisation (contd.)

- existential/universal elimination:

$$\frac{C \vee [\Pi^\alpha A]^T}{C \vee [A X_\alpha]^T} \Pi^T \quad \frac{C \vee [\Pi^\alpha A]^F}{C \vee [A s_\alpha]^F} \Pi^F$$

X_α is a new free variable and s_α is a new Skolem term

Clause Normalisation (contd.)

- existential/universal elimination:

$$\frac{C \vee [\Pi^\alpha A]^T}{C \vee [A X_\alpha]^T} \Pi^T \quad \frac{C \vee [\Pi^\alpha A]^F}{C \vee [A s_\alpha]^F} \Pi^F$$

X_α is a new free variable and s_α is a new Skolem term

- additionally Andrews presents rules addressing commutativity and associativity of the \vee -operator connecting the clauses literals; we have already mentioned the implicit treatment of these aspects here

Resolution & Factorisation

- Instead of a resolution and a factorisation rule —which work in connection with unification —Andrews presents a simplification and a cut rule. The cut rule is only applicable to clauses with two complementary literals which have identical atoms. Similarly Sim is defined only for clauses with two identical literals. In order to generate identical literal atoms during the refutation process these two rules have to be combined with the substitution rule Sub presented below.

Clause Normalisation (contd.)

- existential/universal elimination:

$$\frac{C \vee [\Pi^\alpha A]^T}{C \vee [A X_\alpha]^T} \Pi^T \quad \frac{C \vee [\Pi^\alpha A]^F}{C \vee [A s_\alpha]^F} \Pi^F$$

X_α is a new free variable and s_α is a new Skolem term

- additionally Andrews presents rules addressing commutativity and associativity of the \vee -operator connecting the clauses literals; we have already mentioned the implicit treatment of these aspects here
- we refer with $\text{Cnf}(A)$ to the set of clauses obtained from formula A by exhaustive clause normalisation

Resolution & Factorisation

- Instead of a resolution and a factorisation rule —which work in connection with unification —Andrews presents a simplification and a cut rule. The cut rule is only applicable to clauses with two complementary literals which have identical atoms. Similarly Sim is defined only for clauses with two identical literals. In order to generate identical literal atoms during the refutation process these two rules have to be combined with the substitution rule Sub presented below.
- Simplification:

$$\frac{[A]^\mu \vee [A]^\mu \vee C}{[A]^\mu \vee C} \text{ Sim}$$

Resolution & Factorisation

- Instead of a resolution and a factorisation rule —which work in connection with unification —Andrews presents a simplification and a cut rule. The cut rule is only applicable to clauses with two complementary literals which have identical atoms. Similarly Sim is defined only for clauses with two identical literals. In order to generate identical literal atoms during the refutation process these two rules have to be combined with the substitution rule Sub presented below.

- Simplification:

$$\frac{[A]^{\mu} \vee [A]^{\mu} \vee C}{[A]^{\mu} \vee C} \text{ Sim}$$

- Cut:

$$\frac{[A]^{\mu} \vee C \quad [A]^{\nu} \vee D}{C \vee D} \text{ Cut}$$

Andrews' Higher-Order Resolution \mathcal{R}

Unification & Primitive Substitution

- As higher-order unification was still an open problem in 1971 calculus \mathcal{R} employs the British museum method instead, i.e. it provides a substitution rule that allows to blindly instantiate free variables by arbitrary terms. As the instantiated terms may contain logical constants, instantiation of variables in proper clauses may lead to pre-clauses, which must be normalised again with the clause normalisation rules.

- Substitution of arbitrary terms:

$$\frac{C}{C_{[T_\alpha/X_\alpha]}} \text{ Sub}$$

Andrews' Higher-Order Resolution \mathcal{R}

Unification & Primitive Substitution

- As higher-order unification was still an open problem in 1971 calculus \mathcal{R} employs the British museum method instead, i.e. it provides a substitution rule that allows to blindly instantiate free variables by arbitrary terms. As the instantiated terms may contain logical constants, instantiation of variables in proper clauses may lead to pre-clauses, which must be normalised again with the clause normalisation rules.

Andrews' Higher-Order Resolution \mathcal{R}

Unification & Primitive Substitution

- As higher-order unification was still an open problem in 1971 calculus \mathcal{R} employs the British museum method instead, i.e. it provides a substitution rule that allows to blindly instantiate free variables by arbitrary terms. As the instantiated terms may contain logical constants, instantiation of variables in proper clauses may lead to pre-clauses, which must be normalised again with the clause normalisation rules.

- Substitution of arbitrary terms:

$$\frac{C}{C_{[T_\alpha/X_\alpha]}} \text{ Sub}$$

X_α is a free variable occurring in C .

Extensionality Treatment

- Calculus \mathcal{R} does not provide rules addressing the functional and/or Boolean extensionality principles.

Extensionality Treatment

- Calculus \mathcal{R} does not provide rules addressing the functional and/or Boolean extensionality principles.
- Instead \mathcal{R} assumes that the following extensionality axioms are (in form of respective clauses) explicitly added to the search space. And since the functional extensionality principle is parameterised over arbitrary functional types infinitely many functional extensionality axioms are required.
- Extensionality axioms

$$\text{EXT}_{\alpha \rightarrow \beta}^{\doteq}: \forall F_{\alpha \rightarrow \beta}, \forall G_{\alpha \rightarrow \beta}, (\forall X_\beta, F X \doteq G X) \Rightarrow F \doteq G$$

$$\text{EXT}_o^{\doteq}: \forall A_o, \forall B_o, (A \Leftrightarrow B) \Rightarrow A \doteq^o B$$

Extensionality Treatment

- Calculus \mathcal{R} does not provide rules addressing the functional and/or Boolean extensionality principles.
- Instead \mathcal{R} assumes that the following extensionality axioms are (in form of respective clauses) explicitly added to the search space. And since the functional extensionality principle is parameterised over arbitrary functional types infinitely many functional extensionality axioms are required.

Extensionality Treatment (contd.)

- The extensionality clauses derived from the extensionality axioms have the following form (note the many free variables, especially at literal head position, that are introduced into the search space – they heavily increase the amount of blind search in any attempt to automate the calculus):

Extensionality Treatment (contd.)

- The extensionality clauses derived from the extensionality axioms have the following form (note the many free variables, especially at literal head position, that are introduced into the search space – they heavily increase the amount of blind search in any attempt to automate the calculus):

$$\mathcal{E}_1^{\alpha \rightarrow \beta} : [p(F s)]^T \vee [Q F]^F \vee [Q G]^T$$

$$\mathcal{E}_2^{\alpha \rightarrow \beta} : [p(G s)]^F \vee [Q F]^F \vee [Q G]^T$$

$$\mathcal{E}_1^o : [A]^F \vee [B]^F \vee [P A]^F \vee [P B]^T$$

$$\mathcal{E}_2^o : [A]^T \vee [B]^T \vee [P A]^F \vee [P B]^T$$

Extensionality Treatment (contd.)

- The extensionality clauses derived from the extensionality axioms have the following form (note the many free variables, especially at literal head position, that are introduced into the search space – they heavily increase the amount of blind search in any attempt to automate the calculus):

$$\mathcal{E}_1^{\alpha \rightarrow \beta} : [p(F s)]^T \vee [Q F]^F \vee [Q G]^T$$

$$\mathcal{E}_2^{\alpha \rightarrow \beta} : [p(G s)]^F \vee [Q F]^F \vee [Q G]^T$$

$$\mathcal{E}_1^o : [A]^F \vee [B]^F \vee [P A]^F \vee [P B]^T$$

$$\mathcal{E}_2^o : [A]^T \vee [B]^T \vee [P A]^F \vee [P B]^T$$

$p_{\beta \rightarrow o}, s_\alpha$ are Skolem terms and $A_o, B_o, P_{o \rightarrow o}, Q_{(\alpha \rightarrow \beta) \rightarrow o}$ are new free variables.

Extensionality Treatment (contd.)

- The extensionality clauses derived from the extensionality axioms have the following form (note the many free variables, especially at literal head position, that are introduced into the search space – they heavily increase the amount of blind search in any attempt to automate the calculus):

$$\mathcal{E}_1^{\alpha \rightarrow \beta} : [p(F s)]^T \vee [Q F]^F \vee [Q G]^T$$

$$\mathcal{E}_2^{\alpha \rightarrow \beta} : [p(G s)]^F \vee [Q F]^F \vee [Q G]^T$$

$$\mathcal{E}_1^o : [A]^F \vee [B]^F \vee [P A]^F \vee [P B]^T$$

$$\mathcal{E}_2^o : [A]^T \vee [B]^T \vee [P A]^F \vee [P B]^T$$

Proof Search

- initially the proof problem is negated and normalised

Proof Search

- initially the proof problem is negated and normalised
- proof search then starts with the normalised clauses and applies the cut and simplification rule in close connection with the substitution rule

Proof Search

- initially the proof problem is negated and normalised
- proof search then starts with the normalised clauses and applies the cut and simplification rule in close connection with the substitution rule
- intermediate applications of the clause normalisation rules may be needed to normalise temporarily generated pre-clauses
- the extensionality treatment in \mathcal{R} simply assumes to add at the beginning of the refutation process the above clauses obtained from the extensionality axioms

Proof Search

- initially the proof problem is negated and normalised
- proof search then starts with the normalised clauses and applies the cut and simplification rule in close connection with the substitution rule
- intermediate applications of the clause normalisation rules may be needed to normalise temporarily generated pre-clauses

Proof Search

- initially the proof problem is negated and normalised
- proof search then starts with the normalised clauses and applies the cut and simplification rule in close connection with the substitution rule
- intermediate applications of the clause normalisation rules may be needed to normalise temporarily generated pre-clauses
- the extensionality treatment in \mathcal{R} simply assumes to add at the beginning of the refutation process the above clauses obtained from the extensionality axioms
- the proof search can be graphically illustrated as follows:

ext. axioms	proof search & blind variable instantiation
----------------	---

Completeness

- [Andrews71] gives a completeness proof for calculus \mathcal{R} with respect to the semantical notion of V-complexes (corresponds to our weakest model class $\mathfrak{M}_\beta(\Sigma)$)

Completeness

- [Andrews71] gives a completeness proof for calculus \mathcal{R} with respect to the semantical notion of V-complexes (corresponds to our weakest model class $\mathfrak{M}_\beta(\Sigma)$)
- as the extensionality principles are not valid in this rather weak semantical structures, the extensionality axioms are not needed in this completeness proof
- Theorem: (V-completeness of \mathcal{R}) The calculus \mathcal{R} is (sound and) complete with respect to the notion of V-complexes.

Proof: [Andrews71].

Completeness

- [Andrews71] gives a completeness proof for calculus \mathcal{R} with respect to the semantical notion of V-complexes (corresponds to our weakest model class $\mathfrak{M}_\beta(\Sigma)$)
- as the extensionality principles are not valid in this rather weak semantical structures, the extensionality axioms are not needed in this completeness proof

Henkin Completeness

- We can also prove Henkin completeness of calculus \mathcal{R} .

Henkin Completeness

- We can also prove Henkin completeness of calculus \mathcal{R} .
- Theorem: (Henkin completeness of \mathcal{R}) The calculus \mathcal{R} is (sound and) complete with respect to Henkin semantics provided that the infinitely many extensionality axioms are given.

Proof: exercise

Example Proofs

Exercise: How are the following theorems proved in calculus \mathcal{R} ?

- Leibniz equality and η -equality:

$$f_{\iota \rightarrow \iota} \doteq \lambda X_{\iota \bullet} f X$$

Henkin Completeness

- We can also prove Henkin completeness of calculus \mathcal{R} .
- Theorem: (Henkin completeness of \mathcal{R}) The calculus \mathcal{R} is (sound and) complete with respect to Henkin semantics provided that the infinitely many extensionality axioms are given.

Proof: exercise

Example Proofs

Exercise: How are the following theorems proved in calculus \mathcal{R} ?

