Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Master's Program in Computer Science

Master's Thesis

Proof Representations for Higher-Order Logic

submitted by
Christine S Rizkallah
on December 5, 2009

Supervisor
Prof. Dr. Gert Smolka
Advisor
Dr. Chad E Brown

Reviewers
Prof. Dr. Gert Smolka
Dr. Chad E Brown

Eidesstattliche Erklärung
Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath
I hereby confirm under oath that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung
Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent
I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, ...
(Datum / Date) (Unterschrift / Signature)

Acknowledgment

First of all, I would like to thank Chad E Brown and Prof. Gert Smolka for offering me this exciting topic. I am very grateful to Chad for the huge amount of time that he invested and for the constant assistance he gave me. I also thank Prof. Smolka for offering me advice when I needed it. Finally, I thank Georg Neis for proofreading my thesis.

Abstract

We provide a mapping from classical extensional tableau proofs of higher-order formulas to intuitionistic non-extensional natural deduction proofs of semantically equivalent formulas. We show that the Kuroda transformation, which is known to map from first-order classical logic to first-order intuitionistic logic, extends to elementary type theory. Moreover, we introduce a transformation that we call Girard-Kuroda-Per and prove that this transformation maps from classical extensional to intuitionistic non-extensional simple type theory.

Contents

Chapter 1

Introduction

Given a classical extensional tableau proof of a higher-order formula s, it is not always possible to find an intuitionistic non-extensional natural deduction proof of s. For example, the formula $p \vee \neg p$ is provable classically but not intuitionistically. Our aim is to find a transformed formula s' that is semantically equivalent to s and for which we are able to give an intuitionistic non-extensional natural deduction proof. We define a transformation which translates formulas s to semantically equivalent formulas s' and give a procedure for creating a intuitionistic non-extensional natural deduction proof of s' given a classical extensional proof of s.

Over the years a lot of work has been done on translating from classical to intuitionistic logic. In 1929, Glivenko found a straightforward transformation from propositional classical to propositional intuitionistic logic [19]. *Glivenko's theorem* states that an arbitrary propositional formula f is classically provable, if and only if $\neg\neg f$ is intuitionistically provable [19, 32, 31].

Glivenko's theorem does not extend to first-order predicate logic. However, there are several logical transformations that transform classical first-order logic to intuitionistic first-order logic: the *Kolmogorov negative translation* [29], the *Gödel-Gentzen negative translation* [29], and the *Kuroda negative translation* [29, 24]. The latter double negates the formula and adds double negations after each occurrence of the $\forall$ quantifier. A recent paper by Zdanowski [32] shows that Glivenko's theorem also holds for second-order propositional logic without the $\forall$ quantifier.

The Kolmogorov negative translation and the Gödel-Gentzen negative translation do not extend using our definitions to higher-order logic. This is because they do not map propositional variables to themselves but rather to their double negation and thus they are not compositional. We depend on the fact that the transformations we use are compositional in proving that they satisfy our goal. Thus we use the Kuroda transformation which is compositional and show that it extends to elementary type theory.

In 1956, Gandy introduced a transformation that maps from extensional to non-extensional simple type theory [14]. It translates equality with the help of a binary relation and a predicate that are defined by mutual recursion. The transformation we introduce, called Girard-Kuroda-Per, transforms equality with the help of a single binary relation that is defined inductively on types, and which turns out to be a partial equivalence relation. We prove that our transformation maps from classical extensional to intuitionistic non-extensional simple type theory.

As we have seen, mapping from one logic to another is an interesting problem that has been investigated for a long time. Our practical motivation for working on this problem is related to a software called JavaScript Interactive Higher-Order Tableau Prover (Jitpro) developed by Brown [33]. Jitpro is a relatively new interactive theorem prover for higher-order logic that outputs classical extensional tableau proofs. Our goal is to be able to translate these proofs

into intuitionistic non-extensional natural deduction proof terms, which can then be verified by Coq [5], a widely used proof assistant.

The thesis is structured as follows. In Chapter 2 we give a brief overview of simple type theory and in Chapter 3 we introduce the tableau calculi that we consider. We then present the targeted natural deduction calculus $\mathcal{N}$ in Chapter 4. In Chapter 5 we declare some definitions and helpful theorems that we extensively use in the subsequent three chapters. In particular, we introduce in the notion of logical transformation and of a logical transformation respecting a tableau calculus, and then prove that our problem is reducible to providing a logical transformation that respects a complete higher-order tableau calculus. In Chapter 6 we introduce a logical transformation (*Girard transformation*) that respects most of the tableau rules. In Chapter 7 we modify this logical transformation to obtain one (*Girard-Kuroda transformation*) that respects an additional rule. We show that Girard-Kuroda maps classical elementary type theory to intuitionistic elementary type theory. Chapter 8 includes the *Girard-Kuroda-Per transformation*, which respects a specific complete higher-order tableau calculus called $\mathcal{T}_{Full_{res}}$. We conclude in Chapter 9 and provide suggestions for future work.

Chapter 2

Simple Type Theory

We use *simple type theory* based on the simply typed λ-calculus, which is the most prominent form of higher-order logic [13, 2]. Simple type theory originated in 1940 with Church [10] but goes back to ideas of Ramsey [26] and Chwistek [11] in the 1920s. Its purpose was to simplify the ramified theory of types that was first introduced by Russell in 1908 [27], then used for formalizing some fragments of mathematics in *Principia Mathematica* by Russell and Whitehead in 1913 [30]. It provides the logical base of the proof assistants Isabelle [25] and HOL [20].

Henkin defined *standard* and *general* interpretations for typed λ-calculus [21]. He proved the completeness of simple type theory with respect to general interpretations [9]. We use the general Henkin interpretation, since it corresponds to our tableau calculi and it makes our result more general.

We now give a brief introduction to simple type theory. For more details, please see Barendregt [3] and Hindley [22].

2.1 Syntax

We will now define simply typed terms as syntactic objects.

Definition 2.1.1 (Types)**.** Let $\{o, \iota\}$ be the set of *basic types*. The set $\mathbb{T}$ of *types* is defined inductively as

$$\mathbb{T}(\sigma, \tau, \dots) ::= \iota \mid o \mid \sigma \to \tau$$

where o is the type of propositions, ι the type of individuals, and $\to$ is a function type constructor.

We use α to range over the *basic types* and $\sigma, \tau, \sigma_1, \sigma_2, \dots$ to range over elements of $\mathbb{T}$. A type of the form $\sigma \to \tau$ is called a *function type*. Parentheses in types will often be omitted by association to the right. For example, by $\sigma \to \sigma \to \tau$ we mean $\sigma \to (\sigma \to \tau)$. Moreover, we often drop the arrows, so that the example becomes $\sigma\sigma\tau$.

Definition 2.1.2 (Terms)**.** Let $\mathcal{V}$ be a countably infinite set of *variables*. Assume $\mathbb{T} \cap \mathcal{V} = \emptyset$. The set of *terms* Ter is defined as

$$Ter(s, t, \dots) ::= x \mid c \mid s\ t \mid \lambda x.s$$

where $x \in \mathcal{V}$ and $c \in \{\top, \bot, \neg, \to, \wedge, \vee\} \cup \{\forall^\sigma, \exists^\sigma, =_\sigma \mid \sigma \in \mathbb{T}\}$.

This means that a term is either a variable, a logical constant, an application or a λ-abstraction. We use x, y to range over the variables and $s, t, s_1, s_2, \ldots$ to range over elements of *Ter*. The $\forall$, $\exists$ and $=$ constants are indexed by a type σ to denote that the set of logical constants includes quantifiers and equality at all types. The set is infinite.

We assume that every variable has a unique type. While names and abstractions are always well-formed, an application $s\ t$ is only well-formed if the type of s is a function type whose argument type is the type of t.

The following are the typing rules of simply typed terms:

$$\frac{}{x : \sigma}\ x \text{ has type } \sigma \qquad \frac{x : \sigma \qquad s : \tau}{\lambda x.s : \sigma\tau} \qquad \frac{s : \sigma\tau \qquad t : \sigma}{st : \tau}$$

The types of the logical constants are as follows:

$$\top : o,\ \bot : o,\ \neg : oo,\ \rightarrow : ooo,\ \wedge : ooo,\ \vee : ooo,\ \forall^\sigma : (\sigma o)o,\ \exists^\sigma : (\sigma o)o, =_\sigma : \sigma\sigma o$$

Following the typing rules, every well-formed term will have a unique type. We write $s : \sigma$ to say that s is a term of type σ. We use Ter^σ to mean the set of terms of type σ. We only consider well typed terms.

Definition 2.1.3 (Free Variables FV)**.** The set of *free variables* of a term s, written FV, is defined as follows:

$$\begin{array}{lcl} FV(x) & := & \{x\} \\ FV(s\,t) & := & FV(s) \cup FV(t) \\ FV(\lambda x.s) & := & FV(s) - \{x\} \end{array}$$

Definition 2.1.4 (Ground Term)**.** A term s is *ground* if $FV(s) = \emptyset$.

Definition 2.1.5 (Free Variables FV^*)**.** The set of *free variables* of a set of terms X is defined as:

$$FV^*(X) := \bigcup_{s \in X} FV(s)$$

Definition 2.1.6 (Fragment)**.** A *fragment* $\mathcal{F}$ is a subset of the set of terms *Ter*. *Ter* itself is called the *full fragment*.

We have the usual notion of α-, β-, η- equivalence ($\sim_\alpha$, $\sim_\beta$, $\sim_\eta$) and of β normal form ($\lceil s \rceil^\beta$). We consider equality of terms up to α-equivalence.

We write $s[x := t]$ for the capture avoiding substitution of term t for variable x in term s. A *simultaneous substitution* θ substitutes several variables simultaneously. The *identity substitution* substitutes each variable by itself yielding the same term it has been applied to. We use the notation $\theta, [x := t]$ to mean the simultaneous substitution that agrees with θ on all variables except (possibly) x, which is mapped to t.

2.2 Semantics

In this section, we define the notion of a general Henkin interpretation of types and terms. We will define formulas and interpretations and explain what it means that an interpretation satisfies a formula.

Before we define what an interpretation is, we need a general notion of function.

Definition 2.2.1 (Function)**.** A *function* f is a set of pairs such that for no pair $(x,\ y)\ \in\ f$ there is a $z\ \neq\ y$ with $(x,\ z)\ \in\ f$. The *domain* and the *range* of a function f are defined as follows:

$$\text{Dom } f\ :=\ \{x \mid \exists y :\ (x,\ y)\ \in\ f\}$$
$$\text{Ran } f\ :=\ \{y \mid \exists x :\ (x,\ y)\ \in\ f\}$$

We write $f : M \to N$ to state that f is a function such that Dom $f = M$ and Ran $f \subseteq N$. Moreover, we write $f(x)$ to refer to the y such that $(x,\ y)\ \in\ f$.

We first define the interpretation of types, then use it to define the interpretation of terms.

Definition 2.2.2 (Frame)**.** A *frame* is a function $\mathcal{D}$ defined on $\mathbb{T}$ that satisfies the following properties:

1. $\mathcal{D}(o) = \{0, 1\}$
2. $\forall \sigma \in \mathbb{T} :\ \mathcal{D}(\sigma) \neq \emptyset$
3. $\forall \sigma, \tau \in \mathbb{T} :\ \mathcal{D}(\sigma\tau) \subseteq \{f \mid f : \mathcal{D}(\sigma) \to \mathcal{D}(\tau)\}$

We extend frames to *assignments*, which also give meaning to variables. Then we define an operator that lifts an assignment to an *evaluation*, which assigns meanings to terms. Based on that we define the class of interpretations we are actually interested in.

Definition 2.2.3 (Assignment)**.** An *assignment into a frame* $\mathcal{D}$ is a function $\mathcal{I}$ defined on $\mathbb{T} \cup \mathcal{V}$ such that

1. $\mathcal{D} \subseteq \mathcal{I}$
2. $\mathcal{I}(x) \in \mathcal{I}(\sigma)$ for all types σ and variables $x : \sigma$

Let $\mathcal{I}$ be an assignment into a frame $\mathcal{D}$, $x : \sigma$ be a variable, and $a \in \mathcal{I}(\sigma)$. We write $\mathcal{I}^x_a$ to denote the assignment into $\mathcal{D}$ that agrees everywhere with $\mathcal{I}$ except possibly on x where it yields a.

Definition 2.2.4 (Evaluation Function)**.** We define a function $\hat{}$ that maps every assignment $\mathcal{I}$ into a function $\hat{\mathcal{I}} \subseteq \{(s, a) \mid \exists \sigma :\ (s : \sigma) \wedge a \in \mathcal{I}(\sigma)\}$ as follows:

1. $\hat{\mathcal{I}}(x) := \mathcal{I}(x)$
2. $\hat{\mathcal{I}}(c) := f$ if $c : \sigma$, $f \in \mathcal{I}(\sigma)$, and f has the usual classical meaning of c
3. $\hat{\mathcal{I}}(s\,t) := \hat{\mathcal{I}}(s)(\hat{\mathcal{I}}(t))$
4. $\hat{\mathcal{I}}(\lambda x.s) := f$ if $\lambda x.s : \sigma\tau$, $f \in \mathcal{I}(\sigma\tau)$, and $\forall a \in \mathcal{I}(\sigma) :\ \widehat{\mathcal{I}^x_a}(s) = fa$

We call $\hat{\mathcal{I}}$ the *evaluation function* of $\mathcal{I}$.

Definition 2.2.5 (Interpretation)**.** $\mathcal{I}$ is an *interpretation* if $\hat{\mathcal{I}}$ is a total evaluation function, i.e., if $\hat{\mathcal{I}}$ is an evaluation function that assigns a meaning to *every* (well typed) term. We write *Interp* for the set of all interpretations.

Note that not every evaluation function is total. This is due to the interpretation of λ-abstractions and logical constants. Consider the logical constant $\neg : oo$. $\hat{\mathcal{I}}(\neg)$ might not be defined. This is because, even though we know that the negation function is in $\mathcal{I}(o) \to \mathcal{I}(o)$, it might not be in $\mathcal{I}(oo)$. Recall that in the definition of frame we allow $\mathcal{I}(oo)$ to be a subset of $\mathcal{I}(o) \to \mathcal{I}(o)$. For more details see [28].

Definition 2.2.6 (Satisfies)**.** A *formula* is a term of type o. We say an interpretation $\mathcal{I}$ *satisfies* a formula s if $\hat{\mathcal{I}}(s) = 1$.

Definition 2.2.7 (Satisfiable / Unsatisfiable)**.** A formula is *satisfiable* if there exists an interpretation that satisfies it. It is *unsatisfiable* if it is not satisfiable.

Definition 2.2.8 (Valid)**.** A formula is *valid* if all interpretations satisfy it.

Proposition 2.2.9. A formula is *valid* if and only if its negation is *unsatisfiable*.

Chapter 3

Tableau Calculi

In order to determine whether or not a given formula is valid, we could either introduce a proof system for checking its validity or give a refutation system for checking the unsatisfiability of the formula's negation (recall Proposition 2.2.9). A *tableau system* is a refutation system that provides a mechanical method of refuting a formula. In this section, we will introduce a tableau refutation calculus and describe how to use it. First, we need to introduce some preliminary definitions.

3.1 Definitions

Definition 3.1.1 (Branch)**.** A *branch* is a finite set of β-normal formulas.

Given a fragment $\mathcal{F}$, an $\mathcal{F}$-*branch* is a branch containing only formulas that lie in $\mathcal{F}$.

Definition 3.1.2 (*Branches*)**.** *Branches* is the set of all branches.

Definition 3.1.3 (Ground Branch)**.** A branch A is *ground* if $FV^*(A) = \emptyset$.

Definition 3.1.4 (Tableau Step)**.** A *tableau step* is a tuple of branches $\langle A, A_1, \ldots, A_n \rangle$ with $n \geq 0$ such that $A \subset A_i$ for each $i \in 1, ..., n$. It is presented as a refutation rule of the form,

$$\frac{A_1 \vdash_{\mathcal{T}} \bot \quad \ldots \quad A_n \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot}$$

or in a tableau view of the form,

$$\begin{array}{c|c|c} \multicolumn{3}{c}{A} \\ \hline A_1 & \ldots & A_n \end{array}$$

We call A the *head* of the tableau step and each A_i an *alternative* of the step. If $n \geq 2$ we say the step is *branching*.

A *tableau rule* is a set of tableau steps. We often give these as schemas.

A tableau step *applies to* a branch A if A is the head of this step. A tableau rule *applies to* a branch A if one of its steps applies to A.

Definition 3.1.5 (Tableau Calculus)**.** A *tableau calculus* $\mathcal{T}$ is a tuple $\langle \mathcal{F}, \mathcal{R} \rangle$ where $\mathcal{F}$ is a fragment and $\mathcal{R}$ is a set of tableau rules. If the $\mathcal{F}$ is not explicitly specified, we assume it is the full fragment.

Definition 3.1.6 (Closed)**.** A tableau step is *closed* if it has no alternatives. Let $\mathcal{T}$ be a tableau calculus. A branch A is $\mathcal{T}$-*closed* if A is the head of one of the closed steps in the rules in $\mathcal{T}$.

Definition 3.1.7 (Refutable)**.** Let $\mathcal{T} = \langle \mathcal{F}, \mathcal{R} \rangle$ be a tableau calculus. An $\mathcal{F}$-branch A is $\mathcal{T}$-*refutable* if there is an $r \in \mathcal{R}$ that contains a tableau step $\langle A, A_1, \ldots, A_n \rangle$ where $A_1, \ldots, A_n$ are $\mathcal{F}$-branches that are $\mathcal{T}$-refutable.

Theorem 3.1.8. Let $\mathcal{T} = \langle \mathcal{F}, \mathcal{R} \rangle$ and $\mathcal{T}' = \langle \mathcal{F}', \mathcal{R}' \rangle$ be two tableau calculi such that $\mathcal{R} \subseteq \mathcal{R}'$ and $\mathcal{F} \subseteq \mathcal{F}'$. If an $\mathcal{F}$-branch A is $\mathcal{T}$-refutable, then it is also $\mathcal{T}'$-refutable.

Proof. Assume A is an $\mathcal{F}$-branch that is $\mathcal{T}$-refutable. We prove this theorem by induction on the derivation of A being $\mathcal{T}$-refutable. By definition of $\mathcal{T}$-refutable, there is an $r \in \mathcal{R}$ that contains a tableau step $\langle A, A_1, \ldots, A_n \rangle$ where $A_1, \ldots, A_n$ are $\mathcal{F}$-branches that are $\mathcal{T}$-refutable. We know, $\mathcal{F} \subseteq \mathcal{F}'$ therefore $A, A_1, \ldots, A_n$ are also $\mathcal{F}'$-branches. Moreover, $\mathcal{R} \subseteq \mathcal{R}'$ therefore $r \in \mathcal{R}'$. By induction hypothesis, if $A_1, \ldots, A_n$ are $\mathcal{T}$-refutable then they are also $\mathcal{T}'$-refutable. □

3.2 Our Tableau Rules

Figure 3.1 presents all the tableau rules we will consider throughout this thesis and Figure 3.2 presents them in a tableau view. The tableau rules Closed$\bot$, Closed$\neg\top$, Closed, Closed $\neq$, and ClosedSym have no alternatives. The Cut rule is special in that its head has no restriction. This means that it can be applied any time.

There are rules for each of the propositional logical constants and quantifiers. Namely, DNeg for eliminating of a double negation. For each other logical constant a positive and a negative rule is presented, for example, Imp and NegImp for implication, And and NegAnd for conjunction, and so on.

In addition, rules for handling equality at all types are presented. The rules for Boolean equality and Boolean extensionality, called Bool= and BoolExt respectively, handle equality at base type o.

The functional equality and functional extensionality rules (Func=, FuncExt) deal with equality at function types.

The Leibniz equality rule (Leibniz) is used to handle equality without introducing the concept of extensionality. The reason we include this rule is to enable us to consider non-extensional tableau calculi which include equality in their fragment.

The mating, decomposition and confrontation rules (Mat, Dec, Con) are included to enable us to consider complete cut free tableau calculi (see [7, 8, 6]).

To prove that a formula is valid using tableaux, first we negate the formula, then we construct a tableau proof by repeatedly applying the tableau rules until all branches are closed.

3.3 Examples

Now that we defined all the basic definitions regarding tableaux and presented the tableau rules we consider, we illustrate how a tableau system could be used by giving some examples.

Example 3.3.1 (Peirce's Law)**.** *Peirce's law* is simply the following formula

$$((p \to q) \to p) \to p$$

where $p,\ q : o$.

To prove that this formula is valid, a tableau refutation is constructed for its negation. We construct a tableau refutation using the tableau calculus containing the rules Imp, NegImp, and Closed.

First, NegImp rule is applied to $\neg(((p \to q) \to p) \to p)$ since the outermost logical constants in this formula are negation and implication.

$$\begin{array}{c} \neg(((p \to q) \to p) \to p) \\ (p \to q) \to p \\ \neg p \end{array}$$

Then, Imp rule is applied to $(p \to q) \to p$ so we get,

$$\begin{array}{c|c} \neg(((p \to q) \to p) \to p) & \\ (p \to q) \to p & \\ \neg p & \\ \hline \neg(p \to q) & p \end{array}$$

The Closed rule is applied to the right branch of the tableau since it contains both p and $\neg p$. Now, the right branch is closed.

NegImp rule is applied to the formula $\neg(p \to q)$ resulting in a p on this branch also, so this branch is similarly closed.

$$\begin{array}{c|c} \neg(((p \to q) \to p) \to p) & \\ (p \to q) \to p & \\ \neg p & \\ \hline \neg(p \to q) & p \\ p & \\ \neg q & \end{array}$$

This results in a full tableau refutation of the formula's negation, since all branches are closed. We can conclude that Peirce's Law is valid.

