An Introduction Logical Foundations of Types and Programming Language

Brigitte Pientka

School of Computer Science
McGill University
Montreal, Canada

Contents

Chapter 1

Introduction

> Powerful insights arise from linking two fields of study previously thought separate. Examples include Descartes's coordinates, which links geometry to algebra, Planck's Quantum Theory, which links particles to waves, and Shannon's Information Theory, which links thermodynamics to communication. Such a synthesis is offered by the principle of Propositions as Types, which links logic to computation. At first sight it appears to be a simple coincidence—almost a pun—but it turns out to be remarkably robust, inspiring the design of automated proof assistants and programming languages, and continuing to influence the forefronts of computing. P. Wadler [Wadler(2015)]

Logic provides computer science with both a unifying foundational framework and a tool for modelling. In fact, logic has been called "the calculus of computer science", playing a crucial role in diverse areas such as artificial intelligence, computational complexity, distributed computing, database systems, hardware design, programming languages, and software engineering

These notes are designed to provide to give a thorough introduction to modern constructive logic, its numerous applications in computer science, and its mathematical properties. In particular, we provide an introduction to its proof-theoretic foundations and roots. Following Gentzen's approach we define the meaning of propositions by introduction rules, which assert a given proposition and explain how to conclude a given proposition, and elimination rules, which justify how we can use a given proposition and what consequences we can derive from it. In proof-theory, we are interested in studying the structure of proofs which are constructed according to axioms and inference rules of the logical system. This is in contrast to model theory, which is semantic in nature.

From a programming languages point of view, understanding the proof-theoretic foundations is particularly fascinating because of the intimate deep connection between

propositions and proofs and types and programs which is often referred to as the Curry-Howard isomorphism and establishes that proofs are isomorphic to programs. This correspondence has wide-ranging consequences in programming languages: it provides insights into compiler and program transformations; it forms the basis of modern type theory and directly is exploited in modern proof assistants such as Coq or Agda or Beluga where propositions are types and proofs correspond to well-typed programs; meta-theoretic proof techniques which have been developed for studying proof systems are often used to establish properties and provide new insights about programs and programming languages (for example, type preservation or normalization).

These lecture notes provide an introduction to Gentzen's natural deduction system and its correspondance to the lambda-calculus. We will also study meta-theoretic properties of both the natural deduction system and the well-typed lambda-calculus and highlight the symmetry behind introduction and elimination rules in logic and programming languages. Starting from intuitionistic propositional logic, we extend these ideas to first-order logic and discuss how to add induction over a given domain, if time permits. This gives rise to a simple dependently typed language (i.e. indexed types) over a given domain. Finally, we will study consistency of our logic. There are two dual approaches: the first, pursued by Gentzen, concentrates on studying the structure of proofs; we establish consistency of the natural deduction system by translating it to a sequent calculus using cut-rule; subsequently we prove that the cut-rule is admissible. As a consequence, every natural deduction proof also has a cut-free proof (i.e. normal proof). While sequent calculi are interesting in their own right, we will take a different approach of proving consistency: we are studying the structure of programs and we prove that every program normalizes, i.e. evaluation of that program will terminate and yield a normal form (which you can think of a value). This will introduce a powerful and important proof technique, proofs using logical relations following Tait.

Last, we will see how to extend these ideas to sub-strutural systems such as linear logic, where we use assumptions exactly once. This leads to the question of how can different logics (and from our perspective different programming languages) interoperate? – This will give a gentle introduction to the fundamenteal ideas of adjoint logic that allows us to accommodate a range of logics – from modal logic S4, to lax logic, affine logic, linear logic, and intuitionistic (unrestricted) logic.

Many of the early chapters of these notes are inspired by lectures and lecture notes by Frank Pfenning.

Chapter 2

Natural Deduction

> "Ich wollte nun zunächst einmal einen Formalismus aufstellen, der dem wirklichen Schließen möglichst nahe kommt. So ergab sich ein "Kalkül des natürliche Schließens".
>
> *Untersuchungen über das logische Schließen* [Gentzen(1935)]

In this chapter, we explore the fundamental principles of defining logics by revisiting Gentzen's system NJ [Gentzen(1935)], the calculus of natural deduction. The calculus was designed and developed by Gentzen to capture mathematical reasoning practice; his calculus stands in contrast to the common systems of logic at that time proposed by Frege, Russel, and Hilbert all of which have few reasoning reasoning principles, namely modus ponens, and several axioms. In Gentzen's system on the other hand we do not in general start from axioms to derive eventually our proposition; instead, we reason from assumptions. The meaning of each logical connective is given by rules which *introduce* it into the discourse together with rules which *eliminate* it, i.e. rules which tell us how to use the information described by a logical connective in the discourse. To put it differently, the meaning of a proposition is its use. An important aspect of Gentzen's system is that the meaning (i.e. the introduction and elimination rules) is defined without reference to any other connective. This allows for modular definition and extension of the logic, but more importantly this modularity extends to meta-theoretic study of natural deduction and greatly simplifies and systematically structures proofs about the logical system. We will exploit this modularity of logics often throughout this course as we consider many fragments and extension.

Gentzen's work was a milestone in the development of logic and it has had wide ranging influence today. In particular, it has influenced how we define programming languages and type systems based on the observation that proofs in natural deduction are isomorphic to terms in the λ-calculus. The relationship between proofs and programs was first observed by Curry for Hilbert's system of logic; Howard subsequently

observed that proofs in natural deduction directly correspond to functional programs. This relationship between proofs and programs is often referred as the Curry-Howard isomorphism. In this course we will explore the intimate connection between propositions and proofs on the one hand and types and programs on the other.

2.1 Propositions

There are two important ingredients in defining a logic: what are the valid propositions and what is their meaning. To define valid propositions, the simplest most familiar way is to define their grammar using Backus-Naur form (BNF). To begin with we define our propositions consisting of true ($\top$), conjunction ($\wedge$), implication ($\supset$), and disjunction ($\vee$).

$$\text{Propositions} \quad A, B, C \quad ::= \quad \top \mid A \wedge B \mid A \supset B \mid A \vee B$$

We will use A, B, C to range over propositions. The grammar only defines when propositions are well-formed. It essentially gives an inductive definition of well-formed propositions:

- $\top$ is well-formed proposition
- If A and B are well-formed proposition, then $A \wedge B$ and $A \supset B$, and $A \vee B$ are a well-formed proposition.
- Nothing else is a well-formed proposition.

2.2 Judgements and Meaning

To define the meaning of a proposition we will introduce the idea of a judgement, or more precisely, an *analytic judgment*. So, what is an an analytic judgment? – According to Per Martin Loef, "judgments are those that become evident merely by conceptual analysis" [Martin-L"of(1994)]. An example Per Martin Loef gives is the following:

The judgment "The temperature is $+25^o$C" is not an analytic judgment by itself. However, taken together with the actual thermometer showing 25^oC it makes the judgment evident. It is analytical in the sense that we can check it, namely by looking at the thermometer. What does this have to to with logic? – In logic, we may have different judgments about or involving propositions, the objects that we are analyzing. First, we revisit our definition of well-formed propositions using judgments.

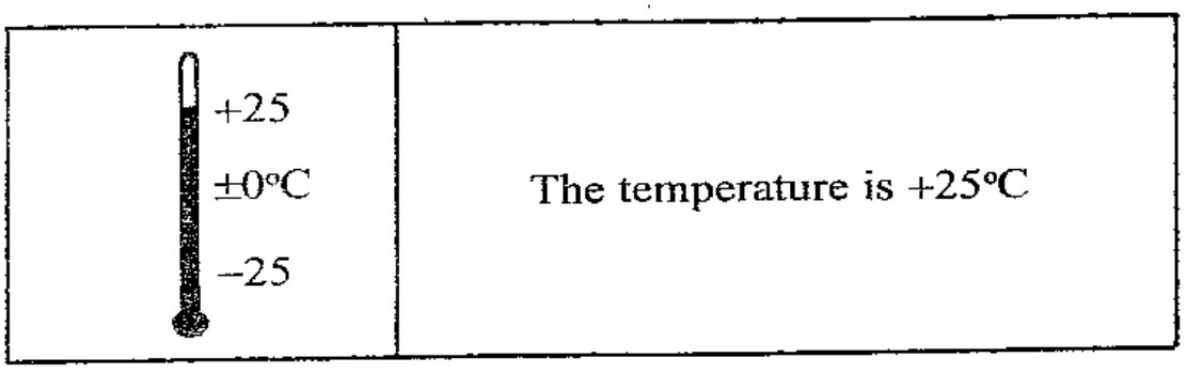

Figure 2.1: Example from [Martin-L"of(1994)]

The (analytic) judgment $\boxed{A \ \mathsf{wf}}$ defines when a proposition is well-formed. In particular, we can check and decide when this is the case. We will define the judgment $A \ \mathsf{wf}$ using inference rules. The general form of an inference rule is

$$\frac{J_1 \quad \ldots \quad J_n}{J} \text{ name}$$

where $J_1, \ldots, J_n$ are called the *premises* and J is called the *conclusion*. We can read the inference rule as follows: Given the premises $J_1, \ldots, J_n$, we can conclude J. An inference rule with no premises is called an *axiom*.

Nowe we are ready to define the well-formedness of proposition as follows.

$\boxed{A \ \mathsf{wf}}$ Proposition A is well-formed

$$\frac{}{\top \ \mathsf{wf}} \qquad \frac{A \ \mathsf{wf} \quad B \ \mathsf{wf}}{(A \ \mathbf{op} \ B) \ \mathsf{wf}} \ \{\wedge, \ \vee, \ \supset\} \in \mathsf{op}$$

Our judgmental definition of when a proposition is well-formed provides a more general approach to defining well-formed (and in general meaningful) objects than the BNF grammar approach. Especially as we move to richer languages simply giving the grammar of objects in a language may not be enough to capture what objects we want to consider well-formed.

There are many other judgements one might think of defining. In particular, we will be concerned with defining when a proposition is true using the judgment $\boxed{A \ \mathsf{true}}$ below. But there are other judgments that we could define. We list a few below:

- A false (to define when a proposition A is false)

- A possible (to define when a proposition is possible typical in modal logics)
- A true at time$t$ (to define when a proposition A is true at time $t$false)

2.2.1 The meaning of true

We are here concerned with defining the meaning of a proposition A by defining when it is true using the judgment $\boxed{A \text{ true}}$. We consider each connective individually.

Conjunction We define the meaning of $A \wedge B$ true using introduction (i.e. how to introduce the connective) and elimination rules (i.e. how to use the information contained in the connective).

$$\frac{A \text{ true} \quad B \text{ true}}{A \wedge B \text{ true}} \wedge I$$

$$\frac{A \wedge B \text{ true}}{A \text{ true}} \wedge E_l \qquad \frac{A \wedge B \text{ true}}{B \text{ true}} \wedge E_r$$

The name $\wedge I$ stands for "conjunction introduction". Given A true and B true, we can conclude that $A \wedge B$ true. The connective $\wedge$ internalizes the "and" as a proposition. The rule $\wedge I$ specifies the meaning of conjunction. How can we use the information contained in $A \wedge B$ true? To put it differently, what can we deduce from $A \wedge B$ true? - Clearly, for $A \wedge B$ true to hold, we must have A true and also B true. Note that we can have only one conclusion and we cannot write

$$\frac{A \wedge B \text{ true}}{A \text{ true} \qquad B \text{ true}} \; BAD\ FORMAT$$

Instead, we simply define two elimination rules: $\wedge E_l$ (getting the left part of the conjunction) and $\wedge E_r$ (getting the right part of the conjunction).