- Leibniz equality and η -equality:

$$f_{\iota \rightarrow \iota} \doteq \lambda X_{\iota \bullet} f X$$

- The set of all red balls equals the set of all balls that are red: $\{X | \text{red } X \wedge \text{ball } X\} = \{X | \text{ball } X \wedge \text{red } X\}$. This problem can be encoded as

$$(\lambda X_{\iota \bullet} \text{red } X \wedge \text{ball } X) = (\lambda X_{\iota \bullet} \text{ball } X \wedge \text{red } X)$$

Exercise: How are the following theorems proved in calculus \mathcal{R} ?

- All unary logical operators $O_{o \rightarrow o}$ which map the propositions a and b to \top consequently also map $a \wedge b$ to \top :

$$\forall O_{o \rightarrow o} (O a_o) \wedge (O b_o) \Rightarrow (O (a_o \wedge b_o))$$



Approaches to Higher-Order
Resolution: \mathcal{CR}

Exercise: How are the following theorems proved in calculus \mathcal{R} ?

- In Henkin semantics the domain D_o of all Booleans contains exactly the truth values \perp and \top . Consequently the domain of all mappings from Booleans to Booleans contains exactly contains in each Henkin model at most four elements. And because of the requirement, that the function domains in Henkin models must be rich enough such that every term has a denotation, it follows that $D_{o \rightarrow o}$ contains exactly the pairwise distinct denotations of the following four terms: $\lambda X_o. X_o$, $\lambda X_o. \neg X_o$, $\lambda X_o. \perp$, and $\lambda X_o. \top$. This theorem can be formulated as follows (where $f_{o \rightarrow o}$ is a constant):

$$(f = \lambda X_o. X_o) \vee (f = \lambda X_o. \neg X_o) \vee (f = \lambda X_o. \perp) \vee (f = \lambda X_o. \top)$$

Huet's Constrained Resolution \mathcal{CR}

We transform Huet's constrained resolution approach [Huet72,Huet73] in our uniform notation. The calculus here is the unsorted fragment of the variant of Huet's approach as presented in [Kohlhase94]. In the remainder of this paper we refer to this calculus with \mathcal{CR} .

λ -Conversion

- Calculus \mathcal{CR} assumes that terms, literals, and clauses are implicitly reduced to β -normal form.

We transform Huet's constrained resolution approach [Huet72,Huet73] in our uniform notation. The calculus here is the unsorted fragment of the variant of Huet's approach as presented in [Kohlhase94]. In the remainder of this paper we refer to this calculus with \mathcal{CR} .

λ -Conversion

- Calculus \mathcal{CR} assumes that terms, literals, and clauses are implicitly reduced to β -normal form.
- Furthermore, we assume that α -equality is treated implicitly, i.e. we identify all terms that differ only with respect to the names of bound variables.

Clause Normalisation

- [Huet72] does not explicitly present clause normalisation rules but assumes that they are given. Here we employ the rules \neg^T , \neg^F , \vee^T , \vee_l^F , \vee_r^F , Π^T , and Π^F as already defined for calculus \mathcal{R} before.

Clause Normalisation

- [Huet72] does not explicitly present clause normalisation rules but assumes that they are given. Here we employ the rules \neg^T , \neg^F , \vee^T , \vee_l^F , \vee_r^F , Π^T , and Π^F as already defined for calculus \mathcal{R} before.

Resolution & Factorisation

- As first-order unification is decidable and unitary it can be employed as a strong filter in first-order resolution [Robinson65].

Resolution & Factorisation

- As first-order unification is decidable and unitary it can be employed as a strong filter in first-order resolution [Robinson65].
- Unfortunately higher-order unification is not decidable (cf. [Lucchesi72,Huet73,Goldfarb81]) and thus it can not be applied in the sense of a terminating side computation in higher-order theorem proving.

Resolution & Factorisation (contd.)

- In his original approach Huet presented a hyper-resolution rule which simultaneously resolves on the resolution literals A^1, \dots, A^n ($1 \leq n$) and B^1, \dots, B^m ($1 \leq m$) of two given clauses and adds the unification constraint $[\neq^? (A^1, \dots, A^n, B^1, \dots, B^m)]$ to the resolvent:

$$\frac{[A^1]^{\mu} \vee \dots \vee [A^n]^{\mu} \vee C \quad [B^1]^{\nu} \vee \dots \vee [B^m]^{\nu} \vee D}{C \vee D \vee [\neq^? (A^1, \dots, A^n, B^1, \dots, B^m)]} \text{ Hres}$$

(where $\mu \neq \nu$).

Resolution & Factorisation

- As first-order unification is decidable and unitary it can be employed as a strong filter in first-order resolution [Robinson65].
- Unfortunately higher-order unification is not decidable (cf. [Lucchesi72,Huet73,Goldfarb81]) and thus it can not be applied in the sense of a terminating side computation in higher-order theorem proving.
- Huet therefore suggests in [Huet72,Huet73] to delay the unification process and to explicitly encode unification problems occurring during the refutation search as unification constraints.

Resolution & Factorisation (contd.)

- In order to ease the comparison with the two other approaches discussed in this paper we instead employ a resolution rule Res and a factorisation rule Fac.

Resolution & Factorisation (contd.)

- In order to ease the comparison with the two other approaches discussed in this paper we instead employ a resolution rule Res and a factorisation rule Fac.
- Like Hres both rules encode the unification problem to be solved as a unification constraint:

Resolution & Factorisation (contd.)

- In order to ease the comparison with the two other approaches discussed in this paper we instead employ a resolution rule Res and a factorisation rule Fac.
- Like Hres both rules encode the unification problem to be solved as a unification constraint:

Constrained resolution:

$$\frac{[A]^\mu \vee C \quad [B]^\nu \vee D}{C \vee D \vee [A \neq? B]} \text{ Res}$$

(where $\mu \neq \nu$).

Resolution & Factorisation (contd.)

- In order to ease the comparison with the two other approaches discussed in this paper we instead employ a resolution rule Res and a factorisation rule Fac.
- Like Hres both rules encode the unification problem to be solved as a unification constraint:

Constrained resolution:

$$\frac{[A]^\mu \vee C \quad [B]^\nu \vee D}{C \vee D \vee [A \neq? B]} \text{ Res}$$

(where $\mu \neq \nu$).

Constrained factorisation:

$$\frac{[A]^\mu \vee [B]^\mu \vee C}{[A]^\mu \vee C \vee [A \neq? B]^F} \text{ Fac}$$

Resolution & Factorisation (contd.)

- One can easily prove by induction on $n + m$ that each proof step applying rule Hres can be replaced by a corresponding derivation employing Res and Fac.

Resolution & Factorisation (contd.)

- One can easily prove by induction on $n + m$ that each proof step applying rule Hres can be replaced by a corresponding derivation employing Res and Fac.
- For a formal proof note that the unification constraint $[\neq? (A^1, \dots, A^n, B^1, \dots, B^m)]$ is equivalent to $[A^1 \neq? A^2] \vee [A^2 \neq? A^3] \vee \dots \vee [A^{n-1} \neq? A^n] \vee [A^n \neq? B^1] \vee [B^1 \neq? B^2] \vee [B^2 \neq? B^3] \vee \dots \vee [B^{n-1} \neq? B^n]$.

Unification & Splitting

- [Huet75] introduces higher-order unification and higher-order pre-unification and shows that higher-order pre-unification is sufficient to verify the soundness of a refutation in which the occurring unification problems have been delayed until the end.
- The higher-order pre-unification rules presented here are discussed in detail in [Benzmüller-PhD-99]. They furthermore closely reflect the rules as presented in [SnyderGallier89].

Unification & Splitting

- [Huet75] introduces higher-order unification and higher-order pre-unification and shows that higher-order pre-unification is sufficient to verify the soundness of a refutation in which the occurring unification problems have been delayed until the end.

Unification & Splitting

- [Huet75] introduces higher-order unification and higher-order pre-unification and shows that higher-order pre-unification is sufficient to verify the soundness of a refutation in which the occurring unification problems have been delayed until the end.
 - The higher-order pre-unification rules presented here are discussed in detail in [Benzmüller-PhD-99]. They furthermore closely reflect the rules as presented in [SnyderGallier89].
 - Elimination of trivial pairs:
- $$\frac{C \vee [A \neq? A]}{C} \text{ Triv}$$

Unification & Splitting

- [Huet75] introduces higher-order unification and higher-order pre-unification and shows that higher-order pre-unification is sufficient to verify the soundness of a refutation in which the occurring unification problems have been delayed until the end.
- The higher-order pre-unification rules presented here are discussed in detail in [Benzmüller-PhD-99]. They furthermore closely reflect the rules as presented in [SnyderGallier89].

- Elimination of trivial pairs:

$$\frac{C \vee [A \neq? A]}{C} \text{ Triv}$$

- Decomposition

$$\frac{C \vee [h\bar{U^n} \neq? h\bar{V^n}]}{C \vee [U^1 \neq? V^1] \vee \dots \vee [U^n \neq? V^n]} \text{ Dec}$$

Unification & Splitting (contd.)

- Elimination of λ -binders:
(weak functional extensionality)

$$\frac{C \vee [M_{\alpha \rightarrow \beta} \neq? N_{\alpha \rightarrow \beta}]}{C \vee [M s_\alpha \neq? N s_\alpha]} \text{ Func}$$

s_α is a new Skolem term.

Unification & Splitting (contd.)

- Elimination of λ -binders:
(weak functional extensionality)

$$\frac{C \vee [M_{\alpha \rightarrow \beta} \neq? N_{\alpha \rightarrow \beta}]}{C \vee [M s_\alpha \neq? N s_\alpha]} \text{ Func}$$

Unification & Splitting (contd.)

- Elimination of λ -binders:
(weak functional extensionality)

$$\frac{C \vee [M_{\alpha \rightarrow \beta} \neq? N_{\alpha \rightarrow \beta}]}{C \vee [M s_\alpha \neq? N s_\alpha]} \text{ Func}$$

s_α is a new Skolem term.

- Imitation of rigid heads: $\frac{C \vee [F_\gamma \bar{U^n} \neq? h\bar{V^m}] \quad G \in AB_\gamma^h}{C \vee [F \neq? G] \vee [F \bar{U^n} \neq? h\bar{V^m}]} \text{ FlexRigid}$

Unification & Splitting (contd.)

Elimination of λ -binders:

- (weak functional extensionality)

$$\frac{C \vee [M_{\alpha \rightarrow \beta} \neq? N_{\alpha \rightarrow \beta}]}{C \vee [M s_\alpha \neq? N s_\alpha]} \text{ Func}$$

s_α is a new Skolem term.

- Imitation of rigid heads: $\frac{C \vee [F_\gamma \overline{U^n} \neq? h \overline{V^m}] \quad G \in AB_\gamma^h}{C \vee [F \neq? G] \vee [F \overline{U^n} \neq? h \overline{V^m}]} \text{ FlexRigid}$

AB_γ^h is the set of general bindings of type γ for head h .

Unification & Splitting (contd.)

- Huet points to the usefulness of eager unification to filter out clauses with non-unifiable unification constraints or to back-propagate the solutions of easily solvable constraints (e.g., in case of first-order unification problems occurring during the proof search): many of the higher-order unification problems occurring in practice are decidable and have only finitely many solutions.

Unification & Splitting (contd.)

- Huet points to the usefulness of eager unification to filter out clauses with non-unifiable unification constraints or to back-propagate the solutions of easily solvable constraints (e.g., in case of first-order unification problems occurring during the proof search): many of the higher-order unification problems occurring in practice are decidable and have only finitely many solutions.
- Hence, even though higher-order unification is generally not decidable it is sensible in practice to apply the unification algorithm with a particular resource, such that only those unification problems which may have further solutions beyond this bound need to be delayed.

Unification & Splitting (contd.)

