Formal Logics

Introduction

Adrien Pommellet

June 23, 2025

What is this class about?

Logics. A coherent mode of reasoning that allows one to assess the truth of statements.

A Brief History of Logics

Blame the Greeks

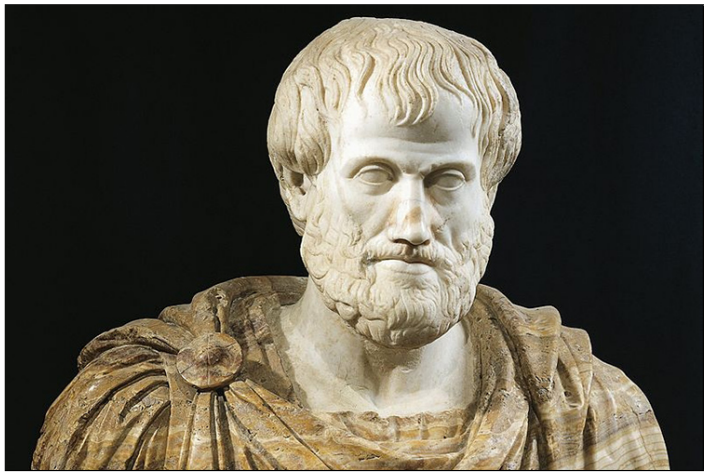


Figure 1: Aristotle (384–322 BC).

A Brief History of Logics

Aristotle's syllogism

- All men are mortal.
- Socrates is a man.
- Thus Socrates is mortal.

A Brief History of Logics

Leibniz's universal language

Figure 2: Gottfried Wilhelm Leibniz (1646–1716).

A Brief History of Logics

Hilbert's "We must know — we will know."

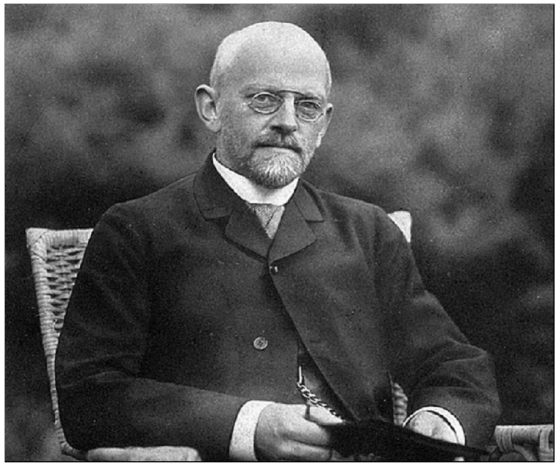


Figure 3: David Hilbert (1862–1943).

Syntax and Semantics

Chasing truth

Syntax. The structural analysis of statements expressed as formulas.
Truth is what we **build** using proofs.

Semantics. Interpreting formulas according to a mathematical model.
Truth is a pre-existing absolute meant to be **discovered**.

Syntax and Semantics

The duality of truth

Are these two notions equivalent?

Course Overview
A brief primer

- The full course consists in six two hour long **lectures**.
- The slides and detailed class notes can be found on **Moodle**. Beware of updates!
- You may ask questions in-between classes on a dedicated **Moodle** forum.

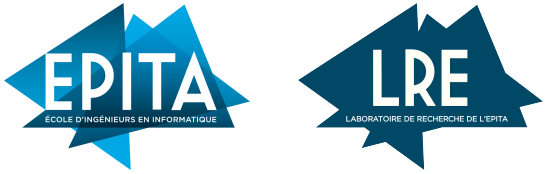
Course Overview
The dreaded exams

- A short mid-term exam (**7 points**) + a final exam (**13 points**).
- Make sure that you can properly access **Moodle Exam**.
- A short mock exam will allow you to get used to the interface.

Good luck!

Formal Logics
Propositional Logic

Adrien Pommellet



June 23, 2025

Constructing arbitrarily complex objects from simpler ones through the means of fixed rules.

Inductive definition of a set $\mathcal{T}$

It features three things:

Atomic objects. A set $\mathcal{A}$.

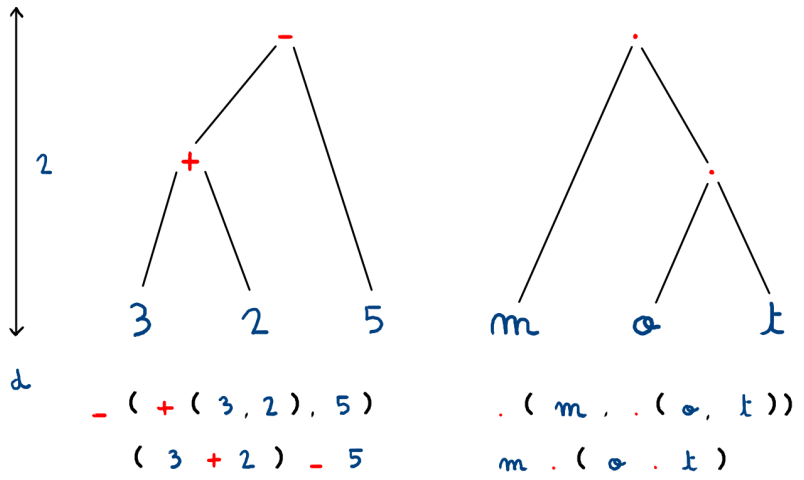
Constructors. A set $\mathcal{C}$; to each $op \in \mathcal{C}$, we match its **arity** $ar(op) \in \mathbb{N}$.

Depth. $d \in \mathbb{N} \cup \{\infty\}$.

Starting from the atoms $\mathcal{A}$, we add new elements to $\mathcal{T}$ by combining existing elements using constructors in $\mathcal{C}$, allowing a **nesting depth** of at most d (or finite but unbounded if $d = \infty$).

Arithmetic terms	**Words Σ^***
Atoms. $\mathbb{N}$.	Atoms. $\Sigma \cup \{\varepsilon\}$.
Rules. $\{+^2, -^2\}$.	Rules. $\{\cdot^2\}$.
Depth. ∞.	Depth. ∞.

The exponent ^{2} stands for the **arity** of each rule. Depth ∞ means that rules can be applied an unbounded but still **finite** number of times.



We can define **functions** on $\mathcal{T}$ such as depth inductively as well:

Depth of a term

Given an inductively defined set $\mathcal{T}$, we will define the **depth** function $\delta_{\mathcal{T}} : \mathcal{T} \to \mathbb{N}$ as follows:

- On $\mathcal{A}$. For each atom $a \in \mathcal{A}, \delta_{\mathcal{T}}(a) = 0$.
- Using $\mathcal{C}$. For each $op \in \mathcal{C}$ such that $ar(op) = n$ and $(t_1, \ldots, t_n) \in \mathcal{T}^n$, $\delta_{\mathcal{T}}(op(t_1, \ldots, t_n)) = \max(\{\delta_{\mathcal{T}}(t_1), \ldots, \delta_{\mathcal{T}}(t_n)\}) + 1$.

Intuitively, $\delta_{\mathcal{T}}(t)$ is the depth of t's **syntactic tree**.

Goal. Given an inductively defined set $\mathcal{T}$ and a predicate $\mathcal{P}$, prove that $\forall x \in \mathcal{T}, P(x)$ holds.

We are going to use a proof by **structural** induction.

- Base case. Prove that $\forall a \in \mathcal{A}, \mathcal{P}(a)$ holds.
- Inductive case. For each constructor $op \in \mathcal{C}$, prove that if $ar(op) = n$ and $t_1, \ldots, t_n \in \mathcal{T}$ are n terms such that $\mathcal{P}(t_i)$ holds for any $i \in \{1, \ldots n\}$, then $\mathcal{P}(op(t_1, \ldots, t_n))$ holds.

Exercise 1. Prove that an arithmetic expression (as defined inductively earlier) with n operators always features $n + 1$ integers.

We are going to use a proof by **structural** induction on the set of arithmetic expressions.

- Base case. $\mathcal{A} = \mathbb{N}$, and any integer $n \in \mathbb{N}$ features 0 operator as well as a single integer (itself).
- Inductive case. Consider two arithmetic expressions e_1, e_2 featuring respectively n_1 and n_2 operators as well as $n_1 + 1$ and $n_2 + 1$ integers. Then $e_1 + e_2$ features $n_1 + n_2 + 1$ operators and $n_1 + n_2 + 2$ integers.

  In a similar fashion, the property holds for $e_1 - e_2$.

Propositional Formulas
An inductive definition

Propositional formulas

The set $\mathcal{F}_0 = \mathcal{F}_{\{\top,\bot,\neg,\wedge,\vee,\Rightarrow,\Leftrightarrow\}}$ is defined inductively as follows:

- $\mathcal{A}$. $\mathcal{V} \cup \{\top, \bot\}$ where $\mathcal{V}$ is a set of **variables**.
- $\mathcal{C}$. $\{\neg^1, \wedge^2, \vee^2, \Rightarrow^2, \Leftrightarrow^2\}$.
- d. ∞.

E Consider $(A \wedge (\neg B)) \Rightarrow C \in \mathcal{F}_0$.

Propositional Formulas
Valuations

Valuation

It is a function $\nu : \mathcal{V} \to \{\texttt{true}, \texttt{false}\}$.

Truth assignment function

Given a valuation ν, it is a function $|\ |_\nu : \mathcal{F}_0 \to \{\texttt{true}, \texttt{false}\}$.

Propositional Formulas
Semantics

Tarski's semantics

Defined inductively as follows:

- $|\top|_\nu = \texttt{true}$.
- $|\bot|_\nu = \texttt{false}$.
- Given $x \in \mathcal{V}$, $|x|_\nu = \nu(x)$.
- Given $\varphi \in \mathcal{F}_0$, $|\neg\varphi|_\nu = \texttt{true}$ if and only if $|\varphi|_\nu = \texttt{false}$.
- Given $\varphi, \psi \in \mathcal{F}_0$.
  - $|\varphi \vee \psi|_\nu = \texttt{true}$ if and only if $|\varphi|_\nu = \texttt{true}$ or $|\psi|_\nu = \texttt{true}$.
  - $|\varphi \wedge \psi|_\nu = \texttt{true}$ if and only if $|\varphi|_\nu = \texttt{true}$ and $|\psi|_\nu = \texttt{true}$.
  - $|\varphi \Rightarrow \psi|_\nu = \texttt{true}$ if and only if $|\varphi|_\nu = \texttt{true}$ implies $|\psi|_\nu = \texttt{true}$.
  - $|\varphi \Leftrightarrow \psi|_\nu = \texttt{true}$ if and only if $|\varphi|_\nu = \texttt{true}$ is equivalent to $|\psi|_\nu = \texttt{true}$.

About Formulas
Syntactic conventions and semantic properties

The following properties and conventions hold:

$\vee$ and $\wedge$. **Commutative** w.r.t Tarski's semantics: $|\varphi \vee \psi|_\nu = |\psi \vee \varphi|_\nu$.
Associative as well: $|\psi_1 \vee (\psi_2 \vee \psi_3)|_\nu = |(\psi_1 \vee \psi_2) \vee \psi_3|_\nu$.

$\Rightarrow$ and $\Leftrightarrow$. By convention, **right associative**: $\psi_1 \Rightarrow \psi_2 \Rightarrow \psi_3$ means $\psi_1 \Rightarrow (\psi_2 \Rightarrow \psi_3)$.

Priority rules. The order $\Leftrightarrow < \Rightarrow < \wedge < \vee < \neg$ applies by convention.

About Formulas

An example

$$(\neg X \vee Y \wedge Z \Rightarrow U) \Leftrightarrow V$$
$$((\neg X) \vee Y \wedge Z \Rightarrow U) \Leftrightarrow V$$
$$(((\neg X) \vee Y) \wedge Z \Rightarrow U) \Leftrightarrow V$$
$$((((\neg X) \vee Y) \wedge Z) \Rightarrow U) \Leftrightarrow V$$

About Formulas

Tautologies and antilogies

Tautology

A propositional formula φ such that for any valuation ν, $|\varphi|_\nu = \texttt{true}$.

Antilogy

A propositional formula φ such that for any valuation ν, $|\varphi|_\nu = \texttt{false}$.

Satisfiable

A propositional formula φ such that there exists a valuation ν verifying $|\varphi|_\nu = \texttt{true}$.

Semantic Equivalence

A proper definition

Equivalence

φ and ψ are **semantically equivalent** if for any valuation ν, $|\varphi|_\nu = |\psi|_\nu$. Then $\varphi \equiv \psi$.

Any tautology is semantically equivalent to $\top$, and any antilogy to $\bot$.

Semantic Equivalence

An equivalence relation

The semantic equivalence $\equiv$ is an equivalence relation:

- **Reflexive.** $\varphi \equiv \varphi$.
- **Symmetric.** $\varphi \equiv \psi$ if and only if $\psi \equiv \varphi$.
- **Transitive.** If $\psi_1 \equiv \psi_2$ and $\psi_2 \equiv \psi_3$ then $\psi_1 \equiv \psi_3$.

Semantic Equivalence
Sub-formulas

A property of sub-formulas

Let ψ_1 be a **sub-formula** of φ_1. If $\psi_2 \in \mathcal{F}_0$ is such that $\psi_1 \equiv \psi_2$, then replacing ψ_1 with ψ_2 in φ_1's definition results in a new formula $\varphi_2 \in \mathcal{F}_0$ such that $\varphi_1 \equiv \varphi_2$.

It can be proven by structural induction on φ.

Semantic Equivalence
What about $\Leftrightarrow$?

Property

$\varphi \equiv \psi$ if and only if $(\varphi \Leftrightarrow \psi)$ is a tautology.

It's a consequence of Tarski's semantics.

Formal Logics
Properties of Propositional Formulas

Adrien Pommellet

June 23, 2025

Truth Tables
A definition

Truth table of φ

A table that sets out the **values** of $|\varphi|_\nu$ for **each possible valuation** ν of its relevant logical variables.

Conventionally, we write $\texttt{true} := 1$ and $\texttt{false} := 0$ in truth tables.

Truth Tables

The main operators I

A	B	$A \wedge B$
0	0	0
0	1	0
1	0	0
1	1	1

A	B	$A \vee B$
0	0	0
0	1	1
1	0	1
1	1	1

A	B	$A \Rightarrow B$
0	0	1
0	1	1
1	0	0
1	1	1

Truth Tables

The main operators II

A	B	$A \Leftrightarrow B$
0	0	1
0	1	0
1	0	0
1	1	1

A	$\neg A$
0	1
1	0

Truth Tables

An example

Prove that $\psi = P \Rightarrow Q \Rightarrow P$ is a tautology.

E

P	Q	$Q \Rightarrow P$	$P \Rightarrow (Q \Rightarrow P)$
0	0	1	1
0	1	0	1
1	0	1	1
1	1	1	1

Practical Application

Exercise 1. Prove that $\varphi = A \vee B \Rightarrow (A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C$ is a tautology.

Answer

As $\Rightarrow$ is right associative, $\varphi = (A \vee B) \Rightarrow ((A \Rightarrow C) \Rightarrow ((B \Rightarrow C) \Rightarrow C))$.

A	B	C	$B \Rightarrow C$	$(B \Rightarrow C) \Rightarrow C$	$A \Rightarrow C$	$(A \Rightarrow C) \Rightarrow ((B \Rightarrow C) \Rightarrow C)$	$A \vee B$	φ
0	0	0	1	0	1	0	0	1
0	0	1	1	1	1	1	0	1
0	1	0	0	1	1	1	1	1
0	1	1	1	1	1	1	1	1
1	0	0	1	0	0	1	1	1
1	0	1	1	1	1	1	1	1
1	1	0	0	1	0	1	1	1
1	1	1	1	1	1	1	1	1

Truth Tables

Equivalence

Property

Two formulas with the same set of input variables are equivalent if and only if they have the same truth table.

It is a direct consequence of the definition of truth tables.

Properties of $\mathcal{F}_0$

Distributivity and De Morgan's laws

Distributivity

For any $P, Q, R \in \mathcal{F}_0$:

$$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$$
$$P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$$

De Morgan's laws

For any $P, Q \in \mathcal{F}_0$:

$$\neg(P \wedge Q) \equiv \neg P \vee \neg Q$$
$$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$$

Properties of $\mathcal{F}_0$

Double negation and material implication

Double negation

For any $P \in \mathcal{F}_0$:

$$\neg(\neg P) \equiv P$$

Material implication

For any $P, Q \in \mathcal{F}_0$:

$$(P \Rightarrow Q) \equiv (\neg P \vee Q)$$

Properties of $\mathcal{F}_0$

Double implication and law of the excluded middle

Double implication

For any $P, Q \in \mathcal{F}_0$:

$$(P \Leftrightarrow Q) \equiv (P \Rightarrow Q) \wedge (Q \Rightarrow P)$$

Law of the excluded middle

For any $P \in \mathcal{F}_0$, $P \vee \neg P$ is a **tautology** and $P \wedge \neg P$ is an **antilogy**.

Properties of $\mathcal{F}_0$

Simplifying formulas

As a consequence of the previous properties:

Theorem

Given a formula $\varphi \in \mathcal{F}_0$, there exists $\psi \in \mathcal{F}_{\{\bot,\neg,\wedge,\vee,\Rightarrow\}}$ such that $\varphi \equiv \psi$.

