Universität des Saarlandes
Naturwissenschaftlich–Technische Fakultät I
Informatik

Diplomarbeit

Proof Nets for Intuitionistic Logic

Vorgelegt von
Matthias Horbach
am 25. Juli 2006

Angefertigt unter der Leitung von
Prof. Dr. Gert Smolka

Betreut von
Dr. Lutz Straßburger

Begutachtet von
Prof. Dr. Gert Smolka
Privatdozent Dr. Christoph Weidenbach

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbst erstellt und dazu keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Saarbrücken, den 25.07.2006

Abstract

Until the beginning of the 20th century, there was no way to reason formally about proofs. In particular, the question of proof equivalence had never been explored. When Hilbert asked in 1920 for an answer to this very question in his famous program, people started looking for proof formalizations.

Natural deduction and sequent calculi, which were invented by Gentzen in 1935, quickly became two of the main tools for the study of proofs. Gentzen's Hauptsatz on normal forms for his sequent calculi, and later on Prawitz' analog theorem for natural deduction, put forth a first notion of equivalent proofs in intuitionistic and classical logic.

However, natural deduction only works well for intuitionistic logic. This is why Girard invented proof nets in 1986 as an analog to natural deduction for (the multiplicative fragment of) linear logic. Their universal structure made proof nets also interesting for other logics. Proof nets have the great advantage that they eliminate most of the bureaucracy involved in deductive systems and so are probably closer to the essence of a proof. There has recently been an increasing interest in the development of proof nets for various kinds of logics. In 2005 for example, Lamarche and Straßburger were able to express sequent proofs in classical logic as proof nets.

In this thesis, I will, starting from proof nets for classical logic, turn the focus back on intuitionistic logic and propose proof nets that are suited as an extension of natural deduction. I will examine these nets and characterize those corresponding to natural deduction proofs. Additionally, I provide a cut elimination procedure for the new proof nets and prove termination and confluence for this reduction system, thus effectively a new notion of the equivalence of intuitionistic proofs.

Acknowledgments

At this place I would like to take the opportunity to thank the people that made this thesis possible.

First of all, I want to thank Prof. Gert Smolka. His lectures on logic captivated me very early in the course of my studies, and he kept my fascination awake all the time.

I am particularly grateful to my adviser Lutz Straßburger. He introduced me to proof theory and guided me through this thesis, always providing stimulating suggestions and ideas for further investigations.

I also like to thank the other members of the Programming Systems Lab and the Fachschaftsrat Mathematik for creating a pleasant atmosphere that that has made me enjoy working here.

Last but not least, I want to thank Andrea Heyl for allowing me to hunt for results longer than I had promised, and for her constant support in a stressful time.

Contents

Chapter 1

Introduction

What is a proof?

Whenever we study mathematical objects, one of the most important questions is whether two given objects should be identified. Examples are isomorphic groups or vector spaces, or maps that coincide on an interesting subset of their domain. One type of objects that completely resisted such an examination until the 20th century were proofs. Before that time, all proofs were mainly texts in a natural language, which made any arguments about them very awkward. So in fact, they were not even real mathematical objects.

The complete absence of any possibility to reason rigorously about proofs (and hence also about their equality) was the incitement of Hilbert's Program, in which he demanded to strictly formalize all mathematical reasoning. Hilbert had even considered including it into his famous lecture in 1900 as a 24th problem [Hil00, TW01]. So Hilbert may be seen as the founder of modern proof theory, whose main interests are the following (cf. [Pra71, Section I]):

(1) The basic question of defining the notion of proof, including the question of the distinction between different kinds of proofs such as constructive proofs and classical proofs.

(2) Investigation of the structures of (different kinds of) proofs, including e.g. questions concerning the existence of certain normal forms.

(3) The representation of proofs by formal derivations. In the same way as one asks when two formulas define the same set or two sentences express the same proposition, one asks when two derivations represent the same proof; in other words, one asks for identity criteria for proofs of for a "synonymity" (or equivalence) relation between derivations.

(4) Application of insights about the structure of proofs to other logical questions that are not formulated in terms of the notion of proof.

Evidently the most fundamental question, on which all others build, is the first one: What is a (formal) proof?

This question has of course to be answered separately for every logical system. However, most approaches to a definition of proof calculi work for a wide range of systems.

Hilbert himself favoured axiomatic systems, where a set of axioms is used to logically derive theorems. Russell and Whitehead tried in their "Principia Mathematica" [RW10] to infer all mathematical truths in this style.

It took more than 20 years to find alternative approaches. In his "Untersuchungen über das logische Schließen" [Gen35], Gentzen developed two new formal methods to write down proofs: on the one hand natural deduction, which uses deduction rules instead of axioms to try and model logical reasoning as it is used by logicians, and on the other hand sequent calculi, which use inference rules to derive provability statements (sequents) and were used by Gentzen primarily as a tool for studying natural deduction.

It turned out that the sequent calculi, which Gentzen introduced for intuitionistic and classical logic, also provide a solution to the second problem: sorting out a class of normal forms. Gentzen's Hauptsatz shows that each proof in one of the calculi can be transformed into a proof that does not use the so-called cut rule. However, this transformation does not lead to a unique normal form, and so it does not give a satisfactory answer to the question of proof equivalence.

This problem could be remedied by Prawitz [Pra65]. He identified the appropriate normal proofs in natural deduction as those in which no formula occurrence is both the principal premise of an elimination rule and the conclusion of an introduction rule, and proved an analog to the Hauptsatz for natural deduction. Because Prawitz' normal form are unique, natural deduction even provides a notion of the equivalence of intuitionistic proofs.

Recent Developments — Proof Nets

Although natural deduction seems perfectly apt for intuitionistic logic, it does not work so well with logics whose sequent systems allow for multiple consequences. This is the reason why, when Girard [Gir87] introduced linear logic, he was faced with a lack of adequate possibilities to describe the identity of proofs. In the presence of an involutive negation in linear logic, the distinction between input and output in natural deduction no longer made sense, such that the tree form of natural deduction proofs had to be given up. His solution was what he called "a linear natural deduction": proof nets.

Proof nets, which consist mainly of a set of formulas with links between atoms, have undergone a long development. At first, they could only handle the multiplicative fragment of linear logic and used sequents in disguise (called boxes) for the additive features. It took more than 15 years until proof nets for the whole calculus could be found [GLR95].

Although proof nets were primarily designed for linear logic, their structure is so universal that it was only a matter of time until they were adapted to other logics. In 2005, Lamarche and Straßburger [LS05] succeeded to classify a set of proof nets that correspond to classical sequent proofs.

Contributions

In this thesis, we will concentrate on intuitionistic propositional logic. Following the guideline given by the first three above mentioned goals of proof theory, we will present a proof system for intuitionistic logic and distinguish especially simple proofs to which every proof can be reduced. As we will prove this reduction to be terminating and confluent, we thus provide a new notion of the equivalence of proofs. This means, that we give an alternative answer to the basic question: "What is an intuitionistic proof?"

In particular, we will, starting from the proof nets for classical logic developed by Lamarche and Straßburger, introduce proof nets for intuitionistic propositional logic and classify the proof nets corresponding to natural deduction proofs.

Additionally, we provide a cut elimination procedure for the new proof nets, which is loosely connected to normalization in natural deduction (or, equivalently, in the typed λ-calculus), and prove termination and confluence for this system.

Related Work

Recently, Guglielmi [Gug02] and Tiu [Tiu05] have been working on new calculi for intuitionistic logic, that extend Gentzens [Gen35] sequent calculi by allowing transformations not only at top level.

In parallel, there has been quite some work on prenets for different logics. Girard [Gir87, GLR95] worked for years on a proof net calculus for full linear logic. Important for the intuitionistic fragment were e.g. Danos and Regnier [DR89], who developed a polarization system for intuitionistic formulas, and Lamarche [Lam95], who used their theory to examine proof nets for the intuitionistic fragment of linear logic.

Proof nets for classical logic were analyzed by Lamarche and Straßburger [LS05, Str05].

Outline

To get started, we will present some basic ideas of intuitionistic logic in chapter 2. We will especially address natural deduction, the typed λ-calculus, and the connection between both systems, given by the Curry-Howard-Isomorphism.

The second ingredient, and later on the motivating reference system, are proof nets for classical logic. While we will also look back at the early development of proof nets, our main focus in chapter 3 will be on the classical case. Here we will see the idea behind both the translation of classical sequent proofs into proof nets and the cut elimination procedure for these nets.

Chapter 4 is devoted to the introduction of proof nets for intuitionistic logic. After laying the foundations, which come in the shape of the used mathematical objects and above all a refined definition of proof nets, we will see how typed λ-terms can be translated into these nets. We will then study properties of these nets and give an algorithm that, given a proof net, recovers a term that corresponds to this proof net. Finally, we will define and examine cut elimination for intuitionistic proof nets. We will prove termination and confluence of the cut elimination procedure, and we will show in how far it corresponds to the reduction of λ-terms.

A short recapitulation of the results in chapter 5, along with an outlook towards interesting further research, concludes the thesis.

Chapter 2

Intuitionistic Logic

Intuitionism emerged at the beginning of the 20th century, mainly as a development of mathematical fundamental research. It based on a criticism of the classical logical proof methods. Advocates of intuitionism, like L. Brouwer [Bro07] and later on A. Heyting [Hey25], were mainly worried about metalogical principles, above all the *tertium non datur* assumption, which states that every statement is either true or false. They interpreted this principle in such a way, that the truth or falsity of every statement can actually be proved. In his introductory book on intuitionism, Heyting [Hey56] gives the following example statements:

(1) k is the greatest prime such that $k-1$ is also a prime, or $k=1$ if such a number does not exist.

(2) l is the greatest prime such that $l-2$ is also a prime, or $l=1$ if such a number does not exist.

He explains [Hey56, p. 2]: “Classical mathematics neglects altogether the obvious difference in character between these definitions. k can actually be calculated ($k=3$), whereas we possess no method for calculating l, as it is not known whether the sequence of pairs of twin primes p, $p+2$ is finite or not. Therefore intuitionists reject (2) as a definition of an integer; they consider an integer to be well defined only if a method for calculating it is given. Now this line of thought leads to the rejection of the principle of excluded middle, for if the sequence of twin primes were either finite or infinite, (2) would define an integer.”

A notable feature of intuitionistic logic is the Brouwer–Heyting–Kolmogorow interpretation, where formulas are interpreted by means of their proofs. It turns out that, apart from Gentzen’s natural deduction systems [Gen35], one of the approaches best suited to a formal description of intuitionism is the λ-calculus as introduced by Alonzo Church [Chu33, Chu40] (untyped version) and Curry [Cur34] (typed version).

It took some time until Curry and Feys [CF58] and later on Howard [How80] discovered a close correspondence between natural deduction proofs and the functional programs of the typed λ-calculus, nowadays known as Curry-Howard isomorphism. This isomorphism basically states that intuitionistically valid formulas correspond to inhabited types, and natural deduction proofs correspond to typed λ-terms.

In this chapter, we will have a short look at natural deduction for propositional intuitionistic logic, the typed λ-calculus and the connection between both systems. For further information on natural deduction, the reader is referred to the original works of Gentzen [Gen35] and Prawitz [Pra65]. Among others, Barendregt [Bar92] gives an extensive overview of the typed λ-calculus.

2.1 Natural Deduction

A system of natural deduction can be thought of as a set of rules that determines the concept of deduction in a logic. Such a system constitutes a logical framework that is natural in several ways. First of all, it corresponds closely to the methods found in intuitive, informal reasoning. The ideas of intuitive proofs can usually be translated into a natural deduction proof and give it an appearance that allows to retrieve the main structure. Secondly, Gentzen's rules allow for the distinction of normal forms, and for the transformation of every proof into such a normal form. This *Hauptsatz*, first formulated for different sequent calculi, was one of Gentzen's [Gen35] most important results and later on proved for natural deduction systems by Prawitz [Pra65].

Gentzen himself gave natural deduction systems for intuitionistic and classical first order logic. However, we restrict ourselves to the intuitionistic case, and therein to propositional logic.

The framework of formulas and sequents is the following:

Definition and Notation 2.1.1. Let $\mathcal{A} = \{a, b, \ldots\} \cup \{\bot\}$ be a countable set, called the set of *atoms*, containing a special symbol $\bot$ that denotes falsity. The set $\mathcal{F}$ of formulas is defined by the following abstract syntax:

$$\mathcal{F} := \mathcal{A} \mid \mathcal{F} \to \mathcal{F} \mid \mathcal{F} \vee \mathcal{F} \mid \mathcal{F} \wedge \mathcal{F}$$

We write $A, B, \ldots$ for formulas and Γ for multisets of formulas, and we use the shorthand notations

$$\Gamma, A := \Gamma \cup \{A\} \qquad \text{and}$$
$$A \to B \to \ldots \to C \to D := A \to (B \to (\ldots \to (C \to D) \ldots)) \ ,$$

i.e. implication is right associative.

Axioms and Falsity

$$\frac{}{A \vdash A}\ \text{axiom} \qquad\qquad \frac{\Gamma \vdash \bot}{\Gamma \vdash A}\ \text{falsum}$$

Rules for Implication

$$\frac{\Gamma, \overbrace{A, \ldots, A}^{n \text{ times}} \vdash B}{\Gamma \vdash A \to B}\ \to\mathrm{I}_n \qquad\qquad \frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash A \to B}{\Gamma_1, \Gamma_2 \vdash B}\ \to\mathrm{E}$$

Rules for Conjunction

$$\frac{\Gamma_1 \vdash A \quad \Gamma_2 \vdash B}{\Gamma_1, \Gamma_2 \vdash A \wedge B}\ \wedge\mathrm{I}$$

$$\frac{\Gamma \vdash A \wedge B}{\Gamma \vdash A}\ \wedge\mathrm{E1} \qquad\qquad \frac{\Gamma \vdash A \wedge B}{\Gamma \vdash B}\ \wedge\mathrm{E2}$$

Rules for Disjunction

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \vee B}\ \vee\mathrm{I1} \qquad\qquad \frac{\Gamma \vdash B}{\Gamma \vdash A \vee B}\ \vee\mathrm{I2}$$

$$\frac{\Gamma_1 \vdash A \vee B \quad \Gamma_2 \vdash A \to C \quad \Gamma_3 \vdash B \to C}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash C}\ \vee\mathrm{E}$$

Table 2.1: Natural Deduction Rules for Propositional Intuitionistic Logic

Table 2.1 summarizes the natural deduction proof rules for intuitionistic logic. Despite the danger to blur the difference between natural deduction and sequent calculi, the rules are presented in a sequent style to provide a more concise representation. As a result, the rule $\to$I usually found in natural deduction comes in the different forms $\to\mathrm{I}_0, \to\mathrm{I}_1, \ldots$, depending on how many hypotheses are discharged by the rule. With the exception of the rules for axioms and falsity, all rules come in two forms, as introduction rules and elimination rules, that allow inference to or from a formula with a given principal connector.

Example 2.1.2. The following are natural deduction proofs of the formula $(a \to a) \to a \to a$:

$$\frac{\dfrac{}{a \to a \vdash a \to a}\ \text{axiom}}{\vdash (a \to a) \to a \to a}\ \to\mathrm{I}_1 \qquad \frac{\dfrac{\dfrac{}{a \vdash a}\ \text{axiom}}{\vdash a \to a}\ \to\mathrm{I}_1}{\vdash (a \to a) \to a \to a}\ \to\mathrm{I}_0$$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{a \vdash a}\,\text{axiom} \qquad \dfrac{}{a \to a \vdash a \to a}\,\text{axiom}}{a \to a, a \vdash a}\,{\to}\mathrm{E}}{a \to a \vdash a \to a}\,{\to}\mathrm{I}_1}{\vdash (a \to a) \to a \to a}\,{\to}\mathrm{I}_1$$

The introduction rules have the *subformula property*: The premise of each rule contains only formulas that appear in the conclusion. However, this does not hold for the elimination rules, in which a formula simply vanishes on its way from premise to conclusion. This behavior of the elimination rules is undesirable in various applications, e.g. in proof search, where it corresponds to guessing a formula.

However, many instances of elimination rules are not necessary. For example, a proof containing

$$\begin{array}{c} \vdots \qquad \vdots \\ \dfrac{\dfrac{\vdash A \quad \vdash B}{\vdash A \wedge B}\,\wedge\mathrm{I}}{\vdash A}\,\wedge\mathrm{E}_1 \\ \vdots \end{array}$$

could be simplified to just the following subproof:

$$\begin{array}{c} \vdots \\ \vdash A \\ \vdots \end{array}$$

The reduced proof does not contain the superfluous elimination rule any more. Prawitz [Pra65, Pra71] found a general principle behind these simplifications. He developed a method of converting any natural deduction proof into a proof with the same conclusion and with a certain shape. Roughly speaking, the assumptions in such a proof are first broken down into their parts by elimination rules and then recombined by introduction rules, until the conclusion is reached.

The concrete nature of the reductions is not important for the following chapters, so we will not go into details. It is however important, that Prawitz found a notion of normal forms of intuitionistic proofs, and a way to normalize any given proof.

We will come back to this when we describe the analogous theorem in the typed λ-calculus.

2.2 The Typed λ-Calculus and the Curry-Howard-Isomorphism

The primary goal of this section is to give a short overview of the typed λ-calculus and its connection to intuitionistic logic. The following remarks are partially inspired by the works of Barendregt [Bar84, Bar92, Bar97], which also provide more information on different λ-calculi. Other sources providing further background are the books by Pierce [Pie02], Hindley [Hin97], and Girard, Taylor and Lafont [GTL89].

We will have a short look at the syntax of the typed λ-calculus, as well as recall the bijection between the sets of possible types of λ-terms and intuitionistically provable formulas.

Definition 2.2.1. The set $\mathcal{V} = \{x, y, \ldots\}$ denotes a countable set of variables. A *type assignment* is a function $\mathcal{V} \to \mathcal{F}$ that assigns a formula to each variable. The value of a variable v under a given type assignment is called the *type* of v.

Notation 2.2.2. In what follows, we will stick to one fixed type assignment $\tau\colon \mathcal{V} \to \mathcal{F}$. We assume that the preimage of every type under τ is countable, i.e. there are countably many variables of each type.

For our convenience, we will assume in all examples, that x, y and f are variables, and that a is an atom, such that $\tau(x) = \tau(y) = a$ and $\tau(f) = (a \to a)$.

Definition 2.2.3. The set Λ of *λ-preterms* is defined by the following abstract syntax:

$$\begin{aligned}\Lambda := {} & \mathcal{V} \mid \Lambda\,\Lambda \mid \lambda\mathcal{V}.\Lambda \\ & \mid \pi_1\,\Lambda \mid \pi_2\,\Lambda \mid \mathrm{pair}\,\Lambda\,\Lambda \\ & \mid \mathrm{inl}\,\Lambda \mid \mathrm{inr}\,\Lambda \mid \mathrm{case}\,\Lambda\,\Lambda\,\Lambda \mid \mathrm{null}\,\Lambda\end{aligned}$$

The λ-preterms of the form $e_1\,e_2$ are called *applications*, those of the form $\lambda v.e_1$ are called *abstractions*, λv is called a *variable binder*, and π_1, π_2, pair, inl, inr, case and null are *constants*.

If e is a λ-preterm, and if there is a formula A, such that the statement $e : A$ can be derived by the typing rules summarized in table 2.2, then we say that e *is of type* A. In this case, e is called *typed λ-term*, or just *λ-term*.

We write $\lambda^{\to}$ and $\lambda^{\to\wedge}$ for the calculi of those typed λ-terms whose typing derivations contain only types consisting of implications, or of implications and conjunctions, respectively, and only the corresponding derivation rules and the axiom rule.

Notation 2.2.4. We use brackets to avoid ambiguities in the string representation of terms. However, we stick to two conventions that allow us to minimize the use of brackets:

$$
\begin{array}{llll}
\dfrac{}{v : \tau(v)} & \text{var} & \dfrac{e : A \wedge B}{\pi_1\, e : A} & \text{proj}_1 \\[2ex]
\dfrac{e_2 : A \quad e_1 : A \to B}{e_1\, e_2 : B} & \text{app} & \dfrac{e : A \wedge B}{\pi_2\, e : B} & \text{proj}_2 \\[2ex]
\dfrac{v : A \quad e : B}{\lambda v.e : A \to B} & \text{abs} & \dfrac{e_1 : A \quad e_2 : B}{\text{pair}\, e_1\, e_2 : A \wedge B} & \text{pair}
\end{array}
$$

$$
\begin{array}{cl}
\dfrac{e : A}{\text{inl}\, e : A \vee B} & \text{left} \\[2ex]
\dfrac{e : B}{\text{inr}\, e : A \vee B} & \text{right} \\[2ex]
\dfrac{e_1 : A \vee B \quad e_2 : A \to C \quad e_3 : B \to C}{\text{case}\, e_1\, e_2\, e_3 : C} & \text{case} \\[2ex]
\dfrac{e : \bot}{\text{null}\, e : A} & \text{null}
\end{array}
$$

Table 2.2: Typing rules for λ-terms

- Applications are left-associative, i.e.

  $$e_1\, e_2\, e_3\, \ldots\, e_n := (\ldots((e_1\, e_2)\, e_3)\, \ldots\, e_n)\ .$$

  This corresponds to the right-associativity of implication.

- The scope of a variable binder is always maximal, i.e.

  $$\lambda v.e_1\, e_2 := \lambda v.(e_1\, e_2)\ .$$

Example 2.2.5. The three expressions $\lambda f.f$, $\lambda f.\lambda x.x$, and $\lambda f.\lambda x.f\,x$ are typed λ-terms of type $(a \to a) \to a \to a$. They have the following typing derivations, respectively:

$$
\dfrac{\dfrac{}{f : a \to a}\,\text{var} \qquad \dfrac{}{f : a \to a}\,\text{var}}{\lambda f.f : (a \to a) \to a \to a}\,\text{abs}
\qquad
\dfrac{\dfrac{}{f : a \to a}\,\text{var} \qquad \dfrac{\dfrac{}{x : a}\,\text{var} \quad \dfrac{}{x : a}\,\text{var}}{\lambda x.x : a \to a}\,\text{abs}}{\lambda f.\lambda x.x : (a \to a) \to a \to a}\,\text{abs}
$$

$$
\dfrac{\dfrac{}{f : a \to a}\,\text{var} \qquad \dfrac{\dfrac{}{x : a}\,\text{var} \quad \dfrac{\dfrac{}{x : a}\,\text{var} \quad \dfrac{}{f : a \to a}\,\text{var}}{f\,x : a}\,\text{app}}{\lambda x.f\,x : a \to a}\,\text{abs}}{\lambda f.\lambda x.f\,x : (a \to a) \to a \to a}\,\text{abs}
$$

When Church presented the (at that time still untyped) λ-calculus, he tried to provide a general theory of functions and logic. This means that the intuition behind λ-terms is to model mathematical functions as we understand them today. For example, he wrote the function that adds 1 to every natural number as $\hat{x}.x + 1$, which was changed by typesetters to $\lambda x.x + 1$. Later on, the constants were introduced to model functions from or to direct products and sums of sets.

However, the representation of a function as a λ-term is by no means unique. The successor function above might as well be written as $\lambda y.y{+}1$, or as $\lambda y.((\lambda x.x + 1)y)$, as both functions increment an input by 1. To identify all these terms, that mean essentially the same, there are several notions of term equivalence.

To be able to properly formulate these notions, we need the concepts of free variables and substitutions.

Definition 2.2.6. The function $FV\colon \Lambda \to \mathcal{V}$ assigning to each λ-term its set of *free variables* is inductively defined as

$$\begin{aligned}
FV(v) &= v && \text{if } v \in \mathcal{V}\\
FV(e_1\,e_2) &= FV(e_1) \cup FV(e_2)\\
FV(\lambda v.e) &= FV(e) \setminus \{v\}\\
FV(c\,e_1) &= FV(e_1) && \text{if } c \text{ is a constant}\\
FV(c\,e_1\,e_2) &= FV(e_1) \cup FV(e_2) && \text{if } c \text{ is a constant}\\
FV(c\,e_1\,e_2\,e_3) &= FV(e_1) \cup FV(e_2) \cup FV(e_3) && \text{if } c \text{ is a constant}
\end{aligned}$$

Each variable that occurs but is not free in a λ-term e is *bound*. A term without free variables is *closed*.

The substitution function replacing each free instance of a variable v in a λ-term e' by a term e of the same type (i.e. $e : \tau(v)$) is written $e'[v \mapsto e]$ and inductively defined as follows:

$$\begin{aligned}
v[v \mapsto e] &= e\\
w[v \mapsto e] &= w && \text{if } v \neq w \in \mathcal{V}\\
(c\,e_1)[v \mapsto e] &= c && \text{if } c \text{ is a constant}\\
(c\,e_1\,e_2)[v \mapsto e] &= c && \text{if } c \text{ is a constant}\\
(c\,e_1\,e_2\,e_3)[v \mapsto e] &= c && \text{if } c \text{ is a constant}\\
(e_1\,e_2)[v \mapsto e] &= (e_1[v \mapsto e])\,(e_2[v \mapsto e])\\
(\lambda v.e_1)[v \mapsto e] &= (\lambda v.e_1)\\
(\lambda w.e_1)[v \mapsto e] &= \lambda w.(e_1[v \mapsto e]) && \text{if } v \neq w \text{ and } w \notin FV(e)\\
(\lambda w.e_1)[v \mapsto e] &= \lambda w'.((e_1[w \mapsto w'])[v \mapsto e])\\
&\qquad \text{for some } w' \notin FV(e) \cup FV(e_1) && \text{if } v \neq w \text{ and } w \in FV(e)
\end{aligned}$$

The most obvious equivalence of λ-terms is caused by (bound) variable renaming:

Definition 2.2.7. An *α-conversion* step is the procedure of replacing a λ-term $\lambda v.e$ by $\lambda w.(e[v \mapsto w])$, where $w \notin FV(e)$.

Two λ-terms are *α-equivalent*, if they can be transformed into each other by a series of α-conversions.

Example 2.2.8. If $\tau(y) = \tau(x) = a$, then $\lambda x.x$ and $\lambda y.y$ are α-equivalent, but $\lambda x.y$, $\lambda x.x$ and $\lambda y.x$ are pairwise α-inequivalent.

If we regard abstractions as functions and applications as evaluation of functions, we get a second notion of equivalence: A term of the form $(\lambda v.e_1)\, e_2$ should be equivalent to the result of its evaluation.

On the other hand, an abstraction $\lambda v.e\, v$, where v is not free in e, denotes the same function as just the term e. This formalizes the concept, that two functions are equal, if both always yield the same result when applied to the same argument.

Definition 2.2.9. A λ-term of the form $(\lambda v.e_1)\, e_2$ is called *β-redex* (reducible expression). A λ-term of the form $\lambda v.e\, v$, where v is not free in e is called *η-redex*.

A *β-reduction* (or *η-reduction*, respectively) step is the procedure of replacing a β-redex $(\lambda v.e_1)\, e_2$ by $e_1[v \mapsto e_2]$ (or an η-redex $\lambda v.e\, v$ by e).

A λ-term is called *β-normal*, if it contains no β-redex, and *η-normal*, if it contains no η-redex. When both reduction strategies are combined, we speak of *$\beta\eta$-reduction* and of *$\beta\eta$-normal forms*.

Example 2.2.10. The term $\lambda f.\lambda x.f\, x$ reduces in one η-step to the $\beta\eta$-normal term $\lambda f.f$, and $(\lambda f.\lambda x.f\, x)\, (\lambda x.x)$ reduces via $\beta\eta$ to the normal term $\lambda x.x$.

Tait [Tai67] showed that $\beta\eta$-reduction is terminating and uniquely normalizing for typed λ-terms, i.e. that every typed λ-term has exactly one normal form with respect to $\beta\eta$-reduction, and every sequence of $\beta\eta$-reduction steps leads to this normal form.

The reductions we have talked about so far are only concerned with the purely functional part of the λ-calculus. Both β- and η-reduction simplify terms in the cases where an abstraction and an application, i.e. the creation and evaluation of a function, come together.

However, we also have an intuitive understanding of the analogous concepts for sums and products of sets. For example, the projection of the pair $(1, 2)$ of natural numbers to its first component equals 1. This interaction between pairing and projection and its analog for sums can easily be translated into the language of λ-terms.

Definition 2.2.11. A λ-term of the form $\pi_i(\mathrm{pair}\, e_1\, e_2)$ is called *product-redex*. A *product-reduction* step is the procedure of replacing a product-redex $\pi_i(\mathrm{pair}\, e_1\, e_2)$ by e_i.