- In our presentation of calculus \mathcal{CR} we explicitly address the aspect of eager unification and substitution by rule Subst. This rule back-propagates eagerly computed unifiers to the literal part of a clause.

Unification & Splitting (contd.)

- In our presentation of calculus \mathcal{CR} we explicitly address the aspect of eager unification and substitution by rule Subst. This rule back-propagates eagerly computed unifiers to the literal part of a clause.
- Eager unification & substitution:

$$\frac{C \vee [X \neq? A] \quad X \notin \text{free}(A)}{C_{[A/X]}} \text{ Subst}$$

Unification & Splitting (contd.)

- The literal heads of our clauses may consist of set variables and it may be necessary to instantiate them with terms introducing new logical constant at head position in order to find a refutation.
- Unfortunately not all appropriate instantiations can be computed with the calculus rules presented so far.

Unification & Splitting (contd.)

- The literal heads of our clauses may consist of set variables and it may be necessary to instantiate them with terms introducing new logical constant at head position in order to find a refutation.

Unification & Splitting (contd.)

- The literal heads of our clauses may consist of set variables and it may be necessary to instantiate them with terms introducing new logical constant at head position in order to find a refutation.
- Unfortunately not all appropriate instantiations can be computed with the calculus rules presented so far.
- To address this problem Huet's approach provides the following splitting rules:

Unification & Splitting (contd.)

- Instantiate set variables:

Unification & Splitting (contd.)

- Instantiate set variables:

$$\frac{[P A]^T \vee C}{[Q]^T \vee [R]^T \vee C \vee [P A \neq? (Q_o \vee R_o)]} S^T_{\vee}$$

Unification & Splitting (contd.)

- Instantiate set variables:

$$\frac{[P A]^T \vee C}{[Q]^T \vee [R]^T \vee C \vee [P A \neq? (Q_o \vee R_o)]} S^T_{\vee}$$

$$\frac{[P A]^{\mu} \vee C}{[Q]^{\nu} \vee C \vee [P A \neq? \neg Q_o]} S^T_{\neg}$$

(where $\mu \neq \nu$)

Unification & Splitting (contd.)

- Instantiate set variables:

$$\frac{[P A]^T \vee C}{[Q]^T \vee [R]^T \vee C \vee [P A \neq? (Q_o \vee R_o)]} S^T_{\vee}$$

$$\frac{[P A]^F \vee C}{[Q]^F \vee C \vee [P A \neq? (Q_o \vee R_o)]} S^F_{\vee}$$

$$\frac{[P A]^{\mu} \vee C}{[Q]^{\nu} \vee C \vee [P A \neq? \neg Q_o]} S^T_{\neg}$$

(where $\mu \neq \nu$)

$$\frac{[P A]^F \vee C}{[Q]^F \vee C \vee [P A \neq? (Q_o \vee R_o)]} S^F_{\vee}$$

$$\frac{[P A]^{\mu} \vee C}{[R]^F \vee C \vee [P A \neq? (Q_o \vee R_o)]} S^F_{\neg}$$

$$\frac{[P A_{\alpha \rightarrow o}]^T \vee C}{[M_{\alpha \rightarrow o}]^T \vee C \vee [P A \neq? \Pi^{\alpha} M]} S^T_{\Pi}$$

Huet's Constrained Resolution \mathcal{CR}



Unification & Splitting (contd.)

- Instantiate set variables:

$$\frac{[P\ A]^{\mu} \vee C}{[Q]^{\nu} \vee C \vee [P\ A \neq? \neg Q_o]} S_{\neg}^T \quad (\text{where } \mu \neq \nu)$$

$$\frac{[P\ A]^T \vee C}{[Q]^T \vee [R]^T \vee C \vee [P\ A \neq? (Q_o \vee R_o)]} S_v^T$$

$$\frac{[P\ A]^F \vee C}{[Q]^F \vee C \vee [P\ A \neq? (Q_o \vee R_o)]} S_v^F$$

$$[R]^F \vee C \vee [P\ A \neq? (Q_o \vee R_o)]$$

$$\frac{[P\ A_{\alpha \rightarrow o}]^T \vee C}{[M_{\alpha \rightarrow o}]^T \vee C \vee [P\ A \neq? \Pi^{\alpha} M]} S_{\Pi}^T$$

$$\frac{[P\ A_{\alpha \rightarrow o}]^F \vee C}{[M_{\alpha \rightarrow o}]^F \vee C \vee [P\ A \neq? \Pi^{\alpha} M]} S_{\Pi}^F$$



Huet's Constrained Resolution \mathcal{CR}



Unification & Splitting (contd.)

- A theorem which is not refutable in \mathcal{CR} if the splitting rules are not available is $\exists A_o.A$:

Huet's Constrained Resolution \mathcal{CR}



Unification & Splitting (contd.)

- Instantiate set variables:

$$\frac{[P\ A]^{\mu} \vee C}{[Q]^{\nu} \vee C \vee [P\ A \neq? \neg Q_o]} S_{\neg}^T \quad (\text{where } \mu \neq \nu)$$

$$\frac{[P\ A]^T \vee C}{[Q]^T \vee [R]^T \vee C \vee [P\ A \neq? (Q_o \vee R_o)]} S_v^T$$

$$\frac{[P\ A]^F \vee C}{[Q]^F \vee C \vee [P\ A \neq? (Q_o \vee R_o)]} S_v^F$$

$$[R]^F \vee C \vee [P\ A \neq? (Q_o \vee R_o)]$$

$$\frac{[P\ A_{\alpha \rightarrow o}]^T \vee C}{[M_{\alpha \rightarrow o}]^T \vee C \vee [P\ A \neq? \Pi^{\alpha} M]} S_{\Pi}^T$$

$$\frac{[P\ A_{\alpha \rightarrow o}]^F \vee C}{[M_{\alpha \rightarrow o}]^F \vee C \vee [P\ A \neq? \Pi^{\alpha} M]} S_{\Pi}^F$$

- S_{Π}^T and S_{Π}^F are infinitely branching as they are parameterised over type α . $Q_o, R_o, M_{\alpha \rightarrow o}, Z_{\alpha}$ are new variables and s_{α} is a new Skolem constant.



Huet's Constrained Resolution \mathcal{CR}



Unification & Splitting (contd.)

- A theorem which is not refutable in \mathcal{CR} if the splitting rules are not available is $\exists A_o.A$:
- After negation this statement normalises to clause $C_1 : [A]^F$, such that none but the splitting rules are applicable. With the help of rule S_{\neg}^T and eager unification, however, we can derive $C_2 : [A']^T$ which is then successfully resolvable against C_1 .



Extensionality Treatment

- On the one hand η -convertibility is built-in in higher-order unification, such that calculus \mathcal{CR} already supports functional extensionality reasoning to a certain extend.

Extensionality Treatment

- On the one hand η -convertibility is built-in in higher-order unification, such that calculus \mathcal{CR} already supports functional extensionality reasoning to a certain extend.
- On the other hand \mathcal{CR} nevertheless fails to address full extensionality as it does not realise the required subtle interplay between the functional and Boolean extensionality principles.

Extensionality Treatment

- On the one hand η -convertibility is built-in in higher-order unification, such that calculus \mathcal{CR} already supports functional extensionality reasoning to a certain extend.
- On the other hand \mathcal{CR} nevertheless fails to address full extensionality as it does not realise the required subtle interplay between the functional and Boolean extensionality principles.
- Without employing additional (Boolean and functional!) extensionality axioms \mathcal{CR} is, e.g., not able to prove the rather simple examples presented before.

Proof Search

- Initially the proof problem is negated and normalised. The main proof search then operates on the generated clauses by applying the resolution, factorisation, and splitting rules.

Proof Search

- Initially the proof problem is negated and normalised. The main proof search then operates on the generated clauses by applying the resolution, factorisation, and splitting rules.
- Despite the possibility of eager unification \mathcal{CR} generally foresees to delay the higher-order unification process in order to overcome the undecidability problem.

Proof Search

- Initially the proof problem is negated and normalised. The main proof search then operates on the generated clauses by applying the resolution, factorisation, and splitting rules.
- Despite the possibility of eager unification \mathcal{CR} generally foresees to delay the higher-order unification process in order to overcome the undecidability problem.
- When deriving a potentially empty clause (no normal literals), \mathcal{CR} then tests whether the accumulated unification constraints justifying this particular refutation are solvable.

Proof Search (contd.)

- Like \mathcal{R} , the extensionality treatment of \mathcal{CR} requires to add infinitely many extensionality axioms to the search space.

Proof Search (contd.)

- Like \mathcal{R} , the extensionality treatment of \mathcal{CR} requires to add infinitely many extensionality axioms to the search space.
- The following figure graphically illustrates the main ideas of the proof search in \mathcal{CR} .



Completeness Results

- [Huet72,Huet73] analyses completeness of \mathcal{CR} formally only with respect to Andrews V-complexes, i.e. Huet verifies that the set of non-refutable sentences in \mathcal{CR} is an abstract consistency class for V-complexes.

Completeness Results

- [Huet72,Huet73] analyses completeness of \mathcal{CR} formally only with respect to Andrews V-complexes, i.e. Huet verifies that the set of non-refutable sentences in \mathcal{CR} is an abstract consistency class for V-complexes.
- Theorem (V-completeness of \mathcal{CR}): The calculus \mathcal{CR} is complete with respect to the notion of V-complexes.

Proof: [Huet72,Huet73]

Completeness Results

- [Huet72,Huet73] analyses completeness of \mathcal{CR} formally only with respect to Andrews V-complexes, i.e. Huet verifies that the set of non-refutable sentences in \mathcal{CR} is an abstract consistency class for V-complexes.
- Theorem (V-completeness of \mathcal{CR}): The calculus \mathcal{CR} is complete with respect to the notion of V-complexes.

Proof: [Huet72,Huet73]

Exercise: How are the following theorems proved in calculus \mathcal{CR} ?

- Leibniz equality and η -equality:

$$f_{\nu \rightarrow \nu} \doteq \lambda X_{\nu}. f X$$

Proof: exercise

Example Proofs



Exercise: How are the following theorems proved in calculus \mathcal{CR} ?

- Leibniz equality and η -equality:

$$f_{\nu \rightarrow \nu} \doteq \lambda X_{\nu}. f X$$

- The set of all red balls equals the set of all balls that are red:
 $\{X | \text{red } X \wedge \text{ball } X\} = \{X | \text{ball } X \wedge \text{red } X\}$. This problem can be encoded as

$$(\lambda X_{\nu}. \text{red } X \wedge \text{ball } X) = (\lambda X_{\nu}. \text{ball } X \wedge \text{red } X)$$

Example Proofs



Exercise: How are the following theorems proved in calculus \mathcal{CR} ?

- In Henkin semantics the domain \mathcal{D}_o of all Booleans contains exactly the truth values \perp and \top . Consequently the domain of all mappings from Booleans to Booleans contains exactly four elements. And because of the requirement, that the function domains in Henkin models must be rich enough such that every term has a denotation, it follows that $\mathcal{D}_{o \rightarrow o}$ contains exactly the pairwise distinct denotations of the following four terms: $\lambda X_o.X_o$, $\lambda X_o.\neg X_o$, $\lambda X_o.\perp$, and $\lambda X_o.\top$. This theorem can be formulated as follows (where $f_{o \rightarrow o}$ is a constant):

$$(f = \lambda X_o.X_o) \vee (f = \lambda X_o.\neg X_o) \vee (f = \lambda X_o.\perp) \vee (f = \lambda X_o.\top)$$

Example Proofs



Exercise: How are the following theorems proved in calculus \mathcal{CR} ?

