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

1 Introduction 1

2 Simple Type Theory 3
   2.1 Syntax .............................................................................. 3
   2.2 Semantics ........................................................................ 4

3 Tableau Calculi 7
   3.1 Definitions ....................................................................... 7
   3.2 Our Tableau Rules ............................................................ 8
   3.3 Examples .......................................................................... 8

4 Natural Deduction Calculus 13
   4.1 A Natural Deduction Calculus \( \mathcal{N} \) ................................. 13
   4.2 Coq ................................................................................. 14

5 Logical Transformations 15
   5.1 Definitions ........................................................................ 15
   5.2 Translation ........................................................................ 16

6 The Girard Transformation 17
   6.1 The Girard Transformation \( \Psi_G \) ........................................ 17
   6.2 The Tableau Calculus \( \mathcal{T}_{\rightarrow, \bot} \) ................................. 18
   6.3 The Tableau Calculus \( \mathcal{T}_{\text{sec}} \) ...................................... 21
      6.3.1 \( \Psi_G \) does not Respect \( \mathcal{T}_{\text{sec}} \) .......................... 21
      6.3.2 Tableau Rules in \( \mathcal{T}_{\text{sec}} \) that are Respected by \( \Psi_G^* \) 22
      6.3.3 Tableau Rules in \( \mathcal{T}_{\text{sec}} \) that are not Respected by \( \Psi_G^* \) 22
   6.4 The Tableau Calculus \( \mathcal{T}_{\text{Full}} \) ......................................... 23
      6.4.1 \( \Psi_G \) does not Respect \( \mathcal{T}_{\text{Full}} \) .............................. 23
      6.4.2 Tableau Rules in \( \mathcal{T}_{\text{Full}} \) that are Respected by \( \Psi_G^* \) 23
      6.4.3 Tableau Rules in \( \mathcal{T}_{\text{Full}} \) that are not Respected by \( \Psi_G^* \) 24

7 The Girard-Kuroda Transformation 27
   7.1 Properties of the Girard-Kuroda Transformation ........................ 27
   7.2 The Tableau Calculus \( \mathcal{T}_{\text{elem}} \) ...................................... 28
CONTENTS

8 The Girard-Kuroda-Per Transformation  29
  8.1 The Girard-Kuroda-Per Transformation $\Psi_{GKP}$ 29
  8.2 $\Psi_{GKP}$ is a Logical Transformation 30
  8.3 Properties of $R$ 31
  8.4 The Tableau Calculus $\mathcal{T}_{Fullres}$ 36
  8.5 $\Psi_{GKP}$ respects $\mathcal{T}_{Fullres}$ 37

9 Conclusion and Future Work 43

A Girard Transformation 47
  A.1 Defining basic types $i$, $o$ 47
  A.2 Girard Transformation 47
    A.2.1 Short Hand 47
  A.3 Symmetry of equality 47
  A.4 Transitivity of equality 48
  A.5 Negative transitivity 48
  A.6 Lemmas 48
    A.6.1 Closed False Rule 48
    A.6.2 Closed Not True Rule 48
    A.6.3 Closed Rule 48
    A.6.4 Closed Neg Equal Rule 48
    A.6.5 Closed Symmetric Rule 48
    A.6.6 Double Negation Rule 48
    A.6.7 Cut Rule 49
    A.6.8 Implication Rule 49
    A.6.9 Negative Implication Rule 49
    A.6.10 And Rule 49
    A.6.11 Or Rule 49
    A.6.12 Neg And Rule 49
    A.6.13 Neg Or Rule 49
    A.6.14 Forall Rule 49
    A.6.15 Exists Rule 50
    A.6.16 DeMorgan Exists Rule 50
    A.6.17 Boolean Equality Rule 50
    A.6.18 Leibniz Rule - not used in the Thesis 50
    A.6.19 Functional Equality Rule 50
    A.6.20 Mating Rule - 2 arguments 50
    A.6.21 Decomposition Rule - 2 arguments 50
    A.6.22 Confrontation Rule 51

