

Logic in Computer Science V

Lesson 10, λ -calculus and intuitionistic logic

Recommended reading

- ▶ Zlatuška, *Lambda-kalkul*
- ▶ Barendregt, *Chapter D.7. in Handbook of Logic*
- ▶ Constable, *Chapter X. Types in Logic, Mathematics and Programming, in Handbook of Proof Theory*
- ▶ Sørensen and Urzyczyn, *Lectures on the Curry-Howard Isomorphism*

Lesson 10, λ -calculus and intuitionistic logic

λ -calculus is an important calculus that can be used (mainly) for

- ▶ formalizing computations
- ▶ programming languages
- ▶ formalizing logic

It is connected with intuitionistic logic. Extensions that are connected with classical logic are also known, but they are not so natural.

Lesson 10, λ -calculus and intuitionistic logic

λ -calculus is an important calculus that can be used (mainly) for

- ▶ formalizing computations
- ▶ programming languages
- ▶ formalizing logic

It is connected with intuitionistic logic. Extensions that are connected with classical logic are also known, but they are not so natural.

We will see that the formalizations based on the λ -calculus are similar to those we have seen.

main types of λ calculus

1. type-free λ -calculus

- ▶ combinatory algebra, a.k.a. *combinatory logic*
- ▶ term rewriting system

2. typed λ -calculus, a.k.a. *type theory*;

main types of λ calculus

1. type-free λ -calculus
 - ▶ combinatory algebra, a.k.a. *combinatory logic*
 - ▶ term rewriting system
2. typed λ -calculus, a.k.a. *type theory*; this is connected with intuitionistic propositional logic.

main types of λ calculus

1. type-free λ -calculus

- ▶ combinatory algebra, a.k.a. *combinatory logic*
- ▶ term rewriting system

2. typed λ -calculus, a.k.a. *type theory*; this is connected with intuitionistic propositional logic.

For a connection with first order logic, one needs *dependent types*.

combinatory algebra

Idea: every object is a function and an argument at the same time.

combinatory algebra

Idea: every object is a function and an argument at the same time.

- ▶ one binary operation, *application*, xy (“ x applied to y ”)
we will use *association to the left*

combinatory algebra

Idea: every object is a function and an argument at the same time.

▶ one binary operation, *application*, xy (“ x applied to y ”)
we will use *association to the left*

▶ axioms:

1. *combinatory completeness*: for every term A ,

$$\exists f \forall x_1 \dots \forall x_n (fx_1 \dots x_n = A),$$

2. *extensionality*:

$$\forall x (fx = gx) \rightarrow f = g.$$

3. *nontriviality*:

$$\exists x, y (x \neq y).$$

To get the combinatorial completeness one can use

1. either λ -terms, $\lambda x.A$ ¹ with axioms

$$(\lambda x.A)B = A[x/B],$$

called β -conversion,²

2. or constants K, S , called *combinators*, and axioms
 - ▶ $Kxy = x$,
 - ▶ $Sxyz = xz(yz)$.

¹Applying λx to a term is called λ -abstraction; x is not free in $\lambda x.A$.

²in less precise, but more intuitive notation: $(\lambda x.A[x])B = A[B]$

To get the combinatorial completeness one can use

1. either λ -terms, $\lambda x.A$ ¹ with axioms

$$(\lambda x.A)B = A[x/B],$$

called β -conversion,²

2. or constants K, S , called *combinators*, and axioms
 - ▶ $Kxy = x$,
 - ▶ $Sxyz = xz(yz)$.

Note: Two special instances suffice for combinatorial completeness!

¹Applying λx to a term is called λ -abstraction; x is not free in $\lambda x.A$.

²in less precise, but more intuitive notation: $(\lambda x.A[x])B = A[B]$

To get the combinatorial completeness one can use

1. either λ -terms, $\lambda x.A$ ¹ with axioms

$$(\lambda x.A)B = A[x/B],$$

called β -conversion,²

2. or constants K, S , called *combinators*, and axioms
 - ▶ $Kxy = x$,
 - ▶ $Sxyz = xz(yz)$.

Note: Two special instances suffice for combinatorial completeness!

Example

- ▶ $K = \lambda x \lambda y.x$
- ▶ $S = \lambda x \lambda y \lambda z.xz(yz)$

¹Applying λx to a term is called λ -abstraction; x is not free in $\lambda x.A$.

²in less precise, but more intuitive notation: $(\lambda x.A[x])B = A[B]$

Proof.

ad 1. by iterating $(\lambda x.A)y = A[x/y]$ we get

$$(\lambda x_1 \dots \lambda x_n.A)y_1 \dots y_n = A[x_1/y_1, \dots, x_n/y_n].$$

Recall that we needed an f such that

$$fy_1 \dots y_n = A[x_1/y_1, \dots, x_n/y_n].$$



Proof.

ad 2. we construct the λ -terms from the combinators K, S .

Proof.

ad 2. we construct the λ -terms from the combinators K, S .

- ▶ define the combinator $I := SKK$ and show $Ix = x$
(Exercise!)

Proof.

ad 2. we construct the λ -terms from the combinators K, S .

▶ define the combinator $I := SKK$ and show $Ix = x$
(Exercise!)

