Natural Deduction and the Curry-Howard-Isomorphism

Andreas Abel

August 2016

Abstract

We review constructive propositional logic and natural deduction and connect it to the simply-typed lambda-calculus with the Curry-Howard Isomorphism.

Constructive Logic

A fundamental property of constructive logic is the *disjunction property*:

> If the disjunction $A \vee B$ is provable, then either A is provable or B is provable.

This property is not compatible with the principle of the excluded middle (*tertium non datur*), which states that $A \vee \neg A$ holds for any proposition A. While each fool can state the classical tautology "aliens exist or don't", this certainly does not give us a means to decide whether aliens exist or not. A constructive proof of the fool's statement would require either showcasing an alien or a stringent argument for the impossibility of their existence.[1]

1 Natural deduction for propositional logic

The proof calculus of natural deduction goes back to Gentzen [1935].

[1] For a more mathematical example of undecidability, refer to the *continuum hypothesis* CH which states that no cardinal exists between the set of the natural numbers and the set of reals. It is independent of ZFC, Zermelo-Fränkel set theory with the axiom of choice, meaning that in ZFC, neither CH nor $\neg\mathsf{CH}$ is provable.

1.1 Propositions

Formulæ of propositional logic are given by the following grammar:

$$
\begin{array}{lll}
P, Q & & \text{atomic proposition} \\
A, B, C ::= & P & \\
& \mid A \Rightarrow B & \text{implication} \\
& \mid A \wedge B \mid \top & \text{conjunction, truth} \\
& \mid A \vee B \mid \bot & \text{disjunction, absurdity}
\end{array}
$$

Even though we write formulas in linearized (string) form, we think of them as (abstract syntax) *trees*. Formulas of propositional logic are binary trees where the inner nodes are labelled with one of the connectives $\Rightarrow$ or $\wedge$ or $\vee$, and leafs with a generic atomic proposition P or with $\top$ or $\bot$. The label of the root of such a tree is also called the *principal connective* of the formula. Any subtree of a formula A is called a *subformula* of A. The children of the root node are called the *immediate subformulæ*.

Example 1 (Propositions)**.** Given

$$
\begin{array}{lcl}
\mathsf{SH} & := & \text{“Sokrates is a human”} \\
\mathsf{FL} & := & \text{“Sokrates has four legs”}
\end{array}
$$

we can form the implication $\mathsf{SH} \Rightarrow \mathsf{FL}$ verbalized as “if Sokrates is a human, he has four legs”.

This example presents a well-formed proposition, but its truth is not without conditions. It is important to clearly separate between *well-formedness* and *truth*. However, speaking about truth usually presupposes well-formedness; discussions about the truth of ill-formed propositions maybe be entertaining after sufficient consumption of good wine, though. While well-formedness of propositional formulas is simply adherence to a simple grammar, truth requires more sophisticated tools, such as *proof*.

1.2 Judgements and derivations

Definition 1. A proof of a proposition A is a derivation of the judgement

$$A \ \mathit{true}$$

by the proof rules of propositional logic (to follow).

In general, we refer to *judgements* such as A *true* with letter J. A *rule* r has the following form

$$\frac{J_1 \ldots J_n}{J}\ r$$

where $J_{1..n}$ (with $n \geq 0$) are the *premises* and J is the *conclusion*. Informally, it means that if we can make all the judgements J_i, then we can also make judgements J, by virtue of applying rule r. Rules without premises $n = 0$ are called *axioms*.

Subsequent rule applications are organized into a *tree*, for instance:

$$\dfrac{\dfrac{}{J_1}\, r_1 \qquad \dfrac{\dfrac{}{J_3}\, r_3 \qquad J_4 \qquad J_5}{J_2}\, r_2}{J_0}\, r_0$$

In this case, the final judgement J_0 is justified by a binary rule r_0 whose first premise J_1 is justified by axiom r_1. Its second premise J_2 is justified by a ternary rule r_2. The first of these new three premises J_3 is justified by axiom r_3, the other two premises J_4 and J_5 are missing justification.

The tree leading to judgement J_0 is called the *derivation* of J_0. We use letters $\mathcal{D}$ and $\mathcal{E}$, to refer to derivations and write $\boxed{\mathcal{D} :: J}$ for "tree $\mathcal{D}$ is a derivation of judgement J". If all leaves of the derivation trees are axioms, we speak of a *closed derivation*, otherwise of an *open derivation*, like the example above. Suppose we find justifications $\mathcal{D}_4 :: J_4$ and $\mathcal{D}_5 :: J_5$ of the remaining open premises, then we can close the derivation of J_0 like this:

$$\dfrac{\dfrac{}{J_1}\, r_1 \qquad \dfrac{\dfrac{}{J_3}\, r_3 \qquad \mathcal{D}_4 :: J_4 \qquad \mathcal{D}_5 :: J_5}{J_2}\, r_2}{J_0}\, r_0$$

The whole derivation $\mathcal{D}_0 :: J_0$ can now be written in prefix notation as

$$\mathcal{D}_0 = r_0^{J_0}(r_1^{J_1}, r_2^{J_2}(r_3^{J_3}, \mathcal{D}_4, \mathcal{D}_5))$$

where the superscripts annotate each rule with the judgement it derives, or, dropping annotations, briefly as $\mathcal{D}_0 = r_0(r_1, r_2(r_3, \mathcal{D}_4, \mathcal{D}_5))$.

1.3 Introduction and elimination rules

The judgement A *true* for a propositional formula A is established by rules that fall into two classes:

1. Introduction rules. These establish the truth of a compound proposition, for instance conjuction $A \wedge B$ from the truth of its subformulas A and B.

$$\dfrac{A \text{ true} \qquad B \text{ true}}{A \wedge B \text{ true}}\, \wedge\mathsf{I}$$

2. Elimination rules. These allow us to use (parts of) a true compound proposition to derive the truth of other propositions. For instance, the truth of $A \wedge B$ entails the truth of its subformulas A and B.

$$\frac{A \wedge B \ \mathit{true}}{A \ \mathit{true}} \wedge\mathsf{E}_1 \qquad \frac{A \wedge B \ \mathit{true}}{B \ \mathit{true}} \wedge\mathsf{E}_2$$

Connectives that can be defined with introduction and elimination rules referring only to subformulas (or generic new formulas) are called *orthogonal*.