We will see later how to guarantee that these introduction and elimination rules fit together harmonically.

Truth The proposition "truth" is written as $\top$. The proposition $\top$ should always be true. As a consequence, the judgement $\top$ true holds unconditionally and has no premises. It is an axiom in our logical system.

$$\frac{}{\top \text{ true}} \top I$$

Since $\top$ holds unconditionally, there is no information to be obtained from it; hence there is no elimination rule.

A simple proof Before we go on and discuss other propositions, we consider what it means to prove a given proposition. Proving means constructing a derivation. Since these derivation take the form of a tree with axioms at the leafs, we also often call it a proof tree or derivation tree.

$$\dfrac{\dfrac{}{\top\ \mathsf{true}}\,\top I \qquad \dfrac{\dfrac{}{\top\ \mathsf{true}}\,\top I \qquad \dfrac{}{\top\ \mathsf{true}}\,\top I}{\top \wedge \top\ \mathsf{true}}\,\wedge I}{\top \wedge (\top \wedge \top)\ \mathsf{true}}\,\wedge I$$

Derivations convince us of the truth of a proposition. As we will see, we distinguish between proof and derivation following philosophical ideas by Martin Löfs. A proof, in contrast to a derivation, contains all the data necessary for computational (i.e. mechanical) verification of a proposition.

2.3 Hypothetical judgements and derivations

So far, we cannot prove interesting statements. In particular, we cannot accept as a valid derivation

$$\dfrac{\dfrac{A \wedge (B \wedge C)\ \mathsf{true}}{B \wedge C\ \mathsf{true}}\,\wedge l}{B\ \mathsf{true}}\,\wedge r$$

While the use of the rule $\wedge l$ and $\wedge r$ is correct, $A \wedge (B \wedge C)\ \mathsf{true}$ is unjustified. It is certainly not true unconditionally. However, we might want to say that we can derive $B\ \mathsf{true}$, *given the assumption* $A \wedge (B \wedge C)\ \mathsf{true}$. This leads us to the important notion of a *hypothetical derivation* and *hypothetical judgement*. In general, we may have more than one assumption, so a hypothetical derivation has the form

$$\dfrac{\begin{array}{ccc}\overline{J_1} & \ldots & \overline{J_n} \\ & \vdots & \end{array}}{J}$$

We can derive J given the assumptions $J_1, \ldots, J_n$. Note, that we make no claims as to whether we can in fact prove $J_1, \ldots, J_n$; they are unproven assumptions. However, if we do have derivations establishing that J_i is true, then we can replace the use of the assumption J_i with the corresponding derivation tree and eliminate the use of this assumption. This is called the *substitution principle* for hypothesis.

Implications Using a hypothetical judgement, we can now explain the meaning of $A \supset B$ (i.e. A implies B) which internalizes hypothetical reasoning on the level of propositions.

We introduce $A \supset B$ true, if we have established A true under the assumption B true.

$$\dfrac{\dfrac{}{A\ \mathsf{true}}\,u \\ \vdots \\ B\ \mathsf{true}}{A \supset B\ \mathsf{true}}\,\supset I^u$$

The label u indicates the assumption A true; using the label as part of the name $\supset I^u$ makes it clear that the assumption u can only be used to establish B true, but it is discharged in the conclusion $A \supset B$ true; we internalized it as part of the proposition $A \supset B$ and the assumption A true is no longer available. Hence, assumptions exist only within a certain scope.

Many mistakes in building proofs are made by violating the scope, i.e. using assumptions where they are not available. Let us illustrate using the rule $\supset I$ in a concrete example.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A\ \mathsf{true}}\,u \quad \dfrac{}{B\ \mathsf{true}}\,v}{A \wedge B\ \mathsf{true}}\,\wedge I}{B \supset (A \wedge B)\ \mathsf{true}}\,\supset I^v}{A \supset B \supset (A \wedge B)\ \mathsf{true}}\,\supset I^u$$

Note implications are right associative and we do not write parenthesis around $B \supset (A \wedge B)$. Also observe how we discharge all assumptions. It is critical that all labels denoting assumptions are distinct, even if they denote the "same" assumption. Consider for example the following proof below.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A\ \mathsf{true}}\,u \quad \dfrac{}{A\ \mathsf{true}}\,v}{A \wedge A\ \mathsf{true}}\,\wedge I}{A \supset (A \wedge A)\ \mathsf{true}}\,\supset I^v}{A \supset A \supset (A \wedge A)\ \mathsf{true}}\,\supset I^u$$

We introduce A true twice giving each assumption a distinct label. There are in fact many proofs we could have given for $A \supset A \supset (A \wedge A)$. Some variations we give below.

$$\dfrac{\dfrac{\dfrac{}{A\ \mathsf{true}}\,u \quad \dfrac{}{A\ \mathsf{true}}\,v}{A \wedge A\ \mathsf{true}}\wedge I}{A \supset (A \wedge A)\ \mathsf{true}}\supset I^{v}}{A \supset A \supset (A \wedge A)\ \mathsf{true}}\supset I^{u} \qquad \dfrac{\dfrac{\dfrac{}{A\ \mathsf{true}}\,u \quad \dfrac{}{A\ \mathsf{true}}\,u}{A \wedge A\ \mathsf{true}}\wedge I}{A \supset (A \wedge A)\ \mathsf{true}}\supset I^{v}}{A \supset A \supset (A \wedge A)\ \mathsf{true}}\supset I^{u} \qquad \dfrac{\dfrac{\dfrac{}{A\ \mathsf{true}}\,v \quad \dfrac{}{A\ \mathsf{true}}\,v}{A \wedge A\ \mathsf{true}}\wedge I}{A \supset (A \wedge A)\ \mathsf{true}}\supset I^{v}}{A \supset A \supset (A \wedge A)\ \mathsf{true}}\supset I^{u}$$

The rightmost derivation does not use the assumption u while the middle derivation does not use the assumption v. This is fine; assumptions do not have to be used and additional assumptions do not alter the truth of a given statement. Moreover, we note that both trees use an assumption more than once; this is also fine. Assumptions can be use as often as we want to. Finally, we note that the order in which assumptions are introduced does not enforce order of use, i.e. just because we introduce the assumption u before v, we are not required to first use u and then use v. The order of assumptions is irrelevant. We will make these structural properties about assumptions more precise when we study the meta-theoretic properties of our logical system.

Since we have ways to introduce an implication $A \supset B$, we also need a rule which allows us to use an implication and derive information from it. If we have a derivation for $A \supset B$ and at the same time have a proof for A, we can conclude B. This is justified by the substitution principle for hypothetical derivations.

$$\dfrac{A \supset B\ \mathsf{true} \quad A\ \mathsf{true}}{B\ \mathsf{true}}\supset E$$

A few examples using hypothetical derivations We give here a few examples. Consider first constructing a derivation for $(A \wedge B) \supset (B \wedge A)\ \mathsf{true}$. We do it here incrementally. A good strategy is to work from the conclusion towards the assumptions by applying a series of intro-rules; once we cannot apply any intro-rules any more, we try to close the gap to the assumptions by reasoning from the assumptions using elimination rules. Later, we will make this strategy more precise and show that this strategy is not only sound but also complete.

Employing this strategy, we first use $\supset I$ followed by $\wedge I$ to find the derivation for $(A \wedge B) \supset (B \wedge A)\ \mathsf{true}$.

$$\dfrac{\dfrac{\dfrac{}{A \wedge B\ \mathsf{true}}\,u}{\vdots}}{\dfrac{\dfrac{B\ \mathsf{true} \quad A\ \mathsf{true}}{B \wedge A\ \mathsf{true}}\wedge I}{(A \wedge B) \supset (B \wedge A)\ \mathsf{true}}\supset I^{u}}$$

We now try to close the gap by reasoning from the assumption $A \wedge B$ true; this can be accomplished by using the elimination rules $\wedge l$ and $\wedge r$.

$$\dfrac{\dfrac{\dfrac{\overline{A \wedge B \text{ true}}\,u}{B \text{ true}}\wedge E_r \quad \dfrac{\overline{A \wedge B \text{ true}}\,u}{A \text{ true}}\wedge E_l}{B \wedge A \text{ true}}\wedge I}{(A \wedge B) \supset (B \wedge A) \text{ true}} \supset I^u$$

Note again that we re-use the assumption u.

In the next example, we prove distributivity law allowing us to move implications over conjunctions. We again follow the strategy of applying all introduction rules first.

$$\dfrac{\dfrac{\dfrac{\overline{A \supset (B \wedge C) \text{ true}}\,u \quad \overline{A \text{ true}}\,v}{\vdots}}{B \text{ true}}\supset I^v \qquad \dfrac{\dfrac{\dfrac{\overline{A \supset (B \wedge C) \text{ true}}\,u \quad \overline{A \text{ true}}\,v}{\vdots}}{C \text{ true}}\supset I^v}{A \supset C \text{ true}}\supset I^v}{\dfrac{(A \supset B) \wedge (A \supset C) \text{ true}}{(A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C)) \text{ true}} \supset I^u}\wedge I$$

We now close the gap by using elimination rules $\supset E$ and $\wedge E_r$ ($\wedge E_l$ respectively).

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{A \supset (B \wedge C) \text{ true}}\,u \quad \overline{A \text{ true}}\,v}{B \wedge C \text{ true}}\supset E}{B \text{ true}}\wedge E_l}{A \supset B \text{ true}}\supset I^v \qquad \dfrac{\dfrac{\dfrac{\overline{A \supset (B \wedge C) \text{ true}}\,u \quad \overline{A \text{ true}}\,v}{B \wedge C \text{ true}}\supset E}{C \text{ true}}\wedge E_r}{A \supset C \text{ true}}\supset I^v}{\dfrac{(A \supset B) \wedge (A \supset C) \text{ true}}{(A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C)) \text{ true}} \supset I^u}\wedge I$$

Disjunction We now consider disjunction $A \vee B$ (read as "A or B"). This will use the concepts we have seen so far, but is slightly more challenging. The meaning of disjunction is characterized by two introduction rules.

$$\dfrac{A \text{ true}}{A \vee B \text{ true}} \vee I_l \qquad \dfrac{B \text{ true}}{A \vee B \text{ true}} \vee I_r$$

How should we define the elimination rule for $A \vee B$? - We may think to describe it as follows

$$\frac{A \vee B \ \mathsf{true}}{A \ \mathsf{true}} \ BAD\ RULE$$

This would allow us to obtain a proof for A from the information $A \vee B$ true; but if we know $A \vee B$ true, it could well be that A is false and B is true. So concluding from $A \vee B$ is unsound! In particular, we can derive the truth of any proposition A.

Thus we take a different approach. If we know $A \vee B$ true, then we consider two cases: A true and B true. If in both cases we can establish C true, then it must be the case that C is true!

$$\frac{\begin{array}{ccc} & \overline{A \ \mathsf{true}}^{\,u} & \overline{B \ \mathsf{true}}^{\,u} \\ & \vdots & \vdots \\ A \vee B \ \mathsf{true} & C \ \mathsf{true} & C \ \mathsf{true} \end{array}}{C \ \mathsf{true}} \vee E^{u,v}$$

We again use hypothetical judgement to describe the rule for disjunction. Note the scope of the assumptions. The assumption A true labelled u can only be used in the middle premise, while the assumption B true labelled v can only be used in the rightmost premise. Both premises are discharged at the disjunction elimination rule.