▶ prove combinatorial completeness by induction

▶ base cases:

$$\lambda x.x \mapsto I,$$

$$\lambda x.y \mapsto Ky.$$

▶ induction step: $\lambda x.AB \mapsto S(\lambda x.A)(\lambda x.B)$; then

$$(S(\lambda x.A)(\lambda x.B))z =$$

$$((\lambda x.A)z)(\lambda x.B)z = \quad (\text{by definition of } S)$$

$$A[x/z]B[x/z] = AB[x/z] \quad (\text{by induction assumption})$$



Fixed Point Theorem

Theorem

1. For every λ -term A there exists a λ -term B such that

$$B = AB.$$

Fixed Point Theorem

Theorem

1. For every λ -term A there exists a λ -term B such that

$$B = AB.$$

2. Moreover, there exists a λ -term F that produces fixed-points for every term A

$$FA = A(FA)$$

Fixed Point Theorem

Theorem

1. For every λ -term A there exists a λ -term B such that

$$B = AB.$$

2. Moreover, there exists a λ -term F that produces fixed-points for every term A

$$FA = A(FA)$$

Proof.

1. Define $C := \lambda x.A(xx)$ and $B := CC$. Then

Fixed Point Theorem

Theorem

1. For every λ -term A there exists a λ -term B such that

$$B = AB.$$

2. Moreover, there exists a λ -term F that produces fixed-points for every term A

$$FA = A(FA)$$

Proof.

1. Define $C := \lambda x.A(xx)$ and $B := CC$. Then

$$B = CC = (\lambda x.A(xx))C = A(CC) = AB.$$

2. (Exercise)



Fixed Point Theorem

Theorem

1. For every λ -term A there exists a λ -term B such that

$$B = AB.$$

2. Moreover, there exists a λ -term F that produces fixed-points for every term A

$$FA = A(FA)$$

Proof.

1. Define $C := \lambda x.A(xx)$ and $B := CC$. Then

$$B = CC = (\lambda x.A(xx))C = A(CC) = AB.$$

2. (Exercise)



Intuition: $C \leftrightarrow$ “ x written twice has property A ”

Exercise

- ▶ *Prove 2.*
- ▶ *Write the fixed-point using combinators I , K , S .*

term rewriting

Often we can simplify λ -terms by rewriting:

³terminology: “conversion” for $=$, “reduction” for \rightarrow

⁴we will not use η -reduction in the sequel

term rewriting

Often we can simplify λ -terms by rewriting:

- ▶ $(\lambda x.A)B \rightarrow A[x/B]$ (β -reduction)³
- ▶ $\lambda x.Ax \rightarrow A$ (η -reduction) if $x \notin \text{Var}(A)$,⁴

³terminology: “conversion” for $=$, “reduction” for \rightarrow

⁴we will not use η -reduction in the sequel

term rewriting

Often we can simplify λ -terms by rewriting:

- ▶ $(\lambda x.A)B \rightarrow A[x/B]$ (β -reduction)³
- ▶ $\lambda x.Ax \rightarrow A$ (η -reduction) if $x \notin \text{Var}(A)$,⁴

but not always.

Example

$\Omega := (\lambda x.xx)(\lambda x.xx)$ *remains the same after β -reduction.*

³terminology: “conversion” for $=$, “reduction” for \rightarrow

⁴we will not use η -reduction in the sequel

term rewriting

Often we can simplify λ -terms by rewriting:

- ▶ $(\lambda x.A)B \rightarrow A[x/B]$ (β -reduction)³
- ▶ $\lambda x.Ax \rightarrow A$ (η -reduction) if $x \notin \text{Var}(A)$,⁴

but not always.

Example

$\Omega := (\lambda x.xx)(\lambda x.xx)$ remains the same after β -reduction.

β -reduction can *increase* the size.

Example

Suppose B is a long term, then

- ▶ $(\lambda x.xx)B \rightarrow BB$

produces almost a twice long term.

³terminology: “conversion” for $=$, “reduction” for \rightarrow

⁴we will not use η -reduction in the sequel

Definition

1. A λ -term is in a **normal form** if it does not contain a subterm $(\lambda x.A)B$ (called *redex*).

Definition

1. A λ -term is in a **normal form** if it does not contain a subterm $(\lambda x.A)B$ (called *redex*).
2. **Normalization** is a sequence of β -reductions that produces a term in the normal form.

Definition

1. A λ -term is in a **normal form** if it does not contain a subterm $(\lambda x.A)B$ (called *redex*).
2. **Normalization** is a sequence of β -reductions that produces a term in the normal form.

We will see that

- ▶ $\text{redex} \leftrightarrow \text{cut}$
- ▶ $\text{normalization} \leftrightarrow \text{cut-elimination}$

Definition

1. A λ -term is in a **normal form** if it does not contain a subterm $(\lambda x.A)B$ (called *redex*).
2. **Normalization** is a sequence of β -reductions that produces a term in the normal form.

We will see that

- ▶ $\text{redex} \leftrightarrow \text{cut}$
- ▶ $\text{normalization} \leftrightarrow \text{cut-elimination}$

Also very important (but we will not deal with it)

- ▶ $\text{normalization} \leftrightarrow \text{computation}$

Theorem

If a λ -term can be reduced to a normal form, then the normal form is unique.

Theorem

If a λ -term can be reduced to a normal form, then the normal form is unique.