Introduction and elimination rules need to fit together. Introduction rules allow us to assemble evidence, and elimination rules allow us to use it, or to disassemble it. The elimination rules for a connective are adequate if we can retrieve by the elimination rules every piece of information we put in by introduction rules, and only the information we put in. Adequacy can be formulated as *local soundness* and *local completeness*, i. e., the presence of certain proof transformations detailed as follows.

Local soundness states that we can simplify any proof detour that consists of the introduction of a connective immediately followed by the connective. There are $n \cdot m$ different detour shapes for a connective with n introduction rules and m elimination rules. The removal of such a detour is also called β-reduction, sometimes qualified by the connective, e. g., $\beta_\wedge$-reduction.

In case of conjunction, there are two possible detours that qualify for β-reduction:

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ A \ \mathit{true} & B \ \mathit{true}\end{array}}{A \wedge B \ \mathit{true}} \wedge\mathsf{I}}{\dfrac{A \wedge B \ \mathit{true}}{A \ \mathit{true}} \wedge\mathsf{E}_1} \qquad \dfrac{\dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ A \ \mathit{true} & B \ \mathit{true}\end{array}}{A \wedge B \ \mathit{true}} \wedge\mathsf{I}}{\dfrac{A \wedge B \ \mathit{true}}{B \ \mathit{true}} \wedge\mathsf{E}_2}$$

Both pieces of evidence (A *true* and B *true*) fed into the introduction rule $\wedge\mathsf{I}$ can be recovered by the elimination rules $\wedge\mathsf{E}_1$ and $\wedge\mathsf{E}_2$. The β-reductions simplify these proof trees as follows:

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ A \ \mathit{true} & B \ \mathit{true}\end{array}}{A \wedge B \ \mathit{true}} \wedge\mathsf{I}}{\dfrac{A \wedge B \ \mathit{true}}{A \ \mathit{true}} \wedge\mathsf{E}_1} \longrightarrow_\beta \quad \begin{array}{c}\mathcal{D}_1 \\ A \ \mathit{true}\end{array}$$

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ A \ \mathit{true} & B \ \mathit{true}\end{array}}{A \wedge B \ \mathit{true}} \wedge\mathsf{I}}{\dfrac{A \wedge B \ \mathit{true}}{B \ \mathit{true}} \wedge\mathsf{E}_2} \longrightarrow_\beta \quad \begin{array}{c}\mathcal{D}_2 \\ B \ \mathit{true}\end{array}$$

These reductions show that the eliminations $\wedge\mathsf{E}_{1/2}$ are not too strong: they do not produce more than we put in with the introduction rule $\wedge\mathsf{I}$. This is why we can cut the detours out.

Exercise 1. Invent a (locally) unsound elimination rule for conjunction, which does not allow the reduction of detours.

Local completeness in contrast expresses that the elimination rules are not too weak. The elimination rules for a connective should give us all the evindence such that we can reconstruct a proof for the formula we eliminated. This means that any proof for a proposition should be presentable by introducing the principal connective of the proposition. The premises for the introduction rule(s) are constructed via the elimination rule. Local completeness is witnesses by *η-expansion*, a proof transformation that introduces a detour of the form *elimination rule followed by introduction rules*.

For conjunction, η-expansion creates the following detour:

$$\begin{array}{c}\mathcal{D}\\ A \wedge B\ \mathit{true}\end{array} \quad \longrightarrow_{\eta^-} \quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\ \mathit{true}\end{array}}{A\ \mathit{true}}\wedge\mathsf{E}_1 \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\ \mathit{true}\end{array}}{B\ \mathit{true}}\wedge\mathsf{E}_2}{A \wedge B\ \mathit{true}}\wedge\mathsf{I}$$

Exercise 2. Reflect on the fate of local completeness for conjunction when we drop one of the elimination rules.

1.4 Trivial proposition

With binary conjunction, we can express any finite conjunction except for the empty conjunction $\top$. A proof of the empty conjunctions asks us to provide proofs for all of the 0 conjuncts. Thus, we get an introduction rule without premises.

$$\frac{}{\top\ \mathit{true}}\ \top\mathsf{I}$$

There is no elimination rule for $\top$: A proof of $\top$ is constructed from nothing, so there is no information contained in it we could retrieve by elimination.

As a consequence, there is no β-reduction. Local completeness is witnessed by the η-expansion:

$$\begin{array}{c}\mathcal{D}\\ \top\ \mathit{true}\end{array} \quad \longrightarrow_{\eta^-} \quad \frac{}{\top\ \mathit{true}}\ \top\mathsf{I}$$

This proof transformation replaces any derivation of the trivial proposition by the trivial derivation.

1.5 Hypothetical judgements

The elimination rule for implication is the well-known *modes ponens*: If A implies B and A holds, then B must hold as well.

$$\frac{A \Rightarrow B \ \mathit{true} \qquad A \ \mathit{true}}{B \ \mathit{true}} \Rightarrow\mathsf{E}$$

While, classically, implication $A \Rightarrow B$ can be reduced to disjunction and negation $\neg A \vee B$, the disjunction property would be immediately violated by such an interpretation. A universal truth such as "if aliens exist, there must a be habitable planet" does not allow us to decide whether there is a habitable planet or no aliens exist.

