Chapter 1

Introduction

According to Wikipedia, logic is the study of the principles of valid inferences and demonstration. From the breadth of this definition it is immediately clear that logic constitutes an important area in the disciplines of philosophy and mathematics. Logical tools and methods also play an essential role in the design, specification, and verification of computer hardware and software. It is these applications of logic in computer science which will be the focus of this course. In order to gain a proper understanding of logic and its relevance to computer science, we will need to draw heavily on the much older logical traditions in philosophy and mathematics. We will discuss some of the relevant history of logic and pointers to further reading throughout these notes. In this introduction, we give only a brief overview of the contents and approach of this class.

The course is divided into four parts:

I. Proofs as Evidence for Truth

II. Proofs as Programs

III. Proofs as Computations

IV. Proofs as Refutations

Proofs are central in all parts of the course, and give it its constructive nature. In each part, we will exhibit connections between proofs and forms of computations studied in computer science. These connections will take quite different forms, which shows the richness of logic as a foundational discipline at the nexus between philosophy, mathematics, and computer science.

In Part I we establish the basic vocabulary and systematically study propositions and proofs, mostly from a philosophical perspective. The treatment will be rather formal in order to permit an easy transition into computational applications. We will also discuss some properties of the logical systems we develop and strategies for proof search. We aim at a systematic account for the usual

forms of logical expression, providing us with a flexible and thorough foundation for the remainder of the course. Exercises in this section will test basic understanding of logical connectives and how to reason with them.

In Part II we focus on constructive reasoning. This means we consider only proofs that describe algorithms. This turns out to be quite natural in the framework we have established in Part I. In fact, it may be somewhat surprising that many proofs in mathematics today are *not* constructive in this sense. Concretely, we find that for a certain fragment of logic, constructive proofs correspond to functional programs and vice versa. More generally, we can extract functional programs from constructive proofs of their specifications. We often refer to constructive reasoning as *intuitionistic*, while non-constructive reasoning is *classical*. Exercises in this part explore the connections between proofs and programs, and between theorem proving and programming.

In Part III we study a different connection between logic and programs where proofs are the result of computation rather than the starting point as in Part II. This gives rise to the paradigm of *logic programming* where the process of computation is one of systematic proof search. Depending on how we search for proofs, different kinds of algorithms can be described at a very high level of abstraction. Exercises in this part focus on exploiting logic programming to implement various algorithms in concrete languages such as Prolog.

In Part IV we study fragments of logic for which the question whether a proposition is true of false can be effectively decided by an algorithm. Such fragments can be used to specify some aspects of the behavior of software or hardware and then automatically verify them. A key technique here is model-checking that exhaustively explores the truth of a proposition over a finite state space. Model-checking and related methods are routinely used in industry, for example, to support hardware design by detecting design flaws at an early stage in the development cycle. In this application area, the constructive nature of proofs is usually exploited to generate counterexamples which are embedded in refutations of conjectures. Exercises in this part may involve the use of tools for model-checking, or the implementation of decision procedures for simple theories.

There are several related goals for this course. The first is simply that we would like students to gain a good working knowledge of constructive logic and its relation to computation. This includes the translation of informally specified problems to logical language, the ability to recognize correct proofs and construct them. The skills further include writing and inductively proving the correctness of recursive programs.

The second set of goals concerns the transfer of this knowledge to other kinds of reasoning. We will try to illuminate logic and the underlying philosophical and mathematical principles from various points of view. This is important, since there are many different kinds of logics for reasoning in different domains or about different phenomena[1], but there are relatively few underlying

[1] for example: classical, intuitionistic, modal, second-order, temporal, belief, linear, relevance, affirmation, ...

philosophical and mathematical principles. Our second goal is to teach these principles so that students can apply them in different domains where rigorous reasoning is required.

A third set of goals relates to specific, important applications of logic in the practice of computer science. Examples are the design of type systems for programming languages, specification languages, or verification tools for finite-state systems. While we do not aim at teaching the use of particular systems or languages, students should have the basic knowledge to quickly learn them, based on the materials presented in this class.

These learning goals present different challenges for students from different disciplines. Lectures, recitations, exercises, and the study of these notes are all necessary components for reaching them. These notes do not cover all aspects of the material discussed in lecture, but provide a point of reference for definitions, theorems, and motivating examples. Recitations are intended to answer students' questions and practice problem solving skills that are critical for the homework assignments. Exercises are a combination of written homework to be handed at lecture and theorem proving or programming problems to be submitted electronically using the software written in support of the course. A brief introduction to this software is included in these notes, a separate manual is available with the on-line course material.

Chapter 2

Propositions and Proofs

The goal of this chapter is to develop the two principal notions of logic, namely *propositions* and *proofs*. There is no universal agreement about the proper foundations for these notions. One approach, which has been particularly successful for applications in computer science, is to understand the meaning of a proposition by understanding its proofs. In the words of Martin-Löf [ML96, Page 27]:

> *The meaning of a proposition is determined by [. . .] what counts as a verification of it.*

A *verification* may be understood as a certain kind of proof that only examines the constituents of a proposition. This is analyzed in greater detail by Dummett [Dum91] although with less direct connection to computer science. The system of inference rules that arises from this point of view is *natural deduction*, first proposed by Gentzen [Gen35] and studied in depth by Prawitz [Pra65].

In this chapter we apply Martin-Löf's approach, which follows a rich philosophical tradition, to explain the basic propositional connectives. We will see later that universal and existential quantifiers and types such as natural numbers, lists, or trees naturally fit into the same framework.

2.1 Judgments and Propositions

The cornerstone of Martin-Löf's foundation of logic is a clear separation of the notions of judgment and proposition. A *judgment* is something we may know, that is, an object of knowledge. A judgment is *evident* if we in fact know it.

We make a judgment such as "*it is raining*", because we have evidence for it. In everyday life, such evidence is often immediate: we may look out the window and see that it is raining. In logic, we are concerned with situation where the evidence is indirect: we deduce the judgment by making correct inferences from other evident judgments. In other words: a judgment is evident if we have a proof for it.

The most important judgment form in logic is "*A is true*", where A is a proposition. In order to reason correctly, we therefore need a second judgment form "*A is a proposition*". But there are many others that have been studied extensively. For example, "*A is false*", "*A is true at time t*" (from temporal logic), "*A is necessarily true*" (from modal logic), "*program M has type τ*" (from programming languages), etc.

Returning to the first two judgments, let us try to explain the meaning of conjunction. We write *A prop* for the judgment "*A is a proposition*" and *A true* for the judgment "*A is true*" (presupposing that *A prop*). Given propositions A and B, we want to form the compound proposition "*A and B*", written more formally as $A \wedge B$. We express this in the following inference rule:

$$\frac{A\ \mathit{prop} \quad B\ \mathit{prop}}{A \wedge B\ \mathit{prop}}\ \wedge F$$

This rule allows us to conclude that $A \wedge B$ *prop* if we already know that *A prop* and *B prop*. In this inference rule, A and B are *schematic variables*, and $\wedge F$ is the name of the rule (which is short for "conjunction formation"). The general form of an inference rule is

$$\frac{J_1\ \ldots\ J_n}{J}\ \mathit{name}$$

where the judgments $J_1, \ldots, J_n$ are called the *premises*, the judgment J is called the *conclusion*. In general, we will use letters J to stand for judgments, while A, B, and C are reserved for propositions.

Once the rule of conjunction formation ($\wedge F$) has been specified, we know that $A \wedge B$ is a proposition, if A and B are. But we have not yet specified what it *means*, that is, what counts as a verification of $A \wedge B$. This is accomplished by the following inference rule:

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

Here the name $\wedge I$ stands for "conjunction introduction", since the conjunction is introduced in the conclusion. We take this as specifying the meaning of $A \wedge B$ completely. So what can be deduce if we know that $A \wedge B$ is true? By the above rule, to have a verification for $A \wedge B$ means to have verifications for A and B. Hence the following two rules are justified:

$$\frac{A \wedge B\ \mathit{true}}{A\ \mathit{true}}\ \wedge E_L \qquad\qquad \frac{A \wedge B\ \mathit{true}}{B\ \mathit{true}}\ \wedge E_R$$

The name $\wedge E_L$ stands for "left conjunction elimination", since the conjunction in the premise has been eliminated in the conclusion. Similarly $\wedge E_R$ stands for "right conjunction elimination".

We will later see what precisely is required in order to guarantee that the formation, introduction, and elimination rules for a connective fit together correctly. For now, we will informally argue the correctness of the elimination rules.

As a second example we consider the proposition "*truth*" written as $\top$.

$$\frac{}{\top\ \mathit{prop}}\ \top F$$

Truth should always be true, which means its introduction rule has no premises.

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

Consequently, we have no information if we know $\top$ *true*, so there is no elimination rule.

A conjunction of two propositions is characterized by one introduction rule with two premises, and two corresponding elimination rules. We may think of truth as a conjunction of zero propositions. By analogy it should then have one introduction rule with zero premises, and zero corresponding elimination rules. This is precisely what we wrote out above.

2.2 Hypothetical Judgments

Consider the following derivation, for some arbitrary propositions A, B, and C:

$$\dfrac{\dfrac{A \wedge (B \wedge C)\ \mathit{true}}{B \wedge C\ \mathit{true}}\ \wedge E_R}{B\ \mathit{true}}\ \wedge E_L$$

Have we actually proved anything here? At first glance it seems that cannot be the case: B is an arbitrary proposition; clearly we should not be able to prove that it is true. Upon closer inspection we see that all inferences are correct, but the first judgment $A \wedge (B \wedge C)$ *true* has not been justified. We can extract the following knowledge:

> *From the assumption that $A \wedge (B \wedge C)$ is true, we deduce that B must be true.*

This is an example of a *hypothetical judgment*, and the figure above is an *hypothetical derivation*. In general, we may have more than one assumption, so a hypothetical derivation has the form

$$\begin{array}{c} J_1 \quad \cdots \quad J_n \\ \vdots \\ J \end{array}$$

where the judgments $J_1, \ldots, J_n$ are unproven assumptions, and the judgment J is the conclusion. Note that we can always substitute a proof for any hypothesis J_i to eliminate the assumption. We call this the *substitution principle* for hypotheses.

Many mistakes in reasoning arise because dependencies on some hidden assumptions are ignored. When we need to be explicit, we write $J_1, \ldots, J_n \vdash J$ for the hypothetical judgment which is established by the hypothetical derivation above. We may refer to $J_1, \ldots, J_n$ as the antecedents and J as the succedent of the hypothetical judgment.

One has to keep in mind that hypotheses may be used more than once, or not at all. For example, for arbitrary propositions A and B,

$$\dfrac{\dfrac{A \wedge B \ \mathit{true}}{B \ \mathit{true}} \wedge E_R \quad \dfrac{A \wedge B \ \mathit{true}}{A \ \mathit{true}} \wedge E_L}{B \wedge A \ \mathit{true}} \wedge I$$

can be seen a hypothetical derivation of $A \wedge B \ \mathit{true} \vdash B \wedge A \ \mathit{true}$.

With hypothetical judgments, we can now explain the meaning of implication "*A implies B*" or "*if A then B*" (more formally: $A \supset B$). First the formation rule:

$$\dfrac{A \ \mathit{prop} \quad B \ \mathit{prop}}{A \supset B \ \mathit{prop}} \supset F$$

Next, the introduction rule: $A \supset B$ is true, if B is true under the assumption that A is true.

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

The tricky part of this rule is the label u. If we omit this annotation, the rule would read

$$\dfrac{\begin{array}{c} A \ \mathit{true} \\ \vdots \\ B \ \mathit{true} \end{array}}{A \supset B \ \mathit{true}} \supset I$$

which would be incorrect: it looks like a derivation of $A \supset B \ \mathit{true}$ from the hypothesis $A \ \mathit{true}$. But the assumption $A \ \mathit{true}$ is introduced in the process of proving $A \supset B \ \mathit{true}$; the conclusion should not depend on it! Therefore we label uses of the assumption with a new name u, and the corresponding inference which introduced this assumption into the derivation with the same label u.

As a concrete example, consider the following proof of $A \supset (B \supset (A \wedge B))$.

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

Note that this derivation is not hypothetical (it does not depend on any assumptions). The assumption $A\ true$ labeled u is discharged in the last inference, and the assumption $B\ true$ labeled w is discharged in the second-to-last inference. It is critical that a discharged hypothesis is no longer available for reasoning, and that all labels introduced in a derivation are distinct.

Finally, we consider what the elimination rule for implication should say. By the only introduction rule, having a proof of $A \supset B\ true$ means that we have a hypothetical proof of $B\ true$ from $A\ true$. By the substitution principle, if we also have a proof of $A\ true$ then we get a proof of $B\ true$.

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

This completes the rules concerning implication.

With the rules so far, we can write out proofs of simple properties concerning conjunction and implication. The first expresses that conjunction is commutative—intuitively, an obvious property.

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

When we construct such a derivation, we generally proceed by a combination of bottom-up and top-down reasoning. The next example is a distributivity law, allowing us to move implications over conjunctions. This time, we show the partial proofs in each step. Of course, other sequences of steps in proof constructions are also possible.

$$\begin{array}{c}\vdots\\ (A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))\ true\end{array}$$

First, we use the implication introduction rule bottom-up.

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

Next, we use the conjunction introduction rule bottom-up.

$$\dfrac{\dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u \quad \dfrac{}{A \supset (B \wedge C)\ true}\,u}{\dfrac{\begin{array}{cc}\vdots & \vdots \\ A \supset B\ true & A \supset C\ true\end{array}}{(A \supset B) \wedge (A \supset C)\ true}\,\wedge I}}{(A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))\ true}\,\supset I^u$$

We now pursue the left branch, again using implication introduction bottom-up.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u \quad \dfrac{}{A\ true}\,w}{\vdots}}{\dfrac{B\ true}{A \supset B\ true}\,\supset I^w} \quad \dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u}{\vdots}}{\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u \quad \dfrac{}{A\ true}\,w}{\vdots}}{\dfrac{B\ true}{A \supset B\ true}\,\supset I^w} \quad \dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u}{\vdots}}{\dfrac{\begin{array}{cc} & \vdots \\ A \supset B\ true & A \supset C\ true\end{array}}{(A \supset B) \wedge (A \supset C)\ true}\,\wedge I}}{(A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))\ true}\,\supset I^u$$

Note that the hypothesis $A\ true$ is available only in the left branch, but not in the right one: it is discharged at the inference $\supset I^w$. We now switch to top-down reasoning, taking advantage of implication elimination.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u \quad \dfrac{}{A\ true}\,w}{B \wedge C\ true}\,\supset E}{\vdots}}{\dfrac{B\ true}{A \supset B\ true}\,\supset I^w} \quad \dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u}{\vdots}}{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u \quad \dfrac{}{A\ true}\,w}{B \wedge C\ true}\,\supset E}{\vdots}}{\dfrac{B\ true}{A \supset B\ true}\,\supset I^w} \quad \dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u}{\vdots}}{\dfrac{\begin{array}{cc} & \vdots \\ A \supset B\ true & A \supset C\ true\end{array}}{(A \supset B) \wedge (A \supset C)\ true}\,\wedge I}}{(A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))\ true}\,\supset I^u$$

Now we can close the gap in the left-hand side by conjunction elimination.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u \quad \dfrac{}{A\ true}\,w}{B \wedge C\ true}\,\supset E}{B\ true}\,\wedge E_L}{\vdots}}{\dfrac{B\ true}{A \supset B\ true}\,\supset I^w} \quad \dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u}{\vdots}}{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u \quad \dfrac{}{A\ true}\,w}{B \wedge C\ true}\,\supset E}{B\ true}\,\wedge E_L}{\vdots}}{\dfrac{B\ true}{A \supset B\ true}\,\supset I^w} \quad \dfrac{\dfrac{}{A \supset (B \wedge C)\ true}\,u}{\vdots}}{\dfrac{\begin{array}{cc} & \vdots \\ A \supset B\ true & A \supset C\ true\end{array}}{(A \supset B) \wedge (A \supset C)\ true}\,\wedge I}}{(A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))\ true}\,\supset I^u$$

The right premise of the conjunction introduction can be filled in analogously. We skip the intermediate steps and only show the final derivation.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{A \supset (B \wedge C)\ \mathit{true}}\,u \quad \dfrac{}{A\ \mathit{true}}\,w}{B \wedge C\ \mathit{true}}\supset E}{B\ \mathit{true}}\wedge E_L}{A \supset B\ \mathit{true}}\supset I^w \quad \dfrac{\dfrac{\dfrac{\dfrac{}{A \supset (B \wedge C)\ \mathit{true}}\,u \quad \dfrac{}{A\ \mathit{true}}\,v}{B \wedge C\ \mathit{true}}\supset E}{C\ \mathit{true}}\wedge E_R}{A \supset C\ \mathit{true}}\supset I^v}{(A \supset B) \wedge (A \supset C)\ \mathit{true}}\wedge I}{(A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))\ \mathit{true}}\supset I^u$$

2.3 Disjunction and Falsehood

So far we have explained the meaning of conjunction, truth, and implication. The disjunction "*A or B*" (written as $A \vee B$) is more difficult, but does not require any new judgment forms.

$$\frac{A\ \mathit{prop} \quad B\ \mathit{prop}}{A \vee B\ \mathit{prop}}\vee F$$

Disjunction is characterized by two introduction rules: $A \vee B$ is true, if either A or B is true.

$$\frac{A\ \mathit{true}}{A \vee B\ \mathit{true}}\vee I_L \qquad \frac{B\ \mathit{true}}{A \vee B\ \mathit{true}}\vee I_R$$

Now it would be incorrect to have an elimination rule such as

$$\frac{A \vee B\ \mathit{true}}{A\ \mathit{true}}\vee E_L?$$

because even if we know that $A \vee B$ is true, we do not know whether the disjunct A or the disjunct B is true. Concretely, with such a rule we could derive the truth of *every* proposition A as follows:

$$\dfrac{\dfrac{\dfrac{}{\top\ \mathit{true}}\top I}{A \vee \top\ \mathit{true}}\vee I_R}{A\ \mathit{true}}\vee E_L?$$

Thus we take a different approach. If we know that $A \vee B$ is true, we must consider two cases: A *true* and B *true*. If we can prove a conclusion C *true* in both cases, then C must be true! Written as an inference rule:

$$\dfrac{A \vee B\ \mathit{true} \qquad \begin{array}{c}\dfrac{}{A\ \mathit{true}}\,u\\ \vdots\\ C\ \mathit{true}\end{array} \qquad \begin{array}{c}\dfrac{}{B\ \mathit{true}}\,w\\ \vdots\\ C\ \mathit{true}\end{array}}{C\ \mathit{true}}\vee E^{u,w}$$

Note that we use once again the mechanism of hypothetical judgments. In the proof of the second premise we may use the assumption $A\ true$ labeled u, in the proof of the third premise we may use the assumption $B\ true$ labeled w. Both are discharged at the disjunction elimination rule.

