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

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

where $J_1, \dots, J_n$ are called the *premises* and J is called the *conclusion*. We can read the inference rule as follows: Given the premises $J_1, \dots, J_n$, 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).

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

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

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

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

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.

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

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.

$$\frac{\frac{}{\top \text{ true}} \top I \quad \frac{\frac{}{\top \text{ true}} \top I \quad \frac{}{\top \text{ true}} \top I}{\top \wedge \top \text{ true}} \wedge I}{\top \wedge (\top \wedge \top) \text{ true}} \wedge I$$

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

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

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.

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

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.

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

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.

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

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.

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

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

$$\dfrac{A \lor B\ \mathsf{true}}{A\ \mathsf{true}}\ \text{BAD RULE}$$

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

Thus we take a different approach. If we know $A \lor B$ 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!

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

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.

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

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 $A \lor B$ true and then prove $A \lor B$ true under the assumption A true and separately prove $A \lor B$ true under the assumption B true.

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

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.

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

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

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

and symmetrically

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

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

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

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.

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

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?

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

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

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

2.4.4 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.

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

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

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

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.

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

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.

2.5 First-order Logic

So far, we have considered propositional logic and the programming language arising from it is very basic. It does not allow us to reason about data-types such as natural numbers or booleans for example.

In this chapter, we develop first-order logic which allows us to quantify over data. This will allow us to reason about data and from a proof about a given property we are able to extract a programs manipulating data. The resulting program is correct-by-construction. In practice, we rarely formally prove our programs to be correct - for real programs with mutable state or concurrency the specification of what a program is supposed to do may be challenging. Moreover, we cannot mechanically and automatically establish that a program satisfies a given invariant. However, partial invariants are useful in practical programming.

2.5.1 Universal and Existential Quantification

In this section, we introduce logical quantifiers. We extend our grammar for logical formulae with universal quantification, written as $\forall x{:}\tau.A(x)$, and existential quantification $\exists x{:}\tau.A(x)$.

$$\begin{array}{llll}
\text{Terms} & t & ::= & x \mid f\ (t_1, \ldots, t_n) \\
\text{Type} & \tau & & \\
\text{Propositions} & A, B, C & ::= & \ldots \mid P(t) \mid \forall x{:}\tau.A(x) \mid \exists x{:}\tau.A(x)
\end{array}$$

We can read $\forall x{:}\tau.A(x)$ as "for all elements, the proposition $A(x)$ holds". We hence quantify over terms of type τ. $P(t)$ describes some basic predicate which depends on terms. We keep the grammar of terms abstract and simply state that terms are formed with variables and via some predefined function symbols f. First-order logic abstracts over the concrete data we are reasoning about, but it may nevertheless be useful to see specific instances where we choose τ to be nat. In this instance, our terms contain variables, 0 (nullary function symbol or constant), and $\mathsf{suc}\ t$ where suc is a unary function symbol. We can then state some simple facts about even and odd numbers using two predicates even and odd.

$$\begin{array}{l}
\forall x{:}\mathsf{nat}.\mathsf{even}\ x \supset \mathsf{odd}\ (\mathsf{suc}\ x) \\
\mathsf{even}\ 0 \\
\forall x{:}\mathsf{nat}.\mathsf{even}\ x \supset \mathsf{even}\ (\mathsf{suc}\ \mathsf{suc}\ x)
\end{array}$$

The meaning of logical quantifiers is however independent of the concrete domain τ we are reasoning about. We will come back and introduce concrete domains when we extend our logic with induction.

For now, we may ask: what does $\forall x{:}\tau.A(x)$ true mean? Intuitively, we must require that $A(x)$ be valid for arbitrary x, since we do not choose a specific domain τ. We note that we now introduce an assumption about the new parameter x. Hence, we have two kinds of assumptions in proofs now: *hypothetical assumptions* of the form

A true as for example introduced by the rules $\supset I$ or $\vee E$ and *parametric assumptions* of the form $x{:}\tau$.

$$\dfrac{\begin{array}{c}\overline{a{:}\tau}\\ \vdots\\ A(a)\ \mathsf{true}\end{array}}{\forall x{:}\tau.A(x)\ \mathsf{true}}\ \forall I^{a} \qquad\qquad \dfrac{\forall x{:}\tau.A(x)\ \mathsf{true} \qquad t{:}\tau}{A(t)\ \mathsf{true}}\ \forall E$$

If our domain τ is finite, we might hope to check for each element t_i in τ that $A(t_i)$ true. However, our domain may be infinite which makes this approach infeasible. Instead, we make no commitment as to what shape or property the element might have and pick one representative a. Note that a is arbitrary and fresh, i.e. it cannot have been used before in the same derivation. If we are able to establish $A(a)$ true then it is indeed the case that $\forall x{:}\tau.A(x)$ true, because we have proven A generically for an arbitrary a.

If we have a proof of $\forall x{:}\tau.A(x)$ true, then we know that $A(x)$ is true for arbitrary x. Hence, we should be able to obtain specific instances by instantiating x with a concrete term of type τ. In the rule $\forall E$, we explicitly establish the fact that t has type τ using the judgment $t{:}\tau$. We keep the definition of $t{:}\tau$ abstract at this point, but keep in mind for every concrete domain τ we can define terms belonging to it. For example, for the domain nat, we might define

$$\dfrac{}{0 : \mathsf{nat}}\ N_0 \qquad\qquad \dfrac{t : \mathsf{nat}}{\mathsf{suc}\ t : \mathsf{nat}}\ N_{suc}$$

We return to other domains shortly.

We can now prove some simple statements which are true for any domain τ such as the following:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\overline{\forall x{:}\tau.P(x)\wedge Q(x)\ \mathsf{true}}^{\,u} \qquad \overline{a:\tau}}{P(a)\wedge Q(a)\ \mathsf{true}}\ \forall E}{P(a)\ \mathsf{true}}\ \wedge E_l}{\forall x{:}\tau.P(x)\ \mathsf{true}}\ \forall I^{a} \qquad \dfrac{\dfrac{\dfrac{\overline{\forall x{:}\tau.P(x)\wedge Q(x)\ \mathsf{true}}^{\,u} \qquad \overline{b:\tau}}{P(b)\wedge Q(b)\ \mathsf{true}}\ \forall E}{Q(b)\ \mathsf{true}}\ \wedge E_r}{\forall x{:}\tau.Q(x)\ \mathsf{true}}\ \forall I^{b}}{(\forall x{:}\tau.P(x))\wedge(\forall x{:}\tau.Q(x))\ \mathsf{true}}\ \wedge I}{(\forall x{:}\tau.P(x)\wedge Q(x))\supset(\forall x{:}\tau.P(x))\wedge(\forall x{:}\tau.Q(x))\ \mathsf{true}}\ \supset I^{u}$$

We note that the parameter a in the left branch is different from the parameter b in the right branch; we chose different names for clarity, however note since their scope is different, choosing the same name in both branches would still be fine and they would still refer be distinct.

To check that our introduction and elimination rules are harmonic, we give local soundness and completeness proofs.

$$\dfrac{\dfrac{\begin{array}{c}\overline{x{:}\tau}\\ \mathcal{D}\\ A(x)\ \mathsf{true}\end{array}}{\forall x{:}\tau.A(x)\ \mathsf{true}}\,\forall I^{u} \qquad \begin{array}{c}\mathcal{E}\\ t{:}\tau\end{array}}{A(t)\ \mathsf{true}}\,\forall E \qquad\Longrightarrow\qquad \begin{array}{c}\dfrac{\mathcal{E}}{t:\tau}\\ [t/x]\mathcal{D}\\ A(t)\ \mathsf{true}\end{array}$$

Since the derivation $\mathcal{D}$ is parametric in x, we can simply replace all instances of x with concrete terms t of the same type.

We now check whether our elimination rules are strong enough to get all the information out they contain, i.e. can we reconstitute $\forall x{:}\tau.A(x)\ \mathsf{true}$ given a proof for it?

$$\begin{array}{c}\mathcal{D}\\ \forall x{:}\tau.A\ \mathsf{true}\end{array} \qquad\Longrightarrow\qquad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \forall x{:}\tau.A(x)\ \mathsf{true}\end{array} \qquad \overline{x:\tau}}{A(x)\ \mathsf{true}}\,\forall E}{\forall x{:}\tau.A(x)\ \mathsf{true}}\,\forall I^{x}$$

Let us now define the meaning of existential quantification. (Finite) universal quantification corresponds to conjunction; dually, (finite) existential quantification corresponds to disjunction. To prove $\exists x{:}\tau.A(x)$, we pick a term t from τ and show $A(t)\ \mathsf{true}$. Note that we require that t actually exists. This is an important distinction in reasoning constructively. It means we require that the type τ is in fact inhabited, i.e. elements exist and it is not empty. Classically, we are not required to provide an element explicitly. As a consequence, one might argue that constructive reasoning allows us to make more fine-grained distinction between when a domain is empty and when it is not. In fact, constructively, we can interpret the empty type as false which is often exploited in practice.

Our existential introduction rule is similar to disjunction in that we need to pick a t belonging to τ. It involves a choice.

$$\frac{A(t)\ \mathsf{true} \qquad t : \tau}{\exists x{:}\tau.A(x)\ \mathsf{true}}\exists I \qquad \frac{\exists x{:}\tau.A(x) \qquad \begin{array}{c}\overline{a{:}\tau} \quad \overline{A(a)\ \mathsf{true}}^{\,u} \\ \vdots \\ C\ \mathsf{true}\end{array}}{C\ \mathsf{true}}\exists E^{a,u}$$

What can we deduce given a proof for $\exists x{:}\tau.A(x)$? - Although we know that there exists some element a in τ such that $A(a)$ true, we don't know which one. Recall again the elimination rule for disjunction where we had a similar dilemma. Although we have $A \vee B$ true, we do not know whether A true or B true. We therefore split the proof into two cases: Assuming A true we can prove C true and assuming B true we can prove C true. We will define existential elimination similarly; if the domain τ were finite, we would have n cases to consider: assuming $A(t_i)$ true we prove C true for all $1 \leq i \leq n$. However, we do not make any assumptions about our domain. Hence, we hence pick a fresh arbitrary parameter a and assuming $A(a)$ true we establish C true. Since a was arbitrary and we have a proof for $\exists x{:}\tau.A(x)$, we have established C true.

Let us consider an example to see how we prove statements involving existential quantification.

$$(\exists x{:}\tau.P(x) \vee Q(x)) \supset (\exists x{:}\tau.P(x)) \vee (\exists x{:}\tau.Q(x))\ \mathsf{true}$$

We show the proof in two stages, since its proof tree is quite large.

$$\frac{\dfrac{\overline{\exists x{:}\tau.P(x) \vee Q(x)\ \mathsf{true}}^{\,u} \qquad \begin{array}{c}\overline{P(a) \vee Q(a)\ \mathsf{true}}^{\,v} \quad \overline{a{:}\tau} \\ \mathcal{D}^{avu} \\ (\exists x{:}\tau.P(x)) \vee (\exists x{:}\tau.Q(x))\ \mathsf{true}\end{array}}{(\exists x{:}\tau.P(x)) \vee (\exists x{:}\tau.Q(x))\ \mathsf{true}}\exists E^{av}}{(\exists x{:}\tau.P(x) \vee Q(x)) \supset (\exists x{:}\tau.P(x)) \vee (\exists x{:}\tau.Q(x))\ \mathsf{true}} \supset I^{u}$$

We now give the derivation $\mathcal{D}^{auv}$; we write the assumptions available as a superscript.

$$\dfrac{\dfrac{}{P(a) \lor Q(a)\ \mathsf{true}}\,v \qquad \dfrac{\dfrac{\dfrac{}{P(a)\ \mathsf{true}}\,w_1 \qquad \dfrac{}{a{:}\tau}}{\exists x{:}\tau.P(x)\ \mathsf{true}}\,\exists I}{(\exists x{:}\tau.P(x)) \lor (\exists x{:}\tau.Q(x))\ \mathsf{true}}\,\lor I_r \qquad \ldots}{(\exists x{:}\tau.P(x)) \lor (\exists x{:}\tau.Q(x))\ \mathsf{true}}\,\lor E^{w_1 w_2}$$

To understand better the interaction between universal and existential quantification, let's consider the following statement.

$$\exists x{:}\tau.\neg A(x) \supset \neg\forall x{:}\tau.A(x)\ \mathsf{true}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\exists x{:}\tau.\neg A(x)\ \mathsf{true}}\,u \qquad \dfrac{\dfrac{\dfrac{}{\forall x{:}\tau.A(x)\ \mathsf{true}}\,u \qquad \dfrac{}{a{:}\tau}}{A(a)\ \mathsf{true}}\,\forall E \qquad \dfrac{}{\neg A(a)\ \mathsf{true}}\,v}{\bot\ \mathsf{true}}\supset E}{\bot\ \mathsf{true}}\,\exists E^{av}}{\neg\forall x{:}\tau.A(x)\ \mathsf{true}}\supset I^{w}}{\exists x{:}\tau.\neg A(x) \supset \neg\forall x{:}\tau.A(x)\ \mathsf{true}}\supset I^{u}$$

Let's consider the converse;

$$(\neg\forall x{:}\tau.A(x)) \supset \exists x{:}\tau.\neg A(x)\ \mathsf{true}$$

After using implication introduction, we are in the following situation:

$$\dfrac{\begin{array}{c}\dfrac{}{\neg\forall x{:}\tau.A(x)\ \mathsf{true}}\,u \\ \vdots \\ \exists x{:}\tau.\neg A(x)\ \mathsf{true}\end{array}}{(\neg\forall x{:}\tau.A(x)) \supset \exists x{:}\tau.\neg A(x)\ \mathsf{true}}\supset I^{u}$$

But now we are stuck; to use the rule $\exists I$ we need a concrete witness for x, which we do not have, since we know nothing about the domain τ. We also cannot do anything with the assumption $\neg\forall x{:}\tau.A(x)$ which is equivalent to $\forall x{:}\tau.A(x) \supset \bot$. To eliminate it, we would need a proof of $\forall x{:}\tau.A(x)$.

$$\cfrac{\cfrac{\cfrac{\cfrac{}{\neg\forall x{:}\tau.A(x)\ \mathsf{true}}\,u \qquad \cfrac{\begin{array}{c}\cfrac{}{\neg\forall x{:}\tau.A(x)\ \mathsf{true}}\,u\quad \cfrac{}{c{:}\tau}\\ \vdots \\ A(c)\ \mathsf{true}\end{array}}{\forall x{:}\tau.A(x)\ \mathsf{true}}\,\forall I^c}{\bot\ \mathsf{true}}\,\supset E}{\exists x{:}\tau.\neg A(x)\ \mathsf{true}}\,\bot E}{(\neg\forall x{:}\tau.A(x))\supset\exists x{:}\tau.\neg A(x)\ \mathsf{true}}\,\supset I^u$$

But how should we obtain a proof for $A(c)$ given $\neg\forall x{:}\tau.A(x)$ true. There is not much we can do; we can attempt again to derive a contradiction using $\supset E$, but this simply leads to a loop. At the moment we do not have the syntactic tools to argue why this statement is not provable, so this argument may seem unsatisfying. We will get back to more syntactic methods of arguing why something is not provable later. It turns out that if a proof exists, it must exist without any detours (i.e. without any combinations of intro-elim rules) and moreover every proof in first-order logic must satisfy the subformula property, i.e. we can concentrate on using only introduction and elimination rules for connectives which occur in the formula we are trying to prove.

An alternative is to give a counter example by choosing a specific domain and specific predicate instantiation for A.

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

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

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

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

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

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

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

$$\frac{\Gamma \vdash A(t) \text{ true} \quad \Gamma \vdash t{:}\tau}{\Gamma \vdash \exists x{:}\tau.A(x) \text{ true}} \exists I \qquad \frac{\Gamma \vdash \exists x{:}\tau.A(x) \text{ true} \quad \Gamma, a{:}\tau, u{:}A(a) \text{ true} \vdash C \text{ true}}{\Gamma \vdash C \text{ true}} \exists E^{au}$$

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.

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

We hence generalize our judgment A true to $\Gamma \vdash A$ 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 $\mathcal{D} = \dfrac{v : \mathsf{C\ true} \in (\Gamma, \Gamma')}{\Gamma, u : \mathsf{A\ true}, \Gamma' \vdash \mathsf{C\ true}}\, v$

$\Gamma, \Gamma' \vdash \mathsf{C\ true}$ 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 $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{F}\\ \Gamma, u : \mathsf{A\ true}, \Gamma' \vdash \mathsf{C\ true}\end{array} \qquad \begin{array}{c}\mathcal{E}\\ \Gamma, u : \mathsf{A\ true}, \Gamma' \vdash \mathsf{B\ true}\end{array}}{\Gamma, u : \mathsf{A\ true}, \Gamma' \vdash \mathsf{C} \wedge \mathsf{B\ true}}\, \wedge I$

$\Gamma \vdash A\ \mathsf{true}$ by assumption
$\Gamma, \Gamma' \vdash \mathsf{C\ true}$ by i.h. using $\mathcal{F}$ and assumption
$\Gamma, \Gamma' \vdash \mathsf{B\ true}$ by i.h. using $\mathcal{E}$ and assumption
$\Gamma, \Gamma' \vdash \mathsf{C} \wedge \mathsf{B\ true}$ 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 $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{E}\\ \Gamma, u : \mathsf{A\ true}, \Gamma', v{:}\mathsf{B\ true} \vdash \mathsf{C\ true}\end{array}}{\Gamma, u : \mathsf{A\ true}, \Gamma' \vdash \mathsf{B} \supset \mathsf{C\ true}} \supset I^v$

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

2.8 Exercises

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

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

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 true. This is stronger than simply whenever $M : A$ then A true and vice versa. 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.

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:

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

and dually

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

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 $()$.

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

$\top$ 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 $(A \wedge A) \supset A$ true.

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

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 $M\ N$.

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

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 $(A \wedge B) \supset (B \wedge A)$ whose proof we've seen earlier as

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

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.

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

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.

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

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
$A \supset B$	$A \rightarrow B$	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.

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

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

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

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

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

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

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

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

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

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

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

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

□

3.4.2 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

Chapter 4

Induction

So far, we have seen first-order logic together with its corresponding proof-terms. First-order logic corresponds to the dependently typed lambda-calculus. However, if we are to write meaningful programs we need two more ingredients: 1) we need to reason about specific domains such as natural numbers, lists, etc 2) we need to be able to write recursive programs about elements in a given domain. Proof-theoretically, we would like to add the power of induction which as it turns out corresponds to being able to write total well-founded recursive programs.