We can therefore **rewrite** formulas (here, by replacing $\Leftrightarrow$).

Practical Application

Exercise 2. You've just met three people named Alice, Bob, and Carl. They make the following statements:

Alice. "Exactly one of us is telling the truth."

Bob. "We are all lying."

Carl. "The other two are lying."

Can you determine who's lying, and who's telling the truth?

Answer I

Consider three **variables** A, B and C such that A (resp. B, C) is true if and only if Alice (resp. Bob, Carl) tells the truth.

Let us express Alice's (resp. Bob's, Carl's) statement as a propositional formula φ_A (resp. φ_B, φ_C):

$$\begin{aligned} \varphi_A &= (A \wedge \neg B \wedge \neg C) \vee (\neg A \wedge B \wedge \neg C) \vee (\neg A \wedge \neg B \wedge C) \\ \varphi_B &= \neg A \wedge \neg B \wedge \neg C \\ \varphi_C &= \neg A \wedge \neg B \end{aligned}$$

Finally, note that we're looking for a valuation ν such that $\nu(A) = |\varphi_A|_\nu$, $\nu(B) = |\varphi_B|_\nu$, and $\nu(C) = |\varphi_C|_\nu$.

Answer II

The following truth table implies that there is an **unique solution**:

A	B	C	$\varphi_A =$ "*Only one is telling the truth.*"	$\varphi_B =$ "*Everyone is lying.*"	$\varphi_C =$ "*Both A and B are lying.*"
0	0	0	0	1	1
0	0	1	1	0	1
0	1	0	1	0	0
0	1	1	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	1	0	0	0	0
1	1	1	0	0	0

Optional Homework

- Prove two of the aforementioned properties using truth tables.
- Exercises **1A** and **1B** of the 2019-2020 exam.

Formal Logics

The Satisfiability Problem

Adrien Pommellet

June 23, 2025

Introducing SAT

A definition

SAT

The **satisfiability problem** (also written SAT) consists in determining whether a formula $\varphi \in \mathcal{F}_0$ is satisfiable, that is, whether there exists a valuation ν such that $|\varphi|_\nu = \texttt{true}$.

Programs meant to solve this problem are known as SAT **solvers**.

Introducing SAT

Why SAT solvers?

SAT solvers can actually be used to solve a wide variety of problems.

Practical Application

Exercise 1. A, B, C, D, and E all live in a house together. We want to find who is at home and who isn't.

1. If A is at home then so is B.
2. D, E, or both are at home.
3. Either B or C, but not both, are at home.
4. D and C are either both at home or both not at home.
5. If E is at home then A and D are also at home.

Express this problem as a SAT instance.

Answer I

Variables

We first need to introduce Boolean **variables** that will **model** information that is relevant to the problem at hand.

Here, we introduce five variables x_A, x_B, x_C, x_D, and x_E such that x_i being true means i is at home.

Answer II

Constraints

We then express the problem as a set of **constraints**, that is, formulas in $\mathcal{F}_0$ on the variables introduced previously.

$$\begin{aligned}
\varphi_1 &= x_A \Rightarrow x_B \\
\varphi_2 &= x_D \vee x_E \\
\varphi_3 &= (x_B \wedge \neg x_C) \vee (\neg x_B \wedge x_C) \\
\varphi_4 &= (x_D \wedge x_C) \vee (\neg x_D \wedge \neg x_C) \\
\varphi_5 &= x_E \Rightarrow (x_A \wedge x_D)
\end{aligned}$$

Answer III

A SAT instance

We finally reduce the problem to a SAT instance equal to the conjunction of all the constraints $\varphi = \varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5$.

A SAT solver yields a valuation $(x_A, x_B, x_C, x_D, x_E) = (0, 0, 1, 1, 0)$.

Answer IV

Looking for multiple solutions

In order to ensure that $s = (0, 0, 1, 1, 0)$ is the **only** solution, we design a new formula $\varphi_s = \neg x_A \wedge \neg x_B \wedge x_C \wedge x_D \wedge \neg x_E$ that is only satisfied by s.

We then consider the SAT instance $\varphi' = \varphi \wedge \neg\varphi_s$, as φ' is satisfiable if and only if φ admits **another solution** than s.

A SAT solver yields that φ' is not satisfiable. Thus, s is the only solution, and we can conclude that only C and D are at home.

Practical Application

Exercise 2. Can a generic graph $\mathcal{G} = (V, E)$ be coloured using a set C of 3 colours in such a manner two neighbouring vertices in V do not share the same colour? Express this problem as a SAT instance.

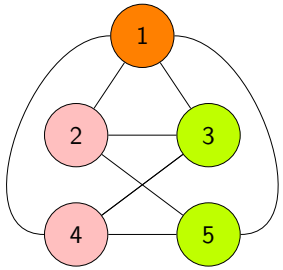


Figure 1: A well-coloured graph $\mathcal{G}$.

Answer I

Not so trivial variables

Superficially, the problem does not seem to involve Boolean variables: the relevant information we want to **encode** is for each vertex in V the colour in $C = \{1, 2, 3\}$ it has been assigned.

However, for each vertex $v \in V$ and each colour $c \in \{1, 2, 3\}$, we can introduce a variable x_v^c meant to be true if node s has been assigned colour c. But this encoding requires **additional constraints**.

Answer II

A constraint on the encoding

Note that for each vertex $v \in V$, exactly one variable among x_v^1, x_v^2, and x_v^3 must be true: a vertex is only coloured once. This intuition yields the following **mutual exclusion** clause for each $v \in V$:

$$\begin{aligned} \varphi_v &= (x_v^1 \wedge \neg x_v^2 \wedge \neg x_v^3) \\ &\vee (\neg x_v^1 \wedge x_v^2 \wedge \neg x_v^3) \\ &\vee (\neg x_v^1 \wedge \neg x_v^2 \wedge x_v^3) \end{aligned}$$

Answer III

The original problem as a constraint

At the **edge** level, the problem is fairly straightforward: if vertices u and v are neighbours, then u being of colour c implies that v is not of colour c. As a consequence, we consider the following constraint:

$$\varphi_E = \bigwedge_{c \in C} \bigwedge_{(u,v) \in E} (x_u^c \Rightarrow \neg x_v^c)$$

Thus, the problem is equivalent to the SAT instance $\varphi = \varphi_E \wedge \bigwedge_{v \in V} \varphi_v$.

Properties of SAT

NP-completeness

Theorem (Cook's)

SAT is NP-complete.

Intuitively, a problem $\mathcal{P}$ is NP if it is **easy to check** (in polynomial time) whether an answer is valid or not.

It **may** still be hard to find a solution (brute forcing SAT is exponential).

Such a problem $\mathcal{P}$ is also NP-complete if any instance of another NP problem can easily (in polynomial time) be **reduced** to an instance of $\mathcal{P}$.

Properties of SAT

Negative normal form

Negative normal form

A formula $\varphi \in \mathcal{F}_0$ is said to be in negative normal form (NNF) if:

- The only constructors connecting sub-statements of φ are $\vee$ and $\wedge$.
- The $\neg$ constructor only appears in front of atomic statements.

Theorem

Given a formula $\varphi \in \mathcal{F}_0$, there exists $\psi \in \mathcal{F}_0$ in NNF such that $\varphi \equiv \psi$.

Consider a proof by **induction** using De Morgan's laws, double negation, material implication, and double implication.

Properties of SAT

Conjunctive normal form

Conjunctive normal form

A formula $\varphi \in \mathcal{F}_0$ is said to be in conjunctive normal form (CNF) if it is of form $\varphi = \bigwedge_i \bigvee_j \psi_{i,j}$ where $\psi_{i,j} = x_{i,j}$ or $\psi_{i,j} = \neg x_{i,j}$ for some $x_{i,j} \in \mathcal{A}$.

Intuitively, φ is a **conjunction of disjunction of variables (or the negation thereof)**.

Theorem

Given a formula $\varphi \in \mathcal{F}_0$, there exists $\psi \in \mathcal{F}_0$ in CNF such that $\varphi \equiv \psi$.

Consider a proof by **induction** on formulas in NNF, then use the previous theorem to generalize this result to generic formulas.

Properties of SAT

The DIMACS format

As CNF allows for various optimizations, most SAT solvers rely on a CNF encoding known as the **DIMACS** format.

$$
\begin{aligned}
& \varphi \\
= \; & (x_1 \vee \neg x_2 \vee x_4) \\
\wedge \; & (\neg x_1 \vee x_3 \vee x_4) \\
\wedge \; & (x_1 \vee x_3)
\end{aligned}
$$

```
1: c CNF, 4 variables, 3 clauses
2: p cnf 4 3
3: 1 -2 4 0
4: -1 3 4 0
5: 1 3 0
```


A Demonstration

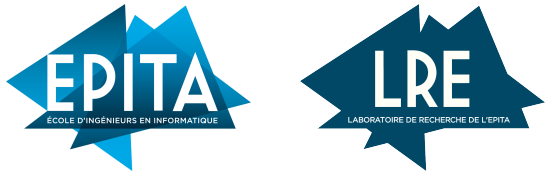
Introducing the `minisat` solver.

See you next class!

Formal Logics

Proof Systems

Adrien Pommellet



June 23, 2025

Defining Proof Systems

Aristotle's example

Hypotheses. We start from propositions that hold true.
Socrates is a man.

Inference. Then we use rules.
Men are mortal.

Conclusion. In order to produce new propositions that are also true.
Socrates is mortal.

Defining Proof Systems

Formal axioms

Axiom

It's a formula $\varphi \in \mathcal{F}_0$ that is **considered true a priori**. It is written:

$$\frac{}{\varphi}\ [a]$$

E A possible axiom would be the *law of the excluded middle*:

$$\frac{}{A \vee \neg A}\ [\textit{Excluded middle}]$$

Defining Proof Systems

Formal inference rules

Inference rule

It consists in a finite set of **premisses** $\{\psi_1, \ldots, \psi_n\}$ and a **conclusion** φ, where φ is meant to be a **consequence** of $\psi_1, \ldots, \psi_n$. It is written:

$$\frac{\psi_1 \quad \ldots \quad \psi_n}{\varphi}\ [r]$$

E Consider the *modus ponens*:

$$\frac{A \Rightarrow B \qquad A}{B}\ [\textit{Modus ponens}]$$

Defining Proof Systems
A formal definition

Hilbert proof system

It consists in a **finite set** of axioms and a **(possibly infinite) set** of inference rules.

Note that the rules and axiom may not necessarily make sense.

A proof system could feature an axiom:

$$\frac{}{A \Rightarrow \neg A}\ [\textit{Absurd axiom}]$$

Writing Proofs
The use of inference rules

- Consider the rule:

$$\frac{A \qquad B}{A \wedge B}\ [\wedge]$$

- A and B are generic variables that can be **substituted**.
- As an example, if $A \leftarrow \neg X$ and $B \leftarrow (X \vee Y)$, then we can consider the **deduction**:

$$\frac{\neg X \qquad X \vee Y}{\neg X \wedge (X \vee Y)}\ [\wedge]$$

Writing Proofs
Formalizing deductions

Deduction under a Hilbert proof system $\mathcal{P}$

It's a tree T whose nodes are **labelled by propositional formulas** in such a manner that each inner node is the **conclusion** of some inference rule of $\mathcal{P}$ after applying a **substitution**, and its children the **premisses**.

Writing Proofs
An example I

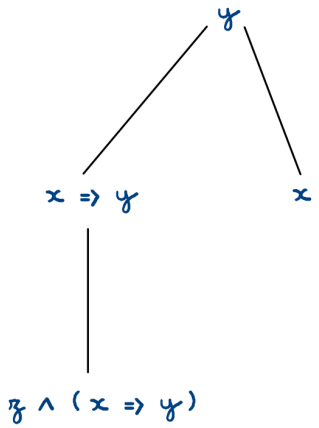
Consider a proof system $\mathcal{P}$ with the two following rules:

$$\frac{A \wedge B}{B}\ [\wedge]$$

$$\frac{A \Rightarrow B \qquad A}{B}\ [\Rightarrow]$$

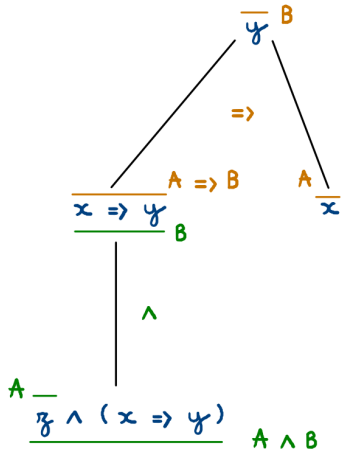
Writing Proofs
An example II

Then the following tree is a **deduction**:



Writing Proofs
An example III

Note that each node can use a different substitution.



Writing Proofs
An example IV

We favour the following notation:

$$\dfrac{\dfrac{z \wedge (x \Rightarrow y)}{x \Rightarrow y}\,[\wedge] \qquad x}{y}\,[\Rightarrow]$$

We then write $\{z \wedge (x \Rightarrow y), x\} \vdash_{\mathcal{P}} y$.

Writing Proofs
Some vocabulary

Hypotheses and conclusions

The labels of a deduction T's leaves are called its **hypotheses**, and the label of its root, its **conclusion**.

The previous example has hypotheses $\{z \wedge (x \Rightarrow y), x\}$ and conclusion y.

Proofs and Theorems

Applying axioms

Cancelling leaves

A leaf φ of a deduction T under $\mathcal{P}$ can be **cancelled** if φ can be matched to a an axiom a of $\mathcal{P}$ after applying a substitution. We then write in the tree:

$$\frac{}{\varphi}\,[a]$$

If T has uncancelled hypotheses $\{H_1, \ldots, H_n\}$ and conclusion C, we then write $\{H_1, \ldots, H_n\} \vdash_{\mathcal{P}} C$.

Proofs and Theorems

Cancelling an example

- Let $\mathcal{P}$ be a proof system with a rule $\dfrac{A \qquad B}{A \Rightarrow B}\,[r]$ and an axiom $\dfrac{}{A \vee \neg A}\,[a]$.
- Then the following tree is a deduction under $\mathcal{P}$:

$$\frac{\dfrac{}{A \vee \neg A}\,[a] \qquad A}{(A \vee \neg A) \Rightarrow A}\,[r]$$

- The leaf $A \vee \neg A$ is cancelled, thus $\{A\} \vdash_{\mathcal{P}} (A \vee \neg A) \Rightarrow A$.

Proofs and Theorems

A proper proof

Proof under a Hilbert system $\mathcal{P}$

It is a deduction T under $\mathcal{P}$ whose leaves **are all cancelled**. Its conclusion C is then called a **theorem** of $\mathcal{P}$ and we write $\vdash_{\mathcal{P}} C$.

Intuitively, a theorem is **what can be deduced from the axioms**.

Proofs and Theorems

Proving a somewhat spurious example

- Let $\mathcal{P}$ be a system with two rules $\dfrac{A \vee B}{B}\,[r_1]$ and $\dfrac{B}{A \Rightarrow B}\,[r_2]$ and a single axiom $\dfrac{}{A \vee \neg A}\,[a]$.
- Then the following tree is a proof:

$$\cfrac{\cfrac{\cfrac{}{P \vee \neg P}\,[a]}{\neg P}\,[r_1]}{P \Rightarrow \neg P}\,[r_2]$$

- Note that **a proof may not make sense w.r.t. the 'usual' logic**.

Exercise 1. Consider the following Hilbert proof system $\mathcal{P}$:

$$\frac{}{\top}\,[\top] \qquad \frac{A \Leftrightarrow B}{B \Leftrightarrow A}\,[\Leftrightarrow]$$

$$\frac{A \Rightarrow B \qquad A}{B}\,[\Rightarrow_E] \qquad \frac{A \Leftrightarrow B}{A \Rightarrow B}\,[\Rightarrow_I]$$

Prove that $X \Leftrightarrow \top \vdash_{\mathcal{P}} X$.

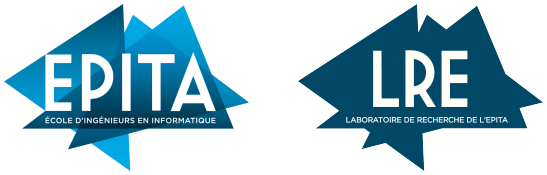
Consider the following deduction tree for $X \Leftrightarrow \top \vdash_{\mathcal{P}} X$:

$$\dfrac{\dfrac{\dfrac{X \Leftrightarrow \top}{\top \Leftrightarrow X}\,[\Leftrightarrow]}{\top \Rightarrow X}\,[\Rightarrow_I] \qquad \dfrac{}{\top}\,[\top]}{X}\,[\Rightarrow_E]$$

Formal Logics

Hilbert Calculus

Adrien Pommellet



June 23, 2025

The canonical Hilbert calculus $\mathcal{H}$

It is a Hilbert proof system $\mathcal{H}$ containing a **single** inference rule:

$$\frac{A \Rightarrow B \qquad A}{B}\,[\text{Modus Ponens}]$$

As well as **twelve** axioms.

The canonical Hilbert calculus $\mathcal{H}$ (cont.)