Example 3.3.2 (Surjective Cantor Theorem)**.** The surjective Cantor theorem states that

$$\neg\exists^{\iota\iota o} g.\forall^{\iota o} f.\exists^{\iota} u.gu = f$$

The following is a tableau refutation of the negation of Cantor's theorem, using the tableau calculus containing the tableau rules DNeg, Exists, Forall, Func=, Closed and Bool=.

$$\begin{array}{c|c} \neg\neg\exists g.\forall f.\exists u.gu = f & \\ \exists g.\forall f.\exists u.gu = f & \\ \forall f.\exists u.gu = f & \\ \exists u.gu = \lambda x.\neg gxx & \\ gd = \lambda x.\neg gxx & \\ gdd = \neg gdd & \\ \hline gdd & \neg gdd \\ \neg gdd & \neg\neg gdd \end{array}$$

$$\mathsf{Closed}_{\bot}\ \frac{}{A, \bot \vdash_{\mathcal{T}} \bot} \qquad \mathsf{Closed}_{\neg\top}\ \frac{}{A, \neg\top \vdash_{\mathcal{T}} \bot} \qquad \mathsf{Closed}\ \frac{}{A, s, \neg s \vdash_{\mathcal{T}} \bot}$$

$$\mathsf{Closed}_{\neq}\ \frac{}{A, s \neq s \vdash_{\mathcal{T}} \bot} \qquad \mathsf{ClosedSym}\ \frac{}{A, (s = t),\ (t \neq s) \vdash_{\mathcal{T}} \bot}$$

$$\mathsf{Cut}\ \frac{A, s \vdash_{\mathcal{T}} \bot \quad A, \neg s \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} \qquad \mathsf{Dneg}\ \frac{A, s \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} \neg\neg s \in A$$

$$\mathsf{And}\ \frac{A, s, t \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} (s \wedge t) \in A \qquad \mathsf{Or}\ \frac{A, s \vdash_{\mathcal{T}} \bot \quad A, t \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} (s \vee t) \in A$$

$$\mathsf{Imp}\ \frac{A, \neg s \vdash_{\mathcal{T}} \bot \quad A, t \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} (s \to t) \in A \qquad \mathsf{NegAnd}\ \frac{A, \neg s \vdash_{\mathcal{T}} \bot \quad A, \neg t \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} \neg(s \wedge t) \in A$$

$$\mathsf{NegOr}\ \frac{A, \neg s, \neg t \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} \neg(s \vee t) \in A \qquad \mathsf{NegImp}\ \frac{A, s, \neg t \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} \neg(s \to t) \in A$$

$$\mathsf{Forall}\ \frac{A, \lceil s\ t \rceil^{\beta} \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} \forall s \in A \qquad \mathsf{DeMorgan}_{\forall}\ \frac{A, \exists x. \neg s\ x \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} \neg\forall s \in A,\ x \notin FV(s)$$

$$\mathsf{Exists}\ \frac{A, \lceil s\ y \rceil^{\beta} \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} \exists s \in A,\ y \text{ is fresh}$$

$$\mathsf{DeMorgan}_{\exists}\ \frac{A, \forall x. \neg s\ x \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} \neg\exists s \in A,\ x \notin FV(s)$$

$$\mathsf{Bool}_{=}\ \frac{A, s, t \vdash_{\mathcal{T}} \bot \quad A, \neg s, \neg t \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} (s \equiv t) \in A$$

$$\mathsf{BoolExt}\ \frac{A, s, \neg t \vdash_{\mathcal{T}} \bot \quad A, t, \neg s \vdash_{\mathcal{T}} \bot}{} \neg(s \equiv t) \in A$$

$$\mathsf{Leibniz}\ \frac{A, (\forall p.\ p\ s \to p\ t) \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} (s =_{\sigma} t) \in A$$

$$\mathsf{Func}_{=}\ \frac{A, \lceil s_1\ t =_{\tau} s_2\ t \rceil^{\beta} \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} s_1 =_{\sigma\tau} s_2 \in A$$

$$\mathsf{FuncExt}\ \frac{A, \lceil s\ x \neq_{\tau} t\ x \rceil^{\beta} \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} s \neq_{\sigma\tau} t \in A,\ x \text{ is fresh}$$

$$\mathsf{Mat}\ \frac{A, s_1 \neq t_1 \vdash_{\mathcal{T}} \bot \quad \ldots \quad A, s_n \neq t_n \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} (x\ s_1 \ldots s_n), (\neg x\ t_1 \ldots t_n) \in A$$

$$\mathsf{Dec}\ \frac{A, s_1 \neq t_1 \vdash_{\mathcal{T}} \bot \quad \ldots \quad A, s_n \neq t_n \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} x\ s_1 \ldots s_n \neq_{\iota} x\ t_1 \ldots t_n \in A$$

$$\mathsf{Con}\ \frac{A, s_1 \neq s_2,\ t_1 \neq s_2 \vdash_{\mathcal{T}} \bot \quad A, s_1 \neq t_2,\ t_1 \neq t_2 \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot} s_1 =_{\iota} t_1,\ s_2 \neq_{\iota} t_2 \in A$$

Figure 3.1: Tableau rules we will consider

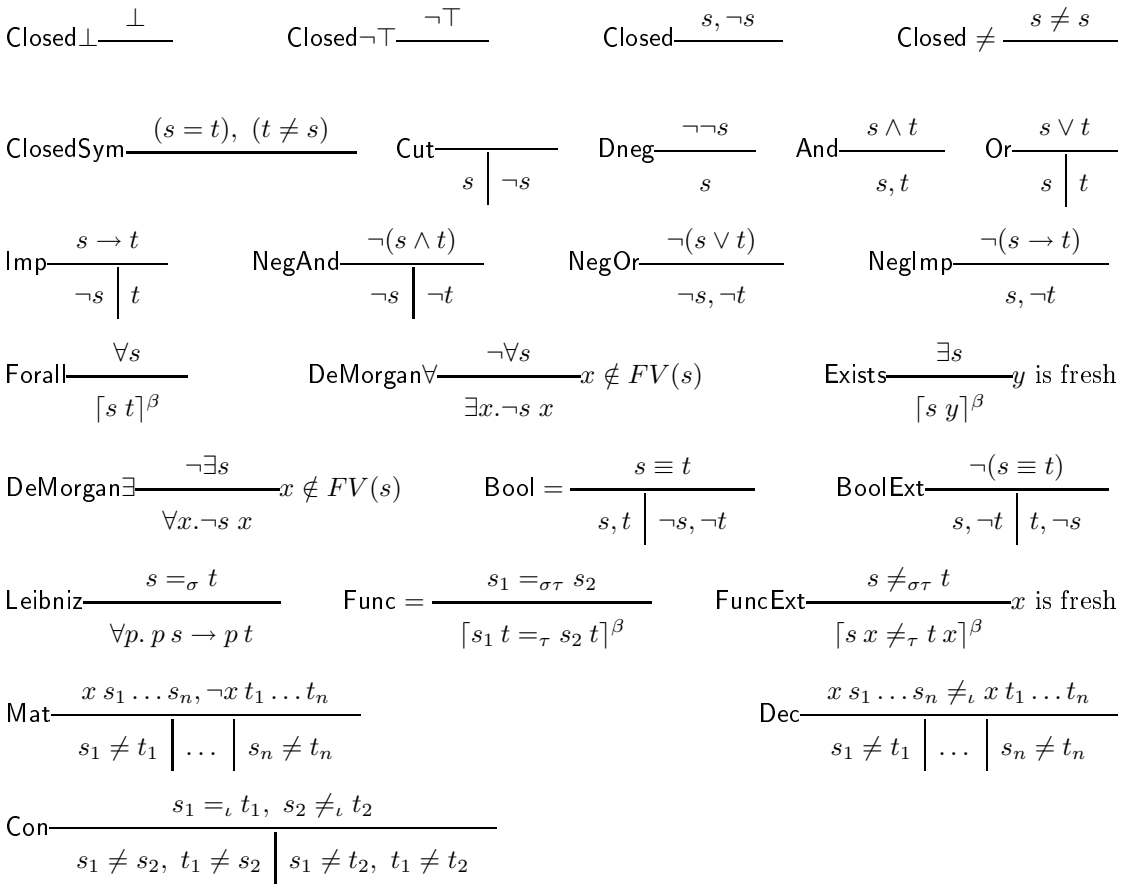

Figure 3.2: Tableau view of rules we will consider

Chapter 4

Natural Deduction Calculus

Natural deduction (ND) is a proof system introduced by Gentzen in 1935 [15]. Rules in this system formalize proof patterns that are used in common mathematical proofs. For this reason, the system is called *natural*.

Proof terms are syntactic objects which correspond to natural deduction proofs. A proof term has a formula as its type, namely the formula that it proves. This idea of proofs as terms and formulas as their types is called Curry-Howard isomorphism [17, 23, 29].

4.1 A Natural Deduction Calculus $\mathcal{N}$

We consider the ND calculus $\mathcal{N}$ which is intuitionistic and non-extensional. It uses terms that only contain the logical constants $\forall$ and $\to$.

Definition 4.1.1 (Natural Deduction Terms)**.** The set $\mathcal{N}$-terms of terms used in the Natural Deduction Calculus $\mathcal{N}$ is defined as

$$\mathcal{N}\text{-terms}(s,t,\dots) ::= x \mid s\ t \mid \lambda x.s \mid \forall^\sigma\ \lambda x.s \mid s \to t$$

where $x \in \mathcal{V}$ and $\sigma \in \mathbb{T}$.

We use $\bot_{\mathcal{N}}$as a short hand for $\forall^o\ \lambda p.p$.

Definition 4.1.2 (Context)**.** A *context* Γ is a subset of $\mathcal{N}$-terms.

Contexts is the set of all contexts.

Definition 4.1.3 (ND Rule)**.** An *ND Rule* is a tuple of pairs $\langle\langle\Gamma, s\rangle, \langle\Gamma_1, s_1\rangle, \dots, \langle\Gamma_n, s_n\rangle\rangle$ where $\Gamma, \Gamma_1 \dots, \Gamma_n$ are contexts and $s, s_1, \dots, s_n$ are β-normal formulas that are elements of $\mathcal{N}$-terms. An ND rule is presented as follows:

$$\frac{\Gamma_1 \vdash_{\mathcal{N}} s_1 \quad \dots \quad \Gamma_n \vdash_{\mathcal{N}} s_n}{\Gamma \vdash_{\mathcal{N}} s}$$

Figure 4.1 presents the rules in our ND calculus $\mathcal{N}$. Note that whenever we write $\Gamma \vdash_{\mathcal{N}} s$ we mean $\lceil\Gamma\rceil^\beta \vdash_{\mathcal{N}} \lceil s\rceil^\beta$.

Definition 4.1.4 (Derivable)**.** Let Γ be a context and s be a β-normal formula that is an element of $\mathcal{N}$-terms. The pair $\langle\Gamma, s\rangle$ is *derivable* if $s \in \Gamma$ or if there is a rule $\langle\langle\Gamma, s\rangle, \langle\Gamma_1, s_1\rangle, \dots, \langle\Gamma_n, s_n\rangle\rangle$ in the ND system $\mathcal{N}$ such that $\langle\Gamma_1, s_1\rangle, \dots, \langle\Gamma_n, s_n\rangle$ are derivable.

We write $\Gamma \vdash_{\mathcal{N}} s$ or say s is derivable in Γ to mean that the pair $\langle\Gamma, s\rangle$ is derivable. Moreover, we usually write $\vdash_{\mathcal{N}} s$ as a short hand for $\emptyset \vdash_{\mathcal{N}} s$.

Note that if for some context Γ and some formula s we are given a derivation of $\Gamma \vdash_{\mathcal{N}} s$ that uses the wk rule, we can construct a derivation of $\Gamma \vdash_{\mathcal{N}} s$ that does not use the wk rule. This basically follows from how the hy rule is stated. We only add the wk rule for convenience.

$$hy\ \frac{t \in \Gamma}{\Gamma \vdash_{\mathcal{N}} t} \qquad wk\ \frac{\Gamma' \vdash_{\mathcal{N}} t}{\Gamma \vdash_{\mathcal{N}} t}\ \Gamma' \subseteq \Gamma$$

$$\forall I\ \frac{\Gamma \vdash_{\mathcal{N}} t}{\Gamma \vdash_{\mathcal{N}} \forall x.t}\ x \notin FV^*(\Gamma) \qquad \to I\ \frac{\Gamma, s \vdash_{\mathcal{N}} t}{\Gamma \vdash_{\mathcal{N}} s \to t}$$

$$\forall E\ \frac{\Gamma \vdash_{\mathcal{N}} \forall x.s}{\Gamma \vdash_{\mathcal{N}} \lceil s[t/x] \rceil^{\beta}} \qquad \to E\ \frac{\Gamma \vdash_{\mathcal{N}} s \to t \quad \Gamma \vdash_{\mathcal{N}} s}{\Gamma \vdash_{\mathcal{N}} t}$$

Figure 4.1: Rules in our ND calculus $\mathcal{N}$

4.2 Coq

Coq is a formal proof management system that is based on the calculus of inductive constructions [5]. It provides a formal language called Gallina to write definitions and theorems, in addition it serves as an interactive theorem prover and a proof checker. We formulate many of our lemmas and their proofs in Gallina. The proofs correspond to proof terms which correspond to ND proofs. So any proof that is written in Coq could be directly translated to an ND proof and vice versa. We use Coq to check those proofs. This way we get an assurance that the proofs are correct.

We will now present a simple lemma that we use later on in some of our proofs. We provide both an ND proof of the lemma and present in Figure 4.2 the corresponding Coq lemma and Coq proof. This is done to provide an intuition about the correspondence of the proofs and to make the reader familiar with Coq proofs. For more information about the correspondance of ND proofs and proof terms please refer to [29].

Lemma 4.2.1. $\vdash_{\mathcal{N}} \forall^o p\,.\, p \to (p \to \bot_{\mathcal{N}}) \to \bot_{\mathcal{N}}$

Proof. By $\forall I$ and $\to I$ it is enough to prove $p, p \to \bot_{\mathcal{N}} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$. We know this by $\to E$. □

```
Definition simple_lemma : ∀ p : Prop, p → (p → (∀ q : Prop, q)) → (∀ q : Prop, q)
  := fun p u v ⇒ (v u).
```

Figure 4.2: Lemma 4.2.1 and its proof in Coq

Chapter 5

Logical Transformations

We first consider transformations Ψ that map terms to $\mathcal{N}$-terms. We would like the transformation to map tableau refutable formulas to ND-refutable $\mathcal{N}$-terms. Note that unsatisfiability of a formula s with free variables $x_1, \ldots, x_n$ is equivalent to unsatisfiability of the ground formula $\exists x_1, \ldots, x_n.s$. For this reason we will be satisfied if the transformation Ψ maps ground tableau refutable formulas to ND-refutable $\mathcal{N}$-terms. Since a tableau calculus operates on branches instead of formulas, we will also need branch transformations Ψ^* that maps branches to contexts. A branch Ψ^* extends a logical transformation Ψ if it agrees with Ψ on ground formulas.

5.1 Definitions

Definition 5.1.1 (Logical Transformation)**.** A *logical transformation* is a function $\Psi : \mathit{Ter} \to \mathcal{N}$-terms such that $\forall \mathcal{I} \in \mathit{Interp} : \forall s \in \mathcal{F} : \hat{\mathcal{I}}(\Psi(s)) = \hat{\mathcal{I}}(s)$.

Definition 5.1.2 (Compositional)**.** A logical transformation Ψ is *compositional* if it satisfies the following:

$$\Psi(x) = x \text{ for } x \in \mathcal{V}$$
$$\Psi(s\,t) = \Psi(s)\ \Psi(t)$$
$$\Psi(\lambda x.s) = \lambda x.\Psi(s)$$
$$FV(\Psi(c)) = \emptyset \text{ for all logical constants } c$$

Definition 5.1.3 (Beta)**.** A logical transformation Ψ respects *beta* if

$$\forall s,\, t \in \mathit{Ter} : (s \sim_\beta t) \implies (\Psi(s) \sim_\beta \Psi(t)).$$

Proposition 5.1.4. If a logical transformation is compositional then it respects beta.

Proof. This follows from the definition of a logical transformation being compositional. □

Definition 5.1.5 (Branch Transformation)**.** A *branch transformation* is a function

$$\Psi^* : \mathit{Branches} \to \mathit{Contexts}.$$

Definition 5.1.6 (Extends)**.** Let Ψ be a logical transformation. A branch transformation Ψ^* *extends* Ψ if for all ground branches A we have $\Psi^*(A) = \{\Psi(s) | s \in A\}$. We say Ψ^* *trivially extends* Ψ, if for all branches A we have $\Psi^*(A) = \{\Psi(s) | s \in A\}$.

Definition 5.1.7 (Branch Transformation Respects a Step)**.** Let Ψ^* be a branch transformation and $A, A_1, \ldots, A_n$ be branches. We say Ψ^* *respects a step* $\langle A, A_1, \ldots, A_n \rangle$ if the following holds:

$$\text{if } \Psi^*(A_1) \vdash_{\mathcal{N}} \bot_{\mathcal{N}} \quad \ldots \quad \Psi^*(A_n) \vdash_{\mathcal{N}} \bot_{\mathcal{N}} \text{ then } \Psi^*(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$$

A *branch transformation respects a rule* if it respects all the steps in the rule. Moreover, a *branch transformation respects a tableau calculus* if it respects all the rules in the calculus. A *logical transformation* Ψ *respects a tableau calculus* $\mathcal{T}$ if there exists a branch transformation Ψ^* which extends Ψ and Ψ^* respects $\mathcal{T}$.

5.2 Translation

Theorem 5.2.1 (Translation)**.** Let $\mathcal{T} = \langle \mathcal{F}, \mathcal{R} \rangle$ be a tableau calculus, Ψ^* be a branch transformation that respects $\mathcal{T}$, and A be an $\mathcal{F}$-branch. If A is $\mathcal{T}$-refutable, then $\Psi^*(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$.

Proof. We know A is $\mathcal{T}$-refutable. By definition of $\mathcal{T}$-refutable, there exists an $r \in \mathcal{R}$ that contains a tableau step $\langle A, A_1, \ldots, A_n \rangle$ where $A_1, \ldots, A_n$ are $\mathcal{F}$-branches that are $\mathcal{T}$-refutable. Suppose A is $\mathcal{T}$-refutable. Then, there is an $r \in \mathcal{R}$ that contains a tableau step $\langle A, A_1, \ldots, A_n \rangle$ and $A_1, \ldots, A_n$ are $\mathcal{T}$-refutable. By induction hypothesis, $\Psi^*(A_1) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}, \ldots, \Psi^*(A_n) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$. We want to show that $\Psi^*(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$. We know that Ψ^* respects $\mathcal{T}$. Hence, $\Psi^*(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$. □

Corollary 5.2.2. Let $\mathcal{T} = \langle \mathcal{F}, \mathcal{R} \rangle$ be a tableau calculus, Ψ be a logical transformation that respects $\mathcal{T}$, and s be a ground formula in $\mathcal{F}$. If $\{s\}$ is $\mathcal{T}$-refutable, then $\{\Psi(s)\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$.

Proof. We know that Ψ respects $\mathcal{T}$. Therefore, there exists a branch transformation Ψ^* which extends Ψ and Ψ^* respects $\mathcal{T}$. By Theorem 5.2.1, we know $\Psi^*(\{s\}) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$. Since s is ground therefore, $\Psi^*(\{s\}) = \{\Psi(s)\}$. Hence, $\{\Psi(s)\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$. □

Chapter 6

The Girard Transformation

In this chapter we give the *Girard transformation* $\Psi_{\mathcal{G}}$ that maps all logical constants to terms containing only the logical constants $\forall$ and $\rightarrow$. This is used to translate general terms *Ter* to $\mathcal{N}$-terms. The definition of $\Psi_{\mathcal{G}}$ is based on Girard's definitions given in [18].

6.1 The Girard Transformation $\Psi_{\mathcal{G}}$

The Girard transformation $\Psi_{\mathcal{G}}$ transforms terms as follows:

$$
\begin{array}{l}
\Psi_{\mathcal{G}}(x) = x \text{ for } x \in \mathcal{V} \\
\Psi_{\mathcal{G}}(s\,t) = \Psi_{\mathcal{G}}(s)\ \Psi_{\mathcal{G}}(t) \\
\Psi_{\mathcal{G}}(\lambda x.s) = \lambda x.\Psi_{\mathcal{G}}(s) \\
\Psi_{\mathcal{G}}(\rightarrow) = \lambda x.\lambda y.x \rightarrow y \\
\Psi_{\mathcal{G}}(\forall) = \lambda f.\forall x.f\ x \\
\Psi_{\mathcal{G}}(=) = \lambda x.\lambda y.\forall p.p\ x \rightarrow p\ y \text{ (Leibniz Equality)} \\
\Psi_{\mathcal{G}}(\bot) = \forall^o p.p \\
\Psi_{\mathcal{G}}(\top) = \forall^o p.p \rightarrow p \\
\Psi_{\mathcal{G}}(\neg) = \lambda x.x \rightarrow \forall^o p.p \\
\Psi_{\mathcal{G}}(\wedge) = \lambda x.\lambda y.\forall p.(x \rightarrow y \rightarrow p) \rightarrow p \\
\Psi_{\mathcal{G}}(\vee) = \lambda x.\lambda y.\forall p.(x \rightarrow p) \rightarrow (y \rightarrow p) \rightarrow p \\
\Psi_{\mathcal{G}}(\exists) = \lambda f.\forall p.(\forall x.f\ x \rightarrow p) \rightarrow p
\end{array}
$$

We write $\neg_{\mathcal{G}}$ and $\bot_{\mathcal{G}}$ as shorthand for $\Psi_{\mathcal{G}}(\neg)$ and $\Psi_{\mathcal{G}}(\bot)$ respectively. We do the same for all other logical constants.

Proposition 6.1.1. The Girard transformation $\Psi_{\mathcal{G}}$ is a logical transformation.

Proof. Follows from the obvious fact that this transformation is meaning preserving. □

Proposition 6.1.2. The Girard transformation $\Psi_{\mathcal{G}}$ respects beta.

Proof. Follows from Proposition 5.1.4 since $\Psi_{\mathcal{G}}$ is compositional. □

Let $\Psi^*_{\mathcal{G}}$ be the branch transformation that trivially extends $\Psi_{\mathcal{G}}$.

Before considering the tableau calculus containing all the tableau rules and the full fragment, we will consider two smaller tableau calculi. The first tableau calculus we consider is $\mathcal{T}_{\rightarrow,\bot}$. The second is $\mathcal{T}_{sec}$. In the following two sections we define these tableau calculi and state whether or not the Girard transformation $\Psi_{\mathcal{G}}$ respects them.

6.2 The Tableau Calculus $\mathcal{T}_{\rightarrow,\bot}$

Let $\mathcal{F}_{\rightarrow,\bot}$ [1] be the fragment of terms containing only the logical constants $\bot, \neg,$ and $\rightarrow$ and $\mathcal{R}_{\rightarrow,\bot}$ be the following set of rules:

$$\{\mathsf{Closed}\bot,\ \mathsf{Closed},\ \mathsf{DNeg},\ \mathsf{Imp},\ \mathsf{NegImp},\ \mathsf{Cut}\}$$

The tableau calculus $\mathcal{T}_{\rightarrow,\bot}$ is the pair $\langle \mathcal{F}_{\rightarrow,\bot}, \mathcal{R}_{\rightarrow,\bot} \rangle$.

Lemma 6.2.1 (NegImp). $\vdash_{\mathcal{N}} \forall^o p\ q.(p \rightarrow (\neg_{\mathcal{G}} q) \rightarrow \bot_{\mathcal{N}}) \rightarrow (\neg_{\mathcal{G}}(p \rightarrow q)) \rightarrow \bot_{\mathcal{N}}$

Proof. Figure 6.1 presents a derivation of $\vdash_{\mathcal{N}} \forall p\ q.(p \rightarrow (\neg_{\mathcal{G}} q) \rightarrow \bot_{\mathcal{N}}) \rightarrow (\neg_{\mathcal{G}}(p \rightarrow q)) \rightarrow \bot_{\mathcal{N}}$. □

Lemma 6.2.2. $\Psi^*_{\mathcal{G}}$ respects the rule NegImp.

Proof. We need to prove that for any branch A and for any two formulas p and q such that $\neg(p \rightarrow q) \in A$ if $\Psi^*_{\mathcal{G}}(A), \Psi_{\mathcal{G}}(p), \Psi_{\mathcal{G}}(\neg q) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$ then $\Psi^*_{\mathcal{G}}(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$

A proof is presented in Figure 6.2. Note that we implicitly make use of the fact that $\Psi_{\mathcal{G}}(\neg(p \rightarrow q))$ is equivalent to $\neg_{\mathcal{G}} \Psi_{\mathcal{G}}(p \rightarrow q)$ and to $\neg_{\mathcal{G}}(\Psi_{\mathcal{G}}(p) \rightarrow \Psi_{\mathcal{G}}(q))$. In addition, this derivation depends on the transformation of the logical constants $\neg$ and $\rightarrow$ which are the ones used in the NegImp rule. □

Lemma 6.2.3 (Closed$\bot$). $\vdash_{\mathcal{N}} \bot_{\mathcal{G}} \rightarrow \bot_{\mathcal{N}}$

Lemma 6.2.4 (Closed). $\vdash_{\mathcal{N}} \forall^o p.p \rightarrow (\neg_{\mathcal{G}} p) \rightarrow \bot_{\mathcal{N}}$

Lemma 6.2.5 (DNeg). $\vdash_{\mathcal{N}} \forall^o p.(p \rightarrow \bot_{\mathcal{G}}) \rightarrow (\neg_{\mathcal{G}} \neg_{\mathcal{G}} p) \rightarrow \bot_{\mathcal{N}}$

Lemma 6.2.6 (Imp). $\vdash_{\mathcal{N}} \forall^o p\ q.((\neg_{\mathcal{G}} p) \rightarrow \bot_{\mathcal{N}}) \rightarrow (q \rightarrow \bot_{\mathcal{N}}) \rightarrow (p \rightarrow q) \rightarrow \bot_{\mathcal{N}}$

Lemma 6.2.7 (Cut). $\vdash_{\mathcal{N}} \forall^o p.(p \rightarrow \bot_{\mathcal{N}}) \rightarrow ((\neg_{\mathcal{G}} p) \rightarrow \bot_{\mathcal{N}}) \rightarrow \bot_{\mathcal{N}}$

Lemma 6.2.8. $\Psi^*_{\mathcal{G}}$ respects all of the rules in $\mathcal{T}_{\rightarrow,\bot}$.

