

# Computational Logic

Vincent van Oostrom  
Course/slides by Aart Middeldorp

Department of Computer Science  
University of Innsbruck

SS 2020



# Outline

- Overview of this lecture
- Natural Deduction
- $\lambda$ -calculus
- Strong Normalization by Strong Computability
- Curry–Howard Isomorphism
- Exercises
- Further Reading

We continue focusing on the **structural** and **representational** aspects of proofs. Last week, we have seen how **combinatory logic** (CL) **terms** could be employed to represent Hilbert System **proofs** for the **implicational fragment** of **intuitionistic** propositional logic. More precisely, typed CL-terms correspond to proofs in Hilbert Systems restricted to Axiom Schemes 1 and 2, the **types** of the CL-terms correspond to the **formulas** proven, and **weak-reduction**  $\rightarrow_w$  of CL terms corresponds to **normalization** of the proofs. These correspondences together constitute the **Curry–Howard** isomorphism. The isomorphism extends to larger fragments (including conjunction, disjunction, etc.) of (intuitionistic) logic, but we will not discuss that in this course.

Instead, we focus on establishing a Curry–Howard isomorphism for the same, implicational, fragment of intuitionistic logic, but for **Natural Deduction** (ND) instead of the Hilbert System (H). In particular, we introduce the  **$\lambda$ -calculus** as a **term** calculus such that **typed**  $\lambda$ -terms correspond to proofs in ND. More precisely,  $\lambda$ -terms are terms constructed from **variables**, **applications**, and  **$\lambda$ -abstractions**, which correspond to (names of) **assumptions**, **Implication Elimination**, and **Implication Introduction** in ND, respectively.

We introduce  **$\beta$ -reduction**  $\rightarrow_{\beta}$  of typed  $\lambda$ -terms, show that it corresponds to **normalization** of ND proofs, and that every  $\lambda$ -term has a (unique) **normal form**, i.e. that every ND can be normalized (doesn't contain an Implication Introduction immediately followed by an Implication Elimination). With the types and formulas as before, this constitutes the Curry–Howard isomorphism between Natural Deduction and the (typed)  $\lambda$ -calculus.

We finish with relating both term-calculi, i.e. combinatory logic and the  $\lambda$ -calculus. In particular, we show that every  $\lambda$ -term can be translated to a CL term of the same type, and vice versa. **Bracket abstraction** is at the heart of this translation. Via the Curry–Howard isomorphism, it allows to translate a Natural Deduction proof of a formula (using Implication Introduction) into a Hilbert System proof (using Axiom Schemes 1 and 2). The translations establish, e.g., that ND is sound and complete for Kripke semantics, since Hilbert Systems (restricted to Axiom Schemes 1 and 2) are, as we showed last time.

## Part I: Propositional Logic

compactness, completeness, Hilbert systems, Hintikka's lemma, interpolation, logical consequence, model existence theorem, propositional semantic tableaux, soundness

## Part II: First-Order Logic

compactness, completeness, Craig's interpolation theorem, cut elimination, first-order semantic tableaux, Herbrand models, Herbrand's theorem, Hilbert systems, Hintikka's lemma, Löwenheim–Skolem, logical consequence, model existence theorem, prenex form, skolemization, soundness

## Part III: Limitations and Extensions of First-Order Logic

**Curry–Howard isomorphism**, intuitionistic logic, Kripke models, ~~second-order logic~~, **simply-typed  $\lambda$ -calculus**, (simply-typed) combinatory logic

# Outline

- Overview of this lecture
- **Natural Deduction**
- $\lambda$ -calculus
- Strong Normalization by Strong Computability
- Curry–Howard Isomorphism
- Exercises
- Further Reading

## Definition (Natural Deduction, Tree Variant)

- Assumption  $\Gamma, \varphi \vdash \varphi$
- Implication Introduction 
$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \phi \supset \psi}$$
- Implication Elimination 
$$\frac{\Gamma \vdash \phi \supset \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}$$

$\Gamma \vdash_{ND} \varphi$  if  $\Gamma \vdash \varphi$  is derivable

## Lemma

$$\Gamma \vdash_{pn} \varphi \iff \Gamma \vdash_{ND} \varphi$$

Here  $\vdash_{pn}$  refers to natural deduction as in Section 4.2 of Fitting (*line-based, not tree-based*) restricted to the inference rules for *implication* only: Modus Ponens (as for Hilbert Systems) and the inference rule given in Figure 4.1.