Proof.

is based on the **Church-Rosser property**:

- ▶ if $A \rightarrow B_1$ and $A \rightarrow B_2$, then there exists C such that $B_1 \rightarrow C$ and $B_2 \rightarrow C$.



typed λ -calculus

Idea: *one can only apply x to y if they have appropriate types.*

typed λ -calculus

Idea: *one can only apply x to y if they have appropriate types.*

Simple types:

- ▶ type variables u, v, \dots ,
- ▶ if σ and τ are types, $\sigma \rightarrow \tau$ is a type.

Notation:

- ▶ “ A has type σ ” is abbreviated by $A : \sigma$ (sometimes also A^σ).

typed λ -calculus

Idea: *one can only apply x to y if they have appropriate types.*

Simple types:

- ▶ type variables u, v, \dots ,
- ▶ if σ and τ are types, $\sigma \rightarrow \tau$ is a type.

Notation:

- ▶ “ A has type σ ” is abbreviated by $A : \sigma$ (sometimes also A^σ).

Rule:

- ▶ AB is well-formed if $A : \sigma \rightarrow \tau$ and $B : \sigma$,
- ▶ then $AB : \tau$.

Given an **untyped λ -term** it **may not be** possible to assign types to variables and combinators so that it is a **well-formed typed term**.

Given an **untyped λ -term** it **may not be** possible to assign types to variables and combinators so that it is a **well-formed typed term**.

If it is possible, we say that the **term is typable**.

Given an **untyped λ -term** it **may not be** possible to assign types to variables and combinators so that it is a **well-formed typed term**.

If it is possible, we say that the **term is typable**.

According to *“typing a λ Church”*, one should always **declare the types of variables and combinators** to prevent untypability.

examples

1. For every triple of types ρ, σ, τ , we have combinators

▶ $I_\rho = \lambda x.x : \rho \rightarrow \rho$

where $x : \rho$,

▶ $K_{\rho,\sigma} = \lambda x \lambda y.x : \rho \rightarrow (\sigma \rightarrow \rho)$

where $x : \rho, y : \sigma$,

▶ $S_{\rho,\sigma,\tau} = \lambda x \lambda y \lambda z.xz(yz) : (\rho \rightarrow (\sigma \rightarrow \tau)) \rightarrow ((\rho \rightarrow \sigma) \rightarrow (\rho \rightarrow \tau))$

where $x : \rho \rightarrow (\sigma \rightarrow \tau), y : \rho \rightarrow \sigma, z : \rho$.

examples

1. For every triple of types ρ, σ, τ , we have combinators

▶ $I_\rho = \lambda x.x : \rho \rightarrow \rho$

where $x : \rho$,

▶ $K_{\rho,\sigma} = \lambda x \lambda y.x : \rho \rightarrow (\sigma \rightarrow \rho)$

where $x : \rho, y : \sigma$,

▶ $S_{\rho,\sigma,\tau} = \lambda x \lambda y \lambda z.xz(yz) : (\rho \rightarrow (\sigma \rightarrow \tau)) \rightarrow ((\rho \rightarrow \sigma) \rightarrow (\rho \rightarrow \tau))$

where $x : \rho \rightarrow (\sigma \rightarrow \tau), y : \rho \rightarrow \sigma, z : \rho$.

2. $II := (\lambda x.x)(\lambda y.y)$ is typable:

▶ let the first $I : (\tau \rightarrow \tau) \rightarrow (\tau \rightarrow \tau)$

▶ the second $I : \tau \rightarrow \tau$

▶ then $II : \tau \rightarrow \tau$

examples

1. For every triple of types ρ, σ, τ , we have combinators
 - ▶ $I_\rho = \lambda x.x : \rho \rightarrow \rho$
where $x : \rho$,
 - ▶ $K_{\rho,\sigma} = \lambda x \lambda y.x : \rho \rightarrow (\sigma \rightarrow \rho)$
where $x : \rho, y : \sigma$,
 - ▶ $S_{\rho,\sigma,\tau} = \lambda x \lambda y \lambda z.xz(yz) : (\rho \rightarrow (\sigma \rightarrow \tau)) \rightarrow ((\rho \rightarrow \sigma) \rightarrow (\rho \rightarrow \tau))$
where $x : \rho \rightarrow (\sigma \rightarrow \tau), y : \rho \rightarrow \sigma, z : \rho$.
2. $II := (\lambda x.x)(\lambda y.y)$ is typable:
 - ▶ let the first $I : (\tau \rightarrow \tau) \rightarrow (\tau \rightarrow \tau)$
 - ▶ the second $I : \tau \rightarrow \tau$
 - ▶ then $II : \tau \rightarrow \tau$
3. $\Omega := (\lambda x.xx)(\lambda x.xx)$ is not typable, for it remains the same after β -reduction.

Algorithms for typing λ -terms are based on unification (of types).

- ▶ $\lambda x.x : \rho \rightarrow \rho$
- ▶ $\lambda x\lambda y.x : \rho \rightarrow (\sigma \rightarrow \rho)$
- ▶ $\lambda x\lambda y\lambda z.xz(yz) : (\rho \rightarrow (\sigma \rightarrow \tau)) \rightarrow ((\rho \rightarrow \sigma) \rightarrow (\rho \rightarrow \tau))$

- ▶ $\lambda x.x : \rho \rightarrow \rho$
- ▶ $\lambda x\lambda y.x : \rho \rightarrow (\sigma \rightarrow \rho)$
- ▶ $\lambda x\lambda y\lambda z.xz(yz) : (\rho \rightarrow (\sigma \rightarrow \tau)) \rightarrow ((\rho \rightarrow \sigma) \rightarrow (\rho \rightarrow \tau))$

Note: The types are [propositional tautologies](#).