A λ-term of the form $\mathrm{case}(\mathrm{inl}\, e_1)\, e_2\, e_3$ or $\mathrm{case}(\mathrm{inr}\, e_1)\, e_2\, e_3$ is called *sum-redex*. A *sum-reduction* step is the procedure of replacing a sum-redex $\mathrm{case}\,(\mathrm{inl}\, e_1)\, e_2\, e_3$ by $e_2\, e_1$, or a sum-redex $\mathrm{case}\,(\mathrm{inr}\, e_1)\, e_2\, e_3$ by $e_3\, e_1$.

The Curry-Howard Correspondence

The work of Curry and Feys [CF58], Howard [How80] and others has shown a way to consider proofs as programs, and vice versa. For example, proofs of a sequent $\vdash A_1 \to \ldots \to A_k \to B$ may be interpreted as functional programs mapping n inputs of types $A_1, \ldots, A_k$ to an output of type B, i.e. as λ-terms of type $A_1 \to \ldots \to A_k \to B$.

We can formalize this idea by the following inductive translation of typed λ-terms into natural deduction proofs. A term e of type B with free variable occurrences $v_1, \ldots, v_k$ (counted with multiplicities) is translated into a proof with conclusion $\tau(v_1), \ldots, \tau(v_k) \vdash B$ as follows:

- If e is a variable, and $\tau(e) = A$, then it is translated to

$$\dfrac{}{A \vdash A}\ \text{axiom}$$

- If $e = \lambda v.e_1$, $\tau(v) = A$, $e_1{:}B$, Π_1 is the translation of e_1, and v appears $n \geq 0$ times freely in e_1, then e is translated as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdots\ \Pi_1}}{\Gamma, A, \ldots, A \vdash B}}{\Gamma \vdash A \to B}\ {\to}\mathrm{I}_n$$

- An application $e = e_1\, e_2$, where $e_1{:}A \to B$, $e_2{:}A$, and Π_i is the translation of e_i, corresponds to

$$\dfrac{\dfrac{\dfrac{}{\vdots\ \Pi_2}}{\Gamma_1 \vdash A} \quad \dfrac{\dfrac{}{\vdots\ \Pi_1}}{\Gamma_2 \vdash A \to B}}{\Gamma_1, \Gamma_2 \vdash B}\ {\to}\mathrm{E}$$

- A pairing $e = \mathrm{pair}\, e_1\, e_2$, where $e_i{:}A_i$ and Π_i is the translation of e_i, is translated to

$$\dfrac{\dfrac{\dfrac{}{\vdots\ \Pi_1}}{\Gamma_1 \vdash A_1} \quad \dfrac{\dfrac{}{\vdots\ \Pi_2}}{\Gamma_2 \vdash A_2}}{\Gamma_1, \Gamma_2 \vdash A_1 \wedge A_2}\ \wedge\mathrm{I}$$

- A projection $e = \pi_i\, e_1$, $i \in \{1,2\}$, where $e_1{:}A_1{\wedge}A_2$ and Π_1 is the translation of e_1, corresponds to

$$\dfrac{\dfrac{\dfrac{}{\vdots\ \Pi_1}}{\Gamma \vdash A_1 \wedge A_2}}{\Gamma \vdash A_i}\ \wedge\mathrm{E}_i$$

- If $e = \mathrm{case}\, e_1\, e_2\, e_3$, $e_1{:}A{\vee}B$, $e_2{:}A{\to}C$, $e_3{:}B{\to}C$, and Π_i is the translation of e_i, then e is translated as follows:

$$\dfrac{\dfrac{\dfrac{}{\vdots\ \Pi_3}}{\Gamma_3 \vdash B \to C} \quad \dfrac{\dfrac{}{\vdots\ \Pi_2}}{\Gamma_2 \vdash A \to C} \quad \dfrac{\dfrac{}{\vdots\ \Pi_1}}{\Gamma_1 \vdash A \vee B}}{\Gamma_1, \Gamma_2, \Gamma_3 \vdash C}\ \vee\mathrm{E}$$

- If $e = \mathrm{inl}\ e_1$, $e_1{:}A_1$, $e{:}A_1{\vee}A_2$, and Π_1 is the translation of e_1, then e corresponds to

$$\dfrac{\dfrac{\dfrac{}{\vdots\ \Pi_1}}{\Gamma \vdash A_1}}{\Gamma \vdash A_1 \vee A_2}\ \vee\mathrm{I1}$$

  If $e = \mathrm{inr}\ e_2$, $e_2{:}A_2$, $e{:}A_1{\vee}A_2$, and Π_2 is the translation of e_2, then e is translated to

$$\dfrac{\dfrac{\dfrac{}{\vdots\ \Pi_2}}{\Gamma \vdash A_2}}{\Gamma \vdash A_1 \vee A_2}\ \vee\mathrm{I2}$$

- A term $e = \mathrm{null}\, e_1$, where $e_1{:}\bot$, $e{:}A$, and Π_1 is the translation of e_1, is translated to

$$\dfrac{\dfrac{\dfrac{}{\vdots\ \Pi_1}}{\Gamma \vdash \bot}}{\Gamma \vdash A}\ \mathrm{falsum}$$

Example 2.2.12. Via this translation, the terms of type $(a \to a) \to a \to a$ from example 2.2.5 correspond to the natural deduction proofs of the formula $(a \to a) \to a \to a$ from example 2.1.2.

The important fact about this translation is the following theorem:

Theorem 2.2.13 (Curry-Howard-Correspondence). *A formula A is intuitionistically valid, if and only if there is a closed typed λ-term of type A. Moreover, the above translation gives a bijection between closed λ-terms of type A and natural deduction proofs of $\vdash A$, and normalization in both systems commutes with this translation.*

In fact, the reductions of λ-terms defined in the last section just translate to reductions of natural deduction proofs as used by Prawitz.

Remember that we want to define proof nets, i.e. a new proof system, for intuitionistic logic. To make any statements about this system, we need to relate it to a system we already know. The great benefit of this theorem in our setting is, that it gives us a second choice for the reference formalism. Compared to natural deduction proofs, λ-terms have the great advantage that they are more compact and that normalization is more intuitive for the λ-calculus than for natural deduction.

Chapter 3

Proof Nets

3.1 The Idea and History of Proof Nets

All logical theories admit different kinds of proof systems. Intuitionistic logic, for example, can be regarded from the perspective of either natural deduction or sequents. According to Prawitz [Pra65], natural deduction proofs should be regarded as "the true ones", while those in sequent calculus distinguish negligibilities like the permutation of two independent rule applications. Hence Prawitz considers them no longer as primitive but only as instructions on how to reconstruct a natural deduction proof.

When Girard [Gir87] developed linear logic, he was faced with the same task, but even more acute: On the one hand, linear logic is also constructive, so one expects an interpretation of proofs as programs, as well as a normalization theorem like Prawitz' theorem for natural deduction. On the other hand, linear logic has very much parallelism in it, taking it close to classical logic, and far away from a nice normalization.

The investigation of these topics lead Girard to the invention of *proof nets*, a proof system that transmits the absence of bureaucracy and the strong notion of proof equivalence from natural deduction to linear logic.

Other researchers had an eye on the computational advantages of proof nets. Lamarche and Retoré [LR96], for example, found out that cut elimination for proof nets in the linguistically important Lambek calculus is especially easy, and that proof nets provide a way to get complexity results for this calculus.

Since the time when Girard first introduced proof nets, the basic constructive idea has always been to represent formulas as trees, and to use some links between the leaves to encode a proof of the formula. As sequents were the proof system of choice, the links in fact always encoded sequent proofs. In this setting, each link corresponds to a sequent calculus rule application.

However in recent years, proof nets relieved themselves of this strong influence of the sequent calculus and became more and more self–contained.

As a proof system of their own right, proof nets should ideally be independent of the proof system used to construct them. They are now seen to be more than a poor copy of sequent calculus systems; today, an alternative approach is:

> A proof net is a formula tree, or a sequent forest, enriched with an additional graph structure.

This graph structure is supposed to reflect the essential part of a proof, not its concrete look in a given proof system. Additionally, proof nets are thought of as graphical representations of proofs that ignore bureaucracy, meaning that e.g. trivial permutations of deduction steps in a proof are not reflected in the corresponding proof net.

Probably the first researchers to deliberately adapt this notion of proof nets were Hughes and van Glabbeek [HvG03] for unit–free multiplicative additive linear logic, and Lamarche and Straßburger [SL04] for multiplicative linear logic with units.

Shortly after their work on multiplicative linear logic, Lamarche and Straßburger [LS05] also developed proof nets for classical logic, being the first to leave the realm of linearity.

3.2 Proof Nets for Classical Propositional Logic

To further introduce the reader to proof nets, we will shortly illustrate the concepts behind one type of proof nets for classical logic (CL), the so-called $\mathbb{N}$-*proof nets*, as presented by Lamarche and Straßburger [LS05] and recapitulated by Straßburger [Str05].

Some of the ideas shown there carry over directly (e.g. the meaning of links) or along general lines (e.g. the definition cut elimination) to proof nets for intuitionistic logic as presented in chapter 4. As for now, we will not give any proofs or technical details in this section, just a first glance at how $\mathbb{N}$-proof nets work.

Definition 3.2.1. Given a countable set $\mathcal{A} = \{a, b, \ldots\}$ of atoms, and their duals $\overline{\mathcal{A}} = \{\overline{a}, \overline{b}, \ldots\}$, the set $\mathcal{F}_c$ of Cl-formulas is defined as follows:

$$\mathcal{F}_c ::= \mathcal{A} \mid \overline{\mathcal{A}} \mid \mathcal{F}_c \wedge \mathcal{F}_c \mid \mathcal{F}_c \vee \mathcal{F}_c$$

(Constants representing truth and falsity may be added, but we will ignore them here to keep things simple.) Additionally, a formula may be of the form $A \between \overline{A}$, where $\overline{(\cdot)}$ is recursively defined by the de Morgan rules and $(\overline{\overline{a}} = a)$. The symbol $\between$ is called the *cut symbol*, and a formula of this shape is called a *cut*.

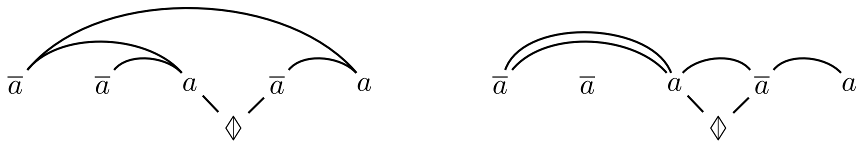
Formulas are considered as binary trees, whose leaves are labeled by atoms, and whose inner nodes are labeled by $\wedge$ and $\vee$. A (one-sided) sequent Γ is considered as a forest. We write leaves(Γ) for its set of leaves.

Given a forest Γ, a *linking* is a symmetrical function L: leaves$(\Gamma) \times$ leaves$(\Gamma) \to \mathbb{N}$, that associates to every two leaves a natural number, which can only be non-zero if the leaves are labeled by dual atoms. In this case, the leaves are said to be *linked*.

A sequent Γ together with a linking L is called $\mathbb{N}$-*prenet*, denoted by $L \rhd \Gamma$.

When we plot prenets, we draw the sequent forest, and if $L(l, l') = n > 0$, we connect l and l' by n edges.

Example 3.2.2. Given the sequent $\Gamma = \overline{a}, \overline{a}, a \Diamond \overline{a}, a$, the following two graphics depict $\mathbb{N}$-prenets over Γ:



Lamarche and Straßburger developed their prenets in such a way that a given (one sided) sequent calculus proof can easily be converted into a proof net. The idea is simple: The sequent forest is created from the proved formula and one tree for each use of a cut rule. The atoms of the proved formula and the atoms introduced by the cuts are pursued on their way through the proof, and whenever two atoms come together in an axiom rule, the corresponding leaves of the sequent forest are connected.

Table 3.1 gives an example of three sequent proofs of the sequent $\overline{a}, a \wedge \overline{a}, a$ along with the corresponding prenets. For a formal treatment compare [LS05].

A much more simplified version of the idea to connect atoms of a formula to represent a proof can already be found in Andrews' *matings* [And76] or Bibel's *connections* [Bib81]. However, both systems pose strong restrictions on the links. For example, they allow links only if the linked atoms are in a disjunction. Because these restrictions cannot be maintained during normalization, their system is too weak for our purpose.

Lamarche and Straßburger also give a geometric criterion to decide whether a prenet is sequentializable, in the sense that it comes from a sequent proof:

Definition 3.2.3. A *conjunctive pruning* of a prenet $L \rhd \Gamma$ is a sub-prenet that is obtained by deleting one child subformula for every conjunction or cut node in Γ, together with all links starting or ending in a deleted formula.

A prenet is called *correct*, if all its conjunctive prunings contain at least one link.

A correct prenet is called *(classical) proof net*.

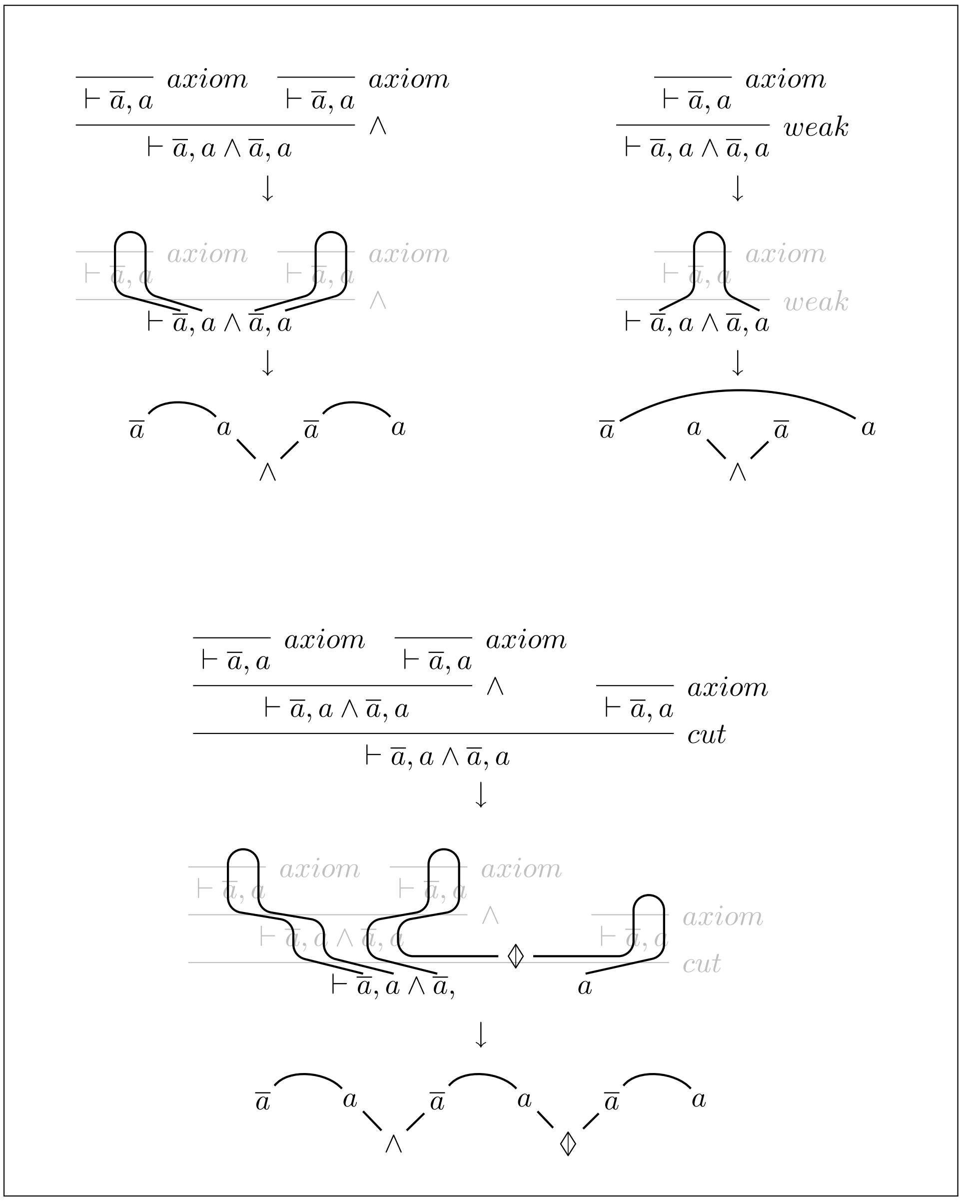
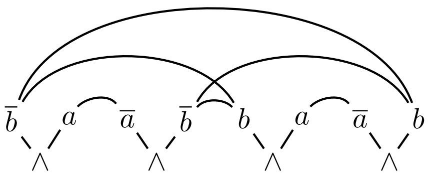


Table 3.1: Translating sequent proofs into prenets

Theorem 3.2.4. *Every prenet that comes from a sequent proof is correct. Conversely, if we restrict the number of links between two nodes to at most* 1*, every correct prenet is sequentializable.*

One of the problems with finding a complete correctness criterion, which also takes into account the exact number of links between two nodes, is that sequentializability depends strongly on the underlying formalism. The prenet



is one example of a proof net that does not come from a sequent calculus proof. However, it can be formalized in the calculus of structures.

Cut Elimination and Normal Forms

Gentzen's Hauptsatz allows to transform any sequent proof in classical logic to a sequent proof without any applications of the cut rule, resulting in normal forms. Classical proof nets may contain cuts as well, so the question arises naturally, how those can be eliminated, and whether this elimination results in normal forms.

Cut elimination in classical proof nets is very similar to cut elimination in proof nets for multiplicative linear logic. In both systems, the elimination of a cut between complex formulas is defined by

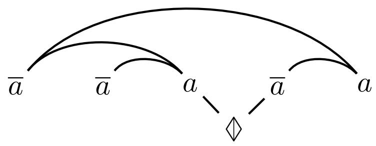
$$L \rhd (A \wedge B) \between (\overline{A} \vee \overline{B}), \Gamma \to L \rhd A \between \overline{A}, B \between \overline{B}, \Gamma$$

and

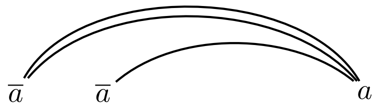
$$L \rhd (A \vee B) \between (\overline{A} \wedge \overline{B}), \Gamma \to L \rhd A \between \overline{A}, B \between \overline{B}, \Gamma \ .$$

When a cut between an atom and its dual is reduced, the paths "through the cut" are counted, and each such path becomes a link.

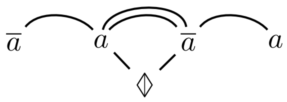
Example 3.2.5. The proof net



reduces to



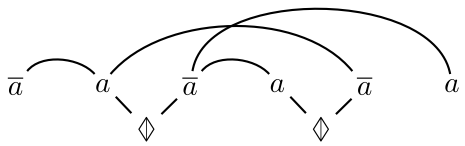
A special case arises when links connect the atoms of the cut:



There are several possible ways to define a cut elimination for this net. The simplest solution is to ignore the two links across the cut, such that the reduced net contains exactly one link. Another possibility is to allow paths that use the crossing links at most a fixed number of times. If we allow at most three uses, the number of links in the reduced net will be $1 + 2^1 + 2^2 + 2^3 = 15$. However, the least problems arise when those links are ignored.

Unfortunately, whichever solution we decide to use, cut elimination for classical proof nets is not confluent.

Example 3.2.6. We look at ways to reduce the following net (ignoring crossing links):



When we reduce the right cut first, and then the left one, the result is:

The other sequence of reductions yields the proof net

which contains one more link.

When we define proof nets for intuitionistic logic and a cut elimination procedure for them, we will focus our attention to making it confluent.

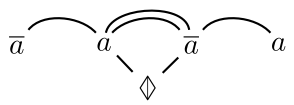
Chapter 4

Proof Nets for Intuitionistic Logic

The main goal of this thesis is the development of proof nets for intuitionistic logic. The general strategy is to mimic basic aspects of proof nets for other logics, e.g. classical logic or intuitionistic linear logic, and combine and generalize them to a proof system.

At the beginning of our exploration stand the definitions of the basic concepts that we will make use of. Hence we will have a look at trees that natually correspond to formulas, and at nets that are constructed out of these trees. As before, there will be special trees corresponding to cut formulas.

An important contribution of this thesis is the idea to attach labels to each link of a net. All types of proof nets developed before treated all labels exactly the same way. This gave rise to some ad hoc decisions that are not natural. Remember for example again the following classical proof net:



We have already remarked that there are several possible ways to reduce this cut. Loop-killing (i.e. brushing off the links crossing the cut) may result in the most compact theory, but it is not obvious that this is the most natural way to deal with the situation.

Link labels allow us to make use of the computational flair of proofs in intuitionistic logic. When we think of proofs by means of λ-terms, an important aspect of computations is that different occurrences of the same variable must be clearly separated. When a cut with crossing links is reduced, we allow basically all uses of crossing links, as long as they respect this separation. This provides a natural interpretation of cut elimination, and it can be achieved by adequate link labels.

However, this liberal treatment of links across cuts may easily result in an infinite number of paths through a cut, and hence to infinite nets. We will discuss a translation of λ-terms into proof nets that mimics the separation idea presented above and at the same time guarantees strong properties of the resulting nets, as for example that they do not become infinite when cuts are reduced.

Which leads us to the second subject of our investigations: cut elimination. As proof nets are supposed to constitute a proof formalism, it is natural to look for normal forms. While the cut elimination procedures for all calculi considered so far are normalizing and terminating, confluence does not always hold, as we have seen in the last chapter. We will define a cut elimination procedure that is not only terminating, but also uniquely normalizing. Although it might be tempting to reach this goal by a close connection of cut elimination to the reduction of λ-terms, we will show that there is no way to define cut elimination in such a way that it simulates these reduction procedures. We will, however, present classes of situations in which a reduction step of a λ-term corresponds exactly, or almost, to a cut elimination step for the corresponding proof net.

4.1 Basic Concepts

Before we can construct proof nets for intuitionistic logic, we have to lay the necessary foundations. We start with a formal introduction of trees, making many of the intuitive notions from chapter 3 explicit. With this basis we can go on to the definition of prenets for intuitionistic logic, which will be slightly more complicated than in the classical case, but also much more expressive.

Notation 4.1.1. We regard a function $f\colon X \to Y$ as the corresponding subset $\{(x,y) \mid f(x) = y\}$ of $X \times Y$. A partial function $f\colon X \rightharpoonup Y$ is a subset of a function in $X \to Y$, such that for two functions $f, g\colon X \rightharpoonup Y$ with disjoint domains, $f \cup g$ is again a partial function from X to Y.

The restriction of f to a subset X' of X is $f|_{X'} := f \cap (X' \times Y)$. We write

$$\mathrm{dom}(f) = \{x \in X \mid \exists y \in Y : (x,y) \in f\}$$

and

$$\mathrm{im}(f) = \{y \in Y \mid \exists x \in X : (x,y) \in f\}$$

for the *domain* and the *image* of a partial function $f\colon X \rightharpoonup Y$.

When we explicitly write down functions, we use the slightly more intuitive notation

$$f = \{1 \mapsto 5, 2 \mapsto 3, \ldots\}$$

to indicate that f maps 1 to 5, 2 to 3 and so on.

4.1.1 Trees

We regard a tree as a function mapping each path in the tree to the label of the respective node. Before we make this precise, shortly recall the notion of paths:

Notation 4.1.2. Let X be a set. The set $X^* := \{\varepsilon\} \cup \bigcup_{n\geq 1} X^n$ is the set of *finite paths* over X, where the symbol $\varepsilon \in X^*$ denotes the *empty path*. If $x_1, x_2, \ldots, x_n$ are elements of X, we write $x_1 x_2 \ldots x_n$ for the tuple $(x_1, x_2, \ldots, x_n) \in X^n$.

We use the canonical identification $X^m \times X^n = X^{m+n}$ for $m, n \in \mathbb{N}$, such that the *concatenation* $\pi\rho$ of two paths $\pi \in X^m$ and $\rho \in X^n$ is the element $(\pi, \rho) \in X^m \times X^n$.

If Y is a set of paths over X, then $\pi \in Y$ is called *maximal* (wrt. Y), if

$$\forall \rho \in X^* : \ \pi\rho \in Y \implies \rho = \varepsilon.$$

We will use trees to represent the types, i.e. formulas, occurring in terms. All trees will be binary, with logical connectors at the inner nodes and atoms at the leaves. Additionally, we again allow special formulas with the main connector $\between$, called *cut*.

As it will later, when we construct proof nets, be important that all occurring leaves are unambiguously distinguishable, we add some further information, stored in the second and third component of each node label. For the time being, it may help to just ignore this and not to try and see a deeper meaning behind the concrete information provided in the examples.

Definition 4.1.3. A *(binary) tree domain* is a finite set $D \subseteq \{1,2\}^*$ with the following properties:

- D is closed under prefixes:

$$\varepsilon \in D \quad \text{and} \quad \forall \pi \in \{1,2\}^* : \forall n \in \{1,2\} : \pi.n \in D \implies \pi \in D$$

- D is strictly binary:

$$\forall \pi \in D : \pi \text{ is maximal wrt. } D \text{ or } \pi.1, \pi.2 \in D$$

Furthermore let $\hat{\mathcal{A}} = \{\hat{a} \mid a \in \mathcal{A}\}$ be the set of *duals* of atoms, and let $\mathcal{N} = (\mathcal{A} \cup \hat{\mathcal{A}} \cup \{\to, \wedge, \vee, \between\}) \times \mathbb{N} \times \{1,2\}^*$ be a set of node labels, consisting of a possibly dualized atom or a binary connector, an index and a path. A *(binary) tree* t is a partial function $t\colon \{1,2\}^* \rightharpoonup \mathcal{N}$ such that

- $\mathrm{dom}(t)$ is a tree domain,
- a cut symbol may only appear at the root, i.e. $t(\pi) = (\between, n, \rho)$ implies $\pi = \varepsilon$, and

- the leaves are labeled with (possibly negated) atoms, and the inner nodes with logical operators: $t(\pi) \in (\mathcal{A} \cup \overline{\mathcal{A}}) \times \mathbb{N} \times \{1,2\}^*$ holds if and only if π is maximal wrt. $\mathrm{dom}(t)$.

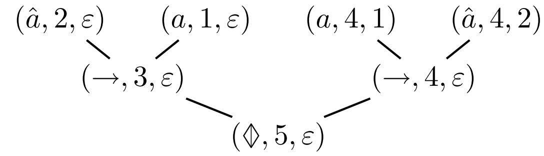
The set of all trees is called $\mathcal{T}$.

We draw trees with the leaves on top, and whenever a node has two children, the first one is drawn left of the second one.

Example 4.1.4. The function

$$
\begin{array}{rcrcll}
t & = & \{ \quad \varepsilon & \mapsto & (\diamondbar, 5, \varepsilon) & , \\
 & & 1 & \mapsto & (\to, 3, \varepsilon) & , \\
 & & 1.1 & \mapsto & (\hat{a}, 2, \varepsilon) & , \\
 & & 1.2 & \mapsto & (a, 1, \varepsilon) & , \\
 & & 2 & \mapsto & (\to, 4, \varepsilon) & , \\
 & & 2.1 & \mapsto & (a, 4, 1) & , \\
 & & 2.2 & \mapsto & (\hat{a}, 4, 2) & \}
\end{array}
$$

is a tree, having the following graphical representation:



Notation 4.1.5. By abuse of notation, a tree t whose root is labelled by a cut ($\diamondbar$) will itself be called a *cut*. The subset

$$
\mathrm{leaves}(t) := \left\{(a, n, \pi) \in \mathrm{im}(t) \mid a \in \mathcal{A} \cup \overline{\mathcal{A}}\right\} \subseteq \mathrm{im}(t)
$$

of nodes labeled by an atom is exactly the set of *leaves* of t. Given a tree t and a path $\pi \in \mathrm{dom}(t)$, the *subtree* $t.\pi$ of t at position π is defined as

$$
t.\pi := \{(\rho \mapsto l) \mid (\pi\rho \mapsto l) \in t\} \ .
$$

Example 4.1.6. The tree from example 4.1.4 has the leaves

$$
\mathrm{leaves}(t) = \{(\hat{a}, 2, \varepsilon), (a, 1, \varepsilon), (a, 4, 1), (\hat{a}, 4, 2)\} \ .
$$

It has exactly six proper subtrees:

$$
\begin{array}{rcrcll}
t.1 & = & \{ \quad \varepsilon & \mapsto & (\to, 3, \varepsilon) & , \\
 & & 1 & \mapsto & (\hat{a}, 2, \varepsilon) & , \\
 & & 2 & \mapsto & (a, 1, \varepsilon) & \}
\end{array}
\qquad
\begin{array}{rcrcll}
t.2 & = & \{ \quad \varepsilon & \mapsto & (\to, 4, \varepsilon) & , \\
 & & 1 & \mapsto & (a, 4, 1) & , \\
 & & 2 & \mapsto & (\hat{a}, 4, 2) & \}
\end{array}
$$

$$
t.11 = \{ \quad \varepsilon \mapsto (\hat{a}, 2, \varepsilon) \quad \} \qquad t.21 = \{ \quad \varepsilon \mapsto (a, 4, 1) \quad \}
$$

$$
t.12 = \{ \quad \varepsilon \mapsto (a, 1, \varepsilon) \quad \} \qquad t.22 = \{ \quad \varepsilon \mapsto (\hat{a}, 4, 2) \quad \}
$$

When we construct prenets, we will sometimes combine two trees to form one big tree. This operation is parallel to the combination of formulas, where e.g. A and B are combined to $A \vee B$ or $A \wedge B$. Furthermore, we will use the "dualization" of trees, which simply exchanges dualized and non-dualized atoms.