Let us justify the conclusion of this rule more explicitly. By the first premise we know $A \vee B\ true$. The premises of the two possible introduction rules are $A\ true$ and $B\ true$. In case $A\ true$ we conclude $C\ true$ by the substitution principle and the second premise: we substitute the proof of $A\ true$ for any use of the assumption labeled u in the hypothetical derivation. The case for $B\ true$ is symmetric, using the hypothetical derivation in the third premise.

Because of the complex nature of the elimination rule, reasoning with disjunction is more difficult than with implication and conjunction. As a simple example, we prove the commutativity of disjunction.

$$\begin{array}{c} \vdots \\ (A \vee B) \supset (B \vee A)\ true \end{array}$$

We begin with an implication introduction.

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

At this point we cannot use either of the two disjunction introduction rules. The problem is that neither B nor A follow from our assumption $A \vee B$! So first we need to distinguish the two cases via the rule of disjunction elimination.

$$\dfrac{\dfrac{}{A \vee B\ true}\,u \qquad \begin{array}{c} \dfrac{}{A\ true}\,v \\ \vdots \\ B \vee A\ true \end{array} \qquad \begin{array}{c} \dfrac{}{B\ true}\,w \\ \vdots \\ B \vee A\ true \end{array}}{\dfrac{B \vee A\ true}{(A \vee B) \supset (B \vee A)\ true}\,\supset I^u}\,\vee E^{v,w}$$

The assumption labeled u is still available for each of the two proof obligations, but we have omitted it, since it is no longer needed.

Now each gap can be filled in directly by the two disjunction introduction rules.

$$\dfrac{\dfrac{}{A \vee B\ true}\,u \qquad \dfrac{\dfrac{}{A\ true}\,v}{B \vee A\ true}\,\vee I_R \qquad \dfrac{\dfrac{}{B\ true}\,w}{B \vee A\ true}\,\vee I_L}{\dfrac{B \vee A\ true}{(A \vee B) \supset (B \vee A)\ true}\,\supset I^u}\,\vee E^{v,w}$$

This concludes the discussion of disjunction. Falsehood (written as $\bot$, sometimes called absurdity) is a proposition that should have no proof! Therefore there are no introduction rules, although we of course have the standard formation rule.

$$\frac{}{\bot\ \mathit{prop}}\ \bot F$$

Since there cannot be a proof of $\bot$ *true*, it is sound to conclude the truth of any arbitrary proposition if we know $\bot$ *true*. This justifies the elimination rule

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

We can also think of falsehood as a disjunction between zero alternatives. By analogy with the binary disjunction, we therefore have zero introduction rules, and an elimination rule in which we have to consider zero cases. This is precisely the $\bot E$ rule above.

From this is might seem that falsehood it useless: we can never prove it. This is correct, except that we might reason from contradictory hypotheses! We will see some examples when we discuss negation, since we may think of the proposition "*not* A" (written $\neg A$) as $A \supset \bot$. In other words, $\neg A$ is true precisely if the assumption A *true* is contradictory because we could derive $\bot$ *true*.

2.4 Natural Deduction

The judgments, propositions, and inference rules we have defined so far collectively form a system of *natural deduction*. It is a minor variant of a system introduced by Gentzen [Gen35] and studied in depth by Prawitz [Pra65]. One of Gentzen's main motivations was to devise rules that model mathematical reasoning as directly as possible, although clearly in much more detail than in a typical mathematical argument.

The specific interpretation of the truth judgment underlying these rules is *intuitionistic* or *constructive*. This differs from the *classical* or *Boolean* interpretation of truth. For example, classical logic accepts the proposition $A \vee (A \supset B)$ as true for arbitrary A and B, although in the system we have presented so far this would have no proof. Classical logic is based on the principle that every proposition must be true or false. If we distinguish these cases we see that $A \vee (A \supset B)$ should be accepted, because in case that A is true, the left disjunct holds; in case A is false, the right disjunct holds. In contrast, intuitionistic logic is based on explicit evidence, and evidence for a disjunction requires evidence for one of the disjuncts. We will return to classical logic and its relationship to intuitionistic logic later; for now our reasoning remains intuitionistic since, as we will see, it has a direct connection to functional computation, which classical logic lacks.

We summarize the rules of inference for the truth judgment introduced so far in Figure 2.1. We omit the straightforward formation rules.

Introduction Rules	**Elimination Rules**
$\dfrac{A\ true \quad B\ true}{A \wedge B\ true}\ \wedge I$	$\dfrac{A \wedge B\ true}{A\ true}\ \wedge E_L \quad \dfrac{A \wedge B\ true}{B\ true}\ \wedge E_R$
$\dfrac{}{\top\ true}\ \top I$	*no* $\top E$ *rule*
$\dfrac{\begin{array}{c}\overline{A\ true}\ u \\ \vdots \\ B\ true\end{array}}{A \supset B\ true}\ \supset I^u$	$\dfrac{A \supset B\ true \quad A\ true}{B\ true}\ \supset E$
$\dfrac{A\ true}{A \vee B\ true}\ \vee I_L \quad \dfrac{B\ true}{A \vee B\ true}\ \vee I_R$	$\dfrac{\begin{array}{ccc} & \overline{A\ true}\ u & \overline{B\ true}\ w \\ A \vee B\ true & \vdots & \vdots \\ & C\ true & C\ true\end{array}}{C\ true}\ \vee E^{u,w}$
no $\bot I$ *rule*	$\dfrac{\bot\ true}{C\ true}\ \bot E$

Figure 2.1: Rules for intuitionistic natural deduction

Bibliography

[Dum91] Michael Dummett. *The Logical Basis of Metaphysics*. Harvard University Press, Cambridge, Massachusetts, 1991. The William James Lectures, 1976.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.

[ML96] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[Pra65] Dag Prawitz. *Natural Deduction*. Almquist & Wiksell, Stockholm, 1965.

2.5 Harmony

In the verificationist definition of the logical connectives via their introduction rules we have briefly justified the elimination rules. In this section we study the balance between introduction and elimination rules more closely. In order to show that the two are in harmony we establish two properties: *local soundness* and *local completeness*.

Local soundness shows that the elimination rules are not too strong: no matter how we apply elimination rules to the result of an introduction we cannot gain any new information. We demonstrate this by showing that we can find a more direct proof of the conclusion of the elimination which does not first introduce and then eliminate the connective in question. This is witnessed by a *local reduction* of the given introduction and the subsequent elimination.

Local completeness shows that the elimination rules are not too weak: there is always a way to apply elimination rules so that we can reconstitute a proof of the original proposition from the results by applying introduction rules. This is witnessed by a *local expansion* of an arbitrary given derivation into some eliminations followed by some introductions.

Connectives whose introduction and elimination rules are in harmony in the sense that they are locally sound and complete are properly defined from the verificationist perspective. If not, the proposed connective should be viewed with suspicion. Another criterion we would like to apply uniformly is that both introduction and elimination rules are *pure*: the may refer and employ different judgments and judgment forms, but they may not refer to other propositions which could create a dangerous dependency of the various connectives on each other. As we present correct definitions we will occasionally also give some counterexamples to illustrate the consequences of violating the principles behind the patterns of valid inference.

In the discussion of each individual connective below we use the notation

$$\begin{array}{c} \mathcal{D} \\ A\ \mathit{true} \end{array} \Longrightarrow_R \begin{array}{c} \mathcal{D}' \\ A\ \mathit{true} \end{array}$$

for the local reduction of a deduction $\mathcal{D}$ to another deduction $\mathcal{D}'$ of the same judgment A *true*. In fact, $\Longrightarrow_R$ can itself be a higher level judgment relating two proofs, $\mathcal{D}$ and $\mathcal{D}'$, although we will not directly exploit this point of view. Similarly,

$$\begin{array}{c} \mathcal{D} \\ A\ \mathit{true} \end{array} \Longrightarrow_E \begin{array}{c} \mathcal{D}' \\ A\ \mathit{true} \end{array}$$

is the notation of the local expansion of $\mathcal{D}$ to $\mathcal{D}'$.

Conjunction. We start with local soundness. Since there are two elimination rules and one introduction, it turns out we have two cases to consider. In either

case, we can easily reduce.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A\ true\end{array}\quad\begin{array}{c}\mathcal{E}\\ B\ true\end{array}}{A \wedge B\ true}\wedge I}{A\ true}\wedge E_L \quad\Longrightarrow_R\quad \begin{array}{c}\mathcal{D}\\ A\ true\end{array}$$

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A\ true\end{array}\quad\begin{array}{c}\mathcal{E}\\ B\ true\end{array}}{A \wedge B\ true}\wedge I}{B\ true}\wedge E_R \quad\Longrightarrow_R\quad \begin{array}{c}\mathcal{E}\\ B\ true\end{array}$$

Local completeness requires us to apply eliminations to an arbitrary proof of $A \wedge B\ true$ in such a way that we can reconstitute a proof of $A \wedge B$ from the results.

$$\begin{array}{c}\mathcal{D}\\ A \wedge B\ true\end{array} \quad\Longrightarrow_E\quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\ true\end{array}}{A\ true}\wedge E_L \quad \dfrac{\begin{array}{c}\mathcal{D}\\ A \wedge B\ true\end{array}}{B\ true}\wedge E_R}{A \wedge B\ true}\wedge I$$

As an example where local completeness might fail, consider the case where we "forget" the right elimination rule for conjunction. The remaining rule is still locally sound, but not locally complete because we cannot extract a proof of B from the assumption $A \wedge B$. Now, for example, we cannot prove $(A \wedge B) \supset (B \wedge A)$ even though this should clearly be true.

Substitution Principle. We need the defining property for hypothetical judgments before we can discuss implication. Intuitively, we can always substitute a deduction of $A\ true$ for any use of a hypothesis $A\ true$. In order to avoid ambiguity, we make sure assumptions are labelled and we substitute for all uses of an assumption with a given label. Note that we can only substitute for assumptions that are not discharged in the subproof we are considering. The substitution principle then reads as follows:

If

$$\begin{array}{c}\dfrac{}{A\ true}\,u\\ \mathcal{E}\\ B\ true\end{array}$$

is a hypothetical proof of $B\ true$ under the undischarged hypothesis $A\ true$ labelled u, and

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

is a proof of $A\ true$ then

$$\dfrac{\begin{array}{c}\mathcal{D}\\ A\ true\end{array}}{\begin{array}{c}\mathcal{E}\\ B\ true\end{array}}\ u$$

is our notation for substituting $\mathcal{D}$ for all uses of the hypothesis labelled u in $\mathcal{E}$. This deduction, also sometime written as $[\mathcal{D}/u]\mathcal{E}$ no longer depends on u.

Implication. To witness local soundness, we reduce an implication introduction followed by an elimination using the substitution operation.

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

The conditions on the substitution operation is satisfied, because u is introduced at the $\supset I^u$ inference and therefore not discharged in $\mathcal{E}$.

Local completeness is witnessed by the following expansion.

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

Here u must be chosen fresh: it only labels the new hypothesis $A\ true$ which is used only once.

Disjunction. For disjunction we also employ the substitution principle because the two cases we consider in the elimination rule introduce hypotheses. Also, in order to show local soundness we have two possibilities for the introduction rule, in both situations followed by the only elimination rule.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ A\ true\end{array}}{A \vee B\ true}\vee I_L \qquad \begin{array}{c}\overline{A\ true}\ u\\ \mathcal{E}\\ C\ true\end{array} \qquad \begin{array}{c}\overline{B\ true}\ w\\ \mathcal{F}\\ C\ true\end{array}}{C\ true}\vee E^{u,w} \quad \Longrightarrow_R \quad \dfrac{\begin{array}{c}\mathcal{D}\\ A\ true\end{array}}{\begin{array}{c}\mathcal{E}\\ C\ true\end{array}}\ u$$

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ B\ true\end{array}}{A \vee B\ true}\vee I_R \qquad \begin{array}{c}\overline{A\ true}\ u\\ \mathcal{E}\\ C\ true\end{array} \qquad \begin{array}{c}\overline{B\ true}\ w\\ \mathcal{F}\\ C\ true\end{array}}{C\ true}\vee E^{u,w} \quad \Longrightarrow_R \quad \dfrac{\begin{array}{c}\mathcal{D}\\ B\ true\end{array}}{\begin{array}{c}\mathcal{F}\\ C\ true\end{array}}\ w$$

An example of a rule that would not be locally sound is

$$\frac{A \vee B \ true}{A \ true} \vee E_L?$$

and, indeed, we would not be able to reduce

$$\dfrac{\dfrac{A \vee B \ true}{B \ true} \vee I_R}{A \ true} \vee E_L?$$

In fact we can now derive a contradiction from no assumption, which means the whole system is incorrect.

$$\dfrac{\dfrac{\dfrac{}{\top \ true} \top I}{\bot \vee \top \ true} \vee I_R}{\bot \ true} \vee E_L?$$

Local completeness of disjunction distinguishes cases on the known $A \vee B \ true$, using $A \vee B \ true$ as the conclusion.

$$\begin{array}{c} \mathcal{D} \\ A \vee B \ true \end{array} \Longrightarrow_L \dfrac{\begin{array}{ccc} \mathcal{D} & \dfrac{\dfrac{}{A \ true} u}{A \vee B \ true} \vee I_L & \dfrac{\dfrac{}{B \ true} w}{A \vee B \ true} \vee I_R \end{array}}{A \vee B \ true} \vee E^{u,w}$$

Visually, this looks somewhat different from the local expansions for conjunction or implication. It looks like the elimination rule is applied last, rather than first. Mostly, this is due to notation: the above represents the step from using the knowledge of $A \vee B \ true$ and eliminating it to obtain the hypotheses $A \ true$ and $B \ true$ in the two cases.

Truth. The local constant $\top$ has only an introduction rule, but no elimination rule. Consequently, there are no cases to check for local soundness: any introduction followed by any elimination can be reduced.

However, local completeness still yields a local expansion: Any proof of $\top \ true$ can be trivially converted to one by $\top I$.

$$\begin{array}{c} \mathcal{D} \\ \top \ true \end{array} \Longrightarrow_E \dfrac{}{\top \ true} \top I$$

Falsehood. As for truth, there is no local reduction because local soundness is trivially satisfied since we have no introduction rule.

Local completeness is slightly tricky. Literally, we have to show that there is a way to apply an elimination rule to any proof of $\bot \ true$ so that we can reintroduce a proof of $\bot \ true$ from the result. However, there will be zero cases

to consider, so we apply no introductions. Nevertheless, the following is the right local expansion.

$$\begin{array}{c}\mathcal{D}\\ \bot\ true\end{array} \quad \Longrightarrow_L \quad \dfrac{\begin{array}{c}\mathcal{D}\\ \bot\ true\end{array}}{\bot\ true}\ \bot E$$

Reasoning about situation when falsehood is true may seem vacuous, but is common in practice because it corresponds to reaching a contradiction. In intuitionistic reasoning, this occurs when we prove $A \supset \bot$ which is often abbreviated as $\neg A$. In classical reasoning it is even more frequent, due to the rule of proof by contradiction.

2.6 Verifications

The verificationist point of view on the meaning of a proposition is that it is determined by its *verifications*. Intuitively, a verification should be a proof that only analyzes the constituents of a propositions. This restriction of the space of all possible proofs is necessary so that the definition is well-founded. For example, if in order to understand the meaning of A, we would have to understand ther meaning of $B \supset A$ and B, the whole program of understanding the meaning of the connectives by their proofs is in jeopardy because B could be a proposition containing, say, A. But the meaning of A would then in turn depend on the meaning of A, creating a vicious cycle.

In this section we will make the structure of verifications more explicit. We write $A\uparrow$ for the judgment "*A has a verification*". Naturally, this should mean that A is true, and that the evidence for that has a special form. Eventually we will also establish the converse: if A is true than A has a verification.

Conjunction is easy to understand. A verification of $A \wedge B$ should consist of a verification of A and a verification of B.

$$\frac{A\uparrow \quad B\uparrow}{A \wedge B\uparrow}\ \wedge I$$

We reuse here the names of the introduction rule, because this rule is strictly analogous to the introduction rule for the truth of a conjunction.

Implication, however, introduces a new hypothesis which is not explicitly justified by an introduction rule but just a new label. For example, in the proof

$$\dfrac{\dfrac{\dfrac{}{A \wedge B\ true}\ u}{A\ true}\ \wedge E_L}{(A \wedge B) \supset A\ true}\ \supset I^u$$

the conjunction $A \wedge B$ is not justified by an introduction.

The informal discussion of proof search strategies earlier, namely to use introduction rules from the bottom up and elimination rules from the top down

contains the answer. We introduce a second judgment, $A\downarrow$ which means "*A may be used*". $A\downarrow$ should be the case when either A *true* is a hypothesis, or A is deduced from a hypothesis via elimination rules. Our local soundness arguments provide some evidence that we cannot deduce anything incorrect in this manner.

We now go through the connectives in turn, defining verifications and uses.