- ▶ $\lambda x.x : \rho \rightarrow \rho$
- ▶ $\lambda x\lambda y.x : \rho \rightarrow (\sigma \rightarrow \rho)$
- ▶ $\lambda x\lambda y\lambda z.xz(yz) : (\rho \rightarrow (\sigma \rightarrow \tau)) \rightarrow ((\rho \rightarrow \sigma) \rightarrow (\rho \rightarrow \tau))$

Note: The types are **propositional tautologies**.

Furthermore, the rule about application

- ▶ if $A : \sigma \rightarrow \tau$ and $B : \sigma$, then $AB : \tau$.

is **modus ponens**.

- ▶ $\lambda x.x : \rho \rightarrow \rho$
- ▶ $\lambda x \lambda y.x : \rho \rightarrow (\sigma \rightarrow \rho)$
- ▶ $\lambda x \lambda y \lambda z.xz(yz) : (\rho \rightarrow (\sigma \rightarrow \tau)) \rightarrow ((\rho \rightarrow \sigma) \rightarrow (\rho \rightarrow \tau))$

Note: The types are **propositional tautologies**.

Furthermore, the rule about application

- ▶ if $A : \sigma \rightarrow \tau$ and $B : \sigma$, then $AB : \tau$.

is **modus ponens**.

Hence, λ -calculus defines some **propositional logic**.

the Curry-Howard correspondence/isomorphism

| | |
|------------------|--------------|
| λ -terms | proofs |
| types | formulas |
| combinators | axioms |
| application | modus ponens |
| and more ... | |

the Curry-Howard correspondence/isomorphism

| | |
|------------------|--------------|
| λ -terms | proofs |
| types | formulas |
| combinators | axioms |
| application | modus ponens |
| and more ... | |

Example

Recall that $SKK = I$ and $I : \tau \rightarrow \tau$. Hence SKK is a proof of $\tau \rightarrow \tau$, *if it can be properly typed*.

Exercise

Find the types for SKK !

Theorem

The λ -calculus defines intuitionistic logic of implication.

Theorem

The λ -calculus defines intuitionistic logic of implication.

Proof-idea

Theorem

The λ -calculus defines intuitionistic logic of implication.

Proof-idea

1. Completeness: Show that the formulas corresponding to the types of K and S and modus ponens axiomatize intuitionistic logic of implication.

Theorem

The λ -calculus defines intuitionistic logic of implication.

Proof-idea

1. Completeness: Show that the formulas corresponding to the types of K and S and modus ponens axiomatize intuitionistic logic of implication.
2. Soundness: Since every λ -term can be constructed from K and S , only intuitionistic tautologies are provable.



intuitionistic logic

The standard logic is called **classical logic** to be distinguished from **intuitionistic logic** which is a.k.a. **constructive logic**.

intuitionistic logic

The standard logic is called **classical logic** to be distinguished from **intuitionistic logic** which is a.k.a. **constructive logic**.

- ▶ language: $\rightarrow, \wedge, \vee, \neg$ and \forall, \exists ;
(often \perp instead of \neg and $\neg A$ is expressed by $A \rightarrow \perp$)
- ▶ weaker than classical logic, e.g. t.f.a. **not** provable in int. logic:
 - ▶ $A \vee \neg A$
 - ▶ $\neg\neg A \rightarrow A$
 - ▶ $\neg\forall x.A \rightarrow \exists x.\neg A$
- ▶ the connectives $\rightarrow, \wedge, \vee, \neg$ and quantifiers \forall, \exists are independent (one cannot be defined from the others)

some constructive properties of intuitionistic logic

- ▶ if $\vdash A \vee B$, then either $\vdash A$ or $\vdash B$
- ▶ if $\vdash \exists x A(x)$, then $\vdash A(t)$ for some term t

some constructive properties of intuitionistic logic

- ▶ if $\vdash A \vee B$, then either $\vdash A$ or $\vdash B$
- ▶ if $\vdash \exists x A(x)$, then $\vdash A(t)$ for some term t
- ▶ one cannot use proofs by contradiction to prove non-negated sentences
 - ▶ if we assume $\neg A$ and get \perp , we only can deduce $\neg\neg A$;
 - ▶ however, to prove $\neg B$, it suffices to assume B and prove \perp .