$\mathcal{H}$ features the following **five** axioms:

$$\frac{}{A \Rightarrow A \vee B}\,[\vee_1] \qquad \frac{}{B \Rightarrow A \vee B}\,[\vee_2]$$

$$\frac{}{A \wedge B \Rightarrow A}\,[\wedge_1] \qquad \frac{}{A \wedge B \Rightarrow B}\,[\wedge_2]$$

$$\frac{}{\bot \Rightarrow A}\,[\bot]$$

The canonical Hilbert calculus $\mathcal{H}$ (cont.)

As well as the following **four** axioms:

$$\frac{}{A \vee \neg A}\,[\textit{Excluded Middle}] \qquad \frac{}{A \Rightarrow (B \Rightarrow A)}\,[\Rightarrow_1]$$

$$\frac{}{(A \Rightarrow \bot) \Rightarrow \neg A}\,[\neg_1] \qquad \frac{}{A \Rightarrow (\neg A \Rightarrow \bot)}\,[\neg_2]$$

The canonical Hilbert calculus $\mathcal{H}$ (cont.)

And finally, the last **three** axioms:

$$\frac{}{A \vee B \Rightarrow ((A \Rightarrow C) \Rightarrow ((B \Rightarrow C) \Rightarrow C))}\,[\vee_3]$$

$$\frac{}{A \Rightarrow (B \Rightarrow A \wedge B)}\,[\wedge_3]$$

$$\frac{}{(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))}\,[\Rightarrow_2]$$

Note that $\mathcal{H}$ can't handle the $\top$ and $\Leftrightarrow$ symbols.

We don't know yet
if this system 'makes sense'.

Applying the Hilbert Calculus

A not-so-easy example I

E Prove that $\vdash_{\mathcal{H}} (X \Rightarrow X)$.

Applying the Hilbert Calculus

The golden rule of proof systems

- We can **only** use the inference rules and axioms of the proof system.
- We need to prove that $(X \Rightarrow X)$ is a theorem, but we **cannot** apply the material implication $(X \Rightarrow X) \equiv (\neg X \vee X)$ in order to prove $(\neg X \vee X)$ instead.
- Don't spoil **syntactic proofs** with **semantic properties**.

Applying the Hilbert Calculus

A not-so-easy example II

We **omit the label** of the only inference rule *Modus Ponens*.

$$\dfrac{\dfrac{\dfrac{}{(X \Rightarrow (X \Rightarrow X) \Rightarrow X) \Rightarrow (X \Rightarrow X \Rightarrow X) \Rightarrow X \Rightarrow X}\,[\Rightarrow_2] \quad \dfrac{}{X \Rightarrow (X \Rightarrow X) \Rightarrow X}\,[\Rightarrow_1]}{(X \Rightarrow X \Rightarrow X) \Rightarrow X \Rightarrow X} \quad \dfrac{}{X \Rightarrow X \Rightarrow X}\,[\Rightarrow_1]}{X \Rightarrow X}$$

Practical Application

Exercise 1. Prove that $P \vdash_{\mathcal{H}} \neg\neg P$ by filling in the blanks of the following deduction tree:

$$\dfrac{\dfrac{}{(\neg P \Rightarrow \bot) \Rightarrow \neg\neg P} \quad \dfrac{\dfrac{}{\qquad}}{P}}{\neg\neg P}$$

Answer

$$\dfrac{\dfrac{}{(\neg P \Rightarrow \bot) \Rightarrow \neg\neg P}\,[\neg_1] \qquad \dfrac{\dfrac{}{P \Rightarrow \neg P \Rightarrow \bot}\,[\neg_2] \qquad P}{\neg P \Rightarrow \bot}}{\neg\neg P}$$

Practical Application

Exercise 2. Prove that $\{P \Rightarrow Q, Q \Rightarrow R\} \vdash_{\mathcal{H}} P \Rightarrow R$ by filling in the blanks of the following deduction tree:

$$\dfrac{\dfrac{}{(P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R}\,[\Rightarrow_2] \qquad \dfrac{}{(Q \Rightarrow R) \Rightarrow P \Rightarrow (Q \Rightarrow R)}}{\dfrac{}{P \Rightarrow R}}$$

Answer

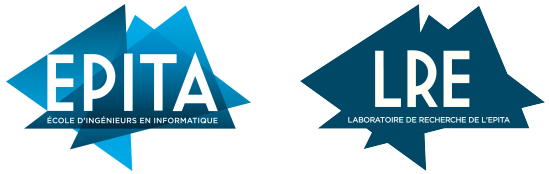
$$\dfrac{\dfrac{\dfrac{}{(P \Rightarrow Q \Rightarrow R) \Rightarrow (P \Rightarrow Q) \Rightarrow P \Rightarrow R}\,[\Rightarrow_2] \qquad \dfrac{}{(Q \Rightarrow R) \Rightarrow P \Rightarrow (Q \Rightarrow R)}\,[\Rightarrow_1] \qquad Q \Rightarrow R}{P \Rightarrow (Q \Rightarrow R)}}{\dfrac{(P \Rightarrow Q) \Rightarrow P \Rightarrow R \qquad P \Rightarrow Q}{P \Rightarrow R}}$$

This is painful.
Why are we doing this?

Formal Logics

Proof Systems and Semantics

Adrien Pommellet



June 23, 2025

Semantics and Syntax

Two different frameworks

Semantics

The truth is **tautological** and proved using **truth tables**.

x_1	x_2	$\dots$	φ
0	0		1
0	1	$\dots$	1
$\dots$	$\dots$		$\dots$

The symbols $\vee, \wedge, \Rightarrow, \neg$ and $\bot$ are **individually interpreted**.

Syntax

The truth consists in **theorems** resulting from **deductions**.

$$\dfrac{\dfrac{}{\mu_1}\ \dfrac{}{\dots}\ \psi_1 \quad \dots \quad \dfrac{}{\mu_n}\ \dfrac{}{\dots}\ \psi_n}{\varphi}$$

The symbols have **no intrinsic meaning** but the one inference rules provide.

Semantics and Syntax

The duality of truth

Are these two notions equivalent?

Practical Application

Exercise 1. Using a Hilbert proof system $\mathcal{P}$:

$$\dfrac{}{\top}\,[\top] \qquad \dfrac{}{\bot \Rightarrow A}\,[\bot]$$

$$\dfrac{A \Leftrightarrow B \qquad A}{B}\,[\Leftrightarrow_E] \qquad \dfrac{A \Rightarrow B \qquad B \Rightarrow A}{A \Leftrightarrow B}\,[\Leftrightarrow_I] \qquad \dfrac{A \Rightarrow B}{B \Rightarrow A}\,[\Rightarrow]$$

Prove that $\vdash_{\mathcal{P}} \top \Leftrightarrow \bot$, then use this result to prove that $\vdash_{\mathcal{P}} \bot$.

Answer I

Consider the following proof tree for $\vdash_{\mathcal{P}} \top \Leftrightarrow \bot$:

$$\dfrac{\dfrac{\dfrac{}{\bot \Rightarrow \top}\,[\bot]}{\top \Rightarrow \bot}\,[\Rightarrow] \qquad \dfrac{}{\bot \Rightarrow \top}\,[\bot]}{\top \Leftrightarrow \bot}\,[\Leftrightarrow_I]$$

Answer II

From this tree, we can design a proof for $\vdash_{\mathcal{P}} \bot$:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\bot \Rightarrow \top}\,[\bot]}{\top \Rightarrow \bot}\,[\Rightarrow] \qquad \dfrac{}{\bot \Rightarrow \top}\,[\bot]}{\top \Leftrightarrow \bot}\,[\Leftrightarrow_I] \qquad \dfrac{}{\top}\,[\top]}{\bot}\,[\Leftrightarrow_E]$$

Of Soundness

A definition

Soundness

A proof system $\mathcal{P}$ is **sound** if any theorem of $\mathcal{P}$ is a tautology.

Intuitively, any formula that can be proved in $\mathcal{P}$ must always be true w.r.t. Tarski's semantics.

The previous system $\mathcal{P}$ is **not sound**.

Of Soundness

Counter-examples

- Some proof systems may not be sound, as the deduction process is **purely syntactic**.
- An **axiom** may not be tautological.

$$\overline{(A \Rightarrow B) \Rightarrow (\neg A \Rightarrow \neg B)}$$

- An inference rule may not preserve truth, i.e. its **conclusion** may not be semantically true even when its **premisses** are.

$$\frac{A \vee B}{A}$$

Of Soundness

What about Hilbert calculus?

Soundness of $\mathcal{H}$

The Hilbert calculus $\mathcal{H}$ is **sound**.

The proof sketch is the following:

1. Prove that the **axioms** are tautologies (use truth tables).
2. Prove that **Modus Ponens** preserves truth (again, use truth tables).
3. Prove that each theorem is a tautology by **induction** on the tree structure of proofs.

Of Completeness

The definition

Completeness

A proof system $\mathcal{P}$ is **complete** if any tautology is a theorem of $\mathcal{P}$.

Intuitively, any tautology can be proven syntactically.

Of Completeness

What about Hilbert calculus?

Completeness of $\mathcal{H}$

The Hilbert calculus $\mathcal{H}$ is **complete**.

We will admit the complex proof of this property.

Of Completeness

A counter-example

The intuitionistic Hilbert calculus $\mathcal{I} = \mathcal{H} - \{\textit{Excluded Middle}\}$ is **sound but not complete**.

Neither $(A \vee \neg A)$ nor $(\neg\neg A \Rightarrow A)$ are theorems of the system $\mathcal{I}$.

Of Completeness
An useful theorem

Herbrand's theorem

$\{H_1, \ldots, H_n\} \vdash_{\mathcal{H}} C$ if and only if $\{H_1, \ldots, H_{n-1}\} \vdash_{\mathcal{H}} H_n \Rightarrow C$.

It ties the logical implication $\Rightarrow$ and the inference relation of $\mathcal{H}$ together, as a consequence of this theorem is $\vdash_{\mathcal{H}} H_1 \Rightarrow \ldots \Rightarrow H_{n-1} \Rightarrow H_n \Rightarrow C$.

Of Completeness
Proving Herbrand's theorem I

Let us suppose $\{H_1, \ldots, H_{n-1}\} \vdash_{\mathcal{H}} H_n \Rightarrow C$.

$$\frac{\begin{array}{ccc} H_1 & & H_{n-1} \\ \vdots & \ldots & \vdots \end{array}}{H_n \Rightarrow C}$$

Then $\{H_1, \ldots, H_n\} \vdash_{\mathcal{H}} C$.

$$\frac{\dfrac{\begin{array}{ccc} H_1 & & H_{n-1} \\ \vdots & \ldots & \vdots \end{array}}{H_n \Rightarrow C} \qquad H_n}{C}\ \text{[Modus Ponens]}$$

Of Completeness
Proving Herbrand's theorem II

We admit that $\{H_1, \ldots, H_n\} \vdash_{\mathcal{H}} C$ implies $\{H_1, \ldots, H_{n-1}\} \vdash_{\mathcal{H}} H_n \Rightarrow C$.

Optional Homework

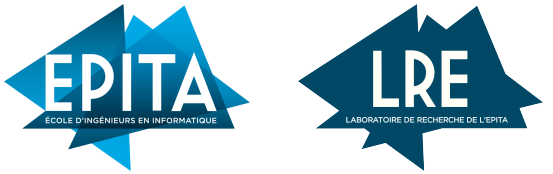
- Write step 3 of the soundness proof.
- Exercises **2A** and **2B** of the 2019-2020 exam.

See you next class!

Formal Logics

Proof Systems with Hypotheses

Adrien Pommellet



June 23, 2025

The Flaws of Hilbert Calculus

A boring proof system

The Hilbert calculus $\mathcal{H}$ is sound, complete, and incredibly unwieldy.

The Flaws of Hilbert Calculus

Too many axioms

- It is **easier** to use inference rules than axioms: $\dfrac{A}{A \vee B}\,[\vee]$ instead of $\dfrac{}{A \vee B \Rightarrow (A \Rightarrow C) \Rightarrow (B \Rightarrow C) \Rightarrow C}\,[\vee]$.
- However, how can we insert $\Rightarrow$ in a sound and complete manner?

$$\frac{?}{A \Rightarrow B}\,[\Rightarrow]$$

- The Hilbert calculus handles this problem by **never inserting new symbols** and **using complex axioms** that we simplify.

The Flaws of Hilbert Calculus

Implying implications

We would like to write something like:

$$\dfrac{\begin{array}{c}\textit{If from } A\\ \vdots\\ \textit{we can deduce } B\end{array}}{\textit{Then } A \Rightarrow B.}\ [\Rightarrow]$$

But this is **not possible** with the current definition of proof systems that can only handle **direct consequences**.

Proof Systems with Hypotheses

A new type of inference rule

Inference rule with hypotheses

It consists in a finite set of **premisses** $\{\psi_1, \ldots, \psi_n\}$, a finite set of **hypotheses** $\{\mu_1, \ldots, \mu_n\}$ and a **conclusion** φ. Intuitively, it means that if for all i, ψ_i is true or can be **inferred from** μ_i, then φ holds. It is written:

$$\dfrac{\begin{array}{ccc}[\mu_1] & & [\mu_n]\\ \vdots & & \vdots\\ \psi_1 & \cdots & \psi_n\end{array}}{\varphi}\ [r]$$

Proof Systems with Hypotheses

An intuition

- Consider the rule $\dfrac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A \Rightarrow B}\ [\Rightarrow]$ that inserts a $\Rightarrow$ symbol.
- It intuitively means that if from A we can deduce B (no matter how long the deduction), then $A \Rightarrow B$ is true.
- These rules therefore allow us to reason on **multi-step** deductions.

Proof Systems with Hypotheses

A stronger formalism

- We may still need to write **direct deductions** such as $\dfrac{A}{A \vee B}$.
- We thus allow **empty hypotheses**, writing ψ_i instead of $\dfrac{\begin{array}{c}[\emptyset]\\ \vdots\\ \psi_i\end{array}}{}$.
- If all the hypotheses of a rule are empty, we end up with a **classical Hilbert inference rule** of the form:

$$\dfrac{\psi_1 \quad \cdots \quad \psi_n}{\varphi}\ [r]$$

Proof Systems with Hypotheses

A new type of proof system

Proof system with hypotheses

It is a **(possibly infinite) set** of inference rules with hypotheses.

Note that there are **no axioms**. We will see why.

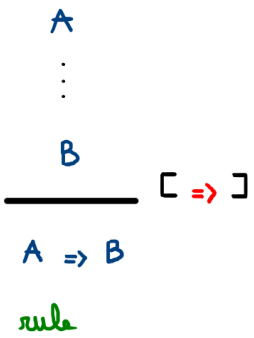
Writing Proofs

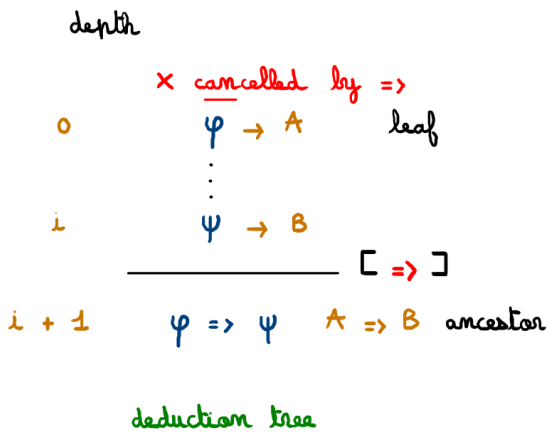
Farewell, axioms!

- If there are **no axioms**, how can we **cancel the leaves** of a deduction in order to write a proof?
- By cancelling the **hypotheses** of an inference rule whenever it is applied in a deduction tree.
- Intuitively, the cancelled leaves have been properly dealt with by being **integrated** into the conclusion of the inference rule.

Writing Proofs

It's intuitive, I swear.





Writing Proofs

An example I

- Consider a proof system with two rules $\dfrac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B}\,[\Rightarrow]$ and $\dfrac{A}{A \vee B}\,[\vee]$.
- We can write the following deduction tree:

$$\dfrac{\dfrac{X}{X \vee Y}\,[\vee]}{X \Rightarrow X \vee Y}\,[\Rightarrow]$$

- How can we cancel the leaf X of this deduction?

Writing Proofs
An example II

- If $A \leftarrow X$ and $B \leftarrow X \vee Y$, then we consider the inference rule:

$$\dfrac{\begin{array}{c} X \\ \vdots \\ X \vee Y \end{array}}{X \Rightarrow X \vee Y}\,[\Rightarrow]$$

- We can cancel leaf X.

$$\dfrac{\dfrac{\dfrac{}{X}\,1}{X \vee Y}\,[\vee]}{X \Rightarrow X \vee Y}\,[\Rightarrow]^1$$

- We **label** with integers the inference rule and its matching hypotheses.

Writing Proofs
Help, I'm being cancelled!

Note that a premiss of an inference rule and its matching hypothesis may be **matched to the very same leaf**.

$$\dfrac{\dfrac{}{A}\,1}{A \Rightarrow A}\,[\Rightarrow]^1$$

Moreover, **different rules** may cancel the same leaf.

$$\dfrac{\dfrac{\dfrac{}{A}\,1,2}{A \Rightarrow A}\,[\Rightarrow]^1}{A \Rightarrow A \Rightarrow A}\,[\Rightarrow]^2$$

Writing Proofs
Optional premisses

- Can we use $\dfrac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \Rightarrow B}\,[\Rightarrow]$ to deduce $A \Rightarrow B$ from B?
- Actually, we can treat $\dfrac{\begin{array}{c}[A] \\ \vdots \\ B\end{array}}{A \Rightarrow B}\,[\Rightarrow]$ as if it were $\dfrac{B}{A \Rightarrow B}\,[\Rightarrow]$.
- Premisses of inference rules **can be ignored** while writing deductions.
- But it can lead to **uncancelled leaves** such as B in $\dfrac{B}{A \Rightarrow B}\,[\Rightarrow]$.

