2. First projection, $\pi_1 \langle M, N \rangle \longrightarrow_\beta M$.
3. Second projection, $\pi_2 \langle M, N \rangle \longrightarrow_\beta N$.
4. Case rule, (case m of $[a].N$; $[b].P$) $\longrightarrow_\beta Na$ if m is of the form $m = \mathrm{inl}\ a$; and (case m of $[a].N$; $[b].P$) $\longrightarrow_\beta$ Pb if m is of the form $m = \mathrm{inr}\ b$.

On the other side, new η-rules are defined, each for every new construction rule.

1. Function extensionality, $\lambda x.Mx \longrightarrow_\eta M$.
2. Definition of product, $\langle \pi_1 M, \pi_2 M \rangle \longrightarrow_\eta M$.
3. Uniqueness of unit, $M \longrightarrow_\eta *$.
4. Case rule, (case m of $[a].P[\mathrm{inl}\ a/c]$; $[b].P[\mathrm{inr}\ b/c]$) $\longrightarrow_\eta P[m/c]$.

3.2 NATURAL DEDUCTION

The natural deduction is a logical system due to Gentzen. We introduce it here following [Sel13] and [Wad15]. Its relationship with the simply-typed lambda calculus will be made explicit in the next section.

We will use the logical binary connectives $\longrightarrow, \wedge, \vee$, and two given propositions, $\top, \bot$ representing the trivially true and false propostiions, respectively. The rules defining natural deduction come in pairs; there are introductors and eliminators for every connective. Every introductor uses a set of assumptions to generate a formula and every eliminator gives a way to extract precisely that set of assumptions.

1. Every axiom on the context can be used.

$$\frac{}{\Gamma, A \vdash A}\ (\mathrm{Ax})$$

2. Introduction and elimination of the $\longrightarrow$ connective. Note that the elimination rule corresponds to *modus ponens* and the introduction rule corresponds to the *deduction theorem*.

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \longrightarrow B}\ (I_{\longrightarrow}) \qquad \frac{\Gamma \vdash A \longrightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ (E_{\longrightarrow})$$

3. Introduction and elimination of the $\wedge$ connective. Note that the introduction in this case takes two assumptions, and there are two different elimination rules.

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ (I_\wedge) \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ (E^1_\wedge) \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ (E^2_\wedge)$$

4. Introduction and elimination of the $\vee$ connective. Here, we need two introduction rules to match the two assumptions we use on the eliminator.

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ (I^1_\vee) \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ (I^2_\vee) \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\ (E_\vee)$$

5. Introduction for $\top$. It needs no assumptions and, consequently, there is no elimination rule for it.

$$\frac{}{\Gamma \vdash \top}\ (I_\top)$$

6. Elimination for $\bot$. It can be eliminated in all generality, and, consequently, there are no introduction rules for it. This elimination rule represents the *"ex falsum quodlibet"* principle that says that falsity implies anything.

$$\frac{\Gamma \vdash \bot}{\Gamma \vdash C}\ (E_\bot)$$

Proofs on natural deduction are written as deduction trees, and they can be simplified according to some simplification rules, which can be applied anywhere on the deduction tree. On these rules, a chain of dots represents any given part of the deduction tree.

1. An implication and its antecedent can be simplified using the antecedent directly on the implication.

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}[A]\\ \vdots^1\\ B\end{array}}{A \longrightarrow B} \qquad \begin{array}{c}\vdots^2\\ A\end{array}}{B}\\ \vdots\end{array} \quad\Longrightarrow\quad \begin{array}{c}\vdots^2\\ A\\ \vdots^1\\ \dfrac{B}{\vdots}\end{array}$$

2. The introduction of an unused conjunction can be simplified as

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\vdots^1\\ A\end{array} \quad \begin{array}{c}\vdots^2\\ B\end{array}}{A \wedge B}}{A}\\ \vdots\end{array} \quad\Longrightarrow\quad \begin{array}{c}\vdots^1\\ \dfrac{A}{\vdots}\end{array}$$

and, similarly, on the other side as

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\vdots^1\\ A\end{array} \quad \begin{array}{c}\vdots^2\\ B\end{array}}{A \wedge B}}{B}\\ \vdots\end{array} \quad\Longrightarrow\quad \begin{array}{c}\vdots^2\\ \dfrac{B}{\vdots}\end{array}$$

