Universität Innsbruck Winter Term 2010/2011

Lecture Notes

Logic (MSc)

Appendix to Notes for the Lecture in the Winter Term 2010/2011

Georg Moser

Winter 2010

This document has been produced with the help of KOMA-Script and LaTeX. For the intensive guidance on LaTeX, I'd like to express my sincere thanks to Christian Sternagel.

1

Digression: The Curry-Howard Isomorphism

In this optional chapter we consider the connection between proofs and programs in more detail. For that we describe the *Curry-Howard isomorphism* between intuitionistic natural deduction and the typed λ-calculus. This correspondence allows us to speak of programs and proofs interchangingly and transform or develop formalisms and methods in one area to apply it to the other. We restrict ourselvs to the bare essentials in presenting the Curry-Howard correspondence. For a complete account the reader is kindly referred to Larrecq and Makie, see [5].

1.1 A Problem with the Excluded Middle

In order to set the table for the presentation of the Curry-Howard isomorphism it is necessary to describe intuitionistic logic and the λ-calculus. In this section we give the usual motivating example of intuitionistic logic and present a calculus for this logic.

Theorem 1.1. *There are solutions of the equation $x^y = z$ with x and y irrational and z rational.*

Proof. We give a non-constructive proof. Clearly $\sqrt{2}$ is an irrational number. Consider $\sqrt{2}^{\sqrt{2}}$: One of the following two cases has to occur:

(i) $\sqrt{2}^{\sqrt{2}}$ is rational. In this case put

$$x = \sqrt{2} \qquad y = \sqrt{2} \qquad z = \sqrt{2}^{\sqrt{2}}$$

Clearly these settings solve the equation $x^y = z$. Thus the theorem is proven.

(ii) $\sqrt{2}^{\sqrt{2}}$ is irrational. In this case put

$$x = \sqrt{2}^{\sqrt{2}} \qquad y = \sqrt{2} \qquad z = (\sqrt{2}^{\sqrt{2}})^{\sqrt{2}} = \sqrt{2}^{\sqrt{2}\cdot\sqrt{2}} = 2$$

	introduction	*elimination*
$\wedge$	$\dfrac{E \quad F}{E \wedge F}\ \wedge\!:\mathsf{i}$	$\dfrac{E \wedge F}{E}\ \wedge\!:\mathsf{e}$ $\quad$ $\dfrac{E \wedge F}{F}\ \wedge\!:\mathsf{e}$
$\vee$	$\dfrac{E}{E \vee F}\ \vee\!:\mathsf{i}$ $\quad$ $\dfrac{F}{F \vee F}\ \vee\!:\mathsf{i}$	$\dfrac{E \vee F \quad \boxed{\begin{array}{c}E\\\vdots\\G\end{array}} \quad \boxed{\begin{array}{c}F\\\vdots\\G\end{array}}}{G}\ \vee\!:\mathsf{e}$
$\rightarrow$	$\dfrac{\boxed{\begin{array}{c}E\\\vdots\\F\end{array}}}{E \rightarrow F}\ \rightarrow\!:\mathsf{i}$	$\dfrac{E \quad E \rightarrow F}{F}\ \rightarrow\!:\mathsf{e}$

Figure 1.1: Intuitionistic Propositional Rules (Part I)

Again the equation $x^y = z$ is solved and the lemma is proven.

□

The problem with the above proof is that it is *non-constructive*: The statement of the theorem is existential: something should exist namely the numbers x, y, and z. Despite this the proof does not provide a method to actually *construct* these numbers. This is a serious problem if we want to extract a program out of the given proof which is exactly the point of the Curry-Howard correspondence. To overcome this problem we consider proofs of a specific form: *intuitionistic proofs*.

1.2 Natural Deduction for Intuitionistic Logic

In this section we introduce a formal system for intuitionistic logic and claim its soundness and completeness with respect to the standard Kripke-semantics. As the focus of this section is on the correspondence between proofs and programs we are not concerned with the semantics of intuitionistic logic here. (For more information see [3].) Note that the semantics of intuitionistic logic (even for the propositional case) is more complex than the one considered for classical (propositional) logic.