Notation 4.1.7. Let $t_1, t_2 \in \mathcal{T}$ be two trees and let $\star \in \{\rightarrow, \wedge, \vee, \between\}$. By $\hat{t}_1$ or $t_1\widehat{\ }$, we denote the unique tree with $\hat{t}_1(\pi) = (a, n, \rho)$, if

- $a \notin \mathcal{A} \cup \hat{\mathcal{A}}$ and $\hat{t}_1(\pi) = t_1(\pi)$, and
- $t_1(\pi) = (a_1, n, \rho)$, $a_1 \in \mathcal{A} \cup \hat{\mathcal{A}}$ and $a = \hat{a}_1$ or $a_1 = \hat{a}$.

The expression $t_1 \star t_2$ denotes a tree with

- $(t_1 \star t_2)(\varepsilon) = (\star, n, \varepsilon)$ for some fresh $n \in \mathbb{N}$,
- $(t_1 \star t_2).1 = t_1$, and
- $(t_1 \star t_2).2 = t_2$.

Example 4.1.8. Consider the three trees

$$
\begin{aligned}
t_1 = \{ \quad & \varepsilon \mapsto (\hat{a}, 2, \varepsilon) \quad \} \\
t_2 = \{ \quad & \varepsilon \mapsto (a, 1, \varepsilon) \quad \} \\
t_3 = \{ \quad & \varepsilon \mapsto (\rightarrow, 4, \varepsilon) \ , \\
& 1 \mapsto (a, 4, 1) \ , \\
& 2 \mapsto (\hat{a}, 4, 2) \quad \}
\end{aligned}
$$

with the following graphical representations:

$$
t_1\colon (\hat{a}, 2, \varepsilon) \qquad t_2\colon (a, 1, \varepsilon) \qquad t_3\colon
\begin{tikzcd}[arrows=dash]
(a,4,1) \arrow[dr] & & (\hat{a},4,2) \arrow[dl] \\
& (\rightarrow, 4, \varepsilon) &
\end{tikzcd}
$$

Then the tree

$$
\begin{tikzcd}[arrows=dash]
(\hat{a},2,\varepsilon) \arrow[dr] & & (a,1,\varepsilon) \arrow[dl] & & (a,4,1) \arrow[dr] & & (\hat{a},4,2) \arrow[dl] \\
& (\rightarrow,3,\varepsilon) \arrow[drr] & & & & (\rightarrow,4,\varepsilon) \arrow[dll] & \\
& & & (\between,5,\varepsilon) & & &
\end{tikzcd}
$$

is of the form $(t_1 \rightarrow t_2) \between t_3$, and

$$
\begin{tikzcd}[arrows=dash]
(a,2,\varepsilon) \arrow[dr] & & (\hat{a},1,\varepsilon) \arrow[dl] \\
& (\rightarrow,4,\varepsilon) &
\end{tikzcd}
$$

is one of the form $(t_1 \rightarrow t_2)\widehat{\ }$.

Now it is almost obvious how to define the tree representing a formula. The only specialty is how to use the additional information in the nodes, and when to use elements of $\mathcal{A}$ or $\hat{\mathcal{A}}$ for the leaves. For the latter, we may regard the difference between both possibilities as the difference between atoms in positive and negative contexts.

Definition 4.1.9. Let $A \in \mathcal{F}$ be a formula, $n \in \mathbb{N}$, $\pi \in \{1,2\}^*$ a path. We define the tree $\mathbf{T}(A, n, \pi)$ as follows:

- If $A = a \in \mathcal{A}$ is an atomic formula, then $\mathbf{T}(a, n, \pi) = \{\varepsilon \mapsto (a, n, \pi)\}$.
- If otherwise $A = A_1 \vee A_2$, then $\mathbf{T}(A, n, \pi)$ is the unique tree t, such that
  - $t(\varepsilon) = (\vee, n, \pi)$,
  - $t.1 = \mathbf{T}(A_1, n, \pi.1)$, and
  - $t.2 = \mathbf{T}(A_2, n, \pi.2)$.
- Analogously, if $A = A_1 \wedge A_2$, $\mathbf{T}(A, n, \pi)$ is the unique tree t, such that
  - $t(\varepsilon) = (\wedge, n, \pi)$,
  - $t.1 = \mathbf{T}(A_1, n, \pi.1)$, and
  - $t.2 = \mathbf{T}(A_2, n, \pi.2)$.
- As the left subformula of an implication is in a negative context, $\mathbf{T}(A_1 \to A_2, n, \pi)$ is the unique tree t, such that
  - $t(\varepsilon) = (\to, n, \pi)$,
  - $t.1 = (\mathbf{T}(A_1, n, \pi.1))\widehat{\ }$, and
  - $t.2 = \mathbf{T}(A_2, n, \pi.2)$.

We will often use the abbreviation $\mathbf{T}(A, i) := \mathbf{T}(A, i, \varepsilon)$.

Conversely, every tree that is not a cut determines a unique type:

Definition 4.1.10. The type $\mathbf{ty}(t)$ coded in a non-cut tree t is defined as follows:

- If t consists of only one node, i.e. if $t(\varepsilon) = (a, n, \pi)$, or $t(\varepsilon) = (\hat{a}, n, \pi)$ for some atom $a \in \mathcal{A}$, then $\mathbf{ty}(t) = a$.
- Otherwise $t(\varepsilon) = (\star, n, \pi)$ for some connector $\star \in \{\to, \wedge, \vee\}$, and $\mathbf{ty}(t) = \mathbf{ty}(t.1) \star \mathbf{ty}(t.2)$.

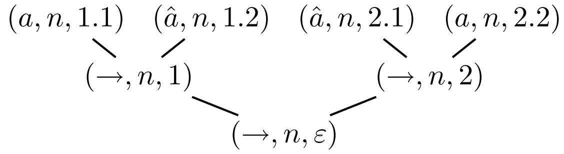
With these two functions, we can move back and forth between types and trees:

Lemma 4.1.11. *The function* **ty** *is left inverse for all functions* $\mathbf{T}(\cdot, n, \pi)$, *where* $n \in \mathbb{N}$ *and* $\pi \in \mathbb{N}^*$, *i.e.*

$$\forall A \in \mathcal{F} : \mathbf{ty}(\mathbf{T}(A, n, \pi)) = A \ .$$

Proof. This follows directly by induction on the structure of A. □

Example 4.1.12. Any tree assigned to the type $(a \to a) \to a \to a$ is of the form



for some $n \in \mathbb{N}$, and the type assigned to all these trees is $(a \to a) \to a \to a$.

The additional information in the nodes is important for no other reason but to make all leaves within a tree, and later on all leaves in the trees of a prenet, distinguishable. For example, you may think of the second component as a counter for different trees, and of the third component as a position in this tree. As the distinction between all nodes is automatically done when we actually draw trees, we exclude this extra information in all further graphics.

Definition 4.1.13. Two trees t and t' are *equivalent*, if both are not cuts, $\mathbf{ty}(t) = \mathbf{ty}(t')$, and all leaves are labeled by the same element of $\mathcal{A} \cup \hat{\mathcal{A}}$, i.e.

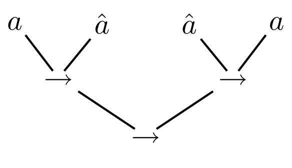
$$\forall \pi \in \mathrm{dom}(t) : \ t(\pi) \in \mathrm{leaves}(t) \wedge t(\pi) = (a, n, \rho) \implies t'(\pi) = (a, n', \rho') \ ,$$

or if both are cuts, and $t.1$ and $t'.1$ are equivalent, as well as $t.2$ and $t'.2$.

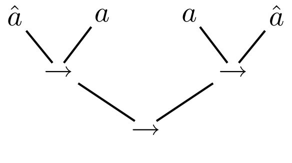
They are *complementary*, if they are not cuts, and if t and $\hat{t}'$ are equivalent.

When we use trees in further examples, we usually give their graph representation only. Additionally, we present them up to equivalence, i.e. we leave out the additional information in the leaves.

Example 4.1.14. All trees from example 4.1.12 are equivalent and will simply be drawn as:



Every tree that is complementary to those trees is of the following form:



4.1.2 Prenets

When working with proof nets, it is a tradition to call the general nets that one wants to consider either "prenets" or "proof structures", and to call only those "proof nets" that are considered proofs in the new calculus.

We are now at the point to give a formal definition of prenets for intuitionistic logic. As mentioned before, we try to find an intuitionistic analog to the notion of prenets for classical logic, which we introduced in chapter 3. This means that our prenets will have a similar shape. They consist mainly of a sequent forest and some links between the leaves of these trees. Where the presented classical prenets may already have multiple links between two leaves, we go one step further and attach a label to each link.

This does not only make the links distinguishable, but will be the key ingredient in the cut elimination procedure for intuitionistic prenets.

Definition 4.1.15. Let $\mathcal{L} = (\{+,-\} \times \mathcal{V} \times \mathbb{N})^*$ be a set of *link labels*. A *prenet* $L \rhd t \,|\, C$ consists of a tree $t \in \mathcal{T}$, a set $C \subset \mathcal{T}$ of cuts, and a partial function $L\colon \mathcal{N} \times \mathcal{N} \rightharpoonup \mathfrak{P}(\mathcal{L})$, called *linking*, such that the following conditions hold:

(0) The main tree t is not a cut: $t(\varepsilon) \neq \Diamond$

(1) Each cut connects two complementary subtrees:

$$(t_1 \Diamond t_2) \in C \implies t_1 \text{ and } t_2 \text{ are complementary}$$

(2) The different trees in the prenet do not share any leaves:

$$t', t'' \in C \cup \{t\} \implies (t' = t'' \;\vee\; \mathrm{leaves}(t') \cap \mathrm{leaves}(t'') = \emptyset)$$

(3) The function L is defined only on the leaves of trees in the prenet:

$$\mathrm{dom}(L) \subseteq \left(\bigcup_{t' \in \{t\} \cup C} \mathrm{leaves}(t') \right) \times \left(\bigcup_{t' \in \{t\} \cup C} \mathrm{leaves}(t') \right)$$

(4) Each link connects dual atoms, or two instances of $\hat{\bot}$:

$$((a,i,\pi),(b,j,\rho)) \in \mathrm{dom}(L) \implies a = \hat{b} \vee a = b = \hat{\bot}$$

We extend the notion of *leaves* to prenets and write

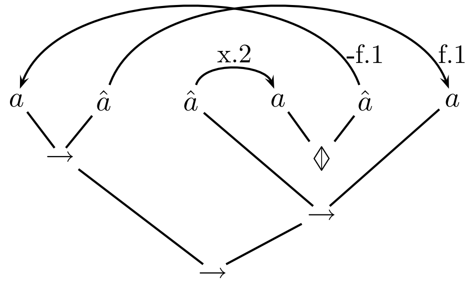
$$\mathrm{leaves}(L \rhd t \,|\, C) = \bigcup_{t' \in \{t\} \cup C} \mathrm{leaves}(t') \; .$$

Notation 4.1.16. By a slight abuse of notation, we will identify a linking $L\colon \mathcal{N} \times \mathcal{N} \rightharpoonup \mathfrak{P}(\mathcal{L})$ of a prenet N with the set

$$\left\{(l, l', \sigma) \mid \sigma \in L(l, l)\right\}.$$

The elements of this set are called the *links* of the prenet.

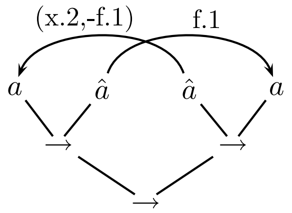
Example 4.1.17. The following graphics depicts a simple prenet.



It consists of the main tree, one cut between a and $\hat{a}$, six leaves and three links. The links are labeled by x.2, -f.1 and f.1 instead of (+,x,2), (−,f,1) and (+,f,1) for better readability. If we call the main tree t, the cut c, and the leaves $l_1, \ldots, l_6$, ordered from left to right, then the picture describes the prenet $L \rhd t \mid \{c\}$ with

$$L = \{(l_5, l_1, (-, f, 1)),\ (l_2, l_6, (+, f, 1)),\ (l_3, l_4, (+, x, 1))\}\ .$$

When we define cut elimination for prenets in 4.4, we will see that the cut elimination of this prenet yields the prenet



with only two links, that bear the labels ((+,x,2),(−,f,1)) and (+,f,1), respectively.

Remark 4.1.18. Following our simplified graphical representation of trees and prenets, we will never look at the additional information stored in the second and third component of the nodes. In particular, two prenets will be considered identical, if they differ only in the additional information.

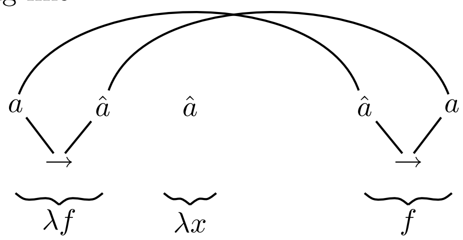
4.2 Typed λ-Terms and Intuitionistic Prenets

Now that we have developed a notion of prenets, we can spend a thought at the question how to relate λ-terms to these prenets. The idea is to use a direct translation of terms into prenets. To keep things simple, we will first restrict ourselves to simply typed λ-terms. This way, we can analyze the main concepts without being overwhelmed by never ending case distinctions.

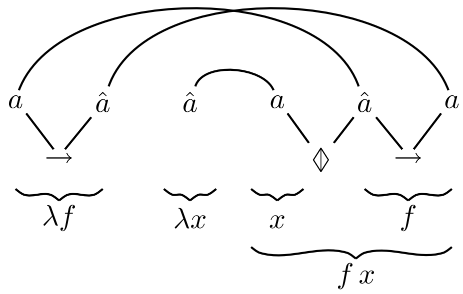
4.2.1 Translating Simply Typed λ-Terms into Prenets

The idea of translating λ-terms into intuitionistic prenets is quite straightforward: It is natural to associate to each variable a tree corresponding to its type. It is also natural to associate to each binder a tree corresponding to the negative type. We use links to represent the binding structure, and cuts for applications.

Example 4.2.1. The simply typed λ-term $\lambda fx.f$ could roughly be translated into something like



and $\lambda fx.fx$ into



Both of these are not real prenets, of course, but they show what we are aiming at.

We will now proceed to do an inductive translation of simply typed λ-terms. Here, we are finally at the point where the additional information at the nodes of trees come into play. To be able to use it properly, we make the following convention:

Notation 4.2.2. When we regard a λ-term e, we assume without loss of generality, that no variable is bound more than once in e. This situation can always be reached by a sequence of α-conversions.

Additionally, we mark all variable and constant occurrences in e with a unique positive natural number. We write v^i for the occurrence of the variable v in e marked by the number i. For example, the term $e = \lambda f.\lambda x.f\,(f\,x)$ might be marked as $\lambda f^5.\lambda x^4.f^1(f^2\,x^3)$. This way, we are able to distinguish any two variable occurrences in e.

Definition 4.2.3. We define the prenet $\mathbf{N}(e)$ associated to a simply typed λ-term e inductively over the structure of e. To build up the prenet, we also assign to every subterm e' a function $l_{e'}\colon \mathcal{V} \to \mathfrak{P}(\mathcal{L})$, that memorizes which

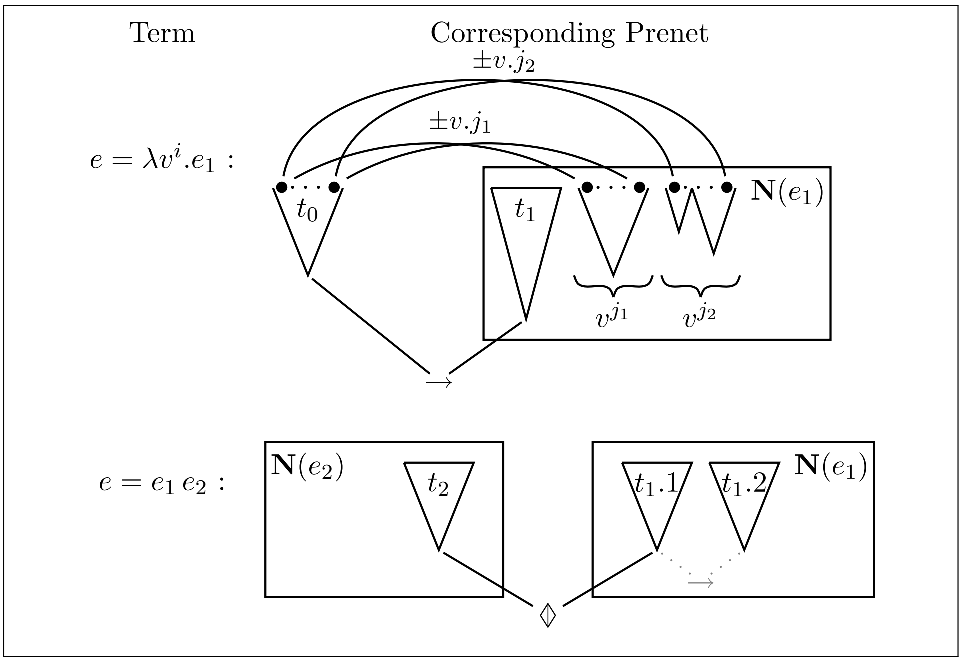


Table 4.1: Translating simply typed lambda terms into prenets

leaves in the prenet of e' were created for which free variable of e'. However, this function will not be used beyond this definition.

Table 4.1 illustrates the composition of prenets in the non-variable cases.

- If $e = v^i$ is a variable, and $t = \mathbf{T}(\tau(v), i)$, then we define

$$\mathbf{N}(e) := \emptyset \rhd t \,|\, \emptyset$$

and

$$l_e(w) = \begin{cases} \text{leaves}(t) & \text{if } w = v \\ \emptyset & \text{if } w \neq v \end{cases}$$

- If $e = \lambda v^i.e_1$ is an abstraction, with $\mathbf{N}(e_1) = L_1 \rhd t_1 \,|\, C_1$ and $t_0 = \mathbf{T}(\overline{\tau(v)}, i)$, then the prenet for e is

$$\mathbf{N}(e) := L \rhd t_0 \rightarrow t_1 \,|\, C_1$$

where corresponding leaves in t_0 and t_1 are linked in such a way, that each link starts at a dualized atom, i.e.

$$\begin{aligned} L = L_1 &\cup \{((\hat{a}, i, \pi), (a, j, \pi), (+, v, j)) \in \text{leaves}(t_0) \times l_{e_1}(v) \times \mathcal{L}\} \\ &\cup \{((\hat{a}, j, \pi), (a, i, \pi), (-, v, j)) \in l_{e_1}(v) \times \text{leaves}(t_0) \times \mathcal{L}\} \end{aligned}$$

and

$$l_e(w) = \begin{cases} \emptyset & \text{if } w = v \\ l_{e_1}(w) & \text{if } w \neq v \end{cases}$$

- If $e = e_1\, e_2$ is an application, and

$$\begin{aligned} \mathbf{N}(e_1) &= L_1 \rhd t_1 \,|\, C_1 \\ \mathbf{N}(e_2) &= L_2 \rhd t_2 \,|\, C_2 \ , \end{aligned}$$

  then

$$\mathbf{N}(e) \quad := \quad L_2 \cup L_1 \rhd t_1.2 \,|\, C_2 \cup \{t_2 \Diamond t_1.1\} \cup C_1$$

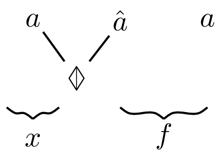
  and $l_e(v) = l_{e_1}(v) \cup l_{e_2}(v)$ for all variables $v \in \mathcal{V}$.

Some remarks on the shape of the links may be appropriate:

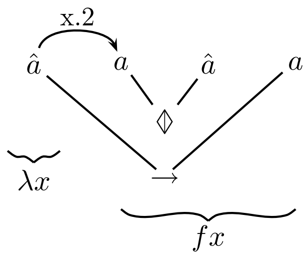
Remark 4.2.4. Until now, the only terms for which links are created are abstractions. In this situation, the nodes coming from the binder are connected to the nodes coming from the bound variables. Each of these connecting links is labeled by the variable name and the occurrence counter, such that we can later distinguish links of the same binder leading to different bound variable occurrences. This will enable us not to mix up these occurrences. The direction of the links is done in such a way that links always lead from dualized to normal atoms. The signs in the labels mark whether a link leaves (+) or enters (−) the binder's tree.

Before we closely examine the translation function, we take a short glance at a few examples.

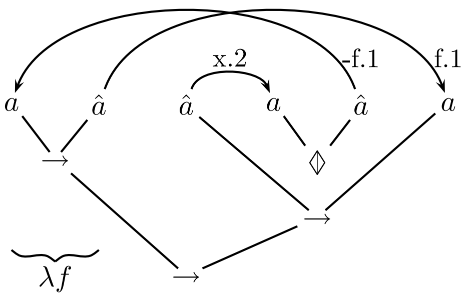
Example 4.2.5. The prenet of the simply typed λ-term $f^1\, x^2$ is



We see that there is only a small difference to example 4.2.1. From this prenet, we can build the one of $\lambda x^3.f^1\, x^2$, namely



and finally the prenet for $\lambda f^4.\lambda x^3.f^1\, x^2$:



Now is the time to consider why the function $\mathbf{N}$ really produces prenets, i.e. why the conditions (1)–(6) of definition 4.1.15 are fulfilled. For this we need a few lemmas:

Lemma 4.2.6. *Let e be a simply typed λ-term, and let $N(e) = L \rhd t \,|\, C$. Then t codes the type of e, i.e. the type of e equals $\mathbf{ty}(t)$.*

Proof. By easy induction on the structure of e. □

Lemma 4.2.7. *If $e = e_1\, e_2$ is a simply typed λ-term, and if the prenets of e_1 and e_2 are N_1 and N_2, then N_1 and N_2 are disjoint, meaning* $\mathrm{leaves}(N_1) \cap \mathrm{leaves}(N_2) = \emptyset$.

Proof. If $l_1 \in \mathrm{leaves}(N_1)$, then either $l_1 \in \mathrm{leaves}(\mathbf{T}(\tau(v), i_1))$ for some variable occurrence v^{i_1} in e_1, or $l_1 \in \mathrm{leaves}(\mathbf{T}(\overline{\tau(v)}, i_1))$ for some variable binder λv^{i_1} in e_1. Either way, l_1 is of the form $l_1 = (a_1, i_1, \pi_1)$. Analogously, any leaf $l_2 \in \mathrm{leaves}(N_2)$ is of the form $l_2 = (a_2, i_2, \pi_2)$, coming from a variable occurrence or binder in e_2. As e_1 and e_2 contain no common variable occurrence, we know that $i_1 \neq i_2$, i.e. $l_1 \neq l_2$. □

Proposition 4.2.8. *Let e be a simply typed λ-term. Then $\mathbf{N}(e)$ is a prenet.*

Proof. We have to verify the conditions in the definition 4.1.15 of prenets for $\mathbf{N}(e) = L \rhd t \,|\, C$.

First of all, it is inductively clear, that t is a tree, and that the elements of C are cuts.

(1) holds by lemma 4.2.6, because the symbol $\between$ cannot appear in a formula. For the other properties, we use induction on the structure of e.

- If e is a variable occurrence, then (2)–(5) hold trivially, because C and $\mathrm{dom}(L)$ are empty.
- Now let $e = \lambda v.e'$ be an abstraction, and $\mathbf{N}(e_1) = L_1 \rhd t_1 \,|\, C_1$.
  - (2) and (3) hold by induction hypothesis, because $C = C_1$, and because t_0 contains only leaves that do not occur in $\mathrm{leaves}(N(e_1))$.
  - For (4) and (5) remark, that $\mathrm{leaves}(\mathbf{N}(e)) \supseteq \mathrm{leaves}(\mathbf{N}(e_1))$, and $\mathrm{dom}(L) \supseteq \mathrm{dom}(L_1)$, so by induction hypothesis we only have to worry about the new links. But these are created between complementary leaves of $\mathbf{N}(e)$, so (4) and (5) hold.

- Let $e = e_1\, e_2$ be an application, and $\mathbf{N}(e_i) = L_i \rhd t_i \,|\, C_i$ for $i \in \{1, 2\}$.
  - As (2)–(3) hold for C_1 and C_2, we only have to look at the new cut $t_2 \between t_1.1 \in C \setminus (C_1 \cup C_2)$.

    As e is well-typed, we know by lemma 4.2.6, that $\mathbf{ty}(t_1) = \hat{T}_2 \vee T_1$ and $\mathbf{ty}(t_1.1) = T_2$ for some types T_1 and T_2, i.e. $\overline{\mathbf{ty}(t_1.1)} = \hat{T}_2 = \mathbf{ty}(t_2)$. So t_2 and $t_1.1$ are complementary, which verifies (2).

    By lemma 4.2.7 and the property (3) of C_1 and C_2, the elements of $C_1 \cup C_2 \cup \{t_1, t_2\}$ do not share any leaves. Obviously, the same holds for the elements of $C_1 \cup C_2 \cup \{t_2 \between t_1.1, t_1.2\} = C \cup \{t\}$.
  - (4) and (5) hold by induction hypothesis, because $L = L_2 \cup L_1$ (which already implies (5)), and so

$$\begin{aligned}\mathrm{dom}(L) &= \mathrm{dom}(L_1) \cup \mathrm{dom}(L_2) \\ &\subseteq (\mathrm{leaves}(N_1) \cup \mathrm{leaves}(N_2)) \times (\mathrm{leaves}(N_1) \cup \mathrm{leaves}(N_2)) \\ &\subseteq \mathrm{leaves}(N) \times \mathrm{leaves}(N)\ .\end{aligned}$$

Thus the proof is complete, and we know that $\mathbf{N}(e)$ is a prenet. □

4.2.2 Translating Typed λ-Terms into Prenets

Now that we have a first impression on how prenets work, we can extend the translation function $\mathbf{N}$ to all λ-terms.

Definition 4.2.9. We define the prenet $\mathbf{N}(e)$ associated to a λ-term e inductively over the structure of e. As a basis, we use the translation from definition 4.2.3, and extend it by the following cases, which are illustrated in table 4.2:

- If $e = \mathrm{pair}^i\, e_1\, e_2$ is a pairing, where $e_1 : A_1$ and $e_2 : A_2$, then e is treated as an application $e = (e'\, e_1)\, e_2$, where $\mathbf{N}(e') = L' \rhd t' \,|\, \emptyset$ is a prenet, such that $t' = \mathbf{T}(A_1 \to (A_2 \to (A_1 \wedge A_2)), i)$, and corresponding leaves in the subtrees coming from the two instances of A_1, as well as those in the subtrees coming from the two instances of A_2, are linked:

$$\begin{aligned}L' = &\{((\hat{a}, i, 1\pi), (a, i, 221\pi), \varepsilon) \in \mathrm{leaves}(t') \times \mathrm{leaves}(t') \times \mathcal{L}\} \\ &\cup \{((\hat{a}, i, 221\pi), (a, i, 1\pi), \varepsilon) \in \mathrm{leaves}(t') \times \mathrm{leaves}(t') \times \mathcal{L}\} \\ &\cup \{((\hat{a}, i, 21\pi), (a, i, 222\pi), \varepsilon) \in \mathrm{leaves}(t') \times \mathrm{leaves}(t') \times \mathcal{L}\} \\ &\cup \{((\hat{a}, i, 222\pi), (a, i, 21\pi), \varepsilon) \in \mathrm{leaves}(t') \times \mathrm{leaves}(t') \times \mathcal{L}\}\end{aligned}$$

  Finally, $l_{e'}(v) = \emptyset$ for all $v \in \mathcal{V}$.