Proof Systems with Hypotheses
Formalizing deductions

Deduction under a proof system with hypotheses $\mathcal{P}$

It's a tree T whose nodes are **labelled by propositional formulas** in such a manner that each inner node is the **conclusion** of some inference rule of $\mathcal{P}$ after applying a **substitution**, and its children the **premisses**.

Hypotheses and conclusions

The labels of a deduction T's leaves are called its **hypotheses**, and the label of its root, its **conclusion**.

Writing Proofs
Formalizing proofs again

Proof under a proof system $\mathcal{P}$ with hypotheses

It is a deduction T under $\mathcal{P}$ whose leaves **are all cancelled**. Its conclusion C is then called a **theorem** of $\mathcal{P}$ and we write $\vdash_{\mathcal{P}} C$.

If T has uncancelled hypotheses $\{H_1, \ldots, H_n\}$ and conclusion C, we then write $\{H_1, \ldots, H_n\} \vdash_{\mathcal{P}} C$.

Writing Proofs
Another example

- Consider a system with two rules $\dfrac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B}\,[\Rightarrow]$ and $\dfrac{A \quad B}{A \wedge B}\,[\wedge]$.
- The following tree is a proof:

$$\dfrac{\dfrac{\dfrac{\overline{X}^{\,1} \quad \overline{Y}^{\,2}}{X \wedge Y}\,[\wedge]}{X \Rightarrow X \wedge Y}\,[\Rightarrow]^1}{Y \Rightarrow X \Rightarrow X \wedge Y}\,[\Rightarrow]^2$$

- $Y \Rightarrow X \Rightarrow X \wedge Y$ is a **theorem** of that system.

Practical Application

Exercise 1. Prove that $Y \Rightarrow X \Rightarrow Y$ is a theorem of the previous system.

Answer

The following deduction tree is a **proof**:

$$\dfrac{\dfrac{\overline{Y}^{\,1}}{X \Rightarrow Y}\,[\Rightarrow]}{Y \Rightarrow X \Rightarrow Y}\,[\Rightarrow]^1$$

Note that we deduced $X \Rightarrow Y$ from Y without cancelling a leaf.

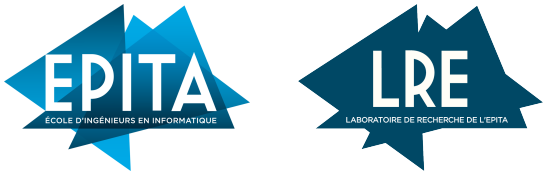
A Brief Summary

- We add **hypotheses** to inference rules.
- Proof systems with hypotheses **no longer use axioms**.
- Deductions **remain unaltered**. We do not make use of these new hypotheses **yet**.
- Leaves are no longer cancelled by axioms but are instead **matched to hypotheses** of inference rules applied during the deduction process.
- A proof is still a deduction **whose leaves are all cancelled**.

Formal Logics

Natural Deduction

Adrien Pommellet



June 23, 2025

Introducing Natural Deduction

A new proof system

Natural deduction $\mathcal{N}$

It is a proof system with hypotheses $\mathcal{N}$ on $\mathcal{F}_{\{\bot,\neg,\wedge,\vee,\Rightarrow\}}$.

For each symbol in $\{\bot, \neg, \wedge, \vee, \Rightarrow\}$, $\mathcal{N}$ features **introduction** and **elimination** rules: the former introduce new symbols, the latter remove them.

Introducing Natural Deduction

The rules I

Natural deduction $\mathcal{N}$ (cont.)

The following rules are matched to the symbol $\Rightarrow$:

$$\dfrac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B}\,[\Rightarrow_I] \qquad \dfrac{A \Rightarrow B \qquad A}{B}\,[\Rightarrow_E]$$

Natural deduction $\mathcal{N}$ (cont.)

The following rules are matched to the symbols $\vee$ and $\wedge$:

$$\frac{A \quad B}{A \wedge B}\,[\wedge_I] \qquad \frac{A \wedge B}{A}\,[\wedge^l_E] \quad \frac{A \wedge B}{B}\,[\wedge^r_E]$$

$$\frac{A}{A \vee B}\,[\vee^l_I] \quad \frac{B}{A \vee B}\,[\vee^r_I] \qquad \frac{A \vee B \quad \begin{array}{c}[A]\\ \vdots\\ C\end{array} \quad \begin{array}{c}[B]\\ \vdots\\ C\end{array}}{C}\,[\vee_E]$$

Natural deduction $\mathcal{N}$ (cont.)

The following rules are matched to the symbols $\neg$ and $\bot$:

$$\frac{\begin{array}{c}[A]\\ \vdots\\ \bot\end{array}}{\neg A}\,[\neg_I] \qquad \frac{A \quad \neg A}{\bot}\,[\neg_E]$$

$$\frac{\neg\neg A}{A}\,[\neg\neg] \qquad \frac{\bot}{A}\,[\bot_E]$$

E Prove that $\vdash_{\mathcal{N}} (X \wedge Y) \Rightarrow (Y \wedge X)$.

The following deduction tree is a **proof**:

$$\dfrac{\dfrac{\dfrac{\overline{X \wedge Y}^{\,1}}{Y}\,[\wedge^r_E] \quad \dfrac{\overline{X \wedge Y}^{\,1}}{X}\,[\wedge^l_E]}{Y \wedge X}\,[\wedge_I]}{(X \wedge Y) \Rightarrow (Y \wedge X)}\,[\Rightarrow_I]^1$$

Practical Application

Exercise 1. Prove that $\vdash_{\mathcal{N}} (X \Rightarrow Y) \wedge (Y \Rightarrow Z) \Rightarrow (X \Rightarrow Z)$.

Applying Natural Deduction

Answer

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{(X \Rightarrow Y) \wedge (Y \Rightarrow Z)}{Y \Rightarrow Z}\,[\wedge_E^r]}{Y}\,[\Rightarrow_E] \quad \dfrac{\dfrac{(X \Rightarrow Y) \wedge (Y \Rightarrow Z)}{X \Rightarrow Y}\,[\wedge_E^l] \quad \dfrac{}{X}^{2}}{Y}\,[\Rightarrow_E]}{Z}\,[\Rightarrow_I]^2}{X \Rightarrow Z}\,[\Rightarrow_I]^1}{(X \Rightarrow Y) \wedge (Y \Rightarrow Z) \Rightarrow (X \Rightarrow Z)}\,[\Rightarrow_I]^1$$

Practical Application

Exercise 2. Prove that $\{A \Rightarrow B, A \Rightarrow C\} \vdash_{\mathcal{N}} A \Rightarrow B \wedge C$.

Answer

The following deduction tree is a **proof**:

$$\dfrac{\dfrac{\dfrac{A \Rightarrow B \quad \dfrac{}{A}^{1}}{B}\,[\Rightarrow_E] \quad \dfrac{A \Rightarrow C \quad \dfrac{}{A}^{1}}{C}\,[\Rightarrow_E]}{B \wedge C}\,[\wedge_I]}{A \Rightarrow B \wedge C}\,[\Rightarrow_I]^1$$

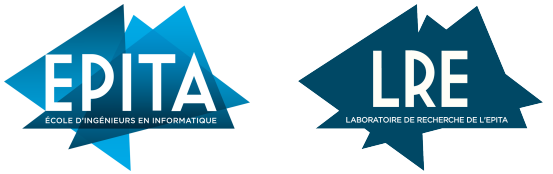
Optional Homework

- Exercises **3A** and **3B** of the 2019-2020 exam.

Formal Logics

Properties of Natural Deduction

Adrien Pommellet



June 23, 2025

Natural Deduction

A quick reminder I

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B}\,[\Rightarrow_I] \qquad \frac{A \Rightarrow B \qquad A}{B}\,[\Rightarrow_E]$$

$$\frac{A \qquad B}{A \wedge B}\,[\wedge_I] \qquad \frac{A \wedge B}{A}\,[\wedge_E^l] \qquad \frac{A \wedge B}{B}\,[\wedge_E^r]$$

$$\frac{A}{A \vee B}\,[\vee_I^l] \qquad \frac{B}{A \vee B}\,[\vee_I^r] \qquad \frac{A \vee B \qquad \begin{array}{c}[A]\\ \vdots \\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots \\ C\end{array}}{C}\,[\vee_E]$$

Natural Deduction

A quick reminder II

$$\frac{\begin{array}{c}[A]\\ \vdots \\ \bot\end{array}}{\neg A}\,[\neg_I] \qquad \frac{A \qquad \neg A}{\bot}\,[\neg_E]$$

$$\frac{\neg\neg A}{A}\,[\neg\neg] \qquad \frac{\bot}{A}\,[\bot_E]$$

Natural Deduction

A last example I

E Prove that $\vdash_{\mathcal{N}} X \vee \neg X$.

Natural Deduction

A last example II

We first try to intuit an **informal proof**:

1. By **contradiction**, assume that $\neg(X \vee \neg X)$ holds.
2. Then assume by **contradiction** that X holds.
   - Then $X \vee \neg X$ holds.
   - But it **contradicts** $\neg(X \vee \neg X)$.
3. Thus $\neg X$ holds.
4. Then $X \vee \neg X$ holds.
5. But it **contradicts** $\neg(X \vee \neg X)$ (again).
6. Finally, $X \vee \neg X$ holds.

Natural Deduction

A last example III

This intuition yields the following deduction tree that is a **proof**:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{X}\,2}{X \vee \neg X}\,[\vee_I^l] \quad \dfrac{}{\neg(X \vee \neg X)}\,1}{\bot}\,[\neg_E]}{\neg X}\,[\neg_I]^2}{X \vee \neg X}\,[\vee_I^r] \quad \dfrac{}{\neg(X \vee \neg X)}\,1}{\bot}\,[\neg_E]}{\neg\neg(X \vee \neg X)}\,[\neg_I]^1}{X \vee \neg X}\,[\neg\neg]$$

Practical Application

Exercise 1. Prove that $\{A \vee B, \neg B\} \vdash_{\mathcal{N}} A$ by filling in the blanks of the following deduction tree:

$$\dfrac{\_\_ \qquad \dfrac{\dfrac{\_\_}{\bot}}{A}}{}\,[\vee_E]$$

Answer I

We try again to intuit an **informal proof**:

1. Consider a **case disjunction** on $A \vee B$.
2. Assume that A holds; then A holds (surprising, isn't it?).
3. Assume that B holds.
   - But $\neg B$ is another hypothesis.
   - Thus there is a **contradiction**: this case cannot happen.
4. We solved both cases.

Answer II

$$\dfrac{A \vee B \qquad \dfrac{}{A}\,1 \qquad \dfrac{\dfrac{\dfrac{}{B}\,1 \qquad \neg B}{\bot}\,[\neg_E]}{A}\,[\bot_E]}{A}\,[\vee_E]^1$$

Practical Application

Exercise 2. Prove that $\{\neg A \vee B\} \vdash_{\mathcal{N}} A \Rightarrow B$ by filling in the blanks of the following deduction tree:

$$\dfrac{\dfrac{\dfrac{\quad \quad}{\quad}}{\dfrac{\bot}{B}} \qquad \dfrac{\quad}{\quad}}{\quad}\,[\vee_E]$$

Answer I

We try again to intuit an **informal proof**:

1. We want to prove $A \Rightarrow B$, thus assume that A holds.
2. Consider a **case disjunction** on hypothesis $\neg A \vee B$.
3. Assume that $\neg A$ holds.
   - We assumed previously that A held.
   - Thus there is a **contradiction**: this case cannot happen.
4. Assume that B holds; then B holds (to everyone's amazement).
5. B holds in both cases.

Answer II

$$\dfrac{\neg A \vee B \qquad \dfrac{\dfrac{\dfrac{}{A}\,1 \qquad \dfrac{}{\neg A}\,2}{\dfrac{\bot}{B}\,[\bot_E]}\,[\neg_E] \qquad \dfrac{}{B}\,2}{B}\,[\vee_E]^2}{A \Rightarrow B}\,[\Rightarrow_I]^1$$

Properties of Natural Deduction

Soundness and Completeness

$\mathcal{N}$ is compatible with Tarski's semantics

Natural deduction $\mathcal{N}$ is **sound** and **complete** w.r.t. the semantics of $\mathcal{F}_{\{\bot,\neg,\wedge,\vee,\Rightarrow\}}$.

We will admit this property.

Properties of Natural Deduction

Of cuts

Cut

It is the **introduction** in a deduction of a connective immediately followed by its **elimination**.

Consider the following deduction featuring a cut:

$$\dfrac{\dfrac{A \qquad B}{A \wedge B}\,[\wedge_I]}{A}\,[\wedge_E']$$

Properties of Natural Deduction

Normalization

Normalization

Given a deduction on $\mathcal{N}$, there exists another deduction **without cuts** sharing the same conclusion and a subset of the original hypotheses.

Intuitively, **normalized** deductions are somewhat **monotonic**: if possible, the premisses should be simpler than their conclusion.

Properties of Natural Deduction

Normalization patterns I

$$\dfrac{\dfrac{A \quad B}{A \wedge B}\,[\wedge_I]}{A}\,[\wedge^l_E] \quad \rightsquigarrow \quad A$$

$$\dfrac{\dfrac{A \quad B}{A \wedge B}\,[\wedge_I]}{B}\,[\wedge^r_E] \quad \rightsquigarrow \quad B$$

Properties of Natural Deduction

Normalization patterns II

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B}\,[\Rightarrow_I] \qquad A}{B}\,[\Rightarrow_E] \quad \rightsquigarrow \quad \begin{array}{c}[A]\\ \vdots \\ B\end{array}$$

Properties of Natural Deduction

Normalization patterns III

$$\dfrac{\dfrac{A}{A \vee B}\,[\vee^l_I] \qquad \begin{array}{c}[A]\\ \vdots \\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots \\ C\end{array}}{C}\,[\vee_E] \quad \rightsquigarrow \quad \begin{array}{c}[A]\\ \vdots \\ C\end{array}$$

$$\dfrac{\dfrac{B}{A \vee B}\,[\vee^r_I] \qquad \begin{array}{c}[A]\\ \vdots \\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots \\ C\end{array}}{C}\,[\vee_E] \quad \rightsquigarrow \quad \begin{array}{c}[B]\\ \vdots \\ C\end{array}$$

Properties of Natural Deduction

Normalization patterns IV

$$\dfrac{A \qquad \dfrac{\begin{array}{c}[A]\\ \vdots \\ \bot\end{array}}{\neg A}\,[\neg_I]}{\bot}\,[\neg_E] \quad \rightsquigarrow \quad \begin{array}{c}[A]\\ \vdots \\ \bot\end{array}$$

Optional Homework

- Exercises **4A** and **4B** of the 2019-2020 exam.

Mid-term Exam

- A **mid-term** exam.
- On the **Moodle Exam** server.
- It will be **30 minutes** long.
- Expect truth tables, Hilbert calculus and natural deduction.
- A **mock exam** will be uploaded beforehand.

See you next class!

Formal Logics

Applying Logics

Adrien Pommellet

June 23, 2025

A Short Guide to Proofs

Using natural deduction $\mathcal{N}$

- Write a proof tree whose **root** is the **theorem** we want to prove.
- From the root, look at the **latest symbol inserted** and apply the matching insertion rule accordingly.
- Guess the leaves by looking at the **hypotheses we can cancel**.
- Two leaves **may share the same label** and may be cancelled by the same rule and the same node.
- **Intuit** the most complex proofs in everyday language first.
- Keep in mind that **only three rules** usually handle the $\bot$ symbol.

The Use of Proof Systems

Evaluating complexity

- Prove that $X_1 \Rightarrow X_2 \Rightarrow \ldots \Rightarrow X_n \Rightarrow X_1 \wedge X_2 \wedge \ldots \wedge X_n$ is 'true'.
- With **Tarski's semantics**, 2^n valuations to check.
- With **natural deduction**, a proof of depth $\leq 2n$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{X_1}\,1 \quad \dfrac{}{X_2}\,2}{X_1 \wedge X_2}\,[\wedge_I]}{X_2 \Rightarrow (X_1 \wedge X_2)}\,[\Rightarrow_I]^2}{X_1 \Rightarrow X_2 \Rightarrow (X_1 \wedge X_2)}\,[\Rightarrow_I]^1$$

The Use of Proof Systems

Practical applications

- Proving theorems **automatically**.
- Reducing various constraint-solving and optimisation problems to a **satisfiability test** (`SAT` solvers).
- Verify that a given set of **specifications** is sensible.

The Use of Proof Systems

Certified programs

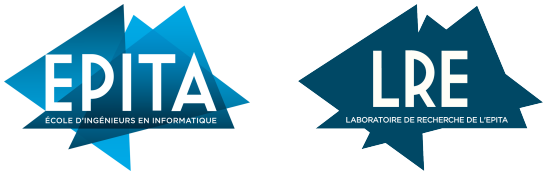
Synthesizing certified code using **CoQ** or **L'atelier B**.