Let us consider an example to understand how to use the disjunction elimination rule and prove commutativity of disjunction.

$$\frac{\begin{array}{c} \overline{A \vee B \ \mathsf{true}}^{\,u} \\ \vdots \\ B \vee A \ \mathsf{true} \end{array}}{(A \vee B) \supset (B \vee A) \ \mathsf{true}} \supset I^u$$

At this point our strategy of continuing to apply introduction rules and working from the bottom-up, does not work, since we would need to commit to prove either A true or B true. Instead, we will use our assumption $A \vee B$ true and then prove $A \vee B$ true under the assumption A true and separately prove $A \vee B$ true under the assumption B true.

$$\frac{\dfrac{\overline{A \vee B \ \mathsf{true}}^{\,u} \qquad \dfrac{\dfrac{\overline{A \ \mathsf{true}}^{\,v}}{B \vee A \ \mathsf{true}} \vee I_l \qquad \dfrac{\overline{B \ \mathsf{true}}^{\,w}}{B \vee A \ \mathsf{true}} \vee I_r}{B \vee A \ \mathsf{true}} \vee E^{v,w}}{(A \vee B) \supset (B \vee A) \ \mathsf{true}} \supset I^u$$

Falsehood Last but not least, we consider the rule for falsehood (written as $\bot$). Clearly, we should never be able to prove (directly) $\bot$. Hence there is no introduction rule which introduces $\bot$. However, we might nevertheless derive $\bot$ (for example because our assumptions are contradictory or it occurs directly in our assumptions) in the process of constructing a derivation. If we have derived $\bot$, then we are able to conclude anything from it, since we have arrived at a contradiction.

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

It might not be obvious that $\bot$ is very useful. It is particularly important in allowing us to define $\neg A$ (read as "not A) as $A \supset \bot$. More on this topic later.

2.4 Local soundness and completeness

One might ask how do we know that the introduction and elimination rules we have given to define the meaning for each proposition are sensible. We have earlier alluded to the unsound proposal for the disjunction rule. Clearly, the meaning is not just defined by any pair of introduction and elimination rules, but these rules must meet certain conditions; in particular, they should not allow us to deduce new truths (soundness) and they should be strong enough to obtain all the information contained in a connective (completeness) [Belnap(1962)]. This is what sometimes is referred to as *harmony* by [Dummett(1993)]. Let us make this idea more precise:

- **Local Soundness** if we introduce a connective and then immediately eliminate it, we should be able to erase this detour and find a more direct derivation ending in the conclusion. If this property fails, the elimination rules are too strong, i.e. they allow us to derive more information than we should.
- **Local completeness:** we can eliminate a connective in such a way that it retains sufficient information to reconstitute it by an introduction rule. If this property fails, the elimination rules are too weak: they do not allow us to conclude everything we should be able to.

2.4.1 Conjunction

We revisit here the harmony of the given introduction and elimination rules for conjunction and check our intuition that they are sensible. If we consider the rule $\wedge I$ as a complete definition for $A \wedge B$ true, we should be able to recover both A true and B true.

Local soundness

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ A\ \mathsf{true}\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ B\ \mathsf{true}\end{array}}{A \wedge B\ \mathsf{true}}\wedge I}{A\ \mathsf{true}}\wedge E_l \quad \Longrightarrow \quad \begin{array}{c}\mathcal{D}_1\\ A\ \mathsf{true}\end{array}$$

and symmetrically

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ A\ \mathsf{true}\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ B\ \mathsf{true}\end{array}}{A \wedge B\ \mathsf{true}}\wedge I}{A\ \mathsf{true}}\wedge E_l \quad \Longrightarrow \quad \begin{array}{c}\mathcal{D}_2\\ B\ \mathsf{true}\end{array}$$

Clearly, it is unnecessary to first introduce a conjunction and then immediately eliminate it, since there is a more direct proof already. These detours are what makes proof search infeasible in practice in the natural deduction calculus. It also means that there are many different proofs for a give proposition many of which can collapse to the more direct proof which does not use the given detour. This process is called normalization - or trying to find a normal form of a proof.

Local completeness We need to show that $A \wedge B\ \mathsf{true}$ contains enough information to rebuild a proof for $A \wedge B\ \mathsf{true}$.

$$\begin{array}{c}\mathcal{D}\\ A \wedge B\ \mathsf{true}\end{array} \quad \Longrightarrow \quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\ \mathsf{true}\end{array}}{A\ \mathsf{true}}\wedge E_l \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\ \mathsf{true}\end{array}}{B\ \mathsf{true}}\wedge E_r}{A \wedge B\ \mathsf{true}}\wedge I$$

2.4.2 Implications

Next, we revisit the given introduction and elimination rules for implications. Again, we first verify that we can introduce $A \supset B$ followed by eliminating it without gaining additional information.

$$\dfrac{\dfrac{\dfrac{}{A\ \mathsf{true}}\,u}{\mathcal{D}}}{\dfrac{B\ \mathsf{true}}{A \supset B\ \mathsf{true}}\supset I^u \qquad \dfrac{\mathcal{E}}{A\ \mathsf{true}}}{B\ \mathsf{true}}\supset E \qquad \Longrightarrow \qquad \dfrac{\mathcal{E}}{\dfrac{}{A\ \mathsf{true}}}\\ \mathcal{D}\\ B\ \mathsf{true}$$

Given the hypothetical derivation $\mathcal{D}$ which depends on the assumption $A\ \mathsf{true}$, we can eliminate the use of this assumption by simply referring to $\mathcal{E}$ the proof for $A\ \mathsf{true}$. We sometimes write

$$\dfrac{\dfrac{\mathcal{E}}{A\ \mathsf{true}}}{\mathcal{D}}\\ B\ \mathsf{true} \qquad \text{more compactly as} \qquad \begin{array}{c}[\mathcal{E}/u]\mathcal{D}\\ B\ \mathsf{true}\end{array}$$

This emphasizes the true nature of what is happening: we are substituting for the assumption u the proof $\mathcal{E}$. Note that this local reduction may significantly increase the overall size of the derivation, since the derivation $\mathcal{E}$ is substituted for each occurrence of the assumption labeled u in $\mathcal{D}$ and may thus be replicated many times.

Local completeness We now check whether our elimination rules are strong enough to get all the information out they contain, i.e. can we reconstitute $A \supset B\ \mathsf{true}$ given a proof for it?

$$\begin{array}{c}\mathcal{D}\\ A \supset B\ \mathsf{true}\end{array} \qquad \Longrightarrow \qquad \dfrac{\dfrac{\mathcal{D}}{A \supset B\ \mathsf{true}} \qquad \dfrac{}{A\ \mathsf{true}}\,u}{B\ \mathsf{true}}\supset E}{A \supset B\ \mathsf{true}}\supset I^u$$

2.4.3 Disjunction

We can now see whether we understand the principle behind local soundness and completeness.

Local soundness We establish again that we cannot derive any unsound information from first introducing $A \vee B$ and then eliminating it. Note that the rule $\vee E^{u,v}$ ends in a generic proposition C which is independent of A and B. We note that we have a proof $\mathcal{E}$ for C which depends on the assumption A true. At the same time we have a proof $\mathcal{D}$ which establishes A true. Therefore by the substitution principle, we can replace and justify any uses of the assumption A true in $\mathcal{E}$ by the proof $\mathcal{D}$.

$$\dfrac{\dfrac{\mathcal{D}}{A}}{\dfrac{}{A \vee B}\vee I_l} \qquad \dfrac{\dfrac{}{A\ \mathsf{true}}\,u}{\mathcal{E}} \qquad \dfrac{\dfrac{}{B\ \mathsf{true}}\,u}{\mathcal{F}}$$
$$\dfrac{}{C\ \mathsf{true}}\vee E^{u,v} \qquad \Longrightarrow \qquad \dfrac{\mathcal{D}}{\dfrac{}{A\ \mathsf{true}}}\ \mathcal{E}\ C\ \mathsf{true}$$

Similarly, we can show

$$\dfrac{\dfrac{\mathcal{D}}{B}}{\dfrac{}{A \vee B}\vee I_r} \qquad \dfrac{\dfrac{}{A\ \mathsf{true}}\,u}{\mathcal{E}} \qquad \dfrac{\dfrac{}{B\ \mathsf{true}}\,u}{\mathcal{F}}$$
$$\dfrac{}{C\ \mathsf{true}}\vee E^{u,v} \qquad \Longrightarrow \qquad \dfrac{\mathcal{D}}{\dfrac{}{B\ \mathsf{true}}}\ \mathcal{F}\ C\ \mathsf{true}$$

Local completeness

$$\dfrac{\mathcal{D}}{A \vee B\ \mathsf{true}} \qquad \Longrightarrow \qquad \dfrac{\mathcal{D}}{A \vee B\ \mathsf{true}} \qquad \dfrac{\dfrac{}{A\ \mathsf{true}}\,u}{A \vee B\ \mathsf{true}}\vee I_l \qquad \dfrac{\dfrac{}{B\ \mathsf{true}}\,v}{A \vee B\ \mathsf{true}}\vee I_r$$
$$A \vee B\ \mathsf{true}\ \vee E^{u,v}$$

2.4.4 Negation

So far we have simply used $\neg A$ as an abbreviation for $A \supset \bot$ and at any point we can expanded $\neg A$ exposing its definition. How could we define the meaning of $\neg A$ direction? - In order to derive $\neg A$, we assume A and try to derive a contradiction. We want to define $\neg A$ without reference to $\bot$; to accomplish this we use a *parametric propositional parameter* p which stands for *any proposition*. We can therefore establish $\neg A$, if we are able to derive any proposition p from the assumption A true. Note that p is fixed but arbitrary once we pick it.

$$\dfrac{\dfrac{\dfrac{}{A\ \mathsf{true}}\,u}{\vdots}}{\dfrac{p\ \mathsf{true}}{\neg A\ \mathsf{true}}\,\neg I^{p,u}} \qquad \dfrac{\neg A\ \mathsf{true} \qquad A\ \mathsf{true}}{C\ \mathsf{true}}\,\neg E$$

We can check again local soundness: if we introduce $\neg A$ and then eliminate it, we have not gained any information.

$$\dfrac{\dfrac{\dfrac{}{A\ \mathsf{true}}\,u}{\mathcal{D}}}{\dfrac{\dfrac{p\ \mathsf{true}}{\neg A\ \mathsf{true}}\,\neg I^{pu} \qquad \mathcal{E}}{C\ \mathsf{true}}\,\neg E} \quad\Longrightarrow\quad \dfrac{\dfrac{\mathcal{E}}{A\ \mathsf{true}}}{[C/p]\mathcal{D}}$$

Since p denotes any proposition and $\mathcal{D}$ is parametric in p, we can replace p with C; moreover, since we have a proof $\mathcal{E}$ for A, we can also eliminate the assumption u by replacing any reference to u with the actual proof $\mathcal{E}$.