- If $e = \pi_1^i\, e_1$ is a projection, where $e_1 : A_1 \wedge A_2$, then e is treated as an application $e = e'\, e_1$, where $\mathbf{N}(e') = L' \rhd t' \,|\, \emptyset$ is a prenet, such that

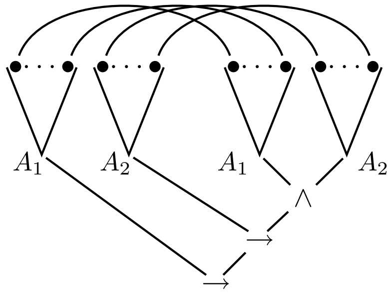
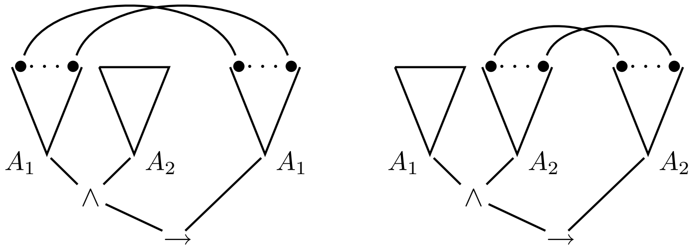
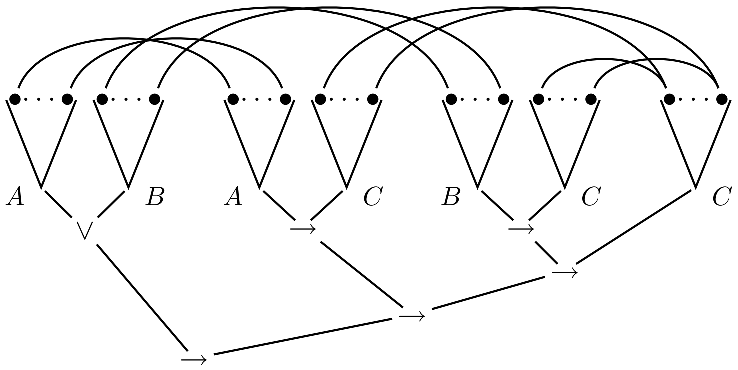
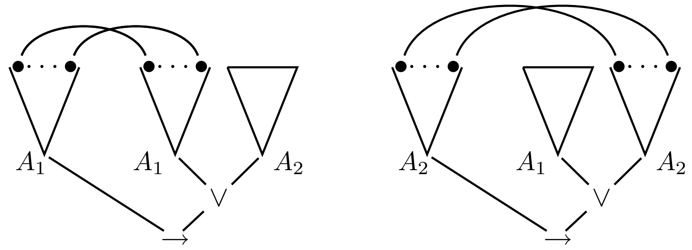
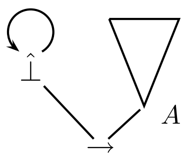
Constant	Corresponding Prenet
$e =$ pair:	
$e = \pi_1$ or π_2:	
$e =$ case:	
$e =$ inl or inr:	
$e =$ null:	

Table 4.2: Translating λ-Terms into Prenets

$t' = \mathbf{T}((A_1 \wedge A_2) \to A_1), i)$, and the leaves in the subtrees coming from the two instances of A_1 are linked:

$$\begin{aligned} L' = \{((\hat{a}, i, 11\pi), (a, i, 2\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\ \cup \{((\hat{a}, i, 2\pi), (a, i, 11\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \end{aligned}$$

Finally, $l_{e'}(v) = \emptyset$ for all $v \in \mathcal{V}$.

- The case $e = \pi_2^i\, e_1$ works analogously, except that $t' = \mathbf{T}((A_1 \wedge A_2) \to A_2), i)$, and the leaves in the subtrees coming from the two instances of A_2 are linked:

$$\begin{aligned} L' = \{((\hat{a}, i, 12\pi), (a, i, 2\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\ \cup \{((\hat{a}, i, 2\pi), (a, i, 12\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \end{aligned}$$

- If $e = \text{inl}^i\, e_1$, where $e : A_1 \vee A_2$, then e is treated as an application $e = e'\, e_1$, where $\mathbf{N}(e') = L' \rhd t' \,|\, \emptyset$ is a prenet, such that $t' = \mathbf{T}(A_1 \to (A_1 \vee A_2), i)$, and corresponding leaves in the subtrees coming from the two instances of A_1 are linked:

$$\begin{aligned} L' = \{((\hat{a}, i, 1\pi), (a, i, 21\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\ \cup \{((\hat{a}, i, 21\pi), (a, i, 1\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \end{aligned}$$

Finally, $l_{e'}(v) = \emptyset$ for all $v \in \mathcal{V}$.

- The case $e = \text{inr}^i\, e_1$ works analog, except that $t' = \mathbf{T}(A_2 \to (A_1 \vee A_2), i)$, and the leaves in the subtrees coming from the two instances of A_2 are linked:

$$\begin{aligned} L' = \{((\hat{a}, i, 1\pi), (a, i, 22\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\ \cup \{((\hat{a}, i, 22\pi), (a, i, 1\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \end{aligned}$$

- If $e = \text{case}^i\, e_1\, e_2\, e_3$, where $e_1 : A \vee B$, $e_2 : A \to C$ and $e_2 : B \to C$, then e is treated as an application $e = ((e'\, e_1)\, e_2)\, e_3$, where $\mathbf{N}(e') = L' \rhd t' \,|\, \emptyset$ is a prenet, such that $t' = \mathbf{T}((A \vee B) \to (A \to C) \to (B \to C) \to C, i)$, and the corresponding leaves in the subtrees coming from the two instances of A, as well as those in the subtrees coming from the two instances of B, are linked, and those in the subtrees coming from the left two instances of C are linked to those of the subtree coming from the rightmost instance of C:

$$\begin{aligned}
L' = &\{((\hat{a}, i, 11\pi), (a, i, 211\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\
&\cup \{((\hat{a}, i, 211\pi), (a, i, 11\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\
&\cup \{((\hat{a}, i, 12\pi), (a, i, 2211\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\
&\cup \{((\hat{a}, i, 2211\pi), (a, i, 12\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\
&\cup \{((\hat{a}, i, 212\pi), (a, i, 222\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\
&\cup \{((\hat{a}, i, 222\pi), (a, i, 212\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\
&\cup \{((\hat{a}, i, 2212\pi), (a, i, 222\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\} \\
&\cup \{((\hat{a}, i, 222\pi), (a, i, 2212\pi), \varepsilon) \in \text{leaves}(t') \times \text{leaves}(t') \times \mathcal{L}\}
\end{aligned}$$

Finally, $l_{e'}(v) = \emptyset$ for all $v \in \mathcal{V}$.

- If $e = \text{null}^i\, e_1$, where $e : A$, then e is treated as an application $e = e'\, e_1$, where $\mathbf{N}(e') = L' \rhd t' \,|\, \emptyset$ is a prenet, such that $t' = \mathbf{T}(\bot \to A, i)$, and there is one link connecting the $\bot$-labeled node to itself:

  $$L' = \{((\hat{\bot}, i, 1), (\hat{\bot}, i, 1), \varepsilon)\}$$

  Finally, $l_{e'}(v) = \emptyset$ for all $v \in \mathcal{V}$.

The images of closed terms under $\mathbf{N}$ are called *sequentializable*.

Remark 4.2.10. The way we treated constants looks a lot like a mixture of variables and abstractions. In fact, we will often take this point of view.

- Let e_1, e_2 and e be typed λ-terms of types A_1, A_2 and $A_1 \wedge A_2$, respectively. Additionally, let v_{π_1}, v_{π_2} and v_{pair} be variables, such that $v_{\pi_1}{:}(A_1 \wedge A_2) \to A_1$, $v_{\pi_2}{:}(A_1 \wedge A_2) \to A_2$, and $v_{\text{pair}}{:}A_1 \to (A_2 \to (A_1 \wedge A_2))$.

  Then the prenets $\mathbf{N}(\text{pair}\, e_1\, e_2)$ and $\mathbf{N}(v_{\text{pair}}\, e_1\, e_2)$ differ only in so far as the second prenet lacks some links between leaves created for v_{pair}.

  Additionally, the prenets $\mathbf{N}(\pi_1\, e)$ and $\mathbf{N}(v_{\pi_1}\, e)$ differ only in so far as the second prenet lacks some links between leaves created for v_{π_1}. The analogous statement is true for $\mathbf{N}(\pi_2\, e)$ and $\mathbf{N}(v_{\pi_2}\, e)$.

  This means, that we can and often will simply consider an instance of one of the constants pair, π_1 and π_2 as a kind of variable, and a term of the form pair $e_1\, e_2$, $\pi_1\, e$ or $\pi_2\, e$ as an application.

  The same holds for the constants case, inl, inr and null, so we also consider these as variables, whenever appropriate.

- The links in the prenets of constants do not carry other labels than ε. We could treat the constants like a combination of binders and variables, and label links going "from left to right" by e.g. $(+, \pi_1, 5)$, and the others by e.g. $(-, \pi_1, 5)$. The following considerations, however, work equally well in both situations, so we may well choose the simpler labels.

As before, we get:

Proposition 4.2.11. *Let e be a λ-term. Then $\mathbf{N}(e)$ is a prenet.*

Proof. The proofs of the lemmas 4.2.6 and 4.2.5 transfer word by word to the extended case.

For the proposition itself, remark that we again have to verify the conditions in the definition 4.1.15.

The cases of variables, abstractions and applications are handled by proposition 4.2.8.

Following our approach to treat constants like normal terms, the remaining case is that $e = c$ is a constant. In this situation, (2) and (3) hold trivially, because C is empty, and (4) and (5) hold because all links are created between complementary leaves of $\mathbf{N}(e)$, if $e \neq$ null, and from a $\hat{\perp}$-leaf in t to itself, if $e =$ null. □

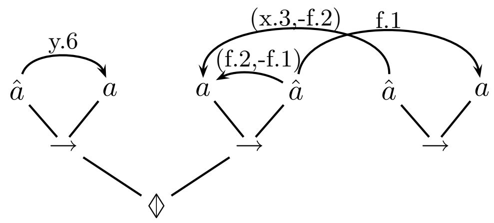
Another well-definedness issue is caused by α-equivalence. We already mentioned that λ-terms are often regarded only up to α-equivalence. Terms are even often given in a variable-free de Bruijn notation [dB72]. Note that two different but α-equivalent λ-terms may correspond to different prenets. However, these prenets differ only by a consistent renaming of link labels. We will see, that the behavior of the prenets during cut elimination does not depend on this difference.

4.3 Properties of Sequentializable Prenets

Whenever one designs prenets for a new logic, it is one of the most interesting questions, which prenets correspond to a proof in a given reference proof system for the logic. While this distinction might sometimes be possible by means of a brute force search, the desired result is a intrinsic property of the prenets, i.e. a criterion for sequenzialization that can be checked without looking at the reference calculus.

The first such criterion for multiplicatice linear logic was already given by Girard [Gir87], and later on many simplified criteria for this calculus were found, e.g. by Danos and Regnier [DR89]. Lamarche [Lam95] examined prenets for intuitionistic linear logic and gave a criterion using a system of polarities, and Lamarche and Straßburger [LS05] found a criterion that applies to a equivalence classes of prenets for classical propositional logic (called $\mathbb{B}$-prenets), but not the full calculus.

Unfortunately, by now there is also no such correctness criterion for intuitionistic prenets. To see the problems with an intrinsic characterization of prenets that are suited as proofs, consider the following prenet:



We will show later, that this prenet can be constructed by a series of cut eliminations from the prenet of the term $(\lambda f.\lambda x.f\ (f\ x))(\lambda y.y)$. As cut elimination will play the part of a proof normalization procedure, we expect our calculus to be closed under cut elimination. However, the above prenet is not the prenet of any λ-term, so it does not correspond to a traditional intuitionistic proof. And even if we only expect the existence of a λ-term whose net contains the same number of links between two given leaves, we will not be able to find such a term.

Instead of giving a complete correctness criterion, we will in this section present several properties of sequentializable prenets, which have a good chance to play an important role in such a criterion. Finally, we will explain why we called the images of terms under **N** "sequentializable". We do so by presenting an algorithm that computes a right-inverse of **N**.

4.3.1 Polarization

The first property that distinguishes prenets coming from a λ-term from many other prenets is whether the main tree of a prenet corresponds to an intuitionistic formula. Lamarche [Lam95], and later on also Retoré [LR96] used a system of polarities to characterize these prenets. They extended the idea of writing a sequent $A, B \vdash C$ as the one-sided sequent $\vdash A^\bullet, B^\bullet, C^\circ$, where the additional markers show which side a formula came from. Although both authors worked on linear logic, the analogous generalization also applies to our system.

Definition 4.3.1. A prenet $L \rhd t \,|\, C$ is called *polarizable*, if its nodes can be decorated with a polarity (we use the symbols $\circ$ and $\bullet$), such that the following rules hold:

(1) Each link connects a $\bullet$ node to a $\circ$ node, or it connects $\hat{\bot}$-labeled $\bullet$ nodes.

(2) The left child of each $\to$-labelled $\circ$ node is a $\bullet$ node, the right child is a $\circ$ node.

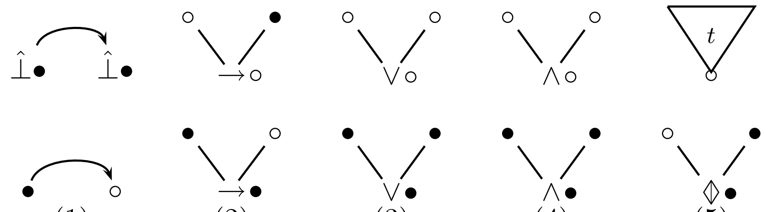
    The left child of each $\to$-labelled $\bullet$ node is a $\circ$ node, the right child is a $\bullet$ node.

(3) Both children of each $\vee$-labelled node have the same polarity as the node itself.

(4) Both children of each $\wedge$-labelled node have the same polarity as the node itself.

(5) Each $\Diamond$-labelled root $c(\varepsilon)$ is a $\bullet$ node, its left child is a $\circ$ node, and its right child is a $\bullet$ node.

The root of t is a $\circ$ node.

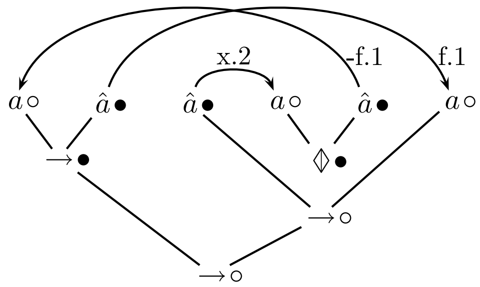


(1) (2) (3) (4) (5)

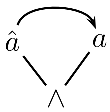
Each decoration of a prenet satisfying the above conditions is called a *polarization*.

A tree t is called *polarizable*, if the prenet $\emptyset \rhd t \,|\, \emptyset$ is polarizable, ignoring rule (5).

Example 4.3.2. The prenet from example 4.1.17 can be uniquely polarized as follows:



The prenet



however does not have a polarization because of a conflict betwee rules (1) and (3).

It is obvious from the shape of the polarization rules that the polarities of two siblings are determined by the polarity of their parent node. So we can directly conclude:

Proposition 4.3.3. *Let N be a prenet. If N is polarizable, then this polarization is unique.*

Example 4.3.2 shows that it is not the case, that every prenet admits a polarization. However we will now show that sequentializable prenets are also polarizable.

Lemma 4.3.4. *Let A be a type. Then $t = \mathbf{T}(A, i)$ is polarizable by the rules of 4.3.1, such that its root is a $\circ$ node.*

Dually, $t' = \mathbf{T}(A, i)^\frown$ is polarizeable by the rules of 4.3.1, such that its root is a $\bullet$-node.

These polarizations can be done in such a way, that each leaf is a $\circ$ node, if and only if it is labeled by a (positive) atom.

Proof. It is obvious from the polarization rules, that any polarization of the root of a tree can be promoted to its leaves. So the only thing we have to show is, that we can produce the required polarization of the leaves.

We prove this by induction on the structure of A.

- If $A = a$ is an atom, there is nothing to show.
- Let $A = A_1 \to A_2$, i.e. $t = t.1 \to t.2$. As the root of t is a $\circ$ node, the root of t_1 is a $\bullet$-node, and the root of t_2 is a $\circ$ node by rule (2).

  As $t.1$ is equivalent to $\mathbf{T}(A_1)^\frown$, and as $t.2$ is equivalent to $\mathbf{T}(A_2)$, the induction hypothesis for $t.1$ and $t.2$ ensures, that we get the desired polarization of the leaves.

  Dually, we can promote a $\bullet$ node at the root of t to a $\circ$ node at the root of t_1, and a $\bullet$ node at the root of t_2, again receiving the result by induction hypothesis.
- Let $A = A_1 \wedge A_2$, i.e. $t = t.1 \wedge t.2$. As the root of t is a $\circ$ node, the roots of t_1 and t_2 are also $\circ$ nodes by rule (2).

  As $t.1$ is equivalent to $\mathbf{T}(A_1)$, and as $t.2$ is equivalent to $\mathbf{T}(A_2)$, the induction hypothesis for $t.1$ and $t.2$ ensures that we get the desired polarization of the leaves.

  Dually, we can promote a $\bullet$ node at the root of t to $\bullet$ nodes at the roots of t_1 and t_2, again receiving the result by induction hypothesis.
- The case $A = A_1 \vee A_2$ is completely analogous to $A = A_1 \wedge A_2$. □

The duality in the above construction of the polarization of complementary trees, together with the uniqueness of polarizations, allows us to conclude:

Corollary 4.3.5. *Let A be a formula, and let i, j any two integers. Then the unique polarization of $\mathbf{T}(A, i)$ marks a node as a $\circ$ node, if and only if the unique polarization of $\mathbf{T}(A, j)^\frown$ marks the corresponding node as a $\bullet$-node.*

Proposition 4.3.6. *Let e be a term. Then $\mathbf{N}(e)$ is polarizable.*

Furthermore, the polarization can be done in such a way, that nodes coming from a variable are polarized dually to the respective nodes coming from this variable's binder.

Proof. We know that there is a unique polarization by rules (1)–(5). We have to show that the links respect rule (1).

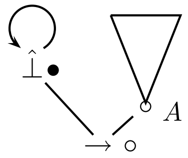
We proceed by induction on the structure of e. Let $\mathbf{N}(e) = L \rhd t \,|\, C$.

- Let $e = v$ be a variable, and $A = \tau(v)$. Since there are no links, there is nothing to show.
- Let $e = \lambda v.e_1$ and let $\mathbf{N}(e_1) = L_1 \rhd t_1 \,|\, C_1$. Because of rules (5) and (2), the root of $t_1 = t.1$ is a $\circ$ node in both $\mathbf{N}(e_1)$ and $\mathbf{N}(e)$. Hence we can apply the induction hypothesis, and the proposition holds for all links in e_1.

  Let t_0 be the tree coming from the binder λv in $\mathbf{N}(e)$. By rules (5) and (2) the root of t_0 is a $\bullet$ node. By lemma 4.3.5, the leaves of t_0 are marked dually to the leaves in e_1 coming from an instance of v. So rule (1) is also respected by all links that are introduced by the abstraction.
- Let $e = e_1\, e_2$, and let $\mathbf{N}(e_i) = L_i \rhd t_i \,|\, C_i$ for $i \in \{1, 2\}$.

  Because of rule (5), the root of $t_2 = c.1$ is a $\circ$ node in both $\mathbf{N}(e_1)$ and $\mathbf{N}(e)$, and the root of $t_1.1 = c.2$ is a $\bullet$ node in both $\mathbf{N}(e_1)$ and $\mathbf{N}(e)$. Hence we can apply the induction hypothesis, and the proposition holds for e_1 and e_2.

  As $\mathbf{N}(e)$ does not contain links that were not already present in $\mathbf{N}(e_1)$ or $\mathbf{N}(e_2)$, rule (1) is fulfilled.
- Let $e = \text{null}$. Then $\mathbf{N}(e)$ can be polarized as follows:

- If e is one of the other constants, the situation is completely analogous to the abstraction case, because any polarization that marks the root as $\circ$ marks the roots of connected subtrees differently, i.e. the duality of the polarization rules allows once again a polarization as required. $\square$

4.3.2 Classical Correctness

Lamarche and Straßburger [LS05] gave a geometric criterion to decide, whether a given classical prenet can be constructed from a classical sequent proof (cf. theorem 3.2.4). As every intuitionistic proof of a formula can also be seen as a classical proof, it is not surprising that intuitionistic prenets fulfill the same criterion.

Definition 4.3.7. A *conjunctive pruning* of a polarized prenet $L \rhd t \,|\, C$ is a triple (t', C', L'), where t' and C' are obtained from t and C by replacing each subtree of type $t_1 \wedge^\circ t_2$, $t_1 \to^\bullet t_2$, $t_1 \vee^\bullet t_2$ or $t_1 \between t_2$ by either t_1 or t_2, and where

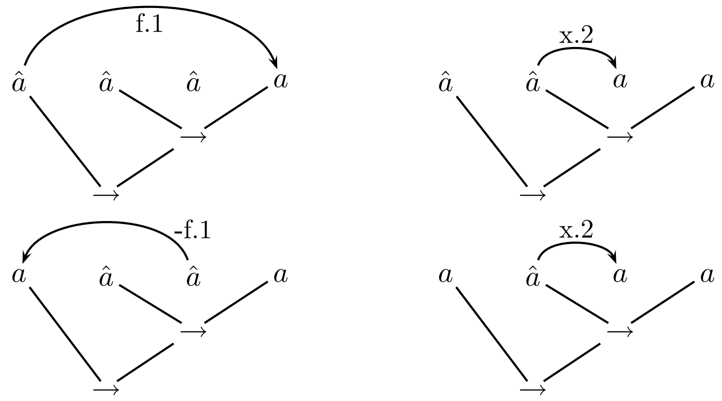
$$L' = L|_{\text{leaves}(\{t'\}\cup C')\times\text{leaves}(\{t'\}\cup C')} \ .$$

When $(t', \emptyset, \emptyset)$ is a conjunctive pruning of $\emptyset \rhd t \,|\, \emptyset$, we call t' the *conjunctive pruning* of t.

A prenet N is *classically correct*, if every conjunctive pruning of N contains at least one link.

Note that a conjunctive pruning is almost again a prenet. The only tender spot is that the elements of C' are not cuts. In [LS05], this distinction is not made, because a prenet may contain more than one non-cut tree.

Example 4.3.8. The following graphs visualize the conjunctive prunings of the prenet from example 4.1.17:



Each of these prunings contains a link, so the original prenet is classically correct.

Proposition 4.3.9. *Let e be a closed term. Then $\mathbf{N}(e)$ is classically correct.*

Proof. We show the classical correctness of $\mathbf{N}(e)$ by induction on the number of cuts in $\mathbf{N}(e)$. Again, we follow remark 4.2.10 to deal with constants. Table 4.3 illustrates the main cases.

- Let e be application-free, but not a constant. Then e is of the form $\lambda v_1 . \dots \lambda v_n . v_i$ with $1 \le i \le n$, and its prenet contains exactly the same links as the prenet of $e' = \lambda v_i . v_i$. Additionally, every pruning of e can be restricted to a pruning of e', and so it suffices to examine this term.

  Let $\mathbf{N}(e') = L \rhd t \,|\, \emptyset$, and let $A = \tau(v_i)$. By construction, t is of the form $t = \mathbf{T}(A, j) \to^\circ \mathbf{T}(A, k)$ for some occurrence markers j and k.

  To prove that every conjunctive pruning of $\mathbf{N}(e')$ contains a link, it suffices to show that every conjunctive pruning of t contains a pair of

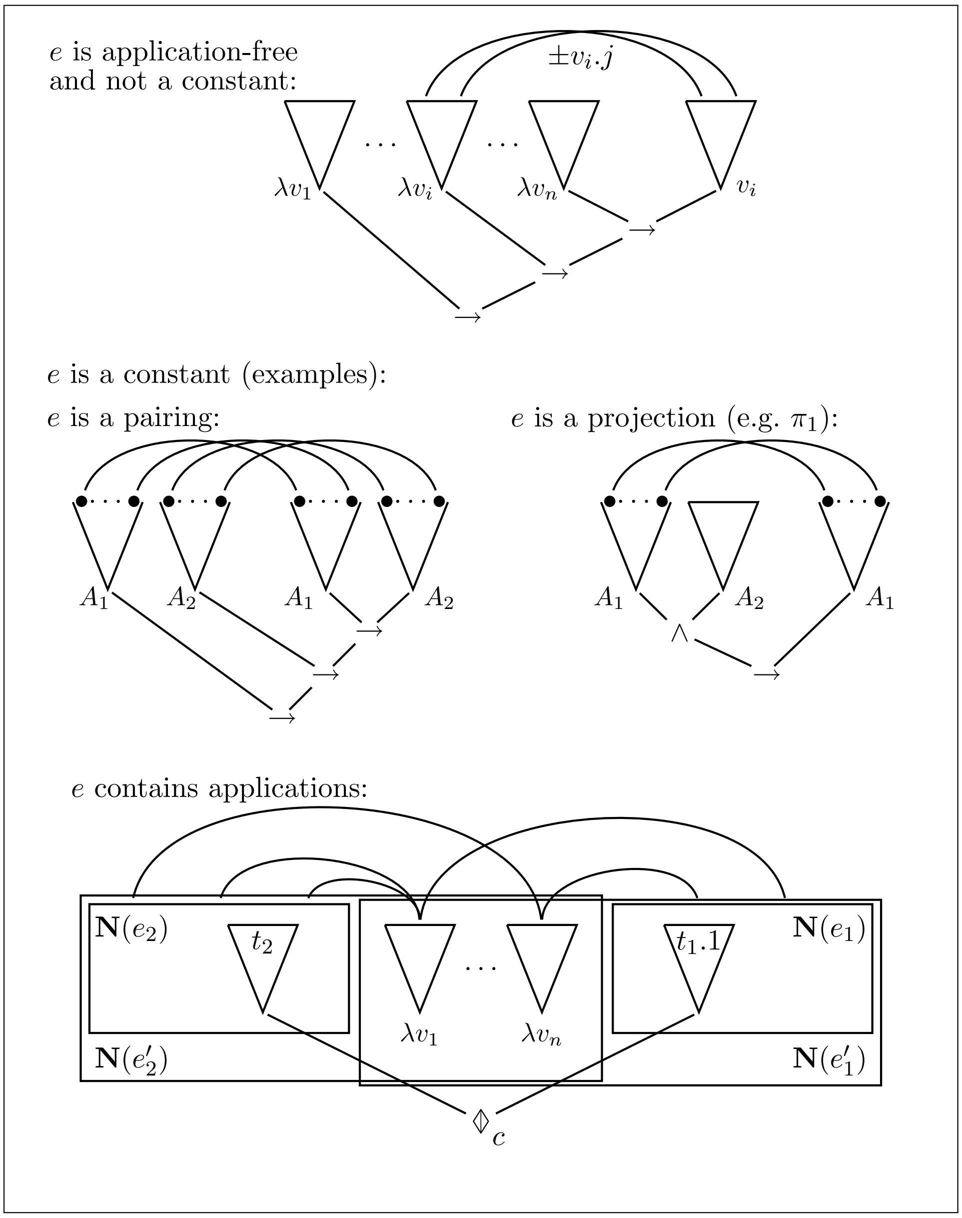


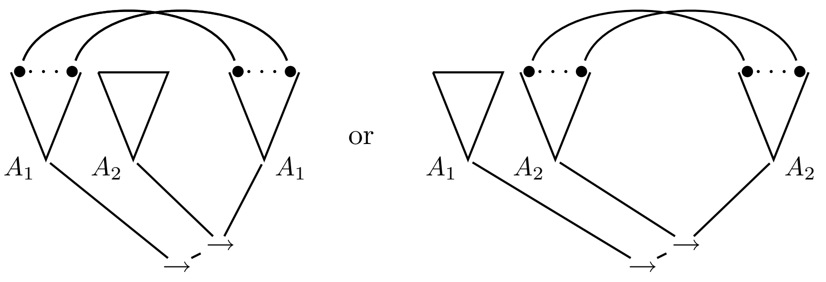
Table 4.3: The Structure of a Closed Term

corresponding leaves in $\mathbf{T}(A,j)^{\frown}$ and $\mathbf{T}(A,k)$, i.e. two leaves $(\hat{a},j,\pi)$ and (a,k,π) (note that the path labels are equal).

We prove this by induction on the structure of A.