Below we give the natural deduction rules of intuitionistic logic, denoted as NJ. In the following the natural deduction rules as defined in [6] are denoted as NK. It suffices if we consider the propositional part only. The rules for the connectives $\neg$, $\vee$, $\wedge$, and $\rightarrow$ are given in Figure 4.1 and 4.2.

	introduction	*elimination*
$\neg$	$\dfrac{\begin{array}{|c|}\hline E \\ \vdots \\ \bot \\ \hline\end{array}}{\neg E}\ \neg\mathsf{:i}$	$\dfrac{F \quad \neg F}{\bot}\ \neg\mathsf{:e}$
$\bot$		$\dfrac{\bot}{F}\ \neg\mathsf{:e}$

Figure 1.2: Intuitionistic Proposition Rules (Part II)

The only difference between the classical rules and those given here is (apart from the omission of equality) is the absence of the double-negation rule:

$$\frac{\neg\neg F}{F}\ \neg\neg\mathsf{:e}$$

This seemingly small change has the effect that in NJ the *tertium non-datur* $F \vee \neg F$ is no longer derivable: $\mathsf{NJ} \nvdash F \vee \neg F$.

1.3 Typed λ-Calculus

In this section we (very) briefly introduce the typed λ-calculus. See [1] for extensive information on the untyped λ-calculus and see [2, 4] for background information on the typed system.

Definition 1.1. We define the set of *types* T as follows:

- a variable type: α, β, γ, ...
- if σ, τ are types, then $(\sigma \times \tau)$ is a (*product*) type
- if σ, τ are types, then $(\sigma \to \tau)$ is a (*function*) type

Definition 1.2. The *typed λ-terms* are defined as follows:

- any (typed) variable $x : \sigma$ is a (typed) term
- if $M : \sigma$, $N : \tau$ are terms, then $\langle M, N\rangle : \sigma \times \tau$ is a term
- if $M : \sigma \times \tau$ is a term, then $\mathsf{fst}(M) : \sigma$ and $\mathsf{snd}(M) : \tau$ are terms
- if $M : \tau$ is a term, $x : \sigma$ a variable,
  then the *abstraction* $(\lambda x^{\sigma}.M) : \sigma \to \tau$ is a term

– if $M : \sigma \to \tau$, $N : \sigma$ are terms, then the *application* $(MN) : \tau$ is a term.

Example 1.1. The following are (well-formed, typed) terms

$$\lambda fx.fx : (\sigma \to \tau) \to \sigma \to \tau \qquad \langle \lambda x.x, \lambda y.y \rangle : (\sigma \to \sigma) \times (\tau \to \tau) \ ,$$

but $\lambda x.xx$ cannot be typed!

Definition 1.3. The set of *free variables* of a term is defined as follows

– $\mathsf{FV}(x) = \{x\}$.

– $\mathsf{FV}(\lambda x.M) = \mathsf{FV}(M) - \{x\}$

– $\mathsf{FV}(MN) = \mathsf{FV}(\langle M, N \rangle) = \mathsf{FV}(M) \cup \mathsf{FV}(N)$.

– $\mathsf{FV}(\mathsf{fst}(M)) = \mathsf{FV}(\mathsf{snd}(M)) = \mathsf{FV}(M)$.

Occurrences of x in the scope of λ are called *bound*: $\lambda x.\mathbf{x}y(\lambda y.\mathbf{xy}(\lambda x.z))y$. This notion is made precise in the next definition.

Definition 1.4. The set of *bound variables* of a term is defined as follows

– $\mathsf{BV}(x) = \varnothing$.

– $\mathsf{BV}(\lambda x.M) = \mathsf{BV}(M) \cup \{x\}$.

– $\mathsf{BV}(MN) = \mathsf{BV}(\langle M, N \rangle) = \mathsf{BV}(M) \cup \mathsf{BV}(N)$.

– $\mathsf{BV}(\mathsf{fst}(M)) = \mathsf{BV}(\mathsf{snd}(M)) = \mathsf{BV}(M)$.