The local expansion is similar to the case for implications.

$$\dfrac{\mathcal{D}}{\neg A\ \mathsf{true}} \quad\Longrightarrow\quad \dfrac{\dfrac{\mathcal{D}}{\neg A\ \mathsf{true}} \qquad \dfrac{}{A\ \mathsf{true}}\,u}{\dfrac{p\ \mathsf{true}}{\neg A\ \mathsf{true}}\,\supset I^{pu}}\,\supset E$$

It is important to understand the use of parameters here. Parameters allow us to prove a given judgment generically without committing to a particular proposition. As a rule of thumb, if one rule introduces a parameter and describes a derivation which holds generically, the other must is a derivation for a concrete instance.

2.5 Localizing Hypothesis

So far, we have considered Gentzen's style natural deduction proofs where assumptions in the proof are implicit. Reasoning directly from assumptions results in compact and elegant (on-paper) proofs. Yet this is inconvenient for several reasons: it is hard to keep track what assumptions are available; it is more difficult to reason about such proofs via structural induction over the introduction and elimination rules since we do

$$\frac{\Gamma \vdash A\ \mathsf{true} \quad \Gamma \vdash B\ \mathsf{true}}{\Gamma \vdash A \wedge B\ \mathsf{true}}\ \wedge I \qquad \frac{\Gamma \vdash A \wedge B\ \mathsf{true}}{\Gamma \vdash A\ \mathsf{true}}\ \wedge E_l \qquad \frac{\Gamma \vdash A \wedge B\ \mathsf{true}}{\Gamma \vdash B\ \mathsf{true}}\ \wedge E_r$$

$$\frac{\Gamma, u{:}A\ \mathsf{true} \vdash B\ \mathsf{true}}{\Gamma \vdash A \supset B\ \mathsf{true}}\ \supset I^u \qquad \frac{\Gamma \vdash A \supset B\ \mathsf{true} \quad \Gamma \vdash A\ \mathsf{true}}{\Gamma \vdash B\ \mathsf{true}}\ \supset E$$

$$\frac{\Gamma \vdash A\ \mathsf{true}}{\Gamma \vdash A \vee B\ \mathsf{true}}\ \vee I_l \qquad \frac{\Gamma \vdash B\ \mathsf{true}}{\Gamma \vdash A \vee B\ \mathsf{true}}\ \vee I_r$$

$$\frac{\Gamma \vdash A \vee B\ \mathsf{true} \quad \Gamma, u : A\ \mathsf{true} \vdash C\ \mathsf{true} \quad \Gamma, v : B\ \mathsf{true} \vdash C\ \mathsf{true}}{\Gamma \vdash C\ \mathsf{true}}\ \vee E^{u,v}$$

$$\frac{}{\Gamma \vdash \top\ \mathsf{true}}\ \top I \qquad \frac{\Gamma \vdash \bot\ \mathsf{true}}{\Gamma \vdash C\ \mathsf{true}}\ \bot E \qquad \boxed{\frac{u : A\ \mathsf{true} \in \Gamma}{\Gamma \vdash A\ \mathsf{true}}\ u}$$

Figure 2.2: Natural Deduction with Explicit Context for Assumptions

not have an explicit base case for assumptions; it is more difficult to state and prove properties about assumptions such as weakening or substitution properties.

For these reasons, we will introduce an explicit context formulation of the natural deduction rules we have seen so far making explicit some of the ambiguity of the two-dimensional notation. We therefore introduce an *explicit* context for bookkeeping, since when establishing properties about a given language, it allows us to consider the variable case(s) separately and to state clearly when considering closed objects, i.e., an object in the empty context. More importantly, while structural properties of contexts are implicitly present in the above presentation of inference rules (where assumptions are managed informally), the explicit context presentation makes them more apparent and highlights their use in reasoning about contexts.

Typically, a context of assumptions is characterized as a sequence of formulas listing its elements. More formally we define contexts as follows.

$$\text{Context } \Gamma \quad ::= \quad \cdot \mid \Gamma, u{:}A\ \mathsf{true}$$

We hence generalize our judgment $A\ \mathsf{true}$ to $\Gamma \vdash A\ \mathsf{true}$ which can be read as "given the assumptions in Γ we can prove A. This makes our assumptions explicit. We interpret all our inference rules within the context Γ (see Fig. 2.2).

We can now state more succinctly structural properties about our logic

1. *Weakening* Extra assumptions don't matter.

2. *Exchange* The order of *hypothetical assumptions* does not matter.

3. *Contraction* An assumption can be used as often as we like.

as actual theorems which can be proven by structural induction.

Theorem 2.5.1.

1. *Weakening. If* $\Gamma, \Gamma' \vdash A$ true *then* $\Gamma, u : B$ true, $\Gamma' \vdash A$ true.

2. *Exchange If* $\Gamma, x : B_1$ true, $y : B_2$ true, $\Gamma' \vdash A$ true
   then $\Gamma, y : B_2$ true, $x : B_1$ true, $\Gamma' \vdash A$ true.

3. *Contraction If* $\Gamma, x : B$ true, $y : B$ true, $\Gamma' \vdash A$ true *then* $\Gamma, x : B$ true, $\Gamma' \vdash A$ true.

In addition to these structural properties, we can now also state succinctly the substitution property.

Theorem 2.5.2 (Substitution)**.**
If $\Gamma, x : A$ true, $\Gamma' \vdash B$ true *and* $\Gamma \vdash A$ true *then* $\Gamma, \Gamma' \vdash B$ true.

2.6 Proofs by structural induction

We will here review how to prove properties about a given formal system; this is in contrast to reasoning within a given formal system. It is also referred to as "meta-reasoning".

One of the most common meta-reasoning techniques is "proof by structural induction on a given proof tree or derivation". One can always reduce this structural induction argument to a mathematical induction purely based on the height of the proof tree. We illustrate this proof technique by proving the substitution property.

Theorem 2.6.1. *If* $\Gamma, u : A$ true, $\Gamma' \vdash C$ true *and* $\Gamma \vdash A$ true *then* $\Gamma, \Gamma' \vdash C$ true.

Proof. By structural induction on the derivation $\Gamma, u : A$ true, $\Gamma' \vdash C$ true. We consider here a few cases, although for it to be a complete proof we must consider all rules.

There are three base cases to consider; to be thorough we write them out.

Case $\mathcal{D} = \dfrac{}{\Gamma, u : A \text{ true}, \Gamma' \vdash \top \text{ true}}\, \top I.$
$\Gamma, \Gamma' \vdash \top$ true by $\top I$

Case $\mathcal{D} = \dfrac{}{\Gamma, u : A \text{ true}, \Gamma' \vdash A \text{ true}}\, u.$

$\Gamma \vdash A$ true — by assumption
$\Gamma, \Gamma' \vdash A$ true — by weakening

Case $\mathcal{D} = \dfrac{v : C \text{ true} \in (\Gamma, \Gamma')}{\Gamma, u : A \text{ true}, \Gamma' \vdash C \text{ true}}\, v$

$\Gamma, \Gamma' \vdash C$ true — by rule v

We now consider some of the step cases. The induction hypothesis allos us to assume the substitution property holds for smaller derivations.

Case $\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{F} & \mathcal{E} \\ \Gamma, u : A \text{ true}, \Gamma' \vdash C \text{ true} & \Gamma, u : A \text{ true}, \Gamma' \vdash B \text{ true}\end{array}}{\Gamma, u : A \text{ true}, \Gamma' \vdash C \wedge B \text{ true}}\, \wedge I$

$\Gamma \vdash A$ true — by assumption
$\Gamma, \Gamma' \vdash C$ true — by i.h. using $\mathcal{F}$ and assumption
$\Gamma, \Gamma' \vdash B$ true — by i.h. using $\mathcal{E}$ and assumption
$\Gamma, \Gamma' \vdash C \wedge B$ true — by rule $\wedge I$

The other cases for $\wedge E_l$, $\wedge E_r$, $\supset E$, $\vee I_l$, $\vee I_r$, or $\bot E$ follow a similar schema. A bit more interesting are those cases where we introduce new assumptions and the context of assumptions grows.

Case $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{E} \\ \Gamma, u : A \text{ true}, \Gamma', v{:}B \text{ true} \vdash C \text{ true}\end{array}}{\Gamma, u : A \text{ true}, \Gamma' \vdash B \supset C \text{ true}}\, \supset I^v$

$\Gamma \vdash A$ true — by assumption
$\Gamma, \Gamma', v : B \text{ true} \vdash C$ true — by i.h. using $\mathcal{E}$
$\Gamma, \Gamma' \vdash B \supset C$ true — by rule $\supset I^v$.

Note that the appeal to the induction hypothesis is valid, because the height of the derivation $\mathcal{E}$ is smaller than the height of the derivation $\mathcal{D}$. Our justification is independent of the fact that the context in fact grew. □

2.7 Exercises

Exercise 2.7.1. Give proofs in natural deduction for the problems listed below. Do these proofs on paper.

- **Recall that in the problem formulations below conjunction ($\wedge$) and disjunction ($\vee$) bind tighter than implications ($\supset$) and implications are right-associative.**
- **Beluga template available upon request; if you are keen to explore Beluga, please contact brigitte.pientka@mcgill.ca.**

B1 Basic $(A \wedge (B \wedge C)) \supset A \wedge B$

B2 Basic $(A \supset B) \supset ((B \supset C) \supset A \supset C)$

B3 Basic $((A \vee B) \supset C) \supset (A \supset C) \wedge (B \supset C)$

B4 Basic $((A \supset C) \wedge (B \supset C)) \supset (A \vee B) \supset C$

B5 Basic $(A \supset B) \wedge (A \vee B) \supset (B \vee C)$

B6 Basic $A \vee (B \wedge C) \supset (A \vee B) \wedge (A \vee C)$

B7 Basic $(A \vee B) \wedge (A \vee C) \supset A \vee (B \wedge C)$

Exercise 2.7.2. Assume someone defines conjunction with the following two rules:

$$\frac{\begin{array}{cc}\overline{A}\,u & \overline{B}\,v \\ & \vdots \\ A \wedge B & C\end{array}}{C}\wedge E^{u,v} \qquad \frac{A \qquad B}{A \wedge B}\wedge I$$

Are these rules sound and complete? – Show local soundness and completeness.

Exercise 2.7.3. Give a direct definition of "A iff B", which means "A implies B and B implies A".

1. Give introduction and elimination rules for iff without recourse to any other logical connectives.

2. Display the local reductions that show the local soundness of the elimination rules.

3. Display the local expansion that show the local completeness of the elimination rules.

Exercise 2.7.4. $A \overline{\wedge} B$ is usually defined as $\neg(A \wedge B)$. In this problem we explore the definition of nand using introduction and elimination rules.

1. Give introduction and elimination rules for nand without recourse to any other logical connectives.

2. Display the local reductions that show the local soundness of the elimination rules.

3. Display the local expansion that show the local completeness of the elimination rules.

Exercise 2.7.5. Extend the proof of the substitution lemma for the elimination rules for conjunction ($\wedge E_i$) and disjunction ($\vee E$).