- All unary logical operators $O_{o \rightarrow o}$ which map the propositions a and b to \top consequently also map $a \wedge b$ to \top :

$$\forall O_{o \rightarrow o}. (O a_o) \wedge (O b_o) \Rightarrow (O (a_o \wedge b_o))$$



Approaches to Higher-Order
Resolution: \mathcal{ER}

Extensional HO Resolution \mathcal{ER}



Clause normalization

$$\begin{array}{c}
 \frac{C \vee [A \vee B]^T}{C \vee [A]^T \vee [B]^T} \vee^T \quad \frac{C \vee [A \vee B]^F}{C \vee [A]^F} \vee^F \quad \frac{C \vee [A \vee B]^F}{C \vee [B]^F} \vee^F \\
 \\
 \frac{C \vee [\neg A]^T}{C \vee [A]^F} \neg^T \quad \frac{C \vee [\neg A]^F}{C \vee [A]^T} \neg^F \\
 \\
 \frac{C \vee [\Pi^\alpha A]^T \quad X_\alpha \text{ new variable}}{C \vee [A X]^T} \Pi^T \\
 \\
 \frac{C \vee [\Pi^\alpha A]^F \quad \text{sk}_\alpha \text{ Skolem term}}{C \vee [A \text{ sk}_\alpha]^F} \Pi^F
 \end{array}$$

This rules may be combined into a single rule Cnf .



©Benzmüller, 2006

ATPHOL06-[15] – p.352

Extensional HO Resolution \mathcal{ER}



(Pre-)unification rules

$$\frac{C \vee [M_{\alpha \rightarrow \beta} \neq? N_{\alpha \rightarrow \beta}]^F \quad s_\alpha \text{ Skolem-Term}}{C \vee [M s \neq? N s]} \text{ Func}$$

$$\frac{C \vee [h \overline{U^n} \neq? h \overline{V^n}] \quad C \vee [U^1 \neq? V^1] \vee \dots \vee [U^n \neq? V^n]}{C} \text{ Dec} \quad \frac{C \vee [A \neq? A]}{C} \text{ Triv}$$

$$\frac{C \vee [F_\gamma \overline{U^n} \neq? h \overline{V^n}] \quad G \in \mathcal{GB}_\gamma^h}{C \vee [F \neq? G] \vee [F \overline{U^n} \neq? h \overline{V^n}]} \text{ Flex/Rigid}$$

$$\frac{C \vee E \quad E \text{ solved for } C}{\text{Cnf}(\text{subst}_E(C))} \text{ Subst}$$



©Benzmüller, 2006

ATPHOL06-[15] – p.354

Extensional HO Resolution \mathcal{ER}



Resolution and Factorisation

$$\begin{array}{c}
 \frac{[N]^\alpha \vee C \quad [M]^\beta \vee D \quad \alpha \neq \beta}{C \vee D \vee [N \neq? M]} \text{ Res} \\
 \\
 \frac{[N]^\alpha \vee [M]^\alpha \vee C \quad \alpha \in \{T, F\}}{[N]^\alpha \vee C \vee [N \neq? M]} \text{ Fac} \\
 \\
 \frac{[Q_\gamma \overline{U^k}]^\alpha \vee C \quad P \in \mathcal{GB}_\gamma^{\{\neg, \vee\} \cup \{\Pi^\beta | \beta \in T^k\}}}{[Q_\gamma \overline{U^k}]^\alpha \vee C \vee [Q \neq? P]} \text{ Prim}^k
 \end{array}$$



©Benzmüller, 2006

ATPHOL06-[15] – p.353

Extensional HO Resolution \mathcal{ER}



Extensionality rules

$$\frac{C \vee [M_o \neq? N_o]^F}{\text{Cnf}(C \vee [M_o \Leftrightarrow N_o]^F)} \text{ Equiv}$$

$$\frac{C \vee [M_\alpha \neq? N_\alpha]^F \quad \alpha \in \{o, \iota\}}{\text{Cnf}(C \vee [\forall P_{\alpha \rightarrow o}. PM \Rightarrow PN]^F)} \text{ Leib}$$



©Benzmüller, 2006

ATPHOL06-[15] – p.355

Extensionality Treatment

- Instead of adding infinitely many extensionality axioms to the search space CR provides two new extensionality rules which closely connect refutation search and eager unification.

Extensionality Treatment

- Instead of adding infinitely many extensionality axioms to the search space CR provides two new extensionality rules which closely connect refutation search and eager unification.
- The idea is to allow for recursive calls from higher-order unification to the overall refutation process.
- This turns the rather weak syntactical higher-order unification approach considered so far into a most general approach for *dynamic* higher-order theory unification.

Extensionality Treatment

- Instead of adding infinitely many extensionality axioms to the search space CR provides two new extensionality rules which closely connect refutation search and eager unification.
- The idea is to allow for recursive calls from higher-order unification to the overall refutation process.

Proof Search

- Initially the proof problem is negated and normalised. The main proof search then closely interleaves the refutation process on resolution layer and unification, i.e. the main proof search rules Res, Fac, and Prim and the unification rules are integrated at a common conceptual level. The calls from unification to the overall refutation process with rules *Leib* and *Equiv* introduce new clauses into the search space which can be resolved against already given ones.

Proof Search

- Initially the proof problem is negated and normalised. The main proof search then closely interleaves the refutation process on resolution layer and unification, i.e. the main proof search rules Res, Fac, and Prim and the unification rules are integrated at a common conceptual level. The calls from unification to the overall refutation process with rules *Leib* and *Equiv* introduce new clauses into the search space which can be resolved against already given ones.
- The following figure graphically illustrates the main ideas of the proof search in \mathcal{ER} .

interleaved proof search & unification

Ex.: Extensional HO Resolution \mathcal{ER}

$$\forall P_{o \rightarrow o} (P a_o) \wedge (P b_o) \Rightarrow (P (a_o \wedge b_o))$$

Negation and clause normalization

$$C_1 : [p a]^T \quad C_2 : [p b]^T \quad C_3 : [p (a \wedge b)]^F$$

Resolution between C_1 and C_3 and between C_2 and C_3

$$C_4 : [p a \neq? p (a \wedge b)] \quad C_5 : [p b \neq? p (a \wedge b)]$$

Decomposition

$$C_6 : [a \neq? (a \wedge b)] \quad C_7 : [b \neq? (a \wedge b)]$$

Recursive call of proof process with rules Equiv and Cnf

$$C_8 : [a]^F \vee [b]^F \quad C_9 : [a]^T \vee [b]^T \quad C_{10} : [a]^T \quad C_{11} : [b]^T$$

Ex.: Extensional HO Resolution \mathcal{ER}

$$\forall B_{\alpha \rightarrow o}, C_{\alpha \rightarrow o}, D_{\alpha \rightarrow o} B \cup (C \cap D) = (B \cup C) \cap (B \cup D)$$

Negation and definition expansion with

$$U = \lambda A_{\alpha \rightarrow o}, B_{\alpha \rightarrow o}, X_{\alpha \rightarrow o} (A X) \vee (B X) \quad \cap = \lambda A_{\alpha \rightarrow o}, B_{\alpha \rightarrow o}, X_{\alpha \rightarrow o} (A X) \wedge (B X)$$

leads to:

$$C_1 : [\lambda X_{\alpha \rightarrow o} (b X) \vee ((c X) \wedge (d X)) \neq? \lambda X_{\alpha \rightarrow o} ((b X) \vee (c X)) \wedge ((b X) \vee (d X))]$$

Goal directed functional and Boolean extensionality treatment:

$$C_2 : [(b x) \vee ((c x) \wedge (d x)) \Leftrightarrow ((b x) \vee (c x)) \wedge ((b x) \vee (d x))]^F$$

Clause normalization results then in a pure propositional, i.e. decidable, set of clauses. Only these clauses are still in the search space of LEO (in total there are 33 clauses generated and LEO finds the proof on a 2,5GHz PC in 820ms).

Similar proof in case of embedded propositions:

$$\forall P_{(o \rightarrow o) \rightarrow o}, B_{\alpha \rightarrow o}, C_{\alpha \rightarrow o}, D_{\alpha \rightarrow o} P(B \cup (C \cap D)) \Rightarrow P((B \cup C) \cap (B \cup D))$$

Ex.: Extensional HO Resolution \mathcal{ER}

Further small examples which test Henkin completeness:

$$\forall F_{o \rightarrow o} (F \doteq \lambda X_o . X_o) \vee (F \doteq \lambda X_o . \neg X_o) \vee (F \doteq \lambda X_o . \perp) \vee (F \doteq \lambda X_o . \top)$$

$$\forall H_{o \rightarrow o} H \perp \doteq H (H \top \doteq H \perp)$$

...

Def.: Sequent Calculi



Higher-Order Sequent
Calculi, Cut, and Saturation

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[16] – p.361

Def.: Sequent Calculi

- A sequent calculus \mathcal{G} provides an inductive definition for when $\vdash_{\mathcal{G}} \Delta$ holds.
- Here we consider a one-sided sequent calculus, thus a sequent is defined as a finite set Δ of β -normal sentences from $cwff_o(\Sigma)$.

Def.: Sequent Calculi



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[16] – p.362

- A sequent calculus \mathcal{G} provides an inductive definition for when $\vdash_{\mathcal{G}} \Delta$ holds.
- Here we consider a one-sided sequent calculus, thus a sequent is defined as a finite set Δ of β -normal sentences from $cwff_o(\Sigma)$.
- Let's compare our one-sided rules to standard two-sided sequent calculus rules (we use $\Delta * A$ to denote the set $\Delta \cup \{A\}$ (which is simply Δ if $A \in \Delta$)):

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[16] – p.362

©Benzmüller, 2006



UNIVERSITÄT
DES
SAARLANDES

ATPHOL06-[16] – p.362

Def.: Sequent Calculi



- A sequent calculus \mathcal{G} provides an inductive definition for when $\vdash_{\mathcal{G}} \Delta$ holds.
- Here we consider a one-sided sequent calculus, thus a sequent is defined as a finite set Δ of β -normal sentences from $cwff_o(\Sigma)$.
- Let's compare our one-sided rules to standard two-sided sequent calculus rules (we use $\Delta * A$ to denote the set $\Delta \cup \{A\}$ (which is simply Δ if $A \in \Delta$)):

$$\frac{}{\Delta * (A \vee B)} \mathcal{G}(V_+) \quad \text{vs.} \quad \frac{\Gamma \implies \Delta * (A \vee B)}{\Gamma \implies \Delta * (A \vee B)} V\text{R}$$

©Benzmüller, 2006



ATPHOL06-[16] – p.362

Def.: Sequent Calculi



- A sequent calculus \mathcal{G} provides an inductive definition for when $\vdash_{\mathcal{G}} \Delta$ holds.
- Here we consider a one-sided sequent calculus, thus a sequent is defined as a finite set Δ of β -normal sentences from $cwff_o(\Sigma)$.
- Let's compare our one-sided rules to standard two-sided sequent calculus rules (we use $\Delta * A$ to denote the set $\Delta \cup \{A\}$ (which is simply Δ if $A \in \Delta$)):

$$\frac{\Delta * A * B}{\Delta * (A \vee B)} \mathcal{G}(V_+) \quad \text{vs.} \quad \frac{\Gamma \implies \Delta * A * B}{\Gamma \implies \Delta * (A \vee B)} V\text{R}$$

©Benzmüller, 2006



ATPHOL06-[16] – p.362

Def.: Sequent Calculi



- A sequent calculus \mathcal{G} provides an inductive definition for when $\vdash_{\mathcal{G}} \Delta$ holds.
- Here we consider a one-sided sequent calculus, thus a sequent is defined as a finite set Δ of β -normal sentences from $cwff_o(\Sigma)$.
- Let's compare our one-sided rules to standard two-sided sequent calculus rules (we use $\Delta * A$ to denote the set $\Delta \cup \{A\}$ (which is simply Δ if $A \in \Delta$)):

$$\frac{\Delta * A * B}{\Delta * (A \vee B)} \mathcal{G}(V_+) \quad \text{vs.} \quad \frac{\Gamma \implies \Delta * A * B}{\Gamma \implies \Delta * (A \vee B)} V\text{R}$$