Propositional intuitionistic logic of **implication** is also weaker:

$$((p \rightarrow q) \rightarrow p) \rightarrow p$$

(*Peirce Law*) is a classical tautology, but not intuitionistic.

proof systems for intuitionistic logic

1. Hilbert style with carefully chosen axioms and rules.
 - ▶ this corresponds to the λ -calculus formalized using combinators

proof systems for intuitionistic logic

1. Hilbert style with carefully chosen axioms and rules.
 - ▶ this corresponds to the λ -calculus formalized using combinators
2. Sequent calculus with the restriction: **at most one formula in the consequent**, i.e.,

$$A_1, \dots, A_n \rightarrow B \quad \text{or} \quad A_1, \dots, A_n \rightarrow$$

proof systems for intuitionistic logic

1. Hilbert style with carefully chosen axioms and rules.
 - ▶ this corresponds to the λ -calculus formalized using combinators
2. Sequent calculus with the restriction: **at most one formula in the consequent**, i.e.,

$$A_1, \dots, A_n \rightarrow B \quad \text{or} \quad A_1, \dots, A_n \rightarrow$$

3. Natural deduction system with the negation elimination rule (=“proof by contradiction”) omitted.
 - ▶ this corresponds to the λ -calculus formalized using λ -terms.

natural deduction and λ -calculus

Again we restrict ourselves to the implicational fragment of propositional logic.

natural deduction and λ -calculus

Again we restrict ourselves to the implicational fragment of propositional logic.

Recall the nat. ded. rules for \rightarrow .

\rightarrow introduction

$$\frac{\begin{array}{c} [A] \\ \vdots \\ B \end{array}}{A \rightarrow B}$$

\rightarrow elimination

$$\frac{A \quad A \rightarrow B}{B}$$

$$\frac{[A] \quad \vdots \quad B}{A \rightarrow B} \qquad \frac{A \quad A \rightarrow B}{B}$$

Suppose we have a term $M : \beta$ with a free variable $x : \alpha$. Then

$$\lambda x.M : \alpha \rightarrow \beta$$

So λ -abstraction corresponds to \rightarrow introduction. The object variable x is the assumption.

$$\frac{[A] \quad \vdots \quad B}{A \rightarrow B} \qquad \frac{A \quad A \rightarrow B}{B}$$

Suppose we have a term $M : \beta$ with a free variable $x : \alpha$. Then

$$\lambda x.M : \alpha \rightarrow \beta$$

So λ -abstraction corresponds to \rightarrow introduction. The object variable x is the assumption.

We already know: application corresponds to \rightarrow elimination (= modus ponens).

In the system of natural deduction we have **normalization** instead of **cut-elimination**. Normal proofs are, essentially, proofs without elimination rules.

In the system of natural deduction we have **normalization** instead of **cut-elimination**. Normal proofs are, essentially, proofs without elimination rules.

Thus we can extend ...

the Curry-Howard correspondence/isomorphism

| | |
|------------------------|----------------------------|
| λ -terms | proofs |
| types | formulas |
| combinators | axioms |
| application | \rightarrow elimination |
| object variable | assumption |
| λ -abstraction | \rightarrow introduction |
| normalization of terms | normalization of proofs |
| and more ... | |

Formalization of intuitionistic logic in λ -calculus

Language

- ▶ variables x_1, x_2, \dots
- ▶ **type constructors**, which are functions and operations on types (e.g., binary function \rightarrow)
- ▶ constants, called **combinators** (e.g., I, S, K)
- ▶ functions to form terms (e.g., binary function of application xy)
- ▶ **term operators**, which bind variables in terms (e.g., λx)

Functions can only be applied to terms of appropriate types.

Types are all propositional, or first-order formulas in the logic that we consider. E.g., in the **implicational fragment** of propositional intuitionistic logic, we have **types for propositional variables** and the type constructor \rightarrow .

Formalization of intuitionistic logic in λ -calculus

Language

- ▶ variables x_1, x_2, \dots
- ▶ **type constructors**, which are functions and operations on types (e.g., binary function \rightarrow)
- ▶ constants, called **combinators** (e.g., I, S, K)
- ▶ functions to form terms (e.g., binary function of application xy)
- ▶ **term operators**, which bind variables in terms (e.g., λx)

Functions can only be applied to terms of appropriate types.

Types are all propositional, or first-order formulas in the logic that we consider. E.g., in the **implicational fragment** of propositional intuitionistic logic, we have **types for propositional variables** and the type constructor \rightarrow .

In the sequel we will **not use combinators**.

A **judgment** (česky: **úsuděk**) is

$$x_1 : \sigma_1, x_2 : \sigma_2, \dots, x_n : \sigma_n \vdash M : \tau$$

where

- ▶ x_1, \dots, x_n are variables,
- ▶ $\sigma_1, \dots, \sigma_n, \tau$ are types, and
- ▶ M is a term.
- ▶ $x_1 : \sigma_1, x_2 : \sigma_2, \dots, x_n : \sigma_n$ is called **environment**

Interpretation: M is a proof of τ from assumptions. The variables x_1, \dots, x_n stand for possible proofs that we do not have yet.

implicational fragment

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

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

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

where x does not occur in Γ .

conjunction

We need a new **type constructor**. Conjunction corresponds to the **Cartesian product** \times . For two types σ and τ , we can form the type $\sigma \times \tau$, but we will use the “logic” notation \wedge instead of \times .

Further, we need three **functions** $\langle .. \rangle$ and projections π_1, π_2 .

conjunction

We need a new **type constructor**. Conjunction corresponds to the **Cartesian product** \times . For two types σ and τ , we can form the type $\sigma \times \tau$, but we will use the “logic” notation \wedge instead of \times .