Chapter 3

Proof Terms

> "For my money, Gentzen's natural deduction and Church's lambda calculus are on a par with Einstein's relativity and Dirac's quantum physics for elegance and insight. And the maths are a lot simpler. "
>
> *Proofs as Programs: 19th Century Logic and 21 Century Computing*, P. Wadler [Wadler(2000)]

In this chapter, we describe the relationship between propositions and proofs on the one hand and types and programs on the other. On the propositional fragment of logic this is referred to as the Curry-Howard isomorphism. Martin Löf developed this intimate relationship of propositions and types further leading to what we call type theory. More precisely, we will establish the relationship between natural deduction proofs and programs written in Church's lambda-calculus.

Curry first observed a perhaps surprising but possibly merely coincidental similarity between some axioms of intuitionistic and combinatory logic in the 1930s. He subsequently noted Hilbert-style deduction systems, coincides combinators in the lambda-calculus. Howard noticed in 1969 the correspondance between Gentzen's natural deduction calculus and the simply-typed lambda-calculus. Independently, Reynolds extended the simply-typed lambda-calculus to the polymorphic lambda-calculus which corresponds to to second-order logic where we quantify over propositions. Since then the isomorphism has been applied in a wider variety of cases, and it took on both theoretical and philosophical significance.

Its theoretical significance lies in the fact that it can be used to prove strong normalization of natural deduction, i.e., every derivation in the natural deduction can be translated to a normal derivation. In fact following Tait a now standard way to prove this is by defining reducibility candidates which describe sets of terms that are reducible at a given type (proposition).

Its philosophical significance arises out of the claim that the lambda term assigned to a derivation can be taken to be the "computational content" of the derivation. In fact, the reduction rules for lambda-terms (i.e. how we reduce and compute with lambda-terms) can be derived from local soundness and local completeness properties in the natural deduction calculus. Building a language based on logical foundations also provides insights into the meaning of computational properties and constructs. Rather than adding a feature to a language in an ad-hoc manner, we can ask what is the computational interpretation of this proposition and relate this interpretation to computational practice.

3.1 Propositions as Types

In order to highlight the relationship between proofs and programs, we introduce a new judgement $M : A$ which reads as "M is a proof term for proposition A". Our intention is to capture the structure of the proof using M. As we will see there are also other interpretations of this judgement:

$M : A$	M is a proof term for proposition A M is a program of type A

These dual interpretations are at the heart of the Curry-Howard isomorphism. We can think of M as the term that represents the proof of A true or we think of A as the type of the program M.

Our intention is that

$$M : A \quad \text{iff} \quad A\ \mathsf{true}$$

However, we want in fact more than merely that if a given proposition A is provable there exists a program M that has type A. We want that the derivation for $M : A$ has the *identical structure* as the derivation for A true. We will revisit our natural deduction rules and annotate them with proof terms. The isomorphism between $M : A$ and A true will then become obvious. We will achieve this by annotating our previous introduction and elimination rules in the natural deduction calculus with proof terms.

Conjunction Constructively, we can think of $A \wedge B$ true as a pair of proofs: the proof M for A true and the proof N for B true.

$$\frac{M : A \qquad N : B}{\langle M,\ N\rangle : A \wedge B}\wedge I$$

The elimination rules correspond to the projections from a pair to its first and second element.

$$\frac{M : A \wedge B}{\mathsf{fst}\ M : A}\wedge E_l \qquad \frac{M : A \wedge B}{\mathsf{snd}\ M : B}\wedge E_r$$

In other words, conjunction $A \wedge B$ corresponds to the cross product type $A \times B$. We can also annotate the local soundness rule:

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ M : A & N : B\end{array}}{\langle M,\ N\rangle : A \wedge B}\wedge I}{\mathsf{fst}\ \langle M,\ N\rangle : A}\wedge E_l \qquad \Longrightarrow \qquad \begin{array}{c}\mathcal{D}_1 \\ M : A\end{array}$$

and dually

$$\dfrac{\dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ M : A & N : B\end{array}}{\langle M,\ N\rangle : A \wedge B}\wedge I}{\mathsf{snd}\ \langle M,\ N\rangle : A}\wedge E_l \qquad \Longrightarrow \qquad \begin{array}{c}\mathcal{D}_2 \\ N : B\end{array}$$

The local soundness proofs for $\wedge$ give rise to two reduction rule:

$$\begin{array}{lcl}\mathsf{fst}\ \langle M,\ N\rangle & \Longrightarrow & M \\ \mathsf{snd}\ \langle M,\ N\rangle & \Longrightarrow & N\end{array}$$

We can interpret

$$M \Longrightarrow M' \quad M \text{ reduces to } M'$$

A computation then proceeds by a sequence of reduction steps:

$$M \Longrightarrow M_1 \Longrightarrow \ldots \Longrightarrow M_n$$

We reduce M until we (hopefully) reach a normal form which is the result of the computation which cannot be reduced any further. This normal form corresponds to a normal proof in the natural deduction calculus that does not have any detours. The annotated local soundness proof can be interpreted as:

If $M : A$ and $M \Longrightarrow M'$ then $M' : A$

We can read it as follows: If M has type A, and M reduces to M', then M' has also type A, i.e. reduction preserves types. This statement is often referred to as subject reduction or type preservation in programming languages. Wright and Felleisen [**?**] were the first to advocate using this idea to prove type soundness for programming languages. It is proven by case analysis (and induction) on $M \Longrightarrow M'$. Our local soundness proof for $\wedge$ describes the case for the two reduction rules: $\mathsf{fst}\ \langle M,\ N\rangle \quad \Longrightarrow \quad M$ and $\mathsf{snd}\ \langle M,\ N\rangle \quad \Longrightarrow \quad N$. We will more elaborate on reductions and their theoretical properties later.

Truth Constructively, $\top$ corresponds to the unit element $()$.

$$\frac{}{() : \top}\ \top I$$

$\top$ in logic corresponds to the unit type often written as unit or $\mathbf{1}$. There is no elimination rule for $\top$ and hence there is no reduction. This makes sense, since $()$ is already a value it cannot step.

Implication Constructively, we can think of a proof for $A \supset B$ as a function which given a proof for A, knows how to construct and return a proof for B. This function accepts as input a proof of type A and we returns a proof of type B". We characterize such anonymous functions using λ-abstraction.

$$\frac{\begin{array}{c}\overline{x{:}A}\ u \\ \vdots \\ M : B\end{array}}{\lambda x{:}A.M : A \supset B}\ \supset I^{x,u}$$

The variable x in $\lambda x{:}A.M$ denotes a proof term. However, we also have the typing assumption $x{:}A$ (read as "the variable x has type A"). It corresponds to our labelling hypothesis $A\ \mathsf{true}$ in the natural deduction system.

Consider the trivial proof for $(A \wedge A) \supset A\ \mathsf{true}$.

$$\frac{\dfrac{\dfrac{}{x : A}\ u}{\mathsf{fst}\ x : A}\ \wedge E_l}{\lambda x{:}(A \wedge A).\mathsf{fst}\ x : (A \wedge A) \supset A}\ \supset^{x,u}$$

Note that we use u to justify that x has type A. Since we ensure that the variable that is being introduced is unique in the $\supset$-rule, we often simply identify the assumption

$x : A$ (which reads as "variable x has type A) with the name x in practice, although they actually refer to two different entities.

Note that a different proof where we extract the right A from $A \wedge A$, can results in a different proof term.

$$\dfrac{\dfrac{\dfrac{}{x : A}\,u}{\mathsf{snd}\ x : A}\wedge E_r}{\lambda x{:}(A \wedge A).\mathsf{snd}\ x : (A \wedge A) \supset A}\supset^{x,u}$$

The probably simplest proof for $A \supset A$ can be described by the identity function $\lambda x{:}A.x$.

The elimination rule for $\supset E$ corresponds to function application. Given the proof term M for proposition (type) $A \supset B$ and a proof term N for proposition (type) A, characterize the proof term for B using the application $M\ N$.

$$\dfrac{M : A \supset B \qquad \Gamma \vdash N : A}{M\ N : B}\supset E$$

An implications $A \supset B$ can be interpreted as a function type $A \to B$.The introduction rule corresponds to the typing rule for function abstractions and the elimination rule corresponds to the typing rule for function application.

Note that we continue to recover the natural deduction rules by simply erasing the proof terms. This will continue to be the case and highlights the isomorphic structure of proof trees and typing derivations.

As a second example, let us consider the proposition $(A \wedge B) \supset (B \wedge A)$ whose proof we've seen earlier. We will write it here in sequent-style natural deduction and annotate it with proof terms.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x : A \wedge B}\,u}{\mathsf{snd}\ x : B\ \mathsf{true}}\wedge E_r \qquad \dfrac{\dfrac{}{x : A \wedge B}\,u}{\mathsf{fst}\ x : A\ \mathsf{true}}\wedge E_l}{\langle \mathsf{snd}\ x,\ \mathsf{fst}\ x\rangle : (B \wedge A)\ \mathsf{true}}\wedge I}{\lambda x : (A \wedge B).\langle \mathsf{snd}\ x,\ \mathsf{fst}\ x\rangle : (A \wedge B) \supset (B \wedge A)\ \mathsf{true}}\supset I^u$$

Let us revisit the local soundness proof for $\supset$ to highlight the interaction between function abstraction and function application.

$$\dfrac{\dfrac{\dfrac{}{x:A}\,u}{\begin{array}{c}\mathcal{D}\\ M:B\end{array}}}{\dfrac{}{\lambda x{:}A.B : A \supset B}\supset I^{x,u}} \qquad \dfrac{\mathcal{E}}{N:A} \qquad \supset E \qquad \Longrightarrow \qquad \dfrac{\mathcal{E}}{N:A}\,u \qquad \begin{array}{c}\\ [N/x]\mathcal{D}\\ [N/x]M:B\end{array}$$

This gives rise to the reduction rule for function applications:

$$(\lambda x{:}A.M)\ N \qquad \Longrightarrow \qquad [N/x]M$$

The annotated soundness proof above corresponds to the case in proving that the reduction rule preserves types. It also highlights the distinction between x which describes a term of type A and u which describes the assumption that x has type A. In the proof, we appeal in fact to two substitution lemmas:

1. Substitution lemma on terms: Replace any occurrence of x with N
2. Substitution lemma on judgements: Replace the assumption $N : A$ with a proof $\mathcal{E}$ which establishes $N : A$.

Disjunction Constructively, a proof of $A \vee B$ says that we have either a proof of A or a proof of B. All possible proofs of $A \vee B$ can be described as a set containing proofs of A and proofs of B. We can tag the elements in this set depending on whether they prove A or B. Since A occurs in the left position of $\vee$, we tag elements denoting a proof M of A with $\mathsf{inl}^A\ M$; dually B occurs in the right position of $\vee$ and we tag elements denoting a proof N of B with $\mathsf{inr}^B\ N$. Hence, the set of proofs for $A \vee B$ contains $\mathsf{inl}^A\ M_1, \ldots, \mathsf{inl}^A\ M_n$, i.e. proofs for A, and $\mathsf{inr}^B\ N_1, \ldots, \mathsf{inr}^B\ N_k$,, i.e. proofs for B. From a type-theory point of view, disjunctions correspond to disjoint sums, often written as $A + B$. The introduction rules for disjunction correspond to the left and right injection.

$$\dfrac{M : A}{\mathsf{inl}^A\ M : A \vee B} \vee I^l \qquad \dfrac{N : B}{\mathsf{inr}^B\ N : A \vee B} \vee I^r$$

We annotate inl and inr with the proposition A and B respectively. As a consequence, every proof term correspond to a unique proposition; from a type-theoretic perspective, it means every program has a unique type.