In the definition of β-reduction below we make use of substitution.

Definition 1.5. $M[x := N]$ denotes the result of substituting N for x in M

– $x[x := N] = N$ and if $x \neq y$, then $y[x := N] = y$

– $(\lambda x.M)[x := N] = \lambda x.M$

– $(\lambda y.M)[x := N] = \lambda y.(M[x := N])$, if $x \neq y$ and $y \notin \mathsf{FV}(N)$

– $(M_1 M_2)[x := N] = (M_1[x := N])(M_2[x := N])$

– $\langle M_1, M_2 \rangle[x := N] = \langle M_1[x := N], M_2[x := N] \rangle$

– $\mathsf{fst}(M)[x := N] = \mathsf{fst}(M[x := N])$

– $\mathsf{snd}(M)[x := N] = \mathsf{snd}(M[x := N])$

$$
\begin{array}{c|l}
 & \dfrac{}{x : \sigma \vdash x : \sigma}\ \text{ref} \\[2ex]
\times & \dfrac{\Gamma \vdash M : \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash \langle M, N\rangle : \sigma \times \tau}\ \text{pair} \qquad \dfrac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \mathsf{fst}(M) : \sigma}\ \text{fst} \quad \dfrac{\Gamma \vdash M : \sigma \times \tau}{\Gamma \vdash \mathsf{snd}(M) : \tau}\ \text{snd} \\[2ex]
\to & \dfrac{\Gamma, x : \sigma \vdash M : \tau}{\Gamma \vdash \lambda x.M : \sigma \to \tau}\ \text{abs} \qquad \dfrac{\Gamma \vdash M : \sigma \to \tau \quad \Gamma \vdash N : \sigma}{\Gamma \vdash MN : \tau}\ \text{app}
\end{array}
$$

Figure 1.3: Type Checking System

Now we introduce the notion of computation in the (typed) λ-calculus. This reduction rules are called *β-reduction*.

Definition 1.6.

$$
\begin{aligned}
(\lambda x.M)N &\xrightarrow{\beta} M[x := N] \\
\mathsf{fst}(\langle M, N\rangle) &\xrightarrow{\beta} M \\
\mathsf{snd}(\langle M, N\rangle) &\xrightarrow{\beta} N
\end{aligned}
$$

Note that β-reduction is closed under context:

$$
M \xrightarrow{\beta} N \Longrightarrow \begin{cases}
LM \xrightarrow{\beta} LN \\
ML \xrightarrow{\beta} NL \\
\lambda x.M \xrightarrow{\beta} \lambda x.N \\
\langle M, L\rangle \xrightarrow{\beta} \langle N, L\rangle \\
\langle L, M\rangle \xrightarrow{\beta} \langle L, N\rangle \\
\mathsf{fst}(M) \xrightarrow{\beta} \mathsf{fst}(N) \\
\mathsf{snd}(M) \xrightarrow{\beta} \mathsf{snd}(N)
\end{cases}
$$

1.4 The Curry-Howard Isomorphism

In this section we introduce the *Curry-Howard isomorphism* also know as the *Curry-Howard correspondence*. We start by presenting a basic type checking system for the typed λ-calculus in Figure 4.3.

Note that the system presented in Figure 4.3 is closely related to the type checking system introduced in the lecture on functional programming, see [7, Chapter 9]. The only difference is that we have extended the rules (ref), (abs), and (app) as in [7, Chapter 9] by rules governing the product types. This is due to a different design choice in crafting