4.1 Domain: natural numbers

First-order logic is independent of a given domain and the reasoning principles we defined hold for any domain. Nevertheless, it is useful to consider specific domains. There are several approaches to incorporating domain types or index types into our language: One is to add a general definition mechanism for *recursive types* or *inductive types*. We do not consider this option here, but we return to this idea later. Another one is to use the constructs we already have to define data. This was Church's original approach; he encoded numerals, booleans as well as operations such as addition, multiplication, if-statements, etc. as lambda-terms using a *Church encoding*. We will not discuss this idea in these notes. A third approach is to specify each type directly by giving rules defining how to construct elements of a given type (introduction rule) and how to reason with elements of a given type (elimination rule). This is the approach we will be pursuing here.

We begin by defining concretely the judgement

$$t : \tau \quad \text{Term } t \text{ has type } \tau$$

which we have left more or less abstract for now for concrete instances of τ.

4.1.1 Defining for natural numbers

We define elements belonging to natural numbers via two constructors z and suc. They allow us to introduce natural numbers. We can view these two rules as introduction rules for natural numbers.

$$\frac{}{\mathsf{z} : \mathsf{nat}}\ \mathsf{nat}I_{\mathsf{z}} \qquad \frac{t : \mathsf{nat}}{\mathsf{suc}\, t : \mathsf{nat}}\ \mathsf{nat}I_{\mathsf{suc}}$$

4.1.2 Reasoning about natural numbers

To prove inductively a property $A(t)$ true, we establish three things:

1. t is a natural number and hence we know how to split it into different cases.
2. *Base case*: $A(\mathsf{z})$ true
   Establish the given property for the number z
3. *Step case*: For any $n : \mathsf{nat}$, assume $A(n)$ true (I.H) and prove $A(\mathsf{suc}\, n)$ true.
   We assume the property holds for smaller numbers, i.e. for n, and we establish the property for $\mathsf{suc}\, n$.

More formally, the inference rule capturing this idea is given below:

$$\frac{t : \mathsf{nat} \qquad A(\mathsf{z})\ \mathsf{true} \qquad \begin{array}{c} \overline{n : \mathsf{nat}} \quad \overline{A(n)\ \mathsf{true}}\ i.h \\ \vdots \\ A(\mathsf{suc}\, n)\ \mathsf{true} \end{array}}{A(t)\ \mathsf{true}}\ \mathsf{nat}E^{n,ih}$$

Restating the rule using explicit contexts to localize all assumptions:

$$\frac{\Gamma \vdash t : \mathsf{nat} \qquad \Gamma \vdash A(\mathsf{z})\ \mathsf{true} \qquad \Gamma,\ n{:}\mathsf{nat},\ ih{:}A(n)\ \mathsf{true} \vdash A(\mathsf{suc}\, n)\ \mathsf{true}}{\Gamma \vdash A(t)\ \mathsf{true}}\ \mathsf{nat}E^{n,ih}$$

Let us prove a simple property inductively, to see how we use the rule.

Note the rule $\mathsf{nat}E^{n,ih}$ has implicitly a generalization built-in. If we want to establish a property $A(42)$ true, we may choose to prove it more generally for any number proving that $A(z)$ true (i.e. the property A holds for z) and proving $A(\mathsf{suc}\, n)$ true (i.e. the property A holds for $\mathsf{suc}\, n$) assuming the property A holds for n. Since we now have proven that A holds for all natural numbers, it also must hold for 42.

Let's look at an example; we first encode definitions which will form our signature $\mathcal{S}$.

$$\begin{array}{ll}\mathsf{le\_z} & : \forall x{:}\mathsf{nat}.\ \ z \le x \\ \mathsf{le\_suc} & : \forall x{:}\mathsf{nat}.\forall y{:}\mathsf{nat}.\ \ x \le y \ \ \supset \ \ \mathsf{suc}\, x \le \mathsf{suc}\, y\end{array}$$

Then we would like to prove: $\mathcal{S} \vdash \forall x{:}\mathsf{nat}.\ \ x \le x\ \mathsf{true}$. For better readability, we omit true in the derivations below.

$$\dfrac{\dfrac{\dfrac{}{\mathcal{S},\ a{:}\mathsf{nat} \vdash a : \mathsf{nat}} \qquad \begin{array}{c}\mathcal{D}_z \\ \mathcal{S},\ a{:}\mathsf{nat} \vdash z \le z\end{array} \qquad \begin{array}{c}\mathcal{D}_{\mathsf{suc}} \\ \mathcal{S}, a{:}\mathsf{nat},\ n{:}\mathsf{nat},\ ih{:}n \le n \vdash \mathsf{suc}\, n \le \mathsf{suc}\, n\end{array}}{\mathcal{S},\ a{:}\mathsf{nat} \vdash a \le a}\,\mathsf{natE}^{n,ih}}{\mathcal{S} \vdash \forall x{:}\mathsf{nat}.\ \ x \le x}\,\forall I^{a}$$

We consider the base case $\mathcal{D}_z$ and the step case $\mathcal{D}_{\mathsf{suc}}$ separately. We also write the derivations using implicit context representations to keep them compact.

$$\mathcal{D}_z = \dfrac{\dfrac{}{\forall x{:}\mathsf{nat}.z \le x}\,\mathsf{le\_z} \qquad \dfrac{}{z : \mathsf{nat}}\,\mathsf{nat}_z}{z \le z}\,\forall E$$

$$\mathcal{D}_{\mathsf{suc}} = \dfrac{\dfrac{\dfrac{\dfrac{}{\forall x{:}\mathsf{nat}.\forall y{:}\mathsf{nat}.\ \ x \le y \ \supset\ \mathsf{suc}\, x \le \mathsf{suc}\, y}\,\mathsf{le\_suc} \qquad \dfrac{}{n : \mathsf{nat}}}{\forall y{:}\mathsf{nat}.\ \ n \le y \ \supset\ \ \mathsf{suc}\, n \le \mathsf{suc}\, y}\,\forall E \qquad \dfrac{}{n : \mathsf{nat}}}{n \le n \supset \mathsf{suc}\, n \le \mathsf{suc}\, n}\,\forall E \qquad \dfrac{}{n \le n}\,ih}{\mathsf{suc}\, n \le \mathsf{suc}\, n}\,\supset E$$

4.1.3 Proof terms

We assign a recursive program to the induction rule.

$$\dfrac{\Gamma \vdash t : \mathsf{nat} \qquad \Gamma \vdash M_z : A(z) \qquad \Gamma,\ n{:}\mathsf{nat},\ f\ n{:}A(n)\ \mathsf{true} \vdash M_{\mathsf{suc}} : A(\mathsf{suc}\, n)}{\Gamma \vdash \mathsf{rec}^{\forall x:\mathsf{nat}.A(x)}\ t\ \mathsf{with}\ f\ z \to M_z \mid f\ (\mathsf{suc}\, n) \to M_{\mathsf{suc}} : A(t)}\,\mathsf{natE}^{n, f\, n}$$

The proof term uses the variable f to denote the function we are defining; in some sense, this definition is similar to programs allowing defining functions by equations using simultaneous pattern as in Haskell.

From the proof above for $\mathcal{S} \vdash \forall x{:}\mathsf{nat}.\ x \leq x$ we can synthesize the following program:

$$\begin{array}{l} \lambda a{:}\mathsf{nat}.\ \mathsf{rec}^{\forall x:\mathsf{nat}.\ x \leq x}\ a\ \mathsf{with} \\ \qquad \begin{array}{lll} \mid \mathsf{f}\,\mathsf{z} & \Rightarrow & \mathsf{le\_z}\ \mathsf{z} \\ \mid \mathsf{f}\,(\mathsf{suc}\,n) & \Rightarrow & \mathsf{le\_suc}\ n\,n\,(\mathsf{f}\,n) \end{array} \end{array}$$

How to extend our notion of computation? We will have two reduction rules which allow us to reduce a recursive program:

$$\begin{array}{llcl} \mathsf{rec}^A\ \mathsf{z} & \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to M_\mathsf{z} \mid \mathsf{f}\ (\mathsf{suc}\,n) \to M_\mathsf{suc} & \Longrightarrow & M_\mathsf{z} \\ \mathsf{rec}^A\ (\mathsf{suc}\,t) & \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to M_\mathsf{z} \mid \mathsf{f}\ (\mathsf{suc}\,n) \to M_\mathsf{suc} & \Longrightarrow & [t/n][r/\mathsf{f}\,n]M_\mathsf{suc} \\ & \multicolumn{3}{r}{\text{where } r = \mathsf{rec}^A\ t\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to M_\mathsf{z} \mid \mathsf{f}\ (\mathsf{suc}\,n) \to M_\mathsf{suc}} \end{array}$$

Note that we unroll the recursion by replacing the reference to the recursive call $\mathsf{f}\,n$ with the actual recursive program $\mathsf{rec}^A\ t\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to M_\mathsf{z} \mid \mathsf{f}\ (\mathsf{suc}\,n) \to M_\mathsf{suc}$.

We might ask, how to extend our congruence rules. In our setting, were reductions can happen at any given sub-term, we will have two additional congruence rules. Note that we do not have a rule which evaluates t, the term we are recursing over; at the moment, the only possible terms we can have are those formed by z and suc or variables. We have no computational power on for terms t of our domain.

Congruence rules

$$\frac{N_\mathsf{z} \Longrightarrow N_\mathsf{z}'}{\mathsf{rec}^A\ t\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to N_\mathsf{z} \mid \mathsf{f}\ (\mathsf{suc}\,n) \to N_\mathsf{suc} \Longrightarrow \mathsf{rec}^A\ t\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to N_\mathsf{z}' \mid \mathsf{f}\ (\mathsf{suc}\,n) \to N_\mathsf{suc}}$$

$$\frac{N_\mathsf{suc} \Longrightarrow N_\mathsf{suc}'}{\mathsf{rec}^A\ t\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to N_\mathsf{z} \mid \mathsf{f}\ (\mathsf{suc}\,n) \to N_\mathsf{suc} \Longrightarrow \mathsf{rec}^A\ t\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to N_\mathsf{z} \mid \mathsf{f}\ (\mathsf{suc}\,n) \to N_\mathsf{suc}'}$$

Proving subject reduction We also revisit subject reduction, showing that the additional rules for recursion preserve types. This is a good check that we didn't screw up.

Theorem 4.1.1. *If* $M \Longrightarrow M'$ *and* $\Gamma \vdash M : C$ *then* $\Gamma \vdash M' : C$.

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

Case $\mathcal{D} = \mathsf{rec}^A\ (\mathsf{suc}\,t)\ \mathsf{with}\ f\ z \to M_z \mid f\ (\mathsf{suc}\,n) \to M_{suc} \implies [t/n][r/f\,n]M_{suc}$
where $r = \mathsf{rec}^A\ t\ \mathsf{with}\ f\ z \to M_z \mid f\ (\mathsf{suc}\,n) \to M_{suc}$

$\Gamma \vdash \mathsf{rec}^A\ (\mathsf{suc}\,t)\mathsf{with}\ f\ z \to M_z \mid f\ (\mathsf{suc}\,n) \to M_{suc} : C$	by assumption
$\Gamma \vdash \mathsf{suc}\,t : \mathsf{nat}$	
$\Gamma \vdash M_z : A(z)$	
$\Gamma, n{:}\mathsf{nat}, f\,n : A(n) \vdash M_{suc} : A(\mathsf{suc}\,n)$ where $C = A(\mathsf{suc}\,t)$	by inversion on natE
$\Gamma \vdash t : \mathsf{nat}$	by natI_{suc}
$\Gamma \vdash \mathsf{rec}^A\ t\mathsf{with}\ f\ z \to M_z \mid f\ (\mathsf{suc}\,n) \to M_{suc} : A(t)$	by rule natE
$[t/n][r/f\,n]M_{suc} : A(\mathsf{suc}\,t)$	by substitution lemma (twice)

Case $\mathcal{D} = \dfrac{\overset{\mathcal{D}'}{N_z \implies N_z'}}{\mathsf{rec}^A\ t\ \mathsf{with}\ f\ z \to N_z \mid f\ (\mathsf{suc}\,n) \to N_{suc} \implies \mathsf{rec}^A\ t\ \mathsf{with}\ f\ z \to N_z' \mid f\ (\mathsf{suc}\,n) \to N_{suc}}$

$\Gamma \vdash \mathsf{rec}^A\ t\ \mathsf{with}\ f\ z \to N_z \mid f\ (\mathsf{suc}\,n) \to N_{suc} : C$	by assumption
$\Gamma \vdash t : \mathsf{nat}$	
$\Gamma, n{:}\mathsf{nat}, f\,n : A(n) \vdash N_{suc} : A(\mathsf{suc}\,n)$	
$\Gamma \vdash N_z : A(z)$ where $C = A(t)$	by inversion on natE
$\Gamma \vdash N_z' : A(z)$	by i.h.
$\Gamma \vdash \mathsf{rec}^A\ t\ \mathsf{with}\ f\ z \to N_z' \mid f\ (\mathsf{suc}\,n) \to N_{suc} : C$	by rule natE $\square$

Erasing terms We will next prove next $\forall x{:}\mathsf{nat}.\neg(x = z) \supset \exists y : \mathsf{nat}.\mathsf{suc}\,y = x$. For simplicity, we define equality using reflexivity: $\mathsf{ref} : \forall x{:}\mathsf{nat}.x = x$.

$$\dfrac{\overset{\mathcal{D}}{\mathcal{S}, a : \mathsf{nat} \vdash \neg(a = z) \supset \exists y : \mathsf{nat}.\mathsf{suc}\,y = a}}{\mathcal{S} \vdash \forall x{:}\mathsf{nat}.\neg(x = z) \supset \exists y : \mathsf{nat}.\mathsf{suc}\,y = x}\,\forall I^a$$

To prove $\mathcal{D}$ we will use the rule $\mathsf{natE}^{n,f\,n}$; in particular, we need to prove the following base and step case:
Base case:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg(z = z)}\,u \qquad \dfrac{}{z = z}\,\mathsf{ref}}{\bot}\supset E}{\exists y{:}\mathsf{nat}.\ \mathsf{suc}\,y = z}\,\bot E}{\neg(z = z) \supset \exists y{:}\mathsf{nat}.\ \mathsf{suc}\,y = z}\supset I^u$$

Step case:

$$\dfrac{\dfrac{\dfrac{}{\mathsf{suc}\,(n) = \mathsf{suc}\,(n)}\,\mathsf{ref} \qquad \dfrac{}{n : \mathsf{nat}}}{\exists y{:}\mathsf{nat}.\ \mathsf{suc}\,(y) = \mathsf{suc}\,(n)}\,\exists I}{\neg(\mathsf{suc}\,(n) = \mathsf{z}) \supset \exists y{:}\mathsf{nat}.\ \mathsf{suc}\,(y) = \mathsf{suc}\,(n)}\,\supset I^u$$

Therefore, we have established $\forall x{:}\mathsf{nat}.\neg(x = \mathsf{z}) \supset \exists y : \mathsf{nat}.\mathsf{suc}\,y = x$. Note that we only used the induction rule to split the proof into two different cases, but we did not actually use the induction hypothesis in the proof.

It is particularly interesting to extract the corresponding proof term; we omit the type annotation on the rec for better readability.

$$\begin{array}{l} \lambda a{:}\mathsf{nat}.\ \mathsf{rec}\ a\ \mathsf{with} \\ \qquad \begin{array}{lcl} \mid \mathsf{f}\,\mathsf{z} & \Rightarrow & \lambda u{:}\neg(x = \mathsf{z}).\mathsf{abort}\ (u\ (\mathsf{refl}\ \mathsf{z})) \\ \mid \mathsf{f}\,(\mathsf{suc}\,n) & \Rightarrow & \lambda u{:}\neg(x = \mathsf{z}).\langle n,\ \mathsf{refl}\ \mathsf{z}\rangle \end{array} \end{array}$$

Can you guess what program it implements? - If we erase all subterms pertaining propositions, it becomes even more obvious:

$$\begin{array}{l} \lambda a{:}\mathsf{nat}.\ \mathsf{rec}\ a\ \mathsf{with} \\ \qquad \begin{array}{lcl} \mid \mathsf{f}\,\mathsf{z} & \Rightarrow & \_\_ \\ \mid \mathsf{f}\,(\mathsf{suc}\,n) & \Rightarrow & n \end{array} \end{array}$$

We have obtained a verified predecessor function!

4.2 Domain: Lists

Similar to natural numbers, we can define lists.

We can define lists by two constructors nil and cons. We concentrate on defining lists of natural numbers here to avoid problems with polymorphism.

$$\dfrac{}{\mathsf{nil} : \mathsf{list}} \qquad \dfrac{h : \mathsf{nat} \qquad l : \mathsf{list}}{\mathsf{cons}\,h\ l : \mathsf{nat}}$$

To prove inductively a property $A(t)$ true, we establish three things:

1. t is a natural number and hence we know how to split it into different cases.

2. *Base case*: $A(\mathsf{nil})$ true
   Establish the given property for the number z

3. *Step case*: For any h : nat, t : list, assume $A(t)$ true (I.H) and prove $A(\text{cons}\, h\, t)$ true. We assume the property holds for smaller lists, i.e. for t, and we establish the property for cons h t.

More formally the induction rule then takes on the following form:

$$\frac{\Gamma \vdash s : \text{list} \qquad \Gamma \vdash A(\text{nil}) \qquad \Gamma, n : \text{nat}, t : \text{list}, ih : A(t) \vdash A(\text{cons}\; n\; t)}{\Gamma \vdash A(s)}\ \text{listE}^{n,t,ih}$$

The corresponding annotated rule takes on the form below giving us the ability to write recursive functions about lists.

$$\frac{\Gamma \vdash s : \text{list} \qquad \Gamma \vdash M_{\text{nil}} : A(\text{nil}) \qquad \Gamma,\ h{:}\text{nat},\ t{:}\text{list},\ f\ t{:}A(t)\ \text{true} \vdash M_{\text{cons}} : A(\text{cons}\ h\ t)}{\Gamma \vdash \text{rec}^{\forall x:\text{list}.A(x)}\ s\ \text{with}\ f\ \text{nil} \rightarrow M_{\text{nil}} \mid f\ (\text{cons}\ h\ t) \rightarrow M_{\text{cons}} : A(s)}\ \text{listE}^{n,\, f\, n}$$

$$\begin{array}{lll} \text{rec}^{A}\ \text{nil} & \text{with}\ f\ \text{nil} \rightarrow M_{\text{nil}} \mid f\ (\text{cons}\ h\ t) \rightarrow M_{\text{cons}} & \Longrightarrow \quad M_{\text{nil}} \\ \text{rec}^{A}\ (\text{cons}\ h'\ t') & \text{with}\ f\ \text{nil} \rightarrow M_{\text{nil}} \mid f\ (\text{cons}\ h\ t) \rightarrow M_{\text{cons}} & \Longrightarrow \quad [h'/h][t'/t][r/f\ t]M_{\text{cons}} \\ & \multicolumn{2}{r}{\text{where}\ r = \text{rec}^{A}\ t'\ \text{with}\ f\ \text{nil} \rightarrow M_{\text{nil}} \mid f\ (\text{cons}\ h\ t) \rightarrow M_{\text{cons}}} \end{array}$$

Let's practice writing recursive functions. How would we write the function which given a list reverses it. Its type is: $T = \text{list} \supset \text{list}$. We will write it in a tail-recursive manner and write first a helper function of type $S = \text{list} \supset \text{list} \supset \text{list}$.

$$\Lambda l : \text{list}.\text{rec}^{S}\ l\ \text{with}\ f\ \text{nil} \rightarrow \Lambda r : \text{list}.r \mid f\ (\text{cons}\ h\ t) \rightarrow \Lambda r : \text{list}.f\ t\ (\text{cons}\ h\ r)$$

4.3 Extending the induction principle to reasoning about indexed lists and other predicates

Often when we want to program with lists, we want more guarantees than simply that given a list, we return a list. In the previous example, we might want to say that given a list `l1` of length `n1` and an accumulator `l2` of length `n2`, we return a list `l3` of `n3` where `n3 = n1 + n2`.

