Logic and Computation

Brigitte Pientka

School of Computer Science
McGill University
Montreal, Canada

Contents

Chapter 1

Introduction

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. 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). However, the sequent calculus is interesting in its own since we can further study the structure of proofs and gain insights into how to eliminate further redundancy in proofs leading to focused sequent calculi which have been for example used to provide a type-theoretic foundation for pattern matching and different evaluation strategies. The second approach show consistency concentrates on studying the structure of programs and we prove that every program normalizes using logical relations following Tait. This is a more direct proof than the syntactic cut-elimination proof which relies on proof translations.

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. To define the meaning of a proposition we use a judgement "A true". There are many other judgements one might think of defining: A false (to define when a proposition A is false), A $\mathsf{possible}$ (to define when a proposition is possible typical in modal logics), or simply A prop (to define when a proposition A is well-formed giving us an alternative to BNF grammars).

2.2 Judgements and Meaning

We are here concerned with defining the meaning of a proposition A by defining when it is true. To express the meaning of a given proposition we use 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*. We now define the meaning of each connective in turn.

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).

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

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

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

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

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.

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

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.

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

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

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

While the use of the rule $\wedge l$ and $\wedge r$ is correct, $A \wedge (B \wedge C)$ true is unjustified. It is certainly not true unconditionally. However, we might want to say that we can derive B true, *given the assumption* $A \wedge (B \wedge C)$ 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

$$\dfrac{\begin{matrix} \overline{J_1} & \dots & \overline{J_n} \\ & \vdots & \end{matrix}}{J}$$

We can derive J given the assumptions $J_1, \dots, J_n$. Note, that we make no claims as to whether we can in fact prove $J_1, \dots, J_n$; 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.

$$\dfrac{\begin{array}{c}\overline{A\ \mathsf{true}}\ u\\ \vdots\\ B\ \mathsf{true}\end{array}}{A \supset B\ \mathsf{true}} \supset I^u$$

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.

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

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.

$$\frac{A \supset B \;\mathsf{true} \quad A \;\mathsf{true}}{B \;\mathsf{true}} \supset E$$

A few examples using hypothetical derivations We give here a few examples. Consider first constructing a derivation for $(A \wedge B) \supset (B \wedge A)$ 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 $(A \wedge B) \supset (B \wedge A)$ true.

$$\begin{array}{c} \dfrac{}{A \wedge B \;\mathsf{true}}\, u \\ \vdots \\ \dfrac{\dfrac{B \;\mathsf{true} \quad A \;\mathsf{true}}{B \wedge A \;\mathsf{true}} \wedge I}{(A \wedge B) \supset (B \wedge A) \;\mathsf{true}} \supset I^u \end{array}$$

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 $\wedge r$.

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

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.

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\dfrac{}{A \supset (B \wedge C)\ \mathsf{true}}\,u \qquad \dfrac{}{A\ \mathsf{true}}\,v \\ \vdots \\ B\ \mathsf{true}\end{array}}{A \supset B\ \mathsf{true}}\,\supset I^v \qquad \dfrac{\begin{array}{c}\dfrac{}{A \supset (B \wedge C)\ \mathsf{true}}\,u \qquad \dfrac{}{A\ \mathsf{true}}\,v \\ \vdots \\ C\ \mathsf{true}\end{array}}{A \supset C\ \mathsf{true}}\,\supset I^v}{(A \supset B) \wedge (A \supset C)\ \mathsf{true}}\,\wedge I}{(A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))\ \mathsf{true}}\,\supset I^u$$

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

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

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.

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

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

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

This would allow us to obtain a proof for A from the information $A \vee B\ \mathsf{true}$; but if we know $A \vee B\ \mathsf{true}$, 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 $A \vee B\ \mathsf{true}$, then we consider two cases: $A\ \mathsf{true}$ and $B\ \mathsf{true}$. If in both cases we can establish $C\ \mathsf{true}$, then it must be the case that C is true!

$$\cfrac{A \vee B\ \mathsf{true} \qquad \begin{array}{c}\overline{A\ \mathsf{true}}\ u\\ \vdots\\ C\ \mathsf{true}\end{array} \qquad \begin{array}{c}\overline{B\ \mathsf{true}}\ u\\ \vdots\\ C\ \mathsf{true}\end{array}}{C\ \mathsf{true}}\ \vee E^{u,v}$$

We again use hypothetical judgement to describe the rule for disjunction. Note the scope of the assumptions. The assumption $A\ \mathsf{true}$ labelled u can only be used in the middle premise, while the assumption $B\ \mathsf{true}$ 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.