Proof. In Appendix A, we provide a Coq proof of Lemmas 6.2.1, 6.2.3, 6.2.4, 6.2.5, 6.2.6, and 6.2.7 which correspond to each of the rules in $\mathcal{R}_{\rightarrow,\bot}$. Using those lemmas to prove that $\Psi^*_{\mathcal{G}}$ respects the lemma's corresponding rule is straightforward. We presented a proof that $\Psi^*_{\mathcal{G}}$ respects the rule NegImp by proving Lemma 6.2.2 using Lemma 6.2.1. The rest of the cases are very similar. Thus, this step from now on will be left for the reader. □

Theorem 6.2.9. The Girard transformation $\Psi_{\mathcal{G}}$ respects $\mathcal{T}_{\rightarrow,\bot}$.

Proof. This follows directly from Lemma 6.2.8. □

Corollary 6.2.10. Let s be a ground formula. If $\{s\}$ is $\mathcal{T}_{\rightarrow,\bot}$-refutable, then $\{\Psi_{\mathcal{G}}(s)\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$.

Proof. The proof follows directly from Corollary 5.2.2 and from Theorem 6.2.9 □

[1] Note that we call this fragment $\mathcal{F}_{\rightarrow,\bot}$ because it includes the preimage of terms which only contain $\rightarrow$ and $\bot_{\mathcal{G}}$ under $\Psi_{\mathcal{G}}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}}(p \to q)}\,hy \quad \dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, p \vdash_{\mathcal{N}} p \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{G}}}\,hy \quad \dfrac{}{\Gamma, p \vdash_{\mathcal{N}} p}\,hy}{\Gamma, p \vdash_{\mathcal{N}} (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{G}}}\,{\to E} \quad \dfrac{\dfrac{\dfrac{}{\Gamma, p, q \vdash_{\mathcal{N}} \neg_{\mathcal{G}}(p \to q)}\,hy \quad \dfrac{\dfrac{}{\Gamma, p, q \vdash_{\mathcal{N}} q}\,hy}{\Gamma, p, q \vdash_{\mathcal{N}} p \to q}\,{\to I}}{\Gamma, p, q \vdash_{\mathcal{N}} \bot_{\mathcal{G}}}\,{\to E}}{\Gamma, p \vdash_{\mathcal{N}} (\neg_{\mathcal{G}} q)}\,{\to I}}{\Gamma, p \vdash_{\mathcal{N}} \bot_{\mathcal{G}}}\,{\to E}}{\Gamma, p \vdash_{\mathcal{N}} q}\,{\forall E}}{\Gamma \vdash_{\mathcal{N}} p \to q}\,{\to I}}{(p \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}), (\neg_{\mathcal{G}}(p \to q)) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}}\,{\to E}}{(p \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}) \vdash_{\mathcal{N}} (\neg_{\mathcal{G}}(p \to q)) \to \bot_{\mathcal{N}}}\,{\to I}}{\vdash_{\mathcal{N}} (p \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(p \to q)) \to \bot_{\mathcal{N}}}\,{\to I}}{\vdash_{\mathcal{N}} \forall q.(p \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(p \to q)) \to \bot_{\mathcal{N}}}\,{\forall I}}{\vdash_{\mathcal{N}} \forall p\ q.(p \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(p \to q)) \to \bot_{\mathcal{N}}}\,{\forall I}$$

where $\Gamma = \{(p \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}), (\neg_{\mathcal{G}}(p \to q))\}$.

Figure 6.1: Derivation of proposition corresponding to NegImp translated using $\Psi_{\mathcal{G}}$

Lemma 6.2.1

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\vdash_{\mathcal{N}} \forall t_1 t_2.s_2 \to \neg_{\mathcal{G}}(t_1 \to t_2) \to \bot_{\mathcal{N}}}{\vdash_{\mathcal{N}} \forall t_1.s_1 \to \neg_{\mathcal{G}}(t_1 \to \Psi_{\mathcal{G}}(q)) \to \bot_{\mathcal{N}}}\,{\scriptstyle \forall E}}{\vdash_{\mathcal{N}} s \to \neg_{\mathcal{G}}(\Psi_{\mathcal{G}}(p) \to \Psi_{\mathcal{G}}(q)) \to \bot_{\mathcal{N}}}\,{\scriptstyle \forall E}}{\Psi^*_{\mathcal{G}}(A) \vdash_{\mathcal{N}} s \to \neg_{\mathcal{G}}(\Psi_{\mathcal{G}}(p) \to \Psi_{\mathcal{G}}(q)) \to \bot_{\mathcal{N}}}\,{\scriptstyle wk} \qquad \dfrac{\dfrac{\Psi^*_{\mathcal{G}}(A), \Psi_{\mathcal{G}}(p), \neg_{\mathcal{G}}\Psi_{\mathcal{G}}(q) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}}{\Psi^*_{\mathcal{G}}(A), \Psi_{\mathcal{G}}(p) \vdash_{\mathcal{N}} \neg_{\mathcal{G}}\Psi_{\mathcal{G}}(q) \to \bot_{\mathcal{N}}}\,{\scriptstyle \to I}}{\Psi^*_{\mathcal{G}}(A) \vdash_{\mathcal{N}} s}\,{\scriptstyle \to I}}{\Psi^*_{\mathcal{G}}(A) \vdash_{\mathcal{N}} \neg_{\mathcal{G}}(\Psi_{\mathcal{G}}(p) \to \Psi_{\mathcal{G}}(q)) \to \bot_{\mathcal{N}}}\,{\scriptstyle \to E} \qquad \dfrac{}{\Psi^*_{\mathcal{G}}(A) \vdash_{\mathcal{N}} \neg_{\mathcal{G}}(\Psi_{\mathcal{G}}(p) \to \Psi_{\mathcal{G}}(q))}\,{\scriptstyle hy}}{\Psi^*_{\mathcal{G}}(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}}\,{\scriptstyle \to E}$$

where $s = (\Psi_{\mathcal{G}}(p) \to \neg_{\mathcal{G}}\Psi_{\mathcal{G}}(q) \to \bot_{\mathcal{N}})$,

$s_1 = (t_1 \to \neg_{\mathcal{G}}\Psi_{\mathcal{G}}(q) \to \bot_{\mathcal{N}})$, and

$s_2 = (t_1 \to t_2 \to \bot_{\mathcal{N}})$.

Figure 6.2: $\Psi^*_{\mathcal{G}}$ respects NegImp

6.3 The Tableau Calculus $\mathcal{T}_{sec}$

Let $\mathcal{F}_{sec}$ be the fragment of terms containing the logical constants $\top, \bot, \neg, \rightarrow, \wedge, \vee, \forall^{\iota}, \forall^{\iota o}, \exists^{\iota}$, and $\exists^{\iota o}$ and $\mathcal{R}_{sec}$ be the following set of rules:

$$\{\mathsf{Closed}\bot,\ \mathsf{Closed}\neg\top,\ \mathsf{Closed},\ \mathsf{DNeg},\ \mathsf{And},\ \mathsf{NegAnd},\ \mathsf{Or},\ \mathsf{NegOr},$$

$$\mathsf{Imp},\ \mathsf{NegImp},\ \mathsf{DeMorgan}\forall,\ \mathsf{DeMorgan}\exists,\ \mathsf{Forall},\ \mathsf{Exists},\ \mathsf{Cut}\}$$

The tableau calculus $\mathcal{T}_{sec}$ is the pair $\langle \mathcal{F}_{sec}, \mathcal{R}_{sec} \rangle$. This tableau calculus corresponds to second-order logic.

6.3.1 $\Psi_{\mathcal{G}}$ does not Respect $\mathcal{T}_{sec}$

Definition 6.3.1 (Double Negation Shift)**.** The *double negation shift*, DNS, is the formula:

$$(\forall x.\neg\neg f\ x) \rightarrow \neg\neg\forall x.f\ x$$

Lemma 6.3.2. $\nvdash_{\mathcal{N}} (\forall x.\neg_{\mathcal{G}}\neg_{\mathcal{G}} f\ x) \rightarrow \neg_{\mathcal{G}}\neg_{\mathcal{G}}\forall x.f\ x$

Proof. It is known that DNS is not provable intuitionistically [16, 12, 31]. If we transform the negations in this formula using the intuitionistic Girard transformation the result is also not provable intuitionistically, and thus cannot be derived using our intutionistic ND system $\mathcal{N}$. □

Lemma 6.3.3. $\vdash_{\mathcal{N}} (\Psi_{\mathcal{G}}(\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\,x)\wedge\neg\exists^{\iota} x.\neg f\,x)) \rightarrow \bot_{\mathcal{N}}) \rightarrow (\forall x.\neg_{\mathcal{G}}\neg_{\mathcal{G}} f\,x) \rightarrow \neg_{\mathcal{G}}\neg_{\mathcal{G}}\forall x.f\,x$

Proof. The Coq proof of this lemma is provided in Appendix A. □

Lemma 6.3.4. $\{\Psi_{\mathcal{G}}(\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\ x) \wedge \neg\exists^{\iota} x.\neg f\ x))\} \nvdash_{\mathcal{N}} \bot_{\mathcal{N}}$

Proof. Assume $\{\Psi_{\mathcal{G}}(\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\ x) \wedge \neg\exists^{\iota} x.\neg f\ x))\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$. By the $\rightarrow I$ rule we know $\vdash_{\mathcal{N}} \Psi_{\mathcal{G}}(\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\ x) \wedge \neg\exists^{\iota} x.\neg f\ x)) \rightarrow \bot_{\mathcal{N}}$. Using Lemma 6.3.3 and the $\rightarrow E$ rule, we know $\vdash_{\mathcal{N}} (\forall x.\neg_{\mathcal{G}}\neg_{\mathcal{G}} f\ x) \rightarrow \neg_{\mathcal{G}}\neg_{\mathcal{G}}\forall x.f\ x$. This leads to a contradiction with Lemma 6.3.2. Hence, $\{\Psi_{\mathcal{G}}(\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\ x) \wedge \neg\exists^{\iota} x.\neg f\ x))\} \nvdash_{\mathcal{N}} \bot_{\mathcal{N}}$. □

Theorem 6.3.5. The Girard transformation $\Psi_{\mathcal{G}}$ does not respect $\mathcal{T}_{sec}$.

Proof. Consider the following formula s:

$$\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\ x) \wedge \neg\exists^{\iota} x.\neg f\ x)$$

The $\mathcal{F}_{sec}$-branch $\{\neg\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\ x) \wedge \neg\exists^{\iota} x.\neg f\ x)\}$ is $\mathcal{T}_{sec}$-refutable. A tableau proof that only uses rules from $\mathcal{R}_{sec}$ is given in Figure 6.3. By Lemma 6.3.4 we know that $\{\Psi_{\mathcal{G}}(\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f x)\wedge \neg\exists^{\iota} x.\neg f\ x))\} \nvdash_{\mathcal{N}} \bot_{\mathcal{N}}$. Since $s \in \mathcal{F}_{sec}$ and $FV(s) = \emptyset$, therefore using Corollary 5.2.2 it follows that the logical transformation $\Psi_{\mathcal{G}}$ does not respect $\mathcal{T}_{sec}$. □

If $\Psi_{\mathcal{G}}$ does not respect $\mathcal{T}_{sec}$ then it also does not respect the tableau calculus containing all the rules and the full fragment. Thus we need to create a modified transformation that respects $\mathcal{T}_{sec}$. In order to find this modified transformation, it is helpful to know which tableau rules $\Psi^{*}_{\mathcal{G}}$ respects and which rules it does not respect.

$$\neg\forall f.\neg((\neg\forall x.f\ x) \wedge \neg\exists x.\neg f\ x)$$
$$\exists f.\neg\neg((\neg\forall x.f\ x) \wedge \neg\exists x.\neg f\ x)$$
$$\neg\neg((\neg\forall x.f\ x) \wedge \neg\exists x.\neg f\ x)$$
$$(\neg\forall x.f\ x) \wedge \neg\exists x.\neg f\ x$$
$$\neg\forall x.f\ x$$
$$\neg\exists x.\neg f\ x$$
$$\exists x.\neg f\ x$$

Figure 6.3: Tableau Proof of $\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.p\ x) \wedge \neg\exists^{\iota} x.\neg p\ x)$

6.3.2 Tableau Rules in $\mathcal{T}_{sec}$ that are Respected by $\Psi^*_{\mathcal{G}}$

By Lemma 6.2.8 we know that $\Psi^*_{\mathcal{G}}$ respects the following rules:

Closed$\bot$, Closed, DNeg, Imp, NegImp, Cut

Lemma 6.3.6 (Closed$\neg\top$)**.** $\vdash_{\mathcal{N}} (\neg_{\mathcal{G}} \top_{\mathcal{G}}) \to \bot_{\mathcal{N}}$

Lemma 6.3.7 (And)**.** $\vdash_{\mathcal{N}} \forall p\, q.(p \to q \to \bot_{\mathcal{N}}) \to (p \wedge_{\mathcal{G}} q) \to \bot_{\mathcal{N}}$

Lemma 6.3.8 (NegAnd)**.** $\vdash_{\mathcal{N}} \forall p\, q.((\neg_{\mathcal{G}} p) \to \bot_{\mathcal{N}}) \to ((\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}) \to \neg_{\mathcal{G}}(p \wedge_{\mathcal{G}} q) \to \bot_{\mathcal{N}}$

Lemma 6.3.9 (Or)**.** $\vdash_{\mathcal{N}} \forall p\, q.(p \to \bot_{\mathcal{N}}) \to (q \to \bot_{\mathcal{N}}) \to (p \vee_{\mathcal{G}} q) \to \bot_{\mathcal{N}}$

Lemma 6.3.10 (NegOr)**.** $\vdash_{\mathcal{N}} \forall p\, q.((\neg_{\mathcal{G}} p) \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}) \to \neg_{\mathcal{G}}(p \vee_{\mathcal{G}} q) \to \bot_{\mathcal{N}}$

Lemma 6.3.11 (Forall)**.** $\vdash_{\mathcal{N}} \forall^{\sigma o} f.\forall^{\sigma} t.(f\ t \to \bot_{\mathcal{N}}) \to (\forall^{\sigma}_{\mathcal{G}} f) \to \bot_{\mathcal{N}}$

Lemma 6.3.12 (Exists)**.** $\vdash_{\mathcal{N}} \forall^{\sigma o} f.(\forall^{\sigma} x.f\ x \to \bot_{\mathcal{N}}) \to (\exists^{\sigma}_{\mathcal{G}} f) \to \bot_{\mathcal{N}}$

Lemma 6.3.13 (DeMorgan$\exists$)**.** $\vdash_{\mathcal{N}} \forall^{\sigma \to o} f.((\forall^{\sigma}_{\mathcal{G}}(\lambda x.\neg_{\mathcal{G}} f\ x)) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(\exists^{\sigma}_{\mathcal{G}} f)) \to \bot_{\mathcal{N}}$

Theorem 6.3.14. The branch transformation $\Psi^*_{\mathcal{G}}$ respects the following rules:

Closed$\bot$, Closed$\neg\top$, Closed, DNeg, And, NegAnd, Or,

NegOr, Imp, NegImp, DeMorgan$\exists$, Forall, Exists, Cut

Proof. Follows from Lemmas 6.2.3, 6.3.6, 6.2.4 ,6.2.5, 6.3.7, 6.3.8, 6.3.9, 6.3.10, 6.2.6, 6.2.2, 6.3.13, 6.3.11, 6.3.12, and 6.2.7 respectively. □

6.3.3 Tableau Rules in $\mathcal{T}_{sec}$ that are not Respected by $\Psi^*_{\mathcal{G}}$

Theorem 6.3.15. The branch transformation $\Psi^*_{\mathcal{G}}$ does not respect the DeMorgan$\forall$ rule.

Proof. Let $\mathcal{T}$ be a tableau calculus containing exactly the following rules:

DeMorgan$\forall$, Exists, DNeg, And, Closed

The tableau refutation of the branch $\{\neg\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\ x) \wedge \neg\exists^{\iota} x.\neg f\ x)\}$ given in Figure 6.3 uses only the rules in $\mathcal{T}$. Therefore, this branch is $\mathcal{T}$-refutable. Following the same method used to prove Theorem 6.3.5 we conclude that the logical transformation $\Psi_{\mathcal{G}}$ does not respect $\mathcal{T}$. We know that $\Psi^*_{\mathcal{G}}$ respects the rules Exists, DNeg, And, and Closed using Theorem 6.3.14. Therefore, $\Psi^*_{\mathcal{G}}$ does not respect the DeMorgan$\forall$ rule. □

6.4 The Tableau Calculus $\mathcal{T}_{Full}$

$\mathcal{T}_{Full}$ is the tableau calculus containing all the tableau rules presented in Figure 3.1 except for the Leibniz rule. Moreover, it includes the full fragment of terms. The reason why we exclude the Leibniz rule from $\mathcal{T}_{Full}$ but still include the Cut rule (even though it is not needed for completeness) is that we aim at obtaining the richest complete tableau calculus for which we can find a logical transformation. We easily could handle Cut, but not Leibniz. It is known that Leibniz equality does not follow from Boolean extensionality. Therefore we argue that $\Psi_{\mathcal{GKP}}$ does not respect the Leibniz rule at type o. We already know from Lemma 6.3.15 that $\Psi^*_{\mathcal{G}}$ does not respect the DeMorgan$\forall$ rule. Thus, $\Psi_{\mathcal{G}}$ does not respect the full tableau calculus. However, we still check which rules in this full calculus are respected by $\Psi_{\mathcal{G}}$. This is done to determine which rules are not respected by this transformation, in order to get an intuition on how to modify the transformation such that it respects those rules.

Proposition 6.4.1. The Tableau Calculus $\mathcal{T}_{Full}$ is complete.

Proof. This follows from the main result in [6]. □

6.4.1 $\Psi_{\mathcal{G}}$ does not Respect $\mathcal{T}_{Full}$

Theorem 6.4.2. The Girard transformation $\Psi_{\mathcal{G}}$ does not respect $\mathcal{T}_{Full}$.

Proof. We know $\mathcal{F}_{sec} \subseteq \mathcal{F}_{Full}$, $\mathcal{R}_{sec} \subseteq \mathcal{R}_{Full}$, and that the branch $\{\neg\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\ x) \wedge \neg\exists^{\iota} x.\neg f\ x)\}$ is $\mathcal{T}_{sec}$-refutable. Therefore by Theorem 3.1.8 we know that $\{\neg\forall^{\iota o} f.\neg((\neg\forall^{\iota} x.f\ x) \wedge \neg\exists^{\iota} x.\neg f\ x)\}$ is $\mathcal{T}_{Full}$-refutable. Following the same method used to prove Theorem 6.3.5 we conclude that the logical transformation $\Psi_{\mathcal{G}}$ does not respect $\mathcal{T}_{Full}$. □

6.4.2 Tableau Rules in $\mathcal{T}_{Full}$ that are Respected by $\Psi^*_{\mathcal{G}}$

Using Theorem 6.3.14 we know that the branch transformation $\Psi^*_{\mathcal{G}}$ respects the following rules:

Closed$\bot$, Closed$\neg\top$, Closed, DNeg, And, NegAnd, Or,

NegOr, Imp, NegImp, DeMorgan$\exists$, Forall, Exists, Cut

Lemma 6.4.3 (Closed$\neq$). $\vdash_{\mathcal{N}} \forall^{\sigma} s.\neg_{\mathcal{G}}(s =_{\mathcal{G}} s) \to \bot_{\mathcal{N}}$

Lemma 6.4.4 (ClosedSym). $\vdash_{\mathcal{N}} \forall^{\sigma} s\ t.(s =_{\mathcal{G}} t) \to \neg_{\mathcal{G}}(t =_{\mathcal{G}} s) \to \bot_{\mathcal{N}}$

Lemma 6.4.5 (Bool=). $\vdash_{\mathcal{N}} \forall^{o} p\ q.(p \to q \to \bot_{\mathcal{N}}) \to ((\neg_{\mathcal{G}} p) \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}) \to (p =_{\mathcal{G}} q) \to \bot_{\mathcal{N}}$

Lemma 6.4.6 (Func=). $\vdash_{\mathcal{N}} \forall^{\sigma\tau} k\ h.\forall^{\sigma} t.((k\ t =_{\mathcal{G}} h\ t) \to \bot_{\mathcal{N}}) \to (k =_{\mathcal{G}} h) \to \bot_{\mathcal{N}}$

Lemma 6.4.7 (Mat). $\vdash_{\mathcal{N}} \forall^{\sigma_1\sigma_2\ldots\sigma_n o} p.\forall^{\sigma_1} s_1\ t_1.\forall^{\sigma_2} s_2\ t_2.\ \ldots\ .\forall^{\sigma_n} s_n\ t_n.(\neg_{\mathcal{G}}(s_1 =_{\mathcal{G}} t_1) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(s_2 =_{\mathcal{G}} t_2) \to \bot_{\mathcal{N}}) \to \cdots \to (\neg_{\mathcal{G}}(s_n =_{\mathcal{G}} t_n) \to \bot_{\mathcal{N}}) \to p\ s_1\ s_2 \ldots s_n \to \neg_{\mathcal{G}}(p\ t_1\ t_2 \ldots t_n) \to \bot_{\mathcal{N}}$

Lemma 6.4.8 (Dec). $\vdash_{\mathcal{N}} \forall^{\sigma_1\sigma_2\ldots\sigma_n \iota} h.\forall^{\sigma_1} s_1\ t_1.\forall^{\sigma_2} s_2\ t_2.\ \ldots\ .\forall^{\sigma_n} s_n\ t_n.(\neg_{\mathcal{G}}(s_1 =_{\mathcal{G}} t_1) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(s_2 =_{\mathcal{G}} t_2) \to \bot_{\mathcal{N}}) \to \cdots \to (\neg_{\mathcal{G}}(s_n =_{\mathcal{G}} t_n) \to \bot_{\mathcal{N}}) \to \neg_{\mathcal{G}}(h\ s_1\ s_2 \ldots s_n =_{\mathcal{G}} h\ t_1\ t_2 \ldots t_n) \to \bot_{\mathcal{N}}$

For simplicity the Coq proofs provided in Appendix A for the Mat and Dec rules are for the binary case.

Lemma 6.4.9 (Con). $\vdash_{\mathcal{N}} \forall^{\iota} s_1\ t_1\ s_2\ t_2.(\neg_{\mathcal{G}}(s_1 =_{\mathcal{G}} s_2) \to \neg_{\mathcal{G}}(t_1 =_{\mathcal{G}} s_2) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(s_1 =_{\mathcal{G}} t_2) \to \neg_{\mathcal{G}}(t_1 =_{\mathcal{G}} t_2) \to \bot_{\mathcal{N}}) \to (s_1 =_{\mathcal{G}} t_1) \to \neg_{\mathcal{G}}(s_2 =_{\mathcal{G}} t_2) \to \bot_{\mathcal{N}}$

Theorem 6.4.10. The branch transformation $\Psi^*_{\mathcal{G}}$ respects the following rules:

Closed$\bot$, Closed$\neg\top$, Closed, DNeg, And, NegAnd, Or, NegOr, Imp, NegImp, DeMorgan$\exists$,

Forall, Exists, Cut, Closed $\neq$, ClosedSym, Bool $=$, Func $=$, Mat, Dec, Con

Proof. Follows directly from Theorem 6.3.14 and Lemmas 6.4.3, 6.4.4, 6.4.5, 6.4.6, 6.4.7, 6.4.8, and 6.4.9, respectively. □

6.4.3 Tableau Rules in $\mathcal{T}_{Full}$ that are not Respected by $\Psi^*_{\mathcal{G}}$

In [4] Benzmüller et al. defined a non-extensional model class $\mathcal{M}_\beta$. They prove soundness of an ND calculus $\mathfrak{NK}_\beta$ in Theorem 7.3. It is easy to check that our ND calculus is also sound with respect to $\mathcal{M}_\beta$.