First, how can we define lists which keep track of their length?

$$\frac{}{\text{nil} : \text{list}\ z} \qquad \frac{h : \text{nat} \qquad l : \text{list}\ n}{\text{cons}\ \{n\}\ h\ l : \text{list}\ \text{suc}\ n}$$

We mark the argument denoting the length in cons with curly braces. This highlights the difference between the definition for lists which do not carry the length information and our definition here which is aware of its length. In practice, dependently typed programming languages such as Agda, Coq, Beluga, etc. will reconstruct the arguments marked in curly braces.

We can encode the list containing zeroes and ones as follows where we write implicit parameters in curly braces simply to emphasize this information is there, but you typically can omit the information in curly braces.

```
[. cons {2} z (cons {1} (suc z) (cons {0} z nil )) ]
```

So far we have seen how to reason inductively about natural number and terms. But how can we support structural induction on lists which are indexed by their length, i.e. lists `l` of type list n? - We will refine our induction principle as follows: First, we observe that the property we are trying to prove depends not only on `l` but also on `n`.

To prove inductively a property $A(n, l)$ about a list l of length n, we establish three things:

1. n is a natural number and l is a list of length n; and hence we know how to split l into different cases.

2. *Base case*: $A(\mathsf{z},\ \mathsf{nil})$ true
   Establish the given property for a list nil of length z.

3. *Step case*: For any m:nat, h:nat, t:list m, assume $A(m, t)$ true (I.H) and prove $A(m,\ \mathsf{cons}\{m\}\ h\ t)$ true.

   We assume the property holds for smaller lists, i.e. a list t of length m, and we establish the property for $(\mathsf{cons}\ \{m\}\ h\ t)$, a list of length $(\mathsf{suc}\ m)$.

Our annotated induction rule, then takes on the following form:

$$\frac{\begin{array}{l}(1)\Gamma \vdash s : \mathsf{list}\ n \\ (2)\Gamma \vdash M_{\mathsf{nil}} : A(\mathsf{z},\ \mathsf{nil}) \\ (3)\Gamma,\ m{:}\mathsf{nat},\ h{:}\mathsf{nat},\ t{:}\mathsf{list},\ f\ \{m\}\ t{:}A(m, t)\ \mathsf{true} \vdash M_{\mathsf{cons}} : A(s\ m,\ \mathsf{cons}\ \{m\}\ h\ t)\end{array}}{\Gamma \vdash \mathsf{rec}^{\forall n:\mathsf{nat}.\forall x:\mathsf{list}\ n.A(n,x)}\ s\ \mathsf{with}\ f\ \{z\}\ \mathsf{nil} \to M_{\mathsf{nil}} \mid f\ \{\mathsf{suc}\ m\}\ (\mathsf{cons}\ \{m\}\ h\ t) \to M_{\mathsf{cons}} : A(n, s)}\ \mathsf{listE}^{m,\ f\{m\}\ t}$$

Our recursion expression now performs a *simultaneous pattern mach* on all the arguments the property we are proving depends on.

In some sense, indexed lists are just a predicate about lists and we have just inferred a reasoning principle about predicates. We can generalize this idea of reasoning about general predicates which are inductively defined.

For example, we might want to prove $\forall n : \mathsf{nat}.\forall m : \mathsf{nat}.n \leq m \supset n \leq \mathsf{suc}\, m$. Proving this statement by induction on n is not straightforward; we need a case analysis on both m.

It would be more convenient to interpret:

$$\begin{array}{ll} \mathsf{le\_z} & : \forall x{:}\mathsf{nat}.\ \mathsf{z} \leq x \\ \mathsf{le\_suc} & : \forall x{:}\mathsf{nat}.\forall y{:}\mathsf{nat}.\ x \leq y \ \supset\ \mathsf{suc}\, x \leq \mathsf{suc}\, y \end{array}$$

as inference rules

$$\frac{}{\mathsf{z} \leq X}\,\mathsf{le\_z} \qquad \frac{X \leq Y}{\mathsf{suc}\, X \leq \mathsf{suc}\, Y}\,\mathsf{le\_suc}$$

and then argue that le_z denotes a proof of height 0 and forms our base case and le_suc gives us a step case where we can assume the property holds for $X \leq Y$ and we establish the property for $\mathsf{suc}\, X \leq \mathsf{suc}\, Y$.

To allow induction over the given definition of $\leq$, we generalize our induction rule. First, we allow our property A to take in more than one argument and write $A(X, Y, X \leq Y)$. We omit true from the rule below to simplify it.

$$\frac{\Gamma \vdash D : N \leq M \qquad \Gamma \vdash A(\mathsf{z}, Y, \mathsf{le\_z}\{Y\}) \qquad \begin{array}{l}\Gamma,\ X{:}\mathsf{nat},\ Y{:}\mathsf{nat},\ \mathsf{suc}\, Y,\ D' : X \leq Y, \\ ih{:}A(X, Y, D') \vdash A(\mathsf{suc}\, X, \mathsf{le\_suc}\ \{X\}\{Y\}\, D')\end{array}}{\Gamma \vdash A(N, M, D)}$$

This justifies writing recursive programs directly by pattern matching on the derivation tree. Formally, we write:

$$\begin{array}{ll} \mathsf{rec}\ D\ \mathsf{with}\ \mathsf{f}\ \{\mathsf{z}\}\,\{Y\}\ \mathsf{le\_z}\ \{Y\} & \rightarrow M_{\mathsf{z}} \\ \quad\quad \mid \mathsf{f}\ \{X\}\,\{Y\}\ \mathsf{le\_suc}\ \{X\}\,\{Y\}\ D' & \rightarrow M_{\mathsf{suc}} \end{array}$$

Let's look at a simple proof of transitivity.

Theorem 4.3.1. *If* $M \leq N$ *and* $N \leq K$ *then* $M \leq K$.

Proof. Induction on the derivation $\mathcal{D} : M \leq N$.

Base case $\mathcal{D} = \dfrac{}{\mathsf{z} \leq X}\,\mathsf{le\_z}$

We need to show that assuming $X \leq K$, we have a derivation for $\mathsf{z} \leq K$. This is justified by the rule le_z.

Step case $\mathcal{D} = \dfrac{X \leq Y}{\mathsf{suc}\,X \leq \mathsf{suc}\,Y}\,\mathsf{le\_suc}$

We can assume the property we want to prove holds for $X \leq Y$, i.e. if $X \leq Y$ and $Y \leq K$ then $X \leq K$.

$\mathsf{suc}\,Y \leq \mathsf{suc}\,K$	by assumption from the statement we need to prove
$Y \leq K$	by inversion using $\mathsf{le\_suc}$
$X \leq K$	by i.h.
$\mathsf{suc}\,X \leq \mathsf{suc}\,K$	by using $\mathsf{le\_suc}$

$\square$

In the proof above, we hide a few obvious logical steps such as universally quantifying over M, N, and K; introducing quantifiers, eliminating quantifiers, implication introductions and eliminations, etc. It might be useful to see how we can use the induction rule to justify the given proof;

- We first observe that we are proving $\forall M : \mathsf{nat}, \forall N : \mathsf{nat}. M \leq N \supset \forall K : \mathsf{nat}. N \leq K \supset M \leq K$. Note that we have slightly rewritten the statement.

- We are proving $\forall K : \mathsf{nat}. N \leq K \supset M \leq K$ under the assumptions $M : \mathsf{nat}, N : \mathsf{nat}, \mathcal{D} : M \leq N$;

  In other words, the property $A(M, N, D)$ we are proving by induction is $\forall K : \mathsf{nat}. N \leq K \supset M \leq K$ and our assumptions represent the context Γ in the induction rule above.

Rewriting the proof making the intermediate steps explicit, we get.

Base case We need to prove

$$A(z, X, \mathsf{le\_z}) = \forall K : \mathsf{nat}. X \leq K \supset z \leq K$$

$K : \mathsf{nat}$	by assumption
$X \leq K$	by assumption u
$\forall X{:}\mathsf{nat}.\ z \leq X$	by $\mathsf{le\_z}$
$z : \mathsf{nat}$	by rule for $\mathsf{nat}I_z$
$z \leq K$	by $\forall^E$
$X \leq K \supset z \leq K$	by $\supset I^u$
$\forall K : \mathsf{nat}. X \leq K \supset z \leq K$	by $\forall I^K$

Step Case We need to prove

$$A(\mathsf{suc}\,X, \mathsf{suc}\,Y, \mathsf{le\_suc}\ \{X\}\,\{Y\}\,\mathcal{D}_1) = \forall K : \mathsf{nat}.\mathsf{suc}\,X \leq K \supset \mathsf{suc}\,Y \leq K$$

under the assumptions:

$$X : \mathsf{nat},\ Y : \mathsf{nat},\ \mathcal{D}_1{:}X \leq Y,\ ih{:}A(X,\ Y,\ \mathcal{D}_1)$$

where $A(X,\ Y,\ \mathcal{D}_1) = \forall K : \mathsf{nat}.X \leq K \supset Y \leq K$

$K : \mathsf{nat}$	by assumption
$\mathsf{suc}\,X \leq K$	by assumption u
$X \leq K'$ and $K = \mathsf{suc}\,K'$	by inversion on $\mathsf{le\_suc}$
$\forall K : \mathsf{nat}.X \leq K \supset Y \leq K$	by i.h.
$X \leq K' \supset Y \leq K'$	by $\forall E$
$Y \leq K'$	by $\supset E$
$\mathsf{suc}\,Y \leq \mathsf{suc}\,K'$	by rule $\mathsf{le\_suc}$
$\mathsf{suc}\,Y \leq K$	since $K = \mathsf{suc}\,K'$
$\mathsf{suc}\,X \leq K \supset \mathsf{suc}\,Y \leq K$	by $\supset I^u$
$\forall K : \mathsf{nat}.\mathsf{suc}\,X \leq K \supset \mathsf{suc}\,Y \leq K$	by $\forall I^K$.

As this development shows, all steps in the proof (except the inversion step, which technically is a lemma), are justified by logical rules. Writing proofs in such excruciating detail is usually avoided - however, it highlights that in principle we can automate developing these proofs.

In dependently-typed proof environments, such a proof can be represented in a remarkably succinct way. We show here the example in Beluga syntax, but one can equally take Agda or Coq (left as an exercise). In Beluga, we can write the proof

compactly as a recursive functions of type

```
trans: [leq M N] -> [leq N K] -> [leq M K] =
```

Just as we omitted the explicit quantifiers in our theorem statement, we omit them in the type and let type reconstruction infer them.

The transitivity proof is then implemented as a recursive function on `[leq M N]`. The function deviates slightly from the formal proof. While we could have implemented the function of type `[leq M N]->{K:[nat]}[leq N K]->[leq M N}`, the function we write below is more elegant, as we are not restricted to writing primitive recursive functions.

```
rec trans: [leq M N] -> [leq N K] -> [leq M K] =
fn d => fn e => case d of
| [le_z ] => [le_z]

| [le_s D] =>
```

```
  let [le_s E] = e in
  let [F] = trans [D] [E] in
  [le_s F]
;
```

We observe that every step in the proof corresponds exactly to one step in our informal proof; Beluga let's you hide "logical" details (i.e. universal introduction and eliminations) which polluted our detailed formal proof above, but we concentrate on the essential steps in the proof. Avoiding such clutter is essential in making writing realistic proofs feasible!

To summarize:

On paper	In Beluga
Case analysis	Case analysis
Appeal to induction hypothesis	Recursive call
Inversion	Case analysis with one case usually using a let-expression

4.4 First-order Logic with Domain-specific Induction

More generally, we can extend our first-order logic with specific domains which we call $\mathcal{U}$. We embed predicates belonging to the domain $\mathcal{U}$ into first-order formulas using the necessity modality, written as $[\mathcal{U}]$. This is related to Moggi's monadic language. Here we separate domain-specific definitions/knowledge from our generic first-order logic.

To support inductive reasoning about a domain $\mathcal{U}$ we need the domain to have certain properties: for example it must be decidable when two objects of the domain $\mathcal{U}$ are equal. We also must have a way of splitting an object into different cases and have a measure according to which an object of the domain is considered smaller than another object. These ingredients are essential to generate induction principles for the given domain.

Chapter 5

Sequent Calculus

In this chapter, we prove consistency of our formal system described by natural deduction rules. We employ a classic approach already used by Gentzen in his original paper. We observe that in natural deduction there are many derivations which are not “normal” and the reasoning system admits detours. Consider the following two proofs for $A \supset B \supset A$. The proof on the left is normal while the one one the right is not; it contains a detour, i.e. $\wedge I$ followed by $\wedge E_l$, which can be eliminated by using a local reduction.

$$\dfrac{\dfrac{\dfrac{}{A}\,v}{B \supset A}\supset I^v}{A \supset B \supset A}\supset I^u \qquad\qquad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A}\,u \qquad \dfrac{}{B}\,v}{A \wedge B}\wedge I}{A}\wedge E_l}{B \supset A}\supset I^v}{A \supset B \supset A}\supset I^u$$

It is these detours which make it difficult to argue that our system is consistent, i.e. from no assumptions we cannot derive falsehood. Gentzen hence introduced as a technical device another calculus, a sequent calculus, where such detours are not present. In fact, it is fairly obvious that falsehood is not derivable in the sequent calculus when we have no assumptions. Hence, consistency of the sequent calculus is obvious. He then showed that the sequent calculus plus one additional rule, the cut-rule, is equivalent to the natural deduction system. This is fairly easy. The hard part is to show that the cut-rule is *admissible*, i.e. it is not necessary. As a consequence, we know something stronger: all propositions provable in the natural deduction system are also provable in the sequent calculus without cut. Since we know that the sequent calculus is consistent, we hence also know that the natural deduction calculus is.

The sequent calculus is not only of interest because it is a convenient technical device for establishing consistency of natural deduction. It also gives rise to proof search procedures. In fact, we can study its proof-theoretic properties further and arrive at proof systems where proof search is feasible and amendable to efficient implementations.

5.1 Normal Natural Deductions

As a technical device, we introduce a natural deduction calculus where we restrict derivations to *normal derivations,* i.e. derivations which do not admit detours. As we have seen when we considered proof terms for the natural deduction system, a detour meant we had a redex in our proof term, i.e. a subterm which can be reduced; our proof term is in *normal form*. In fact, it might be helpful to consider the subset of terms consisting of lambda-terms, applications, pairs and projections to characterize normal forms more precisely. We will subsequently add the other proof terms as well.

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

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.

The normal natural deduction calculus, the proof calculus which only admits normal terms, captures also a very intuitive simple strategy which we already used informally when constructing proofs. When proving a proposition, we use introduction rules reasoning bottom-up (from the proposed theorem towards the hypothesis) and elimination rules top-down (from the assumptions towards the proposed theorem) meeting the result of the intro-rules.

We will introduce two new judgements to describe this proof strategy:

$M : A \uparrow$	Proposition A has a normal deduction described by the normal term M
$R\ : A \downarrow$	Proposition A is extracted from a hypothesis described the the neutral term R

We immediately give the judgements annotated with proof terms; this highlights that we are only constructing normal terms. However, we will often simply write $A \uparrow$ and $A \downarrow$, if the proof term itself is not of interest to us.

All assumptions will be written as $x : A \downarrow$, and hence the localized form, i.e. the form where explicitly list our assumptions, can be described as

$$\begin{array}{lcl} u_1{:}A_1 \downarrow, \ldots, u_n{:}A_n \downarrow & \vdash & M : A \uparrow \\ u_1{:}A_1 \downarrow, \ldots, u_n{:}A_n \downarrow & \vdash & R \ : A \downarrow \end{array}$$

We write $\Gamma^{\downarrow}$ for a context $u_1{:}A_1 \downarrow, \ldots, u_n{:}A_n \downarrow$.

Let us know revisit the natural deduction rules.

Hypothesis The general hypothesis rule simply reflects the fact that from a list of assumptions, we can extract one.

$$\frac{}{\Gamma_1^{\downarrow},\ u{:}A \downarrow,\ \Gamma_2^{\downarrow} \vdash u : A \downarrow}\, u$$

Coercion The introduction and elimination rules must be able to meet and we need to be able to switch to extracting information from the assumptions. From the proof term point of view, every neutral term is also a normal term. This is achieved by a coercion.

$$\frac{\Gamma^{\downarrow} \vdash R : A \downarrow}{\Gamma^{\downarrow} \vdash R : A \uparrow}\, {\downarrow}{\uparrow}$$

Note that the opposite direction is not allowed; not all normal terms are neutral terms. It would also contradict our intended strategy.

Conjunction The rules for conjunction are straightforward.

$$\frac{\Gamma^{\downarrow} \vdash M : A \uparrow \qquad \Gamma^{\downarrow} \vdash N : B \uparrow}{\Gamma^{\downarrow} \vdash \langle M,\ N \rangle : A \wedge B \uparrow}\, {\wedge}I$$

$$\frac{\Gamma^{\downarrow} \vdash R : A \wedge B \downarrow}{\Gamma^{\downarrow} \vdash \mathsf{fst}\ R : A \downarrow}\, {\wedge}E_l \qquad \frac{\Gamma^{\downarrow} \vdash R : A \wedge B \downarrow}{\Gamma^{\downarrow} \vdash \mathsf{snd}\ M : B \downarrow}\, {\wedge}E_r$$

Truth The rule for truth, is also straightforward. As it is an introduction rule, it constructs a normal derivation.

$$\frac{}{\Gamma^{\downarrow} \vdash () : \top \uparrow}\, \top I$$

Implication The rule for implication follows a similar recipe as before. In the introduction rule, we add a new assumption labelled as neutral. In the elimination rule, we extract from the assumptions in $\Gamma^{\downarrow}$ a proof R for $A \supset B$; we now can verify that A has a normal derivation described by M and are able to extract a proof for B described by the neutral term $R\ M$.

$$\frac{\Gamma^{\downarrow}, u{:}A \downarrow \vdash M : B \uparrow}{\Gamma^{\downarrow} \vdash \lambda x{:}A.M : A \supset B \uparrow} \supset I^{u} \qquad \frac{\Gamma^{\downarrow} \vdash R : A \supset B \downarrow \qquad \Gamma^{\downarrow} \vdash M : A \uparrow}{\Gamma^{\downarrow} \vdash R\,M : B \downarrow} \supset E$$

Disjunction The introduction rules can easily be annotated with $\mathtt{norm}$. For the elimination rule we again extract the premise $A \vee B$, hence annotating it with $\downarrow$. We choose to identify the main conclusion C with a normal term annotating it with $\uparrow$.