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

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 $A\ \mathsf{true}$ or $B\ \mathsf{true}$. Instead, we will use our assumption $A \vee B\ \mathsf{true}$ and then prove $A \vee B\ \mathsf{true}$ under the assumption $A\ \mathsf{true}$ and separately prove $A \vee B\ \mathsf{true}$ under the assumption $B\ \mathsf{true}$.

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

Falsehood Last but not least, we consider the rule for falsehood (written as $\perp$). Clearly, we should never be able to prove (directly) $\perp$. Hence there is no introduction rule which introduces $\perp$. However, we might nevertheless derive $\perp$ (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 $\perp$, then we are able to conclude anything from it, since we have arrived at a contradiction.

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

It might not be obvious that $\perp$ is very useful. It is particularly important in allowing us to define $\neg A$ (read as "not A) as $A \supset \perp$. 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

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

and symmetrically

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

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 $A \wedge B$ true contains enough information to rebuild a proof for $A \wedge B$ true.

$$\begin{matrix}\mathcal{D} \\ A \wedge B \text{ true}\end{matrix} \quad \Longrightarrow \quad \dfrac{\dfrac{\begin{matrix}\mathcal{D} \\ A \wedge B \text{ true}\end{matrix}}{A \text{ true}} \wedge E_l \qquad \dfrac{\begin{matrix}\mathcal{D} \\ A \wedge B \text{ true}\end{matrix}}{B \text{ true}} \wedge E_r}{A \wedge B \text{ true}} \wedge I$$

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.

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

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

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

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 $A \supset B$ true given a proof for it?

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

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 $\mathcal{D}$.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A\end{array}}{A \vee B}\,\vee I_l \qquad \begin{array}{c}\overline{A\ \mathsf{true}}^{\,u}\\ \mathcal{E}\\ C\ \mathsf{true}\end{array} \qquad \begin{array}{c}\overline{B\ \mathsf{true}}^{\,u}\\ \mathcal{F}\\ C\ \mathsf{true}\end{array}}{C\ \mathsf{true}}\,\vee E^{u,v} \quad \Longrightarrow \quad \begin{array}{c}\mathcal{D}\\ \hline A\ \mathsf{true}\\ \mathcal{E}\\ C\ \mathsf{true}\end{array}$$

Similarly, we can show

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ B\end{array}}{A \vee B}\,\vee I_r \qquad \begin{array}{c}\overline{A\ \mathsf{true}}^{\,u}\\ \mathcal{E}\\ C\ \mathsf{true}\end{array} \qquad \begin{array}{c}\overline{B\ \mathsf{true}}^{\,u}\\ \mathcal{F}\\ C\ \mathsf{true}\end{array}}{C\ \mathsf{true}}\,\vee E^{u,v} \quad \Longrightarrow \quad \begin{array}{c}\mathcal{D}\\ \hline B\ \mathsf{true}\\ \mathcal{F}\\ C\ \mathsf{true}\end{array}$$

Local completeness

$$\begin{array}{c}\mathcal{D}\\ A \vee B\ \mathsf{true}\end{array} \quad \Longrightarrow \quad \dfrac{\begin{array}{c}\mathcal{D}\\ A \vee B\ \mathsf{true}\end{array} \qquad \dfrac{\overline{A\ \mathsf{true}}^{\,u}}{A \vee B\ \mathsf{true}}\,\vee I_l \qquad \dfrac{\overline{B\ \mathsf{true}}^{\,v}}{A \vee B\ \mathsf{true}}\,\vee I_r}{A \vee B\ \mathsf{true}}\,\vee E^{u,v}$$

2.5 Linear notation for proofs

Writing proof trees on paper is the standard in the literature on logic. However, this approach is neither well-suited for writing actual proofs for complex propositions (because proofs become unwieldy) nor is it the best way to learn proving propositions (because you have to wait a while until we have graded your proof!). We introduce briefly here a niftly little proof checker, Tutch, which allows users to type up their proofs in a linear format in ascii. Tutch will then verify that each step in your proof can be justified by one of the rules of the natural deduction calculus. If they are all justified, the proof passes. If there are some steps which are not justified, Tutch flags the offending sub-expression. As a consequence, you get instant feedback when developing proofs.