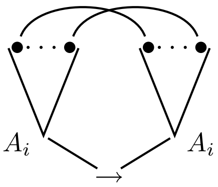
- If A is an atom, then $t = t(1) \to^{\circ} t(2)$ does not contain prunable nodes, so there is only one conjunctive pruning: t itself. The nodes $t(1)$ and $t(2)$ provide the demanded pair.
- If $A = A_1 \to A_2$, then t is of the form $t = (t.11 \to^{\bullet} t.12) \to^{\circ} (t.21 \to^{\circ} t.22)$, and every conjunctive pruning of t is of the form $t' = t'_1 \to^{\circ} (t'_{21} \to^{\circ} t'_{22})$, where t'_{21} and t'_{22} are conjunctive prunings of $t.21$ and $t.22$, respectively, and t'_1 is either a conjunctive pruning of $t.11$ or $t.12$. Let us assume that t'_1 is a conjunctive pruning of $t.11$ (the other case works analogously). As $t'_1 \to^{\circ} t'_{21}$ is a conjunctive pruning of $t.11 \vee t.21$, we can apply the induction hypothesis to conclude that $t'_1 \vee t'_{21}$ contains the wanted pair of leaves, thus so does t'.
- The case of a disjunction type $A = A_1 \vee A_2$ is completely analog to the case of an implication, and in the case of a conjunction type $A = A_1 \wedge A_2$, the only difference is that prunings are of the form $t' = (t'_{11} \wedge^{\circ} t'_{12}) \to^{\circ} t'_2$, but the rest of the argumentation does not change.

• Let $e = \text{pair}$, and $e : A_1 \to A_2 \to (A_1 \wedge A_2)$. Then any conjunctive pruning of $\mathbf{N}(e)$ is a conjunctive pruning of



These prenets are obviously classically correct, if the identically shaped prenets of $\lambda x_1.\lambda x_2.x_1$ and $\lambda x_1.\lambda x_2.x_2$, for variables of type $x_1 : A_1$ and $x_2 : A_2$, are classically correct. We have already shown this in the first part of this proof.

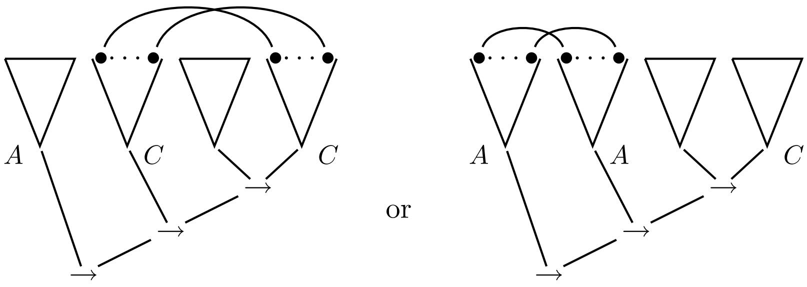
• Let $e = \pi_i$, $i \in \{1,2\}$, and $e : (A_1 \wedge A_2) \to A_i$. Then any conjunctive pruning of $\mathbf{N}(e)$ can be restricted to a conjunctive pruning of the prenet



This prenet is obviously classically correct, if the identically shaped prenet of $\lambda x_i.x_i$, for a variable of type $x_i : A_i$, is classically correct. We have already shown this in the first part of this proof.

- The argumentation for $e = \text{inl}$ and $e = \text{inr}$ is analog to the cases $e = \pi_1$ and $e = \pi_2$.

- Let $e = \text{case}$ and $e : (A \vee B) \rightarrow (A \rightarrow C) \rightarrow (B \rightarrow C) \rightarrow C$. Any conjunctive pruning of e deletes either the leftmost tree coming from A or from B. Without loss of generality, assume that the tree coming from B is deleted. Any such pruning is contained in a conjunctive pruning of



These prenets are obviously classically correct, if the prenets of $\lambda x_1.x_1$ and $\lambda x_2.x_2$, for variables of type $x_1 : A$ and $x_2 : C$, are classically correct. We have already shown this in the first part of this proof.

- Let $e = \text{null}$ and $e : \bot \rightarrow A$. Then any conjunctive pruning of e contains the link connecting $\hat{\bot}$ to itself.

- Let e be a formula containing at least one application. Then e is of the form $\lambda v_1. \dots \lambda v_n.e_1\, e_2$, with $n \geq 0$. The terms $e'_i = \lambda v_1. \dots \lambda v_n.e_i$ are closed and contain strictly less applications than e. So the prenets $\mathbf{N}(e'_i)$ are classically correct by induction hypothesis.

  Let $\mathbf{N}(e) = L \rhd t \,|\, C$, let $c \in C$ be the cut introduced by the application $e_1\, e_2$, and let P be any conjunctive pruning of $\mathbf{N}(e)$. Then there are conjunctive prunings P_1 and P_2 of $\mathbf{N}(e'_1)$ and $\mathbf{N}(e'_2)$, respectively, such that the construction of each P_i agrees with the construction of P on all nodes that $\mathbf{N}(e'_i)$ and $\mathbf{N}(e)$ have in common. So the only choice made for P that is not visible in P_1 or P_2 is the choice at c.

Now, if $c.1$ is deleted in P, P_1 is unaffected by this, and all links of P_1 appear in P. Analogously, if $c.2$ is deleted, the links of P_2 appear in P.

As the prenets $\mathbf{N}(e'_i)$ are classically correct, each P_i contains at least one link. So in any case, P too contains at least one link. □

4.3.3 Paths and Finiteness

In chapter 3, when we discussed proof nets for classical logic, we saw that an atomic cut was eliminated by following all paths through the cut. We follow this idea of paths through a cut and expand it to non-atomic cuts. Although the notion seems to be very close to the one in the context of classical proof nets, there is one big difference: While Lamarche and Straßburger strictly forbade the use of links that cross a cut, and only gave the idea to impose any finite bound on the number of uses of such links, we explicitly encourage following any link any number of times.

Definition 4.3.10. Let $N = L \rhd t \,|\, C$ be a prenet, and let $C' = \{c_1, \ldots, c_k\} \subseteq C$. A triple $(l_0, l'_n, \sigma) \in \text{leaves}(N) \times \text{leaves}(N) \times \mathcal{L}$ is called a *path candidate in N through C'*, if there is a series $l'_0, l_1, l'_1, \ldots, l_{n-1}, l'_{n-1}, l_n \in \text{leaves}(C')$ of leaves such that:

(1) A path candidate does not begin or end in one of the distinguished cuts: $l_0, l'_n \notin \text{leaves}(C')$

(2) It is composed of a series of links: $\forall 0 \leq i \leq n : \; (l_i, l'_i) \in \text{dom}(L)$

(3) These links are connected by cuts, such that if one link ends in a cut, the next starts on the opposite side of the cut:

$\forall 0 \leq i \leq n : \; \exists c \in C' : \; \exists \pi \in \text{dom}(c.1) :$
$(l'_i = c(1\pi) \wedge l_{i+1} = c(2\pi)) \vee (l'_i = c(2\pi) \wedge l_{i+1} = c(1\pi))$

(4) And finally, the label of the path candidate is the concatenation of the labels of all the used links: $\sigma \in L(l_0, l'_0) \times \ldots \times L(l_n, l'_n)$

An explosion of the number of resulting paths is prohibited by a constraint on the labels of the links that paths accumulate:

Definition 4.3.11. Let $\sigma = \sigma_1(-, v, i)\sigma_2(+, v, i)\sigma_3$ be a link label, where $v \in \mathcal{V}$ is a variable, $i \in \mathbb{N}$ is an occurrence marker, and σ_2 does not contain $(\pm, v, j)$ for any $j \in \mathbb{N}$. A *label reduction* step is the procedure of replacing such σ by $\sigma_1\sigma_2\sigma_3$.

Now let σ be any link label. If the normal form of σ under the above reduction relation is not of the form $\sigma_1(-, v, i)\sigma_2(+, v, j)\sigma_3$ with $i \neq j$, then σ is called *well-formed*.

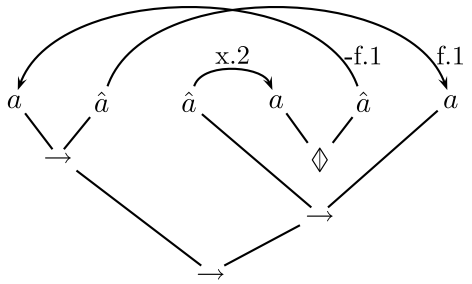
A path candidate with well-formed label in a prenet $N = L \rhd t \,|\, C$ is called *path in* N. We write $\mathrm{path}(N, C')$ for the set of paths in N through $C' \subseteq C$, and abbreviate $\mathrm{path}(N) = \bigcup_{C' \subseteq C} \mathrm{path}(N, C')$ for the set of all paths in N (through any set of cuts).

The general idea of the restriction to well-formed labels is to keep different occurrences of a variable apart. We will discuss this in again when we define a cut elimination procedure for our prenets.

Example 4.3.12.

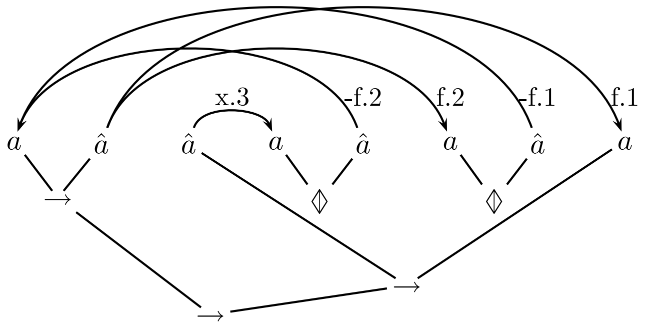
(1) $\mathrm{path}(N, \emptyset)$ consists exactly of the links of N.

(2) There are two paths through the cut of the prenet



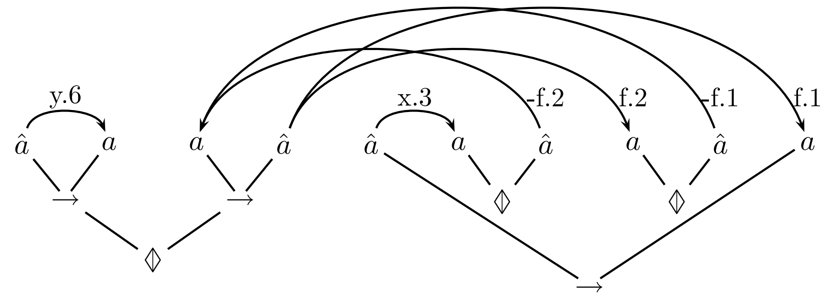
for $\lambda fx.fx$, which we have already seen in example 4.2.5. They bear the labels $(f.1)$ and $(x.2, -f.1)$, respectively.

(3) Let us have a glance at how the well-formedness condition influences the paths in a prenet. Consider the prenet of $\lambda f^5.\lambda x^4.f^1(f^2\, x^3)$, where again upper indices mark the variable occurrences:



There are two paths in this prenet that are not links: one labeled by $(x.3, -f.2)$, and one labeled by $(f.2, -f.1)$. The first label means, that the variable occurrence x^3 is used as an input to f^2, and the second means, that the output of f^2 is used as an input to f^1.

If we apply the above term to $\lambda y^7.y^6$, the resulting prenet is



It contains only one well-formed path candidate through all three cuts, marked by the label $(x.3, -f.2, y.6, f.2, -f.1, y.6, f.1)$. Pay attention to how the path represents the flow of the variable x^3 through the term: x^3 is the input to f^2, hence the component $(x.3, -f.2)$. The variable f^2 is (after a β-reduction) instantiated by $\lambda y^7.y^6$, which is responsible for the first instance of $y.6$. The output of f^2 is used as input of f^1, which is where the part $(f.2, -f.1)$ comes from. The instantiation of f^1 is again responsible for the component $y.6$. Finally the output of f^1 is the output of the whole term, hence the label $f.1$.

The well-definedness condition says here, that when we enter the tree coming from λf by a link labeled $-f.i$, we have to leave it again by a link with the label $+f.i$. If we had allowed any combinations of links to form a path, we would also have received the path $(x.3, -f.2, y.6, f.1)$. This path corresponds to x^3 being used as input to f^2 (nothing new so far), but then miraculously jumping to and leaving f^1. For sure, this is not what we expect x^3 to do, and we exclude this possibility in our paths.

Paths have the nice property that the labels of two paths separated by a cut can always be combined to a well-formed label:

Lemma 4.3.13. *Let $e = \lambda v_1.\ldots\lambda v_n.e_1\, e_2$ with $n \geq 0$ be a closed term. Let $c \in C$ be the cut introduced by the application $e_1\, e_2$, and Let $p_1 = (l_1, l'_1, \sigma_1)$ and $p_2 = (l_2, l'_2, \sigma_2)$ be two path candidates that do not use c.*

If $l'_1 \in \mathrm{leaves}(c.1)$ and $l_2 \in \mathrm{leaves}(c.2)$, or vice versa, then the combined label $\sigma_1\sigma_2$ is well-formed, if and only if σ_1 and σ_2 are well-formed.

Proof. We consider only the case that $l'_1 \in \mathrm{leaves}(c.1)$ and $l_2 \in \mathrm{leaves}(c.2)$. The case $l'_1 \in \mathrm{leaves}(c.2)$ and $l_2 \in \mathrm{leaves}(c.1)$ is analogous.

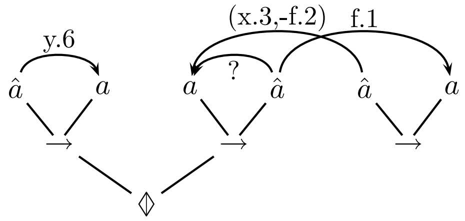
Let $e'_i = \lambda v_1.\ldots\lambda v_n.e_i$. For a visualization of the situation look again at table 4.3. Since p_1 and p_2 do not use c, p_1 is a path candidate in e'_1, and p_2 is a path candidate in e'_2. The only variables that both e'_i share are $v_1, \ldots, v_n$. This means, that reasons for a non-well-formedness of $\sigma_1\sigma_2$ can lie in a non-well-formedness of one of the σ_i, or in components of the form $(\pm, v_j, k)$ that occur in the σ_i.

Let $t_1, \ldots, t_n$ be the trees coming from $\lambda v_1, \ldots, \lambda v_n$. If no t_j contains both l'_1 and l_2 then σ_1 and σ_2 contain no components involving the same variables. So the well-formedness of σ_1 and σ_2 carries over to $\sigma_1\sigma_2$ and vice versa.

Otherwise $\sigma_1 = (+, v_j, k_1)\sigma_1'$ and $\sigma_2 = \sigma_2'(-, v_j, k_2)$, where σ_1' and σ_2' do not involve any common variable nor v_j. Again $\sigma_1\sigma_2$ is well-formed, if and only if σ_1 and σ_2 are well-formed. □

Although the notion of paths looks quite innocent, even small changes in the labels of a prenet may result in enormous changes in the number of paths.

Example 4.3.14. Consider the following prenet template:



If the label at "?" is $(f.2, -f.1)$ (which means, as we will see, that the prenet emerged from the one in example 4.3.12 by elimination of the two cuts on the right), then the path candidate through the cut that does not use the "?"-link is labeled $(x.3, -f.2, y.6, f.1)$. It violates the second well-formedness condition. The only path through the cut uses the "?"-link once and passes the cut three times. It bears the label $(x.3, -f.2, y.6, f.2, -f.1, y.6, f.1)$.

However, if the label at "?" is just $-f.1$, then there is one path for each number of uses of the "?"-link. These paths bear the following labels:

$$
\begin{aligned}
&(x.3, -f.2, y.6, -f.1, y.6, f.1)\\
&(x.3, -f.2, y.6, -f.1, y.6, -f.1, y.6, f.1)\\
&(x.3, -f.2, y.6, -f.1, y.6, -f.1, y.6, -f.1, y.6, f.1)\\
&(x.3, -f.2, y.6, -f.1, y.6, -f.1, y.6, -f.1, y.6, -f.1, y.6, f.1)\\
&\quad\vdots
\end{aligned}
$$

The existence of infinitely many paths is a situation we will want to avoid. Especially when we turn to cut elimination, infinitely many paths will mean that cut elimination turns a prenet with a finite set of links into one with infinitely many links. As we have just seen in the above example, our restriction on the allowed path labels cannot circumvent this problem in all settings. However, we can show that the restriction is sufficient when we work with prenets of λ-terms.

To prove this, we need the concept of loops. When we construct prenets out of terms, a small prenet is often used as one constituent of a larger prenet. A loop is a path that, when seen in the context of a larger prenet, may give rise to infinitely many paths in this prenet.

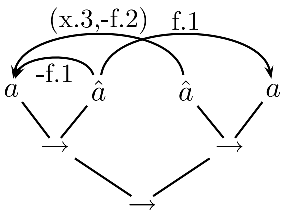
Definition 4.3.15. Let $N = L \rhd t\,|\,C$ be a polarized prenet. A path (l_1, l_2, σ) in N is a *loop*, if the following conditions hold:

- The path connects two dually polarized leaves.
- The combined link label $\sigma\sigma$ (and hence also $\sigma\sigma\sigma$, $\sigma\sigma\sigma\sigma$ and so forth) is well-formed.
- There is a subtree t' of t with a $\bullet$ root (or a cut $c \in C$), such that t' (or $c.2$, respectively) contains both leaves l_1 and l_2.

We speak of a *loop over* t or a *loop over* c.

Example 4.3.16.

- The prenet

  has only one polarization, and it contains one loop labeled $-f.1$. Nevertheless, there are only finitely many (namely 3) paths through this prenet. The last prenet in example 4.3.12 also has exactly one polarization, and it contains the same loop, but now this loop is a loop over a cut and produces an infinite number of paths.

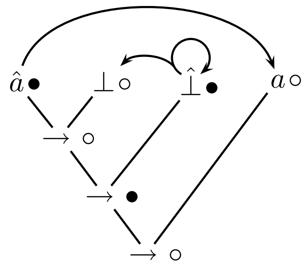
- The typed λ-calculus can be extended to a proof system for full classical logic by including a way to eliminate double negations by deducing A from $\overline{\overline{A}}$, i.e. a construct

  $$\Lambda := \ldots \mid \operatorname{dneg} \Lambda$$

  with the typing rule

  $$\frac{e : (A \to \bot) \to \bot}{\operatorname{dneg} e : A}\ \text{dneg}$$

  Translating the constant dneg into the language of prenets yields for example the following uniquely polarizable prenet:

It is classically correct, but it contains a loop: the link from the $\hat{\bot}$-labeled node to itself.

All the other prenets presented so far are loop free. In fact, this is not a coincidence but common to all prenets associated to λ-terms.

Lemma 4.3.17. *Let $v_1, \ldots, v_n \in \mathcal{V}$ be variables, and $1 \leq i \leq n$. Then the prenet $\mathbf{N}(\lambda v_1. \ldots \lambda v_n.v_i)$ does not contain a loop. The same holds for the prenets of the constants.*

Proof. Let $\mathbf{N}(\lambda v_1. \ldots \lambda v_n.v_i) = L \rhd t \,|\, \emptyset$, and let t' be an arbitrary subtree of t with a $\bullet$ root. Then t' is a (not necessarily strict) subtree of one of the trees coming from a binder λv_j, or of the tree coming from v_i. In both cases there are no links (and, as the prenet is cut-free, no paths) between two leaves of t'. Hence there is no loop over t.

This argument is simiar for all constants except null. But for $\mathbf{N}(\text{null})$, the situation is even simpler: There is only one link, and this link is obviously not a loop. ☐

Proposition 4.3.18. *Let e be a typed λ-term. Then $\mathbf{N}(e)$ does not contain a loop.*

Proof. We apply the same proof technique as in the correctness proof 4.3.9, showing the proposition by induction on the number of cuts in $\mathbf{N}(e)$.

Let $\mathbf{N}(e) = L \rhd t \,|\, C$.

- If e is a constant, then the proposition follows from lemma 4.3.17.
- Let $\mathbf{N}(e)$ be cut-free but not a constant. Then e is of the form $\lambda v_1. \ldots \lambda v_n.v$. If $v = v_i$ with $1 \leq i \leq n$, then the proposition follows again from lemma 4.3.17.

  If $v \neq v_i$ for all $1 \leq i \leq n$, then $\mathbf{N}(e)$ does not contain any links at all, so the proposition is trivially true.
- Let $\mathbf{N}(e)$ be a formula containing at least one cut.

  Then e is of the form $\lambda v_1. \ldots \lambda v_n.e_1\, e_2$, with $n \geq 0$. Let $c \in C$ be the cut introduced by the application $e_1\, e_2$. The terms $e'_i = \lambda v_1. \ldots \lambda v_n.e_i$ contain strictly less applications than e. So the $\mathbf{N}(e'_i)$ do not contain loops.

  Assume that there is a loop in $\mathbf{N}(e)$. By induction hypothesis this loop has to contain parts of both subnets, $\mathbf{N}(e_1)$ and $\mathbf{N}(e_2)$. Thus it has to use the cut c, i.e. without loss of generality, it is a loop over c.

  This loop's label must then be of the form

  $$l_1^{(1)} l_1^{(2)} l_2^{(1)} l_2^{(2)} \ldots l_n^{(1)} \;,$$

with $l_1^{(1)}, \ldots, l_n^{(1)}$ being labels of paths in e_1, with $l_1^{(2)}, \ldots, l_n^{(2)}$ being labels of paths in e_2, and with $l_1^{(1)}$ and $l_n^{(1)}$ polarized dually.

Because of the independence of paths on both sides of c, that we know from lemma 4.3.13, the label $l_1^{(1)} l_2^{(1)} \ldots l_n^{(1)}$ is also well-formed, and thus comes from a loop in e_1. But this is a contradiction to the assumption that e_1 does not contain loops.

Hence there are no loops in $\mathbf{N}(e)$. □

With this information, we can directly conclude:

Corollary 4.3.19. *Let e be a typed lambda term. Then the number of paths in $\mathbf{N}(e)$ is finite.*

So we know that the prenets in examples 4.3.16 are not the prenets of any λ-term.

4.3.4 Ramification

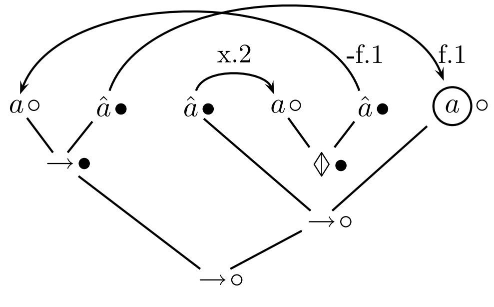
An outstanding difference between sequent systems for classical and intuitionistic logic is the handling of contractions. In a classical sequent proof, contraction may be used on any formula of a sequent. As intuitionistic sequents have only one conclusion (i.e. they are of the form $\Gamma \vdash A$), contraction on the right hand side of a sequent is not possible. As a polarization provides a means to represent the difference between left and right, we can also translate the intuitionistic restriction of contractions into the language of prenets.

Definition 4.3.20. Let N be a polarized prenet. A leaf l of N is called an *output leaf*, if l and all ancestors of l are $\circ$ nodes.

A leaf k of N is called an *input leaf*, if k ia a $\bullet$ node, and if there are natural numbers $m, n \geq 0$ such that the first m ancestors of k are $\bullet$ nodes, while the remaining n ancestors of k are $\circ$ nodes.

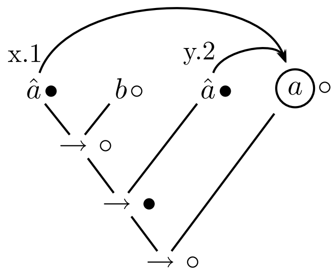
A polarized prenet $N = L \rhd t \,|\, C$ is *unramified*, if there is at most one path in $\text{path}(N, C)$ to every output leaf in N.

Example 4.3.21. The prenet from example 4.1.17 has only the polarization seen in example 4.3.2, and exactly one output leaf (marked by a circle):



Obviously, there is exactly one path ending in this node. All • leaves of this prenet are input leaves.

The following prenet is polarizable and classically correct, however its unique output leaf (the rightmost node) has two incoming paths:



The underlying sequent is Pierce's law $\vdash ((a \to b) \to a) \to a$, which has no intuitionistic proof.

Proposition 4.3.22. *Let A be a type. If $t = \mathbf{T}(A, i)$ polarized by the rules of 4.3.1, such that its root is a $\circ$ node, then there is at least one output leaf.*

If A contains no conjunctions and disjunctions, then the output leaf is unique.

Proof. This follows directly from the polarization rules 4.3.1. $\square$

Lemma 4.3.23. *Let e be a term. If l is an output leaf of $\mathbf{N}(e)$, and if $(k, l, \sigma) \in \mathrm{path}(\mathbf{N}(e))$ is a path ending in l, then k is an input leaf.*

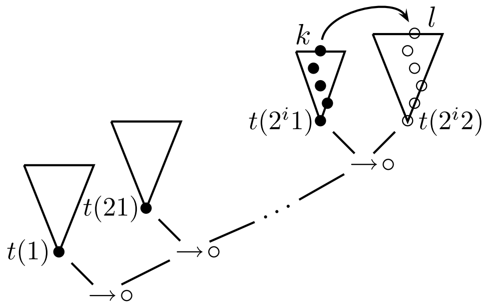
Proof. We again proceed by induction on the number of applications in e.

Let $\mathbf{N}(e) = L \rhd t \,|\, C$, and let l be an output leaf of $\mathbf{N}(e)$. All leaves in $\mathbf{N}(e)$ were introduced either by a variable or constant occurrence or by a variable binder. As lemma 4.3.4 and the unique propagation of polarities (cf. proposition 4.3.3) ensure that the roots of the trees coming from binders are always • nodes, l must come from a variable or a constant.

Assume that l comes from a variable v. The argumentation for the constants is analog, if we interpret them as abstractions as motivated in 4.2.10. If there is any link to l, then it comes from the corresponding node k in the tree t' of λv. As all ancestors of l are $\circ$ nodes, the by 4.3.5 dually markable ancestors of k in t' are all • nodes.

We distinguish two cases:

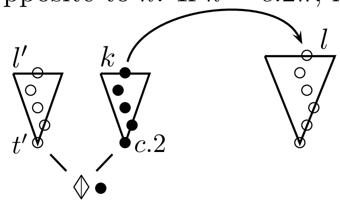
- If e does not contain a subterm of the form $(\lambda v.e_1)\, e_2$, the tree t' is not part of a cut of $\mathbf{N}(e)$, instead, t is of the following shape:



All nodes between $t(\varepsilon)$ and $t(2^i)$ are $\circ$ nodes, and all nodes between $t(2^i1) = t'(\varepsilon)$ and k are $\bullet$ nodes. So k is an input leaf, and k is the only node where a path to l starts.

- If e contains a subterm of the form $(\lambda v.e_1)\, e_2$, there is a cut $c \in C$ such that $c.2 = t'$. As $c(\varepsilon)$ is a $\bullet$ node, and as t' is polarized dually to the tree coming from v, all predecessors of k are $\bullet$ nodes. This means, that k is an input leaf.

  If there is no longer path leading to l, we are done. Otherwise the path enters c in the node opposite to k: If $k = c.2\pi$, it enters in $l' = c.1\pi$.

  Consider the term $e_2' = \lambda v_1. \dots \lambda v_n.e_2$, where e_2 is extended by all the binders of e that bind free variables of e_2. As $c.1$ is polarized dually to $c.2$, all nodes of the path from $c(1)$ to l' are $\circ$ nodes. Hence, l' is an output leaf of e_2'. By induction hypothesis, the paths in $\mathbf{N}(e_2')$ ending in l' begin in input leaves (which cannot be leaves of $c.1$, because $c(1)$ is a $\circ$ node but $c(\varepsilon)$ is not). Expanding these paths by the link from k to l yields the paths to l, so all their start points are input leaves. $\square$

Lemma 4.3.24. *Let e be a term that does not contain the constant case. Let l be an output leaf of $\mathbf{N}(e) = L \rhd t \,|\, C$ and let $(k, l, \sigma) \in \mathrm{path}(\mathbf{N}(e), C)$ be a path ending in l.*

If k and a $\circ$ leaf l' have a common ancestor that is a $\bullet$ node, then there is at most one path $(k', l', \sigma') \in \mathrm{path}(\mathbf{N}(e), C)$ such that $\sigma'\sigma$ is well-formed.

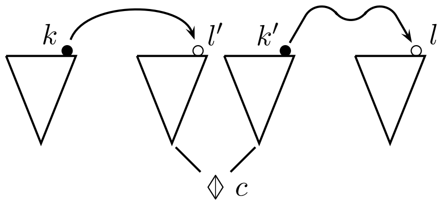
Proof. We show the proposition by induction on the number of applications in e.

- Let e be application-free. If e is of the form $\lambda v_1. \dots \lambda v_n.v_i$ with $1 \leq i \leq n$, then every leaf is the starting point of at most one path. Thus the statement is true. The same holds for all constants except case. If e is of the form $\lambda v_1. \dots \lambda v_n.v$ with $v \neq v_i$ for all $1 \leq i \leq n$, then $\mathbf{N}(e)$ does not contain any leaves at all, so the statement is also true.