3. The introduction of a disjunction followed by its elimination can be also simplified

$$\begin{array}{c}\dfrac{\dfrac{\begin{array}{c}\vdots^1\\ A\end{array}}{A \vee B} \qquad \begin{array}{c}[A]\\ \vdots^2\\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots^3\\ C\end{array}}{C}\\ \vdots\end{array} \quad\Longrightarrow\quad \begin{array}{c}\vdots^1\\ A\\ \vdots^2\\ \dfrac{C}{\vdots}\end{array}$$

and a similar pattern is used on the other side of the disjunction

$$\dfrac{\dfrac{\begin{array}{c}\vdots_1\\ B\end{array}}{A \vee B} \qquad \begin{array}{c}[A]\\ \vdots_2\\ C\end{array} \qquad \begin{array}{c}[B]\\ \vdots_3\\ C\end{array}}{\begin{array}{c}C\\ \hline \vdots\end{array}} \qquad \Longrightarrow \qquad \begin{array}{c}\vdots_1\\ B\\ \vdots_3\\ C\\ \hline \vdots\end{array}$$

3.3 PROPOSITIONS AS TYPES

In 1934, Curry observed in [Cur34] that the type of a function ($A \longrightarrow B$) could be read as an implication and that the existence of a function of that type was equivalent to the provability of the proposition. Previously, the **Brouwer-Heyting-Kolmogorov interpretation** of intuitionistic logic had given a definition of what it meant to be a proof of an intuinistic formula, where a proof of the implication ($A \longrightarrow B$) was a function converting a proof of A into a proof of B. It was not until 1969 that Howard pointed a deep correspondence between the simply-typed λ-calculus and the natural deduction at three levels

1. propositions are types.
2. proofs are programs.
3. simplification of proofs is the evaluation of programs.

In the case of simply typed λ-calculus and natural deduction, the correspondence starts when we describe the following one-to-one relation between types and propositions.

Types	Propositions
Unit type (1)	Truth ($\top$)
Product type ($\times$)	Conjunction ($\wedge$)
Union type ($+$)	Disjunction ($\vee$)
Function type ($\longrightarrow$)	Implication ($\longrightarrow$)
Empty type (0)	False ($\bot$)

Where, in particular, the negation of a proposition $\neg A$ is interpreted as the fact that that proposition implies falsehood, $A \longrightarrow \bot$; and its corresponding type is a function from the type A to the empty type, $A \longrightarrow 0$.

Now it is easy to notice that every deduction rule for a proposition has a correspondence with a typing rule. The only distinction between them is the appearance of λ-terms on the first set of rules. As every typing rule results on the construction of a particular kind of λ-term, they can be interpreted as encodings of proof in the form of derivation trees. That is, terms are proofs of the propositions represented by their types.

Example 2 (Curry-Howard correspondence example). In particular, the typing derivation of the term

$$\lambda a.\lambda b.\langle a, b\rangle$$

can be seen as a deduction tree proving $A \longrightarrow B \longrightarrow A \wedge B$; as the following diagram shows, in which terms are colored in `red` and types are colored in *blue*.

$$\cfrac{\cfrac{\cfrac{\texttt{a} : A \qquad \texttt{b} : B}{\texttt{<a,b>} : A \times B}\,(pair)}{\texttt{λb.<a,b>} : B \longrightarrow A \times B}\,(abs)}{\texttt{λa.λb.<a,b>} : A \longrightarrow B \longrightarrow A \times B}\,(abs)$$

Furthermore, under this interpretation, ***simplification rules are precisely β-reduction rules***. This makes execution of λ-calculus programs correspond to proof simplification on natural deduction. The Curry-Howard correspondence is then not only a simple bijection between types and propositions, but a deeper isomorphism regarding the way they are constructed, used in derivations, and simplified.

Example 3 (Curry-Howard simplification example). As an example of this duality, we will write a proof/term of the proposition/type `A → B + A` and we are going to simplify/compute it using proof simplification rules/β-rules. Similar examples can be found in [Wad15].