Conjunction. In summary of the discussion above, we obtain:

$$\frac{A\uparrow \quad B\uparrow}{A \wedge B\uparrow}\ \wedge I \qquad \frac{A \wedge B\downarrow}{A\downarrow}\ \wedge E_L \qquad \frac{A \wedge B\downarrow}{B\downarrow}\ \wedge E_R$$

The left elimination rule can be read as: "*If we can use $A \wedge B$ we can use A*", and similarly for the right elimination rule.

Implication. The introduction rule creates a new hypothesis, which we may use in a proof. The assumption is therefore of the judgment $A\downarrow$

$$\dfrac{\dfrac{\overline{A\downarrow}\ u}{\vdots}}{A \supset B\uparrow}\ \supset^u$$

In order to use an implication $A \supset B$ we require a verification of A. Just requiring that A may be used would be too weak, as can be seen when trying to prove $((A \supset A) \supset B) \supset B\uparrow$. It should also be clear from the fact that we are not eliminating a connective from A.

$$\frac{A \supset B\downarrow \quad A\uparrow}{B\downarrow}\ \supset E$$

Disjunction. The verifications of a disjunction immediately follow from their introduction rules.

$$\frac{A\uparrow}{A \vee B\uparrow}\ \vee I_L \qquad \frac{B\uparrow}{A \vee B\uparrow}\ \vee I_R$$

A disjunction is used in a proof by cases, called here $\vee E$. This introduces two new hypotheses, and each of them may be used in the corresponding subproof. Whenever we set up a hypothetical judgment we are trying to find a verification of the conclusion, possibly with uses of hypotheses. So the conclusion of $\vee E$ should be a verification.

$$\frac{A \vee B\downarrow \quad \begin{array}{c}\overline{A\downarrow}\ u\\ \vdots\\ C\uparrow\end{array} \quad \begin{array}{c}\overline{B\downarrow}\ w\\ \vdots\\ C\uparrow\end{array}}{C\uparrow}\ \vee E^{u,w}$$

Truth. The only verification of truth is the trival one.

$$\frac{}{\top\uparrow}\ \top I$$

A hypothesis $\top\downarrow$ cannot be used because there is no elimination rule for $\top$.

Falsehood. There is no verification of falsehood because we have no introduction rule.

We can use falsehood, signifying a contradiction from our current hypotheses, to verify any conclusion. This is the zero-ary case of a disjunction.

$$\frac{\bot\downarrow}{C\uparrow}\ \bot E$$

Atomic propositions. How to we construct a verification of an atomic proposition P? We cannot break down the structure of P because there is none, so we can only proceed if we already know P is true. This can only come from a hypothesis, so we have a rule that lets us use the knowledge of an atomic proposition to construct a verification.

$$\frac{P\downarrow}{P\uparrow}\ \downarrow\uparrow$$

This rule has a special status in that it represents a change in judgments but is not tied to a particular local connective. We call this a *judgmental rule* in order to distinguish it from the usual introduction and elimination rules that characterize the connectives.

Global soundness. Local soundness is an intrinsic property of each connective, asserting that the elimination rules for it are not too strong given the introduction rules. Global soundness is its counterpart for the whole system of inference rules. It says that if an arbitrary proposition A has a verification than we may use A without gaining any information. That is, for arbitrary propositions A and C:

$$\begin{array}{c} A\downarrow \\ \vdots \\ \textit{If } A\uparrow \textit{ and } C\uparrow \textit{ then } C\uparrow. \end{array}$$

We would want to prove this using a substitution principle, except that the judgment $A\uparrow$ and $A\downarrow$ do not match. In the end, the arguments for local soundness will help use carry out this proof later in this course.

Global completeness. Local completeness is also an intrinsic property of each connective. It asserts that the elimination rules are not too weak, given the introduction rule. Global completeness is its counterpart for the whole system of inference rules. It says that if we may use A than we can construct from this a verification of A. That is, for arbitrary propositions A:

$$\begin{array}{l} A\uparrow \\ \vdots \\ A\downarrow. \end{array}$$

Global completeness follows from local completeness rather directly by induction on the structure of A.

Global soundness and completeness are properties of whole deductive systems. Their proof must be carried out in a mathematical *metalanguage* which makes them a bit different than the formal proofs that we have done so far within natural deduction. Of course, we would like them to be correct as well, which means they should follow the same principles of valid inference that we have laid out so far.

There are two further properties we would like, relating truth, verifications, and uses. The first is that if A has a verification then A is true. Once we add that if A may be used then A is true, this is rather evident since we have just specialized the introduction and elimination rules, except for the judgmental rule $\downarrow\uparrow$. But under the interpretation of verification and use as truth, this inference becomes redundant.

Significantly more difficult is the property that if A is true then A has a verification. Since we justified the meaning of the connectives from their verifications, a failure of this property would be devastating to the verificationist program. Fortunately it holds and can be proved by exhibiting a process of *proof normalization* that takes an arbitrary proof of A true and constructs a verification of A.

All these properties in concert show that our rules are well constructed, locally as well as globally. Experience with many other logical systems indicates that this is not an isolated phenomenon: we can employ the verificationist point of view to give coherent sets of rules not just for constructive logic, but for classical logic, temporal logic, spatial logic, modal logic, and many other logics that area of interest in computer science. Taken together, these constitute strong evidence that separating judgments from propositions and taking a verificationist point of view in the definition of the logical connectives is indeed a proper and useful foundation for logic.

Chapter 3

Proofs as Programs

In this chapter we investigate a computational interpretation of constructive proofs and relate it to functional programming. On the propositional fragment of logic this is referred to as the Curry-Howard isomorphism [How80]. From the very outset of the development of constructive logic and mathematics, a central idea has been that proofs ought to represent constructions. The Curry-Howard isomorphism is only a particularly poignant and beautiful realization of this idea. In a highly influential subsequent paper, Martin-Löf [ML80] developed it further into a more expressive calculus called *type theory*.

3.1 Propositions as Types

In order to illustrate the relationship between proofs and programs we introduce a new judgment:

$M : A$ $\quad$ M is a proof term for proposition A

We presuppose that A is a proposition when we write this judgment. We will also interpret $M : A$ as "*M is a program of type A*". These dual interpretations of the same judgment is the core of the Curry-Howard isomorphism. We either think of M as a term that represents the proof of A *true*, or we think of A as the type of the program M. As we discuss each connective, we give both readings of the rules to emphasize the analogy.

We intend that if $M : A$ then A *true*. Conversely, if A *true* then $M : A$. But we want something more: every deduction of $M : A$ should correspond to a deduction of A *true* with an identical structure and vice versa. In other words we annotate the inference rules of natural deduction with proof terms. The property above should then be obvious.

Conjunction. Constructively, we think of a proof of $A \wedge B$ *true* as a pair of proofs: one for A *true* and one for B *true*.

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

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

$$\frac{M : A \wedge B}{\mathbf{fst}\, M : A}\ \wedge E_L \qquad \frac{M : A \wedge B}{\mathbf{snd}\, M : B}\ \wedge E_R$$

Hence conjunction $A \wedge B$ corresponds to the product type $A \times B$.

Truth. Constructively, we think of a proof of $\top$ *true* as a unit element that carries now information.

$$\frac{}{\langle\rangle : \top}\ \top I$$

Hence $\top$ corresponds to the unit type $\mathbf{1}$ with one element. There is no elimination rule and hence no further proof term constructs for truth.

Implication. Constructively, we think of a proof of $A \supset B$ *true* as a function which transforms a proof of A *true* into a proof of B *true*.

In mathematics and many programming languages, we define a function f of a variable x by writing $f(x) = \ldots$ where the right-hand side "$\ldots$" depends on x. For example, we might write $f(x) = x^2 + x - 1$. In functional programming, we can instead write $f = \lambda x.\ x^2 + x - 1$, that is, we explicitly form a functional object by λ-*abstraction* of a variable (x, in the example).

We now use the notation of λ-abstraction to annotate the rule of implication introduction with proof terms. In the official syntax, we label the abstraction with a proposition (writing $\lambda u{:}A$) in order to specify the domain of a function unambiguously. In practice we will often omit the label to make expressions shorter—usually (but not always!) it can be determined from the context.

$$\begin{array}{c} \dfrac{}{u : A}\ u \\ \vdots \\ \dfrac{M : B}{\lambda u{:}A.\ M : A \supset B}\ \supset I^u \end{array}$$

The hypothesis label u acts as a variable, and any use of the hypothesis labeled u in the proof of B corresponds to an occurrence of u in M.

As a concrete example, consider the (trivial) proof of $A \supset A$ *true*:

$$\frac{\dfrac{}{A\ true}\ u}{A \supset A\ true}\ \supset I^u$$

If we annotate the deduction with proof terms, we obtain

$$\frac{\dfrac{}{u : A}\ u}{(\lambda u{:}A.\ u) : A \supset A}\ \supset I^u$$

So our proof corresponds to the identity function id at type A which simply returns its argument. It can be defined with $\mathrm{id}(u) = u$ or $\mathrm{id} = (\lambda u{:}A.\ u)$.

The rule for implication elimination corresponds to function application. Following the convention in functional programming, we write $M\,N$ for the application of the function M to argument N, rather than the more verbose $M(N)$.

$$\frac{M : A \supset B \quad N : A}{M\,N : B}\ \supset E$$

What is the meaning of $A \supset B$ as a type? From the discussion above it should be clear that it can be interpreted as a function type $A \to B$. The introduction and elimination rules for implication can also be viewed as formation rules for functional abstraction $\lambda u{:}A.\ M$ and application $M\,N$.

Note that we obtain the usual introduction and elimination rules for implication if we erase the proof terms. This will continue to be true for all rules in the remainder of this section and is immediate evidence for the soundness of the proof term calculus, that is, if $M : A$ then A *true*.

As a second example we consider a proof of $(A \wedge B) \supset (B \wedge A)$ *true*.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{A \wedge B\ \mathit{true}}\ u}{B\ \mathit{true}}\ \wedge E_R \quad \dfrac{\overline{A \wedge B\ \mathit{true}}\ u}{A\ \mathit{true}}\ \wedge E_L}{B \wedge A\ \mathit{true}}\ \wedge I}{(A \wedge B) \supset (B \wedge A)\ \mathit{true}}\ \supset I^u$$

When we annotate this derivation with proof terms, we obtain a function which takes a pair $\langle M, N\rangle$ and returns the reverse pair $\langle N, M\rangle$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\overline{u : A \wedge B}\ u}{\mathbf{snd}\,u : B}\ \wedge E_R \quad \dfrac{\overline{u : A \wedge B}\ u}{\mathbf{fst}\,u : A}\ \wedge E_L}{\langle \mathbf{snd}\,u, \mathbf{fst}\,u\rangle : B \wedge A}\ \wedge I}{(\lambda u.\ \langle \mathbf{snd}\,u, \mathbf{fst}\,u\rangle) : (A \wedge B) \supset (B \wedge A)}\ \supset I^u$$

Disjunction. Constructively, we think of a proof of $A \vee B$ *true* as either a proof of A *true* or B *true*. Disjunction therefore corresponds to a disjoint sum type $A + B$, and the two introduction rules correspond to the left and right injection into a sum type.

$$\frac{M : A}{\mathbf{inl}^B\,M : A \vee B}\ \vee I_L \qquad \frac{N : B}{\mathbf{inr}^A\,N : A \vee B}\ \vee I_R$$

In the official syntax, we have annotated the injections **inl** and **inr** with propositions B and A, again so that a (valid) proof term has an unambiguous type. In writing actual programs we usually omit this annotation. The elimination rule

corresponds to a case construct which discriminates between a left and right injection into a sum types.

$$\frac{\begin{array}{ccc} & \overline{}^{\,u} & \overline{}^{\,w} \\ & \vdots & \vdots \\ M : A \vee B & N : C & O : C \end{array}}{\mathbf{case}\, M\, \mathbf{of}\, \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O : C}\ \vee E^{u,w}$$

Recall that the hypothesis labeled u is available only in the proof of the second premise and the hypothesis labeled w only in the proof of the third premise. This means that the scope of the variable u is N, while the scope of the variable w is O.

Falsehood. There is no introduction rule for falsehood ($\bot$). We can therefore view it as the empty type $\mathbf{0}$. The corresponding elimination rule allows a term of $\bot$ to stand for an expression of any type when wrapped with **abort**. However, there is no computation rule for it, which means during computation of a valid program we will never try to evaluate a term of the form $\mathbf{abort}\, M$.

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

As before, the annotation C which disambiguates the type of $\mathbf{abort}\, M$ will often be omitted.

This completes our assignment of proof terms to the logical inference rules. Now we can interpret the interaction laws we introduced early as programming exercises. Consider the following distributivity law:

(L11a) $\quad (A \supset (B \wedge C)) \supset (A \supset B) \wedge (A \supset C)$ *true*

Interpreted constructively, this assignment can be read as:

> Write a function which, when given a function from A to pairs of type $B \wedge C$, returns two functions: one which maps A to B and one which maps A to C.

This is satisfied by the following function:

$$\lambda u.\ \langle (\lambda w.\ \mathbf{fst}\, (u\, w)), (\lambda v.\ \mathbf{snd}\, (u\, v)) \rangle$$

The following deduction provides the evidence:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{u : A \supset (B \wedge C)}\,u \quad \dfrac{}{w : A}\,w}{u\,w : B \wedge C}\,\supset E}{\mathbf{fst}\,(u\,w) : B}\,\wedge E_L}{\lambda w.\,\mathbf{fst}\,(u\,w) : A \supset B}\,\supset I^w \quad \dfrac{\dfrac{\dfrac{\dfrac{}{u : A \supset (B \wedge C)}\,u \quad \dfrac{}{v : A}\,v}{u\,v : B \wedge C}\,\supset E}{\mathbf{snd}\,(u\,v) : C}\,\wedge E_R}{\lambda v.\,\mathbf{snd}\,(u\,v) : A \supset C}\,\supset I^v}{\langle(\lambda w.\,\mathbf{fst}\,(u\,w)), (\lambda v.\,\mathbf{snd}\,(u\,v))\rangle : (A \supset B) \wedge (A \supset C)}\,\wedge I}{\lambda u.\,\langle(\lambda w.\,\mathbf{fst}\,(u\,w)), (\lambda v.\,\mathbf{snd}\,(u\,v))\rangle : (A \supset (B \wedge C)) \supset ((A \supset B) \wedge (A \supset C))}\,\supset I^u$$

Programs in constructive propositional logic are somewhat uninteresting in that they do not manipulate basic data types such as natural numbers, integers, lists, trees, etc. We introduce such data types in Section **??**, following the same method we have used in the development of logic.

To close this section we recall the guiding principles behind the assignment of proof terms to deductions.

1. For every deduction of A *true* there is a proof term M and deduction of $M : A$.
2. For every deduction of $M : A$ there is a deduction of A *true*
3. The correspondence between proof terms M and deductions of A *true* is a bijection.

3.2 Reduction

In the preceding section, we have introduced the assignment of proof terms to natural deductions. If proofs are programs then we need to explain how proofs are to be executed, and which results may be returned by a computation.

We explain the operational interpretation of proofs in two steps. In the first step we introduce a judgment of *reduction* $M \Longrightarrow_R M'$, read "*M reduces to M'*". A computation then proceeds by a sequence of reductions $M \Longrightarrow_R M_1 \Longrightarrow_R M_2 \ldots$, according to a fixed strategy, until we reach a value which is the result of the computation. In this section we cover reduction; we may return to reduction strategies in a later lecture.

As in the development of propositional logic, we discuss each of the connectives separately, taking care to make sure the explanations are independent. This means we can consider various sublanguages and we can later extend our logic or programming language without invalidating the results from this section. Furthermore, it greatly simplifies the analysis of properties of the reduction rules.

In general, we think of the proof terms corresponding to the introduction rules as the *constructors* and the proof terms corresponding to the elimination rules as the *destructors*.

Conjunction. The constructor forms a pair, while the destructors are the left and right projections. The reduction rules prescribe the actions of the projections.

$$\begin{array}{rcl} \mathbf{fst}\,\langle M, N\rangle & \Longrightarrow_R & M \\ \mathbf{snd}\,\langle M, N\rangle & \Longrightarrow_R & N \end{array}$$

Truth. The constructor just forms the unit element, $\langle\,\rangle$. Since there is no destructor, there is no reduction rule.

Implication. The constructor forms a function by λ-abstraction, while the destructor applies the function to an argument. In general, the application of a function to an argument is computed by *substitution*. As a simple example from mathematics, consider the following equivalent definitions

$$f(x) = x^2 + x - 1 \qquad f = \lambda x.\, x^2 + x - 1$$

and the computation

$$f(3) = (\lambda x.\, x^2 + x - 1)(3) = [3/x](x^2 + x - 1) = 3^2 + 3 - 1 = 11$$

In the second step, we substitute 3 for occurrences of x in $x^2 + x - 1$, the *body of the λ-expression*. We write $[3/x](x^2 + x - 1) = 3^2 + 3 - 1$.

In general, the notation for the substitution of N for occurrences of u in M is $[N/u]M$. We therefore write the reduction rule as

$$(\lambda u{:}A.\, M)\, N \quad \Longrightarrow_R \quad [N/u]M$$

We have to be somewhat careful so that substitution behaves correctly. In particular, no variable in N should be bound in M in order to avoid conflict. We can always achieve this by renaming bound variables—an operation which clearly does not change the meaning of a proof term.

Disjunction. The constructors inject into a sum types; the destructor distinguishes cases. We need to use substitution again.

$$\begin{array}{lcl} \mathbf{case}\,\mathbf{inl}^B\,M\,\mathbf{of}\,\mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow_R & [M/u]N \\ \mathbf{case}\,\mathbf{inr}^A\,M\,\mathbf{of}\,\mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow_R & [M/w]O \end{array}$$

Falsehood. Since there is no constructor for the empty type there is no reduction rule for falsehood.

This concludes the definition of the reduction judgment. In the next section we will prove some of its properties.