Instead the implication $A \Rightarrow B$ allows us to conclude B from the *hypothesis* A. This means that we can assume the truth of A to derive the truth of B. Let us consider the tautology $(A \wedge B) \Rightarrow (B \wedge A)$. Our proof would first derive the conclusion $B \wedge A$ from an assumed truth of $A \wedge B$, by the following open derivation:

$$\dfrac{\dfrac{A \wedge B \ \mathit{true}}{B \ \mathit{true}} \qquad \dfrac{A \wedge B \ \mathit{true}}{A \ \mathit{true}}}{B \wedge A \ \mathit{true}}$$

Then, the introduction of the implication $(A \wedge B) \Rightarrow (B \wedge A)$ enables us to discharge the hypothesis $A \wedge B$ and close the derivation.

$$\dfrac{\dfrac{\dfrac{\overline{\phantom{A \wedge B \ \mathit{true}}}\,x}{A \wedge B \ \mathit{true}}}{B \ \mathit{true}} \qquad \dfrac{\dfrac{\overline{\phantom{A \wedge B \ \mathit{true}}}\,x}{A \wedge B \ \mathit{true}}}{A \ \mathit{true}}}{\dfrac{B \wedge A \ \mathit{true}}{(A \wedge B) \Rightarrow (B \wedge A) \ \mathit{true}} \Rightarrow\mathsf{I}_x}$$

We label the discharged hypothesis with x in the leaves of the derivation and in the instance of the implication introduction rule $\Rightarrow\mathsf{I}_x$, in order to keep track which hypothesis is discharged by which application of implication introduction.

The generic form of $\Rightarrow\mathsf{I}$ is:

$$\dfrac{\begin{array}{c}\overline{\phantom{A \ \mathit{true}}}\,x \\ \vdots \\ B \ \mathit{true}\end{array}}{A \Rightarrow B \ \mathit{true}} \Rightarrow\mathsf{I}_x$$

It turns the meta-implication *from A true we can derive B true* into a proof of propositional implication. Such a meta-implication is also called a *hypothetical judgement*, a judgement under hypotheses.

Exercise 3.

1. Derive $A \Rightarrow A\ \mathit{true}$.
2. Construct different derivations of $A \Rightarrow (A \Rightarrow A)\ \mathit{true}$. How many are there?
3. Is the following derivation valid?

$$\dfrac{\dfrac{\dfrac{}{(A \Rightarrow A) \Rightarrow (A \Rightarrow A)\ \mathit{true}}\, f \quad \dfrac{\dfrac{}{A\ \mathit{true}}\, x}{A \Rightarrow A\ \mathit{true}} \Rightarrow\mathsf{I}_x}{A \Rightarrow A\ \mathit{true}} \Rightarrow\mathsf{E} \quad \dfrac{}{A\ \mathit{true}}\, x}{A\ \mathit{true}} \Rightarrow\mathsf{E}$$
$$\dfrac{}{((A \Rightarrow A) \Rightarrow (A \Rightarrow A)) \Rightarrow A\ \mathit{true}} \Rightarrow\mathsf{I}_f$$

Local completeness for implication is witnessed by the following η-expansion:

$$\begin{array}{c}\mathcal{D}\\ A \Rightarrow B\ \mathit{true}\end{array} \qquad \longrightarrow_{\eta^-} \qquad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A \Rightarrow B\ \mathit{true}\end{array} \qquad \dfrac{}{A\ \mathit{true}}\, x}{B\ \mathit{true}} \Rightarrow\mathsf{E}}{A \Rightarrow B\ \mathit{true}} \Rightarrow\mathsf{I}_x$$

Local soundness requires us to replace a hypothetical derivation, i. e., a discharged hypothesis, by an actual derivation:

$$\dfrac{\dfrac{\begin{array}{c}\dfrac{}{A\ \mathit{true}}\, x\\ \vdots\, \mathcal{E}\\ B\ \mathit{true}\end{array}}{A \Rightarrow B\ \mathit{true}} \Rightarrow\mathsf{I}_x \qquad \begin{array}{c}\mathcal{D}\\ A\ \mathit{true}\end{array}}{B\ \mathit{true}} \Rightarrow\mathsf{E} \qquad \longrightarrow_{\beta} \qquad \begin{array}{c}\mathcal{D}\\ A\ \mathit{true}\\ \vdots\, \mathcal{E}\\ B\ \mathit{true}\end{array}$$

In derivation $\mathcal{E}$, we need to replace every use of hypothesis x by derivation $\mathcal{D}$. A more precise (but less suggestive) notation for the reduct is:

$$\begin{array}{c}\vdots\, \mathcal{E}[\mathcal{D}/x]\\ B\ \mathit{true}\end{array}$$

1.6 Disjunction

A disjunction $A \vee B$ is introduced by deriving any of its disjuncts.

$$\dfrac{A\ \mathit{true}}{A \vee B\ \mathit{true}} \vee\mathsf{I}_1 \qquad \dfrac{B\ \mathit{true}}{A \vee B\ \mathit{true}} \vee\mathsf{I}_2$$

A disjunction is eliminated by *case distinction*. To use $A \vee B\ true$ in the derivation of a proposition C we need to consider the two cases $A\ true$ and $B\ true$. The intuition is that if A implies C and B implies C then $A \vee B$ implies C. We could put this into a rule directly:

$$\frac{A \Rightarrow C\ true \qquad B \Rightarrow C\ true}{(A \vee B) \Rightarrow C\ true}$$

But this violates orthogonality. The rule uses formulas which are not subformulas. And disjunction is not defined independent of implication. Instead we use hypothetical judgements:

$$\frac{A \vee B\ true \qquad \begin{array}{c}\overline{}\,x \\ A\ true \\ \vdots \\ C\ true\end{array} \qquad \begin{array}{c}\overline{}\,y \\ B\ true \\ \vdots \\ C\ true\end{array}}{C\ true}\ \vee\mathsf{E}_{x,y}$$

Exercise 4. Derive the following judgements:

1. $(A \vee B) \Rightarrow (B \vee A)\ true$ (commutativity of disjunction).
2. $((A \vee B) \wedge C) \Rightarrow ((A \wedge C) \vee (B \wedge C))\ true$ (distributivity).