Lemma 6.4.11. $\exists \mathcal{M} \in \mathcal{M}_\beta : \mathcal{M} \vDash \Psi_{\mathcal{G}}(\exists^o x\ y.\exists^{oo} f.\neg(x \to y \to f\ x \to f\ y))$

Proof. The model $\mathcal{M}^{\beta f}$ constructed in Example 5.4 of [4] is in $\mathcal{M}_\beta$ and it is not difficult to verify that it satisfies $\Psi_{\mathcal{G}}(\exists^o x\ y.\exists^{oo} f.\neg(x \to y \to f\ x \to f\ y))$. □

Lemma 6.4.12. $\{\Psi_{\mathcal{G}}(\exists^o x\ y.\exists^{oo} f.\neg(x \to y \to f\ x \to f\ y))\} \nvdash_{\mathcal{N}} \bot_{\mathcal{N}}$

Proof. Assume $\{\Psi_{\mathcal{G}}(\exists^o x\ y.\exists^{oo} f.\neg(x \to y \to f\ x \to f\ y))\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$.
Our ND calculus $\mathcal{N}$ is sound with respect to $\mathcal{M}_\beta$.
Therefore, $\forall \mathcal{M} \in \mathcal{M}_\beta : \mathcal{M} \nvDash \Psi_{\mathcal{G}}(\exists^o x\ y.\exists^{oo} f.\neg(x \to y \to f\ x \to f\ y))$.
This contradicts Lemma 6.4.11. □

Theorem 6.4.13. The branch transformation $\Psi^*_{\mathcal{G}}$ does not respect the BoolExt rule.

Proof. Let $\mathcal{T}$ be a tableau calculus containing exactly the following rules:

Exists, NegImp, Mating, BoolExt, Closed

A tableau refutation of the branch $\{\exists^o x\ y.\exists^{oo} f.\neg(x \to y \to f\ x \to f\ y)\}$ that uses only the rules in $\mathcal{T}$ is given in Figure 6.4. Therefore, this formula is $\mathcal{T}$-refutable. We know by Lemma 6.4.12 that $\{\Psi_{\mathcal{G}}(\exists^o x\ y.\exists^{oo} f.\neg(x \to y \to f\ x \to f\ y)\} \nvdash_{\mathcal{N}} \bot_{\mathcal{N}}$. Since $\exists x\ y.\exists f.\neg(x \to y \to f\ x \to f\ y)$ is a ground formula, therefore using Corollary 5.2.2 it follows that the logical transformation $\Psi_{\mathcal{G}}$ does not respect $\mathcal{T}$. We know that $\Psi^*_{\mathcal{G}}$ respects the rules Exists, NegImp, Mating, and Closed using Theorem 6.4.10. Therefore, $\Psi^*_{\mathcal{G}}$ does not respect the BoolExt rule. □

Lemma 6.4.14. $\exists \mathcal{M} \in \mathcal{M}_\beta : \mathcal{M} \vDash \Psi_{\mathcal{G}}(\exists^{\iota o} f\ g.\neg((\neg\exists^o x.f\ x \neq g\ x) \to (f = g)))$

Proof. The model $\mathcal{M}^{\beta\xi b}$ constructed in Example 5.6 of [4] is in $\mathcal{M}_\beta$ and it is not difficult to verify that it satisfies $\Psi_{\mathcal{G}}(\exists^{\iota o} f\ g.\neg((\neg\exists^o x.f\ x \neq g\ x) \to (f = g)))$. □

Lemma 6.4.15. $\{\Psi_{\mathcal{G}}(\exists^{\iota o} f\ g.\neg((\neg\exists^o x.f\ x \neq g\ x) \to (f = g)))\} \nvdash_{\mathcal{N}} \bot_{\mathcal{N}}$

Proof. Assume $\{\Psi_{\mathcal{G}}(\exists^{\iota o} f\ g.\neg((\neg\exists^o x.f\ x \neq g\ x) \to (f = g)))\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$.
Our ND calculus $\mathcal{N}$ is sound with respect to $\mathcal{M}_\beta$.
Thus, $\forall \mathcal{M} \in \mathcal{M}_\beta : \mathcal{M} \nvDash \Psi_{\mathcal{G}}(\exists^{\iota o} f\ g.\neg((\neg\exists^o x.f\ x \neq g\ x) \to (f = g)))$.
This contradicts Lemma 6.4.14. □

$$\begin{array}{c}
\exists x\, y.\exists f.\neg(x \to y \to f\, x \to f\, y) \\
\exists y.\exists f.\neg(x \to y \to f\, x \to f\, y) \\
\exists f.\neg(x \to y \to f\, x \to f\, y) \\
\neg(x \to y \to f\, x \to f\, y) \\
x \\
\neg(y \to f\, x \to f\, y) \\
y \\
\neg(f\, x \to f\, y) \\
f\, x \\
\neg f\, y \\
x \neq y \\
\hline
\begin{array}{l|l} x & y \\ \neg y & \neg x \end{array}
\end{array}$$

Figure 6.4: Tableau Refutation of $\exists x\, y.\exists f.\neg(x \to y \to f\, x \to f\, y)$

Theorem 6.4.16. The branch transformation $\Psi^*_{\mathcal{G}}$ does not respect the FuncExt rule.

Proof. Let $\mathcal{T}$ be a tableau calculus containing exactly the following rules:

Exists, NegImp, DeMorgan$\exists$, FuncExt, Forall, Closed

A tableau refutation of the branch $\{\exists^{\iota o} f\, g.\neg((\neg\exists^o x.f\, x \neq g\, x) \to (f = g))\}$ that uses only the rules in $\mathcal{T}$ is given in Figure 6.5. Therefore, this formula is $\mathcal{T}$-refutable. We know by Lemma6.4.15 that $\{\Psi_{\mathcal{G}}(\exists^{\iota o} f\, g.\neg((\neg\exists^o x.f\, x \neq g\, x) \to (f = g)))\} \nvdash_{\mathcal{N}} \bot_{\mathcal{N}}$. Since $\exists^{\iota o} f\, g.\neg((\neg\exists^o x.f\, x \neq g\, x) \to (f = g))$ is a ground formula, therefore using Corollary 5.2.2 it follows that the logical transformation $\Psi_{\mathcal{G}}$ does not respect $\mathcal{T}$. We know that $\Psi^*_{\mathcal{G}}$ respects the rules Exists, NegImp, DeMorgan$\exists$, FuncExt, and Closed using Theorem 6.4.10. Therefore, $\Psi^*_{\mathcal{G}}$ does not respect the FuncExt rule. □

$$\begin{array}{c}
\exists f\, g.\neg((\neg\exists^o x.f\, x \neq g\, x) \to (f = g)) \\
\exists g.\neg((\neg\exists^o x.f\, x \neq g\, x) \to (f = g)) \\
\neg((\neg\exists^o x.f\, x \neq g\, x) \to (f = g)) \\
\neg\exists^o x.f\, x \neq g\, x \\
f \neq g \\
\forall x.\neg(f\, x \neq g\, x) \\
f\, x \neq g\, x \\
\neg(f\, x \neq g\, x)
\end{array}$$

Figure 6.5: Tableau Refutation of $\exists^{\iota o} f\, g.\neg((\neg\exists^o x.f\, x \neq g\, x) \to (f = g))$

Theorem 6.4.17. The branch transformation $\Psi^*_{\mathcal{G}}$ does not respect the following rules:

DeMorgan$\forall$, BoolExt, FuncExt,

Proof. Follows directly from Theorems 6.3.15, 6.4.13, and 6.4.16. □

It may be a bit surprising that the only tableau rule which makes use of classical logic is the DeMorgan$\forall$ rule. Thus, this rule is not respected by the straightforward transformation $\Psi^*_{\mathcal{G}}$.

On the other hand, it is not at all surprising that $\Psi^*_{\mathcal{G}}$ does not respect the BoolExt rule and the FuncExt rule. This is because they explicitly make use of the concepts of Boolean and functional extensionality respectively. Note that $\Psi_{\mathcal{G}}$ transforms equality as Leibniz equality and thus doesn't introduce any concept of extensionality. Moreover, the extensionality axioms cannot be proven using Leibniz equality.

Chapter 7

The Girard-Kuroda Transformation

The Girard-Kuroda transformation $\Psi_{\mathcal{GK}}$ is a classical non-extensional logical transformation which slightly modifies the Girard transformation $\Psi_{\mathcal{G}}$. We call this transformation Girard-Kuroda because it turns out to be the same as Kuroda's negative translation that translates from classical first-order logic to intuitionistic first-order logic [24, 29]. Our aim using this transformation is to translate from classical higher-order logic to intuitionistic higher-order logic. Other than the BoolExt and FuncExt rules which introduce the principles of Boolean extensionality and functional extensionality respectively, the only rule that is not respected by the intuitionistic transformation $\Psi_{\mathcal{G}}$ is the DeMorgan$\forall$ rule. This is because the DeMorgan$\forall$ rule makes use of classical principles and does not hold intuitionistically [32].

This year a paper has been published showing that Glivenko's theorem also holds for second-order propositional logic without the $\forall$ quantifier [32]. We have seen this paper on October 29, 2009 and its result is similar to one of our results. Namely, we show that the DeMorgan$\forall$ rule is not provable intuitionistically and that all the other rules are. This is similar to the result in [32] stating that Glivenko's theorem does not hold for the full fragment of second-order propositional logic that includes the $\forall$ however holds for the fragment that excludes the $\forall$ quantifier. One difference is that we show this result for elementary type theory, i.e., classical non-extensional higher-order logic (see [1]), rather than second-order propositional logic.

In the coming two sections we give a logical transformation that we call Girard-Kuroda and prove that it transforms classical elementary type theory to intuitionistic elementary type theory.

7.1 Properties of the Girard-Kuroda Transformation

The logical transformation $\Psi_{\mathcal{GK}}$ modifies the Girard transformation of the logical constant $\forall$ such that the DeMorgan$\forall$ rule is respected.

The Girard-Kuroda transformation $\Psi_{\mathcal{GK}}$ is exactly like $\Psi_{\mathcal{G}}$ except for it transforms the logical constant $\forall$ as follows:

$$\Psi_{\mathcal{GK}}(\forall) = \lambda f.\forall x.\neg_{\mathcal{G}} \neg_{\mathcal{G}} f\ x$$

We write $\forall_{\mathcal{GK}}$ as shorthand for $\Psi_{\mathcal{GK}}(\forall)$.

Proposition 7.1.1. The Girard-Kuroda transformation $\Psi_{\mathcal{GK}}$ is a logical transformation.

Proof. By Proposition 6.1.1 we know that the Girard transformation is a logical transformation. It is easy to see that the Girard-Kuroda transformation is classically equivalent to the Girard transformation. Hence, the Girard-Kuroda transformation is also a logical transformation. □

Proposition 7.1.2. The Girard-Kuroda transformation $\Psi_{\mathcal{GK}}$ respects beta.

Proof. Follows from Proposition 5.1.4 since $\Psi_{\mathcal{GK}}$ is compositional. □

Let $\Psi^*_{\mathcal{GK}}$ be the branch transformation that trivially extends $\Psi_{\mathcal{GK}}$.

7.2 The Tableau Calculus $\mathcal{T}_{elem}$

Let $\mathcal{R}_{elem}$ be the following set of rules:

$$\{\mathsf{Closed}\bot,\ \mathsf{Closed}\neg\top,\ \mathsf{Closed},\ \mathsf{DNeg},\ \mathsf{And},\ \mathsf{NegAnd},\ \mathsf{Or},\ \mathsf{NegOr},$$

$$\mathsf{Imp},\ \mathsf{NegImp},\ \mathsf{DeMorgan}\exists,\ \mathsf{DeMorgan}\forall,\ \mathsf{Forall},\ \mathsf{Exists},\ \mathsf{Cut},$$

$$\mathsf{Closed}\neq,\ \mathsf{ClosedSym},\ \mathsf{Bool}=,\ \mathsf{Leibniz},\ \mathsf{Func}=,\ \mathsf{Mat},\ \mathsf{Dec},\ \mathsf{Con}\}$$

The tableau calculus $\mathcal{T}_{elem}$ is the pair $\langle Ter, \mathcal{R}_{elem}\rangle$. This tableau calculus corresponds to elementary type theory. Elementary type theory is a classical non-extensional higher-order logic (see [1]).

Note that the $\Psi_{\mathcal{GK}}$ transforms all logical constants the same way as the Girard transformation except for the forall logical constant $\forall$. Therefore, for each of the tableau rules that do not contain the logical constant $\forall$ whenever we know that the Girard transformation respects this rule we also know that the Girard-Kuroda transformation respects it. The only tableau rules that contain the $\forall$ logical constant are Forall, the DeMorgan$\exists$, DeMorgan$\forall$, and Leibniz. Thus, we show that the Girard-Kuroda transformation respects those four rules.

The Coq proofs of the following four lemmas are provided in Appendix B.

Lemma 7.2.1 (Forall). $\vdash_{\mathcal{N}} \forall^{\sigma o} f.\forall^{\sigma} t.(f\ t \to \bot_{\mathcal{N}}) \to (\forall^{\sigma}_{\mathcal{GK}} f) \to \bot_{\mathcal{N}}$

Lemma 7.2.2 (DeMorgan$\exists$). $\vdash_{\mathcal{N}} \forall^{\sigma\to o} f.((\forall^{\sigma}_{\mathcal{GK}}(\lambda x.\neg_{\mathcal{G}} f\ x)) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(\exists^{\sigma}_{\mathcal{G}} f)) \to \bot_{\mathcal{N}}$

Lemma 7.2.3 (DeMorgan$\forall$). $\vdash_{\mathcal{N}} \forall^{\sigma\to o} f.((\exists^{\sigma}_{\mathcal{G}}(\lambda x.\neg_{\mathcal{G}} f\ x)) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(\forall^{\sigma}_{\mathcal{GK}} f)) \to \bot_{\mathcal{N}}$

Lemma 7.2.4 (Leibniz). $\vdash_{\mathcal{N}} \forall^{\iota} x\ y.((\forall^{\sigma o}_{\mathcal{GK}}(\lambda f.f\ x \to f\ y)) \to \bot_{\mathcal{N}}) \to (x =_{\mathcal{G}} y) \to \bot_{\mathcal{N}}$

Lemma 7.2.5. The branch transformation $\Psi^*_{\mathcal{GK}}$ respects all of the rules in $\mathcal{T}_{elem}$.

Proof. This follows directly from Theorem 6.4.10 and Lemmas 7.2.1, 7.2.2, and 7.2.3. □

Theorem 7.2.6. The Girard-Kuroda transformation $\Psi_{\mathcal{GK}}$ respects $\mathcal{T}_{elem}$.

Proof. This follows directly from Lemma 7.2.5. □

Corollary 7.2.7. Let s be a ground formula. If $\{s\}$ is $\mathcal{T}_{elem}$-refutable, then $\{\Psi_{\mathcal{GK}}(s)\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$.

Proof. The proof follows directly from Corollary 5.2.2 and from Theorem 7.2.6 □

Using Lemma 7.2.5 we know that $\Psi_{\mathcal{GK}}$ respects all the rules in $\mathcal{T}_{Full}$ except for the Bool $\neq$ and Func $\neq$ rules. Moreover, using similar proofs to those of Theorems 6.4.13 and 6.4.16 provided in Chapter 6 we know that $\Psi_{\mathcal{GK}}$ does not respect the Bool $\neq$ and Func $\neq$ rules .

Chapter 8

The Girard-Kuroda-Per Transformation

The Girard-Kuroda-Per transformation $\Psi_{\mathcal{GKP}}$ is a classical extensional logical transformation which modifies the Girard-Kuroda transformation $\Psi_{\mathcal{GK}}$. It introduces the principles of Boolean and functional extensionality and therefore makes it possible to respect the rules BoolExt and FuncExt. In Section 8.3, we state why we call this transformation Per.

In 1956 Gandy introduced a transformation from extensional to non-extensional simple type theory [14]. His aim was to show that if simple type theory excluding the axioms of extensionality is consistent, then so is simple type theory including extensionality. The $\Psi_{\mathcal{GKP}}$ transformation is similar in that it also transforms extensional to non-extensional simple type theory but additionally it transforms classical to intuitionistic logic. The two transformations are apparently different. Gandy's transformation uses a binary relation and a predicate defined by mutual recursion. Meanwhile, $\Psi_{\mathcal{GKP}}$ uses a single binary relation that defined inductively on types, and which turns out to be a partial equivalence relation. The question of whether Gandy's transformation and $\Psi_{\mathcal{GKP}}$ are the same up to double negations is still open for future work.

8.1 The Girard-Kuroda-Per Transformation $\Psi_{\mathcal{GKP}}$

Definition 8.1.1. For every type σ we define inductively a term R^σ as follows:

$$
\begin{aligned}
R^o &= \lambda x\, y.(x \to y) \wedge_{\mathcal{G}} (y \to x)\\
R^\iota &= \lambda x\, y.\forall q.q\, x \to q\, y\\
R^{\sigma\to\tau} &= \lambda f\, g.\forall x\, y.R^\sigma\, x\, y \to \neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^\tau(f\, x)(g\, y))
\end{aligned}
$$

This term R^σ corresponds to a binary relation on type σ. To give the reader a good intuition we sometimes speak R^σ as a relation rather than as a term. Note that at function types R is defined as a logical relation up to double negations.

The Girard-Kuroda-Per transformation $\Psi_{\mathcal{GKP}}$ agrees with the Girard transformation $\Psi_{\mathcal{G}}$ on all logical constants except for $\forall$, $\exists$ and $=$, which it transformes as follows:

$$
\begin{aligned}
&\Psi_{\mathcal{GKP}}(=^\sigma) = R^\sigma\\
&\Psi_{\mathcal{GKP}}(\forall^\sigma) = \lambda f.\forall x.(R^\sigma x\, x) \to \neg_{\mathcal{G}}\neg_{\mathcal{G}} f\, x\\
&\Psi_{\mathcal{GKP}}(\exists^\sigma) = \lambda f.\forall^o p.(\forall x.(R^\sigma x\, x) \to f\, x \to p) \to p
\end{aligned}
$$

We write $\forall_{\mathcal{GKP}}$ and $\exists_{\mathcal{GKP}}$ as shorthand for $\Psi_{\mathcal{GKP}}(\forall)$ and $\Psi_{\mathcal{GKP}}(\exists)$, respectively.

8.2 $\Psi_{\mathcal{GKP}}$ is a Logical Transformation

Lemma 8.2.1. $\forall \sigma \in \mathbb{T} : \forall \mathcal{I} \in Interp : \forall a, b \in \mathcal{I}(\sigma) : (\hat{\mathcal{I}}(R^\sigma)\, a\, b = 1) \iff a = b$

Proof. We prove this lemma by induction on types. Let $\mathcal{I}$ be an arbitrary interpretation. Let a and b be arbitrary elements in $\mathcal{I}(\sigma)$. We show $(\hat{\mathcal{I}}(R^\sigma)\, a\, b = 1) \iff a = b$.

- Case $\sigma = o$:
  We want to show $\hat{\mathcal{I}}(R^o)\, a\, b = 1 \iff a = b$. By expanding the definition of R^o this reduces to showing $\hat{\mathcal{I}}(\lambda x\, y.(x \to y) \wedge_{\mathcal{G}} (y \to x))\, a\, b = 1 \iff a = b$. This is equivalent to showing $\widehat{\mathcal{I}^{xy}_{ab}}((x \to y) \wedge_{\mathcal{G}} (y \to x)) = 1 \iff a = b$, which is obviously true.

- Case $\sigma = \iota$:
  We want to show $\hat{\mathcal{I}}(R^\iota)\, a\, b = 1 \iff a = b$. We know that R^ι is Leibniz equality and by Proposition 6.1.1 that the Girard transformation is a logical transformation that maps $=$ to Leibniz equality. Therefore, the interpretation of Leibniz equality is ordinary equality.

- Case $\sigma = \sigma_1\sigma_2$:
  We want to show $\hat{\mathcal{I}}(R^{\sigma_1\sigma_2})\, a\, b = 1 \iff a = b$.

  - We show $\hat{\mathcal{I}}(R^{\sigma_1\sigma_2})\, a\, b = 1 \implies a = b$.
    Assume $\hat{\mathcal{I}}(R^{\sigma_1\sigma_2})\, a\, b = 1$. We need to show $a = b$. Let $c \in \mathcal{I}(\sigma_1)$ be given. Now we need to show $a(c) = b(c)$.

$$
\begin{aligned}
& \hat{\mathcal{I}}(R^{\sigma_1\sigma_2})\, a\, b = 1 \\
\iff & \hat{\mathcal{I}}(\lambda f\, g.\forall x\, y.R^{\sigma_1}\, x\, y \to \neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^{\sigma_2}(f\, x)(g\, y)))\, a\, b = 1 \\
\iff & \widehat{\mathcal{I}^{fg}_{ab}}(\forall x\, y.R^{\sigma_1}\, x\, y \to \neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^{\sigma_2}(f\, x)(g\, y))) = 1 \\
\implies & \widehat{\mathcal{I}^{fgxy}_{abcc}}(R^{\sigma_1}\, x\, y \to \neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^{\sigma_2}(f\, x)(g\, y))) = 1 \\
\iff & \widehat{\mathcal{I}^{fgxy}_{abcc}}(R^{\sigma_1}\, x\, y \to (R^{\sigma_2}(f\, x)(g\, y))) = 1 \\
\iff & (\widehat{\mathcal{I}^{fgxy}_{abcc}}(R^{\sigma_1}\, x\, y) = 1 \implies \widehat{\mathcal{I}^{fgxy}_{abcc}}(R^{\sigma_2}(f\, x)(g\, y))) = 1) \\
\iff & (\hat{\mathcal{I}}(R^{\sigma_1})\, c\, c = 1 \implies \hat{\mathcal{I}}(R^{\sigma_2})(a(c))(b(c)) = 1) \\
\iff & (c = c \implies a(c) = b(c)) && \text{by IH} \\
\iff & a(c) = b(c)
\end{aligned}
$$

  - We show $a = b \implies \hat{\mathcal{I}}(R^{\sigma_1\sigma_2})\, a\, b = 1$

Assume $a = b$. We need to show $\hat{\mathcal{I}}(R^{\sigma_1\sigma_2})\, a\, b = 1$.

$$
\begin{aligned}
&\hat{\mathcal{I}}(R^{\sigma_1\sigma_2})\, a\, b = 1 \\
\iff &\hat{\mathcal{I}}(\lambda f\, g. \forall x\, y. R^{\sigma_1}\, x\, y \to \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R^{\sigma_2}(f\, x)(g\, y)))\, a\, b = 1 \\
\iff &\widehat{\mathcal{I}^{fg}_{ab}}(\forall x\, y. R^{\sigma_1}\, x\, y \to \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R^{\sigma_2}(f\, x)(g\, y))) = 1 \\
\implies &\forall c,\, d \in \mathcal{I}(\sigma_1) : \widehat{\mathcal{I}^{fgxy}_{abcd}}(R^{\sigma_1}\, x\, y \to \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R^{\sigma_2}(f\, x)(g\, y))) = 1 \\
\iff &\forall c,\, d \in \mathcal{I}(\sigma_1) : \widehat{\mathcal{I}^{fgxy}_{abcd}}(R^{\sigma_1}\, x\, y \to (R^{\sigma_2}(f\, x)(g\, y))) = 1 \\
\iff &(\forall c,\, d \in \mathcal{I}(\sigma_1) : \widehat{\mathcal{I}^{fgxy}_{abcd}}(R^{\sigma_1}\, x\, y) = 1 \implies \widehat{\mathcal{I}^{fgxy}_{abcd}}(R^{\sigma_2}(f\, x)(g\, y))) = 1) \\
\iff &(\forall c,\, d \in \mathcal{I}(\sigma_1) : \hat{\mathcal{I}}(R^{\sigma_1})\, c\, d = 1 \implies \hat{\mathcal{I}}(R^{\sigma_2})(a(c))(b(d)) = 1) \\
\iff &(\forall c,\, d \in \mathcal{I}(\sigma_1) : c = d \implies a(c) = b(d)) \qquad \text{by IH} \\
\iff &a = b
\end{aligned}
$$

□

Lemma 8.2.2. For every interpretation $\mathcal{I}$ and every a in $\mathcal{I}(\sigma)$ we have $\mathcal{I}(R^\sigma)\, a\, a = 1$.