And now for something entirely different. . .

Formal Logics

Lambda Calculus

Adrien Pommellet



June 23, 2025

A Model for Functional Programming

Two different frameworks

Imperative programming

- Memory states.
- Manipulating structures and classes with statements.
- Side effects.
- **Turing machines**.

Functional programming

- Transforming data.
- Manipulating functions.
- Functions are data.
- **Lambda calculus**.

A Model for Functional Programming

Properties of functional languages I

Functional languages such as `OCaml` can take functions as **arguments**:

```
# let func f x = x + f(x);;
val func : (int -> int) -> int -> int = <fun>
# let square x = x * x;;
val square : int -> int = <fun>
# func square 2;;
- : int = 6
```

The `OCaml` interpreter typed `func` automatically. How?

They can also **return** functions:

```
# let func2 f x = (fun y -> f(y) + x);;
val func2 : ('a -> int) -> int -> 'a -> int = <fun>
# func2 square 1;;
- : int -> int = <fun>
```


Partial application allows one to fix some arguments of a function, resulting in another function with fewer arguments:

```
# let func3 x y z = x * y + z;;
val func3 : int -> int -> int -> int = <fun>
# func3 2 3;;
- : int -> int = <fun>
```


We want to model **nested function calls** in a language-agnostic fashion. To do so, we need a formalism that can handle:

- Variables to name various entities, including arguments of functions.
- Functions of one or more variables.
- Applying functions to inputs, be they functions, variables, or a combination thereof.

The entities we manipulate will be called **terms**.

- We consider an infinite set of symbols $\mathcal{V}$ called variables.
- We model functions **anonymously**: we do not necessarily name them, but we use the symbol λ to state that a term is a function.
- $\lambda x \cdot M$ stands for a function of argument x and body M, where $x \in \mathcal{V}$ and M is a term that may contain x.
- This concept is similar to C++'s **lambda functions** `[](int x){...}` or OCaml's `fun` functions `(fun x -> ...)`.

A Model for Functional Programming

Passing arguments

- A function may have more than one argument: given $x, y \in \mathcal{V}$, $\lambda xy \cdot M$ for stands for a function with two arguments.
- We pass or apply arguments to functions by juxtaposing them (in `OCaml` fashion): $fxyz$ stands for `f(x, y, z)`.
- We may use **parentheses** to remove ambiguity: $fx(gy)$ stands for `f(x, g(y))`.

E $\lambda xf \cdot \text{Plus } x\ (f\ x)$ models `let func x f = x + f(x);;`.

Practical Application

Exercise 1. Model `let func f x y = f(x) + x * f(y);;`.

Answer

$$\lambda fxy \cdot \text{Plus } (f\ x)\ (\text{Mult } x\ (f\ y))$$

Practical Application

Exercise 2. What does the term $\lambda fgxyz \cdot (g(fx)(fy))z$ model?

Answer

```
let func f g x y z = (g (f x) (f y)) z;;
```


Introducing Lambda Calculus

A formal definition I

Pure, untyped λ-calculus Λ

It is the language **generated by the following grammar** displayed in Backus-Naur form:

$$\begin{aligned} \text{<term>} &:= \text{<variable>} \mid \text{<function>} \mid \text{<application>} \\ \text{<variable>} &:= x \in \mathcal{V} \\ \text{<function>} &:= \lambda\text{<variable>} \cdot \text{<term>} \\ \text{<application>} &:= (\text{<term><term>}) \end{aligned}$$

where $\mathcal{V}$ is a set of variables.

Elements of Λ (or λ-terms) are both **data and functions**.

Introducing Lambda Calculus

Syntactic conventions

Application.	Omit outer parentheses.	$MN = (MN)$
	Associates to the left.	$MNL = (MN)L$
Function.	Yields priority to applications.	$\lambda x \cdot MN = \lambda x \cdot (MN)$
Currying.	Allow multiple arguments.	$\lambda xy \cdot M = \lambda x \cdot \lambda y \cdot M.$

Introducing Lambda Calculus

A peculiar mechanism

In **functional languages**, there is no difference between a function that takes n arguments and a single argument function returning a function that takes $n - 1$ arguments, a principle known as **currying**.

```
# let func x y = x * x + y;;
val func : int -> int -> int = <fun>
# func 2;;
- : int -> int = <fun>
```

Here, `func 2` is a function of a single variable `y`, all the occurrences of `x` having been replaced by the constant 2.

$$
\begin{aligned}
& \lambda n \cdot (\lambda f \cdot (\lambda x \cdot (f((nf)x)))) \\
= \quad & \lambda n \cdot (\lambda f \cdot (\lambda x \cdot (f(nfx)))) \\
= \quad & \lambda nfx \cdot (f(nfx)) \\
= \quad & \lambda nfx \cdot f(nfx)
\end{aligned}
$$

Consider the following three terms:

1. $f_1 = \lambda x \cdot gxx$ that models `let func1 x = g x x;;`.
2. $f_2 = \lambda y \cdot gyy$ that models `let func2 y = g y y;;`.
3. $f_3 = \lambda x \cdot hxx$ that models `let func3 x = h x x;;`.

We want to claim that f_1 and f_2 are **equivalent** up to argument renaming. The same cannot be said of f_2 and f_3 are not due to g and h being possibly different **global variables**.

The set FV of free variables of a term

It is defined inductively as follows:

$$
\begin{aligned}
\mathrm{FV}(x) &= \{x\} \\
\mathrm{FV}(\lambda x \cdot M) &= \mathrm{FV}(M) \setminus \{x\} \\
\mathrm{FV}(MN) &= \mathrm{FV}(M) \cup \mathrm{FV}(N)
\end{aligned}
$$

A variable $x \in \mathcal{V}$ is free if it appears **out of the quantification scope** of a function $\lambda x \cdot M$.

If $\mathrm{FV}(M) = \emptyset$, then M is said to be **closed** and called a **combinator**.

The set BV of bound variables of a term

It is defined inductively as follows:

$$
\begin{aligned}
\mathrm{BV}(x) &= \emptyset \\
\mathrm{BV}(\lambda x \cdot M) &= \mathrm{BV}(M) \cup \{x\} \\
\mathrm{BV}(MN) &= \mathrm{BV}(M) \cup \mathrm{BV}(N)
\end{aligned}
$$

A variable $x \in \mathcal{V}$ is bound in M if it appears **inside the quantification scope** of a $\lambda x \cdot M$.

From a certain point of view, free variables are similar to **global** variables, whereas bound variables are **local** arguments.

α-conversion
Handling free and bound variables

- A variable can be **both free and bound**, e.g. x in $x\lambda x \cdot (x\lambda x \cdot xx)$.
- Does $\boxed{x}$ in $x\lambda x \cdot (\boxed{x}\lambda x \cdot xx)$ refer to the free x, the first argument x, or the second argument x?
- We give priority to the **nearest local scope**, hence first argument x.
- Nevertheless, we wish to **rename** local variables to avoid this dilemma.

α-conversion
A matter of scope

? A similar convention holds in `C`, `C++`, and even `Python`.

```
>>> x = 0
... def f(x):
...   print(x)
... def g(y):
...   print(x)
>>> f(1)
1
>>> g(1)
0
```


α-conversion
Finding the scopes

- In order to rename an argument, we need to find which occurrences of a variable are **captured** by a given quantifier.
- As an example, consider term $\lambda x \cdot x(\lambda x \cdot xx)$.
- Note that the occurrences of x captured by the first λx are the **free** occurrences in the body $x(\lambda x \cdot xx)$.
- The **bound** occurrences belong to other subscopes.

α-conversion
Fresh substitution

Fresh substitution of variables

Given $M \in \Lambda$ and $x, y \in \mathcal{V}$ such that $y \notin \mathrm{FV}(M) \cup \mathrm{BV}(M)$, it is defined inductively:

$$\begin{aligned}
x[x \mathbin{/\!\!/} y] &= y \\
z[x \mathbin{/\!\!/} y] &= z \text{ with } x \neq z \\
(NL)[x \mathbin{/\!\!/} y] &= (N[x \mathbin{/\!\!/} y])(L[x \mathbin{/\!\!/} y]) \\
(\lambda x \cdot N)[x \mathbin{/\!\!/} y] &= \lambda x \cdot N \\
(\lambda z \cdot N)[x \mathbin{/\!\!/} y] &= \lambda z \cdot N[x \mathbin{/\!\!/} y] \text{ with } z \neq x
\end{aligned}$$

We replace all the **free** occurrences of x in M by a **new** variable y.

α-congruence (or α-conversion)

This **equivalence relation** (reflexive, symmetric, transitive) is defined inductively:

1. $\lambda x \cdot M \equiv_\alpha \lambda y \cdot M[x /\!\!/ y]$ for $y \notin \mathrm{FV}(M) \cup \mathrm{BV}(M)$.
2. $x \equiv_\alpha x$ for $x \in \mathcal{V}$.
3. $\lambda x \cdot M \equiv_\alpha \lambda x \cdot N$ if $M \equiv_\alpha N$.
4. $MN \equiv_\alpha M'N'$ if $M \equiv_\alpha M'$ and $N \equiv_\alpha N'$.

We only rename occurrences of argument x within body M of function $\lambda x \cdot M$ that are not **captured** by other quantifiers λx within M.

$$
\begin{aligned}
\lambda x \cdot x &\equiv_\alpha \lambda y \cdot y \\
x\lambda x \cdot x &\equiv_\alpha x\lambda y \cdot y \\
\lambda x \cdot x(\lambda x \cdot xz) &\equiv_\alpha \lambda y \cdot y(\lambda x \cdot xz) \\
&\equiv_\alpha \lambda y \cdot y(\lambda y \cdot yz) \\
x\lambda x \cdot x &\not\equiv_\alpha y\lambda y \cdot y \\
\lambda y \cdot \lambda x \cdot xy &\not\equiv_\alpha \lambda x \cdot \lambda x \cdot xx
\end{aligned}
$$

Exercise 3. Among the following terms, which ones are α-equivalent?

$$
\begin{aligned}
(\lambda x \cdot \lambda y \cdot x)x \\
\lambda x \cdot x \\
(\lambda x \cdot \lambda x \cdot x)x \\
(\lambda x \cdot \lambda y \cdot y)x \\
(\lambda x \cdot \lambda y \cdot y)(\lambda x \cdot x) \\
\lambda y \cdot x \\
\lambda y \cdot y
\end{aligned}
$$

$$
\begin{aligned}
\lambda x \cdot x &\equiv_\alpha \lambda y \cdot y \\
(\lambda x \cdot \lambda x \cdot x)x &\equiv_\alpha (\lambda x \cdot \lambda y \cdot y)x
\end{aligned}
$$

Barendregt's convention
Writing unambiguous terms

Despite the aforementioned convention on scopes, we wish to get rid of terms such as $x\lambda x \cdot x$ featuring variables that are both **bound** and **free**.

We therefore enforce a **naming convention** for variables:

Barendregt's convention

A λ-term M should follow these two rules:

- No variable is both free and bound.
- For any variable $x \in \mathcal{V}$ the symbol $\lambda x \cdot$ should never occur more than once in M.

Barendregt's convention
A few examples

- The term $x\lambda z \cdot z$ verifies Barendregt's convention.
- The term $\lambda x \cdot \lambda y \cdot xz$ does too.
- The term $x\lambda x \cdot x$ does not.
- The term $\lambda x \cdot \lambda x \cdot xz$ does not.

Barendregt's convention
Converting terms

Finding well-formed terms

Given a λ-term M, there exists a λ-term N **verifying Barendregt's convention** such that $M \equiv_\alpha N$.

From now on, we will use λ-terms that follow Barendregt's convention and reason over whole α-congruence classes **represented by such terms**.

Practical Application

Exercise 4. Find a term M following Barendregt's convention such that:

$$M \equiv_\alpha x\lambda x \cdot x(\lambda y \cdot xy)(\lambda x \cdot x)(\lambda y \cdot yx)$$

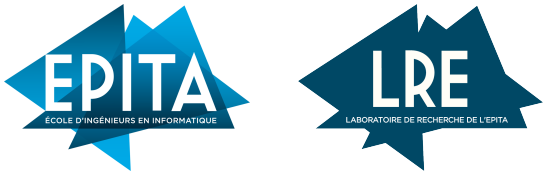
Answer

$$M = x\lambda z \cdot z(\lambda y \cdot zy)(\lambda u \cdot u)(\lambda v \cdot vz)$$

Formal Logics

β-reduction

Adrien Pommellet



June 23, 2025

β-reduction

Passing arguments

Consider the following `OCaml` functions:

```
# let func f x y = f(x) + x * f(y);;
# let square x = x * x;;
```

By currying, passing the arguments `square` and `2` to `func` yields:

```
func square 2 = (2 * 2) + 2 * (y * y)
```

As we pass arguments to functions, we **perform substitutions on their body**.

β-reduction

Performing substitutions

Substitution

This operation is defined inductively **modulo α-congruence** for a variable $x \in \mathcal{V}$ and a term $M \in \Lambda$:

$$\begin{aligned} x[x/M] &= M \\ y[x/M] &= y \text{ with } x \neq y \text{ and } y \in \mathcal{V} \\ (NL)[x/M] &= (N[x/M])(L[x/M]) \\ (\lambda y \cdot N)[x/M] &= \lambda y \cdot N[x/M] \text{ with } x \neq y \text{ and } y \notin \mathrm{FV}(M) \end{aligned}$$

It should not introduce free variables **interfering with a function**.

It is not to be mistaken with the **fresh substitution**.

β-reduction
Conversion

β-conversion

This binary relation **modulo α-congruence** is defined inductively:

$$(\lambda x \cdot M)N \quad \beta \quad M[x/N]$$

We **substitute** in the term (called a **redex**) $(\lambda x \cdot M)N$ the variable x in the body M of the **function** with the **argument** N.

Passing and reducing arguments in a functional language should be understood as a form of **data transformation**.

β-reduction
A matter of bound variables

We say that β-conversion is defined modulo α-congruence because we can replace a term by **an α-equivalent** one that happens to be more convenient (e.g. verifying Barendregt's convention).

Term $(\lambda x \cdot x\lambda y \cdot xy)y$ cannot be reduced: the **free** y would be mistaken for the **bound** occurrence.

But $(\lambda x \cdot x\lambda y \cdot xy)y \equiv_\alpha (\lambda x \cdot x\lambda u \cdot xu)y$, resulting in the reduction $(\lambda x \cdot x\lambda u \cdot xu)y \to_\beta y\lambda u \cdot yu$.

β-reduction
Reduction

β-reduction

Given two λ-terms X and Y such that $X\ \beta\ Y$ and M such that X is a sub-term of M, then we have:

$$M \to_\beta N$$

where N is obtained by **replacing X with Y in M**.

Intuitively, we **replace a sub-term X** of M with another related one Y according to the binary relation β.

β-reduction
A first example

$$\begin{aligned} & (\lambda x \cdot \lambda f \cdot xfy)(\lambda z \cdot z)(\lambda w \cdot ww) \\ \to_\beta \quad & (\lambda f \cdot (\lambda z \cdot z)fy)(\lambda w \cdot ww) \\ \to_\beta \quad & (\lambda f \cdot fy)(\lambda w \cdot ww) \\ \to_\beta \quad & (\lambda w \cdot ww)y \\ \to_\beta \quad & yy \end{aligned}$$

Thus $(\lambda x \cdot \lambda f \cdot xfy)(\lambda z \cdot z)(\lambda w \cdot ww) \to_\beta^* yy$ (i.e. in **0 or more** steps).

β-reduction
Other examples

$$
\begin{array}{rcl}
(\lambda x \cdot xx)y & \to_\beta & yy \\
(\lambda y \cdot yy)(\lambda x \cdot xx) & \to_\beta & (\lambda x \cdot xx)(\lambda x \cdot xx) \\
& \equiv_\alpha & (\lambda y \cdot yy)(\lambda x \cdot xx) \\
(\lambda y \cdot y(yy))(\lambda x \cdot x(xx)) & \to_\beta & (\lambda x \cdot x(xx))((\lambda x \cdot x(xx))(\lambda x \cdot x(xx))) \\
& \equiv_\alpha & (\lambda u \cdot x(uu))((\lambda y \cdot y(yy))(\lambda x \cdot x(xx)))
\end{array}
$$

β-reduction
A few peculiar combinators

Omega combinators

They consist in the three following α-**congruence classes**:

$$
\begin{array}{rcl}
\omega & \equiv_\alpha & \lambda x \cdot xx \\
\Omega & \equiv_\alpha & \omega\omega \\
\tilde{\Omega} & \equiv_\alpha & \lambda x \cdot x(xx)
\end{array}
$$

Note that $\omega\omega \to_\beta \omega\omega$, $\Omega \to_\beta \Omega$ and $\tilde{\Omega}\tilde{\Omega} \to_\beta \tilde{\Omega}(\tilde{\Omega}\tilde{\Omega})$.

Practical Application

Exercise 1. Reduce the term $X = (\lambda x \cdot xx)(\lambda y \cdot yx)z$ until no further reduction can be performed.