Further, we need three **functions** $\langle \cdot \rangle$ and projections π_1, π_2 .

$$\frac{\Gamma \vdash M : \phi \quad \Gamma \vdash N : \psi}{\Gamma \vdash \langle M, N \rangle : \psi \wedge \phi}$$
$$\frac{\Gamma \vdash M : \psi \wedge \phi}{\Gamma \vdash \pi_1(M) : \psi} \quad \frac{\Gamma \vdash M : \psi \wedge \phi}{\Gamma \vdash \pi_2(M) : \phi}$$

Remarks

1. projections can only be applied to terms of types of the form $\psi \wedge \phi$.
2. environment plays no role here

disjunction

Disjunction corresponds to the **disjoint union**. For the type constructor, we will use the logical symbol \vee .

We need two functions in_1^ψ, in_2^ϕ for any pair of types ψ, ϕ , and one binary operator κ .⁵

$$\frac{\Gamma \vdash M : \phi}{\Gamma \vdash in_1^\psi(M) : \psi \vee \phi} \quad \frac{\Gamma \vdash M : \psi}{\Gamma \vdash in_2^\phi(M) : \psi \vee \phi}$$

$$\frac{\Gamma \vdash L : \phi \vee \psi \quad \Gamma, x : \phi \vdash M : \rho \quad \Gamma, y : \psi \vdash N : \rho}{\Gamma \vdash \kappa(L, [x]M, [y]N) : \rho}$$

κ binds variable x in M and variable y in N .

⁵Symbol κ is my notation; Sørensen and Urczyzyn use **case** $in_1^{\psi \vee \phi}(L)$ **of** $[x]M$ **of** $[y]N$, and similar for in_2 .

disjunction

Disjunction corresponds to the [disjoint union](#). For the type constructor, we will use the logical symbol \vee .

We need two functions in_1^ψ, in_2^ϕ for any pair of types ψ, ϕ , and one binary operator κ .⁵

$$\frac{\Gamma \vdash M : \phi}{\Gamma \vdash in_1^\psi(M) : \psi \vee \phi} \quad \frac{\Gamma \vdash M : \psi}{\Gamma \vdash in_2^\phi(M) : \psi \vee \phi}$$

$$\frac{\Gamma \vdash L : \phi \vee \psi \quad \Gamma, x : \phi \vdash M : \rho \quad \Gamma, y : \psi \vdash N : \rho}{\Gamma \vdash \kappa(L, [x]M, [y]N) : \rho}$$

κ binds variable x in M and variable y in N .

$\kappa(L, [x]M, [y]N)$ is a construction that makes a proof of ρ from L and any pair of proofs of ϕ and ψ .

⁵Symbol κ is my notation; Sørensen and Urczyzyn use **case** $in_1^{\psi \vee \phi}(L)$ **of** $[x]M$ **of** $[y]N$, and similar for in_2 .

contradiction

We need another function symbol ϵ_ψ for every type ψ .

$$\frac{\Gamma \vdash M : \perp}{\Gamma \vdash \epsilon^\psi(M) : \psi}$$

Exercise

Prove $\phi \rightarrow ((\phi \rightarrow \perp) \rightarrow \psi)$.

first-order logic in λ -calculus

Variables

1. **individual** x_1, x_2, \dots as before
2. **proof variables** a, b, c, \dots — variables of the first order language⁶

Types

- ▶ any first-order formula

Terms

1. terms of the first-order language, s, t, \dots
2. λ -terms, M, N, \dots ; they may contain **proof variables** and **first-order terms**

⁶paradoxical names; I would call x_1, x_2, \dots proof variables because they represent possible proofs.

Quantifier rules

$$(\forall I) \frac{\Gamma \vdash M : \phi}{\Gamma \vdash \lambda a.M : \forall a \phi} \quad (\forall E) \frac{\Gamma \vdash M : \forall a \phi}{\Gamma \vdash Mt : \phi[a := t]}$$

where a is not in Γ

$$(\exists I) \frac{\Gamma \vdash M : \phi[a := t]}{\Gamma \vdash [t, M]^{\exists a \phi} : \exists a \phi} \quad (\exists E) \frac{\Gamma \vdash M : \exists a \phi \quad \Gamma, x : \phi \vdash N : \psi}{\kappa^1([a]M, [a, x]N) : \psi}$$

where a is not in Γ and ψ

Free and bound variables

- ▶ In $\lambda a.M : \forall a\phi$, λ binds a .
- ▶ In $\kappa^1([a]M, [a, x]N)$, κ binds both a and x . The notation indicates that x can only occur in N .⁷
- ▶ In $[t, M]^{\exists a\phi}$, free variables are the free variables of t , M and $\exists a\phi$.
- ▶ In first order logic we should use $\psi \vee \phi$ as the superscript for in_1 and in_2 because the free variables of in_1 and in_2 include the free variables of $\psi \vee \phi$.

⁷Again, κ^1 is my notation.

Cut

A formalization with the cut rule is possible.

$$\frac{\Gamma \vdash N : \phi \quad \Gamma, x : \phi \vdash M : \psi}{\Gamma \vdash M \langle x = N \rangle : \psi}$$

Gödel's theory T

K. Gödel, “Über eine bisher noch nicht benutzte Erweiterung des finiten Standpunktes”, *Dialectica*, 12, pp. 280-287, (1958)

J. Avigad and S. Feferman, Gödel's Functional Interpretation, Chapter V. of *Handbook of Proof Theory*.

Gödel's goal was to give a proof of the consistency of PA based on something else than was used before.