The elimination rule for disjunctions corresponds to a case-construct which distinguishes between the left and right injection. To put it differently, know we have a proof term for $A \vee B$, we know it is either of the form $\mathsf{inl}^A\ x$ where x is a proof for A or of the form $\mathsf{inr}^B\ y$ where y is a proof for B.

$$\frac{M : A \vee B \qquad \begin{array}{c}\overline{x : A}^{\,u} \\ \vdots \\ N_l : C\end{array} \qquad \begin{array}{c}\overline{y : B}^{\,v} \\ \vdots \\ N_r : C\end{array}}{\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^A\ x \to N_l \mid \mathsf{inr}^B\ y \to N_r : C}\ \vee E^{x,u,y,v}$$

Note that the labelled hypothesis u which stands for the assumption $x : A$ is only available in the proof N_l for C. Similarly, labelled hypothesis v which stands for the assumption $y : B$ is only available in the proof N_r for C. This is also evident in the proof term $\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^A\ x \to N_l \mid \mathsf{inr}^B\ y \to N_r$. The x is only available in N_l, but cannot be used in N_r which lives within the scope of y.

As before (left to an exercise), the local soundness proof for disjunction gives rise to the following two reduction rules:

$$\begin{array}{lcl} \mathsf{case}\ (\mathsf{inl}^A\ M)\ \mathsf{of}\ \mathsf{inl}^A\ x \to N_l \mid \mathsf{inr}^B\ y \to N_r & \Longrightarrow & [M/x]N_l \\ \mathsf{case}\ (\mathsf{inr}^B\ M)\ \mathsf{of}\ \mathsf{inl}^A\ x \to N_l \mid \mathsf{inr}^B\ y \to N_r & \Longrightarrow & [M/y]N_r \end{array}$$

Falsehood Recall that there is no introduction rule for falsehood ($\bot$). We can therefore view it as the empty type, often written as **void** or **0**.

From a computation point of view, if we derived a contradiction, we abort the computation. Since there are no elements of the empty type, we will never be able to construct a value of type **void**; therefore, we will never be able to do any computation with it. As a consequence, there is no reduction rule for abort.

$$\frac{M : \bot}{\mathsf{abort}^C\ M : C}\ \bot E$$

To guarantee that $\mathsf{abort}^C\ M$ has a unique type, we annotate it with the proposition C.

Summary The previous discussion completes the proofs as programs interpretation for propositional logic.

Propositions	Types	
$\top$	() or $\mathbf{0}$	Unit type
$A \wedge B$	$A \times B$	Product type
$A \supset B$	$A \to B$	Function type
$A \vee B$	$A + B$	Disjoint sum type
$\bot$	void or $\mathbf{1}$	Empty type

The proof terms we introduced corresponds to the simply-typed lambda-calculus with products, disjoint sums, unit and the empty type.

$$
\begin{array}{rrcl}
\text{Terms} & M, N & ::= & x \\
& & \mid & \langle M,\ N\rangle \mid \mathsf{fst}\ M \mid \mathsf{snd}\ M \\
& & \mid & \lambda x{:}A.M \mid M\ N \\
& & \mid & \mathsf{inl}^A\ M \mid \mathsf{inr}^B\ N \mid \mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^A\ x \to N_l \mid \mathsf{inr}^B\ y \to N_r \\
& & \mid & \mathsf{abort}^A\ M \mid ()
\end{array}
$$

Remarkably, this relationship between propositions and types can be extended to richer logics. As we will see, first-order logic gives rise to dependent types; second-order logic gives rise to polymorphism and what is generally known as the calculus System F. Adding fix-points to the logic corresponds to recursive data-types in programming languages. Moving on to non-classical logics such as temporal logics their computational interpretation provides a justification for and guarantees about reactive programming; modal logics which distinguish between truths in our current world and universal truths give rise to programming languages for mobile computing and staged computation. The deep connection between logic, propositions and proofs on the one hand and type theory, types, and programs on the other provides a rich and fascinating framework for understanding programming languages, reduction strategies, and how we reason in general.

3.2 Consequences of the Curry-Howard Isomorphism

3.2.1 Proof Normalization is Program Normalization

There are many lessons we can draw from the Curry-Howard isomorphism. Its significance for logic lies in the fact that we can study normal forms both on the levels of proof terms which directly implies that the corresponding proof is also in normal form.

The natural deduction calculus admits "detours" in its proofs. For example, consider the following proof for $(A \wedge ((A \wedge A) \supset B)) \supset B$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge (A \wedge A) \supset B\ \mathsf{true}}\,u}{(A \wedge A) \supset B\ \mathsf{true}}\wedge E_l \quad \dfrac{\dfrac{}{A\ \mathsf{true}}\,v \quad \dfrac{}{A\ \mathsf{true}}\,v}{A \wedge A\ \mathsf{true}}\wedge I}{B\ \mathsf{true}}\supset E}{A \supset B\ \mathsf{true}}\supset^{v} \quad \dfrac{\dfrac{}{A \wedge (A \wedge A) \supset B\ \mathsf{true}}\,u}{A\ \mathsf{true}}\wedge E_r}{B\ \mathsf{true}}\supset E}{(A \wedge ((A \wedge A) \supset B)) \supset B\ \mathsf{true}}\supset I^{u}$$

This is a somewhat convoluted proof. In particular, it has a detour (i.e. e use $\supset I$ followed by the $\supset E$ rule) that seem unnecessary. We highlighted it in red in the derivation. We also highlight that we are proving $B\ \mathsf{true}$ twice in the derivation. A derivation without this detour would simply proceed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \wedge (A \wedge A) \supset B\ \mathsf{true}}\,u}{(A \wedge A) \supset B\ \mathsf{true}}\wedge E_l \quad \dfrac{\dfrac{\dfrac{}{A \wedge (A \wedge A) \supset B\ \mathsf{true}}\,u}{A\ \mathsf{true}}\wedge E_r \quad \dfrac{\dfrac{}{A \wedge (A \wedge A) \supset B\ \mathsf{true}}\,u}{A\ \mathsf{true}}\wedge E_r}{A \wedge A\ \mathsf{true}}\wedge I}{B\ \mathsf{true}}\supset E}{(A \wedge ((A \wedge A) \supset B)) \supset B\ \mathsf{true}}\supset I^{u}$$

Note that the height of the proof tree is smaller. The latter derivation is considered *normal*, as it does not contain any "detours".

How could we transform proof derivations into a normal form? And can we characterize normal forms more precisely?

The Curry-Howard corresponds tells us:

1. If $\mathcal{D} : A\ \mathsf{true}$ then $M : A$.
2. *Local Soundness*: If $M : A$ and $M \Longrightarrow N$ then $N : A$.
3. If $N : A$ then there exists a corresponding derivation $\mathcal{E} : A\ \mathsf{true}$. In particular the derivation $\mathcal{E}$ is a derivation where we have no detours.

Hence, we can obtain a normal form by reducing the proof term. Let's annotate the first derivation. We will omit type annotations on λ-abstractions for ease of readability.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : A \wedge (A \wedge A) \supset B}\,u}{\mathsf{snd}\ x : (A \wedge A) \supset B}\wedge E_l \qquad \dfrac{\dfrac{}{y : A}\,v \qquad \dfrac{}{y : A}\,v}{\langle y,\ y\rangle : A \wedge A}\wedge I}{(\mathsf{snd}\ x)\ \langle y,\ y\rangle : B}\supset E}{\lambda y.(\mathsf{snd}\ x)\ \langle y,\ y\rangle : A \supset B}\supset^{v} \qquad \dfrac{\dfrac{}{ptermx : A \wedge (A \wedge A) \supset B}\,u}{\mathsf{fst}\ x : A}\,u$$

$$\dfrac{\dfrac{\lambda y.(\mathsf{snd}\ x)\ \langle y,\ y\rangle\ (\mathsf{fst}\ x) : B}{\lambda x.(\lambda y.(\mathsf{snd}\ x)\ \langle y,\ y\rangle)\ (\mathsf{fst}\ x) : (A \wedge ((A \wedge A) \supset B)) \supset B}\supset I^{u}$$

We can now reduce $\lambda x.(\lambda y.(\mathsf{snd}\ x)\ \langle y,\ y\rangle)\ (\mathsf{fst}\ x)$:

$$\lambda x.(\lambda y.(\mathsf{snd}\ x)\ \langle y,\ y\rangle)\ (\mathsf{fst}\ x) \Longrightarrow \lambda x.(\mathsf{snd}\ x)\ \langle (\mathsf{fst}\ x),\ (\mathsf{fst}\ x)\rangle$$

By local soundness we know that

$$\lambda x.(\mathsf{snd}\ x)\ \langle (\mathsf{fst}\ x),\ (\mathsf{fst}\ x)\rangle : (A \wedge ((A \wedge A) \supset B)) \supset B$$

Indeed, we can check that this is the case by annotating the alternative derivation without detours that we gave earlier:

$$\dfrac{\dfrac{\dfrac{}{x : A \wedge (A \wedge A) \supset B}\,u}{\mathsf{snd}\ x : (A \wedge A) \supset B}\wedge E_l \qquad \dfrac{\dfrac{\dfrac{}{x : A \wedge (A \wedge A) \supset B}\,u}{\mathsf{fst}\ x : A}\wedge E_r \qquad \dfrac{\dfrac{}{x : A \wedge (A \wedge A) \supset B}\,u}{\mathsf{fst}\ x : A}\wedge E_r}{\langle (\mathsf{fst}\ x),\ (\mathsf{fst}\ x)\rangle : A \wedge A}\wedge I}{(\mathsf{snd}\ x)\ \langle (\mathsf{fst}\ x),\ (\mathsf{fst}\ x)\rangle : B}\supset E$$

$$\dfrac{}{\lambda x.(\mathsf{snd}\ x)\ \langle (\mathsf{fst}\ x),\ (\mathsf{fst}\ x)\rangle : (A \wedge ((A \wedge A) \supset B)) \supset B}\supset I^{u}$$

As a consequence, we can study the problem of "normalization", i.e. does there exist a normal proof in the context of does there exist a normal form for proof terms.

In fact, we can very precisely define a normal form on the level of terms as follows:

$$\begin{array}{llll} \text{Normal Terms} & M, N & ::= & \lambda x{:}A.M \mid \langle M,\ N\rangle \mid () \mid R \\ \text{Neutral Terms} & R & ::= & x \mid \mathsf{fst}\ R \mid \mathsf{snd}\ R \mid R\ N \end{array}$$

As we can see, a term $\lambda x{:}A.(\lambda y{:}A.y)\ x$ is not valid normal term, because we have a redex $(\lambda y{:}A.y)\ x$ which reduces to x. According to our grammar of normal and neutral terms $(\lambda y{:}A.y)\ x$ is ill-formed. We will describe and study these normal terms and normal derivations more in the next chapter. Fundamentally, focusing on the proof terms and how they reduce to a normal form, can be used to prove strong normalization of natural deduction, i.e., every derivation in the natural deduction can be translated

to a normal derivation. In fact following Tait [**?**] a now standard way to prove this is by defining reducibility candidates which describe sets of terms that are reducible at a given type (proposition). We will postpone such a proof to later.