Answer

We first α-convert X to an equivalent term $Y = (\lambda u \cdot uu)(\lambda y \cdot yx)z$ following Barendregt's convention. Then:

$$
\begin{array}{rcl}
X & \equiv_\alpha & (\lambda u \cdot uu)(\lambda y \cdot yx)z \\
& \to_\beta & ((\lambda y \cdot yx)(\lambda y \cdot yx))z \\
& \equiv_\alpha & ((\lambda y \cdot yx)(\lambda v \cdot vx))z \\
& \to_\beta & ((\lambda v \cdot vx)x)z \\
& \to_\beta & (xx)z
\end{array}
$$

Termination of β-reduction

Normal forms

Normal form

A term M is in β-**normal form** is there is no term N such that $M \to_\beta N$.

Normalization

M is said to be β-**normalizable** if there exists N in β-normal form such that $M \to_\beta^* N$.

Strong normalization

M is said to be **strongly** β-**normalizable** if there exists no infinite reduction sequence starting from M. Note that M is therefore normalizable.

Practical Application

Exercise 2. Is the term $M = \lambda x \cdot x(\lambda y \cdot xy)(\lambda u \cdot u)(\lambda v \cdot vx)$ in β-normal form?

Answer

Note that due to the associativity and priority rules, M should be read as $\lambda x \cdot (((x(\lambda y \cdot xy))(\lambda u \cdot u))(\lambda v \cdot vx))$.

The term x is not a function and cannot be fed an argument. Thus, M cannot be reduced and is in β-normal form.

Termination of β-reduction

Normalizable relations

Normalizable relation

A binary relation $\to_\rho$ is **(strongly) normalizable** if any term is (strongly) normalizable w.r.t. $\to_\rho$.

Property

$\to_\beta$ is not normalizable, hence, not strongly normalizable.

Proof. Consider the infinite reduction sequence:

$$\Omega = \omega\omega \to_\beta \omega\omega \to_\beta \omega\omega \to_\beta \dots$$

Since it's the only possible reduction sequence, Ω is not normalizable.

Termination of β-reduction
An example

- Let $I = \lambda x \cdot x$. I is in β-**normal form**.
- Let $K = \lambda x \cdot (\lambda y \cdot x)$.
- $\Omega \to_\beta \Omega$, thus $KI\Omega \to_\beta KI\Omega$. $KI\Omega$ is **not strongly normalizable**.
- However:

$$\begin{array}{rc} & KI\Omega \\ = & (\lambda x \cdot (\lambda y \cdot x))(\lambda z \cdot z)\Omega \\ \to_\beta & (\lambda y \cdot (\lambda z \cdot z))\Omega \\ \to_\beta & \lambda z \cdot z \\ = & I \end{array}$$

Thus, $KI\Omega$ **is normalizable**.

Termination of β-reduction
Of determinism

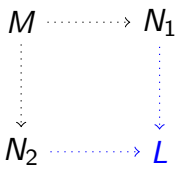
Note that $\to_\beta$ is not deterministic: from $KI\Omega$, there are **two possible reductions sequences**.

Can two reduction sequences starting from the same term **diverge**?

Termination of β-reduction
Converging paths

Church-Rosser

A binary relation $\to_\rho$ is Church-Rosser if:



$M \to_\rho^* N_1$ and $M \to_\rho^* N_2$ implies that $N_1 \to_\rho^* L$ and $N_2 \to_\rho^* L$.

Intuitively, two paths that diverge **will eventually meet again**.

Termination of β-reduction
Normal form property

Unique normal form property

A binary relation $\to_\rho$ on terms satisfies this property if $M \to_\rho^* N_1$, $M \to_\rho^* N_2$, and N_1, N_2 in ρ-normal form implies that $N_1 \equiv_\alpha N_2$.

Property

If $\to_\rho$ is Church-Rosser then it has the unique normal form property.

Theorem

The relation $\to_\beta$ is **Church-Rosser**, thus follows the **unique** β**-normal form** property.

We will **admit** this theorem.

Exercise 3. Reduce the term $A = (\lambda z \cdot z)(\lambda z \cdot zz)(\lambda z \cdot zy)$ until a β-normal form is reached.

We first α-convert A to an equivalent term $B = (\lambda z \cdot z)(\lambda u \cdot uu)(\lambda v \cdot vy)$ following Barendregt's convention. Then:

$$\begin{aligned} A &\equiv_\alpha (\lambda z \cdot z)(\lambda u \cdot uu)(\lambda v \cdot vy) \\ &\to_\beta (\lambda u \cdot uu)(\lambda v \cdot vy) \\ &\to_\beta (\lambda v \cdot vy)(\lambda v \cdot vy) \\ &\equiv_\alpha (\lambda w \cdot wy)(\lambda v \cdot vy) \\ &\to_\beta (\lambda v \cdot vy)y \\ &\to_\beta yy \end{aligned}$$

Exercise 4. Among the following terms, which ones reduce to which?

$$\begin{aligned} (\lambda x \cdot \lambda y \cdot x)x \\ \lambda x \cdot x \\ (\lambda x \cdot \lambda x \cdot x)x \\ (\lambda x \cdot \lambda y \cdot y)x \\ (\lambda x \cdot \lambda y \cdot y)(\lambda x \cdot x) \\ \lambda y \cdot x \\ \lambda y \cdot y \end{aligned}$$

Answer

$$
\begin{array}{rcl}
(\lambda x \cdot \lambda y \cdot x)x & \equiv_\alpha & (\lambda z \cdot \lambda y \cdot z)x \\
& \to_\beta & \lambda y \cdot x \\
(\lambda x \cdot \lambda x \cdot x)x & \equiv_\alpha & (\lambda x \cdot \lambda y \cdot y)x \\
& \equiv_\alpha & (\lambda z \cdot \lambda y \cdot y)x \\
& \to_\beta & \lambda y \cdot y \\
& \equiv_\alpha & \lambda x \cdot x \\
(\lambda x \cdot \lambda y \cdot y)(\lambda x \cdot x) & \equiv_\alpha & (\lambda z \cdot \lambda y \cdot y)(\lambda x \cdot x) \\
& \to_\beta & \lambda y \cdot y
\end{array}
$$

See you next class!

Formal Logics

Reduction Strategies

Adrien Pommellet

June 23, 2025

Evaluating Arguments

A practical issue I

```
void p(const std::string& s) {
 std::cout << "call_p\n" << s;
}

std::string f() {
 std::cout << "loc_f\n";
 return "ret_f\n";
}
```

```
int main() {
 p(f());
 return 0;
}
```

The program outputs `loc_f call_p ret_f` due to the arguments being evaluated **first**.

Evaluating Arguments
A practical issue II

```
int f(int x, int y) {
 if (x == 0) return x;
 return y / x;
}

int main() {
 f(0, so_slow_i_fell_asleep(42));
 return 0;
}
```

Here, `so_slow_i_fell_asleep(42)` is **needlessly** evaluated.

Evaluating Arguments
A practical issue III

```
int f() {
 std::cout << "loc_f\n";
 return 42;
}

int g() {
 std::cout << "loc_g\n";
 return 0;
}
```

```
int main() {
 return f() + g();
}
```

The program's behaviour is **unspecified** in C++.

Evaluating Arguments
Handling non-determinism

- We **perform computations** on λ-terms by β-reducing them to their normal form if it exists.
- However, β-reduction is **not deterministic**.
- We will use **strategies** (arbitrary choices) to enforce determinism.

The Head Reduction Strategy
Introducing strategies

Reduction strategy

It is a **partial function** $f : \Lambda \to \Lambda$ such that $X \to_\beta f(X)$. We then consider the binary relation $X \to_f f(X)$.

Intuitively, a strategy **chooses** which reduction to apply to a term X.

Any term has an **unique reduction sequence** according to f, which may be finite (ending with an f-**normal form**), infinite, or null.

The Head Reduction Strategy

Reducing the head first

Head reduction strategy h

It is defined **inductively** as follows:

- Applications. If $X = (\lambda y \cdot M)\ N\ L_1\ \ldots\ L_m$ for some $m \geq 0$, then $X \to_h M[y/N]\ L_1\ \ldots\ L_m$.
- Functions. If $X = \lambda x \cdot M$ and $M \to_h N$, then $X \to_h \lambda x \cdot N$.

Intuitively, we either try to feed to the leftmost term the second term from left if X is an **application**, or explore its body if iX is a **function** instead.

Head normal terms are of the form $\lambda x_1 \ldots x_n \cdot y\ L_1\ \ldots\ L_m$.

The Head Reduction Strategy

A few examples I

Let $I = \lambda x \cdot x$ and $K = \lambda x \cdot (\lambda y \cdot x)$. Remember the following sequence:

$$\begin{aligned} & KI\Omega \\ = \ & (\lambda x \cdot (\lambda y \cdot x))(\lambda z \cdot z)\Omega \\ \to_h \ & (\lambda y \cdot (\lambda z \cdot z))\Omega \\ \to_h \ & \lambda z \cdot z \\ = \ & I \end{aligned}$$

The Head Reduction Strategy

A few examples II

$$\begin{aligned} & K\omega\omega I \\ = \ & (\lambda x \cdot (\lambda y \cdot x))\omega\omega I \\ \to_h \ & (\lambda y \cdot \omega)\omega I \\ \to_h \ & \omega I \\ = \ & (\lambda x \cdot xx)I \\ \to_h \ & II \\ \to_h \ & I \end{aligned}$$

The Head Reduction Strategy

A problematic case

However, we **cannot reduce** the β-normalizable term $x(Ix) \not\to_h xx$, yet $x(Ix) \to_\beta xx$.

We therefore need to introduce another strategy.

The Leftmost Reduction Strategy

A new strategy

The leftmost reduction strategy I

It consists in performing a single β-conversion step on the **leftmost** $\lambda x \cdot M$ to which an argument can be matched.

Note that any h-reduction sequence is **also** a I-reduction sequence, but the converse is not true.

Consider the following leftmost reduction:

$$\begin{aligned} x(Ix) &= x((\lambda y \cdot y)x) \\ &\to_I xx \end{aligned}$$

The Leftmost Reduction Strategy

Normalizing strategies

If a λ-term is **normalizable**, how can we find its **unique** normal form?

We admit the following theorem:

Theorem

If M is a λ-term with a β-normal form N, then $M \to_I^* N$.

Leftmost reduction is said to be **normalizing** with regards to β-reduction.

Practical Application

Exercise 1. Consider the term $M = (\lambda x \cdot x(\lambda y \cdot y)x)(\lambda z \cdot (\lambda a \cdot aa)zb)$. Reduce M, first using a head reduction strategy, then a leftmost reduction strategy.

Answer I

First, we apply a head reduction strategy:

$$\begin{aligned} M &= (\lambda x \cdot x(\lambda y \cdot y)x)(\lambda z \cdot (\lambda a \cdot aa)zb) \\ &\to_\beta (\lambda z \cdot (\lambda a \cdot aa)zb)(\lambda y \cdot y)(\lambda z \cdot (\lambda a \cdot aa)zb) \\ &\equiv_\alpha (\lambda z \cdot (\lambda a \cdot aa)zb)(\lambda y \cdot y)(\lambda u \cdot (\lambda v \cdot vv)ub) \\ &\to_\beta (\lambda a \cdot aa)(\lambda y \cdot y)b(\lambda u \cdot (\lambda v \cdot vv)ub) \\ &\to_\beta (\lambda y \cdot y)(\lambda y \cdot y)b(\lambda u \cdot (\lambda v \cdot vv)ub) \\ &\equiv_\alpha (\lambda y \cdot y)(\lambda x \cdot x)b(\lambda u \cdot (\lambda v \cdot vv)ub) \\ &\to_\beta (\lambda x \cdot x)b(\lambda u \cdot (\lambda v \cdot vv)ub) \\ &\to_\beta b(\lambda u \cdot (\lambda v \cdot vv)ub) \end{aligned}$$

No further head reduction is possible: the first term is not a function, thus we cannot pass the second term as an argument.

Answer II

Let's use the leftmost reduction strategy instead:

$$
\begin{aligned}
M &= (\lambda x \cdot x(\lambda y \cdot y)x)(\lambda z \cdot (\lambda a \cdot aa)zb) \\
&\to_\beta (\lambda z \cdot (\lambda a \cdot aa)zb)(\lambda y \cdot y)(\lambda z \cdot (\lambda a \cdot aa)zb) \\
&\equiv_\alpha (\lambda z \cdot (\lambda a \cdot aa)zb)(\lambda y \cdot y)(\lambda u \cdot (\lambda v \cdot vv)ub) \\
&\to_\beta ((\lambda a \cdot aa)(\lambda y \cdot y)b)(\lambda u \cdot (\lambda v \cdot vv)ub) \\
&\to_\beta ((\lambda y \cdot y)(\lambda y \cdot y)b)(\lambda u \cdot (\lambda v \cdot vv)ub) \\
&\equiv_\alpha ((\lambda y \cdot y)(\lambda x \cdot x)b)(\lambda u \cdot (\lambda v \cdot vv)ub) \\
&\to_\beta ((\lambda x \cdot x)b)(\lambda u \cdot (\lambda v \cdot vv)ub) \\
&\to_\beta b(\lambda u \cdot (\lambda v \cdot vv)ub) \\
&\to_\beta b(\lambda u \cdot uub)
\end{aligned}
$$

Other Reduction Strategies

Practical reduction strategies

Actual languages uses **various reduction strategies**:

Call by value. A term has to be reduced **before being passed as an argument**.

Call by name. β-reduction is performed as soon as possible; the argument are **substituted early and reduced later**.

Call by need. Similar to call by name, but arguments are only **reduced when needed**. Also called *lazy evaluation*.

Other Reduction Strategies

Call by value

$$
\begin{aligned}
M &= (\lambda x \cdot \lambda y \cdot yx)((\lambda u \cdot u)a)(\lambda v \cdot v) \\
&\to_\beta (\lambda x \cdot \lambda y \cdot yx)(a)(\lambda v \cdot v) \\
&\to_\beta (\lambda y \cdot ya)(\lambda v \cdot v) \\
&\to_\beta (\lambda v \cdot v)a \\
&\to_\beta a
\end{aligned}
$$

Other Reduction Strategies

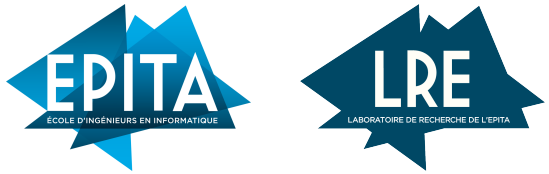
Call by name

$$
\begin{aligned}
M &= (\lambda x \cdot \lambda y \cdot yx)((\lambda u \cdot u)a)(\lambda v \cdot v) \\
&\to_\beta (\lambda y \cdot y((\lambda u \cdot u)a))(\lambda v \cdot v) \\
&\to_\beta (\lambda v \cdot v)((\lambda u \cdot u)a) \\
&\to_\beta (\lambda u \cdot u)a \\
&\to_\beta a
\end{aligned}
$$

Formal Logics

Church Encoding

Adrien Pommellet



June 23, 2025

Programming in Lambda Calculus

Implementation

Executing a program: using a **strategy** to deterministically β-**reduce** a λ-term to its **unique normal form**.

A **real** programming language needs:

- Booleans.
- Integers.
- Predicates (e.g. testing if x is null).
- Arithmetic operations.
- Recursion.

Programming in Lambda Calculus

A first try: handling Booleans

Church Booleans

Conventionally, they consist in the **true** and **false** terms in normal form:

$$\begin{aligned} \mathrm{True} &= \lambda xy \cdot x \\ \mathrm{False} &= \lambda xy \cdot y \end{aligned}$$

Note that $\mathrm{True}XY \to_\beta^* X$ and $\mathrm{False}XY \to_\beta^* Y$.

If $B \equiv_\alpha \mathrm{True}$ or $B \equiv_\alpha \mathrm{False}$ then BXY simulates the instruction `if` B `then` X `else` Y.

Church Arithmetic

Defining integers

Atomic variables in $\mathcal{V}$ don't have an **intrinsic** meaning: they can't be **assigned** actual values (e.g. integers) or reduced.

Our intuition is to use them as **counters**.

Church integers

We **conventionally** define the n-th integer term in normal form:

$$\underline{n} = \lambda fx \cdot f^n x$$

where $f^n x$ stands for $\underbrace{f(f(\ldots(fx)))}_{n \text{ times}}$.

Church Arithmetic

Defining predicates

E Under this notation, $\underline{0} = \lambda fx \cdot x$, $\underline{1} = \lambda fx \cdot fx$, $\underline{2} = \lambda fx \cdot f(fx)$, $\underline{3} = \lambda fx \cdot f(f(fx))$, etc.

How can we test that $\underline{n} = \underline{0}$? We will introduce a relevant **predicate**.

Testing nullity

Let $\text{IsZero} = \lambda x \cdot x(\lambda y \cdot \text{False})\text{True}$. Then:

$$\begin{aligned} \text{IsZero } \underline{0} &\to_\beta^* \text{True} \\ \text{IsZero } \underline{n} &\to_\beta^* \text{False} \end{aligned}$$

for any $n \in \mathbb{N}^*$.

Practical application

Exercise 1. Reduce $\text{IsZero } \underline{0}$ and $\text{IsZero } \underline{2}$.