Proof. Follows directly from Lemma 8.2.1. □

Theorem 8.2.3. Girard-Kuroda-Per is a logical transformation.

Proof. Follows directly from the fact that the Girard transformation is a logical transformation and from Lemmas 8.2.1 and 8.2.2. □

8.3 Properties of R

In order to make progress, we first have to consider which properties of the relation R are provable in the ND calculus $\mathcal{N}$.

Definition 8.3.1 (Reflexive Type)**.** A type σ is *reflexive* if $\vdash_{\mathcal{N}} \forall^\sigma x. R^\sigma x x$.

Definition 8.3.2 (Symmetric)**.** We will say an $\mathcal{N}$-term $G : \sigma\sigma o$ is *symmetric* if the following holds:

$$\vdash_{\mathcal{N}} \forall^\sigma x\, y. (G\, x\, y) \to (G\, y\, x)$$

Definition 8.3.3 (Transitive)**.** We will say an $\mathcal{N}$-term $G : \sigma\sigma o$ is *transitive* if the following holds:

$$\vdash_{\mathcal{N}} \forall^\sigma x\, y\, z. (G\, x\, y) \to (G\, y\, z) \to (G\, x\, z)$$

Definition 8.3.4 (Negatively Transitive)**.** We will say an $\mathcal{N}$-term $G : \sigma\sigma o$ is *negatively transitive* if the following holds:

$$\vdash_{\mathcal{N}} \forall^\sigma x\, y\, z. (G\, x\, y) \to \neg_{\mathcal{G}} (G\, x\, z) \to \neg_{\mathcal{G}} (G\, y\, z)$$

We show that for all types σ the relation R^σ that our transformation transforms equality into is both symmetic and transitive. Moreover, we show that not all types are reflexive. Hence, we can conclude that R^σ is a partial equivalence relation. This is the reason why we call this transformation Girard-Kuroda-Per.

Lemma 8.3.5. For each type σ the relation R^σ is symmetric.

Proof. A Coq proof of this lemma is included in Appendix C. □

Lemma 8.3.6. For each type σ the relation R^σ is transitive.

Proof. A Coq proof of this lemma is included in Appendix C. □

Lemma 8.3.7. For each type σ the relation R^σ is negatively transitive.

Proof. A Coq proof of this lemma is included in Appendix C. □

First we show that the two basic types o and ι are reflexive. Then we show some non-reflexive types, namely the types oo and $o\iota$. This shows that not all types are reflexive.

Note that in Lemma 8.2.1 we were able to show for all types σ that R^σ is equivalent to $=^\sigma$ using the classical extensional semantics. The ND calculus $\mathcal{N}$ is intuitionistic and non-extensional. This is why it does not follow that all types σ are reflexive.

Lemma 8.3.8. The basic type ι is reflexive.

Proof. A Coq proof of this lemma can be found in Appendix C. □

Lemma 8.3.9. The basic type o is reflexive.

Proof. A Coq proof of this lemma can be found in Appendix C. □

Lemma 8.3.10. The types oo and $o\iota$ are not reflexive.

Proof. The model $\mathcal{M}^{\beta f}$ constructed in Example 5.4 of [4] is in $\mathcal{M}_\beta$ and it is not difficult to verify that it satisfies the negation of both $\forall^{oo} f.R^{oo} f f$ and $\forall^{o\iota} f.R^{o\iota} f f$. Since $\mathcal{N}$ is sound with respect to M_β, therefore the types oo and $o\iota$ cannot be reflexive. □

Now we show a very interesting theorem, namely Theorem 8.3.21, stating that if for all the free variables x of a term t we know $\Gamma \vdash_{\mathcal{N}} R\, x\, x$ then we know $\Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ (\Psi_{\mathcal{GKP}}(t))\ (\Psi_{\mathcal{GKP}}(t)))$. At first sight this theorem might seem obvious but in fact it is not very direct to prove. In fact, a more general form of this theorem stating that this property is true for all $\mathcal{N}$-terms does not hold. This theorem is necessary to prove that Girard-Kuroda-Per respects the tableau calculus $\mathcal{T}_{Full_{res}}$ which will be introduced in the next section.

Before proving this theorem we will first show that for all transformations of the logical constants c' we know $\vdash_{\mathcal{N}} R\ c'\ c'$. The most interesting case is the $\exists_{\mathcal{GKP}}$ case, for which we provide a step-by-step explanation of the proof. For each of the other logical constants we provide Coq proofs in Appendix C.

Lemma 8.3.11. $\vdash_{\mathcal{N}} R\, \neg_{\mathcal{G}} \neg_{\mathcal{G}}$

Proof. A Coq proof of this lemma is included in Appendix C. □

Lemma 8.3.12. $\vdash_{\mathcal{N}} R\ \rightarrow_{\mathcal{G}} \rightarrow_{\mathcal{G}}$

Proof. A Coq proof of this lemma is included in Appendix C. □

Lemma 8.3.13. $\vdash_{\mathcal{N}} R\ \wedge_{\mathcal{G}} \wedge_{\mathcal{G}}$

Proof. A Coq proof of this lemma is included in Appendix C. □

Lemma 8.3.14. $\vdash_{\mathcal{N}} R\ \vee_{\mathcal{G}} \vee_{\mathcal{G}}$

Proof. A Coq proof of this lemma is included in Appendix C. □

Lemma 8.3.15. $\vdash_{\mathcal{N}} R\ \exists^{\sigma}_{\mathcal{GKP}} \exists^{\sigma}_{\mathcal{GKP}}$ for any type σ

Proof. A Coq proof of this lemma is included in Appendix C. We also provide an explanation of the proof here.
We show that $\vdash_{\mathcal{N}} R^{(\sigma o)o} \exists^{\sigma}_{\mathcal{GKP}}\ \exists^{\sigma}_{\mathcal{GKP}}$ (for any σ).

1. After unfolding the definition of $R^{(\sigma o)o}$ we need to show $\neg_{\mathcal{G}} \neg_{\mathcal{G}} (R^o (\exists_{\mathcal{GKP}} g_1)\ (\exists_{\mathcal{GKP}} g_2))$ under the assumption $\forall^{\sigma} z_1\ z_2 . (R^{\sigma} z_1\ z_2) \to \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R^o (g_1\ z_1)\ (g_2\ z_2))$.
2. So suppose $\neg_{\mathcal{G}} (R^o (\exists_{\mathcal{GKP}} g_1)\ (\exists_{\mathcal{GKP}} g_2))$.
3. We need to show $R^o (\exists_{\mathcal{GKP}} g_1)\ (\exists_{\mathcal{GKP}} g_2)$, of which we present only the implication from left to right here. (The other part is symmetric.)
4. So suppose $\exists_{\mathcal{GKP}} g_1$, i.e., $\forall^o p.\ (\forall^{\sigma} x.\ R^{\sigma} x\ x \to_{\mathcal{G}} g_1\ x \to_{\mathcal{G}} p) \to_{\mathcal{G}} p$.
5. We need to show $\exists_{\mathcal{GKP}} g_2$, i.e., $\forall^o p.\ (\forall^{\sigma} x . R^{\sigma} x\ x \to_{\mathcal{G}} g_2\ x \to_{\mathcal{G}} p) \to_{\mathcal{G}} p$.
6. It suffices to show p. Note that we will not need to use the assumption $\forall^{\sigma} x . R^{\sigma} x\ x \to_{\mathcal{G}} g_2\ x \to_{\mathcal{G}} p$.
7. Using (4) this reduces to showing $\forall^{\sigma} x . R^{\sigma} x\ x \to_{\mathcal{G}} g_1\ x \to_{\mathcal{G}} p$.
8. So suppose $R^{\sigma} x\ x$ and $g_1\ x$.
9. We need to show p.
10. Instantiating the assumption in (1) yields $\neg_{\mathcal{G}} \neg_{\mathcal{G}} (R^o (g_1\ x)\ (g_2\ x))$.
11. It thus suffices to show $\neg_{\mathcal{G}} (R^o (g_1 x)\ (g_2\ x))$ (since from $\bot_{\mathcal{G}}$ we can conclude p).
12. So suppose $R^o\ (g_1\ x)\ (g_2\ x)$.
13. We need to show $\bot_{\mathcal{G}}$.
14. From $g_1\ x$ and $g_1\ x \to_{\mathcal{G}} g_2\ x$ we get $g_2\ x$.
15. By (2) it suffices to show $R^o (\exists_{\mathcal{GKP}} g_1)\ (\exists_{\mathcal{GKP}} g_2)$ (which is the original goal, but now we can use the accumulated assumptions).
16. The implication from right to left follows easily from (4).
17. For the other direction we need to show $\exists_{\mathcal{GKP}} g_2$, i.e., $\forall^o p.\ (\forall^{\sigma} x . R^{\sigma} x\ x \to_{\mathcal{G}} g_2\ x \to_{\mathcal{G}} p) \to_{\mathcal{G}} p$.
18. So suppose $\forall^{\sigma} x.\ R^{\sigma} x\ x \to_{\mathcal{G}} g_2\ x \to_{\mathcal{G}} q$.
19. We need to show q, which follows from (18), (8), and (14).

□

Lemma 8.3.16. $\vdash_{\mathcal{N}} R\ \forall^{\sigma}_{\mathcal{GKP}} \forall^{\sigma}_{\mathcal{GKP}}$ for any type σ

Proof. A Coq proof of this lemma is included in Appendix C. □

Lemma 8.3.17. $\vdash_{\mathcal{N}} R\ R^{\sigma} R^{\sigma}$ for any type σ

Proof. A Coq proof of this lemma is included in Appendix C. Note that the Coq proof term in Appendix C proves this property for any relation which is symmetric and transitive. Since we know R^σ is symmetric and transitive on all types thus using the Coq lemma we can obtain this lemma. □

In some of the lemmas we prove later on that involve showing a property for R^σ, we provide Coq proofs for the lemma generalized to any relation which is symmetric and transitive. Since we know R^σ is symmetric and transitive on all types σ thus using the Coq proofs we can obtain proofs for the corresponding lemma.

Lemma 8.3.18. $\vdash_{\mathcal{N}} R\ (\Psi_{\mathcal{GKP}}(c))(\Psi_{\mathcal{GKP}}(c))$ where c is a logical constant.

Proof. We prove this lemma for each logical constant c.

- Case $t = \top_{\mathcal{G}}$ or $t = \bot_{\mathcal{G}}$
  We need to show that $\vdash_{\mathcal{N}} R^o \bot\ \bot$ and $\vdash_{\mathcal{N}} R^o \top\ \top$. They both follow directly using Lemma 8.3.9.
- Case $t = \neg_{\mathcal{G}}$, $t = \rightarrow_{\mathcal{G}}$, $t = \wedge_{\mathcal{G}}$, $t = \vee_{\mathcal{G}}$, $t = \exists^\sigma_{\mathcal{GKP}}$, $t = \forall^\sigma_{\mathcal{GKP}}$, or $t = R^\sigma$
  Those cases are proven by Lemmas 8.3.11, 8.3.12, 8.3.13, 8.3.14, 8.3.15, 8.3.16, and 8.3.17 respectively.

□

Lemma 8.3.19. $\vdash_{\mathcal{N}} \forall t_1\ t_2.(\neg_{\mathcal{G}} \neg_{\mathcal{G}} R^{\sigma\tau} t_1\ t_1) \rightarrow (\neg_{\mathcal{G}} \neg_{\mathcal{G}} R^\sigma t_2\ t_2) \rightarrow (\neg_{\mathcal{G}} \neg_{\mathcal{G}} R^\tau (t_1\ t_2)(t_1\ t_2))$

Proof. A Coq proof of this lemma is included in Appendix C. □

Lemma 8.3.20. For all terms t, for all substitutions θ_1, θ_2 and, for all contexts Γ:

$$\text{if } \forall x \in FV(t) : \Gamma \vdash_{\mathcal{N}} R\ (\theta_1(x))(\theta_2(x)) \text{ then } \Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ (\theta_1(\Psi_{\mathcal{GKP}}(t)))(\theta_2(\Psi_{\mathcal{GKP}}(t))))$$

Proof. We prove this lemma by structural induction on $\Psi_{\mathcal{GKP}}(t)$.

- Case t is a variable x
  We know $\Psi_{\mathcal{GKP}}(t) = x$ and $FV(t) = \{x\}$. Hence, this case is trivial due to the assumption and Lemma 4.2.1.
- Case t is a logical constant
  Proof follows by Lemma 4.2.1 and by weakening of Lemma 8.3.18.
- Case $t = t_1\ t_2$
  We know $\Psi_{\mathcal{GKP}}(t) = (\Psi_{\mathcal{GKP}}(t_1))\ (\Psi_{\mathcal{GKP}}(t_2))$.
  Assume that the following holds:

  $$\forall x \in FV(t) : \Gamma \vdash_{\mathcal{N}} R\ (\theta_1(x))\ (\theta_2(x))$$

  We want to show that

  $$\Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ (\theta_1(\Psi_{\mathcal{GKP}}(t_1\ t_2)))\ (\theta_2(\Psi_{\mathcal{GKP}}(t_1\ t_2)))).$$

  Since $FV(t_1) \subseteq FV(t)$, therefore we know:

  $$\forall x \in FV(t_1) : \Gamma \vdash_{\mathcal{N}} R\ (\theta_1(x))\ (\theta_2(x))$$

Similarly since $FV(t_2) \subseteq FV(t)$, therefore we also know:

$$\forall x \in FV(t_2) : \Gamma \vdash_{\mathcal{N}} R\ (\theta_1(x))\ (\theta_2(x))$$

Thus, by applying the induction hypothesis with θ_1, θ_2, Γ we know the following two facts:

$$\Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ (\theta_1(\Psi_{\mathcal{GKP}}(t_1)))(\theta_2(\Psi_{\mathcal{GKP}}(t_1))))$$

$$\text{and } \Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ (\theta_1(\Psi_{\mathcal{GKP}}(t_2)))(\theta_2(\Psi_{\mathcal{GKP}}(t_2))))$$

By Lemma using 8.3.19 we directly get $\Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ (\theta_1(\Psi_{\mathcal{GKP}}(t_1\ t_2)))(\theta_2(\Psi_{\mathcal{GKP}}(t_1\ t_2))))$.

- Case $t = \lambda x.t'$
  We know $\Psi_{\mathcal{GKP}}(t) = \lambda x.(\Psi_{\mathcal{GKP}}(t'))$.
  Assume that the following holds:

$$\forall y \in FV(t) : \Gamma \vdash_{\mathcal{N}} R\ (\theta_1(y))\ (\theta_2(y))$$

We want to show

$$\Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ (\theta_1(\Psi_{\mathcal{GKP}}(\lambda x.t')))\ (\theta_2(\Psi_{\mathcal{GKP}}(\lambda x.t')))).$$

By the definition of substitution and $\Psi_{\mathcal{GKP}}$ we need to show

$$\Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ (\lambda x.\theta_1(\Psi_{\mathcal{GKP}}(t')))\ (\lambda x.\theta_2(\Psi_{\mathcal{GKP}}(t')))).$$

for which it suffices to show

$$\Gamma \vdash_{\mathcal{N}} R\ (\lambda x.\theta_1(\Psi_{\mathcal{GKP}}(t')))\ (\lambda x.\theta_2(\Psi_{\mathcal{GKP}}(t'))).$$

After unfolding the definition of R, we need to show

$$\Gamma, (R\ x_1\ x_2) \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ ((\lambda x.\theta_1(\Psi_{\mathcal{GKP}}(t')))\ x_1)\ ((\lambda x.\theta_2(\Psi_{\mathcal{GKP}}(t')))\ x_2)).$$

where x_1 and x_2 are two distinct fresh variables.
By β-reduction it remains to show

$$\Gamma, (R\ x_1\ x_2) \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ (\theta_1(\Psi_{\mathcal{GKP}}(t')))[x := x_1]\ (\theta_2(\Psi_{\mathcal{GKP}}(t')))[x := x_2]).$$

Defining $\Gamma' = \Gamma, (R\ x_1\ x_2)$ as well as $\theta_1' = \theta_1, [x := x_1]$ and $\theta_2' = \theta_2, [x := x_2]$, it is easy to see that we have

$$\forall y \in FV(t') : \Gamma' \vdash_{\mathcal{N}} R\ (\theta_1'(y))\ (\theta_2'(y)),$$

because of the assumption about $FV(t)$ and the fact that

$$FV(t') \subseteq FV(t) \cup \{x\}.$$

We therefore can apply the induction hypothesis to get

$$\Gamma' \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} (R\ \theta_1'(\Psi_{\mathcal{GKP}}(t'))\ \theta_2'(\Psi_{\mathcal{GKP}}(t'))),$$

which is what we needed to show.

□

$$\mathsf{Forall}_{res}\frac{A, \lceil s\ t\rceil^{\beta} \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot}\ \forall s \in A,\ t \text{ is admissible for } A$$

$$\mathsf{Func}=_{res}\frac{A, \lceil s_1\ t =_{\tau} s_2\ t\rceil^{\beta} \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot}\ s_1 =_{\sigma\tau} s_2 \in A,\ t \text{ is admissible for } A$$

$$\mathsf{Cut}_{res}\frac{A, s \vee \neg s \vdash_{\mathcal{T}} \bot}{A \vdash_{\mathcal{T}} \bot}\ s \text{ is admissible for } A$$

Figure 8.1: Restricted Forall, Func=, and Cut Rules

Theorem 8.3.21. For all terms t, and for all contexts Γ:

$$\text{if } \forall x \in FV(t) : \Gamma \vdash_{\mathcal{N}} R\ x\ x \text{ then } \Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} R\ (\Psi_{\mathcal{GKP}}(t))(\Psi_{\mathcal{GKP}}(t)))$$

Proof. Follows directly from Lemma 8.3.20 by using the identity substitutions. □

Corollary 8.3.22. For all terms t, and for all contexts Γ:

$$\text{if } \forall x \in FV(t) : (R\ x\ x \in \Gamma) \text{ or } x \text{ has a reflexive type, then}$$

$$\Gamma \vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} R\ (\Psi_{\mathcal{GKP}}(t))(\Psi_{\mathcal{GKP}}(t)))$$

Proof. Follows directly from Theorem 8.3.21 because if x has a reflexive type we know by the wk rule that $\Gamma \vdash_{\mathcal{N}} R\ x\ x$, and if $R\ x\ x \in \Gamma$ by hy rule we know $\Gamma \vdash_{\mathcal{N}} R\ x\ x$. □

8.4 The Tableau Calculus $\mathcal{T}_{Full_{res}}$

Definition 8.4.1 (Admissible for a Branch)**.** A term t is *admissible for a branch* A if for each variable $x \in FV(t)$, either $x \in FV^*(A)$ or x has a reflexive type.

The tableau calculus $\mathcal{T}_{Full_{res}}$ contains the full fragment of terms and all the tableau rules that are in $\mathcal{T}_{Full}$ except for Forall, Func=, and Cut, for which it contains restricted forms as shown in Figure 8.1. Recall that $\mathcal{T}_{Full}$ contains all tableau rules except for the Leibniz rule.

Proposition 8.4.2. The tableau calculus $\mathcal{T}_{Full_{res}}$ is complete.

Proof. Let A be a branch. We show that for every term t that is not admissible for A there is a corresponding term t' that could be used instead and is admissible for A. We can construct the term t' by replacing each free variable $x : \sigma_1 \ldots \sigma_n\alpha$ in t that does not occur free in A and does not have a reflexive type by $\lambda y_1 \ldots y_n.z$ where $z : \alpha$. Assume w.l.o.g. that z is not used as a fresh variable somewhere in the refutation being considered. By Lemmas 8.3.8 and 8.3.9 we can infer that z has a reflexive type. Thus, t' is admissible for A. We know by Proposition 6.4.1 that $\mathcal{T}_{Full}$ is complete. Whenever we apply the Forall, Func=, or the Cut rule with a term t we can apply the Forall_{res}, $\mathsf{Func}=_{res}$, or the Cut_{res} rule respectively with a corresponding term t' that is admissible for the branch. Hence, we can conclude that $\mathcal{T}_{Full_{res}}$ is complete. □

Note that not all types are reflexive. For instance by Lemma 8.3.10 the type oo and the type $o\iota$ are not reflexive. Thus, the restricted full tableau calculus $\mathcal{T}_{Full_{res}}$ is strictly less general than the full tableau calculus $\mathcal{T}_{Full}$.

8.5 $\Psi_{\mathcal{GKP}}$ respects $\mathcal{T}_{Full_{res}}$

Lemma 8.5.1. $\nvdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} R^{oo} x\ x$

Proof. The model $\mathcal{M}^{\beta f}$ constructed in Example 5.4 of [4] is in $\mathcal{M}_\beta$ and it is not difficult to verify that it satisfies $\neg_{\mathcal{G}} R^{oo} x\ x$. Thus, it does not satisfy $\neg_{\mathcal{G}} \neg_{\mathcal{G}} R^{oo} x\ x$. Since $\mathcal{N}$ is sound with respect to M_β we know $\nvdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} R^{oo} x\ x$. □

Proposition 8.5.2. The trivial branch transformation $\Psi^*_{\mathcal{GKP}_{tr}}$ that extends $\Psi_{\mathcal{GKP}}$ does not respect $\mathcal{T}_{Full_{res}}$.

Proof. This is because $\Psi^*_{\mathcal{GKP}_{tr}}$ does not respect some of the rules in $\mathcal{T}_{Full_{res}}$.
Assume $\Psi^*_{\mathcal{GKP}_{tr}}$ respects the tableau step $\langle\{x \neq x\}\rangle$. Then we know $\neg_{\mathcal{G}} R^{oo} x\ x \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$. Thus we can infer $\vdash_{\mathcal{N}} \neg_{\mathcal{G}} \neg_{\mathcal{G}} R^{oo} x\ x$. This contradicts Lemma 8.5.1. Hence $\Psi^*_{\mathcal{GKP}_{tr}}$ cannot respect $\langle\{x \neq x\}\rangle$. Since $\langle\{x \neq x\}\rangle \in \mathsf{Closed}{\neq}$, $\Psi^*_{\mathcal{GKP}_{tr}}$ does not respect the $\mathsf{Closed}{\neq}$ rule. □

The objective of the following examples is to illustrate that given a branch A, if its branch transformation includes $R\ x\ x$ for any free variable x in A, then it will respect the $\mathsf{Closed}{\neq}$ rule. Of course, those added formulas have to be discharged at some point in the process of proving that this branch transformation respects $\mathcal{T}_{Full_{res}}$. We also illustrate how this works. Later on, we will introduce a branch transformation doing exactly this and indeed prove that it respects $\mathcal{T}_{Full_{res}}$.