	introduction	*elimination*
	$\dfrac{}{F \vdash F}\ \mathsf{Ax}$	
$\wedge$	$\dfrac{\Gamma \vdash E \quad \Gamma \vdash F}{\Gamma \vdash E \wedge F}\ \wedge : \mathsf{i}$	$\dfrac{\Gamma \vdash E \wedge F}{\Gamma \vdash E}\ \wedge : \mathsf{e}$ $\quad$ $\dfrac{\Gamma \vdash E \wedge F}{\Gamma \vdash F}\ \wedge : \mathsf{e}$
$\vee$	$\dfrac{\Gamma \vdash E}{\Gamma \vdash E \vee F}\ \vee : \mathsf{i}$ $\quad$ $\dfrac{\Gamma \vdash F}{\Gamma \vdash E \vee F}\ \vee : \mathsf{i}$	$\dfrac{\Gamma \vdash E \vee F \quad \Gamma, E \vdash G \quad \Gamma, F \vdash G}{\Gamma \vdash G}\ \vee : \mathsf{e}$
$\rightarrow$	$\dfrac{\Gamma, E \vdash F}{\Gamma \vdash E \rightarrow F}\ \rightarrow : \mathsf{i}$	$\dfrac{\Gamma \vdash E \quad \Gamma \vdash E \rightarrow F}{\Gamma \vdash F}\ \rightarrow : \mathsf{e}$
$\neg$	$\dfrac{\Gamma, F \vdash \bot}{\Gamma \vdash \neg F}\ \neg : \mathsf{i}$	$\dfrac{\Gamma \vdash F \quad \Gamma \vdash \neg F}{\Gamma \vdash \bot}\ \neg : \mathsf{e}$ $\quad$ $\dfrac{\Gamma \vdash \bot}{\Gamma \vdash F}\ \neg : \mathsf{e}$

Figure 1.4: Intuitionistic Propositional Rules (Sequent Style)

our type system which is inessential, but simplifies the description of the Curry-Howard isomorphism.

We recall the system of natural deduction rules presented above. In this context it is useful to change the style of presentation of these rules. For the sequel of this chapter we use the *sequent style form* to present these rules, see Figure 4.4. It is easy to see that these rules are equivalent to those natural deduction rules presented in Figure 4.1 and 4.2.

The crucial advantage of the presentation of natural deduction rules as in Figure 4.4 is the *direct* correspondence to the type checking system given in Figure 4.3. For instance the rule defining (app) in Figure 4.3 and implication elimination ($\rightarrow : \mathsf{e}$) are essentially the same rule. More precisely, the type of a term in the type checking system corresponds to a formula in the natural deduction rules and vice versa. Observe the following correspondence:

$$
\begin{array}{lcl}
(\text{ref}) & \sim & (\mathsf{Ax}) \\
(\text{abs}) & \sim & (\rightarrow : \mathsf{i}) \\
(\text{app}) & \sim & (\rightarrow : \mathsf{e}) \\
(\text{pair}) & \sim & (\wedge : \mathsf{i}) \\
(\text{fst}) & \sim & (\wedge : \mathsf{e}) \\
(\text{snd}) & \sim & (\wedge : \mathsf{e})
\end{array}
$$

In order to make this correspondence complete it suffices to introduce additional type checking rules (and corresponding types) for the the natural deduction rules ($\vee : \mathsf{i}$) and ($\vee : \mathsf{e}$). This is the purpose of the rules given in Figure 4.5. The $\mathsf{case\ of}$ destructor encodes pattern matching. Based on these type checking rules we can complete the table above:

$$\vee \;\left|\; \frac{\Gamma \vdash M : \sigma}{\Gamma \vdash \mathsf{inl}(M) : \sigma + \tau} \qquad \frac{\Gamma \vdash N : \tau}{\Gamma \vdash \mathsf{inr}(N) : \sigma + \tau} \right.$$

$$\left| \; \frac{\Gamma \vdash M : \sigma + \tau \quad \Gamma, x : \sigma \vdash N_1 : \gamma \quad \Gamma, y : \tau \vdash N_2 : \gamma}{\Gamma \vdash \mathsf{case}\ M\ \mathsf{of}\ \ \mathsf{inl}(x) \longrightarrow N_1 \mid \mathsf{inr}(y) \longrightarrow N_2 : \gamma} \right.$$

Figure 1.5: Type Checking System (Part II)

$$\begin{array}{lcl} (\text{inl}) & \sim & (\vee : \mathsf{i}) \\ (\text{inr}) & \sim & (\vee : \mathsf{i}) \\ (\text{case}) & \sim & (\vee : \mathsf{e}) \end{array}$$