B Girard-Kuroda Transformation 53
  B.1 Defining basic types $i$, $o$ 53
  B.2 Girard-Kuroda Transformation 53
    B.2.1 Short Hand 53
  B.3 Symmetry of equality 53
  B.4 Transitivity of equality 54
  B.5 Negative transitivity 54
  B.6 Lemmas 54
    B.6.1 Closed False Rule 54
## C Girard-Kuroda-Per Transformation

C.1 Defining basic types $i, o$ ........................................ 59  
C.2 Girard-Kuroda-Per Transformation .................................... 59  
C.2.1 Short Hand .......................................................... 59  
C.3 Some Definitions ....................................................... 60  
C.3.1 Definition of $\text{SymNeg}$ ......................................... 60  
C.3.2 Recursive definition of $\text{Sym}$ .................................... 60  
C.3.3 Recursive definition of $\text{Tra}$ ................................... 60  
C.3.4 Definition of $\text{TraNeg}$ relative to $\text{Tra}$ ....................... 61  
C.4 Lemmas ................................................................. 61  
C.4.1 Closed False Rule ................................................... 61  
C.4.2 Closed Not True Rule ................................................. 61  
C.4.3 Closed Rule .......................................................... 61  
C.4.4 Closed Neg Equal Rule .............................................. 61  
C.4.5 Closed Symmetric Rule ............................................. 61  
C.4.6 Double Negation Rule ............................................... 61  
C.4.7 Restricted Cut Rule ................................................ 61  
C.4.8 Implication Rule .................................................... 62  
C.4.9 Negative Implication Rule ........................................ 62  
C.4.10 And Rule ............................................................ 62  
C.4.11 Or Rule ............................................................. 62  
C.4.12 Neg And Rule ........................................................ 62  
C.4.13 Neg Or Rule .......................................................... 62  
C.4.14 Restricted Forall Rule ............................................. 62  
C.4.15 DeMorgan Forall Rule .............................................. 62  
C.4.16 Exists Rule .......................................................... 63
C.4.17 DeMorgan Exists Rule ........................................... 63
C.4.18 Boolean Equality Rule ........................................ 63
C.4.19 Boolean Extensionality Rule ................................. 63
C.4.20 Restricted Functional Equality Rule ......................... 63
C.4.21 Functional Extensionality Rule .............................. 63
C.4.22 Mating Rule - 1 argument .................................. 64
C.4.23 Mating Rule - 2 arguments .................................. 64
C.4.24 Decomposition Rule - 1 argument .......................... 64
C.4.25 Decomposition Rule - 2 arguments .......................... 64
C.4.26 Confrontation Rule .......................................... 65
C.5 Other Lemmas ...................................................... 65
C.5.1 Lemma 8.3.9 .................................................... 65
C.5.2 Lemma 8.3.8 .................................................... 65
C.5.3 Lemma 8.3.11 ................................................... 65
C.5.4 Lemma 8.3.12 ................................................... 65
C.5.5 Lemma 8.3.13 ................................................... 66
C.5.6 Lemma 8.3.14 ................................................... 66
C.5.7 Lemma 8.3.17 ................................................... 66
C.5.8 Lemma 8.3.15 ................................................... 67
C.5.9 Lemma 8.3.16 ................................................... 67
C.5.10 Lemma 8.3.19 .................................................. 68
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 \( p \lor \neg p \) 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 an 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...
CHAPTER 1. INTRODUCTION

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 \( T_{\text{Full}} \). 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

\[
\mathbb{T}(\sigma, \tau, \ldots) ::= \iota \mid o \mid \sigma \rightarrow \tau
\]

where \( o \) is the type of propositions, \( \iota \) the type of individuals, and \( \rightarrow \) is a function type constructor.