## Examples

for arbitrary formulas  $\phi, \psi, \chi$ : (we assume  $\supset$  is right-associative)

$$\frac{\frac{\phi, \psi \vdash \phi}{\bullet \phi \vdash \psi \supset \phi}}{\vdash \phi \supset \psi \supset \phi}$$

- For  $\Gamma = \{\phi \supset \psi \supset \chi, \phi \supset \psi, \phi\}$

$$\frac{\frac{\frac{\Gamma \vdash \phi \supset \psi \supset \chi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi \supset \chi} \quad \frac{\Gamma \vdash \phi \supset \psi \quad \Gamma \vdash \phi}{\Gamma \vdash \psi}}{\Gamma \vdash \chi}}{\frac{\phi \supset \psi \supset \chi, \phi \supset \psi \vdash \phi \supset \chi}{\phi \supset \psi \supset \chi \vdash (\phi \supset \psi) \supset \phi \supset \chi}}{\vdash (\phi \supset \psi \supset \chi) \supset (\phi \supset \psi) \supset \phi \supset \chi}$$



# Outline

- Overview of this lecture
- Natural Deduction
- $\lambda$ -calculus
- Strong Normalization by Strong Computability
- Curry–Howard Isomorphism
- Exercises
- Further Reading

## $\lambda$ -abstraction for function specification

In mathematics and programming there are various ways to specify functions. We illustrate several of them by means of the example of the **composition** function.

- **verbose**: composition is the function that takes any two functions  $f$  and  $g$  and yields a function that when applied to  $x$  yields  $f(g(x))$ , which is quite **verbose**
- **naming**:  $(f \cdot g)(x) = f(g(x))$ , which requires **naming** composition as  $(f \cdot g)$
- **naming**: in combinatory logic composition is **named** B (slide 21 of previous week), with reduction rule  $B f g x \rightarrow_w f (g x)$ .
- **anonymously** in Haskell:  $(\backslash f g x \rightarrow f (g x))$ , with type indeed  $(t1 \rightarrow t2) \rightarrow (t3 \rightarrow t1) \rightarrow t3 \rightarrow t2$  according to Haskell
- **anonymously** in  $\lambda$ -calculus:  $\lambda fgx.f (g x)$ . We will see that indeed it has **type**  $(\alpha \rightarrow \beta) \rightarrow (\gamma \rightarrow \alpha) \rightarrow (\gamma \rightarrow \beta)$ , or in other words that  $\lambda fgx.f (g x)$  is a Natural Deduction **proof** of the **proposition**  $(Y \supset Z) \supset (X \supset Y) \supset (X \supset Z)$

### Remark

Upshot:  $\lambda$ -terms **are** (anonymous) functions, which **are** proofs (of implications)

## Definition

set  $\mathcal{L}$  of ( $\lambda$ -)terms is built from

- **variables**  $x, y, z, \dots$
- **$\lambda$ -abstractions**  $\lambda x.M$  for variables  $x$  and  $\lambda$ -terms  $M$
- **application**  $(MN)$  for  $\lambda$ -terms  $M$  and  $N$

## Notational Convention

We assume application is **left associative** to reduce number of parentheses, and **combine  $\lambda$ -abstractions** to reduce number of  $\lambda$ s.