©Benzmüller, 2006



ATPHOL06-[16] – p.362

Def.: Sequent Calculi



- A sequent calculus \mathcal{G} provides an inductive definition for when $\vdash_{\mathcal{G}} \Delta$ holds.
- Here we consider a one-sided sequent calculus, thus a sequent is defined as a finite set Δ of β -normal sentences from $cwff_o(\Sigma)$.
- Let's compare our one-sided rules to standard two-sided sequent calculus rules (we use $\Delta * A$ to denote the set $\Delta \cup \{A\}$ (which is simply Δ if $A \in \Delta$)):

$$\frac{\Delta * A * B}{\Delta * (A \vee B)} \mathcal{G}(V_+) \quad \text{vs.} \quad \frac{\Gamma \implies \Delta * A * B}{\Gamma \implies \Delta * (A \vee B)} V\text{R}$$

$$\frac{\Delta * \neg A \quad \Delta * \neg B}{\Delta * \neg(A \vee B)} \mathcal{G}(V_-) \quad \text{vs.} \quad \frac{\Gamma * B \implies \Delta \quad \Gamma * A \implies \Delta}{\Gamma * A \vee B \implies \Delta} V\text{L}$$

©Benzmüller, 2006



ATPHOL06-[16] – p.362

Def.: Validity of Sequents



Def.: Validity of Sequents



- Given a sequent Δ , a model \mathcal{M} , and a class \mathfrak{M} of models, we say

- Given a sequent Δ , a model \mathcal{M} , and a class \mathfrak{M} of models, we say

Δ is valid for \mathcal{M} (or valid for \mathfrak{M}),

Def.: Validity of Sequents



- Given a sequent Δ , a model \mathcal{M} , and a class \mathfrak{M} of models, we say

Δ is valid for \mathcal{M} (or valid for \mathfrak{M}),

if $\mathcal{M} \models D$ for some $D \in \Delta$ (or Δ is valid for every $\mathcal{M} \in \mathfrak{M}$).

Def.: k-Admissibility of Rules



- We say a sequent calculus rule

Def.: k-Admissibility of Rules



- We say a sequent calculus rule

$$\frac{\Delta_1 \quad \dots \quad \Delta_n}{\Delta} r$$

Def.: k-Admissibility of Rules



- We say a sequent calculus rule

$$\frac{\Delta_1 \quad \dots \quad \Delta_n}{\Delta} r$$

is **admissible** in \mathcal{G}

Def.: k-Admissibility of Rules



- We say a sequent calculus rule

$$\frac{\Delta_1 \quad \dots \quad \Delta_n}{\Delta} r$$

is **admissible** in \mathcal{G}

if $\vdash_{\mathcal{G}} \Delta$ holds whenever $\vdash_{\mathcal{G}} \Delta_i$ for all $1 \leq i \leq n$.

Def.: k-Admissibility of Rules



- We say a sequent calculus rule

$$\frac{\Delta_1 \quad \dots \quad \Delta_n}{\Delta} r$$

is **admissible** in \mathcal{G}

if $\vdash_{\mathcal{G}} \Delta$ holds whenever $\vdash_{\mathcal{G}} \Delta_i$ for all $1 \leq i \leq n$.

- For any $k \geq 0$, we call an admissible rule r

Def.: k-Admissibility of Rules



- We say a sequent calculus rule

$$\frac{\Delta_1 \quad \dots \quad \Delta_n}{\Delta} r$$

is **admissible** in \mathcal{G}

if $\Vdash_{\mathcal{G}} \Delta$ holds whenever $\Vdash_{\mathcal{G}} \Delta_i$ for all $1 \leq i \leq n$.

- For any $k \geq 0$, we call an admissible rule r

k-admissible

Def.: k-Admissibility of Rules



- We say a sequent calculus rule

$$\frac{\Delta_1 \quad \dots \quad \Delta_n}{\Delta} r$$

is **admissible** in \mathcal{G}

if $\Vdash_{\mathcal{G}} \Delta$ holds whenever $\Vdash_{\mathcal{G}} \Delta_i$ for all $1 \leq i \leq n$.

- For any $k \geq 0$, we call an admissible rule r

k-admissible

if any instance of r can be replaced by a derivation with at most k additional proof steps.

Def.: k-Admissibility of Rules



- We say a sequent calculus rule

$$\frac{\Delta_1 \quad \dots \quad \Delta_n}{\Delta} r$$

is **admissible** in \mathcal{G}

if $\Vdash_{\mathcal{G}} \Delta$ holds whenever $\Vdash_{\mathcal{G}} \Delta_i$ for all $1 \leq i \leq n$.

- For any $k \geq 0$, we call an admissible rule r

k-admissible

if any instance of r can be replaced by a derivation with at most k additional proof steps.

- Remark: We use admissibility to obtain more general results.
In fact all rules that are later shown to be k -admissible are actually even k -derivable.

Def.: Sequent Calculus Rules



Basic Rules

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{A \text{ atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{}{\Delta * \neg\neg A} \mathcal{G}(\neg)$$

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{A \text{ atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{A \text{ atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{\Delta * A}{\Delta * \neg\neg A} \mathcal{G}(\neg)$$

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{\text{A atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{\Delta * A}{\Delta * \neg \neg A} \mathcal{G}(\neg)$$

$$\frac{}{\Delta * \neg(A \vee B)} \mathcal{G}(\vee_-)$$

©Benzmüller, 2006



ATPHOL06-[16] – p.365

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{\text{A atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{\Delta * A}{\Delta * \neg \neg A} \mathcal{G}(\neg)$$

$$\frac{\Delta * \neg A \quad \Delta * \neg B}{\Delta * \neg(A \vee B)} \mathcal{G}(\vee_-)$$

©Benzmüller, 2006



ATPHOL06-[16] – p.365

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{\text{A atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{\Delta * A}{\Delta * \neg \neg A} \mathcal{G}(\neg)$$

$$\frac{\Delta * \neg A \quad \Delta * \neg B}{\Delta * \neg(A \vee B)} \mathcal{G}(\vee_-)$$

$$\frac{}{\Delta * (A \vee B)} \mathcal{G}(\vee_+)$$

©Benzmüller, 2006



ATPHOL06-[16] – p.365

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{\text{A atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{\Delta * A}{\Delta * \neg \neg A} \mathcal{G}(\neg)$$

$$\frac{\Delta * \neg A \quad \Delta * \neg B}{\Delta * \neg(A \vee B)} \mathcal{G}(\vee_-)$$

$$\frac{\Delta * A * B}{\Delta * (A \vee B)} \mathcal{G}(\vee_+)$$

©Benzmüller, 2006



ATPHOL06-[16] – p.365

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{\text{A atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{\Delta * A}{\Delta * \neg \neg A} \mathcal{G}(\neg)$$

$$\frac{\Delta * \neg A \quad \Delta * \neg B}{\Delta * \neg(A \vee B)} \mathcal{G}(\vee_-)$$

$$\frac{\Delta * A * B}{\Delta * (A \vee B)} \mathcal{G}(\vee_+)$$

$$\frac{}{\Delta * \neg \Pi^\alpha A} \mathcal{G}(\Pi_-^C)$$

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{\text{A atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{\Delta * A}{\Delta * \neg \neg A} \mathcal{G}(\neg)$$

$$\frac{\Delta * \neg A \quad \Delta * \neg B}{\Delta * \neg(A \vee B)} \mathcal{G}(\vee_-)$$

$$\frac{\Delta * A * B}{\Delta * (A \vee B)} \mathcal{G}(\vee_+)$$

$$\frac{\Delta * \neg(AC) \downarrow_\beta \quad C \in \text{cwff}_\alpha(\Sigma)}{\Delta * \neg \Pi^\alpha A} \mathcal{G}(\Pi_-^C)$$

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{\text{A atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{\Delta * A}{\Delta * \neg \neg A} \mathcal{G}(\neg)$$

$$\frac{\Delta * \neg A \quad \Delta * \neg B}{\Delta * \neg(A \vee B)} \mathcal{G}(\vee_-)$$

$$\frac{\Delta * A * B}{\Delta * (A \vee B)} \mathcal{G}(\vee_+)$$

$$\frac{\Delta * \neg(AC) \downarrow_\beta \quad C \in \text{cwff}_\alpha(\Sigma)}{\Delta * \neg \Pi^\alpha A} \mathcal{G}(\Pi_-^C)$$

$$\frac{}{\Delta * \Pi^\alpha A} \mathcal{G}(\Pi_+^c)$$

Def.: Sequent Calculus Rules



Basic Rules

$$\frac{\text{A atomic (and } \beta\text{-normal)}}{\Delta * A * \neg A} \mathcal{G}(\text{init})$$

$$\frac{\Delta * A}{\Delta * \neg \neg A} \mathcal{G}(\neg)$$

$$\frac{\Delta * \neg A \quad \Delta * \neg B}{\Delta * \neg(A \vee B)} \mathcal{G}(\vee_-)$$

$$\frac{\Delta * A * B}{\Delta * (A \vee B)} \mathcal{G}(\vee_+)$$

$$\frac{\Delta * \neg(AC) \downarrow_\beta \quad C \in \text{cwff}_\alpha(\Sigma)}{\Delta * \neg \Pi^\alpha A} \mathcal{G}(\Pi_-^C)$$

$$\frac{\Delta * (Ac) \downarrow_\beta \quad c_\alpha \in \Sigma \text{ new}}{\Delta * \Pi^\alpha A} \mathcal{G}(\Pi_+^c)$$

Def.: Sequent Calculus Rules



Inversion Rule

Def.: Sequent Calculus Rules



Inversion Rule

$$\frac{}{\Delta * A} \mathcal{G}(Inv^-)$$

Weakening and Cut Rules

Weakening and Cut Rules

Def.: Sequent Calculus Rules



Inversion Rule

$$\frac{\Delta * \neg\neg A}{\Delta * A} \mathcal{G}(Inv^-)$$

Weakening and Cut Rules

Def.: Sequent Calculus Rules



Inversion Rule

$$\frac{\Delta * \neg\neg A}{\Delta * A} \mathcal{G}(Inv^-)$$

Weakening and Cut Rules

$$\frac{}{\Delta \cup \Delta'} \mathcal{G}(weak)$$

Def.: Sequent Calculus Rules



Def.: Sequent Calculus Rules



Inversion Rule

$$\frac{\Delta * \neg\neg A}{\Delta * A} \mathcal{G}(Inv^\neg)$$

Weakening and Cut Rules

$$\frac{\Delta}{\Delta \cup \Delta'} \mathcal{G}(weak)$$

Inversion Rule

$$\frac{\Delta * \neg\neg A}{\Delta * A} \mathcal{G}(Inv^\neg)$$

Weakening and Cut Rules

$$\frac{\Delta}{\Delta \cup \Delta'} \mathcal{G}(weak) \quad \frac{}{\Delta} \mathcal{G}(cut)$$



©Benzmüller, 2006

ATPHOL'06-[16] – p.366



ATPHOL'06-[16] – p.366

Def.: Sequent Calculus Rules



Inversion Rule

$$\frac{\Delta * \neg\neg A}{\Delta * A} \mathcal{G}(Inv^\neg)$$

Weakening and Cut Rules

$$\frac{\Delta}{\Delta \cup \Delta'} \mathcal{G}(weak)$$

$$\frac{\Delta * C \quad \Delta * \neg C}{\Delta} \mathcal{G}(cut)$$

ACC for Sequent Calculi



- For any sequent calculus \mathcal{G} we can define a class $\Gamma_\Sigma^{\mathcal{G}}$ of sets of sentences.



©Benzmüller, 2006

ATPHOL'06-[16] – p.366



ATPHOL'06-[16] – p.367

- For any sequent calculus \mathcal{G} we can define a class $\Gamma_{\Sigma}^{\mathcal{G}}$ of sets of sentences.
- Under certain assumptions, $\Gamma_{\Sigma}^{\mathcal{G}}$ is an abstract consistency class.