Primitive Recursive Arithmetic, PRA, [Skolem 1923]

- ▶ a function symbol for every primitive recursive function
- ▶ only open formulas
- ▶ axioms:
 1. axioms about the projection functions
 2. axioms about the successor function

$$S(x) \neq 0, \quad S(x) = S(y) \rightarrow x = y$$

3. defining equations for the function symbols

$$h(0, x) = f(x), \quad h(S(n), x) = g(n, h(n, x), x)$$

4. induction rule for every open formula ϕ and term t

$$\frac{\phi(0, x), \quad \phi(n, x) \rightarrow \phi(S(n), x)}{\phi(t, x)}$$

Theorem (Parsons, Takeuti)

$I\Sigma_1(PRA)$ is Π_2 conservative over PRA .

Theorem (Parsons, Takeuti)

$I\Sigma_1(PRA)$ is Π_2 conservative over PRA .

Similarly

Theorem

$S_2^1(PV)$ is $\forall\Sigma_1^b$ conservative over PV .

Theorem (Parsons, Takeuti)

$I\Sigma_1(PRA)$ is Π_2 conservative over PRA .

Similarly

Theorem

$S_2^1(PV)$ is $\forall\Sigma_1^b$ conservative over PV .

Gödel's T has similar relation to HA , a fortiori, by double negation interpretation, also to PA .

Skolem vs. Gödel

Consider $\forall x \exists y \forall z \exists u. \phi(x, y, z, u)$ with ϕ open.

1. Skolemization

$$\exists f \exists g \forall x \forall z. \phi(x, f(x), z, g(x, z)),$$

where $f : \mathbb{N} \rightarrow \mathbb{N}$, $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$.

2. Gödel's interpretation using higher order functionals

$$\exists f \exists G \forall x \forall z. \phi(x, f(x), z, G(x)(z))$$

where $f : \mathbb{N} \rightarrow \mathbb{N}$, $G : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$.

Skolem vs. Gödel

Consider $\forall x \exists y \forall z \exists u. \phi(x, y, z, u)$ with ϕ open.

1. Skolemization

$$\exists f \exists g \forall x \forall z. \phi(x, f(x), z, g(x, z)),$$

where $f : \mathbb{N} \rightarrow \mathbb{N}$, $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$.

2. Gödel's interpretation using higher order functionals

$$\exists f \exists G \forall x \forall z. \phi(x, f(x), z, G(x)(z))$$

where $f : \mathbb{N} \rightarrow \mathbb{N}$, $G : \mathbb{N} \rightarrow (\mathbb{N} \rightarrow \mathbb{N})$.

Why *functionals*?

- ▶ If $HA \vdash \forall x \exists y \forall z \exists u. \phi(x, y, z, u)$, then f is computable, but in general, g is not,
- ▶ but functional G is, in a sense, computable.

Language

1. types are O (which is \mathbb{N}) and $\tau \rightarrow \sigma$ for all τ, σ
2. for each type τ , variables $x, y, \dots : \tau$
3. constants $0 : O$ (zero), $S : O \rightarrow O$ (successor function)
4. for all types ρ, σ, τ the combinators $K^{\rho, \sigma}$, $S^{\rho, \sigma, \tau}$ ⁸ and **recursor** $R^{\rho, \sigma}$
5. the equality sign $=$, propositional connectives

Formulas

- ▶ open (= quantifier free) formulas

⁸At the risk of confusion we use S for the successor function and the combinators. What we mean should be possible to tell from the context.

Axioms and rules

1. axioms and rules of intuitionistic propositional logic
2. substitution

$$\frac{s = t}{s[x/u] = t[x/u]}$$

3. axioms for zero and successor (as for PRA)
4. axioms for combinators (types omitted):

$$K(s, t) = s, \quad S(r, s, t) = r(t)(s(t))$$

5. axioms for recursors (types and parameters omitted):

$$R(f, g, 0) = f, \quad R(f, g, S(n)) = g(n, R(f, g, n))$$

6. induction as a rule (as for PRA) for all formulas of \mathcal{T} , (i.e., open formulas)

Axioms and rules

1. axioms and rules of intuitionistic propositional logic
2. substitution

$$\frac{s = t}{s[x/u] = t[x/u]}$$

3. axioms for zero and successor (as for PRA)
4. axioms for combinators (types omitted):

$$K(s, t) = s, \quad S(r, s, t) = r(t)(s(t))$$

5. axioms for recursors (types and parameters omitted):

$$R(f, g, 0) = f, \quad R(f, g, S(n)) = g(n, R(f, g, n))$$

6. induction as a rule (as for PRA) for all formulas of T , (i.e., open formulas)

Possible modifications: add the product type constructor \times , use lambda abstraction instead of combinators.

The *Dialectica* interpretation

Given arithmetical formula ϕ , one constructs inductively a formula, *the Dialectica interpretation*, ϕ^D of the form

$$\exists \bar{x} \forall \bar{y}. \phi_D,$$

where \bar{x} and \bar{y} denote tuples of variables of possibly different types, and ϕ_D is open (i.e., quantifier-free).

The *Dialectica* interpretation

Given arithmetical formula ϕ , one constructs inductively a formula, *the Dialectica interpretation*, ϕ^D of the form

$$\exists \bar{x} \forall \bar{y}. \phi_D,$$

where \bar{x} and \bar{y} denote tuples of variables of possibly different types, and ϕ_D is open (i.e., quantifier-free).