As an example we consider a simple program for the composition of two functions. It takes a pair of two functions, one from A to B and one from B to C and returns their composition which maps A directly to C.

$$\mathsf{comp} \quad : \quad ((A \supset B) \wedge (B \supset C)) \supset (A \supset C)$$

We transform the following implicit definition into our notation step-by-step:

$$\begin{array}{rcl} \mathsf{comp}\,\langle f, g\rangle\,(w) & = & g(f(w)) \\ \mathsf{comp}\,\langle f, g\rangle & = & \lambda w.\ g(f(w)) \\ \mathsf{comp}\,u & = & \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)(w)) \\ \mathsf{comp} & = & \lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w) \end{array}$$

The final definition represents a correct proof term, as witnessed by the following deduction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{u : (A \supset B) \wedge (B \supset C)}\,u}{\mathbf{snd}\,u : B \supset C}\,\wedge E_R \qquad \dfrac{\dfrac{\dfrac{}{u : (A \supset B) \wedge (B \supset C)}\,u}{\mathbf{fst}\,u : A \supset B}\,\wedge E_L \quad \dfrac{}{w : A}\,w}{(\mathbf{fst}\,u)\,w : B}\,\supset E}{(\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w) : C}\,\supset E}{\lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w) : A \supset C}\,\supset I^w}{(\lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w)) : ((A \supset B) \wedge (B \supset C)) \supset (A \supset C)}\,\supset I^u$$

We now verify that the composition of two identity functions reduces again to the identity function. First, we verify the typing of this application.

$$(\lambda u.\ \lambda w.\ (\mathbf{snd}\,u)\,((\mathbf{fst}\,u)\,w))\,\langle(\lambda x.\ x), (\lambda y.\ y)\rangle \quad : \quad A \supset A$$

Now we show a possible sequence of reduction steps. This is by no means uniquely determined.

$$\begin{array}{ll} & (\lambda u.\ \lambda w.\ (\mathbf{snd}\, u)\, ((\mathbf{fst}\, u)\, w))\, \langle (\lambda x.\ x), (\lambda y.\ y)\rangle \\ \Longrightarrow_R & \lambda w.\ (\mathbf{snd}\, \langle (\lambda x.\ x), (\lambda y.\ y)\rangle)\, ((\mathbf{fst}\, \langle (\lambda x.\ x), (\lambda y.\ y)\rangle)\, w) \\ \Longrightarrow_R & \lambda w.\ (\lambda y.\ y)\, ((\mathbf{fst}\, \langle (\lambda x.\ x), (\lambda y.\ y)\rangle)\, w) \\ \Longrightarrow_R & \lambda w.\ (\lambda y.\ y)\, ((\lambda x.\ x)\, w) \\ \Longrightarrow_R & \lambda w.\ (\lambda y.\ y)\, w \\ \Longrightarrow_R & \lambda w.\ w \end{array}$$

We see that we may need to apply reduction steps to subterms in order to reduce a proof term to a form in which it can no longer be reduced. We postpone a more detailed discussion of this until we discuss the operational semantics in full.

3.3 Expansion

We saw in the previous section that proof reductions that witness local soundness form the basis for the computational interpretation of proofs. Less relevant to computation are the local expansions. What they tell us, for example, is that if we need to return a pair from a function, we can always construct it as $\langle M, N\rangle$ for some M and N. Another example would be that whenever we need to return a function, we can always construct it as $\lambda u.\ M$ for some M.

We can derive what the local expansion must be by annotating the deduction from Section 2.5 with proof terms. We leave this as an exercise to the reader. The left-hand side of each expansion has the form $M : A$, where M is an arbitrary term and A is a logical connective or constant applied to arbitrary propositions. On the right hand side we have to apply a destructor to M and then reconstruct a term of the original type. The resulting rules can be found in Figure 3.3.

3.4 Summary of Proof Terms

Judgments.

$M : A$	M is a proof term for proposition A, see Figure 3.1
$M \Longrightarrow_R M'$	M reduces to M', see Figure 3.2
$M : A \Longrightarrow_E M'$	M expands to M', see Figure 3.3

Concrete Syntax. The concrete syntax for proof terms used in the mechanical proof checker has some minor differences to the form we presented above.

Constructors	Destructors
$\dfrac{M : A \quad N : B}{\langle M, N\rangle : A \wedge B}\ \wedge I$	$\dfrac{M : A \wedge B}{\mathbf{fst}\, M : A}\ \wedge E_L$ $\dfrac{M : A \wedge B}{\mathbf{snd}\, M : B}\ \wedge E_R$
$\dfrac{}{\langle\,\rangle : \top}\ \top I$	no destructor for $\top$
$\dfrac{\begin{array}{c}\overline{u : A}\ u \\ \vdots \\ M : B\end{array}}{\lambda u{:}A.\ M : A \supset B}\ \supset I^u$	$\dfrac{M : A \supset B \quad N : A}{M\, N : B}\ \supset E$
$\dfrac{M : A}{\mathbf{inl}^B\, M : A \vee B}\ \vee I_L$ $\dfrac{N : B}{\mathbf{inr}^A\, N : A \vee B}\ \vee I_R$	$\dfrac{M : A \vee B \quad \begin{array}{c}\overline{u : A}\ u \\ \vdots \\ N : C\end{array} \quad \begin{array}{c}\overline{w : B}\ w \\ \vdots \\ O : C\end{array}}{\mathbf{case}\, M\, \mathbf{of}\, \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O : C}\ \vee E^{u,w}$
no constructor for $\bot$	$\dfrac{M : \bot}{\mathbf{abort}^C\, M : C}\ \bot E$

Figure 3.1: Proof term assignment for natural deduction

$$
\begin{array}{rcl}
\mathbf{fst}\,\langle M, N\rangle & \Longrightarrow_R & M \\
\mathbf{snd}\,\langle M, N\rangle & \Longrightarrow_R & N \\
\text{no reduction for } \langle\,\rangle & & \\
(\lambda u{:}A.\ M)\,N & \Longrightarrow_R & [N/u]M \\
\mathbf{case}\,\mathbf{inl}^B\,M\ \mathbf{of}\ \mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow_R & [M/u]N \\
\mathbf{case}\,\mathbf{inr}^A\,M\ \mathbf{of}\ \mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O & \Longrightarrow_R & [M/w]O \\
\text{no reduction for } \mathbf{abort} & &
\end{array}
$$

Figure 3.2: Proof term reductions

$$
\begin{array}{lcl}
M : A \wedge B & \Longrightarrow_E & \langle \mathbf{fst}\,M, \mathbf{snd}\,M\rangle \\
M : A \supset B & \Longrightarrow_E & \lambda u{:}A.\ M\,u \quad \text{for } u \text{ not free in } M \\
M : \top & \Longrightarrow_E & \langle\,\rangle \\
M : A \vee B & \Longrightarrow_E & \mathbf{case}\,M\ \mathbf{of}\ \mathbf{inl}\,u \Rightarrow \mathbf{inl}^B\,u \mid \mathbf{inr}\,w \Rightarrow \mathbf{inr}^A\,w \\
M : \bot & \Longrightarrow_E & \mathbf{abort}^{\bot}\,M
\end{array}
$$

Figure 3.3: Proof term expansions

u	`u`	Variable
$\langle M, N\rangle$	`(M,N)`	Pair
$\mathbf{fst}\,M$	`fst M`	First projection
$\mathbf{snd}\,M$	`snd M`	Second projection
$\langle\,\rangle$	`()`	Unit element
$\lambda u{:}A.\ M$	`fn u => M`	Abstraction
$M\,N$	`M N`	Application
$\mathbf{inl}^B\,M$	`inl M`	Left injection
$\mathbf{inr}^A\,N$	`inr N`	Right injection
$\mathbf{case}\,M$	`case M`	Case analysis
$\quad\mathbf{of}\ \mathbf{inl}\,u \Rightarrow N$	`  of inl u => N`	
$\quad\mid \mathbf{inr}\,w \Rightarrow O$	`   | inr w => O`	
	`end`	
$\mathbf{abort}^C\,M$	`abort M`	Abort

Pairs and unit element are delimited by parentheses '`(`' and '`)`' instead of angle brackets $\langle$ and $\rangle$. The `case` constructs requires an `end` token to mark the end of the a sequence of cases.

Type annotations are generally omitted, but a whole term can explicitly be

given a type. The proof checker (which here is also a type checker) infers the missing information. Occasionally, an explicit type ascription `M : A` is necessary as a hint to the type checker.

For rules of operator precedence, the reader is referred to the on-line documentation of the proof checking software available with the course material. Generally, parentheses can be used to disambiguate or override the standard rules.

As an example, we show the proof term implementing function composition.

```
term comp : (A => B) & (B => C) => (A => C) =
fn u => fn x => (snd u) ((fst u) x);
```

We also allow annotated deductions, where each line is annotated with a proof term. This is a direct transcription of deduction for judgments of the form $M : A$. As an example, we show the proof that $A \vee B \supset B \vee A$, first in the pure form.

```
proof orcomm : A | B => B | A =
begin
[ A | B;
  [ A;
    B | A];
  [ B;
    B | A];
  B | A ];
A | B => B | A
end;
```

Now we systematically annotate each line and obtain

```
annotated proof orcomm : A | B => B | A =
begin
[ u : A | B;
  [ v : A;
    inr v : B | A];
  [ w : B;
    inl w : B | A];
  case u
    of inl v => inr v
     | inr w => inl w
   end : B | A ];
fn u => case u
          of inl v => inr v
           | inr w => inl w
         end : A | B => B | A
end;
```

3.5 Properties of Proof Terms

In this section we analyze and verify various properties of proof terms. Rather than concentrate on reasoning within the logical calculi we introduced, we now want to reason about them. The techniques are very similar—they echo the ones we have introduced so far in natural deduction. This should not be surprising. After all, natural deduction was introduced to model mathematical reasoning, and we now engage in some mathematical reasoning about proof terms, propositions, and deductions. We refer to this as *metalogical reasoning*.

First, we need some more formal definitions for certain operations on proof terms, to be used in our meta-logical analysis. One rather intuitive property of is that variable names should not matter. For example, the identity function at type A can be written as $\lambda u{:}A.\ u$ or $\lambda w{:}A.\ w$ or $\lambda u'{:}A.\ u'$, etc. They all denote the same function and the same proof. We therefore identify terms which differ only in the names of variables (here called u) bound in $\lambda u{:}A.\ M$, $\mathbf{inl}\, u \Rightarrow M$ or $\mathbf{inr}\, u \Rightarrow O$. But there are pitfalls with this convention: variables have to be renamed *consistently* so that every variable refers to the same binder before and after the renaming. For example (omitting type labels for brevity):

$$\begin{array}{rcl} \lambda u.\ u & = & \lambda w.\ w \\ \lambda u.\ \lambda w.\ u & = & \lambda u'.\ \lambda w.\ u' \\ \lambda u.\ \lambda w.\ u & \neq & \lambda u.\ \lambda w.\ w \\ \lambda u.\ \lambda w.\ u & \neq & \lambda w.\ \lambda w.\ w \\ \lambda u.\ \lambda w.\ w & = & \lambda w.\ \lambda w.\ w \end{array}$$

The convention to identify terms which differ only in the naming of their bound variables goes back to the first papers on the λ-calculus by Church and Rosser [CR36], is called the "*variable name convention*" and is pervasive in the literature on programming languages and λ-calculi. The term λ-calculus typically refers to a pure calculus of functions formed with λ-abstraction. Our proof term calculus is called a *typed λ-calculus* because of the presence of propositions (which an be viewed as types).

Following the variable name convention, we may silently rename when convenient. A particular instance where this is helpful is substitution. Consider

$$[u/w](\lambda u.\ w\, u)$$

that is, we substitute u for w in $\lambda u.\ w\, u$. Note that u is a variable visible on the outside, but also bound by λu. By the variable name convention we have

$$[u/w](\lambda u.\ w\, u) = [u/w](\lambda u'.\ w\, u') = \lambda u'.\ u\, u'$$

which is correct. But we cannot substitute without renaming, since

$$[u/w](\lambda u.\ w\, u) \neq \lambda u.\ u\, u$$

In fact, the right hand side below is invalid, while the left-hand side makes perfect sense. We say that u is *captured* by the binder λu. If we assume a hypothesis $u{:}\top \supset A$ then

$$[u/w](\lambda u{:}\top.\ w\, u) : A$$

but

$$\lambda u{:}\top.\ u\,u$$

is not well-typed since the first occurrence of u would have to be of type $\top \supset A$ but instead has type $\top$.

So when we carry out substitution $[M/u]N$ we need to make sure that no variable in M is *captured* by a binder in N, leading to an incorrect result. Fortunately we can always achieve that by renaming some bound variables in N if necessary. We could now write down a formal definition of substitution, based on the cases for the term we are substituting into. However, we hope that the notion is sufficiently clear that this is not necessary.

Instead we revisit the substitution principle for hypothetical judgments. It states that if we have a hypothetical proof of C *true* from A *true* and we have a proof of A *true*, we can substitute the proof of A *true* for uses of the hypothesis A *true* and obtain a (non-hypothetical) proof of A *true*. In order to state this more precisely in the presence of several hypotheses, we recall that

$$\begin{array}{c} A_1\ \mathit{true} \ldots A_n\ \mathit{true} \\ \vdots \\ C\ \mathit{true} \end{array}$$

can be written as

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

Generally we abbreviate several hypotheses by Δ. We then have the following properties, evident from the very definition of hypothetical judgments and hypothetical proofs

Weakening: If $\Delta \vdash C\ \mathit{true}$ then $\Delta, \Delta' \vdash C\ \mathit{true}$.

Substitution: If $\Delta, A\ \mathit{true}, \Delta' \vdash C\ \mathit{true}$ and $\Delta \vdash A\ \mathit{true}$ then $\Delta, \Delta' \vdash C\ \mathit{true}$.

As indicated above, weakening is realized by adjoining unused hypotheses, substitutions is realized by substitution of proofs for hypotheses.

For the proof term judgment, $M : A$, we use the same notation and write

$$\begin{array}{c} u_1{:}A_1\ \ldots\ u_n{:}A_n \\ \vdots \\ N : C \end{array}$$

as

$$\underbrace{u_1{:}A_1, \ldots, u_n{:}A_n}_{\Gamma} \vdash N : C$$

We use Γ to refer to collections of hypotheses $u_i{:}A_i$. In the deduction of $N : C$, each u_i stands for an unknown proof term for A_i, simply assumed to exist. If we actually find a proof $M_i{:}A_i$ we can eliminate this assumption, again by substitution. However, this time, the substitution has to perform two operations:

we have to substitute M_i for u_i (the unknown proof term variable), and the deduction of $M_i : A_i$ for uses of the hypothesis $u_i{:}A_i$. More precisely, we have the following two properties:

Weakening: If $\Gamma, \Gamma' \vdash N : C$ then $\Gamma, u{:}A, \Gamma' \vdash N : C$.

Substitution: If $\Gamma, u{:}A, \Gamma' \vdash N : C$ and $\Gamma \vdash M : A$ then $\Gamma, \Gamma' \vdash [M/u]N : C$.

Now we are in a position to state and prove our second meta-theorem, that is, a theorem about the logic under consideration. The theorem is called *subject reduction* because is concerns the *subject* M of the judgment $M : A$. It states that reduction preserves the type of an object. We make the hypotheses explicit as we have done in the explanations above.

Theorem 3.1 (Subject Reduction)
If $\Gamma \vdash M : A$ *and* $M \Longrightarrow_R M'$ *then* $\Gamma \vdash M' : A$.

Proof: We consider each case in the definition of $M \Longrightarrow_R M'$ in turn and show that the property holds. This is simply an instance of *proof by cases*.

Case: $\mathbf{fst}\,\langle M_1, M_2\rangle \Longrightarrow_R M_1$. By assumption we also know that

$$\Gamma \vdash \mathbf{fst}\,\langle M_1, M_2\rangle : A.$$

We need to show that $\Gamma \vdash M_1 : A$.

Now we inspect all inference rules for the judgment $M : A$ and we see that there is only one way how the judgment above could have been inferred: by $\wedge E_L$ from

$$\Gamma \vdash \langle M_1, M_2\rangle : A \wedge A_2$$

for some A_2. This step is called *inversion*, since we infer the premises from the conclusion of the rule. But we have to be extremely careful to inspect all possibilities for derivations so that we do not forget any cases.

Next, we apply inversion again: the judgment above could only have been inferred by $\wedge I$ from the two premises

$$\Gamma \vdash M_1 : A$$

and

$$\Gamma \vdash M_2 : A_2$$

But the first of these is what we had to prove in this case and we are done.

Case: $\mathbf{snd}\,\langle M_1, M_2\rangle \Longrightarrow_R M_2$. This is symmetric to the previous case. We write it an abbreviated form.

$\Gamma \vdash \mathbf{snd}\,\langle M_1, M_2\rangle : A$	Assumption
$\Gamma \vdash \langle M_1, M_2\rangle : A_1 \wedge A$ for some A_1	By inversion
$\Gamma \vdash M_1 : A_1$ and	
$\Gamma \vdash M_2 : A$	By inversion

Here the last judgment is what we were trying to prove.

Case: There is no reduction for $\top$ since there is no elimination rule and hence no destructor.

Case: $(\lambda u{:}A_1.\ M_2)\, M_1 \Longrightarrow_R [M_1/u]M_2$. By assumption we also know that

$$\Gamma \vdash (\lambda u{:}A_1.\ M_2)\, M_1 : A.$$

We need to show that $\Gamma \vdash [M_1/u]M_2 : A$.

Since there is only one inference rule for function application, namely implication elimination ($\supset E$), we can apply inversion and find that

$$\Gamma \vdash (\lambda u{:}A_1.\ M_2) : A_1' \supset A$$

and

$$\Gamma \vdash M_1 : A_1'$$

for some A_1'. Now we repeat inversion on the first of these and conclude that

$$\Gamma, u{:}A_1 \vdash M_2 : A$$

and, moreover, that $A_1 = A_1'$. Hence

$$\Gamma \vdash M_1 : A_1$$

Now we can apply the substitution property to these to judgments to conclude

$$\Gamma \vdash [M_1/u]M_2 : A$$

which is what we needed to show.