## Definition

**( $\beta$ -)reduction** is smallest relation  $\rightarrow_\beta$  on  $\lambda$ -terms such that

$$\frac{}{(\lambda x.M)N \rightarrow_\beta M[x:=N]} \quad \frac{M \rightarrow_\beta N}{\lambda x.M \rightarrow_\beta \lambda x.N} \quad \frac{M \rightarrow_\beta N}{MP \rightarrow_\beta NP} \quad \frac{M \rightarrow_\beta N}{PM \rightarrow_\beta PN}$$

for all  $\lambda$ -terms  $M, N$ , all variables  $x$ , and all  $\lambda$ -terms  $P$

## Definition

- $\lambda x.M$  **binds** occurrences of  $x$  in  $M$ . occurrence of  $x$  is **free** if not bound.
- capture avoiding** substitution

$$x[x:=N] = N$$

$$y[x:=N] = y$$

$$(\lambda x.M)[x:=N] = \lambda x.M$$

$$(\lambda y.M)[x:=N] = \lambda y.M[x:=N] \quad \text{if } y \text{ not free in } N$$

$$(\lambda y.M)[x:=N] = \lambda z.M[y:=z][x:=N] \quad \text{if some } y \text{ free in } N$$

$$(M_1 M_2)[x:=N] = M_1[x:=N] M_2[x:=N]$$

with  $z$  **fresh** (first variable not in  $x, y, M, N$ )

## Example

$(\lambda x.x)x$  **bound free**  $x$ .  $\lambda y.(\lambda xy.x)y \rightarrow_{\beta} \lambda yz.y$ , since is  
 $\lambda y.((\lambda x.(\lambda y.x))y) \rightarrow_{\beta} \lambda y.(\lambda z.y)$  per notational convention, which follows from  
 $(\lambda x.(\lambda y.x))y \rightarrow_{\beta} (\lambda y.x)[x:=y] = \lambda z.y$ , which follows by **3rd clause** for  
 $\lambda$ -abstraction since  $y$  is free in  $y$ .

## Definitions

- $\rightarrow_{\beta}^*$  is transitive and reflexive closure of  $\rightarrow_{\beta}$
- **normal form** is  $\lambda$ -term  $M$  such that  $M \rightarrow_{\beta} N$  for no  $\lambda$ -term  $N$
- $=_{\beta}$  is transitive, reflexive, and symmetric closure of  $\rightarrow_{\beta}$
- $\lambda$ -term  $M$  is **normalizing** if  $M \rightarrow_{\beta}^* N$  for some normal form  $N$
- **infinite reduction** is sequence  $(M_i)_{i \geq 0}$  such that  $M_i \rightarrow_{\beta} M_{i+1}$  for all  $i \geq 0$
- $\lambda$ -term  $M$  is **strongly normalizing** if there are no infinite reductions from  $M$

## Example

$\lambda$ -term  $\Omega = (\lambda x.x x)(\lambda x.x x)$  not strongly normalizing:  $\Omega \rightarrow_{\beta} \Omega$

## Theorem

- (**Confluence**) if  $M \rightarrow_{\beta}^* N_1$ ,  $M \rightarrow_{\beta}^* N_2$  then  $N_1 \rightarrow_{\beta}^* N_3$ ,  $N_2 \rightarrow_{\beta}^* N_3$  for some  $N_3$
- (**Consistency**) there are  $M, N$  such that  $M \not\equiv_{\beta} N$ , e.g. distinct normal forms.

## Definitions

- **simple type** is implicational propositional formula
- **environment** is finite set of pairs  $\Gamma = \{x_1 : \tau_1, \dots, x_n : \tau_n\}$  with pairwise distinct variables  $x_1, \dots, x_n$  and simple types  $\tau_1, \dots, \tau_n$
- $\text{dom}(\Gamma) = \{x \mid (x : \tau) \in \Gamma\}$  and  $\text{ran}(\Gamma) = \{\tau \mid (x : \tau) \in \Gamma\}$
- **judgement**  $\Gamma \vdash M : \tau$  ( $\lambda$ -term  $M$  has type  $\tau$  in environment  $\Gamma$ ) is defined by **type assignment rules**
  - variable  $\Gamma, x : \tau \vdash x : \tau$
  - $\lambda$ -abstraction 
$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x. M : \sigma \rightarrow \tau}$$
  - application 
$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (M N) : \tau}$$

## Remark

for convenience we assume distinct  $\lambda$ s bind distinct variables. for instance,  $\vdash \lambda yx. x : \sigma \rightarrow (\tau \rightarrow \tau)$  instead of  $\vdash \lambda xx. x : \sigma \rightarrow (\tau \rightarrow \tau)$  (see exercises).