Jumping ahead, our goal is to prove

Theorem (Gödel, 1958)

If ϕ is an arithmetical formula and $HA \vdash \phi$, then there exists a tuple of terms \bar{t} such that $T \vdash \phi_D(\bar{t}, \bar{x})$.

Let $\phi^D = \exists \bar{x} \forall \bar{y} \phi_D$ and $\psi^D = \exists \bar{u} \forall v \psi_D$.

1. $\phi^D = \phi_D := \phi$ for ϕ atomic
2. $(\phi \wedge \psi)^D := \exists \bar{x}, \bar{u} \forall \bar{y}, \bar{v} (\phi_D \wedge \psi_D)$
3. $(\phi \vee \psi)^D := \exists z, \bar{x}, \bar{u} \forall \bar{y}, \bar{v} ((z = 0 \wedge \phi_D) \vee (z = 1 \wedge \psi_D))$
4. $(\forall z. \phi(z))^D := \exists \bar{X} \forall z, \bar{y} \phi_D(\bar{X}(z), \bar{y}, z)$
5. $(\exists z. \phi(z))^D := \exists z, \bar{x} \forall \bar{y} \psi_D$
6. $(\phi \rightarrow \psi)^D := \exists \bar{U}, \bar{Y} \forall \bar{x}, \bar{v} (\phi_D(\bar{x}, \bar{Y}(\bar{x}, \bar{v})) \rightarrow \psi_D(\bar{U}(\bar{x}), \bar{v}))$
7. $(\neg \phi)^D := \exists \bar{Y} \forall \bar{x} \neg \phi_D(\bar{x}, \bar{Y}(\bar{x}))$

Exercise

Compute $\neg\neg\forall x\exists y \phi(x, y)$, for ϕ open.

1. $(\forall x\exists y \phi(x, y))^D = \exists Y\forall x \phi(x, Y(x))$,
where $Y : \mathcal{O} \rightarrow \mathcal{O}$
2. $(\neg\forall x\exists y \phi(x, y))^D = \exists Y\forall X \neg\phi(Y(X), X(Y(X)))$,
where $X : \mathcal{O} \rightarrow \mathcal{O}$, $Y : (\mathcal{O} \rightarrow \mathcal{O}) \rightarrow \mathcal{O}$
3. $(\neg\neg\forall x\exists y \phi(x, y))^D = \exists Y\forall X \neg\neg\phi(X(Y(X)), Y(X)(X(Y(X))))$,
where $X : (\mathcal{O} \rightarrow \mathcal{O}) \rightarrow \mathcal{O}$, $Y : ((\mathcal{O} \rightarrow \mathcal{O}) \rightarrow \mathcal{O}) \rightarrow (\mathcal{O} \rightarrow \mathcal{O})$

Theorem (Gödel, 1958)

If ϕ is an arithmetical formula and $HA \vdash \phi$, then there exists a tuple of terms \bar{t} such that $T \vdash \phi_D(\bar{t}, \bar{x})$.

The proof proceed by induction on the length of the proof of ϕ . We only consider two cases.⁹

⁹from now on the bars denoting tuples will be omitted

Theorem (Gödel, 1958)

If ϕ is an arithmetical formula and $HA \vdash \phi$, then there exists a tuple of terms \bar{t} such that $T \vdash \phi_D(\bar{t}, \bar{x})$.

The proof proceed by induction on the length of the proof of ϕ . We only consider two cases.⁹

1. (modus ponens) Suppose ψ is derived in HA from ϕ and $\phi \rightarrow \psi$. By the induction assumption we have:

$$T \vdash \phi_D(r, y) \text{ and } T \vdash \phi_D(x, s(x, v)) \rightarrow \psi_D(t(x), v)$$

for some terms r, s, t . Substitute $x := r$ and $y := s(r, v)$. Thus we get

$$T \vdash \phi_D(r, s(r, v)) \text{ and } T \vdash \phi_D(r, s(r, v)) \rightarrow \psi_D(t(r), v).$$

Whence we get, by modus ponens,

$$T \vdash \psi_D(t(r), v).$$

⁹from now on the bars denoting tuples will be omitted

2. (induction) Suppose $\phi(p)$ was derived in HA from $\phi(0)$ and $\phi(x) \rightarrow \phi(S(x))$, where p is a term, $p : O$.

By the induction hypothesis we have:

$$T \vdash \phi_D(r, y, 0) \text{ and } T \vdash \phi_D(x, s(x, v, z), z) \rightarrow \phi_D(t(x, S(z)), v, S(z)),$$

where x, y, z, v are variables and r, s, t are terms. As a warm-up, we note that

$$T \vdash \phi_D(t(r, S(0)), v, S(0)).$$

follows the same way as modus ponens.

The term for $\phi(p)$ will be p -times iterated t , which we will get using the recursor.

But to show that it is **provable in T** , we also need to iterate term s , which is also constructed using the recursor.

The situation can be *schematically* described as follows. By substitution we have

$$T \vdash \phi_D(r, s^p(x), 0)$$

and

$$T \vdash \phi_D(t^i(r), s^{p-i}(x), i) \rightarrow \phi_D(t^{i+1}(r), s^{p-i-1}(x), i+1).$$

From this, we prove by induction

$$T \vdash \phi_D(t^p(r), x, p).$$