- For any sequent calculus \mathcal{G} we can define a class $\Gamma_{\Sigma}^{\mathcal{G}}$ of sets of sentences.
- Under certain assumptions, $\Gamma_{\Sigma}^{\mathcal{G}}$ is an abstract consistency class.
- First we adopt the notation $\neg\Phi$ and Φ_{β} for the sets $\{\neg A | A \in \Phi\}$ and $\{A_{\beta} | A \in \Phi\}$, resp., where $\Phi \subseteq \text{cwf}_{\circ}(\Sigma)$.

- For any sequent calculus \mathcal{G} we can define a class $\Gamma_{\Sigma}^{\mathcal{G}}$ of sets of sentences.
- Under certain assumptions, $\Gamma_{\Sigma}^{\mathcal{G}}$ is an abstract consistency class.
- First we adopt the notation $\neg\Phi$ and Φ_{β} for the sets $\{\neg A | A \in \Phi\}$ and $\{A_{\beta} | A \in \Phi\}$, resp., where $\Phi \subseteq \text{cwf}_{\circ}(\Sigma)$.
- Furthermore, we assume this use of \neg binds more strongly than \cup or $*$, so that $\neg\Phi \cup \Delta$ means $(\neg\Phi) \cup \Delta$ and $\neg\Phi * A$ means $(\neg\Phi) * A$.

Let \mathcal{G} be a sequent calculus.

Let \mathcal{G} be a sequent calculus. We define $\Gamma_{\Sigma}^{\mathcal{G}}$ to be the class of all finite $\Phi \subset \text{cwff}_o(\Sigma)$ such that $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta}$ does not hold.

Let \mathcal{G} be a sequent calculus such that $\mathcal{G}(Inv^{\neg})$ is admissible.

Lemma: Consequence of $\mathcal{G}(Inv^{\neg})$

Let \mathcal{G} be a sequent calculus such that $\mathcal{G}(Inv^{\neg})$ is admissible. For any finite sets Φ and Δ of sentences, if $\Phi \cup \neg\Delta \notin \Gamma_{\Sigma}^{\mathcal{G}}$, then $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} \cup \Delta \downarrow_{\beta}$ holds.

Lemma: Consequence of $\mathcal{G}(Inv^{\neg})$

Let \mathcal{G} be a sequent calculus such that $\mathcal{G}(Inv^{\neg})$ is admissible. For any finite sets Φ and Δ of sentences, if $\Phi \cup \neg\Delta \notin \Gamma_{\Sigma}^{\mathcal{G}}$, then $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} \cup \Delta \downarrow_{\beta}$ holds.

Proof:

Lemma: Consequence of $\mathcal{G}(Inv^\neg)$



Let \mathcal{G} be a sequent calculus such that $\mathcal{G}(Inv^\neg)$ is admissible. For any finite sets Φ and Δ of sentences, if $\Phi \cup \neg\Delta \notin \Gamma_\Sigma^{\mathcal{G}}$, then $\vdash_{\mathcal{G}} \neg\Phi \downarrow_\beta \cup \Delta \downarrow_\beta$ holds.

Proof: Suppose $\Phi \cup \neg\Delta \notin \Gamma_\Sigma^{\mathcal{G}}$.

Lemma: Consequence of $\mathcal{G}(Inv^\neg)$



Let \mathcal{G} be a sequent calculus such that $\mathcal{G}(Inv^\neg)$ is admissible. For any finite sets Φ and Δ of sentences, if $\Phi \cup \neg\Delta \notin \Gamma_\Sigma^{\mathcal{G}}$, then $\vdash_{\mathcal{G}} \neg\Phi \downarrow_\beta \cup \Delta \downarrow_\beta$ holds.

Proof: Suppose $\Phi \cup \neg\Delta \notin \Gamma_\Sigma^{\mathcal{G}}$. By definition, $\vdash_{\mathcal{G}} \neg\Phi \downarrow_\beta \cup \neg\neg\Delta \downarrow_\beta$ holds.



Let \mathcal{G} be a sequent calculus such that $\mathcal{G}(Inv^\neg)$ is admissible. For any finite sets Φ and Δ of sentences, if $\Phi \cup \neg\Delta \notin \Gamma_\Sigma^{\mathcal{G}}$, then $\vdash_{\mathcal{G}} \neg\Phi \downarrow_\beta \cup \Delta \downarrow_\beta$ holds.

Proof: Suppose $\Phi \cup \neg\Delta \notin \Gamma_\Sigma^{\mathcal{G}}$. By definition, $\vdash_{\mathcal{G}} \neg\Phi \downarrow_\beta \cup \neg\neg\Delta \downarrow_\beta$ holds. Applying $\mathcal{G}(Inv^\neg)$ to each member of $\Delta \downarrow_\beta$, we have $\vdash_{\mathcal{G}} \neg\Phi \downarrow_\beta \cup \Delta \downarrow_\beta$.



Thm.: Sufficient Conditions for $\Gamma_\Sigma^{\mathcal{G}} \in \mathfrak{Acc}_\beta$



Let \mathcal{G} be a sequent calculus.



Thm.: Sufficient Conditions for $\Gamma_{\Sigma}^{\mathcal{G}} \in \mathfrak{Acc}_{\beta}$

Let \mathcal{G} be a sequent calculus. If the rules $\mathcal{G}(Inv^-)$, $\mathcal{G}(\neg)$, $\mathcal{G}(weak)$, $\mathcal{G}(init)$, $\mathcal{G}(\vee_-)$, $\mathcal{G}(\vee_+)$, $\mathcal{G}(\Pi_-^C)$ and $\mathcal{G}(\Pi_+^c)$ are admissible in \mathcal{G} , then $\Gamma_{\Sigma}^{\mathcal{G}} \in \mathfrak{Acc}_{\beta}$.



Thm.: Sufficient Conditions for $\Gamma_{\Sigma}^{\mathcal{G}} \in \mathfrak{Acc}_{\beta}$

Let \mathcal{G} be a sequent calculus. If the rules $\mathcal{G}(Inv^-)$, $\mathcal{G}(\neg)$, $\mathcal{G}(weak)$, $\mathcal{G}(init)$, $\mathcal{G}(\vee_-)$, $\mathcal{G}(\vee_+)$, $\mathcal{G}(\Pi_-^C)$ and $\mathcal{G}(\Pi_+^c)$ are admissible in \mathcal{G} , then $\Gamma_{\Sigma}^{\mathcal{G}} \in \mathfrak{Acc}_{\beta}$.



Thm.: Sufficient Conditions for $\Gamma_{\Sigma}^{\mathcal{G}} \in \mathfrak{Acc}_{\beta}$

Let \mathcal{G} be a sequent calculus. If the rules $\mathcal{G}(Inv^-)$, $\mathcal{G}(\neg)$, $\mathcal{G}(weak)$, $\mathcal{G}(init)$, $\mathcal{G}(\vee_-)$, $\mathcal{G}(\vee_+)$, $\mathcal{G}(\Pi_-^C)$ and $\mathcal{G}(\Pi_+^c)$ are admissible in \mathcal{G} , then $\Gamma_{\Sigma}^{\mathcal{G}} \in \mathfrak{Acc}_{\beta}$.



Proof: We prove $\Gamma_{\Sigma}^{\mathcal{G}}$ is closed under subsets and satisfies ∇_c , ∇_{\neg} , ∇_{\vee} , ∇_{\wedge} and ∇_{β} . The remaining conditions are proven analogously.

- Suppose $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$, If $\Phi_0 \subseteq \Phi$ and $\Phi_0 \notin \Gamma_{\Sigma}^{\mathcal{G}}$, then $\vdash_{\mathcal{G}} \neg \Phi_0 \downarrow_{\beta}$ and so $\vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta}$ by admissibility of $\mathcal{G}(weak)$. Hence $\Gamma_{\Sigma}^{\mathcal{G}}$ is closed under subsets.

Thm.: Sufficient Conditions for $\Gamma_{\Sigma}^{\mathcal{G}} \in \mathfrak{Acc}_{\beta}$

Let \mathcal{G} be a sequent calculus. If the rules $\mathcal{G}(Inv^-)$, $\mathcal{G}(\neg)$, $\mathcal{G}(weak)$, $\mathcal{G}(init)$, $\mathcal{G}(\vee_-)$, $\mathcal{G}(\vee_+)$, $\mathcal{G}(\Pi_-^C)$ and $\mathcal{G}(\Pi_+^c)$ are admissible in \mathcal{G} , then $\Gamma_{\Sigma}^{\mathcal{G}} \in \mathfrak{Acc}_{\beta}$.



Proof: We prove $\Gamma_{\Sigma}^{\mathcal{G}}$ is closed under subsets and satisfies ∇_c , ∇_{\neg} , ∇_{\vee} , ∇_{\wedge} and ∇_{β} . The remaining conditions are proven analogously.

- Suppose $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$, If $\Phi_0 \subseteq \Phi$ and $\Phi_0 \notin \Gamma_{\Sigma}^{\mathcal{G}}$, then $\vdash_{\mathcal{G}} \neg \Phi_0 \downarrow_{\beta}$ and so $\vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta}$ by admissibility of $\mathcal{G}(weak)$. Hence $\Gamma_{\Sigma}^{\mathcal{G}}$ is closed under subsets.
- Suppose $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$ and $\mathbf{A}, \neg \mathbf{A} \in \Phi$ where \mathbf{A} is atomic. By admissibility of $\mathcal{G}(init)$, $\vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * \mathbf{A} \downarrow_{\beta}$ since $\neg \mathbf{A} \downarrow_{\beta} \in \neg \Phi \downarrow_{\beta}$. By admissibility of $\mathcal{G}(\neg)$, $\vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta}$ since $\neg \neg \mathbf{A} \downarrow_{\beta} \in \neg \Phi \downarrow_{\beta}$, contradicting $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$. Thus ∇_c holds.

Thm.: Sufficient Condition for $\Gamma_{\Sigma}^{\mathcal{G}} \in \text{Acc}_{\beta}$



Proof (contd.):

Thm.: Sufficient Condition for $\Gamma_{\Sigma}^{\mathcal{G}} \in \text{Acc}_{\beta}$



Proof (contd.):

- Suppose $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$, $\neg\neg A \in \Phi$ and $\Phi * A \notin \Gamma_{\Sigma}^{\mathcal{G}}$. Hence $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg A \downarrow_{\beta}$ and so $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg\neg\neg A \downarrow_{\beta}$ by admissibility of $\mathcal{G}(\neg)$. Since $\neg\neg A \in \Phi$, we know $\neg\Phi \downarrow_{\beta}$ is equal to $\neg\Phi \downarrow_{\beta} * \neg\neg\neg A \downarrow_{\beta}$. Hence $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta}$, contradicting $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$. Thus ∇_{\neg} holds.

Thm.: Sufficient Condition for $\Gamma_{\Sigma}^{\mathcal{G}} \in \text{Acc}_{\beta}$



Proof (contd.):

- Suppose $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$, $\neg\neg A \in \Phi$ and $\Phi * A \notin \Gamma_{\Sigma}^{\mathcal{G}}$. Hence $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg A \downarrow_{\beta}$ and so $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg\neg\neg A \downarrow_{\beta}$ by admissibility of $\mathcal{G}(\neg)$. Since $\neg\neg A \in \Phi$, we know $\neg\Phi \downarrow_{\beta}$ is equal to $\neg\Phi \downarrow_{\beta} * \neg\neg\neg A \downarrow_{\beta}$. Hence $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta}$, contradicting $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$. Thus ∇_{\neg} holds.
- Suppose $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$, $(A \vee B) \in \Phi$, $\Phi * A \notin \Gamma_{\Sigma}^{\mathcal{G}}$ and $\Phi * B \notin \Gamma_{\Sigma}^{\mathcal{G}}$. Hence $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg A \downarrow_{\beta}$ and $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg B \downarrow_{\beta}$. Applying $\mathcal{G}(\vee_{-})$, we have $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta}$ since $\neg(A \vee B) \downarrow_{\beta} \in \neg\Phi \downarrow_{\beta}$, contradicting $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$. Thus ∇_{\vee} holds.