- Let e be a formula containing at least one application. Then e is of the form $\lambda v_1. \dots \lambda v_n.e_1\, e_2$, with $n \geq 0$. Let $c \in C$ be the cut introduced by the application $e_1\, e_2$. The terms $e'_i = \lambda v_1. \dots \lambda v_n.e_i$ contain strictly less applications than e, so we may use the induction hypothesis for both terms.

  Let $(k, l, \sigma) \in \text{path}(\mathbf{N}(e), C)$. If the path lies completely in $\mathbf{N}(e'_1)$, the lemma holds by induction hypothesis for e'_1.

  Otherwise, the path uses the cut c. Then we can split it into two parts (k, l', σ_1) and (k', l, σ_2), such that (k, l', σ_1) does not use c.



  By lemma 4.3.23, k' is an input node, and since $c.1$ and $c.2$ are dually polarized, l' is an output leaf of $\mathbf{N}(e'_2)$.

  The induction hypothesis for (k, l', σ_1) and e'_2 proves the claim. □

Proposition 4.3.25. *Let e be a $\lambda^{\to\wedge}$-term. Then $\mathbf{N}(e)$ is unramified.*

Proof. To simplify the notation in this proof, we will say that a path p is a *full path* in a prenet N , if it is a path through all cuts of N.

Let $\mathbf{N}(e) = L \rhd t \,|\, C$ be equipped with a polarization. We proceed by induction on the number of applications in e.

- Let e be application-free. Then e is a constant, or e is of the form $\lambda v_1. \dots \lambda v_n.v$ with $n \geq 0$ (and not necessarily $v = v_i$ for any i). There is at most one link ending in any leaf $\mathbf{N}(e)$, and as the prenet does not contain cuts, all paths are links.

- Let e be a formula containing at least one application. Then e is of the form $\lambda v_1. \dots \lambda v_n.e_1\, e_2$, with $n \geq 0$. Let $c \in C$ be the cut introduced by the application $e_1\, e_2$. The terms $e'_i = \lambda v_1. \dots \lambda v_n.e_i$ contain strictly less applications than e. So $\mathbf{N}(e'_1)$ and $\mathbf{N}(e'_2)$ are unramified.

  Let l be any output leaf of $\mathbf{N}(e)$. We will now incrementally construct the possible ending of a path to l. To do this, we divide each such possible ending into its parts inside e'_1 and e'_2, respectively. This is not hard, but it means that we have to jump to and fro in $\mathbf{N}(e)$.

  As an output leaf of $\mathbf{N}(e)$, l is also an output leaf of $\mathbf{N}(e'_1)$. By induction hypothesis for e'_1, there is at most one full path (k, l, σ) in

$\mathbf{N}(e'_1)$. If such a path does not exist, there can also be no full path to l in $\mathbf{N}(e)$. If such a path exists and k is not a leaf of c, then (k, l, σ) is also the only full path to l in $\mathbf{N}(e)$.

So assume that $k = c(2\pi)$ is a leaf of (the right subtree of) c. Because k is an input leaf by lemma 4.3.23 for e'_1, and because $c.1$ and $c.2$ are dually polarized, $l' := c(1\pi)$ is an output leaf of $\mathbf{N}(e'_2)$. By induction hypothesis for e'_2, there is at most one full path (k', l', σ') in $\mathbf{N}(e'_2)$. As before, the only interesting case is $k' = c(1\rho)$.

Lemma 4.3.24 for e'_1 ensures that there is at most one full path in e'_1 that can be prefixed to (k, l, σ). Because paths in e'_1 and e'_2 are independent by lemma 4.3.13, the same holds also for the combined path $(k', l, \sigma'\sigma)$. Again, the only interesting case is that the prefix starts in a leaf of c.

If this is the case, the same argument as before yields the existence of at most one prefix in e'_2. Iteration of this process for e'_1 and e'_2 in turn always yields at most one new prefix. The iteration terminates, because leaves(c) is finite and $\mathbf{N}(e)$ contains no loops by proposition 4.3.18. □

4.3.5 Sequentialization

We will now justify the terminology "sequentializeble": If we have a sequentializable prenet, there is a way to retrieve a term, i.e. a natural deduction or sequent proof, that is translated into this prenet:

Theorem 4.3.26. *It is decidable, whether a prenet is the image under* $\mathbf{N}$ *of a closed* λ*-term.*

Let $N = L \rhd t \,|\, C$ *be the prenet of a closed* λ*-term* e*, and let* n_{nodes} *be the number of nodes in* N*. If* e *contains at most one instance of the constant null, then a term* e' *with* $N = \mathbf{N}(e')$ *can be computed in time* $O(n_{nodes})$*.*

Proof. Decidability is obvious, because we can enumerate all λ-terms whose corresponding prenets contain n_{nodes} nodes, i.e. all λ-terms whose variable occurrences have types containing all in all n_{nodes} symbols. The really interesting statement is that the reconstruction of e' can be done in linear time.

We first describe an algorithm computing e' from N in the case of a simply typed term e. This allows us to concentrate on the essential ideas. Then this algorithm is extended to the full calculus, and we finally analyze the complexity.

Analyzing prenets of closed $\lambda^{\to}$*-terms:* In a preprocessing step, we index all links by their label, and we mark all trees coming from a binder.

Step 1: Find the output leaf l of t. This leaf exists and is unique because of proposition 4.3.22. As e is closed, and l comes from a variable, there is exactly one link touching l. This link is unique, because N is unramified. Let it be $(l', l, (+, v, i))$.

Let $v_1, \ldots, v_k$ be the different variables occurring in other links that use leaves of t (sorted, starting from the leftmost leaf). These correspond to the binders whose trees are subtrees of t.

Step 2: We know that l' comes from the binder λv_0. Follow, starting with the leftmost sibling of l', all other links labeled $(\pm, v, i)$.

Step 3: If such a link leads to the right side of a cut, this cut must come from an application of v^i to another term.

So let $c_1, \ldots, c_m$ be these cuts, i.e. $c_1.2, \ldots, c_m.2$ are reached by the links. Restart the algorithm with $c_1.1, \ldots, c_m.1$ instead of t, which produces terms $e_1, \ldots, e_m$.

Step 4: Collecting all the information we have so far, the net associated to the term $e := \lambda v_1. \ldots \lambda v_k.v_0\, e_1\, \ldots\, e_m$ is contained in N. Additionally, it contains all nodes visited so far in any instance of step 1, but not necessarily all from an instance of step 2.

If all binders encountered so far have been used for a term, return e. If otherwise all possible instances of a variable v' are contained in e, look at the tree coming from the binder $\lambda v'$. As this is not a subtree of t (those have been handled in step 1), it must be the right direct subtree of a cut c'. Restart the algorithm with c' instead of t, resulting in a term e'. Replace e by $(\lambda v'.e)\, e'$, and repeat this step.

It is obvious that every leaf of N is contained in $\mathbf{N}(e)$, that the application structure is done in such a way that the cuts of $\mathbf{N}(e)$ correspond to those of N, and that the linkings of both prenets are identical.

Extension to $\lambda^{\to\wedge}$-terms: If e may contain conjunctions, then t may have multiple output leaves. We can proceed with the algorithm by following the link to an arbitrary output node. Here again, we use the fact that the constants pair, π_1 and π_2 can be regarded as abstractions. The difference between these constants can easily be read off from the shape of the tree that is linked to l. The rest of the algorithm does not change at all, except that v may not only be a variable, but also one of the conjunctive constants.

Including the other λ-terms: Including the constants case, inl and inr is completely analogous to the inclusion of pair, π_1 and π_2, except that the concrete constant can directly be read off the shape of t, and for case have to follow two links. A special case occurs, if the constant null comes into play. Then all output leaves are not part of the linking of N. However, if there is only one occurrence of null in e, then there is a unique $\hat{\perp}$-labeled leaf in N, which is linked to itself. Then we can continue the algorithm as if l had been linked to this node.

Runtime: We analyze only the elementary algorithm. The extensions do not change the complexity. Let n_{nodes} be the number of nodes in N, and let n_{links} be the number of links.

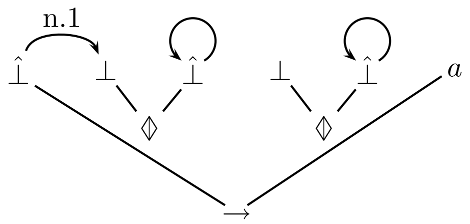
The preprocessing can obviously be done in time $O(_{\text{links}}+n_{\text{links}})$.

It is easy to model the necessary data structures in such a way, that finding the links needed in steps 2 and 3 (and the $\hat{\perp}$-leaf) takes constant time per link. Each link of N is visited at most twice, always directly between the visits to the nodes it connects, and all links coming from the same variable occurrence are visited at the same time. So all instances of link operations together take $O(n_{\text{links}})$. Walking up the trees in all runs of step 1 takes $O(n_{\text{nodes}})$, and walking down to the root in step 3 can be done in constant time per walk, i.e. $O(_{\text{links}})$ altogether. If we count the number of unprocessed occurrences of the variables incrementally in step 1, which can be done in constant time per update, each instance of step 4 also takes constant time.

All in all, the algorithm for $\lambda^{\rightarrow}$ has a runtime of $O(_{\text{links}}+n_{\text{links}})$. As each link touches at least on leaf that is touched by no other link, $_{\text{links}} < n_{\text{links}}$, i.e. the runtime is $O(n_{\text{links}})$. $\square$

Remark 4.3.27.

- The problem with the constant null comes from its fragmentation of the prenet. To get an idea, consider, for a variable n of type $\perp$, the prenet of the term $\lambda n.\text{null}\,(\text{null}\,n)$ of type a:



  After finding the (unique) output node, we know that we have to continue at one of the $\hat{\perp}$-nodes with a link to itself. However, we can only nondeterministically guess which one to take. This means, that an extension of the reconstruction algorithm to arbitrary terms has the complexity $O(n \cdot n_{\text{null}}!)$, where n_{null} is the number of occurrences of the constant null in e.

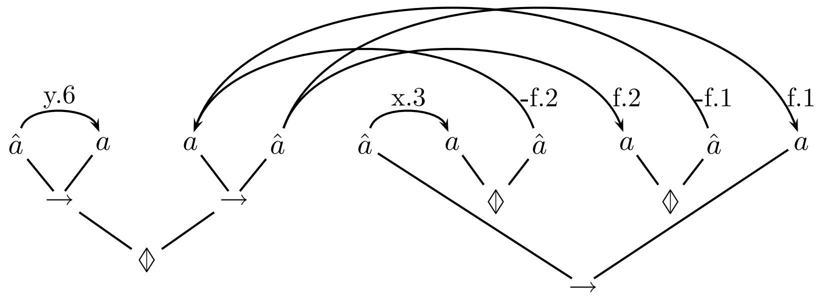
- The term found by the algorithm is in general not the only term that is translated to the given prenet N. E.g. the two terms

  $$(\lambda f^5.\lambda x^4.f^1(f^2\,x^3))\,(\lambda y^7.y^6)$$

  and

  $$\lambda x^4.(\lambda f^5.f^1(f^2\,x^3)\,(\lambda y^7.y^6))$$

  both translate to the prenet



from example 4.3.12. Applying the algorithm to this prenet yields the first term.

4.4 Cut Elimination

Whenever a new proof system is introduced, one is interested in some major properties. And when a system contains a cut rule, probably the most important question is whether this rule is admissible, i.e. whether there is a way to transform a given proof into one that does not contain any cuts. This idea was introduced by Gentzen [Gen35]. When he discovered natural deduction and the sequent calculus for intuitionistic and classical logic, the highlight was his *Hauptsatz*, that proved admissibility of the cut rule in his sequent calculi. The potential complexity of this question can be seen in the fact that it took about 30 years until Prawitz [Pra65] succeeded in proving the same theorem for natural deduction.

In the realm of proof nets, cut elimination is usually much easier to cope with. Whether we look at proof nets for multiplicative linear logic, or for extensions, or at classical proof nets — in every case, cut admissibility was proved along with the introduction of the system.

We will continue this tradition by giving a cut elimination procedure for intuitionistic prenets. This cut elimination procedure takes the general idea of chasing links through cuts from cut elimination in the context of classical proof nets. However, we use the label attached to each link to choose "good" link combinations. Proceeding this way, in which the good link combinations bear a meaning, we get a somehow canonical definition of cut elimination and avoid ad hoc decisions. At the same time, this cut elimination has many desirable properties. Above all, we will show that it is terminating and confluent.

Knowing this, we can also finally distinguish real intuitionistic proof nets, i.e. a sufficiently large class of prenets that can be regarded as intuitionistic proofs.

As it will (on the long run) be interesting to see how proof nets of different calculi can be seen as parts of one general theory, we also show how to define another cut elimination that produces the same normal forms but is even closer to classical cut elimination, because it first simplifies cuts between complex trees and then only has to care about cuts between atoms.

Our final goal is to compare cut elimination and reduction procedures for λ-terms. Although it would be nice to have as close as possible a correspondence between both, we will show that anything near a bisimulation is impossible. However, we will characterize many cases, in which term reduction does not or only slightly change the normal form of the corresponding prenet.

4.4.1 Intuitionistic Cut Elimination

As intuitionistic logic is naturally a fragment of classical logic, and as intuitionistic prenets look almost like their classical counterparts, a first approach could be to regard intuitionistic prenets as classical prenets and copy the classical cut elimination into our framework. However, this method has a huge drawback: cut elimination for classical prenets is not confluent, and it is yet unknown, whether a restriction to intuitionistic prenets solves this problem.

However, we will maintain a basic idea of cut elimination in classical prenets: Eliminiating a cut means following the paths through this cut.

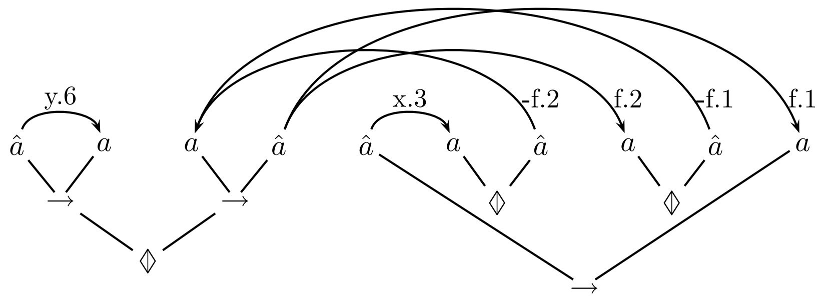
Definition 4.4.1. Let $N = L \rhd t \,|\, C$ be a prenet and $c \in C$. The result of the *cut elimination* of c in N is the prenet

$$\mathrm{CE}(N, c) := \mathrm{path}(N, \{c\}) \rhd t \,|\, C \setminus \{c\} \ .$$

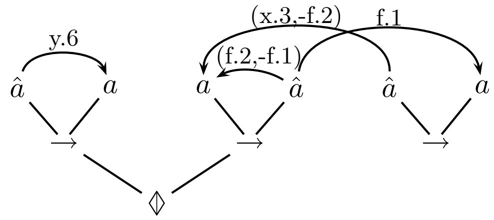
At first glance, this is exactly the way cut elimination was defined for classical propositional logic. Note however, that our notion of the paths through a cut imposes a restriction on the labels of the links that form these paths, thus eliminating some potential paths we would allow when doing cut elimination in classical logic. On the other hand, we relax the somehow unnatural condition to rule out all links crossing a cut, which means that we may allow more paths.

Example 4.4.2.

- Let us have a look at the prenet of $(\lambda fx.f(fx))(\lambda y.y)$:

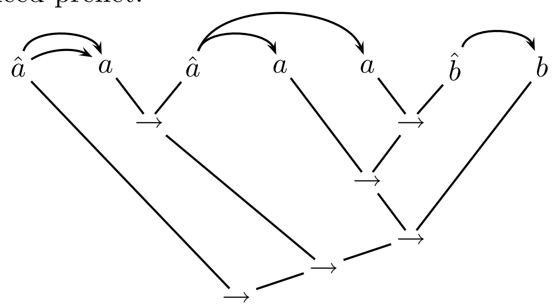


Reduction of the two rightmost cuts yields the following prenet, which we already know from example 4.3.14:

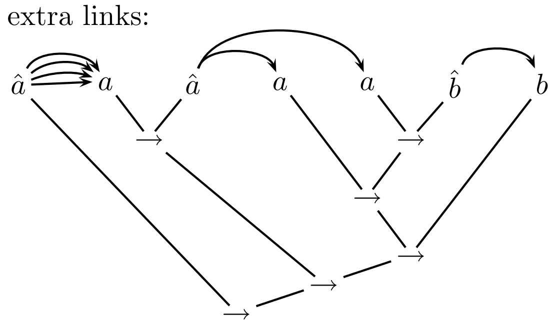


When we do cut elimination as if this were a classical prenet, we get a net with one link that would be labeled $(x.3, -f.2, y.6, f.1)$ (and thus not be well-formed). With the new cut reduction procedure, we get a link with a different label, namely $(x.3, -f.2, y.6, f.2, -f.1, y.6, f.1)$. This means, that we took a different path through the cut. Remark again how this path reflects the input/output flow in the term: x is the input to one instance of f, which is instantiated by the identity, produces an output that is used as the input to the other instance of f, which is again instantiated by the identity function and produces the output of the term.

- In general, classical cut elimination for these prenets does not follow this flow, but it merges disjoint parts of terms. Consider for example the term $\lambda x.(\lambda f.\lambda h.\lambda g.g\,(h\,(f\,x))\,(h\,(f\,x)))id$, where $h : a \to a$ and $g : a \to a \to b$. This is (up to link labels) the corresponding intuitionistically reduced prenet:



If we look at the classically reduced prenet of the term, we see that it contains two extra links:



These come from the fact, that the two instances of f are not kept apart by the classical reduction, and so both instances of x are linked to both instances of h.

Above all, this example shows, that intuitionistic cut elimination is not just a different way to model classical cut reduction restricted to intuitionistic prenets.

Let us now throw a glance at some important properties that we want cut elimination to enjoy. First of all, a normal form should be reached after finitely many steps.

Proposition 4.4.3. *Cut elimination for intuitionistic prenets is terminating.*

Proof. As every step of cut elimination reduces the number of cuts by one, the cut elimination of a prenet $L \triangleright t \,|\, C$ terminates after exatly $|C|$ steps. ☐

When we define cut elimination as above, we even get confluence for free:

Proposition 4.4.4. *Cut elimination for intuitionistic prenets is confluent.*

Proof. Let N be a prenet containing two cuts c_1, c_2, and let $N_i = L_i \triangleright t \,|\, C_i$ be the result of eliminating the cut c_i. Then

$$\begin{aligned}
\mathrm{CE}(N_1, c_2) &= \mathrm{path}(N_1, \{c_2\}) \triangleright t \,|\, C_1 \setminus \{c_2\} \\
&= \mathrm{path}(N, \{c_1, c_2\}) \triangleright t \,|\, C \setminus \{c_1, c_2\} \\
&= \mathrm{path}(N_2, \{c_1\}) \triangleright t \,|\, C_2 \setminus \{c_1\} \\
&= \mathrm{CE}(N_2, c_1)
\end{aligned}$$

This proves that cut elimination is locally confluent. As we already know that it is also terminating, local confluence implies global confluence. ☐

Notation 4.4.5. If $N = L \triangleright t \,|\, C$ is a prenet, and $C' = \{c_1, \ldots, c_n\} \subset C$ is a set of cuts, then we write $\mathrm{CE}(N, C')$ for the successive cut elimination of the elements of C' (in any order), and we write $\mathrm{CE}(N) = \mathrm{CE}(N, C)$ for the prenet that results from the elimination of all cuts in N.

Putting these results together yields:

Theorem 4.4.6. *Each prenet has exactly one normal form with respect to cut elimination.*

Proof. This follows directly from the propositions 4.4.3 and 4.4.4. ☐

We can even easily describe the unique normal form of a prenet without doing every single reduction step:

Corollary 4.4.7. *Let $N = L \triangleright t \,|\, C$ be a prenet. The unique normal form of N is the prenet*

$$\mathrm{CE}(N) = \mathrm{path}(N, C) \triangleright t \,|\, \emptyset \ .$$

Remember that one idea behind prenets is that proofs can be represented in a compact way, and that cut elimination is supposed to further simplify this representation. So above all, cut elimination should not transform a finite prenet into an infinite one. The last example in 4.3.12 shows, that with one step of cut elimination, a finite prenet may indeed become infinite. But we already know that this does not happen when we restrict ourselves to the normalization of nets coming from λ-terms:

Proposition 4.4.8. *Let e be a typed lambda term. If any number of cuts in $\mathbf{N}(e)$ is eliminated, the resulting net contains only finitely many links.*

Proof. Let $\mathbf{N}(e) = L \triangleright t \,|\, C$, and let $N' = L' \triangleright t \,|\, C'$ be the result of any series of cut eliminations in $\mathbf{N}(e)$.

We know from proposition 4.3.19, that the number $|\mathrm{path}(N)|$ of paths in $\mathbf{N}(e)$ is finite. As the set of links of N', which equals $\mathrm{path}(N', \emptyset)$, is a subset, it must be finite as well:

$$|\mathrm{path}(N', \emptyset)| = |\mathrm{path}(\mathbf{N}(e), C \setminus C')| \leq |\mathrm{path}(\mathbf{N}(e))| < \infty \qquad \square$$

We can say even more about properties of prenets that are preserved by cut elimination:

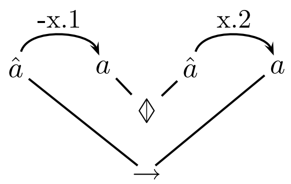
Theorem 4.4.9. *Cut elimination preserves polarizability, finiteness and non-ramification.*

Proof. Let e be a prenet, $\mathbf{N}(e) = L \triangleright t \,|\, C$, $c \in C$ and $N' = \mathrm{CE}(N, c) = L' \triangleright t \,|\, C \setminus \{c\}$.

If $\mathbf{N}(e)$ is polarizable, then all trees of $\{t\} \cup C$ are polarizable. In particular, all trees of $\{t\} \cup C \setminus \{c\}$ are polarizable. Have have to show, that the new links obey rule (1). Since every link of $\mathbf{N}(e)$ starts in a $\bullet$ node, so does every path of $\mathbf{N}(e)$. If a path of $\mathbf{N}(e)$ ends in a $\bullet$ node, then its last link ends in a $\hat{\bot}$-labeled node, so the same holds for the whole path.

If $\mathbf{N}(e)$ is finite, then the set $\mathrm{path}(N') \subseteq \mathrm{path}(N)$ is also finite (cf. proposition 4.4.8). The same argument shows, that N' is unramified, whenever N is unramified. $\square$

Remark 4.4.10. In general, a prenet may well reduce to a prenet that is not classically correct. For example, the prenet



reduces to the following, not classically correct prenet:

$$\begin{tikzcd}[column sep=small] a \arrow[dr, dash] & & \hat{a} \arrow[dl, dash] \\ & \to & \end{tikzcd}$$

4.4.2 Prenet Equivalence and Proof Nets

The link labels are mainly an additional algorithmic information that is supposed to control the behavior of a prenet under cut elimination. Hence two nets are equivalent, if they have the same shape, and if the prenets "behave the same way" with respect to cut elimination:

Definition 4.4.11. Given two intuitionistic prenets $N = L \rhd t \,|\, C$ and $N' = L' \rhd t' \,|\, C'$, we say that N is *stronger* than N', if there is a bijection $f\colon \{t\} \cup C \to \{t'\} \cup C'$ and a partial, surjective function $g\colon \mathrm{path}(N) \rightharpoonup \mathrm{path}(N')$, such that:

(1) The bijection f maps each tree to an equivalent tree:

$$\forall t_1 \in \{t\} \cup C : f(t_1) \equiv t_1$$

(2) g identifies only paths between corresponding leaves: If the canonical extension of f to the leaves in N and N' is denoted by

$$\hat{f}\colon \mathrm{leaves}(N) \to \mathrm{leaves}(N') ,$$

then

$$\forall (l_1, l_2, \sigma) \in \mathrm{dom}(g) : \exists \sigma' \in \mathcal{L} : g(l_1, l_2, \sigma) = (\hat{f}(l_1), \hat{f}(l_2), \sigma') .$$

(3) The labels of two paths $p_1, p_2 \in \mathrm{dom}(g)$ can be combined to a well-formed label, if and only if the same holds for the labels of $g(p_1)$ and $g(p_2)$.

If N is stronger than N', and N' is stronger than N, then N and N' are called *strongly equivalent*.

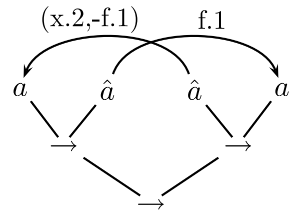
Remark 4.4.12.

- The intuition behind this comparison is, that the stronger a prenet is, the more information is contained in it, and that strongly equivalent prenets contain exactly the same information.
- The relation *stronger than* is a preorder on the set of prenets, i.e. it is reflexive and transitive.
- Because of the first condition, only prenets of the same type can be compared. The second condition guarantees, that there are at least as many path candidates in N as in N', and the third condition extends this guaranty to paths.

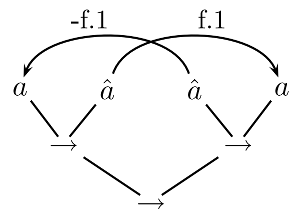
- Strong equivalence of nets trivially defines an equivalence relation, i.e. it is reflexive, symmetric and transitive.

Example 4.4.13.

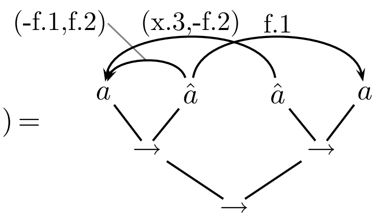
- If two terms e and e' are α-equivalent, and both terms bind no variable more than once (which is a precondition for their translation into prenets, cf. 4.2.2), then their proof nets $\mathbf{N}(e)$ and $\mathbf{N}(e')$ are strongly equivalent.
- The two prenets

  $\mathrm{CE}(\mathbf{N}(\lambda f.\lambda x.f\,x)) =$ 

  and

  $\mathbf{N}(\lambda f.f) =$ 

  are strongly equivalent, and both are neither weaker nor stronger than

  $\mathrm{CE}(\mathbf{N}(\lambda f.\lambda x.f(f\,x))) =$ 

The last example is just one instance of the general situation that, if a certain variable occurs only once in a term, i.e. if a label featuring this variable must have a fixed occurrence marker, then we can delete the label components containing the variable:

Lemma 4.4.14. *Let N be a prenet, let $v \in \mathcal{V}$ be a variable and i_0 a fixed integer. If each component of a link label of N involving v is of the form $(\pm, v, i_0)$, then replacing all these components by ε yields a prenet N' which is strongly equivalent to N.*

Proof. Let d: $\mathcal{L} \to \mathcal{L}$ be the function that deletes all label components containing v:

$$\begin{aligned} d(\varepsilon) &:= \varepsilon \\ d((\pm, v, i)) &:= \varepsilon && \text{for all } i \in \mathbb{N} \\ d((\pm, w, i)) &:= (\pm, w, i) && \text{for all } w \in \mathcal{V} \setminus \{v\}, i \in \mathbb{N} \\ d(\sigma\sigma') &:= d(\sigma)d(\sigma') && \text{for all } \sigma \neq \varepsilon \neq \sigma' \end{aligned}$$

Let g: $\mathrm{path}(N) \to \mathrm{path}(N')$ be the function deleting all occurrences of v:

$$g(l, l', \sigma) := (l, l', d(\sigma))$$

Then g is by construction a bijection between the paths in N and N', identifying only paths between corresponding leaves. As label components involving v do not occur in the form $(\pm, v, i)$ and $(\pm, v, j)$ for $i \neq j$, these components do not affect the well-formedness of any combination of path labels in N.

Hence g meets the demands of definition 4.4.14, and N and N' are strongly equivalent. □

We can further use the recursive definition of well-formedness to simplify the labels in a prenet:

Lemma 4.4.15. *Let N be a prenet, let v be a variable, and let $(l, l', \sigma_1\sigma_2\sigma_3)$ be a link in N, such that σ_2 does not contain components of the form $(\pm, v, i)$ for any integer i.*

Then replacing $(l, l', \sigma_1\sigma_2\sigma_3)$ by $(l, l', \sigma_1(-, v, i)\sigma_2(+, v, i)\sigma_3)$ produces a strongly equivalent prenet.

Proof. Let $N = L \rhd t \,|\, C$, and let $N' = L' \rhd t \,|\, C$ be the prenet produced by the replacement.

If f: $\{t\} \cup C \to \{t\} \cup C$ and g: $\mathrm{path}(N) \to \mathrm{path}(N')$ are both chosen to be the respective identity functions, then we only have to show that the labels of two paths in N can be combined to a well-formed label, if and only if the same holds for the corresponding paths in N'.