Local soundness again requires substitution of derivations:

$$\frac{\dfrac{\mathcal{D}}{A\ true}\vee\mathsf{I}_1 \qquad \begin{array}{c}\overline{}\,x \\ A\ true \\ \vdots \\ \mathcal{E}_1 \\ \vdots \\ C\ true\end{array} \qquad \begin{array}{c}\overline{}\,y \\ B\ true \\ \vdots \\ \mathcal{E}_2 \\ \vdots \\ C\ true\end{array}}{C\ true}\ \vee\mathsf{E}_{x,y} \quad \longrightarrow_\beta \quad \begin{array}{c}\mathcal{E}_1[\mathcal{D}/x] \\ \vdots \\ C\ true\end{array}$$

$$\frac{\dfrac{\mathcal{D}}{B\ true}\vee\mathsf{I}_2 \qquad \begin{array}{c}\overline{}\,x \\ A\ true \\ \vdots \\ \mathcal{E}_1 \\ \vdots \\ C\ true\end{array} \qquad \begin{array}{c}\overline{}\,y \\ B\ true \\ \vdots \\ \mathcal{E}_2 \\ \vdots \\ C\ true\end{array}}{C\ true}\ \vee\mathsf{E}_{x,y} \quad \longrightarrow_\beta \quad \begin{array}{c}\mathcal{E}_2[\mathcal{D}/y] \\ \vdots \\ C\ true\end{array}$$

Local completeness can only be formulated with elimination preformed before introduction, because we have two introduction rules, and we cannot *a priori* decide which of these to use:

$$\begin{array}{c}\mathcal{D} \\ A \vee B\ true\end{array} \quad \longrightarrow_{\eta^-} \quad \frac{\mathcal{D} \qquad \dfrac{\overline{}\,x}{A \vee B\ true}\vee\mathsf{I}_1 \qquad \dfrac{\overline{}\,y}{A \vee B\ true}\vee\mathsf{I}_2}{A \vee B\ true}\ \vee\mathsf{E}_{x,y}$$

There is a stronger version of this proof transformation which let us eliminate an arbitrary occurrence of a disjunction at the root of derivation:

$$\begin{array}{c} \mathcal{D} \\ A \vee B \ \mathit{true} \\ \vdots\, \mathcal{E} \\ C \ \mathit{true} \end{array} \quad \Longrightarrow_\eta \quad \dfrac{\mathcal{D}\quad \dfrac{\dfrac{}{A \ \mathit{true}}\, x}{A \vee B \ \mathit{true}} \vee\mathsf{I}_1 \quad \begin{array}{c} \\ \vdots\, \mathcal{E} \\ C \ \mathit{true} \end{array} \quad \dfrac{\dfrac{}{B \ \mathit{true}}\, y}{A \vee B \ \mathit{true}} \vee\mathsf{I}_2 \quad \begin{array}{c} \\ \vdots\, \mathcal{E} \\ C \ \mathit{true} \end{array}}{A \vee B \ \mathit{true} \qquad C \ \mathit{true}} \vee\mathsf{E}_{x,y}$$

However, this transformation is only sound if $\mathcal{D}$ does not use any hypotheses that are introduced during course of derivation $\mathcal{E}$ (via $\Rightarrow\mathsf{I}$ and $\vee\mathsf{E}$). We can make this precised by requiring that the derivation to transform arises from substituting wellformed derivation $\mathcal{D}$ for an hypothesis x. Let

$$\begin{array}{c} \mathcal{D} \\ A \vee B \ \mathit{true} \end{array}$$

then

$$\begin{array}{c} \\ \vdots\, \mathcal{E}[\mathcal{D}/x] \\ C \ \mathit{true} \end{array} \quad \Longrightarrow_\eta \quad \dfrac{\mathcal{D}\quad \dfrac{\dfrac{}{A \ \mathit{true}}\, x}{A \vee B \ \mathit{true}} \vee\mathsf{I}_1 \quad \begin{array}{c} \\ \vdots\, \mathcal{E} \\ C \ \mathit{true} \end{array} \quad \dfrac{\dfrac{}{B \ \mathit{true}}\, x}{A \vee B \ \mathit{true}} \vee\mathsf{I}_2 \quad \begin{array}{c} \\ \vdots\, \mathcal{E} \\ C \ \mathit{true} \end{array}}{A \vee B \ \mathit{true} \qquad C \ \mathit{true}} \vee\mathsf{E}_{x,x}$$

Exercise 5 (Strong η)**.** Find two derivations $\mathcal{D}$ and $\mathcal{E}$ such that $\mathcal{D} \Longrightarrow_\eta \mathcal{E} \Longrightarrow_\eta \mathcal{D}$.

1.7 Absurdity

Falsehood or *absurdity* $\bot$ is the empty disjunction. We have 0 introduction rules and for elimination, we have to consider 0 cases:

$$\frac{\bot \ \mathit{true}}{C \ \mathit{true}}\, \bot\mathsf{E}$$

This rule is also known as *ex falsum quod libet*.

Again there is no β-reduction, but we have strong η-expansion. Let

$$\begin{array}{c} \mathcal{D} \\ \bot \ \mathit{true} \end{array}$$

then

$$\begin{array}{c} \vdots \\ \mathcal{E}[\mathcal{D}/x] \\ \vdots \\ C\ \mathit{true} \end{array} \quad \Longrightarrow_\eta \quad \dfrac{\begin{array}{c}\mathcal{D} \\ \bot\ \mathit{true}\end{array}}{C\ \mathit{true}}\ \bot\mathsf{E}$$

A use of strong η might let us shorten a complicated proof of C to one directly using absurdity elimination.

Exercise 6 (Strong η for absurdity)**.**

- First derive $\bot \Rightarrow ((\bot \Rightarrow C) \Rightarrow C)$ *true* via modus ponens. Then apply η.
- Find a derivation $\mathcal{D}$ that gets bigger by strong η-expansion for $\bot$.

1.8 Notational definitions

Negation $\neg A$ can be defined via implication and absurdity:

$$\neg A = A \Rightarrow \bot$$

We consider $\neg A$ as just an abbreviation or notation for $A \Rightarrow \bot$.