Thm.: Sufficient Condition for $\Gamma_{\Sigma}^{\mathcal{G}} \in \text{Acc}_{\beta}$



Proof (contd.):

- Suppose $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$, $\neg\neg A \in \Phi$ and $\Phi * A \notin \Gamma_{\Sigma}^{\mathcal{G}}$. Hence $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg A \downarrow_{\beta}$ and so $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg\neg\neg A \downarrow_{\beta}$ by admissibility of $\mathcal{G}(\neg)$. Since $\neg\neg A \in \Phi$, we know $\neg\Phi \downarrow_{\beta}$ is equal to $\neg\Phi \downarrow_{\beta} * \neg\neg\neg A \downarrow_{\beta}$. Hence $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta}$, contradicting $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$. Thus ∇_{\neg} holds.
- Suppose $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$, $(A \vee B) \in \Phi$, $\Phi * A \notin \Gamma_{\Sigma}^{\mathcal{G}}$ and $\Phi * B \notin \Gamma_{\Sigma}^{\mathcal{G}}$. Hence $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg A \downarrow_{\beta}$ and $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta} * \neg B \downarrow_{\beta}$. Applying $\mathcal{G}(\vee_{-})$, we have $\Vdash_{\mathcal{G}} \neg\Phi \downarrow_{\beta}$ since $\neg(A \vee B) \downarrow_{\beta} \in \neg\Phi \downarrow_{\beta}$, contradicting $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$. Thus ∇_{\vee} holds.
- By a similar argument, admissibility of $\mathcal{G}(II_{-}^C)$ implies ∇_{\forall} .

Thm.: Sufficient Condition for $\Gamma_{\Sigma}^G \in \text{Acc}_{\beta}$



Proof (contd.):

Thm.: Sufficient Condition for $\Gamma_{\Sigma}^G \in \text{Acc}_{\beta}$



Proof (contd.):

- Suppose $\Phi \in \Gamma_{\Sigma}^G$, $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B \notin \Gamma_{\Sigma}^G$. By Lemma 'Consequence of $\mathcal{G}(Inv^-)$ ', $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * A \downarrow_{\beta} * B \downarrow_{\beta}$. Applying $\mathcal{G}(\vee_+)$, we have $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * (A \vee B) \downarrow_{\beta}$. Applying $\mathcal{G}(\neg)$, we have $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta}$ since $\neg(A \vee B) \in \Phi$, contradicting $\Phi \in \Gamma_{\Sigma}^G$. Thus ∇_{Λ} holds.

Thm.: Sufficient Condition for $\Gamma_{\Sigma}^G \in \text{Acc}_{\beta}$



Proof (contd.):

- Suppose $\Phi \in \Gamma_{\Sigma}^G$, $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B \notin \Gamma_{\Sigma}^G$. By Lemma 'Consequence of $\mathcal{G}(Inv^-)$ ', $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * A \downarrow_{\beta} * B \downarrow_{\beta}$. Applying $\mathcal{G}(\vee_+)$, we have $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * (A \vee B) \downarrow_{\beta}$. Applying $\mathcal{G}(\neg)$, we have $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta}$ since $\neg(A \vee B) \in \Phi$, contradicting $\Phi \in \Gamma_{\Sigma}^G$. Thus ∇_{Λ} holds.
- By a similar argument, admissibility of $\mathcal{G}(\Pi_+^c)$, $\mathcal{G}(Inv^-)$ and $\mathcal{G}(\neg)$ imply ∇_{\exists} .

Thm.: Sufficient Condition for $\Gamma_{\Sigma}^G \in \text{Acc}_{\beta}$



Proof (contd.):

- Suppose $\Phi \in \Gamma_{\Sigma}^G$, $\neg(A \vee B) \in \Phi$ and $\Phi * \neg A * \neg B \notin \Gamma_{\Sigma}^G$. By Lemma 'Consequence of $\mathcal{G}(Inv^-)$ ', $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * A \downarrow_{\beta} * B \downarrow_{\beta}$. Applying $\mathcal{G}(\vee_+)$, we have $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * (A \vee B) \downarrow_{\beta}$. Applying $\mathcal{G}(\neg)$, we have $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta}$ since $\neg(A \vee B) \in \Phi$, contradicting $\Phi \in \Gamma_{\Sigma}^G$. Thus ∇_{Λ} holds.
- By a similar argument, admissibility of $\mathcal{G}(\Pi_+^c)$, $\mathcal{G}(Inv^-)$ and $\mathcal{G}(\neg)$ imply ∇_{\exists} .
- Suppose $\Phi \in \Gamma_{\Sigma}^G$, $A \in \Phi$, $A =_{\beta} B$ and $\Phi * B \notin \Gamma_{\Sigma}^G$. Hence $\Vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * B \downarrow_{\beta}$, contradicting $A \downarrow_{\beta} \in \Phi \downarrow_{\beta}$ and $\Phi \in \Gamma_{\Sigma}^G$. Thus ∇_{β} holds.

Thm.: Saturation and Cut



Let \mathcal{G} be a sequent calculus.

Thm.: Saturation and Cut



Let \mathcal{G} be a sequent calculus.

1. If $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} , then $\Gamma_{\Sigma}^{\mathcal{G}}$ is saturated.

Thm.: Saturation and Cut



Let \mathcal{G} be a sequent calculus.

1. If $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} , then $\Gamma_{\Sigma}^{\mathcal{G}}$ is saturated.
2. If $\mathcal{G}(\neg)$ and $\mathcal{G}(\text{Inv}^{\neg})$ are admissible in \mathcal{G} and $\Gamma_{\Sigma}^{\mathcal{G}}$ is saturated, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Thm.: Saturation and Cut



Let \mathcal{G} be a sequent calculus.

1. If $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} , then $\Gamma_{\Sigma}^{\mathcal{G}}$ is saturated.
2. If $\mathcal{G}(\neg)$ and $\mathcal{G}(\text{Inv}^{\neg})$ are admissible in \mathcal{G} and $\Gamma_{\Sigma}^{\mathcal{G}}$ is saturated, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Proof: Suppose $\mathcal{G}(\text{cut})$ is admissible, $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$, $\mathbf{A} \in \text{cwf}_o(\Sigma)$, $\Phi * \mathbf{A} \notin \Gamma_{\Sigma}^{\mathcal{G}}$ and $\Phi * \neg \mathbf{A} \notin \Gamma_{\Sigma}^{\mathcal{G}}$. Hence $\vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * \neg \mathbf{A} \downarrow_{\beta}$ and $\vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta} * \neg \neg \mathbf{A} \downarrow_{\beta}$. Using $\mathcal{G}(\text{cut})$, we have $\vdash_{\mathcal{G}} \neg \Phi \downarrow_{\beta}$, contradicting $\Phi \in \Gamma_{\Sigma}^{\mathcal{G}}$.

Thm.: Saturation and Cut



Let \mathcal{G} be a sequent calculus.

1. If $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} , then $\Gamma_\Sigma^{\mathcal{G}}$ is saturated.
2. If $\mathcal{G}(\neg)$ and $\mathcal{G}(\text{Inv}^\neg)$ are admissible in \mathcal{G} and $\Gamma_\Sigma^{\mathcal{G}}$ is saturated, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Proof: Suppose $\mathcal{G}(\text{cut})$ is admissible, $\Phi \in \Gamma_\Sigma^{\mathcal{G}}$, $A \in \text{cwf}_o(\Sigma)$, $\Phi * A \notin \Gamma_\Sigma^{\mathcal{G}}$ and $\Phi * \neg A \notin \Gamma_\Sigma^{\mathcal{G}}$. Hence $\vdash_{\mathcal{G}} \neg \Phi \downarrow_\beta * \neg A \downarrow_\beta$ and $\vdash_{\mathcal{G}} \neg \Phi \downarrow_\beta * \neg \neg A \downarrow_\beta$. Using $\mathcal{G}(\text{cut})$, we have $\vdash_{\mathcal{G}} \neg \Phi \downarrow_\beta$, contradicting $\Phi \in \Gamma_\Sigma^{\mathcal{G}}$.

Suppose $\Gamma_\Sigma^{\mathcal{G}}$ is saturated, $\vdash_{\mathcal{G}} \Delta * C$ and $\vdash_{\mathcal{G}} \Delta * \neg C$ hold but $\vdash_{\mathcal{G}} \Delta$ does not. Applying $\mathcal{G}(\neg)$ to every member of Δ and to C we have

$\vdash_{\mathcal{G}} \neg \neg \Delta * \neg \neg C$ and $\vdash_{\mathcal{G}} \neg \neg \Delta * \neg C$. By Lemma 'Consequence of $\mathcal{G}(\text{Inv}^\neg)$ ', we know $\neg \Delta \in \Gamma_\Sigma^{\mathcal{G}}$. By saturation, we must have $\neg \Delta * C \in \Gamma_\Sigma^{\mathcal{G}}$ or $\neg \Delta * \neg C \in \Gamma_\Sigma^{\mathcal{G}}$. The first case contradicts $\vdash_{\mathcal{G}} \neg \neg \Delta * \neg C$ while the second case contradicts $\vdash_{\mathcal{G}} \neg \neg \Delta * \neg \neg C$.

Def.: Saturated Extension



Since saturation is equivalent to admissibility of cut, we need (are interested in) weaker conditions than saturation. A natural condition to consider is the existence of saturated extensions.

Def. (Saturated Extension): Let $\Gamma_\Sigma, \Gamma'_\Sigma \in \mathfrak{Acc}_*$ be abstract consistency classes. We say Γ'_Σ is an **extension** of Γ_Σ if $\Phi \in \Gamma'_\Sigma$ for every sufficiently Σ -pure $\Phi \in \Gamma_\Sigma$. We say Γ'_Σ is a **saturated extension** of Γ_Σ if Γ'_Σ is saturated and an extension of Γ_Σ .

Def.: Saturated Extension



Since saturation is equivalent to admissibility of cut, we need (are interested in) weaker conditions than saturation. A natural condition to consider is the existence of saturated extensions.

Ex.: ACC without Saturated Extension



There exist abstract consistency classes Γ in $\mathfrak{Acc}_{\beta\text{ff}}$ which have no saturated extension.

Ex.: ACC without Saturated Extension



There exist abstract consistency classes Γ in $\text{Acc}_{\beta\text{fb}}$ which have no saturated extension.

Example:

Ex.: ACC without Saturated Extension



There exist abstract consistency classes Γ in $\text{Acc}_{\beta\text{fb}}$ which have no saturated extension.

Example:

Let $a_o, b_o, q_{o \rightarrow o} \in \Sigma$ and $\Phi := \{a, b, (qa), \neg(qb)\}$. We construct an abstract consistency class Γ_Σ from Φ by first building the closure Φ' of Φ under relation $=_\beta$ and then taking the power set of Φ' . It is easy to check that this Γ_Σ is in $\text{Acc}_{\beta\text{fb}}$.

Ex.: ACC without Saturated Extension



There exist abstract consistency classes Γ in $\text{Acc}_{\beta\text{fb}}$ which have no saturated extension.

Example:

Let $a_o, b_o, q_{o \rightarrow o} \in \Sigma$ and $\Phi := \{a, b, (qa), \neg(qb)\}$. We construct an abstract consistency class Γ_Σ from Φ by first building the closure Φ' of Φ under relation $=_\beta$ and then taking the power set of Φ' . It is easy to check that this Γ_Σ is in $\text{Acc}_{\beta\text{fb}}$. Suppose we have a saturated extension Γ'_Σ of Γ_Σ in $\text{Acc}_{\beta\text{fb}}$.

Ex.: ACC without Saturated Extension



There exist abstract consistency classes Γ in $\text{Acc}_{\beta\text{fb}}$ which have no saturated extension.