Example 8.5.3. Consider the formula $\exists z.z \neq z$. A tableau refutation of this formula using the Exists and $\mathsf{Closed}{\neq}$ rules looks as follows:

$$\exists z.z \neq z$$

$$y \neq y$$

To show that the $\mathsf{Closed}{\neq}$ step in this refutation is respected we need to show that the following holds:

$$\{\neg_{\mathcal{G}} R\ y\ y,\ R\ y\ y\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$$

This is obvious.
In oder to show that the Exists step in this refutation is respected we assume

$$\{\exists_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z,\ \neg_{\mathcal{G}} R\ y\ y,\ R\ y\ y\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$$

and want to show

$$\{\exists_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

This follows if we know

$$\vdash_{\mathcal{N}} (R\ y\ y \to \neg_{\mathcal{G}} R\ y\ y \to \bot_{\mathcal{N}}) \to (\exists_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z) \to \bot_{\mathcal{N}},$$

which is obtained by instantiating Lemma 8.5.13.

Example 8.5.4. Consider the formula $\forall z.z \neq z$ where z has a reflexive type. A tableau refutation of this formula using the Forall_{res} and $\mathsf{Closed}{\neq}$ rules looks as follows:

$$\forall z.z \neq z$$

$$y \neq y$$

As shown in the last example, proving that the Closed$\neq$ step in this refutation is respected is obvious. In oder to show that the Forall$_{res}$ step in this refutation is respected we assume

$$\{\forall_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z,\ \neg_{\mathcal{G}} R\ y\ y,\ R\ y\ y\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$$

and want to show

$$\{\forall_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

This follows if we know

$$\{((\forall_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z) \to (\neg_{\mathcal{G}} R\ y\ y) \to (R\ y\ y) \to \bot_{\mathcal{N}}), (\forall_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z)\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

From the restriction on the Forall$_{res}$ rule we can infer $\vdash_{\mathcal{N}} R\ y\ y$. Therefore it suffices to show

$$\{R\ y\ y, ((\forall_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z) \to (\neg_{\mathcal{G}} R\ y\ y) \to (R\ y\ y) \to \bot_{\mathcal{N}}), (\forall_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z)\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$$

We know $\forall_{\mathcal{GKP}} z.\neg_{\mathcal{G}} R\ z\ z = \forall z.R\ z\ z \to \neg_{\mathcal{G}}\neg_{\mathcal{G}}\neg_{\mathcal{G}} R\ z\ z$. By instantiating this with y we get $R\ y\ y \to \neg_{\mathcal{G}}\neg_{\mathcal{G}}\neg_{\mathcal{G}} R\ y\ y$ and by using the assumption $R\ y\ y$ we know $\neg_{\mathcal{G}}\neg_{\mathcal{G}}\neg_{\mathcal{G}} R\ y\ y$. Using Lemma 4.2.1 with the assumption $R\ y\ y$ we know $\neg_{\mathcal{G}}\neg_{\mathcal{G}} R\ y\ y$. By $\neg_{\mathcal{G}}\neg_{\mathcal{G}}\neg_{\mathcal{G}} R\ y\ y$ and $\neg_{\mathcal{G}}\neg_{\mathcal{G}} R\ y\ y$ we can infer $\bot_{\mathcal{N}}$.

Definition 8.5.5 (The Branch Transformation $\Psi^*_{\mathcal{GKP}}$)**.** The branch transformation $\Psi^*_{\mathcal{GKP}}$ extends $\Psi_{\mathcal{GKP}}$ and is defined as follows:

$$\text{For any branch A } : \Psi^*_{\mathcal{GKP}}(A) = \{\Psi_{\mathcal{GKP}}(s) | s \in A\} \cup \{R^\sigma x\ x \mid x : \sigma \text{ and } x \in FV^*(A)\}$$

Lemma 8.5.6. The branch transformation $\Psi^*_{\mathcal{GKP}}$ respects the following rules:

Closed$\bot$, Closed$\neg\top$, Closed, Cut$_{res}$, DNeg, And, NegAnd, Or, NegOr, Imp, NegImp

Proof. Follows by Lemmas 6.2.3, 6.3.6, 6.2.4, 6.2.7, 6.2.5, 6.3.7, 6.3.9, 6.3.8, 6.3.10, 6.2.6, and 6.2.1, respectively. Note that the proof that $\Psi^*_{\mathcal{GKP}}$ respects the Cut$_{res}$ rule (using Lemma 6.2.7) relies on the restriction imposed by Cut$_{res}$: any term s that it adds to a branch is admissible and therefore we know $\vdash_{\mathcal{N}} R\ x\ x$ for each x that is free in s but not in the branch. □

Lemma 8.5.7 (ClosedSym)**.** $\vdash_{\mathcal{N}} \forall^\sigma x\ y.(R^\sigma x\ y) \to \neg_{\mathcal{G}}(R^\sigma y\ x) \to \bot_{\mathcal{N}}$

Lemma 8.5.8 (DeMorgan$\forall$)**.** $\vdash_{\mathcal{N}} \forall^{\sigma\to o} f.((\exists^\sigma_{\mathcal{GKP}}(\lambda x.\neg_{\mathcal{G}} f\ x)) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(\forall^\sigma_{\mathcal{GKP}} f)) \to \bot_{\mathcal{N}}$

Lemma 8.5.9 (DeMorgan$\exists$)**.** $\vdash_{\mathcal{N}} \forall^{\sigma\to o} f.((\forall^\sigma_{\mathcal{GKP}}(\lambda x.\neg_{\mathcal{G}} f\ x)) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(\exists^\sigma_{\mathcal{GKP}} f)) \to \bot_{\mathcal{N}}$

Lemma 8.5.10 (Bool=)**.** $\vdash_{\mathcal{N}} \forall^o p\ q.(p \to q \to \bot_{\mathcal{N}}) \to ((\neg_{\mathcal{G}} p) \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}) \to (R^o p\ q) \to \bot_{\mathcal{N}}$

Lemma 8.5.11 (BoolExt)**.** $\vdash_{\mathcal{N}} \forall^o p\ q.(p \to (\neg_{\mathcal{G}} q) \to \bot_{\mathcal{N}}) \to (q \to (\neg_{\mathcal{G}} p) \to \bot_{\mathcal{N}}) \to \neg_{\mathcal{G}}(R^o p\ q) \to \bot_{\mathcal{N}}$

Lemma 8.5.12 (Con)**.** $\vdash_{\mathcal{N}} \forall^\iota s\ t\ u\ v.(\neg_{\mathcal{G}}(R^\iota s\ u) \to \neg_{\mathcal{G}}(R^\iota t\ u) \to \bot_{\mathcal{N}}) \to (\neg_{\mathcal{G}}(R^\iota s\ v) \to \neg_{\mathcal{G}}(R^\iota t\ v) \to \bot_{\mathcal{N}}) \to (R^\iota s\ t) \to \neg_{\mathcal{G}}(R^\iota u\ v) \to \bot_{\mathcal{N}}$

Lemma 8.5.13 (Exists)**.** $\vdash_{\mathcal{N}} \forall^{\sigma o} f.(\forall^\sigma x.(R^\sigma x\ x) \to (f\ x) \to \bot_{\mathcal{N}}) \to (\exists^\sigma_{\mathcal{GKP}} f) \to \bot_{\mathcal{N}}$

We provide a proof that $\Psi^*_{\mathcal{GKP}}$ respects the Exists rule using Lemma 8.5.13 because it is not straightforward.

Lemma 8.5.14. The branch transformation $\Psi^*_{\mathcal{GKP}}$ respects the Exists rule.

Proof. Let s be a term, A be a branch containing $\exists s$, and y be a fresh variable.
Assume that

$$\Psi^*_{\mathcal{GKP}}(A \cup \{\lceil s\ y\rceil^\beta\}) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

We want to show that

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

Note that if y occurs free in $\lceil s\ y\rceil^\beta$ then

$$\Psi^*_{\mathcal{GKP}}(A \cup \{\lceil s\ y\rceil^\beta\}) = \Psi^*_{\mathcal{GKP}}(A) \cup \{\Psi_{\mathcal{GKP}}(\lceil s\ y\rceil^\beta)\} \cup \{R\ y\ y\},$$

otherwise

$$\Psi^*_{\mathcal{GKP}}(A \cup \{\lceil s\ y\rceil^\beta\}) = \Psi^*_{\mathcal{GKP}}(A) \cup \{\Psi_{\mathcal{GKP}}(\lceil s\ y\rceil^\beta)\}.$$

By applying the $\forall E$ rule with $\Psi_{\mathcal{GKP}}(s)$ to Lemma 8.5.13 we get

$$\vdash_{\mathcal{N}} (\forall^\sigma x.(R^\sigma x\ x) \to ((\Psi_{\mathcal{GKP}}(s))\ x) \to \bot_{\mathcal{N}}) \to (\exists^\sigma_{\mathcal{GKP}}(\Psi_{\mathcal{GKP}}(s))) \to \bot_{\mathcal{N}}.$$

By using the wk rule we can get

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} (\forall^\sigma x.(R^\sigma x\ x) \to ((\Psi_{\mathcal{GKP}}(s))\ x) \to \bot_{\mathcal{N}}) \to (\exists^\sigma_{\mathcal{GKP}}(\Psi_{\mathcal{GKP}}(s))) \to \bot_{\mathcal{N}}.$$

By the $\to E$ rule we know that given a proof of

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} (\forall^\sigma x.(R^\sigma x\ x) \to ((\Psi_{\mathcal{GKP}}(s))\ x) \to \bot_{\mathcal{N}})$$

we obtain a proof of

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} (\exists^\sigma_{\mathcal{GKP}}(\Psi_{\mathcal{GKP}}(s))) \to \bot_{\mathcal{N}}.$$

$\Psi_{\mathcal{GKP}}$ is compositional, so

$$\exists^\sigma_{\mathcal{GKP}}(\Psi_{\mathcal{GKP}}(s)) = \Psi_{\mathcal{GKP}}(\exists s).$$

Since $\exists s \in A$, we know $\Psi_{\mathcal{GKP}}(\exists s) \in \Psi^*_{\mathcal{GKP}}(A)$ and thus $\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \Psi_{\mathcal{GKP}}(\exists s)$ by the hy rule.
By applying the $\to E$ rule we obtain

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

It remains to show that

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} (\forall^\sigma x.(R^\sigma x\ x) \to ((\Psi_{\mathcal{GKP}}(s))\ x) \to \bot_{\mathcal{N}}).$$

By assumption and possibly the wk rule we know

$$\Psi^*_{\mathcal{GKP}}(A) \cup \{\Psi_{\mathcal{GKP}}(\lceil s\ y\rceil^\beta)\} \cup \{R\ y\ y\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

By applying the $\to I$ rule twice and then the $\forall I$ rule, we obtain

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} (\forall^\sigma x.(R^\sigma x\ x) \to ((\Psi_{\mathcal{GKP}}(s))\ x) \to \bot_{\mathcal{N}}).$$

Note that in the last step we implicitly make use of the fact that $\Psi_{\mathcal{GKP}}$ is compositional, that it respects beta, and that $y \notin FV^*(\Psi^*_{\mathcal{GKP}}(A))$. □

Lemma 8.5.15 (Forall_{res}). $\vdash_{\mathcal{N}} \forall^{\sigma o} f.\forall^\sigma t.\neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^\sigma t\ t) \to ((f\ t) \to \bot_{\mathcal{N}}) \to (\forall^\sigma_{\mathcal{GKP}} f) \to \bot_{\mathcal{N}}$

Proving that $\Psi^*_{\mathcal{GKP}}$ respects the Forall_{res} rule using Lemma 8.5.15 is not straightforward and therefore we provide the proof. It is similar to the proof that $\Psi^*_{\mathcal{GKP}}$ respects the $\mathsf{Func}{=}_{res}$ rule using Lemma 8.5.18, which is therefore omitted. Moreover, the proof that $\Psi^*_{\mathcal{GKP}}$ respects the rule $\mathsf{Closed}{\neq}$ using Lemma 8.5.19 is also similar but simpler, since the $\mathsf{Closed}{\neq}$ rule does not introduce new terms, and therefore is also omitted.

Lemma 8.5.16. The branch transformation $\Psi^*_{\mathcal{GKP}}$ respects the Forall_{res} rule.

Proof. Let s be a term, A be a branch containing $\forall s$, and t be a term admissible for A. Assume that

$$\Psi^*_{\mathcal{GKP}}(A \cup \{\lceil s\ t\rceil^\beta\}) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

We want to show that

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

Note that

$$\Psi^*_{\mathcal{GKP}}(A \cup \{\lceil s\ t\rceil^\beta\}) \subseteq \Psi^*_{\mathcal{GKP}}(A) \cup \{\Psi_{\mathcal{GKP}}(\lceil s\ t\rceil^\beta)\} \cup \{R\ x\ x \mid x \in FV(t)\}.$$

By applying the $\forall E$ rule twice to Lemma 8.5.15 with $\Psi_{\mathcal{GKP}}(s)$ and $\Psi_{\mathcal{GKP}}(t)$ respectively and then using the wk rule we get $\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^\sigma \Psi_{\mathcal{GKP}}(t)\ \Psi_{\mathcal{GKP}}(t)) \to ((\Psi_{\mathcal{GKP}}(s)\ \Psi_{\mathcal{GKP}}(t)) \to \bot_{\mathcal{N}}) \to (\forall^\sigma_{\mathcal{GKP}} \Psi_{\mathcal{GKP}}(s)) \to \bot_{\mathcal{N}}$.
By the $\to E$ rule we know that given proofs of

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^\sigma\ \Psi_{\mathcal{GKP}}(t)\ \Psi_{\mathcal{GKP}}(t))$$

and

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} (\Psi_{\mathcal{GKP}}(s)\ \Psi_{\mathcal{GKP}}(t)) \to \bot_{\mathcal{N}}$$

we obtain a proof of

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} (\forall^\sigma_{\mathcal{GKP}} \Psi_{\mathcal{GKP}}(s)) \to \bot_{\mathcal{N}}.$$

Since $\Psi_{\mathcal{GKP}}$ is compositional,

$$(\forall^\sigma_{\mathcal{GKP}} \Psi_{\mathcal{GKP}}(s)) = \Psi_{\mathcal{GKP}}(\forall s).$$

Since $\forall s \in A$, we know $\Psi_{\mathcal{GKP}}(\forall s) \in \Psi^*_{\mathcal{GKP}}(A)$ thus$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \Psi_{\mathcal{GKP}}(\forall s)$ by the hy rule. Hence, by applying the $\to E$ rule we obtain

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

It remains to show two things, namely

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^\sigma\ \Psi_{\mathcal{GKP}}(t)\ \Psi_{\mathcal{GKP}}(t))$$

and

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} (\Psi_{\mathcal{GKP}}(s)\ \Psi_{\mathcal{GKP}}(t)) \to \bot_{\mathcal{N}}.$$

Applying Corollary 8.3.22 with $\Gamma = \Psi^*_{\mathcal{GKP}}(A)$ and making use of the restriction on the Forall_{res} rule we obtain

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} \neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^\sigma\ \Psi_{\mathcal{GKP}}(t)\ \Psi_{\mathcal{GKP}}(t)).$$

By assumption and possibly the wk rule we know

$$\Psi^*_{\mathcal{GKP}}(A) \cup \{\Psi_{\mathcal{GKP}}(\lceil s\ t\rceil^\beta)\} \cup \{R\ x\ x \mid x \in FV(t)\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}.$$

We know that $\Psi_{\mathcal{GKP}}$ is compositional and respects beta, therefore by applying the $\rightarrow I$ rule we obtain

$$\Psi^*_{\mathcal{GKP}}(A) \cup \{R\ x\ x \mid x \in FV(t)\} \vdash_{\mathcal{N}} (\Psi_{\mathcal{GKP}}(s)\ \Psi_{\mathcal{GKP}}(t)) \rightarrow \bot_{\mathcal{N}}.$$

By the restriction on the Forall_{res} rule we know $\forall x \in FV(t) : \Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} R\ x\ x$. By repetitive applications of the $\rightarrow I$ rule and the $\rightarrow E$ rule we obtain

$$\Psi^*_{\mathcal{GKP}}(A) \vdash_{\mathcal{N}} (\Psi_{\mathcal{GKP}}(s)\ \Psi_{\mathcal{GKP}}(t)) \rightarrow \bot_{\mathcal{N}}.$$

□

Lemma 8.5.17 (FuncExt). $\vdash_{\mathcal{N}} \forall^{\sigma\tau} kh.\neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^{\sigma\tau} h\ h) \rightarrow (\forall^{\sigma} x.(R^{\sigma} x\ x) \rightarrow \neg_{\mathcal{G}}(R^{\tau}(k\ x)(h\ x)) \rightarrow \bot_{\mathcal{N}}) \rightarrow \neg_{\mathcal{G}}(R^{\sigma\tau} k\ h) \rightarrow \bot_{\mathcal{N}}$

The proof that $\Psi^*_{\mathcal{GKP}}$ respects the FuncExt rule using Lemma 8.5.17 uses arguments similar to the the ones given in the proofs of Lemmas 8.5.14 and 8.5.16. Therefore, we leave the proof for the reader.

Lemma 8.5.18 (Func$=_{res}$). $\vdash_{\mathcal{N}} \forall^{\sigma\tau} k\ h.\forall^{\sigma} t.\neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^{\sigma} t\ t) \rightarrow ((R^{\tau}(k\ t)\ (h\ t)) \rightarrow \bot_{\mathcal{N}}) \rightarrow (R^{\sigma\tau} k\ h) \rightarrow \bot_{\mathcal{N}}$

Lemma 8.5.19 (Closed$\neq$). $\vdash_{\mathcal{N}} \forall^{\sigma} x.\neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^{\sigma} x\ x) \rightarrow \neg_{\mathcal{G}}(R^{\sigma} x\ x) \rightarrow \bot_{\mathcal{N}}$

Lemma 8.5.20 (Mat). $\vdash_{\mathcal{N}} \forall^{\sigma_1\sigma_2\ldots\sigma_n o} p.\forall^{\sigma_1} s_1\ t_1.\forall^{\sigma_2} s_2\ t_2.\ \ldots\ .\forall^{\sigma_n} s_n\ t_n.\neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^{\sigma_1\sigma_2\ldots\sigma_n o} p\ p) \rightarrow (\neg_{\mathcal{G}}(R^{\sigma_1} s_1\ t_1) \rightarrow \bot_{\mathcal{N}}) \rightarrow (\neg_{\mathcal{G}}(R^{\sigma_2} s_2\ t_2) \rightarrow \bot_{\mathcal{N}}) \rightarrow \ldots \rightarrow (\neg_{\mathcal{G}}(R^{\sigma_n} s_n\ t_n) \rightarrow \bot_{\mathcal{N}}) \rightarrow p\ s_1\ s_2 \ldots s_n \rightarrow \neg_{\mathcal{G}}(p\ t_1\ t_2 \ldots t_n) \rightarrow \bot_{\mathcal{N}}$

Lemma 8.5.21 (Dec). $\vdash_{\mathcal{N}} \forall^{\sigma_1\sigma_2\ldots\sigma_n\iota} h.\forall^{\sigma_1} s_1\ t_1.\forall^{\sigma_2} s_2\ t_2.\ \ldots\ .\forall^{\sigma_n} s_n\ t_n.\neg_{\mathcal{G}}\neg_{\mathcal{G}}(R^{\sigma_1\sigma_2\ldots\sigma_n\iota} h\ h) \rightarrow (\neg_{\mathcal{G}}(R^{\sigma_1} s_1\ t_1) \rightarrow \bot_{\mathcal{N}}) \rightarrow (\neg_{\mathcal{G}}(R^{\sigma_2} s_2\ t_2) \rightarrow \bot_{\mathcal{N}}) \rightarrow \ldots \rightarrow (\neg_{\mathcal{G}}(R^{\sigma_n} s_n\ t_n) \rightarrow \bot_{\mathcal{N}}) \rightarrow \neg_{\mathcal{G}}(R^{\iota}(h\ s_1\ s_2 \ldots s_n)(h\ t_1\ t_2 \ldots t_n)) \rightarrow \bot_{\mathcal{N}}$

For simplicity the Coq proofs provided in Appendix C for the Mat and Dec rules are for the unary and the binary cases.

Theorem 8.5.22. The branch transformation $\Psi^*_{\mathcal{GKP}}$ respects all of the rules in $\mathcal{T}_{Full_{res}}$.

Proof. This follows from Lemmas 8.5.6, 8.5.8, 8.5.9, 8.5.10, 8.5.11, 8.5.12, 8.5.13, 8.5.17, 8.5.15, 8.5.18, 8.5.20, 8.5.21, 8.5.19, and 8.5.7. □

Theorem 8.5.23. The Girard-Kuroda-Per transformation $\Psi_{\mathcal{GKP}}$ respects $\mathcal{T}_{Full_{res}}$.

Proof. This follows directly from Theorem 8.5.22. □

Corollary 8.5.24. Let s be a ground formula. If $\{s\}$ is $\mathcal{T}_{Full_{res}}$-refutable, then $\{\Psi_{\mathcal{GKP}}(s)\} \vdash_{\mathcal{N}} \bot_{\mathcal{N}}$.

Proof. The proof follows directly from Corollary 5.2.2 and from Theorem 8.5.23 □

Chapter 9

Conclusion and Future Work

Given a higher-order formula s and a classical extensional tableau proof of s, our aim was to find a formula s' that is semantically equivalent to s and construct an intuitionistic non-extensional ND proof of s'. In order to achieve this goal we introduced the notions of logical transformation and branch transformation. Furthermore, we defined what it means for a branch transformation to respect a tableau rule and for a logical transformation to respect a tableau calculus.

We argued in the introduction of Chapter 5 stating that it is sufficient to consider only closed formulas. Corollary 5.2.2 states that if a logical transformation Ψ respects a tableau calculus $\mathcal{T}$ then Ψ maps any closed $\mathcal{T}$-refutable formula s to an ND-refutable formula $\Psi(s)$. Hence, there is a formula s' that is semantically equivalent to s, namely $s' := (\Psi(s) \to \bot_{\mathcal{N}}) \to \bot_{\mathcal{N}}$, for which there is an intuitionistic non-extensional ND proof. This corollary reduced our problem to giving a logical transformation that respects a complete higher-order tableau calculus.

On the way to finding this logical transformation we obtained a few other interesting results by considering certain fragments of higher-order logic. We introduced the logical transformations $\Psi_{\mathcal{G}}$, $\Psi_{\mathcal{GK}}$, and $\Psi_{\mathcal{GKP}}$ and corresponding branch transformations. We then showed which tableau rules are respected by each of the branch transformations and which are not.

If $\mathcal{T}$ is a tableau calculus containing only rules that are respected by a branch transformation Ψ^* that extends a logical transformation Ψ, then Ψ respects $\mathcal{T}$. We showed that $\Psi_{\mathcal{G}}$ respects the tableau calculus $\mathcal{T}_{\to,\bot}$. Moreover, that the branch transformation trivially extending $\Psi_{\mathcal{G}}$ respects all the tableau rules except for DeMorgan$\forall$, BoolExt, and FuncExt. Thus, $\Psi_{\mathcal{G}}$ respects all tableau calculi that do not include those three rules. In addition we showed that $\Psi_{\mathcal{GK}}$ respects the tableau calculus for elementary type theory $\mathcal{T}_{elem}$. Finally, we showed that $\Psi_{\mathcal{GKP}}$ respects a complete tableau calculus for higher-order logic $\mathcal{T}_{Full_{res}}$. From here we fulfilled our goal.

Several issues are still open for future work. First, we would like to determine the precise relationship between our Girard-Kuroda-Per transformation and the transformation given by Gandy [14]. We want to find out whether they are equivalent up to double negations.

We are also keen to know whether the Girard-Kuroda-Per transformation respects $\mathcal{T}_{Full}$. For this we have to find a different branch transformation and may have to change some definitions or add more definitions.