Similarily, logical equivalence $A \iff B$ can be defined via implication and conjunction:

$$A \iff B = (A \Rightarrow B) \wedge (B \Rightarrow A)$$

Exercise 7. Prove the following tautologies:

1. $\neg(A \wedge \neg A)$
2. $A \Rightarrow \neg\neg A$
3. $\neg\neg\neg A \iff \neg A$
4. $(\neg A \vee B) \Rightarrow (A \Rightarrow B)$
5. $(\neg A \wedge \neg B) \iff \neg(A \vee B)$
6. $(\neg A \vee \neg B) \Rightarrow \neg(A \wedge B)$

1.9 Summary

Figure 1 summarizes the inference rules for constructive propositional logic.

$\boxed{A\ true}$ "Proposition A is true".

Implication.

$$\dfrac{\begin{array}{c}\overline{A\ true}\,x\\ \vdots\\ B\ true\end{array}}{A \Rightarrow B\ true}\Rightarrow\mathsf{I}_x \qquad \dfrac{A \Rightarrow B\ true \qquad A\ true}{B\ true}\Rightarrow\mathsf{E}$$

Truth and conjunction.

$$\dfrac{A\ true \qquad B\ true}{A \wedge B\ true}\wedge\mathsf{I} \qquad \dfrac{A \wedge B\ true}{A\ true}\wedge\mathsf{E}_1 \qquad \dfrac{A \wedge B\ true}{B\ true}\wedge\mathsf{E}_2$$

Absurdity and disjunction.

$$\dfrac{A\ true}{A \vee B\ true}\vee\mathsf{I}_1 \qquad \dfrac{B\ true}{A \vee B\ true}\vee\mathsf{I}_2$$

$$\dfrac{A \vee B\ true \qquad \begin{array}{c}\overline{A\ true}\,x\\ \vdots\\ C\ true\end{array} \qquad \begin{array}{c}\overline{B\ true}\,y\\ \vdots\\ C\ true\end{array}}{C\ true}\vee\mathsf{E}_{x,y}$$

Truth and absurdity.

$$\dfrac{}{\top\ true}\top\mathsf{I} \qquad \dfrac{\bot\ true}{C\ true}\bot\mathsf{E}$$

Figure 1: Summary: Natural deduction for propositional logic.

2 Explicit hypotheses

We have presented the introduction of hypotheses via open derivations that were closed by the discharging of hypotheses. This is motivated by the view of implication internalizing a meta-implication into propositional logic. However, the handling of hypotheses can be error-prone when large proof trees are involved. Thus we consider a more explicit and verbose, but more local handling of hypotheses. We introduce the *hypothetical judgement*

$$A_1 \ \mathit{true}, \ldots, A_n \ \mathit{true} \vdash C \ \mathit{true}$$

stating "under the assumptions of the truth of $A_{1..n}$, proposition C is true". We now may also drop the prefix *true* since there is no confusion between a proposition C and its truth

$$A_1, \ldots, A_n \vdash C.$$

Finally, we let Γ denote a list of hypotheses $A_{1..n}$ and write $A_i \in \Gamma$ if $1 \leq i \leq n$.

Application of a hypothesis now becomes an explicit rule invokation.

$$\frac{A \in \Gamma}{\Gamma \vdash A}\ \mathsf{hyp}$$

Rules introducing hypotheses add them at the end of the list:

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \Rightarrow\!\mathsf{I} \qquad \frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee\mathsf{E}$$

All other rules leave the context unchanged. See Figure 2 for a summary of the rules.

$\boxed{\Gamma \vdash A}$ "Under assumptions Γ, proposition A is true".

Hypotheses.

$$\frac{A \in \Gamma}{\Gamma \vdash A}\ \mathsf{hyp}$$

Implication.

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}\ \Rightarrow\!\mathsf{I} \qquad \frac{\Gamma \vdash A \Rightarrow B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \Rightarrow\!\mathsf{E}$$

Truth and conjunction.

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \wedge B}\ \wedge\mathsf{I} \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\mathsf{E}_1 \qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\mathsf{E}_2$$

Absurdity and disjunction.

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\mathsf{I}_1 \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\mathsf{I}_2$$

$$\frac{\Gamma \vdash A \vee B \qquad \Gamma, A \vdash C \qquad \Gamma, B \vdash C}{\Gamma \vdash C}\ \vee\mathsf{E}$$

Truth and absurdity.

$$\frac{}{\Gamma \vdash \top}\ \top\mathsf{I} \qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash C}\ \bot\mathsf{E}$$

Figure 2: Inference rules with explicit hypotheses.

3 Simply typed λ-calculus

The lambda calculus [Barendregt, 1984] forms the core of most functional (aka applicative) programming languages (Scheme, SML, ocaml, Haskell, etc.). The simply-typed lambda calculus goes back to Church [1940] (Simple Theory of Types).

Here, we consider the lambda-calculus with tuples and variants. It allows us to construct and apply functions, form tuples and project from them, and form alternatives and distinguish cases over them.

Grammar (lambda-calculus with tuples and variants).

x, y, z		variables
r, s, t	$::= x \mid \lambda x.t \mid r\,s$	variables, functions, applications
	$\mid \langle s, t\rangle \mid \mathsf{fst}\,r \mid \mathsf{snd}\,r$	pairs and projections
	$\mid \mathsf{inl}\,t \mid \mathsf{inr}\,t$	injections
	$\mid \mathsf{case}\,r\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow s \mid \mathsf{inr}\,y \Rightarrow t$	case distinction
	$\mid \langle\rangle$	empty tuple
	$\mid \mathsf{abort}\,r$	exception

Again we consider terms t generated by the above grammar as abstract syntax trees which we write in linear form.

The expression forms $\lambda x.t$ and $\mathsf{case}\,r\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow s \mid \mathsf{inr}\,y \Rightarrow t$ are *binders*. Term constructor $\lambda x.t$ binds variable x in the function body t. Likewise, $\mathsf{case}\,r\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow s \mid \mathsf{inr}\,y \Rightarrow t$ binds x in s and y in t.