## Example

$\vdash \lambda f g x. f (g x) : (\alpha \rightarrow \beta) \rightarrow (\gamma \rightarrow \alpha) \rightarrow (\gamma \rightarrow \beta)$  for all simple types  $\alpha, \beta, \gamma$

$$\begin{array}{c}
 \Gamma \vdash g : \gamma \rightarrow \alpha \quad \Gamma \vdash x : \gamma \\
 \hline
 \Gamma \vdash g x : \alpha \\
 \Gamma \vdash f : \alpha \rightarrow \beta \\
 \hline
 \Gamma \vdash f (g x) : \beta \\
 \hline
 f : \alpha \rightarrow \beta, g : \gamma \rightarrow \alpha \vdash \lambda x. f (g x) : \gamma \rightarrow \beta \\
 \hline
 f : \alpha \rightarrow \beta \vdash \lambda g x. f (g x) : (\gamma \rightarrow \alpha) \rightarrow \gamma \rightarrow \beta \\
 \hline
 \vdash \lambda f g x. f (g x) : (\alpha \rightarrow \beta) \rightarrow (\gamma \rightarrow \alpha) \rightarrow \gamma \rightarrow \beta
 \end{array}$$

with  $\Gamma = \{f : \alpha \rightarrow \beta, g : \gamma \rightarrow \alpha, x : \gamma\}$

## Definitions

- set  $FV(M)$  of (free) variables of  $\lambda$ -term  $M$ :

$$FV(M) = \begin{cases} \{x\} & \text{if } M = x \\ FV(M_1) - \{x\} & \text{if } M = \lambda x.M_1 \\ FV(M_1) \cup FV(M_2) & \text{if } M = M_1 M_2 \end{cases}$$

- $\lambda$ -term  $M$  is **typable** if  $\Gamma \vdash M : \tau$  for some environment  $\Gamma$  with  $\text{dom}(\Gamma) = FV(M)$  and simple type  $\tau$

## Lemma (Subject Reduction)

*if  $\Gamma \vdash M : \tau$  and  $M \rightarrow_{\beta}^* N$  then  $\Gamma \vdash N : \tau$*

## Theorem (Strong Normalization)

*typable  $\lambda$ -terms are strongly normalizing*



## Normalization by substitution

- A  $\lambda$ -term  $(\lambda x.M)N$  corresponds to having a proof  $M$  of some proposition  $Y$  **under the assumption** named  $x$  that  $X$  holds, **and** a proof  $N$  that the assumption  $X$  in fact holds. Such a proof can be **normalized** by directly proving  $Y$  using  $X$  everywhere where the assumption that  $X$  holds, was used
- At term level, this is brought about by  $M[x:=N]$ , that is, the proof obtained from  $M$  by **substituting** the proof  $N$  everywhere for  $x$  in the proof  $M$ . Strong normalization expresses that this process, of repeatedly doing  $\rightarrow_{\beta}$ -steps, must terminate on **typable**  $\lambda$ -terms. To prepare for that, we will first show strong normalization of typable CL-terms wrt.  $\rightarrow_w$  (left unproven last week), as that is analogous but easier

## Remark

Before, we have given **some strategy** to successively eliminate cuts from tableau proofs, this is called **weak** normalization (WN). For  $\rightarrow_{\beta}$  we prove something stronger, appropriately called **strong** normalization (SN), namely that doing **arbitrary**  $\rightarrow_{\beta}$  steps (**every strategy**) must terminate, resulting in a  $\lambda$ -term without subterms of shape  $(\lambda x.M)N$ .