This is trivial, if none of the involved label components is the one we changed.

So assume, that the combination of the labels of two paths p_1 and p_2 in N, one of which uses the link $(l, l', \sigma_1\sigma_2\sigma_3)$, is well-formed. We write this combined label as $\sigma'\sigma_1\sigma_2\sigma_3\sigma''$. By definition 4.3.11, $\sigma'\sigma_1\sigma_2\sigma_3\sigma''$ is well-formed, if and only if $\sigma'\sigma_1(-, v, i)\sigma_2(+, v, i)\sigma_3\sigma''$, i.e. the combination of the corresponding path labels in N', is well-formed. □

Although we now have some notion of equivalence of equally shaped prenets, what we are really interested in is a description of those nets that reduce to the same normal form.

Definition 4.4.16. Two prenets N and N' are *equivalent*, if their normal forms $\mathrm{CE}(N)$ and $\mathrm{CE}(N')$ are strongly equivalent.

We can now distinguish a class of prenets that correspond — under this notion of equivalence — to an intuitionistic proof in natural deduction:

Definition 4.4.17. A prenet is called an *(intuitionistic) proof net*, if it is equivalent to $\mathbf{N}(e)$ for some λ-term e.

Example 4.4.18. All intuitionistic prenets that we have seen so far are proof nets, except the prenets of Pierce's law and double negation, and the loop-containing prenet from example 4.3.16. Even the prenets of the constants are proof nets. The proof net of $\lambda n.\mathrm{null}\, n$, for example, reduces to a prenet that is strongly equivalent to $\mathbf{N}(\mathrm{null})$.

We directly see, that proof nets are closed under cut elimination:

Theorem 4.4.19. *Cut elimination transforms proof nets into proof nets.*

The following shows, that this notion of equivalence interacts well with strong equivalence.

Proposition 4.4.20. *Let $N = L \rhd t \,|\, C$ be stronger than $N' = L' \rhd t' \,|\, C'$, and let the cut $c \in C$ corresponds to $c' \in C'$. Then $\mathrm{CE}(N, c)$ is stronger than $\mathrm{CE}(N', c')$.*

Proof. Without loss of generality, we assume $t = t'$ and $C = C'$. Then

$$\begin{aligned} \mathrm{CE}(N, c) &= \mathrm{path}(N, c) \rhd t \,|\, C \setminus \{c\} \quad \text{and} \\ \mathrm{CE}(N', c) &= \mathrm{path}(N', c) \rhd t \,|\, C \setminus \{c\} \ . \end{aligned}$$

We have to show that there is a partial surjection of $\mathrm{path}(N, c)$ onto $\mathrm{path}(N', c)$ that is subject to the requirements in definition 4.4.11.

Let $g\colon \mathrm{path}(N) \rightharpoonup \mathrm{path}(N')$ be the corresponding surjection for N and N'. Let $p' \in \mathrm{path}(N', c)$ be any path in $\mathrm{CE}(N', c)$. Since g is surjective, there is a preimage $p \in \mathrm{path}(N)$, such that $g(p) = p'$. Furthermore, condition (2) ensures $p \in \mathrm{path}(N, c)$.

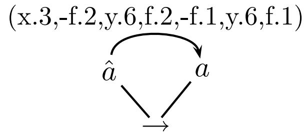
Hence the restriction

$$g|_{g^{-1}(\mathrm{path}(N',c))}\colon g^{-1}(\mathrm{path}(N', c)) \to \mathrm{path}(N', c)$$

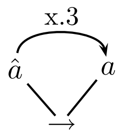
is the searched-for function. $\square$

Corollary 4.4.21. *Let $N = L \rhd t \,|\, C$ and $N' = L' \rhd t' \,|\, C'$ be two strongly equivalent prenets, and let the cut $c \in C$ correspond to $c' \in C'$. Then $\mathrm{CE}(N, c)$ and $\mathrm{CE}(N', c')$ are strongly equivalent.*

Example 4.4.22. The two prenets from example 4.4.2 are equivalent to their common normal form



and to the following normal prenet:



4.4.3 Atomic Cut Elimination

In all former proof net calculi, e.g. in the case of classical proof nets, cut elimination is done in two structurally different steps: Cuts between the trees of complex formulas are first split into cuts between subformulas, and only when a cut between atoms is reduced, real work is done. The presented cut elimination procedure for intuitionistic prenets works differently: No matter how large a cut is, it is eliminated in one step.

However, we can also express this cut elimination as the combination of cut simplifications and the reduction of cuts between atoms.

Definition 4.4.23. Let $N = L \rhd t \,|\, C$ be a prenet and $c = (t_1 \between t_2) \in C$. The result of the *atomic cut elimination* of c in N is the prenet

$$\mathrm{CE_a}(N, c) := \begin{cases} \mathrm{path}(N, \{c\}) \rhd t \,|\, C \setminus \{c\} & \text{if } \mathrm{dom}(t_1) = \{\varepsilon\} \\ L \rhd t \,|\, C \setminus \{c\} \cup \{t_1.1 \between t_2.1, t_1.2 \between t_2.2\} & \text{otherwise.} \end{cases}$$

Proposition 4.4.24. *Given a prenet N, the two notions of cut elimination reduce N to the same normal form.*

Proof. It suffices to show, that cut elimination via $\mathrm{CE_a}$ terminates, and that this reduction relation has the same effect on the paths as CE.

- We first prove the termination of $\mathrm{CE_a}$. Given a prenet $N = L \rhd t \,|\, C$, let $n(N)$ be the overall number of nodes belonging to a cut of N:

  $$n(N) := \sum_{c' \in C} |\mathrm{dom}(c')|$$

  Obviously $n(N) \geq 0$, and we show that $n(N)$ decreases by at least 1 during each step of atomic cut elimination.

  Let $c = (t_1 \between t_2) \in C$ be a cut of N. If c is a cut between two atoms, then

  $$\begin{aligned} n(\mathrm{CE_a}(N, c)) &= \sum_{c' \in C \setminus \{c\}} |\mathrm{dom}(c')| \\ &= \left(\sum_{c' \in C} |\mathrm{dom}(c')| \right) - |\mathrm{dom}(c)| = n(N) - 3 \;. \end{aligned}$$

Otherwise let $c_1 := t_1.1 \between t_2.1$ and $c_2 := t_1.2 \between t_2.2$. Then

$$\begin{aligned} n(\mathrm{CE_a}(N,c)) &= \sum_{c' \in C \setminus \{c\} \cup \{c_1,c_2\}} |\mathrm{dom}(c')| \\ &= \left(\sum_{c' \in C} |\mathrm{dom}(c')|\right) - |\mathrm{dom}(c)| + |\mathrm{dom}(c_1)| + |\mathrm{dom}(c_2)| \\ &= n(N) - 1 \ . \end{aligned}$$

- Let $N = L \rhd t \,|\, C$ be a prenet, $c \in C$ and $\mathrm{CE_a}(N,c) = L' \rhd t' \,|\, C'$. We show, that the paths in N through C are exactly the paths in $\mathrm{CE_a}(N,c)$ through C'.

  If c is a cut between two atoms, then $\mathrm{CE_a}(N,c) = \mathrm{CE}(N,c)$, so there is nothing to show.

  Otherwise let $c_1 := t_1.1 \between t_2.1$ and $c_2 := t_1.2 \between t_2.2$.

  Let $p \in \mathrm{path}(N,C)$, and let $l'_0, l_1, l'_1, \ldots, l_{n-1}, l'_{n-1}, l_n \in \mathrm{leaves}(C)$ be the series of links lying on the path, i.e. the series such that the requirements of the definition 4.3.10 are fulfilled.

  Then the same series of links also fulfills the requirements of a path candidate in $\mathrm{CE_a}(N,c)$ through C': (1), (2) and (4) are automatically fulfilled, because $\mathrm{leaves}(C') = \mathrm{leaves}(C)$ and $L = L'$, and (3) is fulfilled, because we can replace each occurrence of (c,π) in (3) by (c_1,ρ), if $\pi = 1.\rho$, and by (c_2,ρ), if $\pi = 2.\rho$.

  Since the well-formedness of the label of a path candiate does not depend on the surrouding prenet, p is also a path in $\mathrm{path}(\mathrm{CE_a}(N,c), C')$.

  This transfer is reversible, just by replacing each occurrence of (c_i,π) by $(c.i\pi)$. So $\mathrm{path}(N,C) = \mathrm{path}(\mathrm{CE_a}(N,c), C')$. □

Corollary 4.4.25. *When computing the normal form of a prenet with respect to* CE*, we may freely use both reductions with* CE *and* $\mathrm{CE_a}$*.*

4.4.4 Cut Elimination and Other Normalization Procedures

Cut elimination for multiplicative linear logic has an analog in the calculus of structures [Gug02], where cuts can be eliminated via a technique called *splitting*.

Lamarche and Straßburger [SL04] quickly found out, that their cut elimination procedure does not work parallel to cut elimination in the sequent calculus, and conjectured it also to be modeled by splitting in an adjusted version of the calculus of structures.

Of course, a strong connection to a different proof system has the advantage that many properties of the new system come for free. In general, there seem to be at least two ways to describe the relation between two normalization strategies. The strongest is (bi-)simulation. In our setting, this is the question, whether a step of $\beta\eta$-reduction corresponds exactly to one step of cut elimination:

$$\begin{tikzcd} e \arrow[r] \arrow[d, "\beta\eta"'] & \mathbf{N}(e) \arrow[d, "\text{cut elimination}"] \\ e' \arrow[r] & \mathbf{N}(e') \end{tikzcd}$$

If this is not possible, there is also the request that normal forms in both formalisms correspond. However, when we compare $\beta\eta$-reduction and cut elimination, we have to realize that both goals are not satisfiable:

Example 4.4.26. Let $f^n x$ be an abbreviation for $f(f^{n-1}x)$, if $n \geq 1$, and for x, if $n = 0$,.

The terms $e_n = \lambda f.\lambda x.f^n x$ are $\beta\eta$-normal, but their proof nets contain n cuts and hence are not normal for $n \geq 1$. So the second goal is not achievable, no matter what cut elimination procedure we choose.

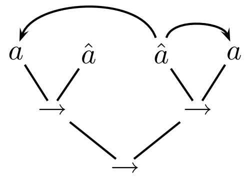
The same holds for the first goal: Cut elimination in all theories of prenets developed so far (including ours) reduces not more than one cut per step, so this seems to be a reasonable assumption. This assumption also means, that the number of steps to a normal form is independent of the reduction strategy. However, this does in general not hold in the λ-calculus. The term $(\lambda f.x)(e_n(\lambda x.x))$ for example takes any number of steps between 1 and $n+2$ to reach a normal form.

Another question is, whether proof nets and their cut elimination are monotone, in the sense that if e.g. $e \overset{\beta}{\rightsquigarrow} e'$, then the number of links in the normal form $\mathrm{CE}(\mathbf{N}(e))$ is always larger, or always smaller than the number of links in $\mathrm{CE}(\mathbf{N}(e'))$. However, this monotonicity does not hold:

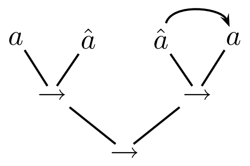
Example 4.4.27. We ignore link labels in this example.

The normal form of the proof net of $\lambda f.\lambda x.(\lambda y.x)\,(f\,x)$ contains one link more than the proof net of its β-normal form $\lambda f.\lambda x.x$:

$\mathrm{CE}(\mathbf{N}(\lambda f.\lambda x.\ (\lambda y.x)\,(f\,x)))$:

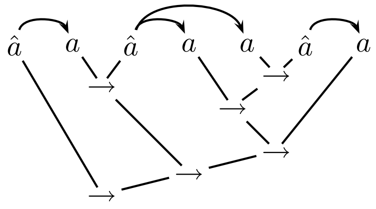


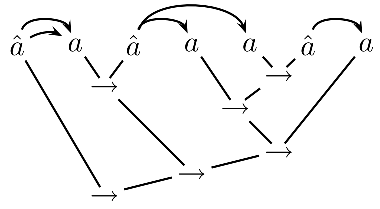
$\mathrm{CE}(\mathbf{N}(\lambda f.\lambda x.x))$:



For $g : a \to a \to a$, the term $\lambda x.\lambda f.\lambda g.(\lambda y.g\,y\,y)\,(f\,x)$ and its β-normal form $\lambda x.\lambda f.\lambda g.\lambda y.g\,(f\,x)\,(f\,x)$, the situation is the other way round:

$\mathrm{CE}(\mathbf{N}(\lambda x.\lambda f.\lambda g.\lambda y.g\,y\,y)\,(f\,x)))$: $\mathrm{CE}(\mathbf{N}(\lambda x.\lambda f.\lambda g.\lambda y.g\,(f\,x)\,(f\,x)))$:





This means, that we can only hope for some basic parallels between cut elimination and the reduction of λ-terms. But in fact, there are some. An almost trivial one is, that if $\mathbf{N}(e)$ is a normal proof net, then e is also normal:

Proposition 4.4.28. *Let e be a λ-term, such that $\mathbf{N}(e)$ is normal with respect to cut elimination. The e is $\beta\eta$-normal and does not contain a sum- or product-redex.*

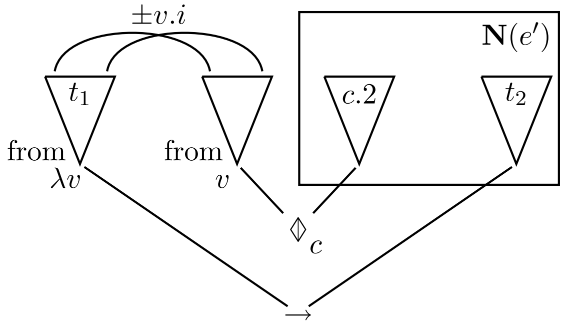
Proof. As $\mathbf{N}(e)$ does not contain any cuts, it is of the form $\lambda v_1.\ldots.\lambda v_n.v$ with $n \geq 0$. $\square$

Considering now each of the different reduction procedures by itself, we see several parallels to cut elimination. In particular, η-reduction can be simulated by cut elimination:

Proposition 4.4.29. *Let e and e' be any λ-terms, such that e reduces in one η-step to e'. Then the proof nets $\mathbf{N}(e)$ and $\mathbf{N}(e')$ are equivalent.*

Proof. For simplicity, we assume that the reduction happens at top level, i.e. $e = \lambda v.e_1\,v$ with $v \notin FV(e_1)$. The case of a reduction deep within e is completely analogous.

Then $\mathbf{N}(e) = L \rhd t \,|\, C$ has the following shape:



That v is not free in e_1 ensures that no link connects $\mathbf{N}(e_1)$ and the tree coming from λv.

Let $\mathbf{N}(e_1) = L_1 \rhd t_1 \,|\, C_1$. If $c \in C$ is the cut introduced by the application $e_1\,v$, and if L_v contains the links introduced by the abstraction $\lambda v.e_1\,v$. Then

$$\mathbf{N}(e_1\,v) = L_1 \rhd t_1.2 \,|\, C_1 \cup \{c\}$$

and

$$\mathbf{N}(\lambda v.e_1\, v) = L_1 \cup \lambda_v \rhd t_0 \to t_1.2 \,|\, C_1 \cup \{c\}$$

where t_0 is equivalent to $t_1.1$. Hence

$$\mathrm{CE}(\mathbf{N}(\lambda v.e_1\, v), c) = L' \rhd t_0 \to t_1.2 \,|\, C_1 \ ,$$

where

$$\begin{aligned} L' = &\{(t_0.\pi, l, (+, v, i)\sigma) \,|\, (c.2\pi, l, \sigma) \in L_1\} \\ &\cup \{(l, t_0.\pi, \sigma(-, v, i)) \,|\, (l, c.2\pi, \sigma) \in L_1\} \\ &\cup \big\{(l, l', \sigma) \in L_1 \,|\, l, l' \notin \mathrm{leaves}(c.2)\big\} \ . \end{aligned}$$

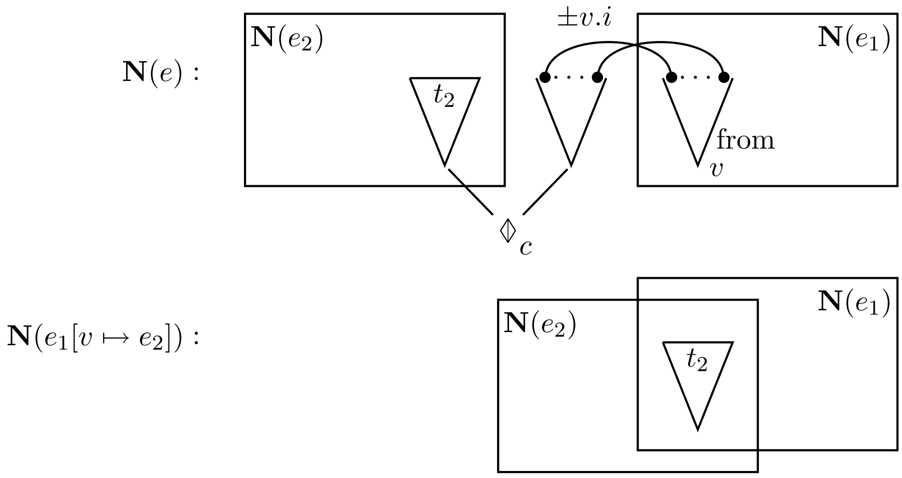
Using lemma 4.4.14 and the equivalence of t_2 and $t_0 \vee t_1.2$, we can directly conclude that the two prenets $\mathrm{CE}(\mathbf{N}(\lambda v.e_1\, v), c)$ and $\mathbf{N}(e_1)$ are strongly equivalent. □

The other situation that we can describe completely is β-reduction in the linear fragment of the calculus, i.e. the fragment where a variable binder λv binds exact one occurrence of v:

Proposition 4.4.30. *Let $e = (\lambda v.e_1)\, e_2$ be a λ-term, where v occurs exactly once freely in e_1.*

Then the proof nets $\mathbf{N}(e)$ and $\mathbf{N}(e_1[v \mapsto e_2])$ are equivalent.

Proof. The situation is as follows:



Let $\mathbf{N}(e) = L \rhd t \,|\, C$, let $\mathbf{N}(e_i) = L_i \rhd t_i \,|\, C_i$, and let c be the cut in $\mathbf{N}(e)$ introduced by the application. If L_v contains the links introduced by the abstraction $\lambda v.e_1$, and if $c \in C$ is the cut introduced by the application $(\lambda v.e_1)\, e_2$, then $\mathbf{N}(e_1[v \mapsto e_2])$ is of the form

$$\mathbf{N}(e_1[v \mapsto e_2]) = L_1 \cup L_2 \rhd t_1 \,|\, C_1 \cup C_2 \ .$$

On the other hand, we have

$$\mathbf{N}(\lambda v.e_1) = L_1 \cup L_v \rhd t_0 \to t_1 \,|\, C_1$$

(for some tree t_0) and

$$\mathbf{N}((\lambda v.e_1)\, e_2) = L_1 \cup L_v \cup L_2 \rhd t_1 \,|\, C_1 \cup C_2 \cup \{c\} \ .$$

This means, that $\mathrm{CE}(\mathbf{N}(e), c)$ is of the form

$$\mathrm{CE}(\mathbf{N}(e), c) = L_1 \cup L' \rhd t_1 \,|\, C_1 \cup C_2 \ ,$$

where

$$\begin{aligned} L' = &\{(l, t_0.\pi, \sigma(+, v, i)) \,|\, (l, c.1\pi, \sigma) \in L_2 \wedge l \notin \mathrm{leaves}(c.1)\} \\ &\cup \{(t_0.\pi, l, (-, v, i)\sigma) \,|\, (c.1\pi, l, \sigma) \in L_2 \wedge l \notin \mathrm{leaves}(c.1)\} \\ &\cup \big\{(t_0.\pi, t_0.\pi', (-, v, i)\sigma(+, v, i)) \,|\, (c.1\pi, c.1\pi', \sigma) \in L_2\big\} \\ &\cup \big\{(l, l', \sigma) \in L_2 \,|\, l, l' \notin \mathrm{leaves}(c.1)\big\} \ . \end{aligned}$$

Thus, deletion of the label components of the form $(\pm, v, i)$ yields exactly the proof net $\mathbf{N}(e_1[v \mapsto e_2])$. By lemma 4.4.14, both nets are strongly equivalent. □

The second situation where β-reduction corresponds exactly to cut eliminations is the one in which the argument term is closed:

Proposition 4.4.31. *Let $e = (\lambda v.e_1)\, e_2$ be a λ-term, where e_2 is a closed term.*

Then the proof nets $\mathbf{N}(e)$ and $\mathbf{N}(e_1[v \mapsto e_2])$ are equivalent.

Proof. Let c be the cut in $\mathbf{N}(e)$ introduced by the application. We show that the elimination of c and all cuts introduced by an occurrence of e_2 in $\mathbf{N}(e)$ and $\mathbf{N}(e_1[v \mapsto e_2])$ yields strongly equivalent proof nets. Table 4.4 shows the nets we will consider during this proof.

Let $\mathbf{N}(e) = L \rhd t \,|\, C$, let $\mathbf{N}(e_i) = L_i \rhd t_i \,|\, C_i$ be the subnet of $\mathbf{N}(e)$ introduced by e_i, and let L_v contain the links introduced by the abstraction $\lambda v.e_1$. Then $L = L_1 \cup L_v \cup L_2$.

Let $\mathrm{CE}(\mathbf{N}(e_2)) = L_2' \rhd t_2 \,|\, \emptyset$. If $v^1, \ldots, v^n$ are the free occurrences of v in e_1, then reducing $\mathbf{N}(e_2)$ in $\mathbf{N}(e)$ yields

$$\begin{aligned} \mathrm{CE}(\mathbf{N}(e), C_2) &= (L \setminus L_2) \cup L_2' \rhd t \,|\, C \setminus C_2 \\ &= L_1 \cup L_v \cup L_2' \rhd t \,|\, \{c\} \cup C_1 \ . \end{aligned}$$

So

$$\mathrm{CE}(\mathbf{N}(e), \{c\} \cup C_2) = L_1 \cup L^{(1)} \cup \ldots \cup L^{(n)} \rhd t \,|\, C_1 \ ,$$

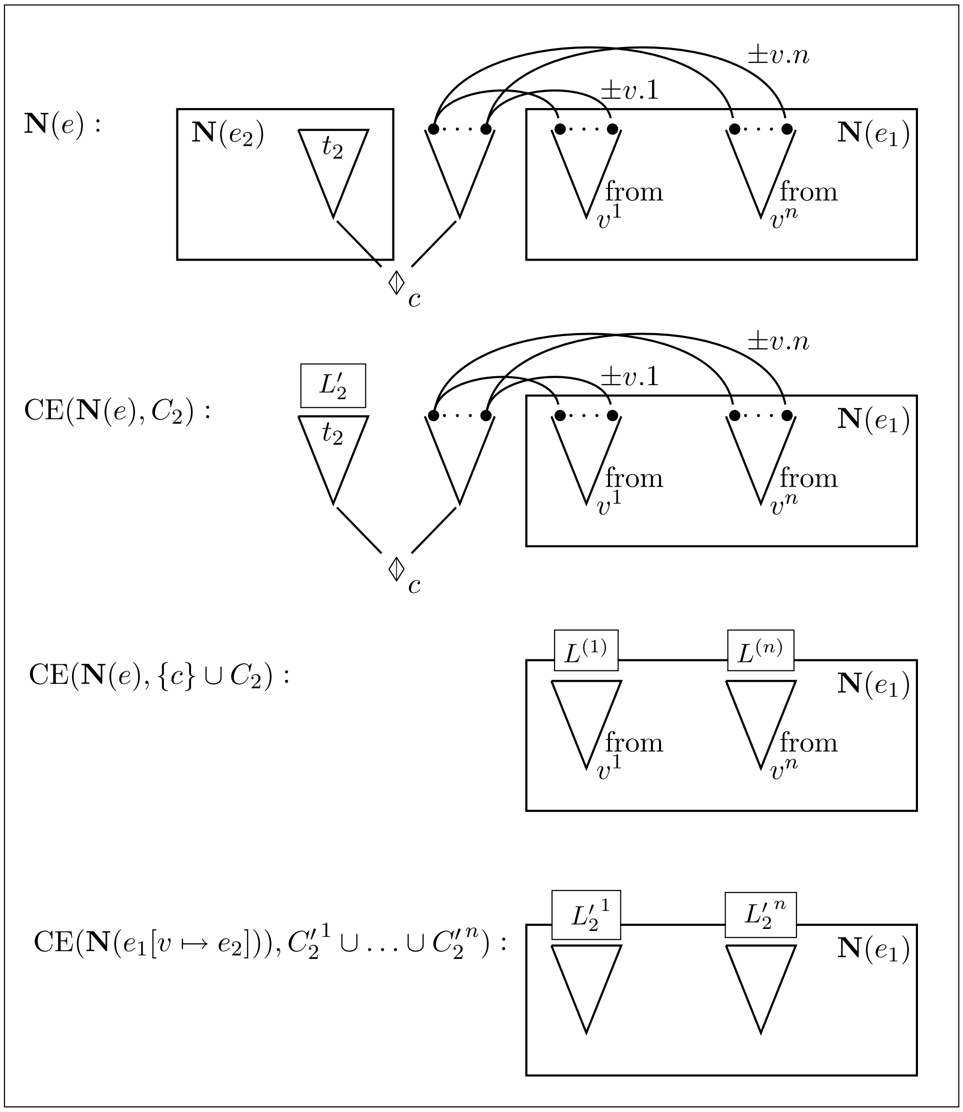


Table 4.4: Proof nets occurring in the proof of 4.4.31

where

$$\begin{aligned} L^{(i)} = \{\, (l, l', (-,v,i)\sigma(+,v,i)) \mid\ & (l, c.2\pi, (-,v,i)) \in L_v, \\ & (c.1\pi, c.1\pi', \sigma) \in L'_2, \\ & (c.2\pi', l', (+,v,i)) \in L_v \,\} \end{aligned}$$

(labels of the form $(-,v,i)\sigma(+,v,j)$ for $i \neq j$ are not well-formed).

On the other hand, let $\mathbf{N}(e_1[v \mapsto e_2]) = L_s \rhd t \,|\, C_s$. This proof net contains n copies N_2^j, $j \in \{1, \dots n\}$ of $\mathbf{N}(e_2)$ (i.e. each N_2^j is strongly equivalent to $\mathbf{N}(e_2)$).

But then the normal forms ${N'_2}^j = {L'_2}^j \rhd {t'_2}^j \,|\, {C'_2}^j$ of the N^j are strongly equivalent to $\mathrm{CE}(\mathbf{N}(e_2))$, so reducing these nets in $\mathbf{N}(e_1[v \mapsto e_2])$ yields

$$\mathrm{CE}(\mathbf{N}(e_1[v \mapsto e_2]), {C'_2}^1 \cup \ldots \cup {C'_2}^n) = L_1 \cup {L'_2}^1 \cup \ldots \cup {L'_2}^n \rhd t \,|\, C_1 \ .$$

The links in $L^{(i)}$ and ${L'_2}^i$ correspond, and deletion of the label components of the form $(\pm, v, i)$ in the former yields exactly the the latter sets.

By lemma 4.4.15, the two proof nets are strongly equivalent. □

Remark 4.4.32. The last two propositions also hold, if the β-reduction occurs anywhere inside a term. The proofs are analog, if $\mathbf{N}(e_1)$ is replaced by "the whole of N, except $\mathbf{N}(e_2)$ and the part coming from the abstraction λv".

When weakening comes into play, i.e. when a binder does not bind any variable occurrence, we may lose information during β-reduction, that we keep with cut elimination. We have already seen an example in 4.4.27, but this is also a general result.

Proposition 4.4.33. *Let e be a λ-term containing the β-redex $(\lambda v.e_1)\, e_2$, where v does not occur freely in e_1. If e' is obtained from e by β-reduction of this subterm, then the reduced net* $\mathrm{CE}(\mathbf{N}(e))$ *is stronger than the reduced net* $\mathrm{CE}(\mathbf{N}(e'))$.

Proof. Let c be the cut in $\mathbf{N}(e)$ introduced by the application $(\lambda v.e_1)\, e_2$. We show that the elimination of c and all cuts introduced by e_2 in $\mathbf{N}(e)$ produces a proof net that is stronger than $\mathbf{N}(e')$. Table 4.5 shows the nets we will consider during this proof.

Let $\mathbf{N}(e) = L \rhd t \,|\, C$, and let $\mathbf{N}(e_2) = L_2 \rhd t_2 \,|\, C_2$ be the subnet of $\mathbf{N}(e)$ introduced by e_2.