3.2.2 Propositions as Types

The significance of the Curry-Howard isomorphism is maybe even greater from a type theory and programming languages perspective. Extensions of the natural deduction calculus have direct counterparts in type theory and programming languages. In fact, we can extend the previous table as follows:

Propositions	Types	
$\top$	() or **0**	Unit type
$A \wedge B$	$A \times B$	Product type
$A \supset B$	$A \to B$	Function type
$A \vee B$	$A + B$	Disjoint sum type
$\bot$	void or **1**	Empty type
$\forall x{:}\tau.A$	$\Pi x{:}\tau.A$	Indexed (Dependent) Type (Π)
$\exists x{:}\tau.A$	$\Sigma x{:}\tau.A$	Indexed (Dependent) Type (Σ)
$\forall \alpha{:}o.A$	$\forall \alpha.A$	Polymorphic Type

This correspondance extends to a variety of logics.

Logic (ND)	Programming paradigm
Propositional logic	Simply-typed functional programming
2nd-order logic	Polymorphically typed functional programming
first-order logic	Indexed (Dependently-typed) functional programming (see for example DML)
Modal Logic S4	Staged computation in functional programming
Modal Logic S5	Distributed programming
LTL	Reactive programming

We can also observe the isomorphism for sub-structural logics, i.e. logics where we omit weakening and / or contraction. Furthermore, the Curry-Howard correspondence also exists for the sequent calculus. In particular:

Logic (sequent calculus)	Programming paradigm
Linear logic	Concurrent Processes

Rather than adding features in an ad-hoc manner to a given language, we can study computational features in a systematic way by developing logical foundations grounded in the Curry-Howard isomorphism.

3.2.3 Curry-Howard's Influence on the Theory of Programming Languages

We have already seen that reductions of proof terms preserves the provability of a given proposition.

In particular:

Theorem 3.2.1 (Local Soundness aka Subject Reduction or Type Preservation)**.**
If $M : A$ *and* $M \Longrightarrow N$ *then* $N : A$.

This property is well-know in the theory of programming language under the name "Subject Reduction" or "Type Preservation". It is one part of the mantra of establishing that a language is type-safe going back to Felleisen and Wright.

Theorem 3.2.2 (Type Safety)**.**

1. *Type Preservation: If* $M : A$ *and* $M \Longrightarrow N$ *then* $N : A$
2. *Progress: If* $M : A$ *then either* M *is a normal form or we can take a step, i.e there exists a term* N *s.t.* $M \Longrightarrow N$.

3.2.4 Proofs as Programs

One important consequence of the relationship between a proof and a well-typed program, is that instead of constructing a derivation for a proposition A, we simply write a program of type A. By the Curry-Howard isomorphism, this program will be correct by construction!

As computer scientists, we are familiar with writing programs, maybe more than writing proof derivations. It is often also a lot more compact and less time consuming, to directly write the program corresponding to a given type A. We can then simply check that the program has type A, which boils down to constructing the proof tree which establishes that A is true. The good news is that such proof checking can be easily implemented by a small trusted type checker, a program of a few lines.

We've already seen some simple programs, such as the identity function, or the function which given a pair returns the first or second projection of it. Let's practice some more.

Function composition The proposition $((A \supset B) \land (B \supset C)) \supset A \supset C$ can be read computationally as function composition. Given a pair of functions, where the first element is a function $f : A \supset B$ and the second element is a function $g : B \supset C$, we can construct a function of type $A \supset C$, by assuming $x{:}A$, and then first feeding it to f, and passing the result of fx to g, i.e. returning a function $\lambda x{:}A.g\ (f\ x)$.

Since our language does not use pattern matching to access the first and second element of a pair, we write $\mathsf{fst}\ u$ instead of f and $\mathsf{snd}\ u$ instead of g, where u denotes the assumption $(A \supset B) \land (B \supset C)$.

Given this reasoning, we can write function composition as

$$\lambda u{:}(A \supset B) \land (B \supset C).\lambda x{:}A.\mathsf{snd}\ u\ ((\mathsf{fst}\ u)\ x)$$

This gives rise to two readings for $M : A$.

1. If $A\ \mathsf{true}$ then there exists a proof term M for this exact derivation s.t. $M : A$. (*Generate Proof Witness*)

2. If M and $M : A$ (M is well-typed) then there exists a proof derivation $A\ \mathsf{true}$. (*Proof Checking*)

3.2.5 Type Theory: A Foundation for Formalized Mathematics

The Curry-Howard correspondance is particular pronounced in type theories, such as Martin-öf Type Theory or the (inductive) Calculus of Construction. These system extend the correspondance to richer types (propositions) called dependent types, include inductive types, and equality. They also include a full universe hierarchy to avoid Russel's paradox.

What is important is that these type theories are implemented in proof assistants such as Rocq, Lean, and Agda. In particular in Agda, we write proofs as programs. Even in Lean and Rocq we often write programs that constitute proofs.

These proof assistants have been used widely for verifying software and mathematics. A milestone in verifying safety-critical software has been the Compcert Compiler [Leroy(2009a), Leroy(2009b)]). Proof assistants, in particular Rocq and Lean have also been used to formalize a significant portion of mathematics with the goal of establishing a new standard of rigor [Avigad and Harrison(2014)] in this field. For example, MathLib [The Mathlib Community(2020)] in the Lean proof assistant has roughly half a million lines of code, and contains formalizations of many nontrivial mathematics, from number theory to perfectoid spaces [Buzzard et al.(2020)Buzzard, Commelin, and Massot]. These endeavors are testimony to the success of proof assistants in practice.

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash N : B}{\Gamma \vdash \langle M,\ N\rangle : A \wedge B}\ \wedge I \qquad \frac{\Gamma \vdash \mathsf{fst}\ M : A \wedge B}{\Gamma \vdash M : A}\ \wedge E_l \qquad \frac{\Gamma \vdash \mathsf{snd}\ M : A \wedge B}{\Gamma \vdash M : B}\ \wedge E_r$$

$$\frac{\Gamma, u{:}A \vdash M : B}{\Gamma \vdash \lambda u{:}A.M : A \supset B}\ \supset I^u \qquad \frac{\Gamma \vdash M : A \supset B \quad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}\ \supset E$$

$$\frac{\Gamma \vdash N : A}{\Gamma \vdash \mathsf{inl}^B\ N : A \vee B}\ \vee I_l \qquad \frac{\Gamma \vdash N : B}{\Gamma \vdash \mathsf{inr}^A\ N : A \vee B}\ \vee I_r$$

$$\frac{\Gamma \vdash M : A \vee B \quad \Gamma, u : A \vdash N_l : C \quad \Gamma, v : B \vdash N_r : C}{\Gamma \vdash \mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^u\ B \to N_l \mid \mathsf{inr}^v\ A \to N_r : C}\ \vee E^{u,v}$$

$$\frac{}{\Gamma \vdash () : \top}\ \top I \qquad \frac{\Gamma \vdash \mathsf{abort}^C\ M : \bot}{\Gamma \vdash M : C}\ \bot E \qquad \boxed{\frac{u : A \in \Gamma}{\Gamma \vdash u : A}\ u}$$

$$\frac{\Gamma, a{:}\tau \vdash M : A(a)}{\Gamma \vdash \lambda a : \tau.M : \forall x{:}\tau.A(x)}\ \forall I^a \qquad \frac{\Gamma \vdash M : \forall x{:}\tau.A(x) \quad \Gamma \vdash t{:}\tau}{\Gamma \vdash M\ t : A(t)}\ \forall E$$

$$\frac{\Gamma \vdash M : A(t) \quad \Gamma \vdash t{:}\tau}{\Gamma \vdash \langle M,\ t\rangle : \exists x{:}\tau.A(x)}\ \exists I \qquad \frac{\Gamma \vdash M : \exists x{:}\tau.A(x) \quad \Gamma, a{:}\tau, u{:}A(a) \vdash N : C}{\Gamma \vdash \mathsf{let}\ \langle u,\ a\rangle = M\ \mathsf{in}\ N : C}\ \exists E^{au}$$

Figure 3.1: Summary of typing rules

3.3 Meta-theoretic properties

We consider here additional properties of the proof terms, typing and reductions. For this discussion it is useful to have all the typing and reduction rules in one place.

If we look back at our reduction rules we notice that reduction does not always take place at the top-level. A redex, i.e. a term which matches one of the left hand sides of our reduction rules, may be embedded in a given term. For example, we may want to evaluate:

$$\lambda y{:}A.\langle(\ \lambda x{:}A.x)y,\ y\rangle$$

Here the redex $(\ \lambda x{:}A.x)y$ is burried underneath a lambda-abstraction and a pair. In order to allow reduction of a redex which is not at the top-level, we need to introduce additional reduction rules for $M \Longrightarrow M'$ allowing us to get to the redex inside another term. This is accomplished by so-called *congruence rules*. Note our congruence rules are non-deterministic; they also reduce under a lambda-abstraction. Both of

these characteristics may not be wanted if we are to define a determinstic call-by-value evaluation strategy. However, at this point, we retain as much flexibility as possible.

3.3.1 Subject reduction

We prove here a key property: Subject reduction.

Theorem 3.3.1. *If $M \Longrightarrow M'$ and $\Gamma \vdash M : C$ then $\Gamma \vdash M' : C$.*

Proof. By structural induction on $M \Longrightarrow M'$.

The reduction rules for each redex form the base cases in the proof. We consider here the rule for reducing $(\lambda x{:}A.M)\ N$ one as an example.

Case $\mathcal{D} = (\lambda x{:}A.M)\ N \implies [N/x]M$

$\Gamma \vdash (\lambda x{:}A.M)\ N : C$	by assumption
$\Gamma \vdash \lambda x{:}A.M : A' \supset C$	
$\Gamma \vdash N : A'$	by rule $\supset E$
$\Gamma, x : A \vdash M : C$ and $A = A'$	by rule $\supset I^x$
$\Gamma \vdash [N/x]M : C$	by substitution lemma

We next consider a representative from the step cases which arise due to the congruence rules.

Case $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}' \\ M \Longrightarrow M'\end{array}}{\lambda x{:}A.M \Longrightarrow \lambda x{:}A.M'}$

$\Gamma \vdash \lambda x{:}A.M : C$	by assumption
$\Gamma, x : A \vdash M : B$ and $C = A \supset B$	by rule $\supset I^x$
$\Gamma, x : A \vdash M' : B$	by i.h. on $\mathcal{D}'$
$\Gamma \vdash \lambda x{:}A.M' : A \supset B$	by rule $\supset I^x$

□

3.3.2 Equivalence of Proof Systems

We have so far described the elimination rules for conjunction using the projections fst and snd to extract each component of a pair.