Step 1: Instal Tutch Please see instructions on the course web-site.

Step 2: Logical symbols in Tutch Since logical symbols are not available on a keyboard, Tutch uses the following concrete syntax for propositions:

$A \equiv B$	`A<=>B`	A if and only if B
$A \supset B$	`A=>B`	A implies B
$A \vee B$	`A|B`	A or B
$A \wedge B$	`A & B`	A and B
$\neg A$	`A`	not A

We list the operators in order of increasing binding strength. Moreover, implication (`=>`), disjunction (`|`), and conjunction (`&`) associate to the right, just like the corresponding notation from earlier in this chapter.

The linear format is mostly straightforward. A proof is written as a sequence of judgments separated by semi-colon `;`. Typically, we write one judgment followed by semi-colon per line. Later judgments must follow from earlier ones by a single inference rule. Since we typically can easily verify what rule justifies what line, you do not have to give explicitly the name of the rule which justifies an inference. To simplyfy matters further, we write for A true simply `A` omitting true.

To write hypothetical proofs such as

$$\begin{array}{c} \overline{A\ \mathsf{true}}^{\,u} \\ \vdots \\ C\ \mathsf{true} \end{array}$$

we write

```
[A;

 C];
```

The square brackets delimit the scope of the assumption `A`. If we need more than one hypothesis, we can nest scopes. Let's walk through an example. We want to prove: $(A \supset B) \wedge (B \supset C) \supset A \supset C$ true.

In emacs we begin by stating the problem as follows:

```
proof distAnd1 : (A => B) & (B => C) => (A => C) =
begin




(A => B) & (B => C) => (A => C)
end;
```

We first give the conjecture the name `distAnd1` and translate it into notation Tutch understands. We note that `&` binds tighter than `->`. The actual proof goes between `begin` and `end`. At the bottom we state our conjecture, which we need to prove.

Our first step is to use the rule $\supset I^u$ rule, which introduces the assumption `(A => B) & (B => C)`. In the Tutch proof, we open a scope using `[ .... ]`. We write comments using %.

```
proof distAnd1 : (A => B) & (B => C) => (A => C) =
begin
[(A => B) & (B => C);            % Assumption u




 (A => C) ];
(A => B) & (B => C) => (A => C)  % ImpI (discharge u)
end;
```

Next, we introduce the assumption `A` and prove `C` applying again the rule $\supset I^v$.

```
proof distAnd1 : (A => B) & (B => C) => (A => C) =
begin
[(A => B) & (B => C);            % Assumption u
 [A ;                            % Assumption v




  C ] ;
 (A => C) ];                       % ImpI (discharge v)
(A => B) & (B => C) => (A => C)  % ImpI (discharge u)
end;
```

To prove C, we can use the inner assumption A, but also the outer assumption (A => B) & (B => C).

We cannot proceed further from the bottom up; we hence look at our assumptions and see what valid inferences we can draw. For example, using the assumption (A => B) & (B => C), we can justify A => B by rule $\wedge E_l$.

```
proof distAnd1 : (A => B) & (B => C) => (A => C) =
begin
[(A => B) & (B => C);              % Assumption u
 [A ;                              % Assumption v
  A => B;                          % andeL



  C ] ;
 (A => C) ];                       % ImpI (discharge v)
(A => B) & (B => C) => (A => C)  % ImpI (discharge u)
end;
```

Using the assumption A and the just derived judgment A => B, we can conclude B by the rule $\supset E$.

```
proof distAnd1 : (A => B) & (B => C) => (A => C) =
begin
[(A => B) & (B => C);              % Assumption u
 [A ;                              % Assumption v
  A => B;                          % AndeL
  B;                               % ImpEl

  C ] ;
 (A => C) ];                       % ImpI (discharge v)
(A => B) & (B => C) => (A => C)  % ImpI (discharge u)
end;
```

Furthermore, we can justify B => C from the assumption (A => B) & (B => C) by the rule $\wedge E_r$

```
proof distAnd1 : (A => B) & (B => C) => (A => C) =
begin
[(A => B) & (B => C);              % Assumption u
 [A ;                              % Assumption v
  A => B;                          % AndeL
  B;                               % ImpEl
  B => C;                          % AndeR
  C ] ;
 (A => C) ];                       % ImpI (discharge v)
(A => B) & (B => C) => (A => C)   % ImpI (discharge u)
end;
```

Since we can also justify C from the judgment B and B => C using the rule $\supset E$, we are done.