We start with the following derivation tree; in which terms are colored in `red` and types are colored in *blue*

$$\cfrac{\cfrac{\cfrac{\cfrac{\texttt{b} : [A + B] \qquad \cfrac{\texttt{c} : A}{\texttt{inr c} : B + A}\,(inr) \qquad \cfrac{\texttt{c} : B}{\texttt{inl c} : B + A}\,(inl)}{\texttt{case b of [c].inr c; [c].inl c} : B + A}\,(case)}{\texttt{λb.case b of [c].inr c; [c].inl c} : A + B \longrightarrow B + A}\,(abs) \qquad \cfrac{\texttt{a} : A}{\texttt{inl a} : A + B}\,(inl)}{\texttt{(λb.case b of [c].inr c; [c].inl c)(inl a)} : B + A}\,(app)}{\texttt{λa.(λb.case b of [c].inr c; [c].inl c) (inl a)} : A \longrightarrow B + A}\,(abs)$$

which is encoded by the term `λa.(λc.case c of [a].inr a; [b].inl b) (λz.inl z)`. We apply the simplification rule/β-rule of the implication/function application to get

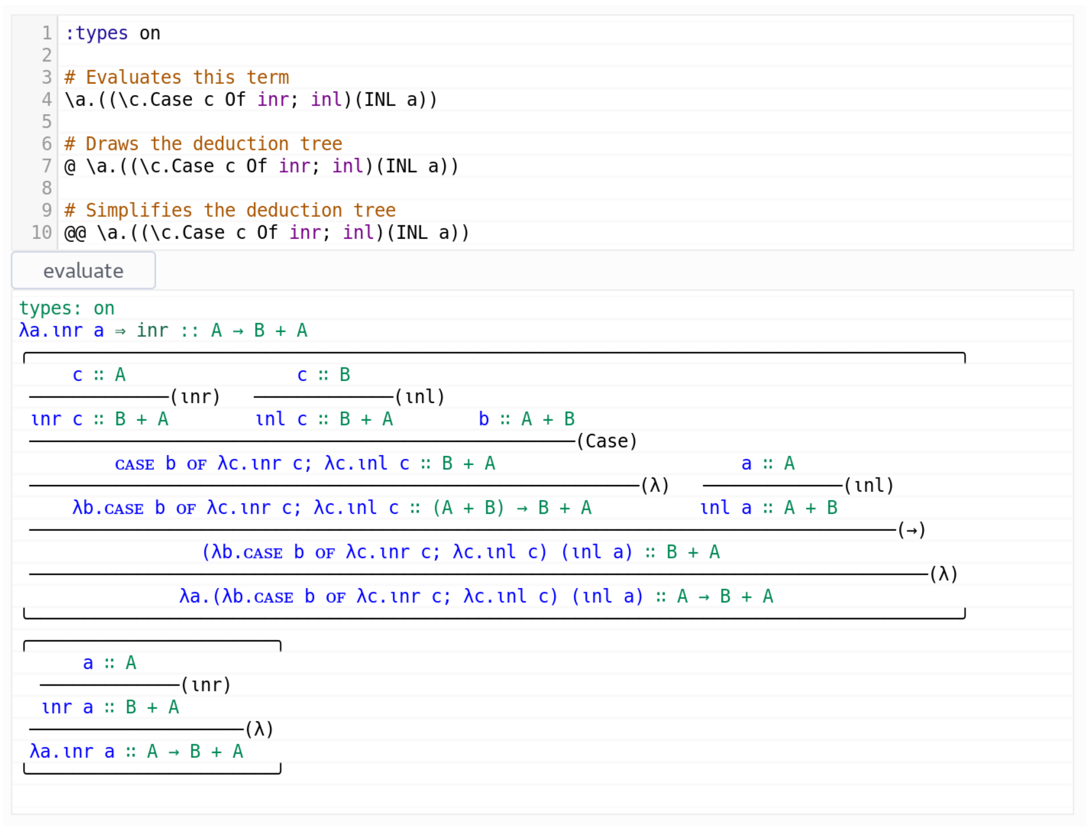
$$\cfrac{\cfrac{\cfrac{\texttt{z} : A}{\texttt{inl z} : A + B}\,(inl) \qquad \cfrac{\texttt{a} : A}{\texttt{inr a} : B + A}\,(inr) \qquad \cfrac{\texttt{b} : B}{\texttt{inl b} : B + A}\,(inl)}{\texttt{case (inl z) of [a].inr a; [b].inl b} : B + A}\,(case)}{\texttt{λ z.case (inl z) of [a].inr a; [b].inl b} : A \longrightarrow B + A}\,(abs)$$

which is encoded by the term `λa.case (inl a) of (inr) (inl)`. We finally apply the `case` simplification/reduction rule to get

$$\cfrac{\cfrac{\texttt{a} : A}{\texttt{inr a} : B + A}\,(inr)}{\texttt{λ a.inr a} : A \longrightarrow B + A}\,(abs)$$

which is encoded by `λa.(inr a)`.