Case: $(\mathbf{case}\,\mathbf{inl}^C\, M_1\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) \Longrightarrow_R [M_1/u]N$. By assumption we also know that

$$\Gamma \vdash (\mathbf{case}\,\mathbf{inl}^C\, M_1\ \mathbf{of}\ \mathbf{inl}\, u \Rightarrow N \mid \mathbf{inr}\, w \Rightarrow O) : A$$

Again we apply inversion and obtain three judgments

$$\begin{array}{l}\Gamma \vdash \mathbf{inl}^C\, M_1 : B' \vee C' \\ \Gamma, u{:}B' \vdash N : A \\ \Gamma, w{:}C' \vdash O : A\end{array}$$

for some B' and C'.

Again by inversion on the first of these, we find

$$\Gamma \vdash M_1 : B'$$

and also $C' = C$. Hence we can apply the substitution property to get

$$\Gamma \vdash [M_1/u]N : A$$

which is what we needed to show.

Case: $(\mathbf{case}\,\mathbf{inr}^B\,M_1\ \mathbf{of}\ \mathbf{inl}\,u \Rightarrow N \mid \mathbf{inr}\,w \Rightarrow O) \Longrightarrow_R [M_1/u]N$. This is symmetric to the previous case and left as an exercise.

Case: There is no introduction rule for $\bot$ and hence no reduction rule.

□

The important techniques introduced in the proof above are *proof by cases* and *inversion*. In a proof by cases we simply consider all possibilities for why a judgment could be evident and show the property we want to establish in each case. Inversion is very similar: from the shape of the judgment we see it could have been inferred only in one possible way, so we know the premises of this rule must also be evident. We see that these are just two slightly different forms of the same kind of reasoning.

If we look back at our early example computation, we saw that the reduction step does not always take place at the top level, but that the redex may be embedded in the term. In order to allow this, we need to introduce some additional ways to establish that $M \Longrightarrow_R M'$ when the actual reduction takes place *inside* M. This is accomplished by so-called *congruence rules*.

Bibliography

[CR36] Alonzo Church and J.B. Rosser. Some properties of conversion. *Transactions of the American Mathematical Society*, 39(3):472–482, May 1936.

[Dum91] Michael Dummett. *The Logical Basis of Metaphysics*. Harvard University Press, Cambridge, Massachusetts, 1991. The William James Lectures, 1976.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935. English translation in M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131, North-Holland, 1969.

[How80] W. A. Howard. The formulae-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 479–490. Academic Press, 1980. Hitherto unpublished note of 1969, rearranged, corrected, and annotated by Howard.

[ML80] Per Martin-Löf. Constructive mathematics and computer programming. In *Logic, Methodology and Philosophy of Science VI*, pages 153–175. North-Holland, 1980.

[ML96] Per Martin-Löf. On the meanings of the logical constants and the justifications of the logical laws. *Nordic Journal of Philosophical Logic*, 1(1):11–60, 1996.

[Pra65] Dag Prawitz. *Natural Deduction*. Almquist & Wiksell, Stockholm, 1965.

Chapter 4

First-Order Logic and Type Theory

In the first chapter we developed the logic of pure propositions without reference to data types such as natural numbers. In the second chapter we explained the computational interpretation of proofs. Later in this chapter we will introduce data types and ways to compute with them using primitive recursion. Together, these will allows us to reason about data and programs manipulating data. In other words, we will be able to prove our programs correct with respect to their expected behavior on data. The principal means for will be induction, introduced towards the end of this chapter. There are several ways to employ the machinery we will develop. For example, we can execute proofs directly, using their interpretation as programs. Or we can *extract* functions, ignoring some proof objects that have are irrelevant with respect to the data our programs return. That is, we can contract proofs to programs. Or we can simply write our programs and use the logical machinery we have developed to prove them correct.

In practice, there are situations in which each of them is appropriate. However, we note that in practice we rarely formally prove our programs to be correct. This is because there is no mechanical procedure to establish if a given programs satisfies its specification. Moreover, we often have to deal with input or output, with mutable state or concurrency, or with complex systems where the specification itself could be as difficult to develop as the implementation. Instead, we typically convince ourselves that central parts of our program and the critical algorithms are correct. Even if proofs are never formalized, this chapter will help you in reasoning about programs and their correctness.

There is another way in which the material of this chapter is directly relevant to computing practice. In the absence of practical methods for verifying full correctness, we can be less ambitious by limiting ourselves to program properties that can indeed be mechanically verified. The most pervasive application of this idea in programming is the idea of *type systems*. By checking the type

correctness of a program we fall far short of verifying it, but we establish a kind of consistency statement. Since languages satisfy (or are supposed to satisfy) type preservation, we know that, if a result is returned, it is a value of the right type. Moreover, during the execution of a program (modeled here by reduction), all intermediate states are well-typed which prevents certain absurd situations, such as adding a natural number to a function. This is often summarized in the slogan that "*well-typed programs cannot go wrong*". Well-typed programs are *safe* in this respect. In terms of machine language, assuming a correct compiler, this guards against irrecoverable faults such as jumping to an address that does not contain valid code, or attempting to write to inaccessible memory location.

There is some room for exploring the continuum between types, as present in current programming languages, and full specifications, the domain of *type theory*. By presenting these elements in a unified framework, we have the basis for such an exploration.

We begin this chapter with a discussion of the universal and existential quantifiers, followed by a number of examples of inductive reasoning with data types.

4.1 Quantification

In this section, we introduce universal and existential quantification. As usual, we follow the method of using introduction and elimination rules to explain the meaning of the connectives. An important aspect of the treatment of quantifiers is that it should be completely independent of the domain of quantification. We want to capture what is true of all quantifiers, rather than those applying to natural numbers or integers or rationals or lists or other type of data. We will therefore quantify over objects of an unspecified (arbitrary) type τ. Whatever we derive, will of course also hold for specific domain (for example, $\tau = \mathsf{nat}$). The basic judgment connecting objects t to types τ is $t : \tau$. We also have a judgment that τ is a valid type, written as $\tau\ \mathit{type}$. We will refer to these judgments here, but not define any specific instances until later in the chapter when discussing data types.

First, universal quantification, written as $\forall x{:}\tau.\ A(x)$. Here x is a bound variable and can therefore be renamed as discussed in the preceding chapter. When we write $A(x)$ we mean an arbitrary proposition which may depend on x. We will als say that A is *predicate* on elements of type τ.

For the quantification to be well-formed, the body must be well-formed under the assumption that a is an object of type τ, and τ must be a valid type, two new judgments we consider in this chapter.

$$\frac{\tau\ \mathit{type} \quad \begin{array}{c}\overline{a : \tau}\\ \vdots \\ A(a)\ \mathit{prop}\end{array}}{\forall x{:}\tau.\ A(x)\ \mathit{prop}}\ \forall F^a$$

For the introduction rule we require that $A(a)$ be true for arbitrary a. In other words, the premise contains a *parametric judgment*, explained in more detail below.

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

It is important that a be a new parameter, not used outside of its scope, which is the derivation between the new hypothesis $a : \tau$ and the conclusion $A(a)$ *true*. In particular, it may not occur in $\forall x{:}\tau.\ A(x)$.

If we think of this as the defining property of universal quantification, then a verification of $\forall x{:}\tau.\ A(x)$ describes a construction by which an arbitrary $t : \tau$ can be transformed into a proof of $A(t)$ *true*.

$$\dfrac{\forall x{:}\tau.\ A(x)\ \mathit{true} \quad t:\tau}{A(t)\ \mathit{true}}\ \forall E$$

We must verify that $t : \tau$ so that $A(t)$ is a well-formed proposition.

The local reduction uses the following *substitution principle for parametric judgments*:

$$\text{If}\quad \begin{array}{c}\overline{a:\tau}\\ \mathcal{D}\\ J(a)\end{array}\quad\text{and}\quad \begin{array}{c}\mathcal{E}\\ t:\tau\end{array}\quad\text{then}\quad \begin{array}{c}\dfrac{\mathcal{E}}{t:\tau}\\ [t/a]\mathcal{D}\\ J(t)\end{array}$$

The right hand side is constructed by systematically substituting t for a in $\mathcal{D}$ and the judgments occurring in it. As usual, this substitution must be *capture avoiding* to be meaningful. It is the substitution into the judgments themselves which distinguishes substitution for parameters from substitution for hypotheses.

The local reduction for universal quantification then exploits this substitution principle.

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

The local expansion introduces a parameter which we can use the eliminate

the universal quantifier.

$$\begin{array}{c} \mathcal{D} \\ \forall x{:}\tau.\ A(x)\ true \end{array} \quad \Longrightarrow_E \quad \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \forall x{:}\tau.\ A(x)\ true\end{array} \quad \overline{a:\tau}}{A(a)\ true}\ \forall E}{\forall x{:}\tau.\ A(x)\ true}\ \forall I^a$$

As a simple example, consider the proof that universal quantifiers distribute over conjunction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\forall x{:}\tau.\ A(x)) \wedge (\forall x{:}\tau.\ B(x))\ true}\ u}{\forall x{:}\tau.\ A(x)\ true}\ \wedge E_L \quad \overline{a:\tau}}{A(a)\ true}\ \forall E}{\forall x{:}\tau.\ A(x)\ true}\ \forall I^a \quad \dfrac{\dfrac{\dfrac{\dfrac{}{(\forall x{:}\tau.\ A(x)) \wedge (\forall x{:}\tau.\ B(x))\ true}\ u}{\forall x{:}\tau.\ B(x)\ true}\ \wedge E_R \quad \overline{b:\tau}}{B(b)\ true}\ \forall E}{\forall x{:}\tau.\ B(x)\ true}\ \forall I^b}{(\forall x{:}\tau.\ A(x)) \wedge (\forall x{:}\tau.\ B(x))\ true}\ \wedge I}{(\forall x{:}\tau.\ A(x) \wedge B(x)) \supset (\forall x{:}\tau.\ A(x)) \wedge (\forall x{:}\tau.\ B(x))\ true}\ \supset I^u$$

The existential quantifier is more difficult to specify, although the introduction rule seems innocuous enough.

$$\dfrac{t:\tau \quad A(t)\ true}{\exists x{:}\tau.\ A(x)\ true}\ \exists I$$

The elimination rules creates some difficulties. We cannot write

$$\dfrac{\exists x{:}\tau.\ A(x)\ true}{A(t)\ true}\ \exists E?$$

because we do not know for which t is is the case that $A(t)$ holds. It is easy to see that local soundness would fail with this rule, because we would prove $\exists x{:}\tau.\ A(x)$ with one witness t and then eliminate the quantifier using another object t'.

The best we can do is to assume that $A(a)$ is true for some new parameter a. The scope of this assumption is limited to the proof of some conclusion $C\ true$ which does not mention a (which must be new).

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

Here, the scope of the hypotheses a and u is the deduction on the right, indicated by the vertical dots. In particular, C may not depend on a. We use this crucially in the local reduction.

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ t:\tau\end{array}\quad\begin{array}{c}\mathcal{E}\\ A(t)\ true\end{array}}{\exists x{:}\tau.\ A(x)\ true}\exists I \qquad \begin{array}{c}\overline{a:\tau}\\ \vdots\\ C\ true\end{array}\quad\begin{array}{c}\overline{A(a)\ true}\ u\\ \mathcal{F}\end{array}}{C\ true}\exists E^{a,u} \qquad \Longrightarrow_R \qquad \dfrac{\begin{array}{c}\mathcal{D}\\ t:\tau\end{array}\quad\dfrac{\begin{array}{c}\mathcal{E}\\ A(t)\ true\end{array}}{\begin{array}{c}[t/a]\mathcal{F}\\ C\ true\end{array}}\ u}{}$$

The reduction requires two substitutions, one for a parameter a and one for a hypothesis u.

The local expansion is patterned after the disjunction.

$$\begin{array}{c}\mathcal{D}\\ \exists x{:}\tau.\ A(x)\ true\end{array} \quad \Longrightarrow_E \quad \dfrac{\begin{array}{c}\mathcal{D}\\ \exists x{:}\tau.\ A(x)\ true\end{array}\quad\dfrac{\overline{a:\tau}\quad\overline{A(a)\ true}\ u}{\exists x{:}\tau.\ A(x)\ true}\exists I}{\exists x{:}\tau.\ A(x)\ true}\exists E^{a,u}$$

As an example of quantifiers we show the equivalence of $\forall x{:}\tau.\ A(x) \supset C$ and $(\exists x{:}\tau.\ A(x)) \supset C$, where C does not depend on x. Generally, in our propositions, any possibly dependence on a bound variable is indicated by writing a general *predicate* $A(x_1, \ldots, x_n)$. We do not make explicit when such propositions are well-formed, although appropriate rules for explicit A could be given.

When looking at a proof, the static representation on the page is an inadequate image for the dynamics of proof construction. As we did earlier, we give two examples where we show the various stages of proof construction.

$$\begin{array}{c}\vdots\\ ((\exists x{:}\tau.\ A(x)) \supset C) \supset \forall x{:}\tau.\ (A(x) \supset C)\ true\end{array}$$

The first three steps can be taken without hesitation, because we can always apply implication and universal introduction from the bottom up without possibly missing a proof.

$$\dfrac{\dfrac{\dfrac{\begin{array}{c}\overline{(\exists x{:}\tau.\ A(x)) \supset C\ true}\ u\\ \vdots\\ C\ true\end{array}}{A(a) \supset C\ true}\supset I^w}{\forall x{:}\tau.\ A(x) \supset C\ true}\forall I^a \qquad \begin{array}{c}\overline{a:\tau}\\ \overline{A(a)\ true}\ w\end{array}}{((\exists x{:}\tau.\ A(x)) \supset C) \supset \forall x{:}\tau.\ (A(x) \supset C)\ true}\supset I^u$$

At this point the conclusion is atomic, so we must apply an elimination to an assumption if we follow the strategy of *introductions bottom-up* and *eliminations top-down*. The only possibility is implication elimination, since $a : \tau$ and

$A(a)$ *true* are atomic. This gives us a new subgoal.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a:\tau} \quad \dfrac{}{A(a)\ true}\,w}{\vdots}}{\dfrac{}{(\exists x{:}\tau.\ A(x)) \supset C\ true}\,u \qquad \exists x{:}\tau.\ A(x)}\supset E}{\dfrac{C\ true}{A(a)\supset C\ true}\supset I^w}}{\dfrac{}{\forall x{:}\tau.\ A(x)\supset C\ true}\forall I^a}}{((\exists x{:}\tau.\ A(x))\supset C)\supset \forall x{:}\tau.\ (A(x)\supset C)\ true}\supset I^u$$

At this point it is easy to see how to complete the proof with an existential introduction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{(\exists x{:}\tau.\ A(x))\supset C\ true}\,u \quad \dfrac{\dfrac{}{a:\tau} \quad \dfrac{}{A(a)\ true}\,w}{\exists x{:}\tau.\ A(x)}\exists I}{C\ true}\supset E}{A(a)\supset C\ true}\supset I^w}{\forall x{:}\tau.\ A(x)\supset C\ true}\forall I^a}{((\exists x{:}\tau.\ A(x))\supset C)\supset \forall x{:}\tau.\ (A(x)\supset C)\ true}\supset I^u$$

We now consider the reverse implication.

$$\dfrac{}{(\forall x{:}\tau.\ (A(x)\supset C))\supset((\exists x{:}\tau.\ A(x))\supset C)\ true}$$

From the initial goal, we can blindly carry out two implication introductions, bottom-up, which yields the following situation.

$$\dfrac{\dfrac{\dfrac{}{\exists x{:}\tau.\ A(x)\ true}\,w \quad \dfrac{}{\forall x{:}\tau.\ A(x)\supset C\ true}\,u}{\vdots}}{(\forall x{:}\tau.\ (A(x)\supset C))\supset((\exists x{:}\tau.\ A(x))\supset C)\ true}\supset I^u$$

No we have two choices: existential elimination applied to w or universal elimination applied to u. However, we have not introduced any terms, so only the

existential elimination can go forward.

$$\dfrac{\dfrac{\dfrac{}{\exists x{:}\tau.\ A(x)\ true}\,w \qquad \dfrac{\dfrac{\dfrac{}{\forall x{:}\tau.\ A(x)\supset C\ true}\,u \quad \dfrac{}{a:\tau}}{A(a)\supset C\ true}\,\forall E \quad \dfrac{}{A(a)\ true}\,v}{C\ true}\,\supset E \\ \vdots \\ C\ true}{\dfrac{C\ true}{(\exists x{:}\tau.\ A(x))\supset C\ true}\,\supset I^{w}}\,\exists E^{a,v}}{(\forall x{:}\tau.\ (A(x)\supset C))\supset((\exists x{:}\tau.\ A(x))\supset C)\ true}\,\supset I^{u}$$

At this point we need to apply another elimination rule to an assumption. We don't have much to work with, so we try universal elimination.

$$\dfrac{\dfrac{\dfrac{}{\exists x{:}\tau.\ A(x)\ true}\,w \qquad \dfrac{\dfrac{}{\forall x{:}\tau.\ A(x)\supset C\ true}\,u \quad \dfrac{}{a:\tau}}{A(a)\supset C\ true}\,\forall E \quad \dfrac{}{A(a)\ true}\,v \\ \vdots \\ C\ true}{\dfrac{C\ true}{(\exists x{:}\tau.\ A(x))\supset C\ true}\,\supset I^{w}}\,\exists E^{a,v}}{(\forall x{:}\tau.\ (A(x)\supset C))\supset((\exists x{:}\tau.\ A(x))\supset C)\ true}\,\supset I^{u}$$

Now we can fill the gap with an implication elimination.