Answer I

$$\begin{aligned} \text{IsZero } \underline{0} &= (\lambda x \cdot x(\lambda y \cdot \text{False})\text{True})(\lambda fz \cdot z) \\ &\to_\beta (\lambda fz \cdot z)(\lambda y \cdot \text{False})\text{True} \\ &\to_\beta (\lambda z \cdot z)\text{True} \\ &\to_\beta \text{True} \end{aligned}$$

Answer II

$$\begin{aligned} \text{IsZero } \underline{2} &= (\lambda x \cdot x(\lambda y \cdot \text{False})\text{True})(\lambda fz \cdot f(fz)) \\ &\to_\beta (\lambda fz \cdot f(fz))(\lambda y \cdot \text{False})\text{True} \\ &\to_\beta (\lambda z \cdot (\lambda y \cdot \text{False})((\lambda y \cdot \text{False})z))\text{True} \\ &\to_\beta (\lambda z \cdot (\lambda y \cdot \text{False})\text{False})\text{True} \\ &\to_\beta (\lambda z \cdot \text{False})\text{True} \\ &\to_\beta \text{False} \end{aligned}$$

Church Arithmetic

Incrementing integers

We want to be able to perform **simple arithmetic operations** on these integers.

The successor function

Let $\mathrm{Succ} = \lambda yfx \cdot f(yfx)$. Then:

$$\mathrm{Succ}\ \underline{n} \quad \to_\beta^* \quad \underline{n+1}$$

for any $n \in \mathbb{N}$.

? It simulates the instruction `n++`.

Church Arithmetic

Computing a successor

$$\begin{aligned}
\mathrm{Succ}\ \underline{n} &= (\lambda yfx \cdot f(yfx))\underline{n} \\
&\to_\beta \lambda fx \cdot f(\underline{n}fx) \\
&= \lambda fx \cdot f((\lambda uv \cdot u^n v)fx) \\
&\to_\beta^* \lambda fx \cdot f(f^n x) \\
&= \lambda fx \cdot (f^{n+1} x) \\
&= \underline{n+1}
\end{aligned}$$

Church Arithmetic

Adding integers

The addition function

Let $\mathrm{Plus} = \lambda y \cdot y\ \mathrm{Succ}$. Then:

$$\mathrm{Plus}\ \underline{n}\ \underline{m} \quad \to_\beta^* \quad \underline{n+m}$$

for any $n, m \in \mathbb{N}$.

? It simulates the instruction `n + m`.

Church Arithmetic

Computing an addition

$$\begin{aligned}
\mathrm{Plus}\ \underline{n}\ \underline{m} &= (\lambda y \cdot y\ \mathrm{Succ})\ \underline{n}\ \underline{m} \\
&\to_\beta \underline{n}\ \mathrm{Succ}\ \underline{m} \\
&= (\lambda fx \cdot f^n x)\ \mathrm{Succ}\ \underline{m} \\
&\to_\beta (\lambda x \cdot \mathrm{Succ}^n x)\ \underline{m} \\
&\to_\beta \mathrm{Succ}^n\ \underline{m} \\
&\to_\beta^* \underline{n+m}
\end{aligned}$$

Intuitively, $\underline{n}\ \mathrm{Succ}\ \underline{m}$ means: apply function Succ n times to m.

Practical Application

Exercise 2. Define a term $\mathrm{Mult} \in \Lambda$ such that for any natural integers n, m:

$$\mathrm{Mult}\ \underline{n}\ \underline{m} \to_\beta^* \underline{n \times m}$$

Don't forget to prove that this property actually holds!

Answer I

- Computing $n \times m$ consists in **adding** m n times to 0.
- $\underline{n}\ f\ x$ applies function f n times to input x.
- By currying, function $\mathrm{Plus}\ \underline{m}$ adds m to a Church integer.
- We will consider $\mathrm{Mult} = \lambda xy \cdot x\ (\mathrm{Plus}\ y)\ \underline{0}$.

Answer II

$$\begin{aligned}
\mathrm{Mult}\ \underline{n}\ \underline{m} &= (\lambda xy \cdot x\ (\mathrm{Plus}\ y)\ \underline{0})\ \underline{n}\ \underline{m} \\
&\to_\beta (\lambda y \cdot \underline{n}\ (\mathrm{Plus}\ y)\ \underline{0})\ \underline{m} \\
&\to_\beta \underline{n}\ (\mathrm{Plus}\ \underline{m})\ \underline{0} \\
&= (\lambda fx \cdot f^n x)(\mathrm{Plus}\ \underline{m})\ \underline{0} \\
&\to_\beta (\lambda x \cdot (\mathrm{Plus}\ \underline{m})^n x)\ \underline{0} \\
&\to_\beta (\mathrm{Plus}\ \underline{m})^n\ \underline{0} \\
&\to_\beta^* \underline{m + \ldots + m} \\
&= \underline{n \times m}
\end{aligned}$$

Practical Application

Exercise 3. Define in a similar manner a term $\mathrm{Pow} \in \Lambda$ such that for any natural integers n, m:

$$\mathrm{Pow}\ \underline{n}\ \underline{m} \to_\beta^* \underline{n^m}$$

Answer

Computing n^m consists in **multiplying** 1 m times by n. Thus, we consider:

$$\text{Pow} = \lambda xy \cdot y\ (\text{Mult}\ x)\ \underline{1}$$

Church Arithmetic

Encoding rational numbers

Church Pairs

We define the following λ-terms:

$$\begin{aligned} \text{Pair} &= \lambda xyf \cdot fxy \\ \text{First} &= \lambda p \cdot p\text{True} \\ \text{Second} &= \lambda p \cdot p\text{False} \end{aligned}$$

Then the following property holds:

$$\begin{aligned} \text{First}(\text{Pair}AB) &\to_\beta^* A \\ \text{Second}(\text{Pair}AB) &\to_\beta^* B \end{aligned}$$

for any λ-terms A and B.

Church Arithmetic

Computing projections

$$\begin{aligned} \text{First}(\text{Pair}AB) &= (\lambda p \cdot p\text{True})(\text{Pair}AB) \\ &\to_\beta (\text{Pair}AB)\text{True} \\ &= ((\lambda xyf \cdot fxy)AB)\text{True} \\ &\to_\beta^* \text{True}AB \\ &\to_\beta^* A \end{aligned}$$

$$\begin{aligned} \text{Second}(\text{Pair}AB) &\to_\beta^* \text{False}AB \\ &\to_\beta^* B \end{aligned}$$

Fixed-point Combinators

Fixed points

We need the following definitions in order to introduce **recursion**.

Confluence relation

If $A \to_\beta^* C$ and $B \to_\beta^* C$ then $A \leftrightarrow_\beta^* B$.

As a consequence, A, B and C have the **same normal form**: these programs are equivalent.

Fixed-point Combinators
Introducing combinators

Fixed point of A

It is a term M such that $AM \leftrightarrow_\beta^* M$.

Fixed-point combinator

It is a term $M \in \Lambda$ such that for any λ-term A, $MA \leftrightarrow_\beta^* A(MA)$.

Curry's Y combinator

The term $Y = \lambda f \cdot (\lambda x \cdot f(xx))(\lambda y \cdot f(yy))$ is a fixed-point combinator.

Fixed-point Combinators
Reducing Y

$$\begin{array}{rcl} YA & = & (\lambda f \cdot (\lambda x \cdot f(xx))(\lambda y \cdot f(yy)))A \\ & \to_\beta & (\lambda x \cdot A(xx))(\lambda y \cdot A(yy)) \\ & \to_\beta & A((\lambda y \cdot A(yy))(\lambda y \cdot A(yy))) \\ & \equiv_\alpha & A((\lambda x \cdot A(xx))(\lambda y \cdot A(yy))) \\ A(YA) & = & A((\lambda f \cdot (\lambda x \cdot f(xx))(\lambda y \cdot f(yy)))A) \\ & \to_\beta & A((\lambda x \cdot A(xx))(\lambda y \cdot A(yy))) \end{array}$$

Fixed-point Combinators
Finding fixed points

Theorem

Any term $A \in \Lambda$ admits at least one fixed point.

Proof. Consider the term YA. We have $YA \leftrightarrow_\beta^* A(YA)$.

Fixed-point combinators can be used to implement **recursion** in functional languages. There is more than one fixed-point combinator!

Practical Application

Exercise 4. Admit that there exists a **predecessor** function Pred such that $\mathrm{Pred}\ \underline{n+1} \to_\beta^* \underline{n}$ for all $n \in \mathbb{N}$.

Let $F = \lambda g k \cdot (\mathrm{IsZero}\ k)\ \underline{1}\ (\mathrm{Mult}\ k\ (g(\mathrm{Pred}\ k)))$.

What is a fixpoint of F?

Answer I

Translating this function to `OCaml` yields:

```
let func g k = if k = 0 then 1 else (k * g(k-1));;
```

If g is a term such that $\forall n \in \mathbb{N}$, $g\underline{n}$ is a Church integer $\underline{g(n)}$, then:

- $(F\ g)\ \underline{0} \rightarrow_\beta^* \underline{0}$.
- $(F\ g)\ \underline{n+1} \rightarrow_\beta^* \underline{(n+1) \times g(n)}$ for $n \in \mathbb{N}$.

Answer II

A fixpoint f of F is a term such that $F\ f \leftrightarrow_\beta^* f$. Practically speaking:

```
f = func g = fun k -> if k = 0 then 1 else (k * g(k-1));;
```

Term f is such that:

- $f\ \underline{0} \rightarrow_\beta^* \underline{1}$.
- $f\ \underline{n+1} \rightarrow_\beta^* \underline{(n+1) \times f(n)}$ for $n \in \mathbb{N}$.

As a consequence, f implements the **factorial** function.

Optional Homework

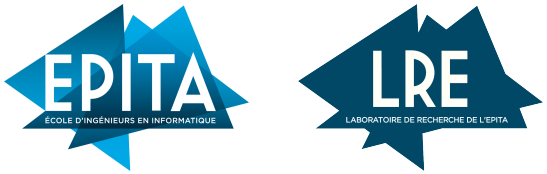
- Exercises **5A**, **5B** and **6** of the 2019-2020 exam (ignore types).

See you next class!

Formal Logics

Simple Type System

Adrien Pommellet



June 23, 2025

Typing Terms

A practical issue

Some functions can be **typed** automatically by the `OCaml` interpreter:

```
# let func f x = x + f(x);;
val func : (int -> int) -> int -> int = <fun>
```

Others, however, result in a somewhat cryptic error:

```
# let omega x = x x;;
Error: This expression has type 'a -> 'b
       but an expression was expected of type 'a
       The type variable 'a occurs inside 'a -> 'b
```


Typing Terms

Typing λ-calculus

- What does $\lambda x \cdot xx$ mean?
- We pass x as an **argument to itself**.
- In a real program, a function and its argument should **behave differently**.
- We therefore need to **type** terms.

Typing Terms

The set of types

Types $\mathcal{T}$

This set is defined inductively as follows:

$\mathcal{A}$. A set of type variables $\mathcal{TV}$.
$\mathcal{C}$. $\{\rightarrow^2\}$.
d. $+\infty$.

By convention, $\rightarrow$ is **right associative**. $\alpha \rightarrow \beta \rightarrow \gamma = \alpha \rightarrow (\beta \rightarrow \gamma)$

Typing Terms
Declaring types

Type statement

It is a pair $M : \sigma$ where M is a λ-term called the **subject** and σ is a type in $\mathcal{T}$ called the **predicate**. It means M is of type σ.

Intuitively, $\alpha \to \beta$ stands for a **functional** type with an **argument** of type α and an **output** of type β.

Typing Terms
Inductively typing terms

Intuitively, we would like to apply the following **typing rules**:

Functions. If $x : \alpha$ and $M : \beta$, then $\lambda x \cdot M : \alpha \to \beta$.

Applications. If $M : \alpha \to \beta$ and $N : \alpha$, then $MN : \beta$.

Type Systems
Using inference rules

We will use **inference rules** on type declarations:

Type inference rule

It consists in a finite set of **premisses** $\{M_1 : \sigma_1, \ldots, M_n : \sigma_n\}$, a finite set of **hypotheses** $\{N_1 : \mu_1, \ldots, N_n : \mu_n\}$ and a **conclusion** $M : \sigma$. It is written:

$$\frac{\begin{array}{ccc}[N_1 : \mu_1] & & [N_n : \mu_n] \\ \vdots & & \vdots \\ M_1 : \sigma_1 & \ldots & M_n : \sigma_n\end{array}}{M : \sigma}\,[t]$$

Intuitively, it means that if for all i, M_i of type σ_i is true or can be **inferred from** N_i of type μ_i, then M is of type σ.

Type Systems
An inference rule

Consider the type inference rule:

$$\frac{M : \sigma \to \tau \qquad N : \sigma}{MN : \tau}\,[A]$$

Note that M, N, σ, and τ are not actual terms and types, but **meta-variables** that can be substituted.

Type Systems
Type rulesets

Type ruleset $\mathcal{R}$

It is a (possibly infinite) **set of type inference rules**.

Simple type system $\mathcal{S}$

It features two type inference rules:

$$\frac{\begin{array}{c}[x:\sigma]\\ \vdots \\ M:\tau\end{array}}{\lambda x \cdot M : \sigma \to \tau}\,[\lambda] \qquad \frac{M:\sigma \to \tau \qquad N:\sigma}{MN:\tau}\,[A]$$

$[\lambda]$ and $[A]$ are respectively called **abstraction** and **application**.

Type Systems
Natural deduction and typing

Natural deduction

- **Deduction** trees.
- Cancelling leaves.
- Conclusion and hypotheses.
- **Theorems** under $\mathcal{N}$.
- A relation $\vdash_{\mathcal{N}}$.

Simple type system

- **Type derivation** trees.
- Cancelling leaves.
- Conclusion and hypotheses.
- **Derivable** statements under $\mathcal{S}$.
- A relation $\vdash_{\mathcal{S}}$, also written $\vdash$.

Type derivations must be **coherent**: a term cannot be assigned two different types.

Applying the Simple Type System
A few examples I

Prove that $\vdash \lambda x \cdot x : \sigma \to \sigma$.

E

$$\frac{\dfrac{}{x:\sigma}\,1}{\lambda x \cdot x : \sigma \to \sigma}\,1$$

We do **not need to label the rules**: there are only two of them.

Applying the Simple Type System
A few examples II

```
# let f x = x;;
val f : 'a -> 'a = <fun>
```

?

Note that σ isn't an actual type in $\mathcal{TV}$ but a **meta-variable** that merely describes a **possible** structure of the predicate.

Applying the Simple Type System
A few examples III

Prove that $\vdash \lambda x \cdot x : (\sigma \to \sigma) \to (\sigma \to \sigma)$.

E

$$\dfrac{\dfrac{}{x : (\sigma \to \sigma)}\,1}{\lambda x \cdot x : (\sigma \to \sigma) \to (\sigma \to \sigma)}\,1$$

?

There may be **multiple** ways to type a term. $M : \sigma$ stands for 'M **can be** of type σ'.

Applying the Simple Type System
Understanding Types

A type merely expresses a **possible shape** of a term based on the constraints implied by the **interactions of its variables**.

Applying the Simple Type System
A few examples IV

Prove that $\vdash \lambda xy \cdot x : \sigma \to \tau \to \sigma$.

E

$$\dfrac{\dfrac{\dfrac{}{x : \sigma}\,1}{\lambda y \cdot x : \tau \to \sigma}}{\lambda xy \cdot x : \sigma \to \tau \to \sigma}\,1$$

?

Remember that there is no **need to always cancel the premisses** in order to apply an inference rule.

Here, we don't need to insert a premiss $y : \tau$. **Implicitly**, the meta-variable τ stands for the type of y.

Applying the Simple Type System
Understanding Type Variables

In a derivation, type statements of the form $x : \sigma$ and $y : \tau$ merely mean the two variables x and y **can** (but do not have to) be of **different types**.

Applying the Simple Type System

A few examples IV

Prove that $\vdash \lambda fx \cdot f(fx) : (\sigma \to \sigma) \to \sigma \to \sigma$.

E

$$\dfrac{\dfrac{}{f : \sigma \to \sigma}\,1 \qquad \dfrac{\dfrac{}{f : \sigma \to \sigma}\,1 \qquad \dfrac{}{x : \sigma}\,2}{fx : \sigma}}{\dfrac{\dfrac{f(fx) : \sigma}{\lambda x \cdot f(fx) : \sigma \to \sigma}\,2}{\lambda fx \cdot f(fx) : (\sigma \to \sigma) \to \sigma \to \sigma}\,1}$$

Practical Application

Exercise 1. Write a type derivation of the term $\lambda xy \cdot xy$.

Answer

$$\dfrac{\dfrac{\dfrac{}{x : \sigma \to \tau}\,1 \qquad \dfrac{}{y : \sigma}\,2}{xy : \tau}}{\dfrac{\lambda y \cdot xy : \sigma \to \tau}{\lambda xy \cdot xy : (\sigma \to \tau) \to \sigma \to \tau}\,1}\,2$$

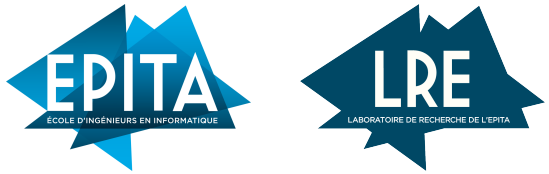
Optional Homework

- Exercises **5A**, **5B** and **7** of the 2019-2020 exam (compute the types).