On the chapter on Mikrokosmos, we develop a λ-calculus interpreter which is able to check and simplify proofs in intuitionistic logic. This example could be checked and simplified by this interpreter as it is shown in image 1.



```
:types on

# Evaluates this term
\a.((\c.Case c Of inr; inl)(INL a))

# Draws the deduction tree
@ \a.((\c.Case c Of inr; inl)(INL a))

# Simplifies the deduction tree
@@ \a.((\c.Case c Of inr; inl)(INL a))
```

```
types: on
λa.ιnr a ⇒ inr :: A → B + A
┌───────────────────────────────────────────────────────────────────────────────────────┐
    c :: A                c :: B
─────────────(ιnr)    ─────────────(ιnl)
ιnr c :: B + A        ιnl c :: B + A        b :: A + B
───────────────────────────────────────────────────────(Case)
        CASE b OF λc.ιnr c; λc.ιnl c :: B + A                      a :: A
─────────────────────────────────────────────────────────(λ)    ─────────────(ιnl)
    λb.CASE b OF λc.ιnr c; λc.ιnl c :: (A + B) → B + A          ιnl a :: A + B
─────────────────────────────────────────────────────────────────────────────────(→)
                (λb.CASE b OF λc.ιnr c; λc.ιnl c) (ιnl a) :: B + A
────────────────────────────────────────────────────────────────────────────────────(λ)
              λa.(λb.CASE b OF λc.ιnr c; λc.ιnl c) (ιnl a) :: A → B + A
└───────────────────────────────────────────────────────────────────────────────────────┘

┌───────────────────────┐
    a :: A
 ─────────────(ιnr)
 ιnr a :: B + A
───────────────────(λ)
λa.ιnr a :: A → B + A
└───────────────────────┘
```

Figure 1: Curry-Howard example in Mikrokosmos.

4

OTHER TYPE SYSTEMS

4.1 λ-CUBE

The λ-**cube** is a taxonomy for Church-style type systems given by Barendregt in [Bar92]. It describes eight type systems based on the λ-calculus along three axes, representing three properties of the systems. These properties are

1. **parametric polymorphism**, terms that depend on types. This is achieved via universal quantification over types. It allows type variables and binders for them. An example is the following parametric identity function

   $$\mathrm{id} \equiv \Lambda\tau.\lambda x.x \;:\; \forall\tau.\tau \longrightarrow \tau,$$

   that can be applied to any particular type σ to obtain the specific identity function for that type as

   $$\mathrm{id}_\sigma \equiv \lambda x.x \;:\; \sigma \longrightarrow \sigma.$$

   System F is the simplest type system on the cube implementing polymorphism.
2. **type operators**, types that depend on types.
3. **dependent types**, types that depend on terms.

$$\begin{tikzcd}
 & & & \text{System F}\omega & & \text{CoC} & \\
 & & & \lambda\omega \arrow[rr] & & \lambda\Pi\omega & \\
 {} \arrow[from=5-1, to=3-1, swap, "\text{terms depend on types}"] & \text{System F} & \lambda 2 \arrow[ur] \arrow[rr] & & \lambda\Pi 2 \arrow[ur] & & \\
 & & & \lambda\underline{\omega} \arrow[uu] \arrow[rr] & & \lambda\Pi\underline{\omega} \arrow[uu] & \text{wCoC} \\
 {} & & \lambda_{\to} \arrow[uu] \arrow[ur] \arrow[rr] & & \lambda\Pi \arrow[uu] \arrow[ur] & & \\
 & & \text{STLC} & & \text{DTLC} & & {} \\
 & & {} \arrow[rr, swap, "\text{types depend on terms}"] & & {} & {} \arrow[ur, swap, "\text{types depend on types}"] &
\end{tikzcd}$$

The following type systems

- **Simply typed λ-calculus** ($\lambda_{\to}$);
- **System F** ($\lambda 2$);
- typed λ-calculus with **dependent types** ($\lambda\Pi$);
- typed λ-calculus with **type operators** ($\lambda\underline{\omega}$);
- **System F-omega** ($\lambda\omega$);

The λ-cube is generalized by the theory of pure type systems.

All systems on the λ-cube are strongly normalizing.

A different approach to higher-order type systems will be presented in the chapter on Type Theory.

Part II

MIKROKOSMOS

We have developed **Mikrokosmos**, an untyped and simply typed λ-calculus interpreter written in the purely functional programming language Haskell [HHJW07]. It aims to provide students with a tool to learn and understand λ-calculus and the relation between logic and types.