On the more practical side, we would like to implement a mapping from tableau proofs to narural deduction proof terms. This would enable proof checking the tableau proofs that Jitpro outputs using the proof checker that Coq provides. This implementation could be done using the Coq lemmas that we provide in the appendix which handle the mapping of each of the rules in the tableau calculus.

Bibliography

[1] P. B. Andrews. Resolution in type theory. *J. Symb. Log.*, 36:414–432, 1971.

[2] P. B. Andrews. *An Introduction to Mathematical Logic and Type Theory: To Truth Through Proof*, volume 27 of *Applied Logic Series*. Kluwer Academic Publishers, second edition, 2002.

[3] H. P. Barendregt. Lambda calculi with types. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 2, pages 117–309. Oxford University Press, 1992.

[4] C. Benzmüller, C. E. Brown, and M. Kohlhase. Higher-order semantics and extensionality. *Journal of Symbolic Logic*, 69:1027–1088, 2004.

[5] Y. Bertot and P. Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions*. Texts in Theoretical Computer Science. Springer Verlag, 2004.

[6] C. E. Brown and G. Smolka. Complete cut-free tableaux for equational simple type theory. Technical report, Programming Systems Lab, Saarland University, April 2009.

[7] C. E. Brown and G. Smolka. Extended first-order logic. In T. Nipkow and C. Urban, editors, *TPHOLs 2009*, volume 5674 of *LNCS*. Springer, August 2009.

[8] C. E. Brown and G. Smolka. Terminating tableaux for the basic fragment of simple type theory. In M. Giese and A. Waaler, editors, *TABLEAUX 2009*, volume 5607 of *LNCS (LNAI)*, pages 138–151. Springer, 2009.

[9] F. Cardone and D. C. Oppen. History of lambda-calculus and combinatory logic. *Handbook of the History of Logic, Volume 5*, 2006. `http://www-maths.swan.ac.uk/staff/jrh/papers/jrhhislamweb.pdf`.

[10] A. Church. A formulation of the simple theory of types. *J. Symb. Log.*, 5(1):56–68, 1940.

[11] L. Chwistek. Antynomje logiki formalnej. *Prezegląd Filozoficzny*, 24:164–171, 1921.

[12] P. Engracia and F. Ferreria. The bounded functional interpretation of the double negation shift. To be Published in the Journal of Symbolic Logic.

[13] W. M. Farmer. The seven virtues of simple type theory. Technical report, McMaster University, Dec. 2003.

[14] R. O. Gandy. On the axiom of extensionality–part i. *J. Symb. Log.*, 21(1):36–48, 1956.

[15] G. Gentzen. Untersuchungen über das natürliche Schließen I, II. *Mathematische Zeitschrift*, 39:176–210, 405–431, 1935.

[16] P. Gerhardy. Functional interpretation and modified realizability interpretation of the double-negation shift. In *Logical Approaches to Computational Barriers*, pages 109–118. Second Conference on Computability in Europe, 2006.

[17] J. H. Geuvers. The calculus of constructions and higher order logic. In P. de Groote, editor, *The Curry-Howard Isomorphism*, pages 139–191. Academia, Louvain-la-Neuve (Belgium), 1995.

[18] J.-Y. Girard, P. Taylor, and Y. Lafont. *Proofs and types*. Cambridge University Press, 1989.

[19] V. Glivenko. Sur quelques points de la logique de M. Brouwer. *Bulletins de la classe des sciences*, 15:183–188, 1929.

[20] J. Harrison. HOL Light tutorial (for version 2.20). `http://www.cl.cam.ac.uk/~jrh13/hol-light/tutorial_220.pdf`, 2006.

[21] L. Henkin. Completeness in the theory of types. *Journal of Symbolic Logic*, 15(2):81–91, June 1950.

[22] J. R. Hindley. *Basic Simple Type Theory*, volume 42 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1997.

[23] W. A. Howard. The formula-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 480–490. Academic Press, 1980.

[24] S. Kuroda. Intuitionistische Untersuchgen der formalistischen Logik. *Nagoya Mathematical Journal*, 2:35–47, 1951.

[25] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[26] F. Ramsey. The foundations of mathematics. *Proc. of the London Math. Society, 2nd series*, 25:338–384, 1925.

[27] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908.

[28] G. Smolka and C. E. Brown. Introduction to computational logic. `http://www.ps.uni-sb.de/courses/cl-ss09/script/icl.pdf`, 2009.

[29] A. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, second edition, 2000.

[30] A. N. Whitehead and B. Russell. *Principia Mathematica*. Cambridge University Press, Cambridge, England, 1913. 3 volumes; first edition 1913, second edition 1927.

[31] Intuitionistic logic. `http://plato.stanford.edu/entries/logic-intuitionistic/`.

[32] K. Zdanowski. On second order intuitionistic propositional logic without a universal quantifier. *Journal of Symbolic Logic*, 74:157–167, 2009.

[33] Jitpro: A JavaScript Interactive higher-order Tableau Prover. `http://www.ps.uni-sb.de/jitpro/`.

Appendix A

Girard Transformation

A.1 Defining basic types i, o

```
Parameter i : Type.
Definition o := Prop.
```

A.2 Girard Transformation

```
Definition Bot := ∀ p : o, p.
Definition Neg := fun p : o ⇒ p → Bot.
Definition Top := ∀ p : o, p → p.
Definition Imp := fun p q : o ⇒ p → q.
Definition Forall (sigma : Type) := fun f : (sigma → o) ⇒ (∀ x : sigma, f x).
Definition And (M : o) (N : o) := (∀ p : o, (M → N → p) → p).
Definition Or (M : o) (N : o) := (∀ p : o, (M → p) → (N → p) → p).
Definition Equal (sigma : Type) (M : sigma) (N : sigma)
  := ∀ f : (sigma → o), (f M) → f N.
Definition Exists (sigma : Type) (M : (sigma → o))
  := ∀ p : o, (∀ x : sigma, (M x) → p) → p.
```

A.2.1 Short Hand

```
Definition ForallNeg (sigma : Type)
  := fun f : (sigma->o) ⇒ Forall sigma (fun x : sigma ⇒ (Neg(f x))).
Definition ExistsNeg (sigma : Type) (M : (sigma → o))
  := Exists sigma (fun x : sigma ⇒ (Neg(M x))).
```

A.3 Symmetry of equality

```
Definition Sym (sigma : Type)
  : ∀ x y : sigma, (Equal sigma x y) → (Equal sigma y x)
  := fun x y u f v ⇒ u (fun z ⇒ f z → f x) (fun w ⇒ w) v.
```

A.4 Transitivity of equality

```
Definition Tra (sigma : Type)
  : ∀ x y z : sigma, (Equal sigma x y) → (Equal sigma y z) → (Equal sigma x z)
  := fun x y z u v p w ⇒ v p (u p w).
```

A.5 Negative transitivity

```
Definition TraNeg (sigma : Type)
  : ∀ x y z : sigma, (Equal sigma x y) → Neg(Equal sigma x z) → Neg(Equal sigma y z)
  := fun x y z u v w ⇒ v (Tra sigma x y z u w).
```

A.6 Lemmas

A.6.1 Closed False Rule

```
Definition closedfalserule : Bot → Bot := fun u ⇒ u.
```

A.6.2 Closed Not True Rule

```
Definition closednottruerule : (Neg Top) → Bot := fun u ⇒ u (fun p v ⇒ v).
```

A.6.3 Closed Rule

```
Definition closedrule : ∀ p : o, p → (Neg p) → Bot
  := fun p u v ⇒ (v u).
```

A.6.4 Closed Neg Equal Rule

```
Definition closednegequalrule (sigma : Type)
  : ∀ (s : sigma), Neg(Equal sigma s s) → Bot
  := fun p u ⇒ u (fun f v ⇒ v).
```

A.6.5 Closed Symmetric Rule

```
Definition closedsymrule (sigma : Type)
  : ∀ (s t : sigma), (Equal sigma s t) → Neg(Equal sigma t s) → Bot
  := fun s t u v ⇒ v (Sym sigma s t u).
```

A.6.6 Double Negation Rule

```
Definition dnegrule
  : ∀ p : o, (p → Bot) → Neg(Neg p) → Bot
  := fun p u v ⇒ v u.
```

A.6.7 Cut Rule

```
Definition cutrule
  : ∀ p : o, (p → Bot) → ((Neg p) → Bot) → Bot
  := fun p u v ⇒ v u.
```

A.6.8 Implication Rule

```
Definition imprule
  : ∀ p q : o, ((Neg p) → Bot) → (q → Bot) → (Imp p q) → Bot
  := fun p q u v w ⇒ u (fun u1 : p ⇒ v (w u1)).
```

A.6.9 Negative Implication Rule

```
Definition negimprule
  : ∀ p q : o, (p → (Neg q) → Bot) → (Neg (Imp p q)) → Bot
  := fun p q u v ⇒ v (fun w1 : p ⇒ u w1 (fun z : q ⇒ v (fun w2 : p ⇒ z)) q).
```

A.6.10 And Rule

```
Definition andrule
  : ∀ p q : o, (p → q → Bot) → (And p q) → Bot
  := fun p q u v ⇒ v Bot u.
```

A.6.11 Or Rule

```
Definition orrule
  : ∀ p q : o, (p → Bot) → (q → Bot) → (Or p q) → Bot
  := fun p q u v w ⇒ w Bot u v.
```

A.6.12 Neg And Rule

```
Definition negandrule
  : ∀ p q : o, ((Neg p) → Bot) → ((Neg q) → Bot) → Neg(And p q) → Bot
  := fun p q u v w ⇒ u (fun u1 ⇒ v (fun u2 ⇒ w (fun p u3 ⇒ u3 u1 u2))).
```

A.6.13 Neg Or Rule

```
Definition negorrule
  : ∀ p q : o, ((Neg p) → (Neg q) → Bot) → Neg(Or p q) → Bot
  := fun p q u v ⇒
       u (fun w ⇒ v (fun r u1 u2 ⇒ u1 w))(fun w ⇒ v (fun r u1 u2 ⇒ u2 w)).
```

A.6.14 Forall Rule

```
Definition forallrule (sigma : Type)
  : ∀ (f : sigma → o) (t : sigma), ((f t) → Bot) → (Forall sigma f) → Bot
  := fun f t u v ⇒ u (v t).
```

A.6.15 Exists Rule

```
Definition existsrule (sigma : Type)
  : ∀ f : (sigma → o), (∀ x : sigma, (f x) → Bot) → (Exists sigma f) → Bot
  := fun f u v ⇒ v Bot u.
```

A.6.16 DeMorgan Exists Rule

```
Definition demorganexistsrule (sigma : Type)
  : ∀ (f : sigma → o), ((ForallNeg sigma f) → Bot) → (Neg (Exists sigma f)) → Bot
  := fun f u v ⇒ u (fun y w1 ⇒ v (fun r w2 ⇒ w2 y w1)).
```

A.6.17 Boolean Equality Rule

```
Definition booleqrule
  : ∀ (p q : o), (p → q → Bot) → ((Neg p) → (Neg q) → Bot) → (Equal o p q) → Bot
  := fun p q u1 u2 u3 ⇒
       u2 (fun v ⇒ u1 v (u3 (fun w ⇒ w) v))
          (u3 (fun w ⇒ Neg w) (fun v ⇒ u1 v (u3 (fun w ⇒ w) v))).
```

A.6.18 Leibniz Rule - not used in the Thesis

```
Definition leibnizrule (sigma : Type)
  : ∀ (x y : sigma), ((Forall (sigma → o) (fun f ⇒ f x → f y)) → Bot) → (Equal sigma x y)
→ Bot
  := fun x y u v ⇒ u v.
```

A.6.19 Functional Equality Rule

```
Definition funceqrule (sigma tau : Type)
  : ∀ (k h : (sigma → tau)) (t : sigma),
    ((Equal tau (k t) (h t)) → Bot) → (Equal (sigma → tau) k h) → Bot
  := fun k h t u1 u2 ⇒
       u1 (u2 (fun r ⇒ Equal tau (k t) (r t)) (fun f v ⇒ v)).
```

A.6.20 Mating Rule - 2 arguments

```
Definition matingrule_2 (sigma tau : Type)
  : ∀ (f : (sigma ->tau → o)) (s1 t1 : sigma) (s2 t2 : tau),
    (Neg(Equal sigma s1 t1) → Bot) → (Neg(Equal tau s2 t2) → Bot) →
    (f s1 s2) → Neg(f t1 t2) → Bot
  := fun f s1 t1 s2 t2 u1 u2 u3 u4 ⇒
       u1 (fun v1 ⇒
             u2 (fun v2 ⇒ u4 (v2 (fun x ⇒ f t1 x) (v1 (fun x ⇒ f x s2) u3)))).
```

A.6.21 Decomposition Rule - 2 arguments

```
Definition decompositionrule_2 (sigma tau : Type)
  : ∀ (h : (sigma → tau → i)) (s1 t1 : sigma) (s2 t2 : tau),
    (Neg(Equal sigma s1 t1) → Bot) → (Neg(Equal tau s2 t2) → Bot) →
```

```
    Neg(Equal i (h s1 s2) (h t1 t2)) → Bot
  := fun h s1 t1 s2 t2 u1 u2 u3 ⇒
       u1 (fun v1 ⇒
             u2 (fun v2 ⇒
                   u3 (fun p v3 ⇒
                         (v2 (fun x ⇒ p (h t1 x)) (v1 (fun x ⇒ p (h x s2)) v3))))).
```

A.6.22 Confrontation Rule

```
Definition confrontationrule
  : ∀ (s1 t1 s2 t2 : i), (Neg(Equal i s1 s2) → Neg(Equal i t1 s2) → Bot) →
    (Neg(Equal i s1 t2) → Neg(Equal i t1 t2)→ Bot) →
    (Equal i s1 t1) → Neg(Equal i s2 t2) → Bot
  := fun s1 t1 s2 t2 u1 u2 u3 u4 ⇒
       u1 (fun v1 ⇒
             u2 (TraNeg i s2 s1 t2 (Sym i s1 s2 v1) u4)
                (TraNeg i s1 t1 t2 u3 (TraNeg i s2 s1 t2 (Sym i s1 s2 v1) u4)))
          (TraNeg i s1 t1 s2 u3 (fun v1 ⇒
                                   u2 (TraNeg i s2 s1 t2 (Sym i s1 s2 v1) u4)
                                      (TraNeg i s1 t1 t2 u3
                                                (TraNeg i s2 s1 t2
                                                        (Sym i s1 s2 v1) u4)))).
```

Appendix B

Girard-Kuroda Transformation

B.1 Defining basic types i, o

```
Parameter i : Type.
Definition o := Prop.
```

B.2 Girard-Kuroda Transformation

```
Definition Bot := ∀ p : o, p.
Definition Neg := fun p : o ⇒ p → Bot.
Definition Top := ∀ p : o, p → p.
Definition Imp := fun p q : o ⇒ p → q.
Definition Forall (sigma : Type)
  := fun f : (sigma → o) ⇒ (∀ x : sigma, Neg(Neg (f x))).
Definition And (M : o) (N : o) := (∀ p : o, (M → N → p) → p).
Definition Or (M : o) (N : o) := (∀ p : o, (M → p) → (N → p) → p).
Definition Equal (sigma : Type) (M : sigma) (N : sigma)
  := ∀ f : (sigma → o), (f M) → f N.
Definition Exists (sigma : Type) (M : (sigma → o))
  := ∀ p : o, (∀ x : sigma, (M x) → p) → p.
```

B.2.1 Short Hand

```
Definition ForallNeg (sigma : Type)
  := fun f : (sigma → o) ⇒ Forall sigma (fun x : sigma ⇒ (Neg(f x))).
Definition ExistsNeg (sigma : Type) (M : (sigma → o))
  := Exists sigma (fun x : sigma ⇒ (Neg(M x))).
```

B.3 Symmetry of equality

```
Definition Sym (sigma : Type)
  : ∀ x y : sigma, (Equal sigma x y) → (Equal sigma y x)
  := fun x y u f v ⇒ u (fun z ⇒ f z → f x) (fun w ⇒ w) v.
```

B.4 Transitivity of equality

```
Definition Tra (sigma : Type)
  : ∀ x y z : sigma, (Equal sigma x y) → (Equal sigma y z) → (Equal sigma x z)
  := fun x y z u v p w ⇒ v p (u p w).
```

B.5 Negative transitivity

```
Definition TraNeg (sigma : Type)
  : ∀ x y z : sigma, (Equal sigma x y) → Neg(Equal sigma x z) → Neg(Equal sigma y z)
  := fun x y z u v w ⇒ v (Tra sigma x y z u w).
```

B.6 Lemmas

B.6.1 Closed False Rule

```
Definition closedfalserule : Bot → Bot := fun u ⇒ u.
```

B.6.2 Closed Not True Rule

```
Definition closednottruerule : (Neg Top) → Bot := fun u ⇒ u (fun p v ⇒ v).
```

B.6.3 Closed Rule

```
Definition closedrule :
  ∀ p : o, p → (Neg p) → Bot
  := fun p u v ⇒ (v u).
```

B.6.4 Closed Neg Equal Rule

```
Definition closednegequalrule (sigma : Type)
  : ∀ (s : sigma), Neg(Equal sigma s s) → Bot
  := fun p u ⇒ u (fun f v ⇒ v).
```

B.6.5 Closed Symmetric Rule

```
Definition closedsymrule (sigma : Type)
  : ∀ (s t : sigma), (Equal sigma s t) → Neg(Equal sigma t s) → Bot
  := fun s t u v ⇒ v (Sym sigma s t u).
```

B.6.6 Double Negation Rule

```
Definition dnegrule
  : ∀ p : o, (p → Bot) → Neg(Neg p) → Bot
  := fun p u v ⇒ v u.
```

B.6.7 Cut Rule

```
Definition cutrule
  : ∀ p : o, (p → Bot) → ((Neg p) → Bot) → Bot
  := fun p u v ⇒ v u.
```

B.6.8 Implication Rule

```
Definition imprule
  : ∀ p q : o, ((Neg p) → Bot) → (q → Bot) → (Imp p q) → Bot
  := fun p q u v w ⇒ u (fun u1 : p ⇒ v (w u1)).
```

B.6.9 Negative Implication Rule

```
Definition negimprule
  : ∀ p q : o, (p → (Neg q) → Bot) → (Neg (Imp p q)) → Bot
  := fun p q u v ⇒ v (fun w1 : p ⇒ u w1 (fun z : q ⇒ v (fun w2 : p ⇒ z)) q).
```

B.6.10 And Rule

```
Definition andrule
  : ∀ p q : o, (p → q → Bot) → (And p q) → Bot
  := fun p q u v ⇒ v Bot u.
```

B.6.11 Or Rule

```
Definition orrule
  : ∀ p q : o, (p → Bot) → (q → Bot) → (Or p q) → Bot
  := fun p q u v w ⇒ w Bot u v.
```

B.6.12 Neg And Rule

```
Definition negandrule
  : ∀ p q : o, ((Neg p) → Bot) → ((Neg q) → Bot) → Neg(And p q) → Bot
  := fun p q u v w ⇒ u (fun u1 ⇒ v (fun u2 ⇒ w (fun p u3 ⇒ u3 u1 u2))).
```

B.6.13 Neg Or Rule

```
Definition negorrule
  : ∀ p q : o, ((Neg p) → (Neg q) → Bot) → Neg(Or p q) → Bot
  := fun p q u v ⇒
       u (fun w ⇒ v (fun r u1 u2 ⇒ u1 w))(fun w ⇒ v (fun r u1 u2 ⇒ u2 w)).
```

B.6.14 Forall Rule

```
Definition forallrule (sigma : Type)
  : ∀ (f : sigma → o) (t : sigma), ((f t) → Bot) → (Forall sigma f) → Bot
  := fun f t u v ⇒ v t u.
```

B.6.15 DeMorgan Forall Rule

```
Definition demorganforallrule (sigma : Type)
  : ∀ (f : sigma → o), ((ExistsNeg sigma f) → Bot) → (Neg (Forall sigma f)) → Bot
  := fun f u v ⇒ v (fun x w ⇒ u ( fun r z ⇒ z x w)).
```

B.6.16 Exists Rule

```
Definition existsrule (sigma : Type)
  : ∀ f : (sigma → o), (∀ x : sigma, (f x) → Bot) → (Exists sigma f) → Bot
  := fun f u v ⇒ v Bot u.
```

B.6.17 DeMorgan Exists Rule

```
Definition demorganexistsrule (sigma : Type)
  : ∀ (f : sigma → o), ((ForallNeg sigma f) → Bot) → (Neg (Exists sigma f)) → Bot
  := fun f u v ⇒ u (fun y w1 ⇒ w1 (fun w2 ⇒ v (fun r w3 ⇒ w3 y w2))).
```

B.6.18 Boolean Equality Rule

```
Definition booleqrule
  : ∀ (p q : o), (p → q → Bot)→ ((Neg p) → (Neg q) → Bot) → (Equal o p q) → Bot
  := fun p q u1 u2 u3 ⇒
       u2 (fun v ⇒ u1 v (u3 (fun w ⇒ w) v))
          (u3 (fun w ⇒ Neg w) (fun v ⇒ u1 v (u3 (fun w ⇒ w) v))).
```

B.6.19 Leibniz Rule

```
Definition leibnizrule (sigma : Type)
  : ∀ (x y : sigma), ((Forall (sigma → o) (fun f ⇒ f x → f y)) → Bot) → (Equal sigma x y)
→ Bot
  := fun x y u v ⇒ u (fun f u1 ⇒ u1 (v f)).
```

B.6.20 Functional Equality Rule

```
Definition funceqrule (sigma tau : Type)
  : ∀ (k h : (sigma → tau)) (t : sigma), ((Equal tau (k t) (h t)) → Bot) →
    (Equal (sigma → tau) k h) → Bot
  := fun k h t u1 u2 ⇒ u1 (u2 (fun r ⇒ Equal tau (k t) (r t)) (fun f v ⇒ v)).
```

B.6.21 Mating Rule - 2 arguments

```
Definition matingrule_2 (sigma tau : Type)
  : ∀ (f : (sigma → tau → o)) (s1 t1 : sigma) (s2 t2 : tau),
    (Neg(Equal sigma s1 t1) → Bot) → (Neg(Equal tau s2 t2) → Bot) →
    (f s1 s2) → Neg(f t1 t2) → Bot
  := fun f s1 t1 s2 t2 u1 u2 u3 u4 ⇒
       u1 (fun v1 ⇒
             u2 (fun v2 ⇒ u4 (v2 (fun x ⇒ f t1 x) (v1 (fun x ⇒ f x s2) u3)))).
```

B.6.22 Decomposition Rule - 2 arguments

```
Definition decompositionrule_2 (sigma tau : Type)
  : ∀ (h : (sigma → tau → i)) (s1 t1 : sigma) (s2 t2 : tau),
    (Neg(Equal sigma s1 t1) → Bot) → (Neg(Equal tau s2 t2) → Bot) →
    Neg(Equal i (h s1 s2) (h t1 t2)) → Bot
  := fun h s1 t1 s2 t2 u1 u2 u3 ⇒
       u1 (fun v1 ⇒
             u2 (fun v2 ⇒
                   u3 (fun p v3 ⇒
                         v2 (fun x ⇒ p (h t1 x)) (v1 (fun x ⇒ p (h x s2)) v3)))).
```

B.6.23 Confrontation Rule

```
Definition confrontationrule
  : ∀ (s1 t1 s2 t2 : i), (Neg(Equal i s1 s2) → Neg(Equal i t1 s2) → Bot) →
    (Neg(Equal i s1 t2) → Neg(Equal i t1 t2)→ Bot) → (Equal i s1 t1) →
    Neg(Equal i s2 t2) → Bot
  := fun s1 t1 s2 t2 u1 u2 u3 u4 ⇒
       u1 (fun v1 ⇒ u2 (TraNeg i s2 s1 t2 (Sym i s1 s2 v1) u4)
                       (TraNeg i s1 t1 t2 u3 (TraNeg i s2 s1 t2
                                                     (Sym i s1 s2 v1) u4)))
          (TraNeg i s1 t1 s2 u3 (fun v1 ⇒
                                   u2 (TraNeg i s2 s1 t2 (Sym i s1 s2 v1) u4)
                                      (TraNeg i s1 t1 t2 u3
                                              (TraNeg i s2 s1 t2
                                                      (Sym i s1 s2 v1) u4)))).
```