The set of *free variables* $\mathsf{FV}(t)$ of term t is computed by recursion on t as follows:

$$\begin{array}{lcl} \mathsf{FV}(x) & = & \{x\} \\ \mathsf{FV}(\lambda x.t) & = & \mathsf{FV}(t) \setminus \{x\} \\ \mathsf{FV}(r\,s) & = & \mathsf{FV}(r) \cup \mathsf{FV}(s) \\ \ldots & & \end{array}$$

Exercise 8 (Definiton of free variables)**.** Complete the definition of FV!

The *substitution* $t[s/x]$ of term s for (free) variable x in t is defined by recursion on t as follows:

$x[s/x]$	$=$	s	
$y[s/x]$	$=$	y	if $x \neq y$
$(t\,t')[s/x]$	$=$	$(t[s/x])\,(t[s/x]')$	
$(\lambda x.t)[s/x]$	$=$	$\lambda x.t$	
$(\lambda y.t)[s/x]$	$=$	$\lambda y.t[s/x]$	if $x \neq y$ and $y \notin \mathsf{FV}(s)$
$(\lambda y.t)[s/x]$	$=$	$\lambda y'.t[y'/y][s/x]$	if $x \neq y \neq y' \neq x$ and $y' \notin \mathsf{FV}(s,t)$
$\ldots$			

The side conditions in the last two cases and the extra substitution $t[y'/y]$ in the last case prevent *variable capture*, i. e., the binding of a free variable via substitution.

Exercise 9 (Definition of substitution)**.** Complete the definition of substitution!

Exercise 10 (Variable capture)**.** Let $t\{s/x\}$ be *textual replacement* of all occurrences of variable x in t (except in binding positions like in $\lambda x.y$ which would result in the ill-formed term $\lambda s.y$). Compare the meaning of $(\lambda x.z)\{\lambda y.x/z\}$ with the one of $(\lambda x.z)[\lambda y.x/z]$.

Exercise 11 (Substitution preserves free variables)**.** Prove: $\mathsf{FV}(t[s/x]) = \mathsf{FV}(\lambda x.t) \cup \mathsf{FV}(s)$.

Exercise 12 (Swapping substitutions)**.** Prove $t[s/y][r/x] = t[r/x][s[r/x]/y]$.

Renaming a bound variable does not change the meaning of a λ-term. Two terms which only differ in the names of their bound variables (but not in their binding structure) are called α-equivalent. Formally, α-equivalence is the least congruence closed under the following axioms:

$$
\begin{array}{lcll}
\lambda x.t & =_\alpha & \lambda x'.t[x'/x] & \text{if } x' \notin \mathsf{FV}(t) \\
\mathsf{case}\, r\, \mathsf{of}\, \mathsf{inl}\, x \Rightarrow s \mid \mathsf{inr}\, y \Rightarrow t & =_\alpha & \mathsf{case}\, r\, \mathsf{of}\, \mathsf{inl}\, x' \Rightarrow s[x'/x] \mid \mathsf{inr}\, y \Rightarrow t & \text{if } x' \notin \mathsf{FV}(s) \\
\mathsf{case}\, r\, \mathsf{of}\, \mathsf{inl}\, x \Rightarrow s \mid \mathsf{inr}\, y \Rightarrow t & =_\alpha & \mathsf{case}\, r\, \mathsf{of}\, \mathsf{inl}\, x \Rightarrow s \mid \mathsf{inr}\, y' \Rightarrow t[y'/y] & \text{if } y' \notin \mathsf{FV}(t)
\end{array}
$$

On paper, we identify α-equivalent terms and silently rename bound variables if they conflict with names of free variables. *A posteriori*, modulo α-equivalence, substitution under λ could be defined by the single clause

$$(\lambda y.t)[s/x] \quad = \quad \lambda y.t[s/x] \qquad \text{if } x \neq y \text{ and } y \notin \mathsf{FV}(s)$$

since the side conditions can always be satisfied by renaming y to a fresh variable y'.

3.1 Type assignment

The lambda-calculus in its untyped form allows terms like $\lambda x.\, x\, x$ "given a function x apply it to itself" which are hard to make sense of, and even nonsensical terms such as $\mathsf{fst}\, (\lambda x.x)$ "the first projection of the identity function" or $(\mathsf{inl}\, t)\, s$ "the left injection of t applied to argument s". Such nonsense can be excluded by simple typing.[2]

Simple types are given by this grammar:

$$
\begin{array}{rcll}
R, S, T, U & ::= & S \to T & \text{function type} \\
& \mid & S \times T & \text{product type} \\
& \mid & S + T & \text{disjoint sum type} \\
& \mid & 1 & \text{unit type} \\
& \mid & 0 & \text{empty type}
\end{array}
$$

[2]However, simple typing severly limits the expressive power of lambda-calculus. The untyped lambda-calculus is Turing-complete, while the simply-typed lambda-calculus can only express polynomials with case distinction.

The purpose of an empty type 0 for programming is questionable, but we leave it for completeness. For a full-fledged programming language, we are lacking recursion (both on the type and the term level).

Let Γ be a finite map from variables to types. We write $\Gamma, x{:}T$ for the insertion or update of key x to value T. That is

$$(\Gamma, x{:}T)(y) = \begin{cases} T & \text{if } x = y \\ \Gamma(y) & \text{otherwise.} \end{cases}$$

It does not hurt to think of Γ as a list of variable-type pairs, $\Gamma = x_1 : T_1, \ldots, x_n : T_n$.