Formal Logics

Type Assignments

Adrien Pommellet



June 23, 2025

Typable Terms

A definition

Typable term

It is a λ-term M such that there exists $\sigma \in \mathcal{T}$, $\vdash M : \sigma$. We then say that σ is **inhabited**.

$\Lambda^{\to}$ then stands for the set of **typable** λ-terms.

Typable Terms

Two examples

$I = \lambda x \cdot x : \sigma \to \sigma$ is **typable**.

$$\frac{\dfrac{}{x : \sigma}\,1}{\lambda x \cdot x : \sigma \to \sigma}\,1$$

E So is $K = \lambda xy \cdot x : \sigma \to \tau \to \sigma$.

$$\frac{\dfrac{\dfrac{}{x : \sigma}\,1}{\lambda y \cdot x : \tau \to \sigma}}{\lambda xy \cdot x : \sigma \to \tau \to \sigma}\,1$$

Typable Terms

A counter-example

Property

$\omega = \lambda x \cdot xx$ is **not typable**.

Proof. Consider the only possible derivation of ω:

$$\frac{\dfrac{x : \mu \to \tau \qquad x : \mu}{xx : \tau}}{\lambda x \cdot xx : \sigma \to \tau}$$

This deduction is **not coherent**: x can't simultaneously be of type $\mu \to \tau$ and of type μ. Moreover, it is **impossible to cancel all the leaves**.

Typable Terms

Typable sub-terms I

Property

If N is a **closed** sub-term of a typable term M, then N is typable.

From a type derivation of M, we can **extract** a type derivation of N. The full proof is available in the class notes.

Typable Terms

Other counter-examples

Property

Ω, $\tilde{\Omega}$ and Y are **not typable**.

Proof. If $\Omega = \omega\omega$ were typable, so would be its closed sub-term ω. Similarly, $\tilde{\Omega} = \lambda x \cdot x(xx)$ and $Y = \lambda f \cdot (\lambda x \cdot f(xx))(\lambda y \cdot f(yy))$ are not typable.

Practical Application

Exercise 1. Prove that $\underline{2}$ and $\underline{1}$ are of the same type.

Answer

Note that $\vdash \underline{2} = \lambda fx \cdot f(fx) : (\sigma \to \sigma) \to \sigma \to \sigma$, as proven previously.

Moreover, $\underline{1} = \lambda fx \cdot fx \equiv_\alpha \lambda xy \cdot xy$. But it has been shown that $\vdash \lambda xy \cdot xy : (\sigma \to \tau) \to \sigma \to \tau$, thus $\vdash \underline{1} : (\sigma \to \sigma) \to \sigma \to \sigma$ if we substitute τ with σ.

Practical Application

Exercise 2. Write a type derivation of the n-th Church integer $\underline{n}$.

Answer

$$\dfrac{\dfrac{}{f : \sigma \to \sigma}\,1 \qquad \dfrac{\dfrac{}{f : \sigma \to \sigma}\,1 \qquad \dfrac{}{x : \sigma}\,2}{fx : \sigma}}{f(fx) : \sigma}$$

$$\dfrac{\dfrac{\dfrac{}{f : \sigma \to \sigma}\,1 \qquad \vdots \; f^{n-1}(x) : \sigma}{f^n(x) : \sigma}}{\dfrac{\lambda x \cdot f^n(x) : \sigma \to \sigma}{\underline{n} = \lambda f x \cdot f^n(x) : (\sigma \to \sigma) \to \sigma \to \sigma}\,1}\,2$$

Practical Application

Exercise 3. Can you guess a type of Plus without writing a type derivation?

Answer

Plus takes two integer arguments and returns an integer. Its type is therefore:

$$((\sigma \to \sigma) \to \sigma \to \sigma) \to ((\sigma \to \sigma) \to \sigma \to \sigma) \to ((\sigma \to \sigma) \to \sigma \to \sigma)$$

Properties of Typable Terms

α-conversion

α-invariance

If $\vdash M : \sigma$ and $M \equiv_\alpha N$ then $\vdash N : \sigma$.

Proof. Any type derivation of M can be transformed into a type derivation of N by relabelling some variables.

Properties of Typable Terms

β-reduction

We admit the following result:

Subject reduction theorem

β-reduction **preserves type**: $\vdash M : \sigma$ and $M \to_\beta N$ then $\vdash N : \sigma$.

Note that the **converse** is not true: $KI\Omega \to_\beta^* I$ and I is typable but $KI\Omega$ isn't; otherwise, its closed sub-term Ω would be typable as well.

Properties of Typable Terms

Normal forms

We also admit the following theorem:

Strong β-normalization

All terms in $\Lambda^\to$ are **strongly β-normalizing**.

$\Lambda^\to$ makes for a **reasonable programming model**: each term is properly typed and always converges to an unique value through a reduction mechanism.

About Typing

Common problems

We consider the following problems:

Type checking. Given $M \in \Lambda$, $\sigma \in \mathcal{T}$, does $\vdash M : \sigma$?

Typability. Given $M \in \Lambda$, is there a $\sigma \in \mathcal{T}$ such that $\vdash M : \sigma$?

Inhabitation. Given $\sigma \in \mathcal{T}$, is there a $M \in \Lambda$ such that $\vdash M : \sigma$?

We will admit that:

Theorem

Typability is **decidable**. Moreover, there exists a **computable principal type** σ such that $\vdash M : \sigma$ and any other type τ of M can be obtained by applying a substitution to σ.

The principal type of $I = \lambda x \cdot x$ is $\sigma \to \sigma$.

E While $\vdash I : (\sigma \to \mu) \to (\sigma \to \mu)$ also holds, it can be obtained by applying the **substitution** of σ by $(\sigma \to \mu)$ to the principal type.

Corrolary

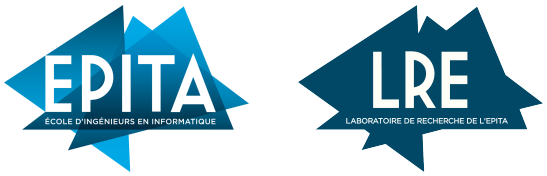
Type checking is **decidable**.

Proof. Determine the principal type μ of M, then check that σ can be obtained from μ by applying a substitution (e.g. by comparing their tree representations).

Formal Logics

The Curry-Howard Isomorphism

Adrien Pommellet



June 23, 2025

We want to show that **natural deduction** and the **simple type system** are somehow equivalent.

$$\frac{\begin{array}{c}[A]\\ \vdots \\ B\end{array}}{A \Rightarrow B}\,[\Rightarrow_I] \qquad \frac{\begin{array}{c}[x : \sigma]\\ \vdots \\ M : \tau\end{array}}{\lambda x \cdot M : \sigma \to \tau}\,[\lambda]$$

$$\frac{A \Rightarrow B \qquad A}{B}\,[\Rightarrow_E] \qquad \frac{M : \sigma \to \tau \qquad N : \sigma}{MN : \tau}\,[A]$$

The Curry-Howard Isomorphism

Consider an isomorphism κ between the set of **propositional variables** $\mathcal{V}$ and the **set of type variables** $\mathcal{TV}$. We define inductively $\mathcal{C} : \mathcal{F}_{\{\Rightarrow\}} \to \mathcal{T}$.

- $\mathcal{C}(x) = \kappa(x)$ for $x \in \mathcal{V}$.
- $\mathcal{C}(A \Rightarrow B) = \mathcal{C}(A) \to \mathcal{C}(B)$.

$\mathcal{C}$ is also an **isomorphism**.

A **propositional formula** is isomorphic to a **type**.

Theorem

P is a theorem under $\mathcal{N}_{\Rightarrow}$ if and only there exists $M \in \Lambda^{\to}$ such that $\vdash M : \mathcal{C}(P)$.

An **inhabited type** is isomorphic to a **theorem** under the **implication fragment** of natural deduction.

The following proof under natural deduction and type derivation are **isomorphic**:

E

$$\dfrac{\dfrac{\dfrac{}{A}\,1}{B \Rightarrow A}\,[\Rightarrow_I]}{A \Rightarrow B \Rightarrow A}\,[\Rightarrow_I]^1 \qquad \dfrac{\dfrac{\dfrac{}{x : \sigma}\,1}{\lambda y \cdot x : \tau \to \sigma}\,[\lambda]}{\lambda xy \cdot x : \sigma \to \tau \to \sigma}\,[\lambda]^1$$

Given a theorem $P \in \mathcal{F}_{\{\Rightarrow\}}$, we compute $M \in \Lambda^{\to}$ such that $\vdash M : \mathcal{C}(P)$ in the following fashion:

1. Compute a **proof tree** under $\mathcal{N}_{\Rightarrow}$ of P.
2. Replace the formulas and the rules of this tree by their isomorphic types and type inference rules. Only the terms should be missing from the tree.
3. Label the leaves with simple λ-calculus variables in $\mathcal{V}$. Use the same variable if several leaves are cancelled by the same rule.
4. Use the type inference rules to iteratively compute the ancestors of the leaves until the **root term** M has been computed.

Defining the Isomorphism

Normalization of type derivations

$$\dfrac{\dfrac{\begin{array}{c}[A]\\ \vdots\\ B\end{array}}{A \Rightarrow B}\,[\Rightarrow_I] \qquad A}{B}\,[\Rightarrow_E] \quad \rightsquigarrow \quad \begin{array}{c}[A]\\ \vdots\\ B\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}[x:\sigma]\\ \vdots\\ M:\tau\end{array}}{\lambda x \cdot M : \sigma \to \tau}\,[\lambda] \qquad \begin{array}{c}\vdots\\ N:\sigma\end{array}}{(\lambda x \cdot M)N : \tau}\,[A] \quad \rightsquigarrow \quad \begin{array}{c}[x:\sigma]\\ \vdots\\ M:\tau\end{array}$$

Defining the Isomorphism

β-reduction

From the type derivation of the **redex** $(\lambda x \cdot M)N : \tau$, we can extract a type derivation of the **reduced** term $M[x/N]$.

$$\begin{array}{c}[x:\sigma]\\ \vdots\\ M:\tau\end{array} \quad \longrightarrow \quad \begin{array}{c}[N:\sigma]\\ \vdots\\ M[x/N]:\tau\end{array}$$

This is a sketch of the proof of the **subject reduction** theorem: β-reduction preserves types.

Extending the Isomorphism

Handling full formulas

- So far, the isomorphism $\mathcal{C}$ can **only** handle formulas in $\mathcal{F}_{\{\Rightarrow\}}$.
- What about **full formulas** in $\mathcal{F}_{\{\bot,\neg,\wedge,\vee,\Rightarrow\}}$?
- We need to **extend** Λ, $\mathcal{T}$, $\mathcal{C}$, β and $\mathcal{S}$.

Extending the Isomorphism

$\wedge$ and pairs I

We add **pairs** in a manner similar to the `C` data type `struct`.

- Λ. We add a **constructor** $\langle\rangle$ of arity 2 and two **constructors** Π_1, Π_2 of arity 1.
- $\mathcal{T}$. We add a **constructor** $\times$ of arity 2.
- $\mathcal{C}$. We define $\mathcal{C}(A \wedge B) = \mathcal{C}(A) \times \mathcal{C}(B)$.
- β. We **extend** β-reduction with the following relations:

$$\begin{array}{lcl}\Pi_1(\langle M, N\rangle) & \beta & M\\ \Pi_2(\langle M, N\rangle) & \beta & N\end{array}$$

Extending the Isomorphism

$\wedge$ and pairs II

We add the following **type inference rules** to the simple type system $\mathcal{S}$:

$$\frac{A \qquad B}{A \wedge B}\,[\wedge_I] \qquad \frac{M : \sigma \qquad N : \tau}{\langle M, N\rangle : \sigma \times \tau}\,[\times_I]$$

$$\frac{A \wedge B}{A}\,[\wedge^l_E] \qquad \frac{M : \sigma \times \tau}{\Pi_1(M) : \sigma}\,[\times^l_E]$$

$$\frac{A \wedge B}{B}\,[\wedge^r_E] \qquad \frac{M : \sigma \times \tau}{\Pi_2(M) : \tau}\,[\times^r_E]$$

Extending the Isomorphism

$\vee$ and unions I

We add **unions** in a manner similar to the `C` data type `union`.

- Λ. We add a **constructor** $\oplus$ of arity 3 and two **constructors** K_1, K_2 of arity 1.
- $\mathcal{T}$. We add a **constructor** $\cup$ of arity 2.
- $\mathcal{C}$. We define $\mathcal{C}(A \vee B) = \mathcal{C}(A) \cup \mathcal{C}(B)$.
- β. We extend β-reduction with the following relations:

$$\oplus(\lambda u \cdot U, \lambda v \cdot V, K_1(M)) \quad \beta \quad U[u/M]$$
$$\oplus(\lambda u \cdot U, \lambda v \cdot V, K_2(M)) \quad \beta \quad V[v/M]$$

Extending the Isomorphism

$\vee$ and unions II

We add the following **type inference rules** to the simple type system $\mathcal{S}$:

$$\frac{A}{A \vee B}\,[\vee^l_I] \qquad \frac{M : \sigma}{K_1(M) : \sigma \cup \tau}\,[\cup^l_I]$$

$$\frac{B}{A \vee B}\,[\vee^r_I] \qquad \frac{M : \tau}{K_2(M) : \sigma \cup \tau}\,[\cup^r_I]$$

$$\frac{A \vee B \qquad \begin{matrix}[A]\\ \vdots\\ C\end{matrix} \qquad \begin{matrix}[B]\\ \vdots\\ C\end{matrix}}{C}\,[\vee_E] \qquad \frac{M : \sigma \cup \tau \qquad \begin{matrix}[u : \sigma]\\ \vdots\\ U : \mu\end{matrix} \qquad \begin{matrix}[v : \tau]\\ \vdots\\ V : \mu\end{matrix}}{\oplus(\lambda u \cdot U, \lambda v \cdot V, M) : \mu}\,[\cup_E]$$

Where $u, v \notin FV(M)$.

Extending the Isomorphism

$\bot$ and the empty type

We deal with $\bot$ by using the **empty type**.

- Λ. We add a **constructor** ε (**error**) of arity 1 to Λ.
- $\mathcal{T}$. We add an **atomic element** $\emptyset$ to $\mathcal{T}$.
- $\mathcal{C}$. We define $\mathcal{C}(\bot) = \emptyset$.

We add the following **type inference rule** to the simple type system $\mathcal{S}$:

$$\frac{\bot}{A}\,[\bot_E] \qquad \frac{M : \emptyset}{\varepsilon(M) : \sigma}\,[\emptyset_E]$$

Extending the Isomorphism

Negation free natural deduction

There is **no** equivalent to $\neg$.

Theorem

P is a **theorem** under $\mathcal{N}' = \mathcal{N} - \{\neg_I, \neg_E, \neg\neg\}$ if and only there exists $M \in \Lambda^{\rightarrow}_{ext}$ such that $\vdash M : \mathcal{C}(P)$.

$\mathcal{N}'$ is called the **negation free** fragment of natural deduction.

Practical Application

Exercise 1. Prove that $\vdash_{\mathcal{N}'} A \wedge B \Rightarrow A$.

Answer

Consider the proof tree:

$$\dfrac{\dfrac{\dfrac{}{A \wedge B}\,1}{A}\,[\wedge^I_E]}{A \wedge B \Rightarrow A}\,[\Rightarrow_I]^1$$

Practical Application

Exercise 2. Find a term in Λ_{ext} of type $\sigma \times \tau \rightarrow \sigma$.

Answer

Consider the **isomorphic** type derivation:

$$\dfrac{\dfrac{\dfrac{}{x : \sigma \times \tau}\,1}{\Pi_1(x) : \sigma}\,[\times_E^I]}{\lambda x \cdot \Pi_1(x) : \sigma \times \tau \to \sigma}\,[\lambda]^1$$

Practical Application

Exercise 3. Prove that $\vdash_{\mathcal{N}'} A \wedge B \Rightarrow A \vee B$, then deduce that the type $\sigma \times \tau \to \sigma \cup \tau$ is inhabited and show a term M of the aforementioned type.

Answer I

Consider the proof tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge B}\,1}{A}\,[\wedge_E^I]}{A \vee B}\,[\vee_I^I]}{A \wedge B \Rightarrow A \vee B}\,[\Rightarrow_I]^1$$

Answer II

Consider the **isomorphic** type derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : \sigma \times \tau}\,1}{\Pi_1(x) : \sigma}\,[\times_E^I]}{K_1(\Pi_1(x)) : \sigma \cup \tau}\,[\cup_I^I]}{\lambda x \cdot K_1(\Pi_1(x)) : \sigma \times \tau \to \sigma \cup \tau}\,[\lambda]^1$$

Extending the Isomorphism

A summary

Logic	**λ-calculus**	**Functional programming**
Proof	Typable term	Halting program
Cut elimination	Reduction	Execution step
Normalization	Normal form	Value
Formula	Type	Interface
$\Rightarrow$	Functional type	Functions
$\wedge$	$\times$	Pairs
$\vee$	$\cup$	Unions

Optional Homework

- Exercises **8A** and **8B** of the 2019-2020 exam.

That's all folks!