# Outline

- Overview of this lecture
- Natural Deduction
- $\lambda$ -calculus
- **Strong Normalization by Strong Computability**
  - Strong normalisation of  $\rightarrow_w$
- Curry–Howard Isomorphism
- Exercises
- Further Reading

## Theorem

$\rightarrow_w$  is strongly normalizing on the typable CL-terms in  $\mathcal{C}$ .

## Proof

How to prove?

- Untyped CL-terms are not strongly normalizing, so **types** need to be exploited
- Idea: define **strong computability** (SC) by induction on types such that SC implies SN (and both are equivalent for **base** types).
- A typed term  $M$  is **strongly computable** if for all (possibly empty) vectors  $\vec{N} = N_1, N_2, \dots$  of (appropriately typed) SC terms, the term  $M\vec{N} = MN_1N_2\dots$  (parenthesized to the left) is SN.
- Cf. this definition to how **functions** (terms of **function** type) are usually compared **via elements** (terms of **base** type) in math:  $f \geq g$  if  $\forall x \in \mathbb{R}, f(x) \geq g(x)$ , so-called **pointwise** comparison. In logic, relations defined in this way, inductively **lifting** a relation on base types to arbitrary types, are called **logical** relations. SC is the logical relation obtained by **lifting** SN.

We show all simply typed CL-terms to be SC by induction on terms

## Theorem

$\rightarrow_\beta$  is strongly normalizing on the typable  $\lambda$ -terms in  $\mathcal{L}$ .

## Proof

We claim for all typable  $\lambda$ -terms  $M$  and for all SC substitutions  $\sigma$  the  $\lambda$ -term  $M\sigma$  is SC. Here, a substitution is SC if it maps variables to SC  $\lambda$ -terms (of the same type), From the claim the theorem follows by taking the identity substitution, mapping  $x$  to  $x$ , noting that it is SC and yields  $M$  when applied to  $M$ . The claim is proven by induction on  $M$ . We only show the  $\lambda$ -abstraction case since the other cases (variable and application) are analogous to those for  $\rightarrow_w$ :

- For a  $\lambda$ -abstraction  $\lambda x.M$  we must show for all SC vectors  $\vec{N} = N_1, N_2, N_3, \dots$  and SC substitutions  $\sigma$ , the  $\lambda$ -term  $(\lambda x.M)\sigma\vec{N}$  is SN. Since by the IH  $M\sigma$  is SC, if there were an infinite reduction it would have shape  $(\lambda x.M)\sigma N_1 N_2 \dots \rightarrow_\beta^* (\lambda x.M')N'_1 N'_2 \dots \rightarrow_\beta M'[x:=N'_1]N'_2 \dots \rightarrow_\beta \dots$ . But then  $(\lambda x.M)\sigma N_1 N_2 \dots \rightarrow_\beta M\sigma[x:=N_1]N_2 \dots \rightarrow_\beta^* M'[x:=N'_1]N'_2 \dots \rightarrow_\beta \dots$  would also be an infinite reduction. But  $M\sigma[x:=N_1]N_2 \dots$  is SN as application of the  $\lambda$ -term  $M\sigma[x:=N_1]$ , SC by the IH, to the SC vector  $N_2, \dots$

## Decision Problems

- **type checking**

instance:  $\lambda$ -term  $M$ , environment  $\Gamma$ , simple type  $\tau$

question:  $\Gamma \vdash M : \tau ?$

- **type inference**

instance:  $\lambda$ -term  $M$

question:  $\Gamma \vdash M : \tau$  for some environment  $\Gamma$  and simple type  $\tau$ ?

- **type inhabitation**

instance: type  $\tau$ , environment  $\Gamma$

question:  $\Gamma \vdash M : \tau$  for some  $\lambda$ -term  $M$ ?

## Theorem

*type checking, inference, and inhabitation are decidable problems*

## Theorem

*type inhabitation is decidable*

## Proof

Idea: **normalization** allows to limit the search for ND proofs to **finitely** many possibilities. Formally, we claim the **subformula** property holds: if  $\Gamma \vdash \phi$ , then there is an ND proof (namely an ND proof in normal form) in which only **subformulas** of formulas in  $\Gamma$  and  $\phi$  occur. From the claim the theorem follows easily.