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

In order to formalize the proof search strategy, we use the judgments A has a verification ($A\uparrow$) and A may be used ($A\downarrow$) as we did in the propositional case. Universal quantification is straightforward:

$$\dfrac{\dfrac{}{a:\tau}\\ \vdots \\ A(a)\uparrow}{\forall x{:}\tau.\ A(x)\uparrow}\,\forall I^{a} \qquad \dfrac{\forall x{:}\tau.\ A(x)\downarrow \quad t:\tau}{A(t)\downarrow}\,\forall E$$

We do not assign a direction to the judgment for typing objects, $t : \tau$.

Verifications for the existential elimination are patterned after the disjunction: we translate a usable $\exists x{:}\tau.\ A(x)$ into a usable $A(a)$ with a limited scope, both in the verification of some C.

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

As before, the fact that every true proposition has a verification is a kind of global version of the local soundness and completeness properties. If we take this for granted (since we do not prove it until later), then we can use this to demonstrate that certain propositions are not true, parametrically.

For example, we show that $(\exists x{:}\tau.\ A(x)) \supset (\forall x{:}\tau.\ A(x))$ is not true in general. After the first two steps of constructing a verification, we arrive at

$$\frac{\dfrac{\dfrac{\overline{\exists x{:}\tau.\ A(x)\downarrow}^{\,u} \quad \overline{a:\tau}}{\vdots}}{A(a)\uparrow}}{\forall x{:}\tau.\ A(x)\uparrow}\ \forall I^a}{(\exists x{:}\tau.\ A(x)) \supset (\forall x{:}\tau.\ A(x))\uparrow}\ \supset I^u$$

At this point we can only apply existential elimination, which leads to

$$\frac{\dfrac{\dfrac{\overline{\exists x{:}\tau.\ A(x)\downarrow}^{\,u} \qquad \begin{array}{c}\overline{b:\tau} \quad \overline{A(b)\downarrow}^{\,v} \quad \overline{a:\tau}\\ \vdots \\ A(a)\uparrow\end{array}}{A(a)\uparrow}\ \exists E^{b,v}}{\forall x{:}\tau.\ A(x)\uparrow}\ \forall I^a}{(\exists x{:}\tau.\ A(x)) \supset (\forall x{:}\tau.\ A(x))\uparrow}\ \supset I^u$$

We cannot close the gap, because a and b are different parameters. We can only apply existential elimination to assumption u again. But this only creates $c : \tau$ and $A(c)\downarrow$ for some new c, so have made no progress. No matter how often we apply existential elimination, since the parameter introduced must be new, we can never prove $A(a)$.

4.2 Computational Meaning of Quantification

Returning to one of our motivating examples, we saw that a constructive proof of $\forall x{:}\mathsf{nat}.\ \exists y{:}\mathsf{nat}.\ y > x \wedge \mathsf{prime}(x)$ should be a function which, when given a natural number x returns a natural number y which is greater than x and prime. We therefore suspect the computational content of a proof of universal quantifier over $\forall x{:}\tau.\ A(x)$ should be function from objects t of type τ to proofs of $A(t)$. The meaning of an existential quantifier $\exists x{:}\tau.\ A(x)$ should consist of a *withness* term t of type τ and a proof that $A(t)$ holds as specified.

Restating the above, the computational meaning of a proof of $\forall x{:}\tau.\ A(x)$ *true* is a function which, when given an argument t of type τ, returns a proof of $A(t)$. If we don't mind overloading notation, we obtain the following proof term assignment. We use the notation with localized hypotheses, which can now be either assumptions $a{:}\tau$ for object parameters a with their types or $u{:}A$ for propositional hypotheses A.

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

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

The computation rule simply performs the required substitution, and expansion generates an abstraction.

$$\begin{array}{lcll} (\lambda a{:}\tau.\ M)\,t & \Longrightarrow_R & [t/a]M & \\ M : \forall x{:}\tau.\ A & \Longrightarrow_E & \lambda a{:}\tau.\ M\,a & \text{where } a \text{ not free in } M \end{array}$$

At this point we may realize that a parameter a is nothing but a variable bound in proofs. We would not lose anything if we named such variables x to unify the notation.

The existential quantifier $\exists x{:}\tau.\ A(x)$ lies at the heart of constructive mathematics. This is because a proof of this should contain a *witness* t of type τ such that $A(t)$ *true*. The proof term assignment and computational contents of these rules is not particularly difficult. The proof term for an existential introduction is a pair consisting of the witness t and the proof that t satisfies the stated property. The elimination rule destructs the pair, making the components accessible.

$$\frac{\Gamma \vdash t : \tau \quad \Gamma \vdash M : A(t)}{\Gamma \vdash \langle t, M\rangle : \exists x{:}\tau.\ A(x)}\ \exists I$$

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

The reduction rule is straightforward, substituting both the witness and the proof term certifying its correctness.

$$\mathbf{let}\langle a, u\rangle = \langle t, M\rangle \mathbf{\ in\ } N \quad \Longrightarrow_R \quad [M/u]\,[t/a]\,N$$

As in the case of the propositional connectives, we now consider various interactions between quantifiers and connectives to obtain an intuition regarding their properties.

By annotating the earlier derivation that universal quantification distributes over conjunction we can extract the following proof term for this judgment (omitting some labels):

$$\begin{array}{rl} & \lambda u.\ \langle \lambda a{:}\tau.\ \mathbf{fst}\,(u\,a), \lambda b{:}\tau.\ \mathbf{snd}\,(u\,b)\rangle \\ : & (\forall x{:}\tau.\ A(x) \wedge B(x)) \supset (\forall x{:}\tau.\ A(x)) \wedge (\forall x{:}\tau.\ B(x)) \end{array}$$

The opposite direction also holds, which means that we can freely move the universal quantifier over conjunctions and vice versa. This judgment (and also the proof above) are parametric in τ. Any instance by a concrete type for τ will be an evident judgment. We show here only the proof term (again omitting some labels):

$$\begin{array}{rl} & \lambda u.\ \lambda a{:}\tau.\ \langle (\mathbf{fst}\ u)\ a, (\mathbf{snd}\ u)\ a\rangle \\ : & (\forall x{:}\tau.\ A(x)) \wedge (\forall x{:}\tau.\ B(x)) \supset (\forall x{:}\tau.\ A(x) \wedge B(x)) \end{array}$$

The proof that an existential can be pushed into the antecedent of an implication has the following proof term.

$$\begin{array}{rl} & \lambda u.\ \lambda a{:}\tau.\ \lambda w.\ u\,\langle a, w\rangle \\ : & ((\exists x{:}\tau.\ A(x)) \supset C) \supset \forall x{:}\tau.\ (A(x) \supset C) \end{array}$$

For the reverse implication we extract from the earlier proof:

$$\begin{array}{rl} & \lambda u.\ \lambda w.\ \mathbf{let}\ \langle a, v\rangle = w\ \mathbf{in}\ (u\,a)\,v \\ : & (\forall x{:}\tau.\ (A(x) \supset C)) \supset ((\exists x{:}\tau.\ A(x)) \supset C) \end{array}$$

4.3 First-Order Logic

First-order logic, also called the predicate calculus, is concerned with the study of propositions whose quantifiers range over a domain about which we make no assumptions. In our case this means we allow only quantifiers of the form $\forall x{:}\tau.\ A(x)$ and $\exists x{:}\tau.\ A(x)$ that are parametric in a type τ. We assume only that τ *type*, but no other property of τ. When we add particular types, such as natural numbers nat, we say that we reason within specific theories. The theory of natural numbers, for example, is called *arithmetic*. When we allow essentially arbitrary propositions and types explained via introduction and elimination constructs (including function types, product types, etc.) we say that we reason in *type theory*. It is important that type theory is open-ended: we can always add new propositions and new types and even new judgment forms, as long as

we can explain their meaning satisfactorily. On the other hand, first-order logic is essentially closed: when we add new constructs, we work in other theories or logics that include first-order logic, but we go beyond it in essential ways.

We have already seen some examples of reasoning in first-order logic in the two previous sections. In this section we investigate the truth of various other propositions in order to become comfortable with first-order reasoning. Just like propositional logic, first-order logic has both classical and constructive variants. We pursue the constructive or intuitionistic point of view. We can recover classical truth either via an interpretation such as Gödel's translation, discussed later in the class, or by adding the law of excluded middle. The practical difference at the first-order level is the interpretation of the existential quantifier. In classical logic, we can prove a proposition $\exists x{:}\tau.\ A(x)$ *true* by proving $\neg\forall x{:}\tau.\ \neg A(x)$ *true* instead. Such a proof may not yield the witness object t such that $A(t)$ is satisfied, which is required under the constructive interpretation of the existential quantifier. But how is it possible to provide witnesses in pure logic, without any assumptions about the domain of quantifiers? The answer is that assumptions about the existence of objects will be introduced locally during the proof. But we have to be careful to verify that the objects we use to witness existential quantifiers or instantiate universal quantifiers are indeed assumed to exist and are available at the right point in the derivation.

As a first concrete example, we investigate the interaction between negation and quantification. We prove

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

The subject of the judgment above is a proposition, assuming τ *type* and $x{:}\tau \vdash A(x)$ *prop*. Since all quantifiers range over the same type τ, we will omit the type label from quantification in all propositions below. The reader should keep in mind that this is merely a shorthand. Furthermore, we will not explicitly state the assumption about the propositional or predicate parameters such as $A(x)$ and omit the judgment *true* for the sake of brevity.

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

The two-dimensional notation for derivations becomes difficult to manage for large proofs, so we extend the linear notation used in the Tutch proof checker. We use the following concrete syntax.

$\forall x{:}\tau.\ A(x)$	`!x:t. A(x)`
$\exists x{:}\tau.\ A(x)$	`?x:t. A(x)`
$c:\tau$	`c : t`

The quantifiers $\forall$ and $\exists$ act like a prefix operator with minimal binding strength, so that

$$\forall x{:}\tau.\ A(x) \supset B$$

is the same as

$$\forall x{:}\tau.\ (A(x) \supset B).$$

One complication introduced by existential quantification is that the elimination rule introduces two new assumptions, $c : \tau$ and $A(c)$ *true*. In order to distinguish between inferred and assumed judgments, new assumptions are separated by commas and terminated by semicolon. Under these conventions, the four rules for quantification take the following form:

Introduction	Elimination

`c : t;`
`A(c);`
`?x:t. A(x);`
	`?x:t. A(x);`
	`[c : t, A(c);`
	`  ... ;`
	`B];`
	`B;`

`[c : t;`
`  ... ;`
`A(c)];`
`!x:t. A(x)`
	`!x:t. A(x);`
	`c : t;`
	`A(c);`

We use `c` as a new parameter to distinguish parameters more clearly from bound variables. Their confusion is a common source of error in first-order reasoning. And we have the usual assumption that the name chosen for c must be new (that is, may not occur in $A(x)$ or B) in the existential elimination and universal introduction rules.

Below we restate the proof from above in the linear notation.

```
[ ?x:t. ~A(x);
  [ !x:t. A(x);
    [ c : t, ~A(c);
      A(c);
      F ];
    F ];
  ~!x:t. A(x) ];
(?x:t. ~A(x)) => ~!x:t. A(x);
```

The opposite implication does not hold: even if we know that it is impossible that $A(x)$ is true for every x, this does not necessarily provide us with enough information to obtain a witness for $\exists x.\ A(x)$.

Now we return to showing that $(\neg\forall x.\ A(x)) \supset \exists x.\ \neg A(x)$ *true* is not derivable. We search for a normal proof, which means the first step in the bottom-up

construction is forced and we are in the state

$$\dfrac{\dfrac{}{\neg\forall x.\ A(x)\downarrow}\,u}{\begin{array}{c}\vdots\\ \exists x.\ \neg A(x)\uparrow\end{array}}{\scriptstyle\supset I^u}$$
$$(\neg\forall x.\ A(x))\supset\exists x.\ \neg A(x)\uparrow$$

At this point it is impossible to apply the existential introduction rule, because no witness object of type τ is available. So we can only apply the implication elimination rule, which leads us to the following situation.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\forall x.\ A(x)\downarrow}\,u}{\begin{array}{c}\vdots\\ \forall x.\ A(x)\uparrow\end{array}}\quad\dfrac{}{\neg\forall x.\ A(x)\downarrow}\,u}{\bot\downarrow}{\scriptstyle\supset E}}{\dfrac{}{\exists x.\ \neg A(x)\uparrow}\,{\scriptstyle\bot E}}{\scriptstyle\supset I^u}$$
$$(\neg\forall x.\ A(x))\supset\exists x.\ \neg A(x)\uparrow$$

Now we can either repeat the negation elimination (which leads nowhere), or use universal introduction.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\neg\forall x.\ A(x)\downarrow}\,u\quad\dfrac{}{c:\tau}}{\begin{array}{c}\vdots\\ A(c)\uparrow\end{array}}}{\forall x.\ A(x)\uparrow}\,{\scriptstyle\forall I^c}\quad\dfrac{}{\neg\forall x.\ A(x)\downarrow}\,u}{\bot\downarrow}{\scriptstyle\supset E}}{\dfrac{}{\exists x.\ \neg A(x)\uparrow}\,{\scriptstyle\bot E}}{\scriptstyle\supset I^u}$$
$$(\neg\forall x.\ A(x))\supset\exists x.\ \neg A(x)\uparrow$$

The only applicable rule for constructing normal deductions now is again the implication elimination rule, applied to the assumption labeled u. This leads to the identical situation, except that we have an additional assumption $d:\tau$ and try to prove $A(d)\uparrow$. Clearly, we have made no progress. Therefore the given proposition has no normal proof and hence, by the completeness of normal proofs, no proof.

As a second example, we see that $(\forall x.\ A(x))\supset\exists x.\ A(x)$ *true* does not have a normal proof. After one forced step, we have to prove

$$\dfrac{\forall x.\ A(x)\downarrow}{\begin{array}{c}\vdots\\ \exists x.\ A(x)\uparrow\end{array}}$$

At this point, no rule is applicable, since we cannot construct *any* term of type τ. Intuitively, this should make sense: if the type τ is empty, then we cannot prove $\exists x{:}\tau.\ A(x)$ since we cannot provide a witness object. Since we make no assumptions about τ, τ may in fact denote an empty type, the above is clearly false.

In classical first-order logic, the assumption is often made that the domain of quantification is non-empty, in which case the implication above is true. In type theory, we can prove this implication for specific types that are known to be non-empty (such as `nat`). We can also model the standard assumption that the domain is non-empty by establishing the corresponding hypothetical judgment:

$$c : \tau \vdash (\forall x{:}\tau.\ A(x)) \supset \exists x{:}\tau.\ A(x)$$

We just give this simple proof in our linear notation.

```
[ c : t;
  [ !x:t. A(x);
    A(c);
    ?x:t. A(x) ];
  (!x:t. A(x)) => ?x:t. A(x)];
```

We can also discharge this assumption to verify that

$$\forall y.\ ((\forall x.\ A(x)) \supset \exists x.\ A(x))\ \mathit{true}$$

without any additional assumption. This shows that, in general, $\forall y.\ B$ is not equivalent to B, even if y does not occur in B! While this may be counterintuitive at first, the example above shows why it must be the case. The point is that while y does not occur in the *proposition*, it does occur in the *proof* and can therefore not be dropped.

4.4 Proof Irrelevance

So far, we have carefully designed the proof term language so that we can reconstruct the original deduction, maintaining a bijection. In many contexts this will give us too much information. Returning to one of our motivating examples,

$$\forall x{:}\mathsf{nat}.\ \exists y{:}\mathsf{nat}.\ y > x \wedge \mathsf{prime}(y)$$

we can see from what we have developed so far, that the computational content of a constructive proof of this proposition will be a function from a natural number x that returns a pair consisting of a witness p and a proof that $p > x \wedge \mathsf{prime}(p)$ which is again a pair of proofs. While it may be important for us to know that these last two proofs exist, in practice we may only be interested in the prime p, and not the proof that it is indeed greater than x or a prime number.

Our objective in this section is to find means to selectively hide portions of a proof. In one direction, this will allows us to extract only part of the computational content of a proof, making the resulting code much more efficient to execute. In the other direction, it will allow us write proof-agnostic functions and reason about them logically.

We will use all concepts and techniques we have developed so to achieve this goal. The basic idea is to introduce a new proposition $[A]$, pronounced "*bracket A*", for any proposition A. $[A]$ should be true if A is true, except that we intend to erase the proof before using it computationally. A first cut at the introduction rule would be

$$\frac{A\ \mathit{true}}{[A]\ \mathit{true}}\ []I?$$

The corresponding elimination rule

$$\frac{A\ \mathit{true}}{[A]\ \mathit{true}}\ []E?$$

is sound with respect to the introduction rule, but unfortunately fails to capture the intent of the type $[A]$. It says that if we have a proof of $[A]$ (which will be erased at runtime) then we can get a proof of A (which will be needed at runtime). It is evident that after erasing the proof of $[A]$ we would have to make up a proof of A out of thin air, which is not possible in general.