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

$$\frac{\Gamma^{\downarrow} \vdash R : A \vee B \downarrow \qquad \Gamma^{\downarrow}, x{:}A \downarrow \vdash M_l : C \uparrow \qquad \Gamma^{\downarrow}, y{:}B \downarrow \vdash M_r : C \uparrow}{\Gamma^{\downarrow} \vdash \mathsf{case}\ R\ \mathsf{of}\ \mathsf{inl}^{A}\ x \rightarrow M_l \mid \mathsf{inr}^{B}\ y \rightarrow M_r : C \uparrow} \vee E^{x,y}$$

It would also be consistent to allow the derivations of C to be extractions, but it is not necessary to obtain a complete search procedure and complicates the relation to the sequent calculus. It also complicates our computational reading of the disjunction rule, since it would mean we extract a proposition C_l in the first branch and a (possibly another) proposition C_r in the second branch, and we then need to check that they are the same.

Falsehood If we can synthesize a contradiction, then we have constructed a proof for C. It would not make sense to have C being synthesized, i.e. being annotated with $\downarrow$, since it would be completely unrestricted.

$$\frac{\Gamma^{\downarrow} \vdash R : \bot \downarrow}{\mathsf{abort}^{C}\ M : C \uparrow} \bot E$$

Exercise 5.1.1.

Annotate the rules for universal and existential quantifiers

Exercise 5.1.2.

Annotate the rules for negation

$$\frac{\Gamma, u : A \vdash p}{\Gamma \vdash \neg A}\neg I^p \qquad \frac{\Gamma \vdash \neg A \qquad\qquad \Gamma \vdash A}{\Gamma \vdash C}\neg E$$

Theoretical properties It is quite easy to see that normal and neutral derivations are sound with respect to natural deduction. In order to state and prove this theorem, we introduce some conventions, namely we can obtain $\Gamma = u_1{:}A_1, \ldots, u_n{:}A_n$ from $\Gamma^{\downarrow}$ simply by dropping the $\downarrow$ annotation from each assumption. In the opposite direction, we simply add the $\downarrow$ annotation to each assumption to obtain a $\Gamma^{\downarrow}$ from Γ.

Theorem 5.1.1 (Soundness). *1. If* $\Gamma^{\downarrow} \vdash M : A \uparrow$ *then* $\Gamma \vdash M : A$ *and*

2. If $\Gamma^{\downarrow} \vdash R : A \downarrow$ *then* $\Gamma \vdash R : A$.

Proof. By induction on the structure of the given derivation. We show only three cases, since the proof is straightforward.

Case $\mathcal{D} = \dfrac{}{\Gamma_1^{\downarrow},\ x{:}A \downarrow,\ \Gamma_2^{\downarrow} \vdash x : A \downarrow}x$

Then we can construct directly $\Gamma_1,\ x{:}A,\ \Gamma_2 \vdash A$.

Case $\mathcal{D} = \dfrac{\overset{\mathcal{D}'}{\Gamma^{\downarrow} \vdash R : A \downarrow}}{\Gamma^{\downarrow} \vdash R : A \uparrow}\uparrow\downarrow$

$\Gamma \vdash R : A$ by i.h. on $\mathcal{D}'$

Case $\mathcal{D} = \dfrac{\overset{\mathcal{D}'}{\Gamma^{\downarrow}, x{:}A \downarrow \vdash M : B \uparrow}}{\Gamma^{\downarrow} \vdash \lambda x{:}A.M : A \supset B \uparrow}\supset^x$

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

□

However, we note that we restricted what derivations we allow; so clearly, it is not obvious that we can translate every natural deduction derivation into a normal natural deduction proof. For example, the derivation we have given at the beginning of the chapter cannot be annotated with our rules.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \downarrow}\,u \qquad \dfrac{}{B \downarrow}\,v}{A \wedge B}\,??}{A \downarrow}\,\uparrow\downarrow}{A \uparrow}\,\wedge E_l}{B \supset A \uparrow}\,\supset I^v}{A \supset B \supset A \uparrow}\,\supset I^u$$

The problem is that while from $A \wedge B \downarrow$ we can extract $A \downarrow$, we cannot construct $A \wedge B \downarrow$. Given our assumptions $A \downarrow$ we can turn it into $A \uparrow$; similarly, we can turn $B \downarrow$ into $B \uparrow$, and conclude $A \wedge B \uparrow$. But there is not way to go from $A \wedge B \uparrow$ to $A \wedge B \downarrow$ - only the opposite direction is allowed!

To resurrect completeness, we temporarily allow the conversion

$$\dfrac{\Gamma^{\downarrow} \vdash M : A \uparrow}{\Gamma^{\downarrow} \vdash (M{:}A) : A \downarrow}\,\downarrow\uparrow$$

Computationally, we can read this rule as follows: we can synthesize a type A for the expression $M : A$, if M checks against A. We keep the proposition we check M against as part of the proof term we construct as evidence in the conclusion. Hence, this rule allows explicit type annotations where we transition.

With this rule $\downarrow\uparrow$ we can now of course translate also the non-normal derivation above. We will distinguish between the bi-directional natural deduction system *with* $\downarrow\uparrow$ rule, written as $\Gamma^{\downarrow} \vdash^{+} J$, and the bi-direction natural deduction system *without* $\downarrow\uparrow$, written as $\Gamma^{\downarrow} \vdash J$. In other words $\Gamma^{\downarrow} \vdash^{+} J$ contains all the bi-directional rules from $\Gamma^{\downarrow} \vdash J$, but in addition we can mark and identify the transitions where our derivation is not normal. These places are justified by the $\downarrow\uparrow$ rule.

$\Gamma^{\downarrow} \vdash M : A \uparrow$ $\Gamma^{\downarrow} \vdash R : A \downarrow$	Characterize only normal derivations
$\Gamma^{\downarrow} \vdash^{+} M : A \uparrow$ $\Gamma^{\downarrow} \vdash^{+} R : A \downarrow$	Characterize all normal derivations and identify non-normal derivations via the rule $\downarrow\uparrow$.

It is easy to show that the extended bi-directional natural deduction system is sound and complete with respect to the original natural natural deduction system.

Theorem 5.1.2 (Soundness).

1. *If* $\Gamma^{\downarrow} \vdash^{+} M : A \uparrow$ *then* $\Gamma \vdash M : A$.
2. *If* $\Gamma^{\downarrow} \vdash^{+} R : A \downarrow$ *then* $\Gamma \vdash R : A$.

Proof. By simultaneous structural induction over the structure of the given derivations. $\square$

Since adding proof terms complicates the completeness theorem and the proof slightly, we omit the proof terms in the statement and proof below. In essence, the proof terms we have in the original system are not exactly the same as the ones in the bi-directional system, because we have type annotations at normal terms which are embedded within neutral terms.

Theorem 5.1.3 (Completeness of annotated natural deduction).

1. *If* $\Gamma \vdash A$ *then* $\Gamma^{\downarrow} \vdash^{+} A \uparrow$ *and* $\Gamma^{\downarrow} \vdash^{+} A \downarrow$.

Proof. By induction over the structure of the given derivation. We show only two cases. We often refer to $\Gamma^{\downarrow} \vdash^{+} A \uparrow$ as (1) and $\Gamma^{\downarrow} \vdash^{+} A \downarrow$ as (2).

Case $\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Gamma \vdash A \supset B\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Gamma \vdash A\end{matrix}}{\Gamma \vdash B} \supset E$

$\Gamma^{\downarrow} \vdash A \supset B \downarrow$	by i.h. (2)
$\Gamma^{\downarrow} \vdash A \uparrow$	by i.h. (1)
$\Gamma^{\downarrow} \vdash B \downarrow$	by $\supset E$ proving (2)
$\Gamma^{\downarrow} \vdash B \uparrow$	by $\uparrow\downarrow$, proving (1)

Case $\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Gamma, x{:}A \vdash B\end{matrix}}{\Gamma \vdash A \supset B} \supset I^{x}$

$\Gamma^{\downarrow}, x{:}A \downarrow \vdash B \uparrow$	by i.h. (1)
$\Gamma^{\downarrow} \vdash A \supset B \uparrow$	by $\supset I^{x}$ proving (1)
$\Gamma^{\downarrow} \vdash A \supset B \downarrow$	by $\downarrow\uparrow$ proving (2) $\square$

Note that although natural deduction and bi-directional natural deduction extended with $\downarrow\uparrow$ rule are very similar, they are not in a bijective correspondence. In the bi-directional natural deduction system we can simply alternate the two coercions an arbitrary number of times and they are identified explicitly, while in the natural deduction system they are invisible.

Finally, we state some substitution properties for normal natural deductions. They take the following form.

Lemma 5.1.4 (Substitution property for normal natural deductions).

1. *If* $\Gamma_1^\downarrow, u{:}A\downarrow, \Gamma_2^\downarrow \vdash C\uparrow$ *and* $\Gamma_1^\downarrow \vdash A\downarrow$ *then* $\Gamma_1^\downarrow, \Gamma_2^\downarrow \vdash C\uparrow$.
2. *IF* $\Gamma_1^\downarrow, u{:}A\downarrow, \Gamma_2^\downarrow \vdash C\downarrow$ *and* $\Gamma_1^\downarrow \vdash A\downarrow$ *then* $\Gamma_1^\downarrow, \Gamma_2^\downarrow \vdash C\downarrow$.

Proof. By induction on the structure of the given derivation of $C\uparrow$ and $C\downarrow$ using weakening. □

5.1.1 Sequent calculus

In this section, we develop a closely related calculus to the bi-directional natural deduction system, called the sequent calculus. Studying meta-theoretical properties such as consistency is easier in this system; moreover, it provides a good calculus for proof search.

In the bi-directional natural deduction calculus, we keep the context $\Gamma^\downarrow$ for bookkeeping, but all the action happens on the right hand side of $\vdash^+$ (or $\vdash$). In particular, we switch from reasoning bottom-up via introduction rules to reasoning top-down via elimination rules; this switch is identified by the $\uparrow\downarrow$ rule.

In the sequent calculus, we only reason from the bottom-up by turning elimination rules into *left* rules that directly manipulate our assumptions in the context Γ. The judgement for a *sequent* is written as:

$$u_1{:}A_1,\ \ldots,\ u_n{:}A_n \implies C$$

Note that the proposition C on the right directly corresponds to the proposition whose truth is established by a natural deduction. The propositions on the left however do not directly correspond to hypothesis in natural deduction, since in general they include hypothesis and propositions derived from assumptions by elimination rules. It is important to keep this difference in mind when we relate sequent proofs to natural deduction derivations.

Since the order of the propositions on the left is irrelevant, we write $\Gamma, u{:}A$ instead of the more pedantic $\Gamma, u{:}A, \Gamma'$.

Initial sequent The initial sequent allows us to conclude from our assumptions that A is true. Note that the initial sequent does not correspond to the hypothesis rule in natural deduction; it corresponds to the coercion rule $\uparrow\downarrow$, since our left hand side contains not only assumptions introduced by for example $\supset$ introduction (right), but also additional assumptions we have extracted from it.

$$\frac{}{\Gamma, u{:}A \Longrightarrow A}\text{init}$$

Conjunction The right and left rules are straightforward; we turn the introduction rules into *right* rules and the elimination rules are turned upside-down.

$$\frac{\Gamma \Longrightarrow A \qquad \Gamma \Longrightarrow B}{\Gamma \Longrightarrow A \wedge B}\wedge R$$

$$\frac{\Gamma, u{:}A \wedge B, v : A \Longrightarrow C}{\Gamma, u{:}A \wedge B \Longrightarrow C}\wedge L_1 \qquad \frac{\Gamma, u{:}A \wedge B, v : B \Longrightarrow C}{\Gamma, u{:}A \wedge B \Longrightarrow C}\wedge L_2$$

In the introduction rule, read from the bottom-up, we propagate Γ to both premises. This is similar to natural deduction and reflects the fact that we can use assumptions as often as we like. In the elimination rule for $A \wedge B$ (see the rule $\wedge L_1$ and $\wedge L_2$), the assumption $A \wedge B$ persists. This reflects that assumptions may be used more than once. We analyze later which of these hypothesis are actually needed and which can be “garbage collected” if all possible information has been extracted from them. For now, however, leaving the assumptions untouched is useful since it will simplify the translation from sequent proofs to normal natural deduction derivations.

Implication The right rule is again straightforward. The left rule however is a bit more difficult. Given the assumption $A \supset B$, we can extract an additional assumption B, provided we are able to prove A.

$$\frac{\Gamma, u{:}A \Longrightarrow B}{\Gamma \Longrightarrow A \supset B}\supset R^u \qquad \frac{\Gamma, u{:}A \supset B \Longrightarrow A \qquad \Gamma, u{:}A \supset B, v{:}B \Longrightarrow C}{\Gamma, u{:}A \supset B \Longrightarrow C}\supset L$$

Disjunction Considering disjunction does not require any new considerations.

$$\frac{\Gamma \Longrightarrow A}{\Gamma \Longrightarrow A \vee B}\vee R_1 \qquad \frac{\Gamma \Longrightarrow B}{\Gamma \Longrightarrow A \vee B}\vee R_2$$

$$\frac{\Gamma, u{:}A \vee B, v{:}A \Longrightarrow C \qquad \Gamma, u{:}A \vee B, w{:}B \Longrightarrow C}{\Gamma, u{:}A \vee B \Longrightarrow C}\vee L^{v,w}$$

Truth Since there is no elimination rule for $\top$, we only have to consider the introduction rule which directly translates to a right rule in the sequent calculus.

$$\frac{}{\Gamma \Longrightarrow \top}\top R$$

Falsehood Since there is no introduction rule for $\perp$, we only consider the elimination rule which turns into a left rule.

$$\frac{}{\Gamma, u{:}\perp \Longrightarrow C}\perp L$$

Note that the left rule has no premise.

Exercise 5.1.3.

Derive the rules for universal and existential quantifiers in the sequent calculus.

Exercise 5.1.4.

Derive the rules for negation in the sequent calculus form.

5.1.2 Theoretical properties of sequent calculus

We first begin by stating and revisiting some of the structural properties.

Lemma 5.1.5 (Structural properties of sequents).

1. *(Weakening) If* $\Gamma \Longrightarrow C$ *then* $\Gamma, u{:}A \Longrightarrow C$.
2. *(Contraction) If* $\Gamma, u{:}A, v{:}A \Longrightarrow C$ *then* $\Gamma, u{:}A \Longrightarrow C$.

Proof. By structural induction on the first derivation. □

Next, we prove that our sequent calculus indeed characterizes normal natural deductions.

Theorem 5.1.6 (Soundness of sequent calculus).
If $\Gamma \Longrightarrow C$ *then* $\Gamma^{\downarrow} \Longrightarrow C \uparrow$.

Proof. By induction on the structure of the derivation $\Gamma \Longrightarrow C$. Surprisingly, this proof is straightforward and we show a few cases.

Case $\mathcal{D} = \dfrac{}{\Gamma, u{:}C \Longrightarrow C}\,\text{init}$

$\Gamma^{\downarrow}, u{:}C \downarrow \vdash C \downarrow$ by hypothesis u
$\Gamma^{\downarrow}, u{:}C \downarrow \vdash C \uparrow$ by rule $\uparrow\downarrow$

This case confirms that initial sequents correspond to coercions $\uparrow\downarrow$.

Case $\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}' \\ \Gamma, u{:}A \Longrightarrow B\end{matrix}}{\Gamma \Longrightarrow A \supset B} \supset R^{u}$

$\Gamma^{\downarrow}, u{:}A \downarrow \vdash B \uparrow$ by i.h. on $\mathcal{D}'$
$\Gamma^{\downarrow} \vdash A \supset B \uparrow$ by rule $\supset R^{u}$

Case $\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Gamma, u{:}A \supset B \Longrightarrow A\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Gamma, u{:}A \supset B, v{:}B \Longrightarrow C\end{matrix}}{\Gamma, u{:}A \supset B \Longrightarrow C} \supset L$

$\Gamma^{\downarrow}, u{:}A \supset B \downarrow \vdash A \uparrow$ by i.h. $\mathcal{D}_1$
$\Gamma^{\downarrow}, u{:}A \supset B \downarrow \vdash A \supset B \downarrow$ by hypothesis u
$\Gamma^{\downarrow}, u{:}A \supset \downarrow \vdash B \downarrow$ by rule $\supset E$
$\Gamma^{\downarrow}, u{:}A \supset B \downarrow, v{:}B \downarrow \vdash C \uparrow$ by i.h. $\mathcal{D}_2$
$\Gamma^{\downarrow}, u{:}A \supset B \downarrow \vdash C \uparrow$ by substitution property 5.1.4 □

We now establish completeness: Every normal natural deduction derivation can be translated into a sequent proof. We cannot prove this statement directly, but we need to generalize is slightly. The readers not familiar with such generalization may want to test their understanding and try proving the completeness statement directly and see where such a direct proof breaks down.

Theorem 5.1.7 (Completeness of sequent calculus).

1. *If* $\Gamma^{\downarrow} \vdash C \uparrow$ *then* $\Gamma der C$.

2. *If* $\Gamma^{\downarrow} \vdash A \downarrow$ *and* $\Gamma, u{:}A \Longrightarrow C$ *then* $\Gamma \Longrightarrow C$.

Proof. By structural induction on the structure of the given derivation $\Gamma^{\downarrow} \vdash C \uparrow$ and $\Gamma^{\downarrow} \vdash A \downarrow$ respectively.