The subformula property is proven by induction on **normalized** ND proofs. The interesting case is Implication Elimination (Modus Ponens).

- Suppose  $\Gamma \vdash (MN) : \phi$  is inferred from  $\Gamma \vdash M : \psi \rightarrow \phi$  and  $\Gamma \vdash N : \psi$ . It suffices to show that  $\psi \rightarrow \phi$  is a subformula of  $\Gamma$ . Because the ND proof is **normalized**,  $M$  must have shape  $xM_1 \dots M_n$  for some variable  $x$  and  $\lambda$ -terms  $M_1, \dots, M_n$ ; otherwise a  $\rightarrow_\beta$ -step would be possible. Thus we must have  $\Gamma \vdash x : \psi_1 \rightarrow \dots \rightarrow \psi_n \rightarrow \psi \rightarrow \phi$  with  $\Gamma \vdash M_i : \psi_i$ . Hence  $\psi_1 \rightarrow \dots \rightarrow \psi_n \rightarrow \psi \rightarrow \phi \in \Gamma$ , from which we conclude that  $\psi \rightarrow \phi$  is indeed a subformula of some formula in  $\Gamma$  (namely of the assumption  $x$ )

# Outline

- Overview of this lecture
- Natural Deduction
- $\lambda$ -calculus
- Strong Normalization by Strong Computability
- **Curry–Howard Isomorphism**
- Exercises
- Further Reading

## type assignment

$$\Gamma, x : \tau \vdash x : \tau$$

$$\frac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash (\lambda x.M) : \sigma \rightarrow \tau}$$

$$\frac{\Gamma \vdash M : \sigma \rightarrow \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash (MN) : \tau}$$

## Natural Deduction

$$\Gamma, \varphi \vdash \varphi$$

$$\frac{\Gamma, \varphi \vdash \psi}{\Gamma \vdash \varphi \supset \psi}$$

$$\frac{\Gamma \vdash \varphi \supset \psi \quad \Gamma \vdash \varphi}{\Gamma \vdash \psi}$$

## Theorem (Curry–Howard)

identifying  $\rightarrow$  and  $\supset$  as before:

- 1 if  $\Gamma \vdash M : \tau$  then  $\text{ran}(\Gamma) \vdash_{ND} \tau$
- 2 if  $\Gamma \vdash_{ND} \varphi$  then  $\Delta \vdash M : \varphi$  for some  $M$  and  $\Delta$  with  $\text{ran}(\Delta) = \Gamma$



## Theorem (Curry–Howard)

1 *if  $\Gamma \vdash M : \tau$  then  $\text{ran}(\Gamma) \vdash_{ND} \tau$*

### Proof

By induction on derivation of judgement  $\Gamma \vdash M : \tau$  as for typed combinatory logic. We only give the new,  $\lambda$ -abstraction, case:

- $M = \lambda x.N$  and  $\tau = \sigma \rightarrow \rho$   
 $\text{ran}(\Gamma), \sigma \vdash_{ND} \rho$  by induction hypothesis  
 $\text{ran}(\Gamma) \vdash_{ND} \sigma \rightarrow \rho$  Implication Introduction

## Theorem (Curry–Howard)

2 *if  $\Gamma \vdash_{ND} \varphi$  then  $\Delta \vdash M : \varphi$  for some  $M$  and  $\Delta$  with  $\text{ran}(\Delta) = \Gamma$*

### Proof

Induction on  $\Gamma \vdash_{ND} \varphi$  as for the Hilbert System H. No interesting new cases.