Reduction rules for redexes

$$
\begin{array}{lcl}
\mathsf{fst}\ \langle M,\ N\rangle & \Longrightarrow & M \\
\mathsf{snd}\ \langle M,\ N\rangle & \Longrightarrow & N \\
(\lambda x{:}A.M)\ N & \Longrightarrow & [N/x]M \\
\mathsf{case}\ (\mathsf{inl}^A\ M)\ \mathsf{of}\ \mathsf{inl}^A\ x \to N_l \mid \mathsf{inr}^B\ y \to N_r & \Longrightarrow & [M/x]N_l \\
\mathsf{case}\ (\mathsf{inr}^B\ M)\ \mathsf{of}\ \mathsf{inl}^A\ x \to N_l \mid \mathsf{inr}^B\ y \to N_r & \Longrightarrow & [M/y]N_r \\
(\lambda a{:}\tau.M)\ t & \Longrightarrow & [t/a]M \\
\mathsf{let}\ \langle u,\ a\rangle = \langle M,\ t\rangle\ \mathsf{in}\ N & \Longrightarrow & [M/u][t/a]M
\end{array}
$$

Congruence rules

$$\frac{M \Longrightarrow M'}{\langle M,\ N\rangle \Longrightarrow \langle M',\ N\rangle} \qquad \frac{N \Longrightarrow N'}{\langle M,\ N\rangle \Longrightarrow \langle M,\ N'\rangle} \qquad \frac{M \Longrightarrow M'}{\mathsf{fst}\ M \Longrightarrow \mathsf{fst}\ M'} \qquad \frac{M \Longrightarrow M'}{\mathsf{snd}\ M \Longrightarrow \mathsf{snd}\ M'}$$

$$\frac{M \Longrightarrow M'}{\lambda x{:}A.M \Longrightarrow \lambda x{:}A.M'} \qquad \frac{M \Longrightarrow M'}{M\ N \Longrightarrow M'\ N} \qquad \frac{N \Longrightarrow N'}{M\ N \Longrightarrow M\ N'}$$

$$\frac{M \Longrightarrow M'}{\mathsf{inl}^B\ M \Longrightarrow \mathsf{inl}^B\ M'} \qquad \frac{M \Longrightarrow M'}{\mathsf{inr}^A\ M \Longrightarrow \mathsf{inr}^A\ M'}$$

$$\frac{M \Longrightarrow M'}{\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^B\ u \to N_l \mid \mathsf{inr}^A\ v \to N_r \Longrightarrow \mathsf{case}\ M'\ \mathsf{of}\ \mathsf{inl}^B\ u \to N_l \mid \mathsf{inr}^A\ v \to N_r}$$

$$\frac{N_l \Longrightarrow N_l'}{\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^B\ u \to N_l \mid \mathsf{inr}^A\ v \to N_r \Longrightarrow \mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^B\ u \to N_l' \mid \mathsf{inr}^A\ v \to N_r}$$

$$\frac{N_r \Longrightarrow N_r'}{\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^B\ u \to N_l \mid \mathsf{inr}^A\ v \to N_r \Longrightarrow \mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^B\ u \to N_l' \mid \mathsf{inr}^A\ v \to N_r'}$$

Figure 3.2: Summary of reduction rules

$$\frac{M : A \qquad N : B}{\langle M,\ N\rangle : A \wedge B}\wedge I \qquad \frac{M : A \wedge B}{\mathsf{fst}\ M : A}\wedge E_l \qquad \frac{M : A \wedge B}{\mathsf{snd}\ M : B}\wedge E_r$$

However, we might also be able to think of using a different rule for conjunction elimination:

$$\frac{\begin{array}{cc} & \begin{array}{cc}\overline{}\,u & \overline{}\,v\\ x : A & y : B\end{array}\\ & \vdots\\ M : A \wedge B & N : C\end{array}}{\mathsf{let}\ \langle x,\ y\rangle = M\ \mathsf{in}\ N : C}\wedge E{-}\mathsf{let}$$

Let's look at a few examples and compare the proof terms:

Proposition	Proof term using projections	Proof term using let
$(A \wedge B) \supset (B \wedge A)$	$\lambda x.\langle \mathsf{snd}\ x,\ \mathsf{fst}\ x\rangle$	$\lambda x.\mathsf{let}\ \langle a,\ b\rangle = x\ \mathsf{in}\ \langle b,\ a\rangle$
$((A \supset B) \wedge A) \supset B$	$\lambda x.(\mathsf{fst}\ x)\ (\mathsf{snd}\ x)$	$\lambda x.\mathsf{let}\ \langle a,\ b\rangle = x\ \mathsf{in}\ a\ b$

How can we translate between proof terms that use a let-expression to one that uses only projections? - One might view this as compiling a term that contains a let-expression (source term) to a term that only contains projectsion (target term). By giving such a translation between terms, we also obtain at the same time a translation of proofs from the system that uses the elimination rule $\wedge E{-}\mathsf{let}$ to the one that uses $\wedge E_l$ and $\wedge E_r$ to

For clarity, we keep the source language separate from the target language and we will use different variables. For the target language we will use T, while we stick to using M or N for the source language.

$$\begin{array}{lcl}\text{Source Language } M, N & := & x \mid \langle M,\ N\rangle \mid \mathsf{let}\ \langle x,\ y\rangle = M\ \mathsf{in}\ N \mid \ldots\\ \text{Target Language } T & := & x \mid \langle T_1,\ T_2\rangle \mid \mathsf{fst}\ T \mid \mathsf{snd}\ T \mid \ldots\end{array}$$

We define here the translation from a language with let-expression using a function $(\ )^-$ which takes a source term or a context of source variables and translates it to its corresponding target term or target context. The translation is defined inductively on the structure of source terms and source contexts, resp.

$$
\begin{array}{lcll}
(x)^- & = & x & \\
(\langle M,\ N\rangle)^- & = & \langle T_1,\ T_2\rangle & T_1 = (M)^- \text{ and } T_2 = (N)^- \\
(\mathsf{let}\ \langle x,\ y\rangle = M\ \mathsf{in}\ N)^- & = & [\mathsf{fst}\ T_1/x,\ \mathsf{snd}\ T_1/y]T_2 & T_1 = (M)^- \text{ and } T_2 = (N)^- \\
 & & & \\
(\cdot)^- & = & \cdot & \\
(\Gamma, x : A)^- & = & (\Gamma)^-, x : A &
\end{array}
$$

The translation of contexts might seem unnecessary and indeed it is pedantic. Morally, the variables in Γ describe source variables and the variables in $(\Gamma)^-$ describe target variables. While we have used the same variable names for both source and target variables, they belong to different languages. The explicit copy of Γ makes this clear.

How can we show that provability is preserved, i.e. if $M : A$ in the system with $\wedge E{-}\mathsf{let}$ then there must be a corresponding proof $T : A$ in the system with $\wedge E_l$ and $\wedge E_r$. Note that as we translate M and construct the proof $M : A$, we reason by assumptions.

Theorem 3.3.2. *If $\Gamma \vdash M : A$ then $(\Gamma)^- \vdash (M)^- : A$.*

Proof. Induction on the typing derivation $\Gamma \vdash M : A$.

Case $\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma \vdash M : A \wedge B & \Gamma, x : A, y : B \vdash N : C\end{array}}{\Gamma \vdash \mathsf{let}\ \langle x,\ y\rangle = M\ \mathsf{in}\ N : C}\ \wedge E{-}\mathsf{let}$

$(\Gamma)^- \vdash (M)^- : A \wedge B$ by IH on $\mathcal{D}_1$
$(\Gamma)^- \vdash \mathsf{fst}\ (M)^- : A$ by $\wedge E_l$
$(\Gamma)^- \vdash \mathsf{snd}\ (M)^- : A$ by $\wedge E_r$
$((\Gamma, x : A, y : B))^- \vdash (N)^- : C$ by IH on $\mathcal{D}_2$
$(\Gamma)^-, x : A, y : B \vdash (N)^- : C$ by definition of $(\ )^-$
$(\Gamma)^- \vdash [\mathsf{fst}\ (M)^-/x,\ \ \mathsf{snd}\ (M)^-/y]((N)^-) : C$ by substitution lemma
$(\Gamma)^- \vdash (\mathsf{let}\ \langle x,\ y\rangle = M\ \mathsf{in}\ N)^- : C$ by definition of $(\ )^-$

Case $\mathcal{D} = \dfrac{\begin{array}{cc}\mathcal{D}_1 & \mathcal{D}_2 \\ \Gamma \vdash M : A & \Gamma \vdash N : B\end{array}}{\Gamma \vdash \langle M,\ N\rangle : A \wedge B}\ \wedge I$

$(\Gamma)^- \vdash (M)^- : A$ by IH $\mathcal{D}_1$

$(\Gamma)^- \vdash (N)^- : B$ by IH $\mathcal{D}_1$
$(\Gamma)^- \vdash \langle (M)^-,\ (N)^- \rangle : A \wedge B$ by $\wedge I$
$(\Gamma)^- \vdash (\langle M,\ N \rangle)^- : A \wedge B$ by definition of $(\ )^-$.
□

3.4 Exercises

Exercise 3.4.1. We mentioned that there is an alternative formulation for eliminating a conjunction that incorporates a light form of pattern matching.

However, we might also be able to think of using a different rule for conjunction elimination while keeping the introduction form for conjunction.

$$\frac{M : A \qquad N : B}{\langle M,\ N \rangle : A \wedge B}\wedge I$$

$$\frac{\begin{array}{cc} & \dfrac{}{x : A}\,u \quad \dfrac{}{y : B}\,v \\ & \vdots \\ M : A \wedge B & N : C \end{array}}{\mathsf{let}\ \langle x,\ y \rangle = M\ \mathsf{in}\ N : C}\wedge E{-}\mathsf{let}$$

- Reformulate these rules with local contexts
- Show local soundness and derive the corresponding reduction rule.

Bibliography

[Avigad and Harrison(2014)] Jeremy Avigad and John Harrison. Formally verified mathematics. *Commun. ACM*, 57(4):66–75, 2014.

[Belnap(1962)] Nuel D. Belnap. Tonk, plonk, and plink. *Analysis*, 22(6):130–134, 1962.

[Buzzard et al.(2020)Buzzard, Commelin, and Massot] Kevin Buzzard, Johan Commelin, and Patrick Massot. Formalising perfectoid spaces. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page 299–312, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450370974. doi: 10.1145/3372885.3373830. URL `https://doi.org/10.1145/3372885.3373830`.

[Dummett(1993)] Michael Dummett. *The Logical Basis of Metaphysics*. Harvard University Press, Cambridge, Massachusetts, 1993.

[Gentzen(1935)] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 1935.

[Leroy(2009a)] Xavier Leroy. Formal verification of a realistic compiler. *Communications of the ACM*, 52(7):107–115, 2009a.

[Leroy(2009b)] Xavier Leroy. A formally verified compiler back-end. *J. Autom. Reasoning*, 43(4):363–446, 2009b.

[Martin-L"of(1994)] Per Martin-L"of. Analytic and Synthetic Judgements in Type Theory. In Paolo Parrini, editor, *Kant and Contemporary Epistemology*, pages 87–99. Kluwer Academic Publishers, 1994.

[The Mathlib Community(2020)] The Mathlib Community. The lean mathematical library. In Jasmin Blanchette and Catalin Hritcu, editors, *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans, Louisiana, USA, January 20-21, 2020*, pages 367–381.

ACM, 2020. doi: 10.1145/3372885.3373824. URL https://doi.org/10.1145/3372885.3373824.

[Wadler(2000)] Philip Wadler. Proofs are programs: 19th century logic and 21st century computing. *Dr. Dobbs Journal (Special supplement on Software in the 21st century)*, pages 1–14, 2000. URL https://homepages.inf.ed.ac.uk/wadler/papers/frege/frege.pdf.

[Wadler(2015)] Philip Wadler. Propositions as types. *Commun. ACM*, 58(12):75–84, November 2015. ISSN 0001-0782. doi: 10.1145/2699407. URL https://doi.org/10.1145/2699407.