Case $\mathcal{D} = \dfrac{}{\Gamma_1^{\downarrow}, u{:}A \downarrow, \Gamma_2^{\downarrow} \vdash A \downarrow}\, u$

$\Gamma_1, u{:}A, \Gamma_2, v{:}A \Longrightarrow C$ by assumption
$\Gamma_1, u{:}A, \Gamma_2 \Longrightarrow C$ by contraction (Lemma 5.1.5)

Case $\mathcal{D} = \dfrac{\overset{\mathcal{D}'}{\Gamma^{\downarrow} \vdash C \downarrow}}{\Gamma^{\downarrow} \vdash C \uparrow} \uparrow\downarrow$

$\Gamma, u{:}C \Longrightarrow C$ by init
$\Gamma \Longrightarrow C$ by i.h. $\mathcal{D}'$

Case $\mathcal{D} = \dfrac{\overset{\mathcal{D}'}{\Gamma^{\downarrow}, u{:}A \downarrow \vdash B \uparrow}}{\Gamma^{\downarrow} \vdash A \supset B \uparrow} \supset I^{u}$

$\Gamma, u{:}A \Longrightarrow B$ by i.h. $\mathcal{D}'$
$\Gamma \Longrightarrow A \supset B$ by $\supset R^{u}$

Case $\mathcal{D} = \dfrac{\overset{\mathcal{D}_1}{\Gamma^{\downarrow} \vdash A \supset B \downarrow} \qquad \overset{\mathcal{D}_2}{\Gamma^{\downarrow} \vdash A \uparrow}}{\Gamma^{\downarrow} \vdash B \downarrow} \supset E$

$\Gamma, u{:}B \Longrightarrow C$ by assumption
$\Gamma, w{:}A \supset B, u{:}B \Longrightarrow C$ by weakening (Lemma 5.1.5)
$\Gamma \Longrightarrow A$ by i.h. $\mathcal{D}_2$
$\Gamma, w{:}A \supset B \Longrightarrow C$ by $\supset L$
$\Gamma \Longrightarrow C$ by i.h. $\mathcal{D}_1$

□

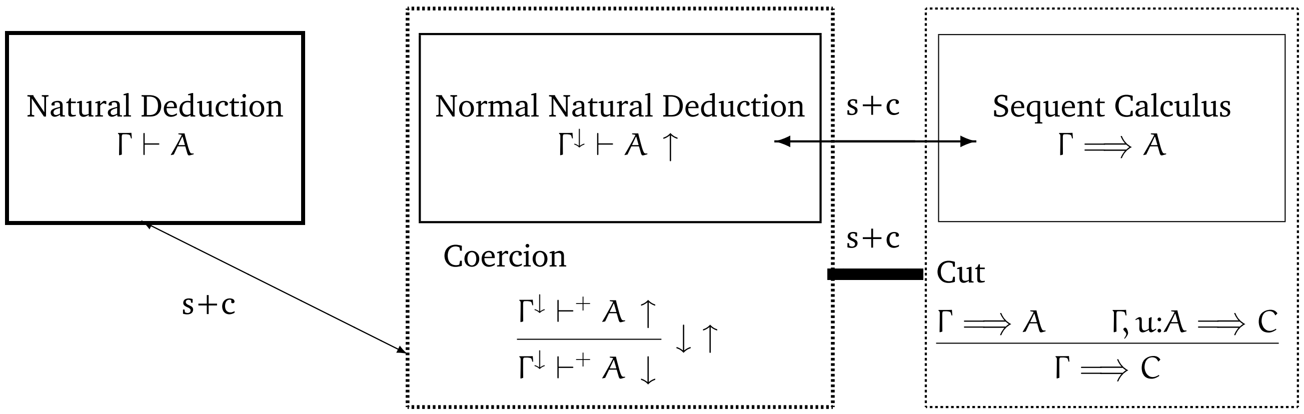
Natural Deduction
$\Gamma \vdash A$
Normal Natural Deduction
$\Gamma^{\downarrow} \vdash A \uparrow$
s+c
Sequent Calculus
$\Gamma \Longrightarrow A$
s+c
Coercion
$\dfrac{\Gamma^{\downarrow} \vdash^{+} A \uparrow}{\Gamma^{\downarrow} \vdash^{+} A \downarrow} \downarrow\uparrow$
Cut
$\dfrac{\Gamma \Longrightarrow A \qquad \Gamma, u{:}A \Longrightarrow C}{\Gamma \Longrightarrow C}$
s+c

Figure 5.1: Proof outline

In order to establish soundness and completeness with respect to arbitrary natural deductions, we need to establish a connection to the bi-directional natural deduction system extended with $\downarrow\uparrow$ rule. We will now extend the sequent calculus with one additional rule which will correspond to the coercion $\downarrow\uparrow$ and is called the *cut* rule.

$$\frac{\Gamma \overset{+}{\Longrightarrow} A \qquad \Gamma, u{:}A \overset{+}{\Longrightarrow} C}{\Gamma \overset{+}{\Longrightarrow} C}\,\text{cut}$$

The overall picture of the argument is depicted in Fig below. We can so far conclude that normal natural deduction derivations correspond to sequent derivations. However, what we still need is the link in bold black between the extended bi-directional natural deduction system and the sequent calculus with cut. If we have that link, we can show that *for all propositions which are provable in natural deduction, there exists a normal proof.*

Soundness of the extended sequent calculus with cut is straightforward.

Theorem 5.1.8 (Soundness of sequent calculus with cut). *If* $\Gamma \overset{+}{\Longrightarrow} C$ *then* $\Gamma^{\downarrow} \vdash^{+} C \uparrow$.

Proof. Induction on the structure of the given derivation as in Soundness Theorem 5.1.6 with one additional case for handling the cut rule.

Case $\mathcal{D} = \dfrac{\begin{matrix}\mathcal{D}_1 \\ \Gamma \overset{+}{\Longrightarrow} A\end{matrix} \qquad \begin{matrix}\mathcal{D}_2 \\ \Gamma, u{:}A \overset{+}{\Longrightarrow} C\end{matrix}}{\Gamma \overset{+}{\Longrightarrow} C}\,\text{cut}$

$\Gamma^{\downarrow} \vdash^{+} A \uparrow$	by i.h. $\mathcal{D}_1$
$\Gamma^{\downarrow} \vdash^{+} A \downarrow$	by $\downarrow\uparrow$
$\Gamma^{\downarrow}, v{:}A \vdash^{+} C \uparrow$	by i.h. $\mathcal{D}_2$
$\Gamma^{\downarrow} \vdash^{+} C \uparrow$	By substitution (Lemma 5.1.4) generalized)

$\square$

Indeed this highlights the fact that the cut rule corresponds to the coercion $\downarrow\uparrow$ from neutral to normal (read from bottom-up).

We can now establish completeness of the sequent calculus with cut.

Theorem 5.1.9 (Completeness of sequent calculus with cut).

1. *If* $\Gamma^{\downarrow} \vdash^{+} C \uparrow$ *then* $\Gamma \overset{+}{\Longrightarrow} C$.

2. *If* $\Gamma^{\downarrow} \vdash^{+} C \downarrow$ *and* $\Gamma, u{:}A \overset{+}{\Longrightarrow} C$ *then* $\Gamma \overset{+}{\Longrightarrow} C$.

Proof. As in the previous proof for completeness between the normal natural deduction and sequent calculus without cut with one additional case.

Case $\mathcal{D} = \dfrac{\begin{array}{c}\mathcal{D}' \\ \Gamma^{\downarrow} \vdash^{+} A \uparrow\end{array}}{\Gamma^{\downarrow} \vdash^{+} A \downarrow}\downarrow\uparrow$

$\Gamma \Longrightarrow A$	by i.h. $\mathcal{D}'$
$\Gamma, u{:}A \overset{+}{\Longrightarrow} C$	by assumption
$\Gamma \overset{+}{\Longrightarrow} C$	by cut

$\square$

We are almost finished. The main theorem still missing is that the cut-rule is not necessary. This will establish that indeed if $\Gamma \overset{+}{\Longrightarrow} C$ then $\Gamma \Longrightarrow C$. In fact, we went through this detour simply because it is easier to show that the cut-rule is not necessary rather than showing directly that all natural deduction derivations can be translated to normal derivations.

Proving that the cut-rule is admissible, is called the cut-elimination theorem (Gentzen's Hauptsatz) and it is one of the central theorems of logic. As an immediate consequence, we have that not every proposition has a proof, since no rule is applicable to derive $\cdot \Longrightarrow \bot$, i.e. given no assumptions we cannot derive falsehood. Hence, it shows that our system is (weak) consistent.

5.1.3 Cut-elimination

In this section, we show one of the fundamental main theorems in logic, i.e. that the rule of cut is redundant in the sequent calculus. First, we prove that cut is *admissible*, i.e. whenever the premise of the cut rules are derivable in the sequent calculus *without cut*, then the conclusion is. Intuitively, it should be clear that adding an admissible rule to a deductive system does not change what can be derived. Formally, we can prove by induction over the structure of derivations that may contain cuts, i.e. $\Gamma \overset{+}{\Longrightarrow} C$ then $\Gamma \Longrightarrow C$.

To prove that cut is admissible, we prove the following theorem:

$$\textit{If } \mathcal{D} : \Gamma \Longrightarrow A \textit{ and } \mathcal{E} : \Gamma, A \Longrightarrow C \textit{ then } \Gamma \Longrightarrow C$$

We call A the *cut formula*. Moreover, recall that each left or right rule in the sequent calculus focuses on an occurrence of a proposition in the conclusion, called the *prinipal formula* of the inference.

In the proof, we reason by induction on the structure of the cut formula and on the structure of the given derivations $\mathcal{D}$ and $\mathcal{E}$. Either the cut formula is strictly smaller or with an identical cut formula, we either have $\mathcal{D}$ is strictly smaller while $\mathcal{E}$ remains the same or $\mathcal{E}$ is strictly smaller while $\mathcal{D}$ remains the same. The proof will first proceed by an outer induction on the structure of the cut-formula and then on an inner induction over the structure of the derivations.

The proof is constructive, which means we show hat to transform a proof $\mathcal{E}$: $\Gamma, A \Longrightarrow C$ and a proof $\mathcal{D} : \Gamma \Longrightarrow A$ into a proof $\Gamma \Longrightarrow C$. The proof is divided into several classes of cases. More than one case may be applicable which just means that the algorithm for constructing derivations of $\Gamma \Longrightarrow C$ is non-deterministic.

Theorem 5.1.10 (Admissibility of Cut).
If $\mathcal{D} : \Gamma \Longrightarrow A$ and $\mathcal{E} : \Gamma, A \Longrightarrow C$ then $\Gamma \Longrightarrow C$.

Proof. By nested induction on the structure of A, the derivation $\mathcal{D}$ of $\Gamma \Longrightarrow A$ and $\mathcal{E}$ of $\Gamma, A \Longrightarrow C$.

Case $\mathcal{D}$ is an initial sequent.

$$\mathcal{D} = \frac{}{\Gamma', A \Longrightarrow A}\,\text{init}$$

$\Gamma = \Gamma', A$ by assumption

$\Gamma', A, A \Longrightarrow C$ by assumption $\mathcal{E}$

$\Gamma', A \Longrightarrow C$ by contraction
$\Gamma \Longrightarrow C$

Case $\mathcal{E}$ is an initial sequent and uses the cut formula

$$\mathcal{E} = \frac{}{\Gamma, A \Longrightarrow A}\,\text{init}$$

$C = A$ by assumption
$\Gamma \Longrightarrow A$ by derivation $\mathcal{D}$

Case $\mathcal{E}$ is an initial sequent and does not use the cut formula

$$\mathcal{E} = \frac{}{\Gamma', C, A \Longrightarrow C}\,\text{init}$$

$\Gamma = \Gamma', C$ by assumption
$\Gamma', C \Longrightarrow C$ by rule init
$\Gamma \Longrightarrow C$ by using the fact that $\Gamma = \Gamma', C$

Case A is the principal formula of the final inference in both $\mathcal{D}$ and $\mathcal{E}$. We show here some of the cases.

Subcase $A = A_1 \wedge A_2$.

$$\mathcal{D} = \frac{\begin{array}{c}\mathcal{D}_1\\ \Gamma \Longrightarrow A_1\end{array} \qquad \begin{array}{c}\mathcal{D}_2\\ \Gamma \Longrightarrow A_2\end{array}}{\Gamma \Longrightarrow A_1 \wedge A_2}\,\wedge R \quad \text{and} \quad \mathcal{E} = \frac{\begin{array}{c}\mathcal{E}_1\\ \Gamma, A_1 \wedge A_2, A_1 \Longrightarrow C\end{array}}{\Gamma, A_1 \wedge A_2 \Longrightarrow C}\,\wedge L_1$$

$\mathcal{D}' : \Gamma, A_1 \Longrightarrow A_1 \wedge A_2$ by weakening
$\mathcal{F}_1 : \Gamma, A_1 \Longrightarrow C$ by i.h. $A_1 \wedge A_2$, $\mathcal{D}'$ and $\mathcal{E}_1$
$\mathcal{F} : \Gamma \Longrightarrow C$ by i.h. A_1, $\mathcal{D}_1$ and $\mathcal{F}_1$

We note that weakening $\mathcal{D}$ to $\mathcal{D}'$ does not alter the size of the derivation. Hence, the appeal to the induction hypothesis using $\mathcal{D}'$ and $\mathcal{E}_1$ is valid, because $\mathcal{E}_1$ is smaller than $\mathcal{E}$. We will not be explicit about such weakening steps subsequently.

We also note that the second appeal to the induction hypothesis using $\mathcal{D}_1$ and $\mathcal{F}_1$ is valid, since the cut formula A_1 is smaller than the original cut-formula $A_1 \wedge A_2$; hence it did not matter that we do know nothing about the size of $\mathcal{F}_1$.

Subcase $A = A_1 \supset A_2$.

$$\mathcal{D} = \cfrac{\overset{\mathcal{D}_1}{\Gamma, A_1 \Longrightarrow A_2}}{\Gamma \Longrightarrow A_1 \supset A_2} \supset R \quad \text{and} \quad \mathcal{E} = \cfrac{\overset{E_1}{\Gamma, A_1 \supset A_2 \Longrightarrow A_1} \qquad \overset{E_2}{\Gamma, A_1 \supset A_2, A_2 \Longrightarrow C}}{\Gamma, A_1 \supset A_2 \Longrightarrow C} \supset L$$

$\mathcal{F}_1 : \Gamma \Longrightarrow A_1$	by i.h. $A_1 \supset A_2$, $\mathcal{D}$ and $\mathcal{E}_1$
$\mathcal{F}_2 : \Gamma \Longrightarrow A_2$	by i.h. A_1, $\mathcal{F}_1$ and $\mathcal{D}_1$
$\mathcal{E}_2' : \Gamma, A_2 \Longrightarrow C$	by i.h. $A_1 \supset A_2$, $\mathcal{D}$, and $\mathcal{E}_2$
$\mathcal{F} : \Gamma \Longrightarrow C$	by i.h. A_2, $\mathcal{F}_2$ and $\mathcal{E}_2'$

Case A is not the principal formula of the last inference in $\mathcal{D}$. In that case $\mathcal{D}$ must end in a left rule and we can directly appeal to the induction hypothesis on one of the premises.

Subcase $\mathcal{D} = \cfrac{\overset{D_1}{\Gamma', B_1 \wedge B_2, B_1 \Longrightarrow A}}{\Gamma', B_1 \wedge B_2 \Longrightarrow A} \wedge L_1$

$\Gamma = \Gamma', B_1 \wedge B_2$	by assumption
$\Gamma', B_1 \wedge B_2, B_1 \Longrightarrow C$	by i.h. A, $\mathcal{D}_1$, and $\mathcal{E}$
$\Gamma', B_1 \wedge B_2 \Longrightarrow C$	by $\wedge L_1$

Subcase $\mathcal{D} = \cfrac{\overset{\mathcal{D}_1}{\Gamma', B_1 \supset B_2 \Longrightarrow B_1} \qquad \overset{\mathcal{D}_2}{\Gamma', B_1 \supset B_2, B_2 \Longrightarrow A}}{\Gamma', B_1 \supset B_2 \Longrightarrow A} \supset L$

$\Gamma = \Gamma', B_1 \supset B_2$	by assumption
$\Gamma', B_1 \supset B_2, B_2 \Longrightarrow C$	by i.h. A, $\mathcal{D}_2$ and $\mathcal{E}$
$\Gamma', B_1 \supset B_2 \Longrightarrow C$	by $\supset L$ using $\mathcal{D}_1$ and the above
$\Gamma \Longrightarrow C$	

Case A is not the principal formula of the last inference in $\mathcal{E}$.

Subcase $\mathcal{E} = \dfrac{\overset{\mathcal{E}_1}{\Gamma, A \Longrightarrow C_1} \qquad \overset{\mathcal{E}_2}{\Gamma, A \Longrightarrow C_2}}{\Gamma, A \Longrightarrow C_1 \wedge C_2} \wedge R$

$C = C_1 \wedge C_2$	by assumption
$\Gamma \Longrightarrow C_1$	by i.h. A, $\mathcal{D}$, and $\mathcal{E}_1$
$\Gamma \Longrightarrow C_2$	by i.h. A, $\mathcal{D}$, and $\mathcal{E}_2$
$\Gamma der C_1 \wedge C_2$	by $\wedge R$ on the above

Subcase $\mathcal{E} = \dfrac{\overset{\mathcal{E}_1}{\Gamma', B_1 \wedge B_2, B_1, A \Longrightarrow C}}{\Gamma', B_1 \wedge B_2, A \Longrightarrow C} \wedge L_1$

$\Gamma = \Gamma', B_1 \wedge B_2$	by assumption
$\Gamma', B_1 \wedge B_2, B_1 \Longrightarrow C$	by i.h. A, $\mathcal{D}$, $\mathcal{E}_1$
$\Gamma', B_1 \wedge B_2 \Longrightarrow C$	by $\wedge L_1$
$\Gamma \Longrightarrow C$	by $\Gamma = \Gamma', B_1 \wedge B_2$

$\square$

As mentioned above, adding an admissible rule does not change the judgements which are derivable.

Theorem 5.1.11 (Cut Elimination).
If $\Gamma \overset{+}{\Longrightarrow} C$ *then* $\Gamma \Longrightarrow C$

Proof. By structural induction on the given derivation $\Gamma \overset{+}{\Longrightarrow} C$. The proof is straightforward, and we only write out the case for cut.

Case $\mathcal{D}^+ = \dfrac{\overset{\mathcal{D}_1^+}{\Gamma \overset{+}{\Longrightarrow} A} \qquad \overset{\mathcal{D}_2^+}{\Gamma, A \overset{+}{\Longrightarrow} c}}{\Gamma \overset{+}{\Longrightarrow} C} \text{cut}$

$\Gamma \Longrightarrow A$	by i.h. on $\mathcal{D}^+ + 1$

$\Gamma, A \Longrightarrow C$ by i.h. on $\mathcal{D}^+ + 2$
$\Gamma \Longrightarrow C$ by admissibility of cut (Theorem 5.1.10)

$\square$

5.2 Consequences of Cut Elimination

The cut elimination theorem is the central piece to complete our proof that it suffices to concentrate on normal natural deduction derivations to find a proof for a given proposition, i.e. if $\Gamma \vdash A$ then $\Gamma^{\downarrow} \vdash A \uparrow$.

Theorem 5.2.1 (Normalization for Natural Deduction).
If $\Gamma \vdash A$ *then* $\Gamma^{\downarrow} \vdash A \uparrow$.

Proof. Direct from the previous lemmas and theorems.