Type assignment is given by the judgement

$$\Gamma \vdash t : T$$

verbalized as "in context Γ, term t has type T", with the following inference rules:

$$\frac{\Gamma(x) = T}{\Gamma \vdash x : T}$$

$$\frac{\Gamma, x{:}S \vdash t : T}{\Gamma \vdash \lambda x.t : S \to T} \qquad \frac{\Gamma \vdash r : S \to T \qquad \Gamma \vdash s : S}{\Gamma \vdash r\, s : T}$$

$$\frac{\Gamma \vdash s : S \qquad \Gamma \vdash t : T}{\Gamma \vdash \langle s, t \rangle : S \times T} \qquad \frac{\Gamma \vdash r : S \times T}{\Gamma \vdash \mathsf{fst}\, r : S} \qquad \frac{\Gamma \vdash r : S \times T}{\Gamma \vdash \mathsf{snd}\, r : T}$$

$$\frac{\Gamma \vdash s : S}{\Gamma \vdash \mathsf{inl}\, s : S + T} \qquad \frac{\Gamma \vdash t : T}{\Gamma \vdash \mathsf{inr}\, t : S + T}$$

$$\frac{\Gamma \vdash r : S + T \qquad \Gamma, x{:}S \vdash s : U \qquad \Gamma, y{:}T \vdash t : U}{\Gamma \vdash \mathsf{case}\, r\, \mathsf{of}\, \mathsf{inl}\, x \Rightarrow s \mid \mathsf{inr}\, y \Rightarrow t : U}$$

$$\frac{}{\Gamma \vdash \langle\rangle : 1} \qquad \frac{\Gamma \vdash r : 0}{\Gamma \vdash \mathsf{abort}\, r : U}$$

The typing rules also ensures *well-scoping*: If $\Gamma \vdash t : T$, then $\mathsf{FV}(t) \subseteq \mathsf{dom}(\Gamma)$, meaning that all free variables of term t are bound to some type in the context Γ.

Exercise 13 (Lambda-terms)**.** Construct terms of the following types:

1. $T \to T$
2. $S \to ((S \to T) \to T)$
3. $(R \to (S \to T)) \to (S \to (R \to T))$

4. $((R \times S) \to T) \to (R \to (S \to T))$
5. $(R \to (S \to T)) \to ((R \times S) \to T)$
6. $(R + S) \to (((R \to T) \times (S \to U)) \to (T + U))$
7. $(T + 0) \to T$

Exercise 14 (Well-scoping)**.** Prove "if $\Gamma \vdash t : T$, then $\mathsf{FV}(t) \subseteq \mathsf{dom}(\Gamma)$" by induction on the typing derivation.

Exercise 15 (Inversion)**.** Prove the following inversion theorem for application:

> If $\Gamma \vdash r\, s : T$ then there exists some type S such that $\Gamma \vdash r : S \to T$ and $\Gamma \vdash s : S$.

Formulate and prove similar theorems for all the other term constructors.

Exercise 16 (Ill-typed terms)**.** Prove that the following typings are impossible for any Γ and T:

1. $\Gamma \vdash \lambda x.(x\, x) : T$
2. $\Gamma \vdash \mathsf{fst}\, (\lambda x.x) : T$
3. $\Gamma \vdash (\mathsf{inl}\, t)s : T$

Hint: Exercise 15 might prove useful here!

A variable $x{:}S$ is a placeholder for an arbitrary term s of type S. This is substantiated in the *substitution lemma*.

Lemma 1 (Substitution)**.** If $\Gamma, x{:}S \vdash t : T$ and $\Gamma \vdash s : S$ then $\Gamma \vdash t[s/x] : T$.

Exercise 17 (Proof of the Substitution Lemma)**.** Prove the Substitution Lemma (1) by induction on the typing derivation of t. You might have to generalize the statement to get the proof through for the case that t is a binder.

3.2 Computation

Terms constitute programs whose value is computed by iterated application of reduction $\boxed{t \longrightarrow t'}$. Here in t is called *redex* (from *red*ucible *ex*pression)

and t' reduct. Reduction is given by the following axioms, aka (redex) *contraction* rules:

$$\begin{array}{lcl}
(\lambda x.t)s & \longrightarrow & t[s/x] \\
\\
\mathsf{fst}\,\langle s,\, t\rangle & \longrightarrow & s \\
\mathsf{snd}\,\langle s,\, t\rangle & \longrightarrow & t \\
\\
\mathsf{case}\,(\mathsf{inl}\,r)\;\mathsf{of}\;\mathsf{inl}\,x \Rightarrow s \mid \mathsf{inr}\,y \Rightarrow t & \longrightarrow & s[r/x] \\
\mathsf{case}\,(\mathsf{inr}\,r)\;\mathsf{of}\;\mathsf{inl}\,x \Rightarrow s \mid \mathsf{inr}\,y \Rightarrow t & \longrightarrow & t[r/y]
\end{array}$$

The first rule computes the application of a function value $\lambda x.t$ to an argument s. The second and third project components from a pair. The fourth and fifth reduce a case distinction for a known scrutinee.

The reduction rules can be applied to any matching subterm. If no further reduction on a term is possible, it is in *normal form*.

Example 2 (Computation)**.**

$$\begin{array}{rl}
 & (\lambda p.\,\mathsf{fst}\,p)\,(\mathsf{case}\,\mathsf{inl}\,\langle\rangle\;\mathsf{of}\;\mathsf{inl}\,x \Rightarrow \langle x,\, x\rangle \mid \mathsf{inr}\,y \Rightarrow y) \\
\longrightarrow & (\lambda p.\,\mathsf{fst}\,p)\,(\langle x,\, x\rangle[\langle\rangle/x]) \\
= & (\lambda p.\,\mathsf{fst}\,p)\,\langle\langle\rangle,\, \langle\rangle\rangle \\
\longrightarrow & \mathsf{fst}\,\langle\langle\rangle,\, \langle\rangle\rangle \\
\longrightarrow & \langle\rangle
\end{array}$$

Exercise 18 (Compatibility rules)**.** Make "can be applied to any matching subterm" explicit by adding compatibility rules for all term constructors, like for application:

$$\frac{r \longrightarrow r'}{r\,s \longrightarrow r'\,s} \qquad \frac{s \longrightarrow s'}{r\,s \longrightarrow r\,s'}$$

Reduction is defined on untyped terms, however, it preserves typing.

Theorem 1 (Subject reduction)**.** If $\Gamma \vdash t : T$ and $t \longrightarrow t'$ then $\Gamma \vdash t' : T$.

Exercise 19. Proof the Subject Reduction Theorem. Inversion of typing (Exercise 15) could be useful.