The here described correspondence between *types* and *formulas* is often referred to as "types as formulas" paradigm. We summarise the isomorphism in the following table:

formulas	$\sim$	types
proofs	$\sim$	programs
normalisation	$\sim$	computation

Above we have already seen the precise connection between formulas and types. What is missing is some intuition about the last correspondence: normalisation of proofs and computations in a (typed) λ-calculus. A detailed presentation of the relation is outside the scope of this lecture. But it is easy to sketch the idea. Consider the following proof Ψ in the type checking system:

$$\dfrac{\dfrac{\begin{array}{cc}\Pi_1 & \Pi_2 \\ \vdots & \vdots \\ \Gamma \vdash M : \sigma & \Gamma \vdash N : \tau\end{array}}{\Gamma \vdash \langle M, N \rangle : \sigma \times \tau}}{\Gamma \vdash \mathsf{fst}(\langle M, N \rangle) : \sigma}$$

If we conceive this proof as a natural deduction proof and focus on the formulas proven we observe that there exists some redundancy in this proof. Essentially we derive the "formula" σ by first introducing the "formula" $\sigma \times \tau$ and then eliminating $\times$ again. A shorter proof Ψ' is given below:

$$\begin{array}{c}\Pi_1 \\ \vdots \\ \Gamma \vdash M : \sigma\end{array}$$

Such a proof transformation is called *normalisation*.

If we now consider the terms occurring in these proofs we see that in the first proof Ψ the term $\mathsf{fst}(\langle M, N \rangle)$ is shown to be well-typed with type σ, while in the second proof Ψ' the term M is shown to have type σ. Thus the *normalisation* from Ψ to Ψ' directly corresponds

the the β-reduction step $\mathsf{fst}(\langle M, N\rangle) \xrightarrow{\beta} M$ and thus to a *computation*. This correspondence between normalisation of proofs and computation in the typed λ-calculus holds in general. As another example we consider the β-reduction $(\lambda x.M)N \xrightarrow{\beta} M[x := N]$ together with the following proof normalisation:

$$\dfrac{\dfrac{\begin{array}{c}\Pi_1\\ \vdots\\ \Gamma, x : \sigma \vdash M : \tau\end{array}}{\Gamma \vdash \lambda x.M : \sigma \to \tau} \quad \begin{array}{c}\Pi_2\\ \vdots\\ \Gamma \vdash N : \tau\end{array}}{\Gamma \vdash (\lambda x.M)N : \tau} \quad \Longrightarrow \quad \begin{array}{c}\Pi_2\\ \Pi_1\\ \vdots\\ \Gamma \vdash M[x := N]\end{array}$$

Here the right proof is to be understood as the extension of proof Π_1 by the inferences in Π_2 such that all occurrences of x in Π_1 are replaced by the term N.

Bibliography

[1] H. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Studies in Logic and the Foundations of Mathematics. Elsevier, second edition, 1985.

[2] H. Barendregt and E. Barendsen. Introduction to lambda calculus. In *Aspenæs Workshop on Implementation of Functional Languages, Göteborg*. Programming Methodology Group, University of Göteborg and Chalmers University of Technology, 1988. Available at ftp://ftp.cs.kun.nl/pub/CompMath.Found/lambda.pdf.

[3] M. Dummett. *Elements of Intuitionism*. Oxford Logic Guides. Oxford University Press, second edition, 2000.

[4] J.R. Hindley and J.P. Seldin. *Lambda-Calculus and Combinators: An Introduction*. Cambridge University Press, second edition, 2008.

[5] J.G. Larrecq and I. Makie. *Proof Theory and Automated Deduction*. Number 6 in Applied Logic Series. Kluwer Academic Publishers, first edition, 2001.

[6] G. Moser. *Logic (MSc)*. Institute for Computer Science, 2010. Available at http://cl-informatik.uibk.ac.at/teaching/ws10/logic/.

[7] C. Sternagel. *Functional Programming*. Institute for Computer Science, 2009. Available at http://cl-informatik.uibk.ac.at/teaching/ws09/fp/material/fpln.pdf.