$\Gamma \vdash A$ by assumption
$\Gamma^{\downarrow} \vdash^{+} A \uparrow$ by completeness of natural deduction (Theorem 5.1.3)
$\Gamma \overset{+}{\Longrightarrow} A$ by completeness of sequent calculus with cut (Theorem 5.1.7)
$\Gamma \Longrightarrow A$ by completeness of sequ. calc. without cut (Cut-elimination Thm. 5.1.11)
$\Gamma^{\downarrow} \vdash A \uparrow$ by soundness of sequent calculus (Theorem 5.1.6)

$\square$

Our normalization theorem justifies that for every proof $\Gamma \vdash A$, there exists some cut-free proof of the same theorem. This is often referred to as *weak normalization*: it suffices to provide some strategy of eliminating the cut.

Another important consequence of cut-elimination is that to find a proof for A in the natural deduction calculus, it suffices to show that there exists a proof in the sequent calculus without cut. As a consequence, if we want a proof for $\perp$, it suffices to show that there exists a proof $\cdot \Longrightarrow \perp$. Since $\Gamma = \cdot$, it is empty, we could not have used a left rule to derive $\perp$. However, there is no right rule which ends with $\perp$. Therefore, it is impossible to derive $\cdot \Longrightarrow \perp$.

Corollary 5.2.2 (Consistency). *There is no derivation for* $\cdot \vdash \perp$.

Proof. Assume there is a proof for $\cdot \vdash \perp$. Then by completeness of annotated deduction (Theorem 5.1.3) and completeness of seq. calculus with cut (Theorem 5.1.7) and cut-elimination (Thm. 5.1.11), it suffice to show that there is a proof for $\cdot \Longrightarrow \perp$. Since $\Gamma = \cdot$, there is no principal formula on the left and no left rule is applicable. There is also no right rule which ends in $\perp$. Therefore $\cdot \Longrightarrow \perp$ cannot be derived and hence $\cdot \vdash \perp$ is not derivable. $\square$

Another consequence, is that we can show that the excluded middle $A \vee \neg A$ is not derivable. We also say that $A \vee \neg A$ is independent for arbitrary A.

Corollary 5.2.3 (Independence of Excluded Middle).
There is no deduction of $\vdash A \vee \neg A$ *for arbitrary* A.

Proof. Assume there is a derivation $\vdash A \vee \neg A$. By the completeness results and cut-elimination, it suffices to show $\cdot \Longrightarrow A \vee \neg A$. By inversion, we must have either $\cdot \Longrightarrow A$ or $\cdot \Longrightarrow \neg A$. The former judgement $\cdot \Longrightarrow A$ has no derivation. The latter judgement can only be inferred from $A \Longrightarrow \bot$. But there is no right rule with the conclusion $\bot$ and we cannot prove given an arbitrary A we can derive a contradiction. Hence, there cannot be a deduction of $\vdash A \vee \neg A$. $\square$

Cut-elimination justifies that we can concentrate on finding a normal proof for a proposition A. We can also observe that proofs in the sequent calculus without cut are already much more restricted. Hence they are more amendable to proof search. The sequent calculus is hence an excellent foundation for proof search strategies. However, some non-determinism is still present. Should we apply a right rules or a left rule? And if we choose to apply a left rule, which formula from Γ should we pick as our principal formula?

Without techniques to restrict some of these choices, proof search is still infeasible. However, the sequent calculus lends itself to study these choices by considering two important properties: *inversion properties* allow us apply rules eagerly, i.e. their order does not matter (don't care non-determinism), and *focusing properties* allow us to chain rules which involve a choice and order does matter (do-care non-determinism), i.e. we make a choice we might as well continue to make choices and not postpone them.

5.3 Towards a focused sequent calculus

The simplest way to avoid non-determinism is to consider those propositions on the left or right for which a unique way to apply a rule. The following property essentially justifies that if we have the conclusion, then we must have had a proof of the premises. For example, to prove $A \wedge B$, we can immediately apply the right rule without loosing completeness. On the other hand, to prove $A \vee B$, we cannot immediately apply the right rule. As a counter example, consider $B \vee A \Longrightarrow A \vee B$; we need to apply first the left rule for splitting the derivation and then apply the right rule.

Inversion property:
The premises of an inference rule are derivable, if and only if the conclusion.

Given a sequent, a number of invertible rules may be applicable. However, the order of this choice, i.e. when to apply an invertible rule, does not matter. In other words, we are replacing *don't know non-determinism* by *don't care non-determinism*.

For controlling and restricting the search space, we can refine the inversion property as stated above further. In particular, in left rules, the principal formula is still present in the premises which means we can continue to apply the same left rule over and over again leading to non-termination. So we require in addition that the principal formula of a left rule is no longer needed, thereby guaranteeing the termination of the inversion phase.

Theorem 5.3.1 (Inversion).

1. *If* $\Gamma \Longrightarrow A \wedge B$ *then* $\Gamma \Longrightarrow A$ *and* $\Gamma \Longrightarrow B$

2. *If* $\Gamma \Longrightarrow A \supset B$ *then* $\Gamma, A \Longrightarrow B$.

3. *If* $\Gamma, A \wedge B \Longrightarrow C$ *then* $\Gamma, A, B \Longrightarrow C$.

4. *If* $\Gamma, A \vee B \Longrightarrow C$ *then* $\Gamma, A \Longrightarrow C$ *and* $\Gamma, B \Longrightarrow C$.

Proof. Proof by structural induction on the given derivation; or, simply taking advantage of cut.

$\Gamma \Longrightarrow A \wedge B$	by assumption
$\Gamma, A \wedge B, A \Longrightarrow A$	by init
$\Gamma, A \wedge B \Longrightarrow A$	by $\wedge L_1$
$\Gamma \Longrightarrow A$	by cut

□

We observe that $\top R$ and $\bot L$ are special; they can be applied eagerly, but they have no premises and therefore do not admit an inversion theorem.

There is a remarkable symmetry between the rules which are invertible and which are not. This picture is slightly spoiled by the left rule for conjunction. This can be corrected in moving to linear logic which we will not pursue here.

Chaining all invertible rules together, already gives us a good proof search strategy. However, it still leaves us with many possible choices once we have applied all invertible rules. How should one proceed to handle such choices? For example, if we pick a rule $\vee R_1$ for $(A \vee B) \vee C$, we now need to prove $A \vee B$ again facing a choice. Shall we stick to our committed choice and further split $A \vee B$ or shall we revisit other possible choices we have? - It turns out that *focusing* properties justify that we can stick to a choice we have made and continue to make further choices.

Focusing properties are dual to inversion properties; while inversion properties allow us to chain together invertible rules, focusing properties allow us to chain together non-invertible rules committing to a choice.

The focusing property and the duality between invertible / non-invertible rules was first observed by Andreoli in linear logic which did not show the anomaly for conjunction. Following Andreoli, we can classify formulas into positive and negative propositions where *positive propositions* are non-invertible on the right, but invertible on the left and *negative propositions* are non-invertible on the left, but invertible on the right.

Formula	A, B	$::=$	$R \mid L$
Positive	R	$::=$	$P^+ \mid A_1 \vee A_2 \mid \exists x{:}\tau.A(x) \mid \bot \mid A_1 \wedge A_2$
Negative	L	$::=$	$P^- \mid A_1 \supset A_2 \mid \forall x{:}\tau.A(x) \mid A_1 \wedge A_2$
Stable Context	Δ	$::=$	$\cdot \mid \Delta, L$

Moreover, we will describe a stable context Δ which only consists of negative formulas. Intuitively, we will first consider first a formula A and apply invertible rules on the right until we obtain a *positive proposition*; at this point, we shift our attention to the context of assumptions and apply invertible left rules until our context is *stable*, i.e. it contains only negative propositions. At this point we have to make a choice. This phase is called the *asynchronous phase*.

From the asynchronous phase, we transition to the synchronous phase. We either commit to work on the right hand side of the sequent ($\Delta > R$) or we commit to work on the left hand side choosing an assumption from Δ, i.e. $\Delta > L \Longrightarrow R$. Let us summarize the four judgements:

$\Delta; \Gamma \Longrightarrow [L]$	Asynchronous phase (right)
$\Delta; [\Gamma] \Longrightarrow R$	Asynchronous phase (left)
$\Delta > R$	Synchronous phase (right)
$\Delta > L \Longrightarrow R$	Synchronous phase (left)

The notation $\Delta > R$ and $\Delta > L \Longrightarrow R$ is chosen to draw attention to the part we focus on via $>$; the left hand side of $>$ describes the narrowing the focus of our attention.

Synchronous phase (left)

$$\frac{\Delta > A \qquad \Delta > B \Longrightarrow R}{\Delta > A \supset B \Longrightarrow R}$$

$$\frac{\Delta > A(t) \Longrightarrow R}{\Delta > \forall x.A(x) \Longrightarrow R}$$

$$\frac{\Delta > A_i \Longrightarrow R}{\Delta > A_1 \wedge A_2 \Longrightarrow R}$$

Asynchronous phase (right)

$$\frac{\Delta; \Gamma, A \Longrightarrow [\ B\ ]}{\Delta; \Gamma \Longrightarrow [\ A \supset B\ ]}$$

$$\frac{\Delta; \Gamma \Longrightarrow [\ A(a)\ ]}{\Delta; \Gamma \Longrightarrow [\ \forall x.A(x)\ ]}$$

$$\frac{\Delta; \Gamma \Longrightarrow [\ A\ ] \qquad \Delta; \Gamma \Longrightarrow [\ B\ ]}{\Delta; \Gamma \Longrightarrow [\ A \wedge B\ ]}$$

Asynchronous phase (left)

$$\frac{\Delta; [\ \Gamma, A(a)\ ] \Longrightarrow R}{\Delta; [\ \Gamma, \exists x.A(x)\ ] \Longrightarrow R}$$

$$\frac{\Delta; [\ \Gamma, A\ ] \Longrightarrow R \qquad \Delta; [\ \Gamma, B\ ] \Longrightarrow R}{\Delta; [\ \Gamma, A \vee B\ ] \Longrightarrow R}$$

$$\frac{\Delta; [\ \Gamma, A, B\ ] \Longrightarrow R}{\Delta; [\ \Gamma, A \wedge B\ ] \Longrightarrow R}$$

Synchronous phase (right)

$$\frac{\Delta > A(t)}{\Delta > \exists x.A(x)}$$

$$\frac{\Delta > A_i}{\Delta > A_1 \vee A_2}$$

$$\frac{\Delta > A_1 \qquad \Delta > A_2}{\Delta > A_1 \wedge A_2}$$

Identity (positive)

$$\frac{}{\Delta, P > P^+}$$

Identity (negative)

$$\frac{}{\Delta > P \Longrightarrow P^-}$$

Transition rules

$$\frac{L \in \Delta \quad \Delta > L \Longrightarrow R}{\Delta; [\ \cdot\ ] \Longrightarrow R}\,\text{choice}_L \qquad \frac{\Delta > R}{\Delta; [\ \cdot\ ] \Longrightarrow R}\,\text{choice}_R$$

$$\frac{\Delta; \cdot \Longrightarrow [\ L\ ]}{\Delta > L}\,\text{Blur}_R \qquad \frac{\Delta; [\ R\ ] \Longrightarrow R'}{\Delta > R \Longrightarrow R'}\,\text{Blur}_L$$

$$\frac{\Delta, L; [\ \Gamma\ ] \Longrightarrow R}{\Delta; [\ \Gamma, L\ ] \Longrightarrow R}\,\text{Move-to-stable-context} \qquad \frac{\Delta; [\ \Gamma\ ] \Longrightarrow R}{\Delta; \Gamma \Longrightarrow [\ R\ ]}\,\text{Transition-to-left}$$

Let's work through an example. We first specify a predicate $\mathsf{fib}(n, x)$ which reads x is the fibonacci number corresponding to n.

$$\begin{aligned}\mathcal{P} = \; & \mathsf{fib}(0,0),\ \mathsf{fib}(1,1),\\ & \forall n \forall x \forall y. \mathsf{fib}(n,x) \supset \mathsf{fib}(s\ n,\ y) \supset \mathsf{fib}(s\ s\ n,\ x+y)\end{aligned}$$

We now want to prove $\mathcal{P} \Longrightarrow \mathsf{fib}(3,2)$. We will consider here the derivation beginning with the focusing phase. Moreover, we will treat the predicate $\mathsf{fib}(m,r)$ as negative. Note that focusing essentially leaves us no choice in the derivation shown below.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\mathcal{P} \Longrightarrow \mathsf{fib}(1,1)}{\mathcal{P} > \mathsf{fib}(1,1)} \qquad \dfrac{\dfrac{\mathcal{P} \Longrightarrow \mathsf{fib}(2,1)}{\mathcal{P} > \mathsf{fib}(2,1)} \qquad \dfrac{}{\mathcal{P} > \mathsf{fib}(3,2) \Longrightarrow \mathsf{fib}(3,2)}\,\text{init}}{\mathcal{P} > \mathsf{fib}(s\ 1,\ 1) \supset \mathsf{fib}(s\,s\,1,\ 1+1) \Longrightarrow \mathsf{fib}(3,2)}}{\mathcal{P} > \mathsf{fib}(1,1) \supset \mathsf{fib}(s\ 1,\ 1) \supset \mathsf{fib}(s\,s\,1,\ 1+1) \Longrightarrow \mathsf{fib}(3,2)}}{\mathcal{P} > \forall y. \mathsf{fib}(1,1) \supset \mathsf{fib}(s\ 1,\ y) \supset \mathsf{fib}(s\,s\,1,\ 1+y) \Longrightarrow \mathsf{fib}(3,2)}}{\mathcal{P} > \forall x \forall y. \mathsf{fib}(1,x) \supset \mathsf{fib}(s\ 1,\ y) \supset \mathsf{fib}(s\,s\,1,\ x+y) \Longrightarrow \mathsf{fib}(3,2)}}{\mathcal{P} > \forall n \forall x \forall y. \mathsf{fib}(n,x) \supset \mathsf{fib}(s\,n,\ y) \supset \mathsf{fib}(s\,s\,n,\ x+y) \Longrightarrow \mathsf{fib}(3,2)}}{\mathcal{P}; [\ \cdot\ ] \Longrightarrow \mathsf{fib}(3,3)}$$

Note that because we have chosen the predicate fib to be negative, we must blur our focus in the two open derivations and also the application of the init rule is determined. We can now in fact collapse this derivation into a "big-step" derived rule:

$$\dfrac{\mathcal{P} \Longrightarrow \mathsf{fib}(1,1) \qquad \mathcal{P} \Longrightarrow \mathsf{fib}(2,1)}{\mathcal{P} \Longrightarrow \mathsf{fib}(3,2)}$$

This rule exactly captures our intuition. Another "big-step" rule which can be derived is:

$$\dfrac{\mathcal{P} \Longrightarrow \mathsf{fib}(0,0) \qquad \mathcal{P} \Longrightarrow \mathsf{fib}(1,1)}{\mathcal{P} \Longrightarrow \mathsf{fib}(2,1)}$$

Proof search then amounts to searching over "big-step" rules - this makes proof search more efficient and easier to interact with.

We can prove that our given focused calculus is sound and complete. It is also interesting to note that by giving different polarities to atoms, we obtain different proof search strategies - assigning negative polarities to atoms allows us to model backwards proof search as we have illustrated; assigning positive polarities to atoms in fact leads to forward proof search. Hence the system is general enough to model different proof search strategies. This was observed by Chaudhuri and Pfenning.

Moreover, it is worth noting that we can give it a computational interpretations; focusing calculi provide a type system for languages supporting pattern matching and different proof strategies correspond to different evaluation strategies modelling call-by-value or call-by-name evaluation (see for example work by Noam Zeilberger).

Chapter 6

Normalization

We discuss here an alternative proof method for proving normalization. We will focus here on a *semantic* proof method using *saturated sets*. This proof method goes back to Girard (1972) building on some previous ideas by Tait.

The key question is how to prove that give a lambda-term, its evaluation terminates, i.e. normalizes. Recall the lambda-calculus together with its reduction rules.

$$\text{Terms} \quad M, N \quad ::= \quad x \mid \lambda x.M \mid M\,N$$

We consider as the main rule for reduction (or evaluation) applying a term to an abstraction, called β-*reduction*.

$$(\lambda x.M)\ N \quad \longrightarrow \quad [N/x]M \qquad\qquad \beta\text{-reduction}$$

The β-reduction rule only applies once we have found a redex. However, we also need congruence rules to allow evaluation of arbitrary subterms.

$$\frac{M \longrightarrow M'}{M\,N \longrightarrow M'\,N} \qquad \frac{N \longrightarrow N'}{M\,N \longrightarrow M\,N'} \qquad \frac{M \longrightarrow M'}{\lambda x.M \longrightarrow \lambda x.M'}$$

The question then is, how do we know that reducing a well-typed lambda-term will halt? - This is equivalent to asking does a well-typed lambda-term normalize, i.e. after some reduction steps we will end up in a normal form where there are no further reductions possible. Since a normal lambda-term characterizes normal proofs, normalizing a lambda-term corresponds to normalizing proofs and demonstrates that every proof in the natural deduction system indeed has a normal proof.

Proving that reduction must terminate is not a simple syntactic argument based on terms, since the β-reduction rule may yield a term which is bigger than the term we started with. We hence need to find a different inductive argument. For the simply-typed lambda-calculus, we can prove that while the expression itself does not get smaller, the type of an expression is. This is a syntactic argument; it however does not scale to polymorphic lambda-calculus. We will here instead discuss a *semantic* proof method where we define the meaning of well-typed terms using the abstract notion of *reducibility candidates*.

6.1 General idea

We can define the meaning of a well-typed term M in the context Γ of type A as follows: for all grounding instantiations σ providing values for the variables declared in Γ, $[\sigma]M$ is in the denotation of A. We write for the denotation of A as $[\![A]\!] = \mathcal{A}$. Similarly, the denotation of Γ is written as $[\![\Gamma]\!] = \mathcal{G}$.

$[\![A]\!]$ is interpreted as the sets of strongly normalizing terms of type A, i.e. $[\![A]\!] \in \mathsf{SN}$. We prove that if a term is well-typed, then it is strongly normalizing in two steps:

Step 1 If $M \in [\![A]\!]$ then $M \in \mathsf{SN}$.

Step 2 If $\Gamma \vdash M : A$ and $\sigma \in [\![\Gamma]\!]$ then $[\sigma]M \in [\![A]\!]$.

Therefore, we can conclude that if a term M has type A then $M \in \mathsf{SN}$, i.e. M is strongly normalizing and its reduction is finite, choosing σ to be the identity substitution.

We remark first, that all variables are in the denotations of a type A, i.e. $\mathsf{Var} \subseteq [\![A]\!]$, and variables are strongly normalizing, i.e. they are already in normal form.

Next, we define the denotations of the base type o and the function type $A \to B$.

- A term $M \in [\![o]\!]$ iff M is strongly normalizing, i.e. $M \in \mathsf{SN}$.
- A term $M \in [\![A \to B]\!]$ iff $M \in [\![A]\!] \Longrightarrow [\![B]\!]$, i.e. for all $N \in [\![A]\!].M\ N \in [\![B]\!]$.