## Definition

Translations  $(-)_{CL} : \mathcal{L} \rightarrow \mathcal{C}$  and  $(-)_{\lambda} : \mathcal{C} \rightarrow \mathcal{L}$  are defined by mapping variables and applications to 'themselves' (**homomorphically**) and defining

$$\begin{aligned}(\lambda x.M)_{CL} &= [x](M)_{CL} \\ (K)_{\lambda} &= \lambda xy.x \\ (S)_{\lambda} &= \lambda xyz.xz(yz)\end{aligned}$$

where the first clause uses the bracket abstraction algorithm

## Examples

- $(\lambda xy.x)_{CL} = [x](\lambda y.x)_{CL} = [x][y](x)_{CL} = [x][y]x = [x](Kx) = K$ , using the optimization of the exercises, for the last equality
- $(SKK)_{\lambda} = (\lambda xyz.xz(yz))(\lambda xy.x)(\lambda xy.x)$ . Recall  $SKK$  is Hilbert System proof of  $X \supset X$ . Note  $(\lambda xyz.xz(yz))(\lambda xy.x)(\lambda xy.x) \rightarrow_{\beta}^* (\lambda yz.z)(\lambda xy.x) \rightarrow_{\beta}^* \lambda z.z$ , with  $\lambda z.z$  the **identity** function, which obviously has type  $\tau \rightarrow \tau$ , which would justify defining  $(I)_{\lambda} = \lambda x.x$

## Theorem

The translations  $(-)_{CL}$  and  $(-)_{\lambda}$  on terms preserve types, showing

$$\Gamma \vdash_H \varphi \iff \Gamma \vdash_{ND} \varphi$$

## Proof

By induction on derivations, using for  $(-)_{CL}$  the lemma on slide 33 of last week, and for  $(-)_{\lambda}$  that the translations  $\lambda xy.x$  and  $\lambda xyz.xz(yz)$  have the same types (in the typed  $\lambda$ -calculus) as  $K$  and  $S$  have (in typed combinatory logic).

## Remark

This result allows to ‘switch freely’ between ND and H, e.g. it follows that ND is sound and complete for Kripke semantics. However, beware the translation  $(-)_{\lambda}$  is **linear**, but ‘the’ reverse translation  $(-)_{CL}$  is **exponential**. That is, ND proofs blow up in size when translated to H proofs, but not vice versa. On the other hand, H proofs do not use  $\lambda$ -abstraction, a notion considered complex. Indeed combinatory logic was invented to get rid of  $\lambda$ -abstractions, more precisely of **bound** variables.

# Outline

- Overview of this lecture
- Natural Deduction
- $\lambda$ -calculus
- Strong Normalization by Strong Computability
- Curry–Howard Isomorphism
- Exercises
- Further Reading

- 1 Give three  $\lambda$ -terms  $M$  such that  $\vdash M : \tau \rightarrow \tau \rightarrow \tau$ , for  $\tau$  an arbitrary type. We count only terms up to renaming of variables; so  $\lambda xy.xy$  and  $\lambda yx.yx$  are considered the same (as they can be obtained by renaming  $x$  into  $y$  and vice versa), but different from  $\lambda xy.yx$ . Is this a reasonable way to count terms inhabiting types, i.e. to count proofs of propositions, vis-à-vis the Curry–Howard isomorphism?
- 2 Bonus: Show that, among the three  $\lambda$ -terms in the previous exercise, at least two must be  $=_{\beta}$ -related.
- 3 Reduce the  $\lambda$ -term  $M = (\lambda x.xx)(\lambda yz.yz)$  to normal form  $N$  (it requires 3  $\rightarrow_{\beta}$ -steps; give each of them).
- 4 Show that there is no  $\lambda$ -term  $M$  such that  $\vdash M : (\tau \rightarrow \tau) \rightarrow \tau$ , for  $\tau$  an arbitrary type.  
Hint: normalization or Kripke models (1 cross each)
- 5 Compute the translation  $(SS(KI))_{\lambda}$ , i.e. the translation of  $W$  on slide 21 of the previous week, and reduce the resulting  $\lambda$ -term to normal form.