Example:

Let $a_o, b_o, q_{o \rightarrow o} \in \Sigma$ and $\Phi := \{a, b, (qa), \neg(qb)\}$. We construct an abstract consistency class Γ_Σ from Φ by first building the closure Φ' of Φ under relation $=_\beta$ and then taking the power set of Φ' . It is easy to check that this Γ_Σ is in $\text{Acc}_{\beta\text{fb}}$. Suppose we have a saturated extension Γ'_Σ of Γ_Σ in $\text{Acc}_{\beta\text{fb}}$. Then $\Phi \in \Gamma'_\Sigma$ since Φ is finite (hence sufficiently pure).

Ex.: ACC without Saturated Extension



There exist abstract consistency classes Γ in $\text{Acc}_{\beta\text{ff}}$ which have no saturated extension.

Example:

Let $a_o, b_o, q_{o \rightarrow o} \in \Sigma$ and $\Phi := \{a, b, (qa), \neg(qb)\}$. We construct an abstract consistency class Γ_Σ from Φ by first building the closure Φ' of Φ under relation $=_\beta$ and then taking the power set of Φ' . It is easy to check that this Γ_Σ is in $\text{Acc}_{\beta\text{ff}}$. Suppose we have a saturated extension Γ'_Σ of Γ_Σ in $\text{Acc}_{\beta\text{ff}}$. Then $\Phi \in \Gamma'_\Sigma$ since Φ is finite (hence sufficiently pure). By saturation, $\Phi * (a \doteq^o b) \in \Gamma'_\Sigma$ or $\Phi * \neg(a \doteq^o b) \in \Gamma'_\Sigma$.

Ex.: ACC without Saturated Extension



There exist abstract consistency classes Γ in $\text{Acc}_{\beta\text{ff}}$ which have no saturated extension.

Example:

Let $a_o, b_o, q_{o \rightarrow o} \in \Sigma$ and $\Phi := \{a, b, (qa), \neg(qb)\}$. We construct an abstract consistency class Γ_Σ from Φ by first building the closure Φ' of Φ under relation $=_\beta$ and then taking the power set of Φ' . It is easy to check that this Γ_Σ is in $\text{Acc}_{\beta\text{ff}}$. Suppose we have a saturated extension Γ'_Σ of Γ_Σ in $\text{Acc}_{\beta\text{ff}}$. Then $\Phi \in \Gamma'_\Sigma$ since Φ is finite (hence sufficiently pure). By saturation, $\Phi * (a \doteq^o b) \in \Gamma'_\Sigma$ or $\Phi * \neg(a \doteq^o b) \in \Gamma'_\Sigma$. In the first case, applying ∇_q with the constant q , ∇_β , ∇_V and ∇_C contradicts $(qa), \neg(qb) \in \Phi$. In the second case, ∇_b and ∇_C contradict $a, b \in \Phi$.

Ex.: ACC without Saturated Extension



There exist abstract consistency classes Γ in $\text{Acc}_{\beta\text{ff}}$ which have no saturated extension.

Example:

Let $a_o, b_o, q_{o \rightarrow o} \in \Sigma$ and $\Phi := \{a, b, (qa), \neg(qb)\}$. We construct an abstract consistency class Γ_Σ from Φ by first building the closure Φ' of Φ under relation $=_\beta$ and then taking the power set of Φ' . It is easy to check that this Γ_Σ is in $\text{Acc}_{\beta\text{ff}}$. Suppose we have a saturated extension Γ'_Σ of Γ_Σ in $\text{Acc}_{\beta\text{ff}}$. Then $\Phi \in \Gamma'_\Sigma$ since Φ is finite (hence sufficiently pure). By saturation, $\Phi * (a \doteq^o b) \in \Gamma'_\Sigma$ or $\Phi * \neg(a \doteq^o b) \in \Gamma'_\Sigma$. In the first case, applying ∇_V with the constant q , ∇_β , ∇_V and ∇_C contradicts $(qa), \neg(qb) \in \Phi$.

Existence of Saturated Extensions and Cut



Existence of any saturated extension of a sound sequent calculus \mathcal{G} implies admissibility of cut:

Existence of Saturated Extensions and Cut



Existence of any saturated extension of a sound sequent calculus \mathcal{G} implies admissibility of cut:

Let \mathcal{G} be a sequent calculus which is sound for $\mathfrak{M}_*(\Sigma)$. If $\Gamma_\Sigma^{\mathcal{G}}$ has a saturated extension $\Gamma'_\Sigma \in \mathfrak{Acc}_*$, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Existence of Saturated Extensions and Cut



Existence of any saturated extension of a sound sequent calculus \mathcal{G} implies admissibility of cut:

Let \mathcal{G} be a sequent calculus which is sound for $\mathfrak{M}_*(\Sigma)$. If $\Gamma_\Sigma^{\mathcal{G}}$ has a saturated extension $\Gamma'_\Sigma \in \mathfrak{Acc}_*$, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Proof: Suppose $\Gamma'_\Sigma \in \mathfrak{Acc}_*$ is a saturated extension of $\Gamma_\Sigma^{\mathcal{G}}$.



Existence of Saturated Extensions and Cut



Existence of any saturated extension of a sound sequent calculus \mathcal{G} implies admissibility of cut:

Let \mathcal{G} be a sequent calculus which is sound for $\mathfrak{M}_*(\Sigma)$. If $\Gamma_\Sigma^{\mathcal{G}}$ has a saturated extension $\Gamma'_\Sigma \in \mathfrak{Acc}_*$, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Proof: Suppose $\Gamma'_\Sigma \in \mathfrak{Acc}_*$ is a saturated extension of $\Gamma_\Sigma^{\mathcal{G}}$. Assume $\vdash_{\mathcal{G}} \Delta * C$ and $\vdash_{\mathcal{G}} \Delta * \neg C$ hold and $\vdash_{\mathcal{G}} \Delta$ does not.

Existence of Saturated Extensions and Cut



Existence of any saturated extension of a sound sequent calculus \mathcal{G} implies admissibility of cut:

Let \mathcal{G} be a sequent calculus which is sound for $\mathfrak{M}_*(\Sigma)$. If $\Gamma_\Sigma^{\mathcal{G}}$ has a saturated extension $\Gamma'_\Sigma \in \mathfrak{Acc}_*$, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Proof: Suppose $\Gamma'_\Sigma \in \mathfrak{Acc}_*$ is a saturated extension of $\Gamma_\Sigma^{\mathcal{G}}$. Assume $\vdash_{\mathcal{G}} \Delta * C$ and $\vdash_{\mathcal{G}} \Delta * \neg C$ hold and $\vdash_{\mathcal{G}} \Delta$ does not. By Lemma 'Consequence of $\mathcal{G}(\text{Inv}^-)$ ', we know $\neg \Delta \in \Gamma_\Sigma^{\mathcal{G}}$.



Existence of Saturated Extensions and Cut



Existence of any saturated extension of a sound sequent calculus \mathcal{G} implies admissibility of cut:

Let \mathcal{G} be a sequent calculus which is sound for $\mathfrak{M}_*(\Sigma)$. If $\Gamma_\Sigma^{\mathcal{G}}$ has a saturated extension $\Gamma'_\Sigma \in \mathfrak{Acc}_*$, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Proof: Suppose $\Gamma'_\Sigma \in \mathfrak{Acc}_*$ is a saturated extension of $\Gamma_\Sigma^{\mathcal{G}}$. Assume $\vdash_{\mathcal{G}} \Delta * C$ and $\vdash_{\mathcal{G}} \Delta * \neg C$ hold and $\vdash_{\mathcal{G}} \Delta$ does not. By Lemma 'Consequence of $\mathcal{G}(Inv^\neg)$ ', we know $\neg\Delta \in \Gamma_\Sigma^{\mathcal{G}}$. Since $\neg\Delta$ is finite (hence sufficiently pure), $\neg\Delta \in \Gamma'_\Sigma$.

Existence of Saturated Extensions and Cut



Existence of any saturated extension of a sound sequent calculus \mathcal{G} implies admissibility of cut:

Let \mathcal{G} be a sequent calculus which is sound for $\mathfrak{M}_*(\Sigma)$. If $\Gamma_\Sigma^{\mathcal{G}}$ has a saturated extension $\Gamma'_\Sigma \in \mathfrak{Acc}_*$, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Proof: Suppose $\Gamma'_\Sigma \in \mathfrak{Acc}_*$ is a saturated extension of $\Gamma_\Sigma^{\mathcal{G}}$. Assume $\vdash_{\mathcal{G}} \Delta * C$ and $\vdash_{\mathcal{G}} \Delta * \neg C$ hold and $\vdash_{\mathcal{G}} \Delta$ does not. By Lemma 'Consequence of $\mathcal{G}(Inv^\neg)$ ', we know $\neg\Delta \in \Gamma_\Sigma^{\mathcal{G}}$. Since $\neg\Delta$ is finite (hence sufficiently pure), $\neg\Delta \in \Gamma'_\Sigma$. By the model existence theorem for saturated abstract consistency classes, there is a model $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ such that $\mathcal{M} \models \neg\Delta$.



Existence of Saturated Extensions and Cut



Existence of any saturated extension of a sound sequent calculus \mathcal{G} implies admissibility of cut:

Let \mathcal{G} be a sequent calculus which is sound for $\mathfrak{M}_*(\Sigma)$. If $\Gamma_\Sigma^{\mathcal{G}}$ has a saturated extension $\Gamma'_\Sigma \in \mathfrak{Acc}_*$, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Proof: Suppose $\Gamma'_\Sigma \in \mathfrak{Acc}_*$ is a saturated extension of $\Gamma_\Sigma^{\mathcal{G}}$. Assume $\vdash_{\mathcal{G}} \Delta * C$ and $\vdash_{\mathcal{G}} \Delta * \neg C$ hold and $\vdash_{\mathcal{G}} \Delta$ does not. By Lemma 'Consequence of $\mathcal{G}(Inv^\neg)$ ', we know $\neg\Delta \in \Gamma_\Sigma^{\mathcal{G}}$. Since $\neg\Delta$ is finite (hence sufficiently pure), $\neg\Delta \in \Gamma'_\Sigma$. By the model existence theorem for saturated abstract consistency classes, there is a model $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ such that $\mathcal{M} \models \neg\Delta$. By soundness of \mathcal{G} , we know both $\Delta * C$ and $\Delta * \neg C$ must be valid in \mathcal{M} .

Existence of Saturated Extensions and Cut



Existence of any saturated extension of a sound sequent calculus \mathcal{G} implies admissibility of cut:

Let \mathcal{G} be a sequent calculus which is sound for $\mathfrak{M}_*(\Sigma)$. If $\Gamma_\Sigma^{\mathcal{G}}$ has a saturated extension $\Gamma'_\Sigma \in \mathfrak{Acc}_*$, then $\mathcal{G}(\text{cut})$ is admissible in \mathcal{G} .

Proof: Suppose $\Gamma'_\Sigma \in \mathfrak{Acc}_*$ is a saturated extension of $\Gamma_\Sigma^{\mathcal{G}}$. Assume $\vdash_{\mathcal{G}} \Delta * C$ and $\vdash_{\mathcal{G}} \Delta * \neg C$ hold and $\vdash_{\mathcal{G}} \Delta$ does not. By Lemma 'Consequence of $\mathcal{G}(Inv^\neg)$ ', we know $\neg\Delta \in \Gamma_\Sigma^{\mathcal{G}}$. Since $\neg\Delta$ is finite (hence sufficiently pure), $\neg\Delta \in \Gamma'_\Sigma$. By the model existence theorem for saturated abstract consistency classes, there is a model $\mathcal{M} \in \mathfrak{M}_*(\Sigma)$ such that $\mathcal{M} \models \neg\Delta$. By soundness of \mathcal{G} , we know both $\Delta * C$ and $\Delta * \neg C$ must be valid in \mathcal{M} . Since $\mathcal{M} \models \neg\Delta$, we must have $\mathcal{M} \models C$ and $\mathcal{M} \models \neg C$, a contradiction.