More examples are available on the course website.

2.5.1 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.

$$\dfrac{\begin{array}{c}\dfrac{}{A \text{ true}}u \\ \vdots \\ p \text{ true}\end{array}}{\neg A \text{ true}}\neg I^{p,u} \qquad \dfrac{\neg A \text{ true} \qquad A \text{ true}}{C \text{ true}}\neg E$$

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

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

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.

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

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.

2.6 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 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 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

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

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

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

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

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

Figure 2.1: Natural Deduction with Explicit Context for Assumptions

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. We interpret all our inference rules within the context Γ (see Fig. 2.1).

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 2.6.1.

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

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

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

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

Theorem 2.6.2 (Substitution).
If $\Gamma, x : A\ \mathsf{true}, \Gamma' \vdash B\ \mathsf{true}$ *and* $\Gamma \vdash A\ \mathsf{true}$ *then* $\Gamma, \Gamma' \vdash B\ \mathsf{true}$.

2.7 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 2.7.1. *If* $\Gamma, u : A\ \mathsf{true}, \Gamma' \vdash C\ \mathsf{true}$ *and* $\Gamma \vdash A\ \mathsf{true}$ *then* $\Gamma, \Gamma' \vdash C\ \mathsf{true}$.

Proof. By structural induction on the derivation $\Gamma, u : A\ \mathsf{true}, \Gamma' \vdash C\ \mathsf{true}$. 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 $\mathcal{D} = \dfrac{}{\Gamma, u : A\ \mathsf{true}, \Gamma' \vdash \top\ \mathsf{true}}\ \top I.$

$\Gamma, \Gamma' \vdash \top\ \mathsf{true}$ by $\top I$

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

$\Gamma \vdash A\ \mathsf{true}$ by assumption
$\Gamma, \Gamma' \vdash A\ \mathsf{true}$ by weakening

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

$\Gamma, \Gamma' \vdash C\ \mathsf{true}$ 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.

$$\textbf{Case}\quad \mathcal{D} = \dfrac{\overset{\mathcal{F}}{\Gamma, u : A\ \mathsf{true}, \Gamma' \vdash C\ \mathsf{true}} \qquad \overset{\mathcal{E}}{\Gamma, u : A\ \mathsf{true}, \Gamma' \vdash B\ \mathsf{true}}}{\Gamma, u : A\ \mathsf{true}, \Gamma' \vdash C \wedge B\ \mathsf{true}} \wedge I$$

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

$$\textbf{Case}\quad \mathcal{D} = \dfrac{\overset{\mathcal{E}}{\Gamma, u : A\ \mathsf{true}, \Gamma', v{:}B\ \mathsf{true} \vdash C\ \mathsf{true}}}{\Gamma, u : A\ \mathsf{true}, \Gamma' \vdash B \supset C\ \mathsf{true}} \supset I^v$$

$\Gamma \vdash A\ \mathsf{true}$	by assumption
$\Gamma, \Gamma', v : B\ \mathsf{true} \vdash C\ \mathsf{true}$	by i.h. using $\mathcal{E}$
$\Gamma, \Gamma' \vdash B \supset C\ \mathsf{true}$	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. □

2.8 Exercises

Exercise 2.8.1. Assume someone defines conjunction with the following two rules:

$$\dfrac{A \wedge B \qquad \begin{array}{c} \dfrac{}{A}u \quad \dfrac{}{B}v \\ \vdots \\ C \end{array}}{C} \wedge E^{u,v} \qquad\qquad \dfrac{A \qquad B}{A \wedge B} \wedge I$$

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

Exercise 2.8.2. 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 2.8.3. $A \overline{\wedge} B$ 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 2.8.4. 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, Gentzens natural deduction and Churchs lambda calculus are on a par with Einsteins relativity and Diracs 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 [**?**]

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.

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 that that the derivation for $M : A$ has the identical structure as the derivation for $A\ \mathsf{true}$. This is stronger than simply whenever $M : A$ then $A\ \mathsf{true}$ and vice versa. We will revisit our natural deduction rules and annotate them with proof terms. The isomorphism between $M : A$ and $A\ \mathsf{true}$ will then become obvious.