We often write these definitions more compactly as follows

Semantic base type	$[\![o]\!]$	$:=$	SN
Semantic function type	$[\![A \to B]\!]$	$:=$	$\{M \mid \forall N \in [\![A]\!].M\ N \in [\![B]\!]\}$

6.2 Defining strongly normalizing terms

Intuitively, a term M is strongly normalizing, if there exists no infinite reduction sequence. Constructively, we can define strong normalization as follows:

Neutral terms

$$\frac{}{x \in \mathsf{SNe}} \qquad \frac{R \in \mathsf{SNe} \qquad s \in \mathsf{SN}}{R\,M \in \mathsf{SNe}}$$

Normal terms

$$\frac{R \in \mathsf{SNe}}{R \in \mathsf{SN}} \qquad \frac{M \in \mathsf{SN}}{\lambda x.M \in \mathsf{SN}} \qquad \frac{M \longrightarrow_{\mathsf{SN}} M' \qquad M' \in \mathsf{SN}}{M \in \mathsf{SN}}$$

Strong head reduction

$$\frac{N \in \mathsf{SN}}{(\lambda x.M)\ N \longrightarrow_{\mathsf{SN}} [N/x]M} \qquad \frac{R \longrightarrow_{\mathsf{SN}} R' \qquad R \text{ is not a } \lambda}{R\,M \longrightarrow_{\mathsf{SN}} R'\,M}$$

Figure 6.1: Inductive definition of strongly normalizing terms

Definition 6.2.1. *A term M is strongly normalizing, if all its reducts are strongly normalizing.*

Moreover, we have that if a given term M is strongly normalizing, then any subterm must be strongly normalizing as well. We omit the proof here and leave it to an exercise.

Theorem 6.2.1 (Subterm property of strong normalization)**.** *Any subterm of a strongly normalizing term is strongly normalizing itself.*

Here, we define inductively the set of normal terms, SN, and the set of neutral terms, SNe, using the following judgements:

$M \in \mathsf{SN}$	M is in the set of normal terms
$M \in \mathsf{SNe}$	M is in the set of neutral terms

The inductive definition given in Fig. 6.1 is often more amendable for proofs than its informal definition, since it allows us to prove properties by structural induction.

We will sketch here that these inductive definition is sound and complete with respect to our informal understanding of strongly normalizing reductions (Def. 6.2.1).

We will write $M \in \mathsf{sn}$ for M is strongly normalizing in our "informal definition", i.e. all reduction sequences starting in M are finite, to distinguish it from our inductive definition in Figure 6.1.

Lemma 6.2.2 (Properties of strongly normalizing terms)**.**

1. *If* $M \in \mathsf{sn}$ *and* $[N/x]M \in \mathsf{sn}$ *and* $N \in \mathsf{sn}$ *then* $(\lambda x.M)\ N \in \mathsf{sn}$.
2. *If* $M \in \mathsf{sn}$ *and* $N \in \mathsf{sn}$ *where* M *is not a* λ *then* $M\ N \in \mathsf{sn}$. *(we also have* $M\ N \longrightarrow M'\ N$ *and* $M'\ N \in \mathsf{sn}$ *as i.h.)*

Proof. By induction on $M \in \mathsf{sn}$ and $N \in \mathsf{sn}$. □

Lemma 6.2.3 (Closure properties of strongly normalizing terms)**.**

1. *If* $[N/x]M \in \mathsf{sn}$ *then* $M \in \mathsf{sn}$.
2. *For all variables* x, $x \in \mathsf{sn}$.
3. *If* $M \in \mathsf{sn}$ *and* $N \in \mathsf{sn}$ *where* $M = x\ \overrightarrow{N}$ *then* $M\ N \in \mathsf{sn}$.
4. *If* $M \in \mathsf{sn}$ *then* $\lambda x.M \in \mathsf{sn}$.
5. ***Expansion.*** *If* $M \longrightarrow_{\mathsf{sn}} M'$ *and* $M' \in \mathsf{sn}$ *then* $M \in \mathsf{sn}$ *where*

$$\frac{N \in \mathsf{sn}}{(\lambda x.M)\ N \longrightarrow_{\mathsf{sn}} [N/x]t} \qquad \frac{M \longrightarrow_{\mathsf{sn}} M' \quad M \textit{ is not a } \lambda}{M\ N \longrightarrow_{\mathsf{sn}} M'\ N}$$

Proof. By case analysis and induction. □

We can now prove our inductive definition to be sound and complete.

Theorem 6.2.4 (Soundness of SN)**.** 1. *If* $M \in \mathsf{SN}$ *then* $M \in \mathsf{sn}$.

2. *If* $M \in \mathsf{SNe}$ *then* $M \in \mathsf{sn}$ *and* $M = x\ \overrightarrow{N}$.
3. *If* $M \longrightarrow_{\mathsf{SN}} M'$ *then* $M \longrightarrow_{\mathsf{sn}} M'$.

Proof. By mutual structural induction on the given derivation using the closure properties. □

Theorem 6.2.5 (Completeness of SN)**.** 1. *If* $R = x\ \overrightarrow{N} \in \mathsf{sn}$ *then* $x\ \overrightarrow{N} \in \mathsf{SNe}$.

2. *If* $R = (\lambda x.M)\ N\ \overrightarrow{N} \in \mathsf{sn}$ *then* $R \longrightarrow_{\mathsf{SN}} [N/x]M\ \overrightarrow{N}$.
3. *If* $R \in \mathsf{sn}$ *then* $R \in \mathsf{SN}$.

Proof. By lexicographic induction on the height of the reduction tree of R and the height of R. □

6.3 Reducibility Candidates

One might ask, what is a good definition of a semantic type? - Rather than attempting the proof of the fundamental lemma directly and then trying to extract additional lemmas one might need about the semantic types, we follow Girard's technique and characterize some key properties our semantic types need to satisfy. If a semantic type satisfies these key properties, then our proof of the fundamental lemma will be straightforward. To put it differently, defining these key properties, will allow for a a modular proof of the fundamental lemma.

Definition 6.3.1 (Reducibility Candidate)**.** *A set* $[\![A]\!]$ *is a reducibility candidate,* $[\![A]\!] \in \mathsf{CR}$ *if the following conditions hold*

- $\mathsf{CR1}$: *If* $M \in [\![A]\!]$ *then* $M \in \mathsf{SN}$*, i.e.* $[\![A]\!] \subseteq \mathsf{SN}$.
- $\mathsf{CR2}$: *If* $M \in \mathsf{SNe}$ *then* $M \in [\![A]\!]$*, i.e.* $\mathsf{SNe} \subseteq [\![A]\!]$.
- $\mathsf{CR3}$: $\dfrac{M \longrightarrow_{\mathsf{SN}} M' \qquad M' \in [\![A]\!]}{M \in [\![A]\!]}$*, i.e.* $[\![A]\!]$ *is closed under reduction.*

The last property is often also referred to as *backward closed*. We show that that all semantic types $[\![A]\!]$ satisfy the conditions above.

Theorem 6.3.1. *For all types* C*,* $[\![C]\!] \in \mathsf{CR}$.

Proof. By induction on the structure of A. We highlight the cases below.

Case: $C = o$.

1. *Show* $\mathsf{CR1}$: By definition, for all $M \in [\![o]\!]$, we have that $M \in \mathsf{SN}$.
2. *Show* $\mathsf{CR2}$: By assumption $M \in \mathsf{SNe}$. By the definition of SN, we therefore know $M \in \mathsf{SN}$; by definition of $[\![o]\!]$, $M \in [\![o]\!]$.
3. *Show* $\mathsf{CR3}$: Trivially true, since there is no step we can take with $\longrightarrow_{\mathsf{SN}}$.

Case: $C = A \to B$

1. *Show* CR1 : if $M \in \llbracket A \to B \rrbracket$, then $M \in \mathsf{SN}$, i.e. $\llbracket A \to B \rrbracket \subseteq \mathsf{SN}$.

   Assume that $M \in \llbracket A \to B \rrbracket$, i.e. for all $N \in \llbracket A \rrbracket$, $M\ N \in \llbracket B \rrbracket$

$x \in \llbracket A \rrbracket$	by assumption $\mathsf{Var} \subseteq \llbracket A \rrbracket$
$M\ x \in \llbracket B \rrbracket$	by previous lines
$M\ x \in \mathsf{SN}$	by i.h. (CR1)
$M \in \mathsf{SN}$	by subterm property

2. *Show* CR2 :if $M \in \mathsf{SNe}$, then $M \in \llbracket A \to B \rrbracket$, i.e. $\mathsf{SNe} \subseteq \llbracket A \to B \rrbracket$.

$M \in \mathsf{SNe}$	by assumption
Assume $N \in \llbracket A \rrbracket$.	
$N \in \mathsf{SN}$	by i.h. (CR1)
$M\ N \in \mathsf{SNe}$	by def. of SNe
$M\ N \in \llbracket B \rrbracket$	by i.h. (CR2)
$M \in \llbracket A \to B \rrbracket$	by definition of $\llbracket A \to B \rrbracket$.

3. *Show* CR3 : if $M \longrightarrow_{\mathsf{SN}} M'$ and $M' \in \llbracket A \to B \rrbracket$, then $M \in \llbracket A \to B \rrbracket$.

$M \longrightarrow_{\mathsf{SN}} M'$	by assumption
$M' \in \llbracket A \to B \rrbracket$	by assumption
for all $N' \in \llbracket A \rrbracket,\ M'\ N' \in \llbracket B \rrbracket$	by definition of $\llbracket A \to B \rrbracket$
Assume $N \in \llbracket A \rrbracket$	
$M'\ N \in \llbracket B \rrbracket$	by previous lines
$M\ N \longrightarrow_{\mathsf{SN}} M'\ N$	by $\longrightarrow_{\mathsf{SN}}$
$M\ N \in \llbracket B \rrbracket$	by i.h. (CR3)
$M \in \llbracket A \to B \rrbracket$	by definition of $\llbracket A \to B \rrbracket$

□

6.4 Proving strong normalization

As mentioned before, we prove that if a term is well-typed, then it is strongly normalizing in two steps:

Step 1 If $M \in \llbracket A \rrbracket$ then $M \in \mathsf{SN}$.

Step 2 If $\Gamma \vdash M : A$ and $\sigma \in [\![\Gamma]\!]$ then $[\sigma]M \in [\![A]\!]$.

The first part described in Step 1, is satisfied by the fact that $[\![A]\!]$ must be a reducibility candidate. Hence by CR1 all terms in $[\![A]\!]$ are strongly normalizing. We now prove the second step, which is often referred to as the *Fundamental Lemma*. It states that if M has type A and we can provide "good" instantiation σ, which provides terms which are themselves normalizing for all the free variables in M, then $[\sigma]M$ is in $[\![A]\!]$.

Lemma 6.4.1 (Fundamental lemma). *If* $\Gamma \vdash M : A$ *and* $\sigma \in [\![\Gamma]\!]$ *then* $[\sigma]M \in [\![A]\!]$.

Proof. By induction on $\Gamma \vdash M : A$.

Case $\mathcal{D} = \dfrac{\Gamma(x) = A}{\Gamma \vdash x : A}$

$\sigma \in [\![\Gamma]\!]$ by assumption
$[\sigma]x \in [\![\Gamma(x)]\!] = [\![A]\!]$ by definition

Case $\mathcal{D} = \dfrac{\Gamma \vdash M : A \to B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B}$

$\sigma \in [\![\Gamma]\!]$ by assumption
$[\sigma]M \in [\![A \to B]\!]$ by i.h.
for all $N' \in [\![A]\!]$. $([\sigma]M)\ N' \in [\![B]\!]$ by definition of $[\![A \to B]\!]$
$[\sigma]N \in [\![A]\!]$ by i.h.
$[\sigma]M\ [\sigma]N \in [\![B]\!]$ by previous lines
$[\sigma](M\ N) \in [\![B]\!]$ by subst. definition

Case $\mathcal{D} = \dfrac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x.M : A \to B}$

$\sigma \in [\![\Gamma]\!]$ by assumption
Assume $N \in [\![A]\!]$
$(\sigma, N/x) \in [\![\Gamma, x : A]\!]$ by definition
$[\sigma, N/x]M \in [\![B]\!]$ by i.h.
$(\lambda x.[\sigma, x/x]M)\ N \longrightarrow_{SN} [\sigma, N/x]M$ by reduction $\longrightarrow_{SN}$
$(\lambda x.[\sigma, x/x]M) = [\sigma](\lambda x.M)$ by subst. def
$([\sigma]\lambda x.M)\ N \in [\![B]\!]$ by CR3
for all $N \in [\![A]\!]$. $([\sigma]\lambda x.M)\ N \in [\![B]\!]$ by previous lines
$[\sigma](\lambda x.M) \in [\![A \to B]\!]$ by definition of $[\![A \to B]\!]$

Neutral terms

$$\frac{M \in \mathsf{SNe} \quad N_1 \in \mathsf{SN} \quad N_2 \in \mathsf{SN}}{\mathsf{case}\, M\, \mathsf{of}\ \mathsf{inl}\, x \Rightarrow N_1 \mid \mathsf{inr}\, y \Rightarrow N_2 \in \mathsf{SNe}}$$

Normal terms

$$\frac{M \in \mathsf{SN}}{\mathsf{inl}\ M \in \mathsf{SN}} \qquad \frac{M \in \mathsf{SN}}{\mathsf{inr}\ M \in \mathsf{SN}}$$

Strong head reduction

$$\frac{M \in \mathsf{SN} \quad N_2 \in \mathsf{SN}}{\mathsf{case}\, (\mathsf{inl}\ M)\, \mathsf{of}\ \mathsf{inl}\, x \Rightarrow N_1 \mid \mathsf{inr}\, y \Rightarrow N_2 \longrightarrow_{\mathsf{SN}} [M/x]N_1}$$

$$\frac{M \in \mathsf{SN} \quad N_1 \in \mathsf{SN}}{\mathsf{case}\, (\mathsf{inr}\ M)\, \mathsf{of}\ \mathsf{inl}\, x \Rightarrow N_1 \mid \mathsf{inr}\, y \Rightarrow N_2 \longrightarrow_{\mathsf{SN}} [M/x]N_2}$$

$$\frac{M \longrightarrow_{\mathsf{SN}} M'}{\mathsf{case}\, M\, \mathsf{of}\ \mathsf{inl}\, x \Rightarrow N_1 \mid \mathsf{inr}\, y \Rightarrow N_2 \longrightarrow_{\mathsf{SN}} \mathsf{case}\, M'\, \mathsf{of}\ \mathsf{inl}\, x \Rightarrow N_1 \mid \mathsf{inr}\, y \Rightarrow N_2}$$

Figure 6.2: Inductive definition of strongly normalizing terms - extended for case-expressions and injections

□

Corollary 6.4.2. *If* $\Gamma \vdash M : A$ *then* $M \in \mathsf{SN}$.

Proof. Using the fundamental lemma with the identity substitution $\mathsf{id} \in [\![\Gamma]\!]$, we obtain $M \in [\![A]\!]$. By CR1, we know $M \in \mathsf{SN}$. □

6.5 Extension: Disjoint sums

We will now extend our simply-typed lambda-calculus to disjoint sums.

Types	A	$::=$	$\ldots \mid A + B$
Terms	M	$::=$	$\ldots \mid \mathsf{inl}\ M \mid \mathsf{inr}\ M \mid \mathsf{case}\, M\, \mathsf{of}\ \mathsf{inl}\, x \Rightarrow N_1 \mid \mathsf{inr}\, y \Rightarrow N_2$

Let us first extend our definition of SN and SNe (see Fig. 6.2).

Next, we extend our definition of semantic type to disjoint sums. A first attempt might be to define $[\![A + B]\!]$ as follows

Attempt 1

$$[\![A+B]\!] := \{\mathsf{inl}\ M \mid M \in [\![A]\!]\} \cup \{\mathsf{inr}\ M \mid M \in [\![B]\!]\}$$

However, this definition would not satisfy the key property CR3 and hence would fail to be a reducibility candidate. For example, while $\mathsf{inl}\ y$ is in $[\![A+B]\!]$, $(\lambda x.\mathsf{inl}\ x)\ y$ would not be in $[\![A+B]\!]$ despite the fact that $(\lambda x.\mathsf{inl}\ x)\ y \longrightarrow_{\mathsf{SN}} \mathsf{inl}\ y$.

Our definition of $[\![A+B]\!]$ is not closed under the reduction relation $\longrightarrow_{\mathsf{SN}}$. Let $\mathcal{A}$ denote the denotation of $[\![A]\!]$. We then define the closure of $[\![A]\!] = \mathcal{A}$, written as $\overline{\mathcal{A}}$, inductively as follows:

$$\frac{M \in \mathcal{A}}{M \in \overline{\mathcal{A}}} \qquad \frac{M \in \mathsf{SNe}}{M \in \overline{\mathcal{A}}} \qquad \frac{M \in \overline{\mathcal{A}} \qquad N \longrightarrow_{\mathsf{SN}} M}{N \in \overline{\mathcal{A}}}$$

and we define

$$[\![A+B]\!] \quad = \quad \overline{\{\mathsf{inl}\ M \mid M \in [\![A]\!]\} \cup \{\mathsf{inr}\ M \mid M \in [\![B]\!]\}}$$

6.5.1 Semantic type $[\![A+B]\!]$ is a reducibility candidate

We first extend our previous theorem which states that all denotations of types must be in CR.

Theorem 6.5.1. *For all types* C, $[\![C]\!] \in \mathsf{CR}$.

Proof. By induction on the structure of A. We highlight the case for disjoint sums.

Case $C = A + B$.

1. *Show* CR1. Assume that $M \in [\![A+B]\!]$. We consider different subcases and prove by an induction on the closure defining $[\![A+B]\!]$ that $M \in \mathsf{SN}$.

   Subcase: $M \in \{\mathsf{inl}\ N \mid N \in [\![A]\!]\}$. Therefore $M = \mathsf{inl}\ N$. Since $N \in [\![A]\!]$ and by i.h. (CR1), $N \in \mathsf{SN}$. By definition of SN, we have that $\mathsf{inl}\ N \in \mathsf{SN}$.

   Subcase: $M \in \{\mathsf{inr}\ N \mid N \in [\![B]\!]\}$. Therefore $M = \mathsf{inr}\ N$. Since $N \in [\![B]\!]$ and by i.h. (CR1), $N \in \mathsf{SN}$. By definition of SN, we have that $\mathsf{inr}\ N \in \mathsf{SN}$.

   Subcase: $M \in \mathsf{SNe}$. By definition of SN, we conclude that $M \in \mathsf{SN}$.

Subcase: $M \in [\![A+B]\!]$, **if** $M \longrightarrow_{SN} M'$ **and** $M' \in [\![A+B]\!]$.