Appendix C

Girard-Kuroda-Per Transformation

C.1 Defining basic types ι, o

```
Parameter i : Type.
Definition o := Prop.
```

C.2 Girard-Kuroda-Per Transformation

```
Definition Bot := ∀ p : o, p.
Definition Neg := fun p : o ⇒ p → Bot.
Definition Top := ∀ p : o, p → p.
Definition Imp := fun p q : o ⇒ p → q.
Definition And (M : o) (N : o) := (∀ p : o, (M → N → p) → p).
Definition Or (M : o) (N : o) := (∀ p : o, (M → p) → (N → p) → p).
Definition R_o (M : o) (N : o) := (And (Imp M N) (Imp N M)).
Definition R_i (M : i) (N : i) := (∀ p : i → o, (p M) → (p N)).
Definition R_ar (sigma tau : Type) (R_sigma : sigma → sigma → o)
  (R_tau : tau → tau → o) (M : sigma → tau) (N : sigma → tau)
  := ∀ x y : sigma, R_sigma x y → Neg(Neg(R_tau (M x) (N y))).
Definition Forall (sigma : Type) (R_sigma : sigma → sigma → o)
  := fun f : (sigma → o) ⇒ ∀ t : sigma, Imp (R_sigma t t) (Neg(Neg(f t))).
Definition Exists (sigma : Type) (R_sigma : sigma → sigma → o)
  := fun f : (sigma → o) ⇒ ∀ p : o, (∀ x : sigma, (R_sigma x x) → (f x) → p) → p.
```

C.2.1 Short Hand

```
Definition ForallNeg (sigma : Type) (R_sigma : sigma → sigma → o)
  := fun f:(sigma → o) ⇒ Forall sigma R_sigma (fun x:sigma ⇒ (Neg(f x))).
Definition ExistsNeg (sigma : Type) (R_sigma : sigma → sigma → o) (M : (sigma → o))
  := Exists sigma R_sigma (fun x:sigma ⇒ (Neg(M x))).
```

C.3 Some Definitions

C.3.1 Definition of SymNeg

```
Definition SymNeg (sigma : Type) (R_sigma : sigma → sigma → o)
  : (∀ x y : sigma, (R_sigma x y) → (R_sigma y x)) →
    (∀ x y : sigma, Neg(R_sigma x y) → Neg(R_sigma y x))
  := fun u x y v w ⇒ v (u y x w).
```

C.3.2 Recursive definition of Sym

```
Definition Sym_o
  : ∀ x y : o, (R_o x y) → (R_o y x)
  := fun x y u r v ⇒ u r (fun u1 u2 ⇒ v u2 u1).
Definition Sym_i
  : ∀ x y : i, (R_i x y) → (R_i y x)
  := fun x y u p v ⇒ u (fun z ⇒ p z → p x) (fun w ⇒ w) v.
Definition Sym_ar (sigma tau : Type) (R_sigma : sigma → sigma → o)
  (R_tau : tau → tau → o)
  (Sym_sigma : ∀ (x y : sigma), (R_sigma x y) → (R_sigma y x))
  (Sym_tau : ∀ (x y : tau), (R_tau x y) → (R_tau y x))
  : ∀ (f g : (sigma → tau)),
    (R_ar sigma tau R_sigma R_tau f g) → (R_ar sigma tau R_sigma R_tau g f)
  := fun f g u x y v w ⇒
       u y x (Sym_sigma x y v) (SymNeg tau R_tau Sym_tau (g x) (f y) w).
```

C.3.3 Recursive definition of Tra

```
Definition Tra_o
  : ∀ x y z : o, (R_o x y) → (R_o y z) → (R_o x z)
  := fun x y y u v r w ⇒
       u r (fun u1 u2 ⇒
              v r (fun v1 v2 ⇒
                     w (fun w1 ⇒ v1 (u1 w1)) (fun w2 ⇒ u2 (v2 w2)))).
Definition Tra_i
  : ∀ x y z : i, (R_i x y) → (R_i y z) → (R_i x z)
  := fun x y z u v p w ⇒ v p (u p w).

Definition Tra_ar (sigma tau : Type) (R_sigma : sigma → sigma → o)
  (R_tau : tau → tau → o)
  (Sym_sigma : ∀ (x y : sigma), (R_sigma x y) → (R_sigma y x))
  (Tra_sigma : ∀ x y z : sigma, (R_sigma x y) → (R_sigma y z) → (R_sigma x z))
  (Tra_tau : ∀ x y z : tau, (R_tau x y) → (R_tau y z) → (R_tau x z))
  : ∀ (f g h : (sigma → tau)), (R_ar sigma tau R_sigma R_tau f g) →
    (R_ar sigma tau R_sigma R_tau g h) → (R_ar sigma tau R_sigma R_tau f h)
  := fun f g h u v x y w1 w2 ⇒
       u x y w1 (fun u1 ⇒ v y y (Tra_sigma y x y (Sym_sigma x y w1) w1)
         (fun v1 ⇒ w2 (Tra_tau (f x) (g y) (h y) u1 v1))).
```

C.3.4 Definition of TraNeg relative to Tra

```
Definition TraNeg (sigma : Type) (R_sigma : sigma → sigma → o)
  (Tra_sigma : ∀ x y z : sigma, (R_sigma x y) → (R_sigma y z) → (R_sigma x z))
  : ∀ x y z : sigma, (R_sigma x y) → Neg(R_sigma x z) → Neg(R_sigma y z)
  := fun x y z u v w ⇒ v (Tra_sigma x y z u w).
```

C.4 Lemmas

C.4.1 Closed False Rule

```
Definition closedfalserule : Bot → Bot := fun u ⇒ u.
```

C.4.2 Closed Not True Rule

```
Definition closednottruerule : (Neg Top) → Bot := fun u ⇒ u (fun p v ⇒ v).
```

C.4.3 Closed Rule

```
Definition closedrule : ∀ p : o, p → (Neg p) → Bot := fun p u v ⇒ (v u).
```

C.4.4 Closed Neg Equal Rule

```
Definition closednegequalrule (sigma : Type) (R_sigma : sigma → sigma → o)
  : ∀ (s : sigma), Neg(Neg(R_sigma s s)) → Neg(R_sigma s s) → Bot
  := fun s u v ⇒ u v.
```

C.4.5 Closed Symmetric Rule

```
Definition closedsymrule (sigma : Type) (R_sigma : sigma → sigma → o)
  (Sym_sigma : ∀ (x y : sigma), (R_sigma x y) → (R_sigma y x))
  : ∀ (s t : sigma), (R_sigma s t) → Neg(R_sigma t s) → Bot
  := fun s t u v ⇒ v (Sym_sigma s t u).
```

C.4.6 Double Negation Rule

```
Definition dnegrule
  : ∀ (p : o), (p → Bot) → Neg(Neg p) → Bot
  := fun p u v ⇒ v u.
```

C.4.7 Restricted Cut Rule

```
Definition cutrule
  : ∀ (p : o), (p → Bot) → ((Neg p) → Bot) → Bot
  := fun p u v ⇒ v u.
```

C.4.8 Implication Rule

```
Definition imprule
  : ∀ (p q : o), ((Neg p) → Bot) → (q → Bot) → (Imp p q) → Bot
  := fun p q u v w ⇒ u (fun u1 : p ⇒ v (w u1)).
```

C.4.9 Negative Implication Rule

```
Definition negimprule
  : ∀ (p q : o), (p → (Neg q) → Bot) → (Neg (Imp p q)) → Bot
  := fun p q u v ⇒ v (fun w1 : p ⇒ u w1 (fun z : q ⇒ v (fun w2 : p ⇒ z)) q).
```

C.4.10 And Rule

```
Definition andrule
  : ∀ (p q : o), (p → q → Bot) → (And p q) → Bot
  := fun p q u v ⇒ v Bot u.
```

C.4.11 Or Rule

```
Definition orrule
  : ∀ (p q : o), (p → Bot) → (q → Bot) → (Or p q) → Bot
  := fun p q u v w ⇒ w Bot u v.
```

C.4.12 Neg And Rule

```
Definition negandrule
  : ∀ (p q : o), ((Neg p) → Bot) → ((Neg q) → Bot) → Neg(And p q) → Bot
  := fun p q u v w ⇒ u (fun u1 ⇒ v (fun u2 ⇒ w (fun p u3 ⇒ u3 u1 u2))).
```

C.4.13 Neg Or Rule

```
Definition negorrule
  : ∀ (p q : o), ((Neg p) → (Neg q) → Bot) → Neg(Or p q) → Bot
  := fun p q u v ⇒
       u (fun w ⇒ v (fun r u1 u2 ⇒ u1 w)) (fun w ⇒ v (fun r u1 u2 ⇒ u2 w)).
```

C.4.14 Restricted Forall Rule

```
Definition forallrule (sigma : Type) (R_sigma : sigma → sigma → o)
  : ∀ (f : sigma → o) (t : sigma), Neg(Neg(R_sigma t t)) → ((f t) → Bot) →
    (Forall sigma R_sigma f) → Bot
  := fun f t u v w ⇒ u (fun u1 ⇒ w t u1 v).
```

C.4.15 DeMorgan Forall Rule

```
Definition demorganforallrule (sigma : Type) (R_sigma : sigma → sigma → o)
  : ∀ (f : sigma → o), ((ExistsNeg sigma R_sigma f) → Bot) →
    (Neg (Forall sigma R_sigma f)) → Bot
  := fun f u v ⇒ v (fun x v1 v2 ⇒ u (fun p u1 ⇒ u1 x v1 v2)).
```

C.4.16 Exists Rule

```
Definition existsrule (sigma : Type) (R_sigma : sigma → sigma → o)
  : ∀ (f : sigma → o), (∀ x : sigma, (R_sigma x x) → (f x) → Bot) →
    (Exists sigma R_sigma f) → Bot
  := fun f u v ⇒ v Bot u.
```

C.4.17 DeMorgan Exists Rule

```
Definition demorganexistsrule (sigma : Type) (R_sigma : sigma → sigma → o)
  : ∀ (f : sigma → o), ((ForallNeg sigma R_sigma f) → Bot) →
    (Neg (Exists sigma R_sigma f)) → Bot
  := fun f u v ⇒ u (fun x u1 u2 ⇒ u2 (fun u3 ⇒ v (fun p v1 ⇒ v1 x u1 u3))).
```

C.4.18 Boolean Equality Rule

```
Definition booleqrule
  : ∀ (p q : o), (p → q → Bot) → ((Neg p) → (Neg q) → Bot) → (R_o p q) → Bot
  := fun p q u v w ⇒
       w Bot (fun u1 u2 ⇒ v (fun v1 ⇒ u v1 (u1 v1)) (fun v2 ⇒ u (u2 v2) v2)).
```

C.4.19 Boolean Extensionality Rule

```
Definition boolextrule
  : ∀ (p q : o), (p → (Neg q) → Bot) → (q → (Neg p) → Bot) → Neg(R_o p q) → Bot
  := fun p q u v w ⇒
       w (fun r w1 ⇒
            w1 (fun z1 ⇒
                  (u z1 (fun z2 ⇒
                           w (fun r1 w2 ⇒ w2 (fun z3 ⇒ z2) (fun z4 ⇒ z1)))) q)
               (fun z1 ⇒
                  (v z1 (fun z2 ⇒
                           w (fun r1 w2 ⇒ w2 (fun z3 ⇒ z1) (fun z4 ⇒ z2)))) p)).
```

C.4.20 Restricted Functional Equality Rule

```
Definition funceqrule (sigma tau : Type) (R_sigma : sigma → sigma → o)
  (R_tau : tau → tau → o)
  : ∀ (k h : (sigma → tau)) (t : sigma), Neg(Neg(R_sigma t t)) →
    ((R_tau (k t) (h t)) → Bot) → (R_ar sigma tau R_sigma R_tau k h) → Bot
  := fun k h t u v w ⇒ u (fun u1 ⇒ w t t u1 v).
```

C.4.21 Functional Extensionality Rule

```
Definition funcextrule (sigma tau : Type) (R_sigma : sigma → sigma → o)
  (R_tau : tau → tau → o)
  (Sym_sigma : ∀ (x y : sigma),(R_sigma x y) → (R_sigma y x))
  (Tra_sigma : ∀ x y z : sigma, (R_sigma x y) → (R_sigma y z) → (R_sigma x z))
  (Tra_tau : ∀ x y z : tau, (R_tau x y) → (R_tau y z) → (R_tau x z))
  : ∀ (k h : (sigma → tau)), Neg(Neg(R_ar sigma tau R_sigma R_tau h h)) →
```

```
    (∀ x : sigma, (R_sigma x x) → Neg(R_tau (k x) (h x)) → Bot) →
    Neg(R_ar sigma tau R_sigma R_tau k h) → Bot
  := fun k h u v w ⇒
       w (fun x y w1 w2 ⇒
            v x (Tra_sigma x y x w1 (Sym_sigma x y w1))
            (fun v1 ⇒
                 u (fun u1 ⇒
                      u1 x y w1 (TraNeg tau R_tau Tra_tau (k x) (h x) (h y) v1 w2)))).
```

C.4.22 Mating Rule - 1 argument

```
Definition matingrule_1 (sigma : Type) (R_sigma : sigma → sigma → o)
  : ∀ (f : (sigma → o)) (s t : sigma), Neg(Neg(R_ar sigma o R_sigma R_o f f)) →
    (Neg(R_sigma s t) → Bot) → (f s) → Neg(f t) → Bot
  := fun f s t u0 u1 u2 u3 ⇒
       u1 (fun v1 ⇒
            u0 (fun u00 ⇒ u00 s t v1 (fun v2 ⇒ v2 Bot (fun w1 w2 ⇒ u3 (w1 u2))))).
```

C.4.23 Mating Rule - 2 arguments

```
Definition matingrule_2 (sigma tau : Type) (R_sigma : sigma → sigma → o)
  (R_tau : tau → tau → o)
  : ∀ (p : (sigma ->tau → o)) (s1 t1 : sigma) (s2 t2 : tau),
    Neg(Neg(R_ar sigma (tau → o) R_sigma (R_ar tau o R_tau R_o) p p)) →
    (Neg(R_sigma s1 t1) → Bot) → (Neg(R_tau s2 t2) → Bot) → (p s1 s2) →
    Neg(p t1 t2) → Bot
  := fun p s1 t1 s2 t2 u0 u1 u2 u3 u4 ⇒
       u1 (fun v1 ⇒
            u2 (fun v2 ⇒
                 u0 (fun u00 ⇒
                      u00 s1 t1 v1 (fun w ⇒
                                      w s2 t2 v2 (fun z ⇒
                                                    z Bot (fun z1 z2 ⇒
                                                             u4 (z1 u3))))))).
```

C.4.24 Decomposition Rule - 1 argument

```
Definition decompositionrule_1 (sigma : Type) (R_sigma : sigma → sigma → o)
  : ∀ (h : (sigma → i)) (s t : sigma), Neg(Neg(R_ar sigma i R_sigma R_i h h)) →
    (Neg(R_sigma s t) → Bot) → Neg(R_i (h s) (h t)) → Bot
  := fun h s t u v w ⇒ u (fun u1 ⇒ v (fun v1 ⇒ (u1 s t v1) w)).
```

C.4.25 Decomposition Rule - 2 arguments

```
Definition decompositionrule_2 (sigma tau : Type) (R_sigma : sigma → sigma → o)
  (R_tau : tau → tau → o)
  : ∀ (h : (sigma → tau → i)) (s1 t1 : sigma) (s2 t2 : tau),
    Neg(Neg(R_ar sigma (tau → i) R_sigma (R_ar tau i R_tau R_i) h h)) →
    (Neg(R_sigma s1 t1) → Bot) → (Neg(R_tau s2 t2) → Bot) →
    Neg(R_i (h s1 s2) (h t1 t2)) → Bot
```

```
  := fun h s1 t1 s2 t2 u v1 v2 w ⇒
       u (fun u1 ⇒
            v1 (fun v11 ⇒
                  v2 (fun v21 ⇒ (u1 s1 t1 v11) (fun z1 ⇒ (z1 s2 t2 v21) w)))).
```

C.4.26 Confrontation Rule

```
Definition confrontationrule
  : ∀ (s t u v : i), (Neg(R_i s u) → Neg(R_i t u) → Bot) →
    (Neg(R_i s v) → Neg(R_i t v) → Bot) → (R_i s t) → Neg(R_i u v) → Bot
  := fun s t u v u1 u2 u3 u4 ⇒
       u1 (fun v1 ⇒
             u2 (TraNeg i R_i Tra_i u s v (Sym_i s u v1) u4)
                (TraNeg i R_i Tra_i s t v u3
                        (TraNeg i R_i Tra_i u s v (Sym_i s u v1) u4)))
          (TraNeg i R_i Tra_i s t u u3 (fun v1 ⇒
                                          u2 (TraNeg i R_i Tra_i u s v
                                                     (Sym_i s u v1) u4)
                                             (TraNeg i R_i Tra_i s t v u3
                                                     (TraNeg i R_i Tra_i u s v
                                                             (Sym_i s u v1) u4))))).
```

C.5 Other Lemmas

C.5.1 Lemma 8.3.9

```
Definition Roreft
  : ∀ (x : o), R_o x x
  := fun x p u ⇒ u (fun v ⇒ v) (fun v ⇒ v).
```

C.5.2 Lemma 8.3.8

```
Definition Rireft
  : ∀ (x : i), R_i x x
  := fun x p u ⇒ u.
```

C.5.3 Lemma 8.3.11

```
Definition negreft
  : R_ar o o R_o R_o Neg Neg
  := fun x y u v1 ⇒
       v1 (fun p v ⇒
             v (fun w1 w2 ⇒ u Bot (fun w3 w4 ⇒ w1 (w4 w2)))
               (fun w1 w2 ⇒ u Bot (fun w3 w4 ⇒ w1 (w3 w2)))).
```

C.5.4 Lemma 8.3.12

```
Definition impreft
  : R_ar o (o → o) R_o (R_ar o o R_o R_o) Imp Imp
```

```
  := fun x1 x2 u1 v1 ⇒
       v1 (fun y1 y2 u2 v2 ⇒
             v2 (fun p u3 ⇒
                   u1 p (fun w1 w2 ⇒
                           u2 p (fun w3 w4 ⇒
                                   u3 (fun w5 w6 ⇒ w3 (w5 (w2 w6)))
                                      (fun w5 w6 ⇒ w4 (w5 (w1 w6))))))).
```

C.5.5 Lemma 8.3.13

```
Definition andrefl
  : R_ar o (o → o) R_o (R_ar o o R_o R_o) And And
  := fun x1 x2 u1 v1 ⇒
       v1 (fun y1 y2 u2 v2 ⇒
             v2 (fun p u3 ⇒
                   u1 p (fun w1 w2 ⇒
                           u2 p (fun w3 w4 ⇒
                                   u3 (fun w5 q w6 ⇒
                                         w5 q (fun w7 w8 ⇒
                                                 w6 (w1 w7) (w3 w8)))
                                      (fun w5 q w6 ⇒
                                         w5 q (fun w7 w8 ⇒
                                                 w6 (w2 w7) (w4 w8))))))).
```

C.5.6 Lemma 8.3.14

```
Definition orrefl
  : R_ar o (o → o) R_o (R_ar o o R_o R_o) Or Or
  := fun x1 x2 u1 v1 ⇒
       v1 (fun y1 y2 u2 v2 ⇒
             v2 (fun p u3 ⇒
                   u1 p (fun w1 w2 ⇒
                           u2 p (fun w3 w4 ⇒
                                   u3 (fun w5 q w6 w7 ⇒
                                         w5 q (fun w8 ⇒ w6 (w1 w8))
                                              (fun w8 ⇒ w7 (w3 w8)))
                                      (fun w5 q w6 w7 ⇒
                                         w5 q (fun w8 ⇒ w6 (w2 w8))
                                              (fun w8 ⇒ w7 (w4 w8))))))).
```

C.5.7 Lemma 8.3.17

```
Definition Rrefl (sigma tau : Type) (R_sigma : sigma → sigma → o)
  (Sym_sigma : ∀ (x y : sigma),(R_sigma x y) → (R_sigma y x))
  (Tra_sigma : ∀ x y z : sigma, (R_sigma x y) → (R_sigma y z) → (R_sigma x z))
  : R_ar sigma (sigma → o) R_sigma (R_ar sigma o R_sigma R_o) R_sigma R_sigma
  := fun x1 x2 u1 v1 ⇒
       v1 (fun y1 y2 u2 v2 ⇒
             v2 (fun p u3 ⇒
```

```
                u3 (fun w1 ⇒
                      Tra_sigma x2 y1 y2
                                (Tra_sigma x2 x1 y1 (Sym_sigma x1 x2 u1) w1) u2)
                   (fun w1 ⇒
                      Tra_sigma x1 y2 y1
                                (Tra_sigma x1 x2 y2 u1 w1) (Sym_sigma y1 y2 u2))))).
```

C.5.8 Lemma 8.3.15

```
Definition existsrefl (sigma : Type) (R_sigma : sigma → sigma → o)
  : R_ar (sigma → o) o (R_ar sigma o R_sigma R_o) R_o
         (Exists sigma R_sigma) (Exists sigma R_sigma)
  := fun g1 g2 u1 u2 ⇒
       u2 (fun p u3 ⇒
             u3 (fun v1 q v2 ⇒
                   v1 q (fun x v3 v4 ⇒
                           u1 x x v3 (fun v5 ⇒
                                        v5 Bot (fun v6 v7 ⇒
                                                  u2 (fun r v8 ⇒
                                                        v8 (fun v9 r1 v10 ⇒
                                                              v10 x v3 (v6 v4))
                                                           (fun v9 ⇒ v1))))
                                     q))
                (fun v1 q v2 ⇒
                   v1 q (fun x v3 v4 ⇒
                           u1 x x v3 (fun v5 ⇒
                                        v5 Bot (fun v6 v7 ⇒
                                                  u2 (fun r v8 ⇒
                                                        v8 (fun v9 ⇒ v1)
                                                           (fun v9 r1 v10 ⇒
                                                              v10 x v3 (v7 v4)))))
                                     q))).
```

C.5.9 Lemma 8.3.16

```
Definition forallrefl (sigma : Type) (R_sigma : sigma → sigma → o)
  : R_ar (sigma → o) o (R_ar sigma o R_sigma R_o) R_o
         (Forall sigma R_sigma) (Forall sigma R_sigma)
  := fun g1 g2 u1 u2 ⇒
       u2 (fun p u3 ⇒
             u3 (fun v1 x v2 v3 ⇒
                   v1 x v2 (fun v4 ⇒
                              u1 x x v2 (fun v5 ⇒
                                           v5 Bot (fun v6 v7 ⇒ v3 (v6 v4)))))
                (fun v1 x v2 v3 ⇒
                   v1 x v2 (fun v4 ⇒
                              u1 x x v2 (fun v5 ⇒
                                           v5 Bot (fun v6 v7 ⇒ v3 (v7 v4)))))).
```

C.5.10 Lemma 8.3.19

```
Definition apptra (sigma tau : Type) (R_sigma : sigma → sigma → o)
  (R_tau : tau → tau → o)
  : ∀ (f g : (sigma → tau)) (x y : sigma),
    Neg(Neg(R_ar sigma tau R_sigma R_tau f g)) → Neg(Neg(R_sigma x y)) →
    Neg(Neg(R_tau (f x) (g y)))
  := fun f g x y u v w ⇒ v (fun v1 ⇒ u (fun u1 ⇒ u1 x y v1 w)).
```