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

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

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:

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

and dually

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

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 value which is the result of the computation (or until we are stuck). 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: $\mathsf{fst}\ \langle M,\ N\rangle \Longrightarrow M$ and $\mathsf{snd}\ \langle M,\ N\rangle \Longrightarrow N$. We will more elaborate on reductions and their theoretical properties later.

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

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

$\top$ in type theory 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.

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

We distinguish here in the derivation between the variable x corresponding to A and the assumption which states x has type A. Here x is a proof term, while u the name of the assumption $x : A$. Consider the trivial proof for $(A \land A) \supset A$ true.

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

A different proof where we extract the right A from $A{\land}A$, can results in a different proof term.

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

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 $M\ N$.

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

An implications $A \supset B$ can be interpreted as a function type $A \rightarrow B$.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 $(A \land B) \supset (B \land A)$ whose proof we've seen earlier as

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \land B}\,\land E_r}{B\ \mathsf{true}} \qquad \dfrac{\dfrac{}{A \land B}\,\land E_l}{A\ \mathsf{true}}}{B \land A)\ \mathsf{true}}\,\land I}{(A \land B) \supset (B \land A)\ \mathsf{true}}\,\supset I^u$$

We now annotate the derivation with proof terms.

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

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

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

This gives rise to the reduction rule for function applications:

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

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.

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

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.

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

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 $\mathsf{case}\ M\ \mathsf{of}\ \mathsf{inl}^A\ x \rightarrow N_l \mid \mathsf{inr}^B\ y \rightarrow N_r$. 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:

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

Falsehood Recall that there is no introduction rule for falsehood ($\perp$). 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 **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

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

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

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 Proving = Programming

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. In fact, Tutch gives you the option of either writing a proof for a proposition `A`, writing an annotated proof of a proposition `A`, or simply writing a term whose type is `A`.

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 $((A \supset B) \wedge (B \supset C)) \supset A \supset C$ 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 $f x$ 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 $(A \supset B) \wedge (B \supset C)$.

Given this reasoning, we can write function composition as

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

3.3 Proof terms for first-order logic

Similar to proof terms for propositional logic, we can introduce proof terms for quantifiers. The proof term for introducing an existential quantifier, encapsulates the witness t together with the actual proof M. It is hence similar to a pair and we write it as $\langle M,\ t\rangle$ overloading the pair notation. The elimination rule for existentials is modeled by $\mathsf{let}\ \langle u, a,\ =\rangle M\ \mathsf{in}\ N$ where M is the proof for $\exists x{:}\tau.A(x)$ and N is the proof depending on the assumption $u : A(a)$ and $a : \tau$.

The proof term for introducing a universal quantifier is modeled by lambda-abstaction. Elimination is modeled by application. We again overload abstaction and application.

$$\text{Terms}\quad M, N \quad ::= \quad \lambda a{:}\tau.M \mid M\ t \mid \langle M,\ t\rangle \mid \mathsf{let}\ \langle u,\ a\rangle = M\ \mathsf{in}\ N$$

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

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

We obtain two additional reduction rules.

$$\begin{array}{lcl}
(\lambda a{:}\tau.M)\ t & \Longrightarrow & [t/a]M \\
\mathsf{let}\ \langle u,\ a\rangle = \langle M,\ t\rangle\ \mathsf{in}\ N & \Longrightarrow & [M/u][t/a]M
\end{array}$$

Note that we also overload our substitution operation writing $[M/u]$ to replace a proof assumption u with the proof term M and writing $[t/a]$ to replace a parameter a with the term t from our reasoning domain. We assume that substitution is capture-avoiding.

3.4 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

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

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

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

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

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

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

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

Figure 3.1: Summary of typing rules

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.

Exercise 3.4.1.

Define corresponding congruence rules for universal and existentials.

3.4.1 Subject reduction

We prove here a key property: Subject reduction.

Theorem 3.4.1. *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 $\mathcal{D} = (\lambda x{:}A.M)\ N \Longrightarrow [N/x]M$

$\Gamma \vdash (\lambda x{:}A.M)\ N : C$ 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 $$\mathcal{D} = \dfrac{\overset{\mathcal{D}'}{M \Longrightarrow M'}}{\lambda x{:}A.M \Longrightarrow \lambda x{:}A.M'}$$

$\Gamma \vdash \lambda x{:}A.M : C$ 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 $\supset I^x$

□

3.4.2 Type Uniqueness

Reduction rules for redexes

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

Congruence rules

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

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

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

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

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

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

Figure 3.2: Summary of reduction rules