"Good" normal forms can actually be characterized by a grammar that rules out redexes.

$$\begin{array}{llcll}
\mathsf{Nf} & \ni & v, w & ::= & u \mid \lambda x.v \mid \langle\rangle \mid \langle v,\, w\rangle \mid \mathsf{inl}\,v \mid \mathsf{inr}\,v & \text{normal form} \\
\mathsf{Ne} & \ni & u & ::= & x \mid u\,v \mid \mathsf{fst}\,u \mid \mathsf{snd}\,u \mid \mathsf{abort}\,u & \text{neutral normal form} \\
 & & & \mid & \mathsf{case}\,u\;\mathsf{of}\;\mathsf{inl}\,x \Rightarrow v \mid \mathsf{inr}\,y \Rightarrow w &
\end{array}$$

A neutral term is blocked by a variable.

Example 3 (Normal forms)**.**

- $v := \langle \mathsf{inl}\,\langle\rangle,\ \lambda f.\, f\,(\lambda x.\,(f\,x))\rangle$
- $u := \mathsf{abort}\,(\mathsf{snd}\,((\mathsf{fst}\,x)\,v))$

Exercise 20 (Pathological normal form)**.** Find a term that does not reduce but does not fit into Nf.

Lemma 2 (Soundness of normal form grammar)**.** $v \not\longrightarrow$.

Any well-typed term reduces or is a good normal form. This theorem is called *Progress* as it proves that reduction starting with a well-typed term not in Nf does not get stuck.

Lemma 3 (Progress)**.** If $\Gamma \vdash t : T$ then either $t \longrightarrow t'$ or $t \in \mathsf{Nf}$.

Exercise 21. Prove lemmata 2 and 3.

Theorem 2 (Type soundness)**.** If $\Gamma \vdash t : T$ then either t reduces infinitely or there is some $v \in \mathsf{Nf}$ such that $t \longrightarrow^* v$ and $\Gamma \vdash v : T$.

Proof. Classically, we can case on whether t reduces infinitely or not. Constructively, we emply coindution. In both cases, we rely on Subject Reduction and Progress. □

Due to the absense of recursion in simply-typed lambda calculus, we can actually prove the stronger theorem that all typed terms can be reduced to a normal form.

Theorem 3 (Weak normalization)**.** If $\Gamma \vdash t : T$ then there is some $v \in \mathsf{Nf}$ such that $t \longrightarrow^* v$.

Furthermore, no matter which reduction we choose, we will terminate in the end.

Theorem 4 (Strong normalization)**.** If $\Gamma \vdash t : T$ then any reduction sequence $t \longrightarrow t_1 \longrightarrow t_2 \longrightarrow \ldots$ starting with t is finite.

The normalization theorems are not trivial to prove, they need extra structure.

4 The Curry-Howard Isomorphism

Independently by Haskell Curry and William Alvin Howard and Nicolas de Bruijn a correspondence between logic an computation in general, and natural deduction and lambda-calculus in particular has been discovered.

- Propositional formulæ correspond to simple types.

Proposition	Type
$A \Rightarrow B$	$S \to T$
$A \wedge B$	$S \times T$
$A \vee B$	$S + T$
$\top$	1
$\bot$	0

- The inference rules of natural deduction correspond to the term constructors of lambda-calculus.

Derivation	Term
$\Rightarrow\mathsf{I}_x(\mathcal{D})$	$\lambda x.t$
$\Rightarrow\mathsf{E}(\mathcal{D}_1, \mathcal{D}_2)$	$t_1\, t_2$
$\wedge\mathsf{I}(\mathcal{D}_1, \mathcal{D}_2)$	$\langle t_1,\, t_2 \rangle$
$\wedge\mathsf{E}_1(\mathcal{D})$	$\mathsf{fst}\, t$
$\wedge\mathsf{E}_2(\mathcal{D})$	$\mathsf{snd}\, t$
$\vee\mathsf{I}_1(\mathcal{D})$	$\mathsf{inl}\, t$
$\vee\mathsf{I}_2(\mathcal{D})$	$\mathsf{inr}\, t$
$\vee\mathsf{E}_{x,y}(\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3)$	$\mathsf{case}\, t_1\, \mathsf{of}\, \mathsf{inl}\, x \Rightarrow t_2 \mid \mathsf{inr}\, y \Rightarrow t_3$
$\top\mathsf{I}$	$\langle\rangle$
$\bot\mathsf{E}(\mathcal{D})$	$\mathsf{abort}\, t$

- Proof transformations correspond to computation. In our case, the unnamed reduction $\longrightarrow$ of lambda terms is precisely the β-reduction for natural deduction derivations.

The Curry Howard Isomorphism (or the Curry Howard De Bruijn Correspondence) enables the following synergies:

1. No separate languages are needed for programming and proving.
2. Lambda-terms can be used to write proofs.
3. Types can be mixed with propositions, allowing rich function specifications.

4. Results and insights from logic can be transferred to programming language theory and vice versa.

For instance, we can use the Normalization Theorem of simply-typed lambda-calculus to prove important theorems of propositional logic:

Theorem 5 (Consistency of propositional logic)**.** There is no derivation of $\vdash \bot\ \mathit{true}$.

Proof. Suppose $\mathcal{D} :: \vdash \bot\ \mathit{true}$. By Curry-Howard, there exists a closed term $\vdash t : 0$ of the empty type. By Normalization, there exists a closed normal form $v \in \mathsf{Nf}$ of the empty type $\vdash v : 0$. By Inversion, this can only be a neutral term $v \in \mathsf{Ne}$. Every neutral term has at least one free variable. This is a contradiction to the closedness of v. □

Theorem 6 (Disjunction property)**.** If $\Gamma \vdash A \vee B\ \mathit{true}$ then $\Gamma \vdash A\ \mathit{true}$ or $\Gamma \vdash B\ \mathit{true}$.

Proof. Again, by Curry-Howard, Normalization, and Inversion. □

References

Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North Holland, Amsterdam, 1984.

Alonzo Church. A formulation of the simple theory of types. *J. Symb. Logic*, 5(2):56–68, 1940. doi: 10.2307/2266170. URL http://dx.doi.org/10.2307/2266170.

Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. URL http://gdz.sub.uni-goettingen.de/. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.