- 6 Bonus (1 cross per item): The Haskell expression  $(\backslash x \rightarrow \backslash x \rightarrow x)$  corresponds to the  $\lambda$ -term  $\lambda x.\lambda x.x$ . Asking Haskell the type of the former yields  $p1 \rightarrow p2 \rightarrow p2$ .
- Explain why using the type assignment rules for  $\lambda$ -calculus (slide 13), we can infer  $\vdash \lambda y.\lambda x.x : \sigma \rightarrow (\tau \rightarrow \tau)$ , but not  $\vdash \lambda x.\lambda x.x : \sigma \rightarrow (\tau \rightarrow \tau)$ , assuming  $\sigma \neq \tau$  (cf. the remark there).  
Could one overcome this limitation? That is, could our type assignment system be adapted such that we **do** have  $\vdash \lambda x.\lambda x.x : \sigma \rightarrow (\tau \rightarrow \tau)$ ?
  - Just as per our conventions  $\lambda yx.x$  is shorthand for  $\lambda y.\lambda x.x$ , in Haskell  $(\backslash y \ x \rightarrow x)$  is shorthand for  $(\backslash y \rightarrow \backslash x \rightarrow x)$ . Do  $(\backslash x \ x \rightarrow x)$  and  $(\backslash x \rightarrow \backslash x \rightarrow x)$  have the same type in Haskell? Can you explain why (not)?
- 7 SC is defined (slide 17) in terms of itself: **SC** of  $M$  is defined in terms of **SC** of  $\vec{N}$ . Argue that SC is still well-defined, i.e. that it is a proper inductive definition for simply typed terms (either CL-terms or  $\lambda$ -terms).
- 8 Work out the details of the first item, the variable case, (slide 17) of the proof that all simply typed CL terms are SC. In particular, show that SC entails SN and that every typed variable is SC.

- 9 Complete the details of the proof (slide 20) that inhabitation is decidable. More precisely, give both the details of the proof of the subformula property, and of that the subformula property entails decidability of inhabitation.
- 10 Bonus (worth 4 crosses): implement an inhabitation checker for implicational intuitionistic logic, based on the previous exercise.
- 11 Bonus: Suppose combinatory logic would have another constant  $J$  having some type and reduction rule, for all terms  $M_i$ ,  $JM_1 \dots M_n \rightarrow_w E$  where  $E$  is an expression **only constructed from applications and the  $M_i$**  and both sides have the same base type, e.g.  $\Gamma \vdash J : (\sigma \rightarrow \sigma \rightarrow \tau) \rightarrow \sigma \rightarrow \tau$  with rule  $JM_1M_2 \rightarrow_w M_1M_2M_2$  (note the conditions hold for  $K$  and  $S$ ). Is  $\rightarrow_w$  still strongly normalizing? If it is not, give a specific  $J$  and rule for it allowing an infinite reduction. If it is, give a proof.
- 12 Bonus (worth 3 crosses): **directly** show weak normalization of  $\rightarrow_\beta$  on typable  $\lambda$ -terms, **without** showing (a property that entails) strong normalization . Hint: an idea analogous to that for the cut elimination procedure in the book works, judiciously choosing a  $\rightarrow_\beta$  step among the possible ones and showing that that decreases some measure.

- 13 Bonus (worth 4 crosses): Prove or disprove that the tableau cut-elimination procedure in Fitting is **strongly** normalizing. Proceed as follows: the proof of Lemma 8.9.3 establishes **weak** normalization by transforming **minimal** cuts. (Try to) verify whether
- correctness of the transformations depends on minimality,
  - the induction (see p. 232) used in the proof works for non-minimal cuts,
  - **strong** normalization, when eliminating arbitrary cuts, holds.



# Outline

- Overview of this lecture
- Natural Deduction
- $\lambda$ -calculus
- Strong Normalization by Strong Computability
- Curry–Howard Isomorphism
- Exercises
- **Further Reading**

## Fitting

- Section 4.1 (from Curry–Howard-perspective)
- Section 4.2 (idem)

## Additional Literature

- Philip Wadler, [Propositions as Types](#), Communications of the ACM 58(12), pp. 75–84, 2015
- Morten Heine Sørensen and Pawel Urzyczyn, [Lectures on the Curry–Howard Isomorphism](#), Studies in Logic and the Foundations of Mathematics, volume 149, Elsevier, 2006 (cached PDF of preliminary version on citeseer)