$M \longrightarrow_{SN} M'$ and $M' \in [\![A+B]\!]$	by assumption
$M' \in SN$	by inner i.h.
$M \in SN$	by reduction $\longrightarrow_{SN}$

2. *Show* CR2.if $M \in SNe$, then $M \in [\![A+B]\!]$
   By definition of the closure, if $M \in SNe$, we have $M \in [\![A+B]\!]$.

3. *Show* CR3. if $M \longrightarrow_{SN} M'$ and $M' \in [\![A+B]\!]$ then $M \in [\![A+B]\!]$.
   By definition of the closure, if $M \longrightarrow_{SN} M'$ and $M' \in [\![A+B]\!]$, we have $M \in [\![A+B]\!]$.

□

6.5.2 Revisiting the fundamental lemma

We can now revisit the fundamental lemma.

Lemma 6.5.2 (Fundamental lemma). *If* $\Gamma \vdash M : A$ *and* $\sigma \in [\![\Gamma]\!]$ *then* $[\sigma]M \in [\![A]\!]$.

Proof. By induction on $\Gamma \vdash M : A$.

Case $\mathcal{D} = \dfrac{\Gamma \vdash M : A+B \qquad \Gamma, x{:}A \vdash N_1 : C \qquad \Gamma, y{:}B \vdash N_2 : C}{\Gamma \vdash \mathsf{case}\, M\, \mathsf{of}\ \mathsf{inl}\, x \Rightarrow N_1 \mid \mathsf{inr}\, y \Rightarrow N_2 : C}$

$\sigma \in [\![\Gamma]\!]$	by assumption
$[\sigma]M \in [\![A+B]\!]$	by i.h.

We consider different subcases and prove by induction on the closure defining $[\![A+B]\!]$, that $[\sigma](\mathsf{case}\, M\, \mathsf{of}\ \mathsf{inl}\, x \Rightarrow M_1 \mid \mathsf{inr}\, y \Rightarrow M_2) \in [\![C]\!]$.

Subcase $[\sigma]M \in \{\mathsf{inl}\ N \mid N \in [\![A]\!]\}$

$[\sigma]M = \mathsf{inl}\ N$ for some $N \in [\![A]\!]$	by assumption
$N \in SN$	by CR1
$\sigma \in [\![\Gamma]\!]$	by assumption
$[\sigma, N/x] \in [\![\Gamma, x : A]\!]$	by definition
$[\sigma, N/x]M_1 \in [\![C]\!]$	by outer i.h.
$y \in [\![B]\!]$	by definition
$[\sigma, y/y] \in [\![\Gamma, y : B]\!]$	by definition

$[\sigma, y/y]M_2 \in [\![C]\!]$ by outer i.h.
$[\sigma, y/y]M_2 \in \mathsf{SN}$ by CR1
$\mathsf{case}\,(\mathsf{inl}\,N)\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow [\sigma, x/x]M_1 \mid \mathsf{inr}\,y \Rightarrow [\sigma, y/y]M_2 \longrightarrow_{\mathsf{SN}} [\sigma, N/x]M_1$ by $\longrightarrow_{\mathsf{SN}}$
$\mathsf{case}\,(\mathsf{inl}\,N)\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow [\sigma, x/x]M_1 \mid \mathsf{inr}\,y \Rightarrow [\sigma, y/y]M_2$
$\quad = [\sigma](\mathsf{case}\,M\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow M_1 \mid \mathsf{inr}\,y \Rightarrow M_2)$ by subst. definition and $[\sigma]M = \mathsf{inl}\,N$
$[\sigma](\mathsf{case}\,M\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow M_1 \mid \mathsf{inr}\,y \Rightarrow M_2) \in [\![C]\!]$ by CR3

Subcase $[\sigma]M \in \{\mathsf{inr}\,N \mid N \in [\![B]\!]\}$

similar to the case above.

Subcase: $[\sigma]M \in \mathsf{SNe}$.
$\sigma \in \Gamma$ by assumption
$x \in [\![A]\!]$, $y \in [\![B]\!]$ by definition
$[\sigma, y/y] \in [\![\Gamma, y : B]\!]$ by definition
$[\sigma, x/x] \in [\![\Gamma, x : A]\!]$ by definition
$[\sigma, x/x]M_1 \in [\![C]\!]$ by outer i.h.
$[\sigma, y/y]M_2 \in [\![C]\!]$ by outer i.h.
$[\sigma, y/y]M_2 \in \mathsf{SN}$ by CR1
$[\sigma, x/x]M_1 \in \mathsf{SN}$ by CR1
$\mathsf{case}\,[\sigma]M\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow [\sigma, x/x]M_1 \mid \mathsf{inr}\,y \Rightarrow [\sigma, y/y]M_2 \in \mathsf{SNe}$ by SNe
$[\sigma](\mathsf{case}\,M\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow M_1 \mid \mathsf{inr}\,y \Rightarrow M_2) \in \mathsf{SNe}$ by substitution def.
$[\sigma](\mathsf{case}\,M\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow M_1 \mid \mathsf{inr}\,y \Rightarrow M_2) \in [\![C]\!]$ by CR2

Subcase: $[\sigma]M \in [\![A + B]\!]$**, if** $[\sigma]M \longrightarrow_{\mathsf{SN}} M'$ **and** $M' \in [\![A + B]\!]$
$[\sigma]M \longrightarrow_{\mathsf{SN}} M'$ and $M' \in [\![A + B]\!]$ by assumption
$\mathsf{case}\,M'\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow [\sigma, x/x]M_1 \mid \mathsf{inr}\,y \Rightarrow [\sigma, y/y]M_2 \in [\![C]\!]$ by inner i.h.
$\mathsf{case}\,[\sigma]M\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow [\sigma, x/x]M_1 \mid \mathsf{inr}\,y \Rightarrow [\sigma, y/y]M_2$
$\quad \longrightarrow_{\mathsf{SN}} \mathsf{case}\,M'\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow [\sigma, x/x]M_1 \mid \mathsf{inr}\,y \Rightarrow [\sigma, y/y]M_2$ by $\longrightarrow_{\mathsf{SN}}$
$[\sigma](\mathsf{case}\,M\,\mathsf{of}\,\mathsf{inl}\,x \Rightarrow M_1 \mid \mathsf{inr}\,y \Rightarrow M_2) \in [\![C]\!]$ by CR3 □

6.6 Extension: Recursion

We now extend our simply-typed lambda-calculus to include natural numbers defined by z and $\mathsf{suc}\,t$ as well as a primitive recursion operator written as $\mathsf{rec}\ M\ \mathsf{with}\ f\ \mathsf{z} \to M_z \mid f\ (\mathsf{suc}\,n) \to M_s$ where M is the argument we recurse over, M_z describes the branch taken if $M = \mathsf{z}$ and M_s describes the branch taken when $M = \mathsf{suc}\,N$ where n will be instantiated with N and $f\ n$ describes the recursive call.

$$\begin{array}{lll} \text{Types} & A & ::= \ \ldots \mid \mathsf{nat} \\ \text{Terms} & t & ::= \ \ldots \mid \mathsf{z} \mid \mathsf{suc}\, t \mid \mathsf{rec}\ t\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \rightarrow t_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \rightarrow t_s \end{array}$$

To clarify, we give the typing rules for the additional constructs.

$$\frac{}{\Gamma \vdash \mathsf{z} : \mathsf{nat}} \qquad \frac{\Gamma \vdash M : \mathsf{nat}}{\Gamma \vdash \mathsf{suc}\, M : \mathsf{nat}}$$

$$\frac{\Gamma \vdash M : \mathsf{nat} \quad \Gamma \vdash M_z : C \quad \Gamma, n : \mathsf{nat},\ \mathsf{f}\, n : C \vdash M_s : C}{\Gamma \vdash \mathsf{rec}\ M\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \rightarrow M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \rightarrow M_s : C}$$

We again extend our definition of SN and SNe.

Neutral terms

$$\frac{M \in \mathsf{SNe} \quad M_z \in \mathsf{SN} \quad M_s \in \mathsf{SN}}{\mathsf{rec}\ M\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \rightarrow M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \rightarrow M_s \in \mathsf{SNe}}$$

Normal terms

$$\frac{}{\mathsf{z} \in \mathsf{SN}} \qquad \frac{M \in \mathsf{SN}}{\mathsf{suc}\, M \in \mathsf{SN}}$$

Strong head reduction

$$\frac{M_s \in SN}{\mathsf{rec}\ \mathsf{z}\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \rightarrow M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \rightarrow M_s \longrightarrow_{\mathsf{SN}} M_z}$$

$$\frac{N \in \mathsf{SN} \quad M_z \in \mathsf{SN} \quad M_s \in SN \quad \mathsf{f}_r = \mathsf{rec}\ N\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \rightarrow M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \rightarrow M_s}{\mathsf{rec}\ (\mathsf{suc}\, N)\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \rightarrow M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \rightarrow M_s \longrightarrow_{\mathsf{SN}} [N/n,\ \mathsf{f}_r/\mathsf{f}\, n]M_s}$$

$$\frac{M \longrightarrow_{\mathsf{SN}} M'}{\mathsf{rec}\ M\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \rightarrow M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \rightarrow M_s \longrightarrow_{\mathsf{SN}} \mathsf{rec}\ M'\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \rightarrow M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \rightarrow M_s}$$

6.7 Extension: Natural numbers

Here we add natural numbers to our language and show how the language remains normalizing.

6.7.1 Semantic type $[\![\mathsf{nat}]\!]$

We define the denotation of nat as follows:

$$[\![\mathsf{nat}]\!] := \overline{\{\mathsf{z}\} \cup \{\mathsf{suc}\, M \mid M \in [\![\mathsf{nat}]\!]\}}$$

6.7.2 Semantic type $[\![\mathsf{nat}]\!]$ is a reducibility candidate

We again extend our previous theorem which states that all denotations of types must be in CR.

Theorem 6.7.1. *For all types* C, $[\![C]\!] \in \mathsf{CR}$.

Proof. By induction on the structure of A. We highlight the case for nat.

Case $C = \mathsf{nat}$

1. *Show* $\mathsf{CR1}$: Assume $M \in \mathsf{nat}$. we consider different subcases and prove by induction on the closure defining nat that $M \in \mathsf{SN}$.

   Subcase $M = \mathsf{z}$. By definition of SN, $\mathsf{z} \in \mathsf{SN}$.

   Subcase $M = \mathsf{suc}\, N$ where $N \in [\![\mathsf{nat}]\!]$. By i.h. ($\mathsf{CR1}$), $N \in \mathsf{SN}$. By definition of SN, $\mathsf{suc}\, N \in \mathsf{SN}$.

   Subcase $M \in \mathsf{SNe}$. By definition of SN, $M \in \mathsf{SN}$.

   Subcase $M \in [\![\mathsf{nat}]\!]$, if $M \longrightarrow_{\mathsf{SN}} M'$ and $M' \in [\![\mathsf{nat}]\!]$.
   $M \longrightarrow_{\mathsf{SN}} M'$ and $M' \in [\![\mathsf{nat}]\!]$ by assumption
   $M' \in \mathsf{SN}$ by inner i.h.
   $M \in \mathsf{SN}$ by reduction $\longrightarrow_{\mathsf{SN}}$

Show $\mathsf{CR2}$: By definition of the closure, $M \in \mathsf{SNe}$, then $M \in [\![\mathsf{nat}]\!]$.

Show $\mathsf{CR3}$: $M \in \mathsf{nat}$, if $M \longrightarrow_{\mathsf{SN}} M'$ and $M' \in \mathsf{nat}$. By definition of the closure, we have that $M \in \mathsf{nat}$. □

6.7.3 Revisiting the fundamental lemma

We can now revisit the fundamental lemma.

Lemma 6.7.2 (Fundamental lemma). *If* $\Gamma \vdash M : A$ *and* $\sigma \in [\![\Gamma]\!]$ *then* $[\sigma]M \in [\![A]\!]$.

Proof. By induction on $\Gamma \vdash M : A$.

Case $\mathcal{D} = \dfrac{}{\Gamma \vdash \mathsf{z} : \mathsf{nat}}$

$\mathsf{z} \in [\![\mathsf{nat}]\!]$ by definition.

Case $\mathcal{D} = \dfrac{\Gamma \vdash M : \mathsf{nat}}{\Gamma \vdash \mathsf{suc}\, M : \mathsf{nat}}$

$\sigma \in [\![\Gamma]\!]$ by assumption

$M \in [\![\mathsf{nat}]\!]$ by i.h.

$\mathsf{suc}\, M \in [\![\mathsf{nat}]\!]$ by definition

Case $\mathcal{D} = \dfrac{\Gamma \vdash M : \mathsf{nat} \qquad \Gamma \vdash M_z : C \qquad \Gamma, n : \mathsf{nat},\ f\ n : C \vdash M_s : C}{\Gamma \vdash \mathsf{rec}\ M\ \mathsf{with}\ f\ \mathsf{z} \rightarrow M_z \mid f\ (\mathsf{suc}\, n) \rightarrow M_s : C}$

$\sigma \in [\![\Gamma]\!]$ by assumption

$[\sigma]M \in [\![\mathsf{nat}]\!]$ by i.h.

We distinguish cases based on $M \in [\![\mathsf{nat}]\!]$ and prove by induction on $M \in [\![\mathsf{nat}]\!]$ that $[\sigma](\mathsf{rec}\ M\ \mathsf{with}\ f\ \mathsf{z} \rightarrow M_z \mid f\ (\mathsf{suc}\, n) \rightarrow M_s) \in [\![C]\!]$.

Subcase $[\sigma]M = \mathsf{z}$.

$n \in [\![\mathsf{nat}]\!],\ f\ n \in [\![C]\!]$ by definition

$[\sigma, n/n, f\ n/f\ n] \in [\![\Gamma, n : \mathsf{nat}, f\ n : C]\!]$ by definition

$[\sigma, n/n, f\ n/f\ n]M_s \in [\![C]\!]$ by outer i.h.

$[\sigma, n/n, f\ n/f\ n]M_s \in \mathsf{SN}$ by CR1

$[\sigma]M_z \in [\![C]\!]$ by outer i.h.

$\mathsf{rec}\ \mathsf{z}\ \mathsf{with}\ f\ \mathsf{z} \rightarrow [\sigma]M_z \mid f\ (\mathsf{suc}\, n) \rightarrow [\sigma, n/n, f\ n/f\ n]M_s \longrightarrow_{\mathsf{SN}} [\sigma]M_z$ by $\longrightarrow_{\mathsf{SN}}$

$\mathsf{rec}\ \mathsf{z}\ \mathsf{with}\ f\ \mathsf{z} \rightarrow [\sigma]M_z \mid f\ (\mathsf{suc}\, n) \rightarrow [\sigma, n/n, f\ n/f\ n]M_s = [\sigma](\mathsf{rec}\ M\ \mathsf{with}\ f\ \mathsf{z} \rightarrow M_z \mid f\ (\mathsf{suc}\, n) \rightarrow M_s)$ by subst. def. and $M = \mathsf{z}$

$[\sigma](\mathsf{rec}\ M\ \mathsf{with}\ f\ \mathsf{z} \rightarrow M_z \mid f\ (\mathsf{suc}\, n) \rightarrow M_s \in [\![C]\!]$ by CR3.

Subcase $[\sigma]M = \mathsf{suc}\, M'$ where $M' \in [\![\mathsf{nat}]\!]$.

$M' \in [\![\mathsf{nat}]\!]$ by assumption

$M' \in \mathsf{SN}$ by CR1

$[\sigma]M_z \in [\![C]\!]$ by outer i.h.

$[\sigma]M_z \in \mathsf{SN}$ by CR1

$[\sigma, n/n, f\ n/f\ n]M_s \in [\![C]\!]$ by outer i.h.

$[\sigma, n/n, f\ n/f\ n]M_s \in \mathsf{SN}$ by CR1

$\mathsf{rec}\ M'\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to [\sigma]M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to [\sigma, n/n, f\ n/f\ n]M_s \in [\![C]\!]$ by inner i.h.

$[\sigma, M'/x,\ \mathsf{rec}\ M'\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to [\sigma]M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to [\sigma, n/n, f\ n/f\ n]M_s/f\ n] \in [\![\Gamma, n : \mathsf{nat}, f\ n : C]\!]$ by definition

$[\sigma, M'/x,\ \mathsf{rec}\ M'\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to [\sigma]M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to [\sigma, n/n, f\ n/f\ n]M_s/f\ n]M_s \in [\![C]\!]$ by outer i.h.

$\mathsf{rec}\ (\mathsf{suc}\, M')\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to [\sigma]M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to [\sigma, n/n, f\ n/f\ n]M_s$
$\quad \longrightarrow_{\mathsf{SN}} [\sigma, M'/x,\ \mathsf{rec}\ M'\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to [\sigma]M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to [\sigma, n/n, f\ n/f\ n]M_s/f\ n]M_s$ by $\longrightarrow_{\mathsf{SN}}$

$[\sigma](\mathsf{rec}\ M\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to M_s) \in [\![C]\!]$ by CR3.

Subcase $[\sigma]M \in \mathsf{SNe}$.

$[\sigma]M_z \in [\![C]\!]$ by outer i.h.

$[\sigma]M_z \in \mathsf{SN}$ by CR1

$[\sigma, n/n, f\ n/f\ n]M_s \in [\![C]\!]$ by outer i.h.

$[\sigma, n/n, f\ n/f\ n]M_s \in \mathsf{SN}$ by CR1

$\mathsf{rec}\ [\sigma]M\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to [\sigma]M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to [\sigma, n/n, f\ n/f\ n]M_s \in \mathsf{SNe}$ by SNe

$[\sigma](\mathsf{rec}\ M\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to M_s) \in \mathsf{SNe}$ by subst. def.

$[\sigma](\mathsf{rec}\ M\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to M_s) \in [\![C]\!]$ by CR2.

Subcase $[\sigma]M \in [\![\mathsf{nat}]\!]$, if $[\sigma]M \longrightarrow_{\mathsf{SN}} M'$ and $M' \in [\![\mathsf{nat}]\!]$.

$[\sigma]M \longrightarrow_{\mathsf{SN}} M'$ and $M' \in [\![\mathsf{nat}]\!]$ by assumption.

$\mathsf{rec}\ M'\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to [\sigma]M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to [\sigma, n/n, f\ n/f\ n]M_s \in [\![C]\!]$ by inner i.h.

$\mathsf{rec}\ [\sigma]M\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to [\sigma]M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to [\sigma, n/n, f\ n/f\ n]M_s$
$\quad \longrightarrow_{\mathsf{SN}} \mathsf{rec}\ M'\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to [\sigma]M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to [\sigma, n/n, f\ n/f\ n]M_s$ by $\longrightarrow_{\mathsf{SN}}$

$[\sigma](\mathsf{rec}\ M\ \mathsf{with}\ \mathsf{f}\ \mathsf{z} \to M_z \mid \mathsf{f}\ (\mathsf{suc}\, n) \to M_s) \in [\![C]\!]$ by CR3.

$\square$

Bibliography

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

[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.