We can write the result of the reduction of C_2 in $\mathbf{N}(e)$ in the form

$$\mathrm{CE}(\mathbf{N}(e), C_2) = L' \cup L^{\mathrm{i}} \cup L^{\mathrm{e}} \cup L^{\mathrm{m}} \rhd t \,|\, C \setminus C_2 \ .$$

Here

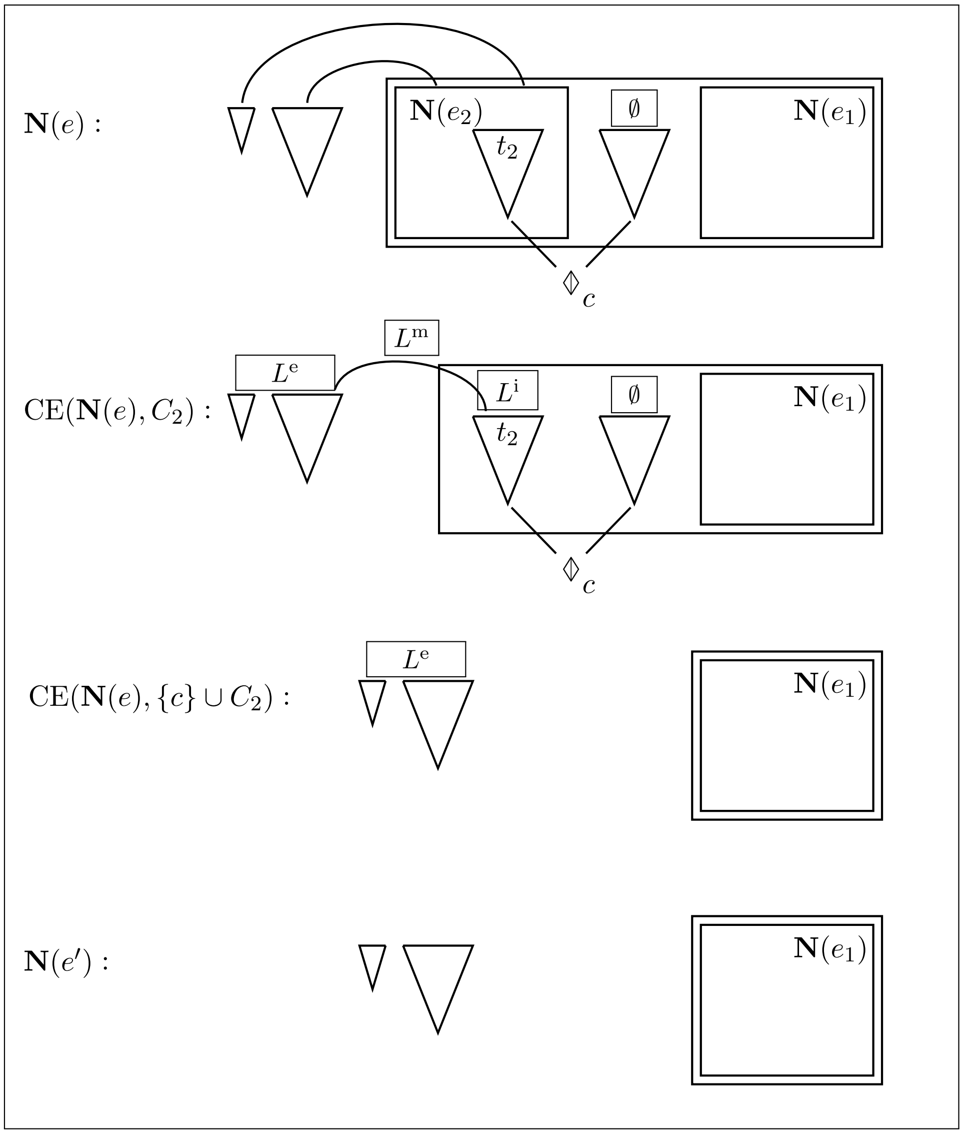


Table 4.5: Proof nets occurring in the proof of 4.4.33

- $L' \subseteq L$ are those links in $\mathbf{N}(e)$ that do not touch e_2 and hence are not affected by the cut elimination,
- L^{i} contains all links that begin and end in t_2 (i means internal),
- L^{e} contains all the new links that do not begin or end in t_2 (e means external), and
- L^{m} contains all links that begin or end in t_2 but not both (m means mixed).

Since v does not occur freely in e_1, the abstraction $\lambda v.e_1$ does not introduce any links. This means, that the elimination of c deletes all links that begin or end in t_2:

$$\mathrm{CE}(\mathbf{N}(e), \{c\} \cup C_2) = L' \cup L^{\mathrm{e}} \rhd t \,|\, C \setminus (\{c\} \cup C_2)$$

On the other hand, e' results from e by replacing the subterm $(\lambda v.e_1)\, e_2$ by e_1, so

$$\mathbf{N}(e') = L' \rhd t \,|\, C \setminus (\{c\} \cup C_2) \ .$$

This means, that $\mathrm{CE}(\mathbf{N}(e), \{c\} \cup C_2)$ is stronger than $\mathbf{N}(e')$, and by proposition 4.4.21 the same holds for $\mathrm{CE}(\mathbf{N}(e))$ and $\mathrm{CE}(\mathbf{N}(e'))$. ☐

The final two reduction methods on λ-terms are product- and sum-reduction. Both are somehow similar to weakening, in so far as they also throw away information that is kept during cut elimination of the corresponding prenet.

Proposition 4.4.34. *Let e be a λ-term containing one of the product-redices $\pi_1(\mathit{pair}\, e_1\, e_2)$ and $\pi_2(\mathit{pair}\, e_1\, e_2)$. If e' is obtained from e by product-reduction of this subterm, then the reduced net $\mathrm{CE}(\mathbf{N}(e))$ is stronger than the reduced net $\mathrm{CE}(\mathbf{N}(e'))$.*

Proof. We only consider the reduction of the product-redex $\pi_1(\mathrm{pair}\, e_1\, e_2)$ to e_1. The case $\pi_2(\mathrm{pair}\, e_1\, e_2) \rightsquigarrow e_2$ is completely analogous. Table 4.6 shows the nets we will talk about during this proof.

Let c_1, c_2 be the cuts in $\mathbf{N}(e)$ introduced by the pairing of e_1 and e_2, and let c be the cut introduced by the projection. We show that the elimination of c_1, c_2, c and all cuts introduced by e_2 in $\mathbf{N}(e)$ yields a stronger prenet than $\mathbf{N}(e')$.

Let $\mathbf{N}(e) = L \rhd t \,|\, C$, let $\mathbf{N}(\pi_1(\mathrm{pair}\, e_1\, e_2)) = L_r \rhd t_r \,|\, C_r$ be the subnet of $\mathbf{N}(e)$ introduced by the redex, and let $\mathbf{N}(e_i) = L_i \rhd t_i \,|\, C_i$, $i \in \{1, 2\}$ be the subnet introduced by e_i.

We can write the result of the reduction of C_2 in $\mathbf{N}(e)$ in the form

$$\mathrm{CE}(\mathbf{N}(e), C_2) = L' \cup L^{\mathrm{i}} \cup L^{\mathrm{e}} \cup L^{\mathrm{m}} \rhd t \,|\, C \setminus C_2 \ .$$

Here again

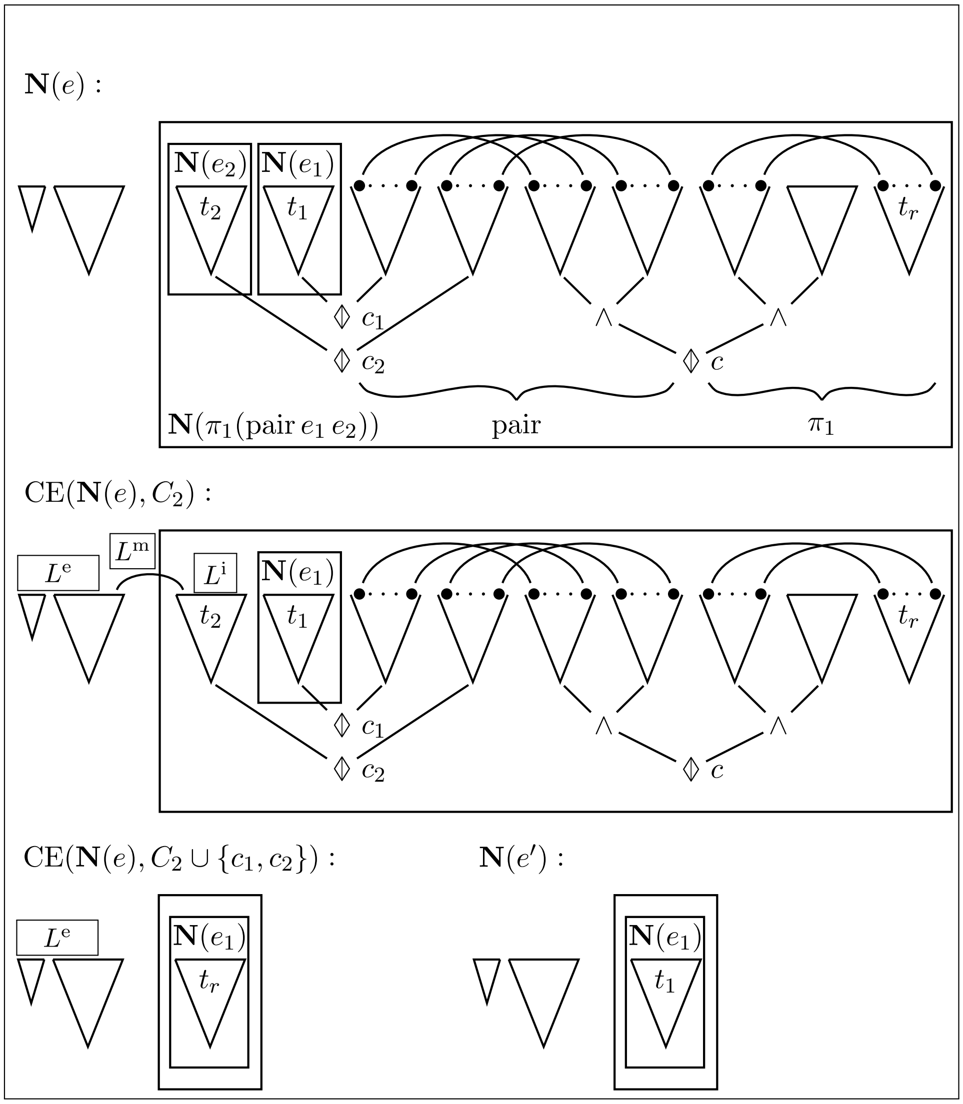


Table 4.6: Proof nets occurring in the proof of 4.4.34

- $L' \subseteq L$ are those links in $\mathbf{N}(e)$ that do not touch e_2 and hence are not affected by the cut elimination,
- L^{i} contains all links that begin and end in t_2,
- L^{e} contains all the new links that do not begin or end in t_2, and
- L^{m} contains all links that begin or end in t_2 but not both.

Further elimination of c_1, c_2 and c deletes all links in L^{i} and L^{m} and extends all other links touching t_1 or t_2 by the ε-labeled links in L_{pair} and L_π. So we can write the result of this second step of cut eliminations as

$$\mathrm{CE}(\mathbf{N}(e), C_2 \cup \{c_1, c_2\}) = \tilde{L}' \cup L^{\mathrm{e}} \rhd t \,|\, C \setminus (C_2 \cup \{c_1, c_2\}) \ ,$$

where t_r takes over the role of t_1, i.e.

$$\begin{aligned}\tilde{L}' = &\big\{(l, t_r.\pi, \sigma) \,|\, (l, t_1.\pi, \sigma) \in L' \wedge l \notin \mathrm{leaves}(t.1)\big\} \\ &\cup \big\{(t_r.\pi, l, \sigma) \,|\, (t_1.\pi, l, \sigma) \in L' \wedge l \notin \mathrm{leaves}(t.1)\big\} \\ &\cup \big\{(t_r.\pi, t_r.\pi', \sigma) \,|\, (t_1.\pi, t_1.\pi', \sigma) \in L'\big\} \\ &\cup \big\{(l, l', \sigma) \in L' \,|\, l, l' \notin \mathrm{leaves}(\{c_1, c_2, c\})\big\} \ .\end{aligned}$$

The obvious identification of the links in $\tilde{L}'$ with those of the proof net $\mathbf{N}(e')$, which does not contain links corresponding to L^{e}, shows that $\mathrm{CE}(\mathbf{N}(e), C_2 \cup \{c_1, c_2\})$ is stronger than $\mathbf{N}(e')$.

According to lemma 4.4.20, this relation is preserved by cut elimination, so $\mathrm{CE}(\mathbf{N}(e))$ is stronger than $\mathrm{CE}(\mathbf{N}(e'))$. $\square$

Proposition 4.4.35. *Let e be a λ-term containing one of the sum-redices $\mathit{case}\,(\mathit{inl}\, e_1)\, e_2\, e_3$ and $\mathit{case}\,(\mathit{inr}\, e_1)\, e_2\, e_3$. If e' is obtained from e by sum-reduction of this subterm, then the reduced net* $\mathrm{CE}(\mathbf{N}(e))$ *is stronger than the reduced net* $\mathrm{CE}(\mathbf{N}(e'))$.

Proof. We only consider the reduction of the sum-redex $\mathrm{case}(\mathrm{inl}\, e_1)\, e_2\, e_3$ to $e_2\, e_1$. The case $\mathrm{case}(\mathrm{inr}\, e_1)\, e_2\, e_3 \rightsquigarrow e_3\, e_1$ is completely analogous.

Let c be the cut introduced by the application $\mathrm{inl}\, e_1$, let c_1, c_2, c_3 be the cuts in $\mathbf{N}(e)$ introduced by the successive application of $(\mathrm{inl}\, e_1)$, e_2 and e_3 to the constant case, and let $\mathbf{N}(e_3) = L_3 \rhd t_3 \,|\, C_3$.

We can apply atomic cut elimination to reduce c_1 to two cuts c_{11} and c_{12}. From here on, the most interesting aspect of the proof is the order in which cuts are reduced. Apart from that, the proof contains no ideas that were not present in the proof of proposition 4.4.34, so we do not show the explicit computation of the cut eliminations in $\mathbf{N}(e)$, but give the shape of the nets that are reached by successive elimination of $\{c, c_2\}$, $\{c_{12}\} \cup C_3$ and c_3, as well as the proof net $\mathbf{N}(e')$ in table 4.7.

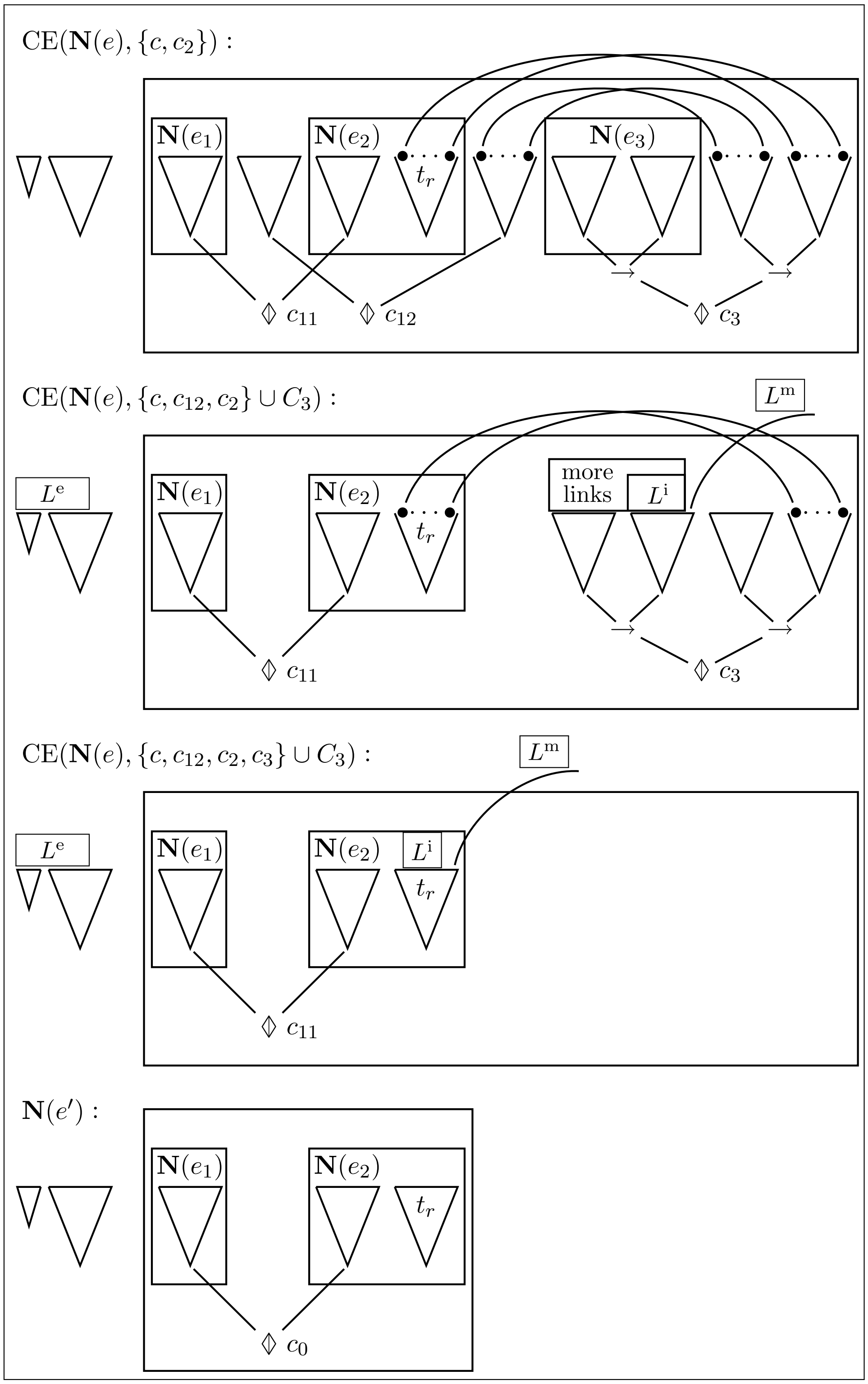


Table 4.7: Proof nets occurring in the proof of 4.4.35

The obvious identification of the links in $\mathrm{CE}(\mathbf{N}(e), \{c, c_{12}, c_2, c_3\} \cup C_2)$, excluding L^{e}, L^{i} and L^{m}, with the links in $\mathbf{N}(e')$, as well as the identification of c_{12} with c' shows that the former net is stronger than the latter.

According to lemma 4.4.20, this relation is preserved by cut elimination, so $\mathrm{CE}(\mathbf{N}(e))$ is stronger than $\mathrm{CE}(\mathbf{N}(e'))$. $\square$

Chapter 5

Conclusions

In this thesis, I presented a new way to write down intuitionistic proofs. Starting from classical proof nets, I developed a class of intuitionistic proof nets and a way to interpret λ-terms as proof nets and vice versa, together with linear time algorithms for this interpretation. I analyzed different geometrical properties of intuitionistic proof nets, e.g. their polarizations and the interaction of their links.

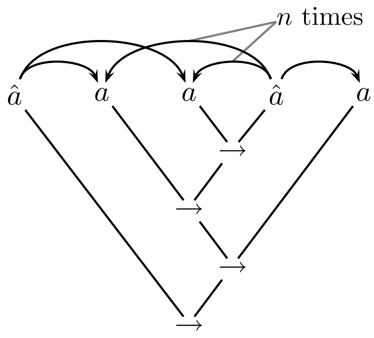
Moreover, I presented a cut elimination for these proof nets and compared this procedure to the well-known normalization procedures of λ-terms. As cut elimination of intuitionistic proof nets is a terminating and confluent normalization procedure, I effectively created a new notion of the equivalence of intuitionistic proofs or, equivalently, of functional programs in the λ-calculus.

The translation of simply typed λ-terms into proof nets and the cut elimination procedure were implemented in Java. The sources can be found online at `http://www.ps.uni-sb.de/~horbach/thesis/`.

Future work

There are a lot of questions that naturally arise during the work with proof nets but still remain to be answered. The following are some of the most important ones:

- By now, little is known about complexity issues. Computations on proof nets are often more space- and time-efficient than the corresponding computations on λ-terms. The β-normal form of the term $\lambda x.\lambda z.(\lambda y.z\, y\, y)^{n+1}\, x$, for example, has a size exponential in n and is reached after up to exponentially many reductions, depending on the order in which redices are evaluated. However, the corresponding normal proof net



has only linearly many links with labels of constant length, and it can be computed in linear time. This effect seems to originate in the immense sharing that is present in prenets.

- Are there sensible extensions of intuitionistic proof nets that work for larger logics? First candidates are classical logic and intuitionistic predicate logic. Is it, for example, possible to equip a classical proof net with links in such a way, that the paths followed during intuitionistic cut elimination are the same ones that are followed during classical cut elimination?

- Intuitionistic logic can be regarded as a fragment of both classical and linear logic. It would be interesting to further compare cut elimination for classical or linear proof nets to the images of intuitionistic cut elimination under these embeddings.

- The calculus of structures allows for much more proofs than the λ-calculus. Is there a decent translation of these proofs into proof nets?

- We still do not have a complete classification of those intuitionistic prenets whose underlying formula is intuitionistically valid.

Bibliography

[And76] Peter B. Andrews. Refutations by matings. *IEEE Transactions on Computers*, C-25:801–807, 1976.

[Bar84] Henk Barendregt. *The Lambda Calculus: Its Syntax and Semantics.* Number 103 in Studies in Logic and the Foundations of Mathematics. North–Holland, Amsterdam, revised edition, 1984.

[Bar92] Henk Barendregt. Lambda calculi with types. In Abramsky, Gabbay, and Maibaum, editors, *Handbook of Logic in Computer Science, Volumes 1 (Background: Mathematical Structures) and 2 (Background: Computational Structures)*, volume 2. Clarendon, 1992.

[Bar97] Henk Barendregt. The impact of the lambda calculus on logic and computer science. *Bulletin of Symbolic Logic*, 3(3):181–215, 1997.

[Bib81] Wolfgang Bibel. On matrices with connections. *Journal of the ACM*, 28:633–645, 1981.

[Bro07] Luitzen Egbertus Jan Brouwer. *Over de Grondslagen der Wiskunde.* PhD thesis, University of Amsterdam, 1907.

[CF58] Haskell Brooks Curry and Robert Feys. *Combinatory Logic.* North-Holland, 1958.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[Chu33] Alonzo Church. A set of postulates for the foundation of logic. *Annals of Mathematics*, 33(2):346–366 and 34, 839–864, 1932/33.

[Cur34] Haskell Brooks Curry. Functionality in combinatory logic. In *Proceedings of Natural Academy of Sciences U.S.A.*, volume 20, pages 584–590, 1934.

[dB72] Nicolas de Bruijn. Lambda-calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the church-rosser theorem. *Indagaciones Mathematische*, 34(5):381–392, 1972.

[DR89] Vincent Danos and Laurant Regnier. The structure of multiplicatives. *Archives of Mathematical Logic*, 28:181–203, 1989.

[Gen35] Gerhard Gentzen. Untersuchungen über das logische Schließen. *Mathematische Zeitschrift*, 39:176–210,405–431, 1935.

[Gir87] Jean Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[GLR95] Jean-Yves Girard, Yves Lafont, and Laurent Regnier, editors. *Geometry of Interaction III: Accommodating the Additives*. Cambridge University Press, 1995.

[GTL89] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and Types*. Cambridge University Press, 1989.

[Gug02] Alessio Guglielmi. A system of interaction and structure. Technical Report WV-02-10, Technische Universität Dresden, 2002. Accepted by ACM Transactions on Computational Logic.

[Hey25] Arend Heyting. *Intuitionistische Axiomatiek der projektieve Meetkunde*. PhD thesis, University of Amsterdam, 1925.

[Hey56] Arend Heyting. *Intuitionism: An Introduction*. North-Holland, 1956.

[Hil00] David Hilbert. Mathematische Probleme. *Nachrichten von der Königlichen Gesellschaft der Wissenschaften zu Göttingen, mathematisch-physikalische Klasse*, 3(1):253–297, 1900.

[Hin97] Roger Hindley. *Basic Simple Type Theory*. Cambridge University Press, 1997.

[How80] William Alvin 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.

[HvG03] Dominic Hughes and Rob van Glabbeek. Proof nets for unit-free multiplicative-additive linear logic. In *18th IEEE Symposium on Logic in Computer Science (LICS 2003)*, pages 1–10, 2003.

[Lam95] François Lamarche. Proof nets for intuitionistic linear logic I: Essential nets. Technical report, Imperial College, London, 1995.

[LR96] François Lamarche and Christian Retoré. Proof nets for the Lambek calculus — an overview. In Michele Abrusci and Claudia Casadio, editors, *Proofs and Linguistic Categories*, volume 46, pages

241–262. Cooperativa Libraria Universitaria Editrice Bologna, 1996.

[LS05] François Lamarche and Lutz Straßburger. Naming proofs in classical propositional logic. In Paweł Urzyczyn, editor, *Typed Lambda Calculi and Applications, TLCA 2005*, volume 3461 of *Lecture Notes in Computer Science*, pages 246–261. Springer-Verlag, 2005.

[Pie02] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, Cambridge, 2002.

[Pra65] Dag Prawitz. *Natural Deduction. A Proof-Theoretic Study.* PhD thesis, Stockholm University, 1965.

[Pra71] Dag Prawitz. Ideas and results in proof theory. In J. E. Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, pages 235–307, Amsterdam, 1971. North-Holland.

[RW10] Bertrand Arthur William Russell and Anthony Whitehead. *Principia Mathematica*. Cambridge University Press, 1910.

[SL04] Lutz Straßburger and François Lamarche. On proof nets for multiplicative linear logic with units. In Jerzy Marcinkowski and Andrzej Tarlecki, editors, *Computer Science Logic, CSL 2004*, volume 3210 of *Lecture Notes in Computer Science*, pages 145–159. Springer-Verlag, 2004.

[Str05] Lutz Straßburger. From deep inference to proof nets. In Paola Bruscoli François Lamarche and Charles Stewart, editors, *Structures and Deduction*, pages 2–18. Satellite Workshop of ICALP05, 2005.

[Tai67] William Tait. Intensional interpretations of functionals of finite type I. *Journal of Symbolic Logic*, 32(2):198–212, 1967.

[Tiu05] Alwen Tiu. A local system for intuitionistic logic: Preliminary results. Draft, March 2005.

[TW01] Ruediger Thiele and Larry Wos. Hilbert's twenty-fourth problem, 2001.

Nomenclature

$\mathcal{A}$	the set of atoms
$\perp$	falsity
$\mathcal{F}$	the set of formulas
$\mathcal{V}$	the set of variables
τ	a fixed type assignment
Λ	the set of all λ-preterms
$e : A$	e is of type A (wrt. τ)
$\lambda^{\rightarrow}$	simply typed λ-terms
$\lambda^{\rightarrow\wedge}$	λ-terms restricted to $\rightarrow$ and $\wedge$
$FV(e)$	the free variables of the term e
$e'[v \mapsto e]$	variable substitution
$f\colon X \rightharpoonup Y$	a partial function f from X to Y
$f\|_X$	the restriction of the domain of f to X
$\mathrm{dom}(f)$	the domain of f
$\mathrm{im}(f)$	the image of f
X^*	the set of finite paths over X
ε	the empty path
$\pi\rho$	the concatenation of the paths π and ρ
$\lozenge$	the cut connector
$\hat{\mathcal{A}}$	the set of duals of atoms
$\mathcal{N}$	the set of nodes of trees

$\mathcal{T}$	the set of (binary) trees
leaves(t)	the set of leaves of the tree t
$t.\pi$	the subtree of t at position π
$t_1 \between t_2$ etc.	a tree with root $\between$ and direct subtrees t_1 and t_2
$\mathbf{T}(A, n, \pi)$	the tree corresponding to the formula A, with additional information n and π
$\mathbf{T}(A, n)$	equals $\mathbf{T}(A, n, \varepsilon)$
$\mathbf{ty}(t)$	the type coded in the tree t
$\mathcal{L}$	the set of link labels
$L \rhd t \mid C$	a prenet with main tree t, cut set C and linking L
leaves(N)	the set of leaves of all trees in N
v^i	the occurrence of v marked by i
$\mathbf{N}(e)$	the prenet assigned to the term e
$\circ$, $\bullet$	polarizations of nodes
path(N, C)	the paths in N through C
path(N)	the paths in N through any set of cuts
CE(N, c)	elimination of the cut c in N
CE(N)	elimination of all cuts in N
$\mathrm{CE_a}(N, c)$	atomic elimination of the cut c in N

Index