In order to capture this, we need a new judgment $A\ \mathit{irr}$ (pronounced "A *irrelevant*") which says that A is true but its proof is not computationally available. The revised elimination rule exploits this judgment to express that if we know $[A]\ \mathit{true}$ we know that A is true, but that we cannot use its proof computationally.

$$\frac{[A]\ \mathit{true} \quad \begin{array}{c}\dfrac{}{A\ \mathit{irr}}\ u\\ \vdots \\ C\ \mathit{true}\end{array}}{C\ \mathit{true}}\ []E^u$$

How do we use assumptions $A\ irr$? From the discussion above it should be clear that it is must remain too weak to prove $A\ true$: the latter requires a computationally relevant proof term but the former cannot offer one. However, when we are trying to prove $[C]\ true$, then we should be allowed to use assumptions $A\ irr$ because a proof of $[C]$ will be erased. The corresponding rule is a bit difficult to express in the two-dimensional format. It says that at the point we introduce $[C]$ we transform every assumption $A\ irr$ to a corresponding assumptione $A\ true$.

$$\dfrac{\dfrac{\overline{A_1\ irr}^{\,u_1}}{A_1\ true} \quad \cdots \quad \dfrac{\overline{A_n\ irr}^{\,u_n}}{A_n\ true}}{\begin{array}{c}\vdots\\ C\ true\end{array}}{}_{[]I}}{[C]\ true}$$

Before carrying out some examples, let us make sure that the rules are locally sound and complete. First, the local reduction which witnesses local soundness.

$$\dfrac{\dfrac{\dfrac{\overline{A_1\ irr}^{\,u_1}}{A_1\ true} \quad \cdots \quad \dfrac{\overline{A_n\ irr}^{\,u_n}}{A_n\ true}}{\begin{array}{c}\mathcal{D}\\ A\ true\end{array}}{}_{[]I}}{[A]\ true} \quad \dfrac{\overline{A\ irr}^{\,u}}{\begin{array}{c}\mathcal{E}\\ C\ true\end{array}}}{C\ true}{}_{[]E^u} \quad \Longrightarrow_R \quad \dfrac{\dfrac{\overline{A_1\ irr}^{\,u_1}}{A_1\ true} \quad \cdots \quad \dfrac{\overline{A_n\ irr}^{\,u_n}}{A_n\ true}}{\begin{array}{c}\mathcal{D}\\ A\ true\end{array}}{}_{u}}{\begin{array}{c}\mathcal{E}\\ C\ true\end{array}}$$

The operation on the right is again not well represented, visually. It requires a new substitution principle to substitute a deduction of $A\ true$ for assumptione $A\ irr$. This works because the assumption $A\ irr$ can only be used when it is promoted to an assumption $A\ true$, at which place we can use the ordinary substitution principle. We will see more formally how this works when we have proof terms below, and hypotheses are written in a localized form.

The local expansion is much simpler.

$$\begin{array}{c}\mathcal{D}\\ [A]\ true\end{array} \quad \Longrightarrow_E \quad \dfrac{\begin{array}{c}\mathcal{D}\\ [A]\ true\end{array} \quad \dfrac{\dfrac{\overline{A\ irr}^{\,u}}{A\ true}}{[A]\ true}{}_{[]I}}{[A]\ true}{}_{[]E^u}$$

Here, $A\ true$ is available to prove the premise of the $[]I$ rule because applying this rule promotes the assumption $A\ irr$.

We consider three examples. The first proves that $A \supset [A]$, that is, we can

forget computational content if we so choose.

$$\dfrac{\dfrac{\dfrac{}{A\ true}\,u}{[A]\ true}\,[\,]I}{A \supset [A]\ true}\,\supset I^u$$

The second one shows that we cannot prove $[A] \supset A$, that is, we cannot spanteously create computational content,

$$\dfrac{\dfrac{\dfrac{}{A\ true}\,u \qquad \begin{array}{c}\dfrac{}{A\ irr}\,w\\ \vdots \\ A\ true\end{array}}{A\ true}\,[\,]E^w}{[A] \supset A\ true}\,\supset I^u$$

Of course, this is not a convincing argument that we cannot prove $[A] \supset A$, but if we also knew that we can always find a proof by using introduction rules from below and elimination rules from above, then indeed there cannot be any proof.

Finally, we can distribute brackets over implication.

$$\dfrac{\dfrac{\dfrac{}{[A \supset B]\ true}\,u \qquad \dfrac{\dfrac{}{[A]\ true}\,w \qquad \dfrac{\dfrac{\dfrac{}{A \supset B\ irr}\,u' \qquad \dfrac{}{A\ irr}\,w'}{A \supset B\ true} \supset E}{B\ true}}{[B]\ true}\,[\,]I}{[B]\ true}\,[\,]E^{w'}}{[A] \supset [B]\ true}\,[\,]E^{u'}}{[A \supset B] \supset [A] \supset [B]\ true}\,\supset I^w}\,\supset I^u$$

Now we move on to proof terms. We write hypotheses in localized form, $\Gamma \vdash M : A$. There are two forms of hypothesis: $u{:}A$ which labels an assumption $A\ true$, and $[u]{:}A$, which labels an assumption $A\ irr$. The brackets around u indicate that it cannot be used directly, but only after a $[\,]$ introduction, which "unlocks" the variable $u{:}A$. In order to describe this process we define the process of *promotion*, written $\Gamma^{\oplus}$.

$$\begin{array}{lcl}(\cdot)^{\oplus} & = & \cdot \\ (\Gamma, u{:}A)^{\oplus} & = & \Gamma^{\oplus}, u{:}A \\ (\Gamma, [u]{:}A)^{\oplus} & = & \Gamma^{\oplus}, u{:}A\end{array}$$

We then have the following two rules.

$$\dfrac{\Gamma^{\oplus} \vdash M : A}{\Gamma \vdash [M] : [A]}\,[\,]I \qquad \dfrac{\Gamma \vdash M : [A] \quad \Gamma, [u]{:}A \vdash N : C}{\Gamma \vdash \mathbf{let}\ [u] = M\ \mathbf{in}\ N : C}\,[\,]E^u$$

What the hypothesis promotion achieves is, intuitively, that a variable $[u]{:}A$ can only be used inside brackets in a term M and not outside. For example:

$$\begin{array}{rcl} \lambda u.\ [u] & : & A \supset [A] \\ \lambda f.\ \lambda x.\ \mathbf{let}\ [f'] = f\ \mathbf{in}\ \mathbf{let}\ [x'] = x\ \mathbf{in}\ [f'\ x'] & : & [A \supset B] \supset [A] \supset [B] \end{array}$$

are well-typed. The first, because we are free to use u (which stands for A *true*) inside brackets, where it will not be used computationally, and the second because both f' and x' are used only inside brackets, where they will not be used computationally. On the other hand

$$\lambda u.\ \mathbf{let}\ [u'] = u\ \mathbf{in}\ u' \quad \not: \quad [A] \supset A$$

because u' is incorrectly used *outside* a bracket context.

Local reductions and expansions, as well as the substitution principle, are now much clearer on proof terms.

$$\textit{If}\ \Gamma^{\oplus} \vdash M : A\ \textit{and}\ \Gamma, [u] : A, \Gamma' \vdash N : C\ \textit{then}\ \Gamma, \Gamma' \vdash [M/u]N : C.$$

This is correct because the only place where $[u]{:}A$ can be used in the second deduction is underneath a $[\,]$ constructor where the context is promoted to $\Gamma^{\oplus}, \Gamma'^{\oplus}$ so that the ordinary substitution principle applies.

$$\begin{array}{lcl} \mathbf{let}\ [u] = [M]\ \mathbf{in}\ N & \Longrightarrow_R & [M/u]N \\ M : [A] & \Longrightarrow_E & \mathbf{let}\ [u] = M\ \mathbf{in}\ [u] \end{array}$$

We now go through a few more examples. In order to write these examples more compactly, we assume that every assumption $v{:}[A]$ is immediately decomposed into $[u]{:}A$, and, moreover, u is longer used. In that case, we can write $\lambda[u].\ M$ instead of $\lambda v.\ \mathbf{let}\ [u] = v\ \mathbf{in}\ M$.

First, we consider variants of the following proposition:

$$(\exists x.\ A(x) \wedge B(x)) \supset (\exists x.\ B(x) \wedge A(x))$$

The starting proof of this is straightforward:

$$\lambda u.\ \mathbf{let}\ \langle y, w\rangle = u\ \mathbf{in}\ \langle y, \langle \mathbf{snd}\, w, \mathbf{fst}\, w\rangle\rangle$$

Hiding in proof information in the output is consistent.

$$\lambda u.\ \mathbf{let}\ \langle y, w\rangle = u\ \mathbf{in}\ \langle y, [\langle \mathbf{snd}\, w, \mathbf{fst}\, w\rangle]\rangle : (\exists x.\ A(x) \wedge B(x)) \supset (\exists x.\ [B(x) \wedge A(x)])$$

We can still hide consistently even if the input proofs (of $A(x)$ and $B(x)$) are not available at runtime.

$$\lambda u.\ \mathbf{let}\ \langle y, [w]\rangle = u\ \mathbf{in}\ \langle y, [\langle \mathbf{snd}\, w, \mathbf{fst}\, w\rangle]\rangle : (\exists x.\ [A(x) \wedge B(x)]) \supset (\exists x.\ [B(x) \wedge A(x)])$$

This is correct because the only occurrences of w in its scope are inside brackets. Erasing w will therefore not lead to any dangling variables, that is, variables

that would be incorrectly assumed to be available at runtime. However, if we also hide the witness y then the result is no longer well-formed.

$$\lambda u.\ \mathbf{let}\ [\langle y, w\rangle] = u\ \mathbf{in}\ \langle y, [\langle \mathbf{snd}\, w, \mathbf{fst}\, w\rangle]\rangle \not: ([\exists x.\ A(x) \wedge B(x)]) \supset (\exists x.\ [B(x) \wedge A(x)])$$

The problem here is that y is bound inside the brackets but used outside. If we erased all the content inside the brackets we would have

$$\lambda u.\ \mathbf{let}\ [\,] = u\ \mathbf{in}\ \langle y, [\,]\rangle \not: [\,] \supset (\exists x.\ [\,])$$

which is not meaningful as a program even if we interpret $[\,]$ at $\top$ in the proposition and the unit element $\langle\,\rangle$ in the term.

It is even possible to be more fine-grained. For example, we can hide the proof of $A(x)$ in the antecedent of the implication of we also hide in the succedent.

Another set of examples comes from a specfication of graph reachability. We specify that for any two nodes x and y in a graph, either there is a path p connecting the two nodes or not.

$$\forall x.\ \forall y.\ (\exists p.\ \mathsf{path}(p, x, y)) \vee \neg(\exists p.\ \mathsf{path}(p, x, y))$$

A proof of this, when viewed computationally, will be a function taking nodes x and y as input and returning either $\mathbf{inl}\, M$ or $\mathbf{inr}\, N$. In the first case, a path exists, and M is a pair consisting of the path p and a proof that it connects x and y in the graph. In the second case, no path exists, and M is a function which derives a contradiction from any candidate path given to it as an argument.

One natural refinement of this specification is to hide the proof that p is indeed a valid path connecting x and y (which can easily verify this ourselves) and the proof that there is no path connecting x and y. For the latter, we would trust the proof (because, say, have verified it before erasing it). To capture this we would write

$$\forall x.\ \forall y.\ (\exists p.\ [\mathsf{path}(p, x, y)]) \vee [\neg(\exists p.\ \mathsf{path}(p, x, y))]$$

Now the function extracted from this proof would at runtime return either $\mathbf{inl}\, \langle p, [\,]\rangle$ (that is, the path connecting x and y) or $\mathbf{inr}\, [\,]$ to indicate there is no such path.

We can take this one step further, also hiding the path itself. In this case, the function returns either $\mathbf{inl}\, [\,]$ if there is a path connecting x and y and $\mathbf{inr}\, [\,]$ if there is none.

$$\forall x.\ \forall y.\ [\exists p.\ \mathsf{path}(p, x, y)] \vee [\neg(\exists p.\ \mathsf{path}(p, x, y))]$$

Summary. We present a summary of the rules for proof irrelevance, using the local form for hypothesis. We have a new form of judgment, $A\ \mathit{irr}$ which means that A is true, but the evidence for that is computationally irrelevant, that is,

may be erased before executing the program. We arrange that $A\ \mathit{irr}$ is only used as a hypothesis. We can *promote* such assumptions using the $\Delta^{\oplus}$ operation:

$$\begin{array}{lcl}(\cdot)^{\oplus} & = & \cdot \\ (\Delta, A\ \mathit{true})^{\oplus} & = & \Delta^{\oplus}, A\ \mathit{true} \\ (\Delta, A\ \mathit{irr})^{\oplus} & = & \Delta^{\oplus}, A\ \mathit{true}\end{array}$$

Then the introduction and elimination rules are

$$\frac{\Delta^{\oplus} \vdash A\ \mathit{true}}{\Delta \vdash [A]\ \mathit{true}}\ []I \qquad \frac{\Delta \vdash [A]\ \mathit{true} \quad \Delta, A\ \mathit{irr} \vdash C\ \mathit{true}}{\Delta \vdash C\ \mathit{true}}\ []E^u$$

In the presence of proof terms, we write $[u]{:}A$ as a labeling of the assumption $A\ \mathit{true}$. Then the promotion operation on labeled contexts Γ becomes

$$\begin{array}{lcl}(\cdot)^{\oplus} & = & \cdot \\ (\Gamma, u{:}A)^{\oplus} & = & \Gamma^{\oplus}, u{:}A \\ (\Gamma, [u]{:}A)^{\oplus} & = & \Gamma^{\oplus}, u{:}A\end{array}$$

With proof terms, the introduction and elimination rules then read as follows:

$$\frac{\Gamma^{\oplus} \vdash M : A}{\Gamma \vdash [M] : [A]}\ []I \qquad \frac{\Gamma \vdash M : [A] \quad \Gamma, [u]{:}A \vdash N : C}{\Gamma \vdash \mathbf{let}\ [u] = M\ \mathbf{in}\ N : C}\ []E^u$$

Lectures 9 and 10: Classical Logic

15-317: Constructive Logic
Dan Licata

September 23-25, 2008

In these two lectures, we will discuss *classical logic*—which is what people were talking about when they taught you about (unqualified) "logic" in other classes—and its relationship to constructive logic—which is what we've covered so far this semester. We use *intuitionistic logic* as a synonym for "constructive logic"; this is helpful so we can use the abbreviations IL (intuitionistic) and CL (classical). We will answer three questions:

- What is classical logic?
- What is the relationship with intuitionistic logic?
- What is the computational meaning of classical proofs?

1 What is classical logic?

Classical logic differs from intuitionistic logic in that classical logic admits *the law of the excluded middle* (LEM), which says that every proposition is either true or false. The simplest way to describe classical logic is to take the natural deduction rules we have seen so far and add LEM:

$$\frac{}{(A \vee \neg A)\ \mathsf{true}}\ LEM$$

Equivalently, we could instead add *double-negation elimination*, which says that $\neg\neg A$ implies A:

$$\frac{\neg\neg A\ \mathsf{true}}{A\ \mathsf{true}}\ DNE$$

Note that double-negation introduction ($A \supset \neg\neg A$) is intuitionistically true.

It's easy to see that these are equivalent, by taking IL + LEM and deriving DNE and vice versa. For example, to derive DNE, assume $\neg\neg A$. Next, we use LEM on A to get $A \vee \neg A$, and therefore have two cases in which we have to show A. In the first, it's true by assumption. In the second, we have $\neg A$ and $\neg\neg A$, a contradiction, so we get A by $\bot$-elimination.

In the other direction, here is an annotated Tutch-style proof using DNE:

```
[u : ~(A | ~A)
 [v : A
  inl v : A | ~A
  u (inl v) : F]
 fn v => u (inl v) : ~A
 inr (fn v => u (inl v)) : A | ~A
 u (inr (fn v => u (inl v))) F]
fn u => u (inr (fn v => u (inl v))) : ~~(A | ~A)
DNE (fn u => u (inr (fn v => u (inl v)))) : (A | ~A)
```

A couple of things to note:

- Everything before the last line is a perfectly good constructive proof of `~~(A | ~A)`.
- The proof term `DNE (fn u => u (inr (fn v => u (inl v))))`. Doesn't tell you which of `A` or `~A` is true. Instead, it assumes `u : ~(A | ~A)` and proves a contradiction. It does this by calling `u` with a proof of `~A`. How does this proof work? When given an `A`, it calls `u` again, this time with that very proof that it was given! To anthropomorphize a little: the first time we call `u`, we bluff that the answer is `~A`. If `u` ever calls our bluff, it must do so by giving us an `A` and requesting a proof of `F`. In that case, we say "dang, you caught me! forget about what I said before—it was `A` after all!". We will talk more about this "time travel" interpretation of classical proofs below.

1.1 A Better Proof Theory

Hopefully you had a bad feeling about the rules LEM and DNE above: they violate some of our principles for what inference rules should look like. In particular, they mention more than one connective, and they are neither intro nor elim rules. Violating these principles can invalidate local soundness and completeness (and their global versions, cut elimination and identity). In this section, we give a cleaner presentation of classical

logic by accounting for DNE at the level of judgments, rather than propositions.

1.1.1 New judgments

To do so, we need two new judgments:

- $\#$ (contradiction)
- $A\,\mathsf{false}$

The judgment $\#$is a judgmental analogue of $\bot$, the false proposition, whereas $A\,\mathsf{false}$ is a judgmental analogue of $A \supset \bot$. The rules for these judgements are as follows:

$$\frac{\#}{J}\ \#E \qquad \frac{\begin{array}{c}\overline{A\,\mathsf{true}}\ u\\ \vdots\\ \#\end{array}}{A\,\mathsf{false}}\ fI^u \qquad \frac{A\,\mathsf{false} \quad A\,\mathsf{true}}{\#}\ fE$$

The first says that from a contradiction, you can conclude any judgment J. The second says that A is false if assuming it's true gives a contradiction. The third says that A being both true and false is contradictory.

A small technical matter: because we now have multiple judgments, we need to change the rules we had before that conclude an arbitrary proposition $C\,\mathsf{true}$ ($\bot$-elimination, $\vee$-elimination) to instead conclude an arbitrary *judgment* J. I.e.

$$\frac{\bot\,\mathsf{true}}{J}\ \bot E \qquad \frac{A \vee B\,\mathsf{true} \quad \begin{array}{c}\overline{A\,\mathsf{true}}\ u\\ \vdots\\ J\end{array} \quad \begin{array}{c}\overline{A\,\mathsf{true}}\ u\\ \vdots\\ J\end{array}}{J}\ \vee E^{u,v}$$

1.1.2 Negation

Using these judgments, we can give a primitive account of negation, rather than treating $\neg A$ as a notational definition for $A \supset \bot$:

$$\frac{A\,\mathsf{false}}{\neg A\,\mathsf{true}}\ \neg I \qquad \frac{\neg A\,\mathsf{true} \quad \begin{array}{c}\overline{A\,\mathsf{false}}\ k\\ \vdots\\ J\end{array}}{J}\ \neg E^k$$

We wrote the rule $\neg E^k$ in the style of $\vee$-elim, but we could equivalently have given a rule that looks like $\wedge$-elim: from $\neg A$ true conclude A false. These two rules are equivalent in our setting here (though keep this distinction in mind when we talk about focusing in a couple of weeks).

1.1.3 Classical logic

Thus far, all of our new judgments are OK constructively. One way to see this to prove that the new judgements can be eliminated, treating $\#$ as a notational definition for $\bot$ true and A false as $A \supset \bot$ true, and checking that all the new rules are derivable. This shows that we have not fundamentally changed the meaning of truth: the truth of $\neg A$, a new proposition, depends on contradiction and falsehood, but the truth of existing propositions such as $A \vee B$ is unchanged.

To get classical logic, we add the judgmental version of DNE:

$$\dfrac{\dfrac{}{A \text{ false}}\ k \\ \vdots \\ \dfrac{\#}{A \text{ true}}}{}\ DNE^k$$

This rule does change the meaning of truth for existing propositions, because the A in the conclusion might be, e.g., a disjunction. Note the symmetry between DNE and fI: classical logic is more symmetric than intuitionistic logic, where truth means something stronger than merely not being false.

Exercise. Give a derivation of $(\neg(A \wedge B)) \supset \neg A \vee \neg B$ true. In Homework 2, you saw that this De Morgan principle was not intuitionistically true, but it is classically true.

Here's a Tutch-like proof to guide you along:

```
[~(A & B) true
 [A & B false
  [~A | ~B false
   [A true
    [B true
     A & B true
     #]
    B false
    ~B true
```

```
    ~A | ~B true
    #]
   A false
   ~A true
   ~A | ~B true
   #]
  ~A | ~B true]
 ~A | ~B true]
~(A & B) => ~A | ~B true
```

2 What is the relationship with intuitionistic logic?

Let's write $\Gamma \vdash_c A\,\mathsf{true}$ for classical truth and $\Gamma \vdash_i A\,\mathsf{true}$ for intuitionistic truth, where we put a context Γ of hypotheses in the judgement form rather than using the two-dimensional notation for hypotheses.

It's easy to prove that:

If $\Gamma \vdash_i A\,\mathsf{true}$ then $\Gamma \vdash_c A\,\mathsf{true}$.

This says that if an intuitionist asserts A, a classicist will believe him, interpreting A classically. Informally, the move from intuitionistic to classical logic consisted of adding new inference rules, so whatever is true intuitionistically must be true classically. This can be formalized as a proof by rule induction on $\Gamma \vdash_i A\,\mathsf{true}$.

Of course, the opposite entailment does not hold (take A to be the law of the excluded middle, or double-negation elimination). However, it is possible to *translate* propositions in such a way that, if a proposition is classically true, then its translation is intuitionistically true. That is, the intuitionist does not believe what the classicist says at face value, but he can figure out what the classicist really meant to say, by means of a *double-negation translation*. The translation inserts enough double-negations into the proposition A that the classical uses of the DNE rule are intuitionistically permissible.

We will use the "Gödel-Gentzen negative translation", which is defined by a function $A^* = A'$ from classical propositions to intuitionistic propositions. On the intuitionistic side, we use the usual notational definition of

$\neg A = (A \supset \bot)$.

$$\begin{array}{rcl}
(\top)^* & = & \top \\
(\bot)^* & = & \bot \\
(A \wedge B)^* & = & A^* \wedge B^* \\
(A \vee B)^* & = & \neg\neg(A^* \vee B^*) \\
(A \supset B)^* & = & (A^* \supset B^*) \\
(\neg A)^* & = & \neg A^* \\
(P)^* & = & \neg\neg P
\end{array}$$

That is, the classicist and the intuitionistic agree about the meaning of all of the connectives except $\vee$ and atomic propositions P. From an intuitionistic point of view, when a classicist says $A \vee B$, he *really* means $\neg\neg(A^* \vee B^*)$, an intuitionistically weaker statement. Thus, **intuitionistic logic is more precise**, because you can say $A \vee B$, if that's the case, or $\neg\neg(A \vee B)$ if you need classical reasoning to do the proof. There is no way to express intuitionistic disjunction in classical logic. If an intuitionist says A to a classicist, and then the classicist repeats it back to him, it will come back as a weaker statement A^*.

On the other hand, the translation has the property that A and A^* are classically equivalent. If a classicist says something to an intuitionist, and then the intuitionist repeats it back to him, the classicist won't know the difference: intuitionistic logic makes finer distinctions.

As an aside, there are several other ways of translating classical logic into intuitionistic logic, which make different choices about where to insert double-negations. Different translations do different things to proofs, which turns out to have interesting consequences for programming.

How do we verify that this translation does the right thing? First, we need to lift the translation to judgments as follows:

$$\begin{array}{rcl}
(A\,\mathsf{true})^* & = & A^*\,\mathsf{true} \\
(A\,\mathsf{false})^* & = & (A \supset \bot)\,\mathsf{true} \\
(\#)^* & = & \bot\,\mathsf{true}
\end{array}$$

and to contexts Γ by translating each assumption in the context. Then we can prove:

Theorem 1. $\Gamma \vdash_c J$ *iff* $\Gamma^* \vdash_i J^*$.

Proof. The "only if" direction is a consequence of two lemmas mentioned above:

1. If $\Gamma \vdash_i J$ then $\Gamma \vdash_c J$
2. For all A, Γ, $\Gamma \vdash_c (A \supset A^*) \wedge (A^* \supset A)\,\mathsf{true}$.

The first is proved by rule induction on $\Gamma \vdash_i J$, the second by induction on A.

The "if" direction is proved by induction on $\Gamma \vdash_c J$. The hard case is eliminating uses of the DNE rule:

$$\frac{\Gamma, A\,\mathsf{false} \vdash_c \#}{\Gamma \vdash A\,\mathsf{true}}\ DNE$$

The inductive hypothesis is

$$(\Gamma, A\,\mathsf{false})^* \vdash_c (\#)^*$$

Expanding the definition of the translation gives:

$$\Gamma^*, (A^* \supset \bot)\,\mathsf{true} \vdash_c \bot\,\mathsf{true}$$

By implication introduction, this gives

$$\Gamma^* \vdash_i \neg\neg A^*$$

On the other hand, translating the conclusion of the rule, we need to show that

$$\Gamma^* \vdash_i A^*$$

.

Uh-oh! This is an instance of double-negation elimination, which we don't have intuitionistically! So why does the translation work?

The key is that we only need double-negation elimination *for the target of the translation*. That is, we need the translation to have the property that $\neg\neg(A^*) \supset A^*$. This lemma is true for the translation defined above (you can prove it by induction on A). But if, for example, we forgot to double-negate disjunction or atoms, the property would not be true. The lemma that

$$\neg\neg(A^*) \supset A^*$$

is how we tell that we've added enough double-negations to allow all the classical reasoning that we need.

□

3 Intermezzo: Truth tables

Once upon a time, someone told you that to check whether a proposition is (classically) true, you build a truth table. What are the reasoning principles behind truth tables?

1. To prove $P\ \mathsf{prop} \vdash A\ \mathsf{true}$, it suffices to prove $[\top/P]A\ \mathsf{true}$ and $[\bot/P]A\ \mathsf{true}$.
2. You compute the truth value of a connective from the truth values of its components, using equations like

$$\begin{array}{lcl} \top \supset \top & \equiv & \top \\ \top \supset \bot & \equiv & \bot \\ \bot \supset \top & \equiv & \top \\ \bot \supset \bot & \equiv & \bot \end{array}$$

The equations in (2) are true, constructively and classically, if you interpret $A \equiv B$ as $(A \supset B) \wedge (B \supset A)$.

But what about (1), the idea that to prove a proposition A, you can distinguish cases on the truth of an atomic proposition P appearing in A?. As you might expect, this reasoning is valid classically but not constructively. By the law of the excluded middle, we can case-analyze $P \vee \neg P$:

$$\dfrac{\dfrac{}{P \vee \neg P}\ LEM \qquad \begin{array}{c}\overline{P\ \mathsf{true}}^{\,u} \\ \vdots \\ J\end{array} \qquad \dfrac{\overline{\neg P\ \mathsf{true}}^{\,v} \qquad \begin{array}{c}\overline{P\ \mathsf{false}}^{\,k} \\ \vdots \\ J\end{array}}{J}\ \neg E^{k}}{J}\ \vee E^{u,v}$$

Thus, to show J, it suffices to show $P\ \mathsf{true} \vdash J$ and $P\ \mathsf{false} \vdash J$.

So, to justify the truth-table reasoning, we just need the following lemma:

Lemma 2.

1. *For all J, if $[\top/P]J$ then $(P\ \mathsf{prop}, P\ \mathsf{true} \vdash J)$*
2. *For all J, if $[\bot/P]J$ then $(P\ \mathsf{prop}, P\ \mathsf{false} \vdash J)$*

Here's an intuition for why this is true: For the first part, everywhere the $\top I$ rule was used, we can instead use the assumption of $P\,\mathsf{true}$. For the second, everywhere the derivation uses $\bot E$ from a proof of $\bot\,\mathsf{true}$, we instead have a proof of $Ptrue$, which can be used with fE and $\#E$ to conclude anything.

4 What is the computational meaning of classical proofs?

4.1 Proof Terms

Let's annotate the above rules with proof terms:

$$\frac{M : \#}{\mathsf{abort}\, M : J} \qquad \frac{\begin{array}{c}\overline{u : A\,\mathsf{true}} \\ \vdots \\ M : \#\end{array}}{\mathsf{cont}\, u.M : A\,\mathsf{false}} \qquad \frac{M : A\,\mathsf{false} \quad N : A\,\mathsf{true}}{\mathsf{throw}\, M\, N : \#}$$

$$\frac{M : A\,\mathsf{false}}{\mathsf{not}\, M : \neg A\,\mathsf{true}} \qquad \frac{M : \neg A\,\mathsf{true} \quad \begin{array}{c}\overline{k : A\,\mathsf{false}} \\ \vdots \\ N : J\end{array}}{\mathsf{notcase}(M, k.N) : J}$$

$$\frac{\begin{array}{c}\overline{k : A\,\mathsf{false}} \\ \vdots \\ M : \#\end{array}}{\mathsf{letcc}(k.M) : A\,\mathsf{true}}$$

4.2 Programming with continuations

As you know, when you give a proof term assignment to some logical rules, the operator names (abort, throw, letcc, ...) are arbitrary. However, the names we've chosen here are fairly standard for the programming feature distinguishes classical proofs from constructive ones: *continuations*.

The term letcc(k.M) is short for "let the current continuation be k in M". What is the "current continuation"? It's all the work that's left to do in the rest of the program. In implementation terms, letcc gives the program access to its own control stack. In letcc, the current control stack gets packed up as value bound to k. Continuations are used by throwing them

a value, which forgets about the current execution context and runs a stack that you previously saved on that value. Continuations can be thrown to multiple times, which makes the control flow in a program with continuations very different than the traditional push/pop behavior that you get from function calls in intuitionistic logic. You might save a stack, run it on a value, go off and do something else for a while, and then come back to that stack again with a different value. Unlike languages without letcc, control stacks must be implemented as persistent data structures, not just as an ephemeral piece of mutable memory. Continuations are a very general mechanism, and can be used to implement other control forms, such as exceptions, coroutines, and threads.

As an example of programming with continuations, consider a function that multiples all the integers in a list. In writing this code, we'll assume that `intlist` and `int` are *propositions*, like they would be in ML, and that we can write pattern-matching functions over them. Here's a first version:

```
mult' : intlist => int
mult' [] = 1
mult' (x :: xs) = x * mult' xs
```

I.e., the multiplication of the empty list is `1`, and the multiplication of `x :: xs` is the head times the multiplication of the tail.

What happens when we call `mult' [1,2,3,0,4,5,...]` where the `...` is 700 billion[1] more numbers? It does a lot more work than necessary to figure out that the answer is 0. Here's a better version:

```
mult' : intlist => int
mult' [] = 1
mult' (0 :: xs) = 0
mult' (x :: xs) = x * mult' xs
```

This version checks for 0, and returns 0 immediately, and therefore does better on the list `[1,2,3,0,4,5,...]`.

But what about the reverse list `[...,5,4,0,1,2,3]`? This version still does all 700 billion multiplications on the way up the call chain, which could also be skipped.

We can do this using continuations:

```
mult xs = letcc k : int false in
  let
```

[1] this week's trendy really-large number to pull out of thin air

```
  mult' : intlist => int
  mult' [] = 1
  mult' (0 :: xs) = abort(throw k 0)
  mult' (x :: xs) = x * (mult' xs)
in throw k (mult' xs)
```

The idea is that we grab a continuation `k` standing for the evaluation context in which `mult` is called. Whenever we find a 0, we **immediately** jump back to this context, with no further multiplications. If we make it through the list without finding a zero, we throw the result of `mult'` to this continuation, returning it from `mult`.

Let's try to run `(mult [0,1,2]) + 5`. It's easiest to define evaluation for proofs of #, so we'll run this term against an initial continuation variable `halt : int false`.

```
      throw halt ((mult [0,1,2]) + 5)
⇒R    throw halt
       (letcc k in let ... in throw k (mult' [0,1,2])) + 5)
⇒R    (throw (cont u.throw halt (u + 5))
       (([(cont u.throw halt (u + 5))/k]mult') [0,1,2])
⇒R    (throw (cont u.throw halt (u + 5))
       (abort (throw (cont u.throw halt (u + 5)) 0)))
⇒R    (throw halt (0 + 5))
⇒R    (throw halt 5)
```

Some things to note:

- In the second reduction step, the whole expression enclosing the `letcc` is packed up as a `cont` and substituted for `k`.
- In the third reduction, we evaluate a `throw` by evaluating the proof of `A true`. Another choice would be to evaluate the proof of `A false`; these correspond to different *evaluation orders* for the programming language.
- In the fourth reduction, we have a `throw` whose true expression is fully evaluated, so we substitute this value into the continuation and forget the enclosing context.

What is the continuation assumption `halt`? This represents the initial continuation of the program (in practice, this might print the final value out to the user). We need this initial continuation assumption because there are no closed contradictions in classical logic!

Exercise. Recast the code for `mult` as a proof of $\forall x : intlist.\exists y : int.\top$, so `intlist` and `int` are treated as types of objects rather than as propositions.

As another example, returning to the proof of LEM above, we can now write it as:

```
letcc k : ~(A | ~A) in
 throw k (inr (not (cont v => k (inl v))))
```

Executing this code will resume the continuation `k` twice: first with `inr M`, and then, if `k` ever uses `M`, with `inl v` where `v` is the value that `k` supplies. The program "time travels" between different moments in its execution.

Exercise. Give proof terms for the following:

- $(A \supset B) \supset (\neg B \supset \neg A)$
- $(\neg B \supset \neg A) \supset (A \supset B)$

4.3 Continuation-Passing Style

Classical logic is a functional-programming language with letcc, and intuitionistic logic is a functional programming language without it. So what is the computational meaning of the double-negation translation A^*, which transforms classical forms into intuitionistic proofs? It is a transformation on programs that eliminates all uses of letcc, usually called a *continuation-passing style (CPS) translation*.

For example:

$$\begin{aligned}(\mathsf{intlist} \supset \mathsf{int})^* &= \mathsf{intlist}^* \supset \mathsf{int}^* \\ &= \neg\neg\mathsf{intlist} \supset \neg\neg\mathsf{int} \\ &= \neg\neg\mathsf{intlist} \supset \neg\mathsf{int} \supset \bot\end{aligned}$$

Here, a function that takes an intlist and returns an int is transformed into a function that:

1. takes an extra argument of type $\neg$int, representing the current continuation (hence, continuation-passing style)
2. never returns (because its result type is $\bot$)

CPS translation is used in compilers for several reasons: First, it reduces the problem of implementing a language with letcc to that of implementing a language without it. Second, even if you're compiling an intuitionistic language, there are reasons to CPS convert it: (1) The control flow that you have to implement is simpler, because functions call but never return. Consequently, there is no need for a control stack—or, more precisely, the control stack is represented as heap objects, just like all other values in the program. (2) CPS conversion makes certain optimizations, such as one called *tail-call optimization*, easier to do.

4.4 Running a CPS program

The CPS conversion of a program like `mult [1,2,3,4,5]` has type $\neg\mathsf{int} \supset \bot$. So how do you actually see what number it produced? One option is to extend the language with an initial continuation, as discussed above. Another is to revise the double-negation translation so use an arbitrary *answer type* in place of $\bot$. Let's define $\neg_\alpha A = A \supset \alpha$. Then the following double-negation translation works:

$$\begin{aligned}
(\top)^* &= \top \\
(\bot)^* &= \alpha \\
(A \wedge B)^* &= A^* \wedge B^* \\
(A \vee B)^* &= \neg_\alpha\neg_\alpha(A^* \vee B^*) \\
(A \supset B)^* &= (A^* \supset B^*) \\
(\neg A)^* &= \neg_\alpha A^* \\
(P)^* &= \neg_\alpha\neg_\alpha P
\end{aligned}$$

The proof is similar to above, but requires the additional lemma that for all A, $\alpha \supset A^*$.

For $\alpha = \bot$, the translation is the same as above. But what if we take $\alpha = \mathsf{int}$. Then

$$\begin{aligned}
\mathsf{int}^* &= \neg_{\mathsf{int}}\neg_{\mathsf{int}}\mathsf{int} \\
&= (\mathsf{int} \supset \mathsf{int}) \supset \mathsf{int}
\end{aligned}$$

Now we can supply the identity function `fn x => x` as the initial continuation, and get back an actual number.

Logically, this says that a classical proof of a base type P (from no hypotheses) determines an intuitionistic proof of P itself—*not* just the double-negation of P. The same device can be used to study other classes of propositions for which intuitionistic and classical logic agree.