We use \( \alpha \) to range over the basic types and \( \sigma, \tau, \sigma_1, \sigma_2, \ldots \) to range over elements of \( \mathbb{T} \). A type of the form \( \sigma \rightarrow \tau \) is called a function type. Parentheses in types will often be omitted by association to the right. For example, by \( \sigma \rightarrow \sigma \rightarrow \tau \) we mean \( \sigma \rightarrow (\sigma \rightarrow \tau) \). 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 \( \text{Ter} \) is defined as

\[
\text{Ter}(s, t, \ldots) ::= x \mid c \mid s t \mid \lambda x.s
\]

where \( x \in \mathcal{V} \) and \( c \in \{\top, \bot, \neg, \land, \lor, \forall, \exists, \}, =_{\sigma} \mid \sigma \in \mathbb{T} \} \).
CHAPTER 2. SIMPLE TYPE THEORY

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 \(\text{Ter}\). The \(\forall, \exists\) and \(=\) constants are indexed by a type \(\sigma\) 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 \(st\) 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:

\[
\frac{}{x : \sigma \quad \text{x has type } \sigma} \quad \frac{x : \sigma \quad s : \tau}{\lambda x.s : \sigma \tau} \quad \frac{s : \sigma \tau \quad t : \sigma}{st : \tau}
\]

The types of the logical constants are as follows:

\[\top : o, \bot : o, \neg : oo, \rightarrow : ooo, \lor : ooo, \forall \sigma : (\sigma o), \exists \sigma : (\sigma o), =_\sigma : \sigma \sigma o\]

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 \(\sigma\). We use \(\text{Ter}^\sigma\) to mean the set of terms of type \(\sigma\). 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:

\[
FV(x) := \{x\} \\
FV(st) := FV(s) \cup FV(t) \\
FV(\lambda x.s) := FV(s) - \{x\}
\]

**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 \(\text{Ter}\). \(\text{Ter}\) itself is called the full fragment.

We have the usual notion of \(\alpha-, \beta-, \eta\)-equivalence \((\sim_\alpha, \sim_\beta, \sim_\eta)\) and of \(\beta\) normal form \(([s]^{\beta})\). We consider equality of terms up to \(\alpha\)-equivalence.

We write \(s[x := t]\) for the capture avoiding substitution of term \(t\) for variable \(x\) in term \(s\). A simultaneous substitution \(\theta\) 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 \(\theta\) 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.
2.2. SEMANTICS

**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 \( f : M \rightarrow N \) to state that \( f \) is a function such that \( \text{Dom} \ f = M \) and \( \text{Ran} \ f \subseteq N \). 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 \( D \) defined on \( T \) that satisfies the following properties:

1. \( D(\circ) = \{0, 1\} \)
2. \( \forall \sigma \in T : D(\sigma) \neq \emptyset \)
3. \( \forall \sigma, \tau \in T : D(\sigma \tau) \subseteq \{f \mid f : D(\sigma) \rightarrow D(\tau)\} \)

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 \( D \) is a function \( I \) defined on \( T \cup V \) such that

1. \( D \subseteq I \)
2. \( I(x) \in I(\sigma) \) for all types \( \sigma \) and variables \( x : \sigma \)

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

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

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

We call \( \hat{I} \) the evaluation function of \( I \).

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

Note that not every evaluation function is total. This is due to the interpretation of \( \lambda \)-abstractions and logical constants. Consider the logical constant \( \neg : oo \). \( \hat{I}(\neg) \) might not be defined. This is because, even though we know that the negation function is in \( I(\circ) \rightarrow I(\circ) \), it might not be in \( I(oo) \). Recall that in the definition of frame we allow \( I(oo) \) to be a subset of \( I(\circ) \rightarrow I(\circ) \). 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 $\mathcal{I}(s) = 1$.

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 $\beta$-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 $\text{FV}^*(A) = \emptyset$.

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 $A \subseteq A_i$ for each $i \in 1, \ldots, n$. It is presented as a refutation rule of the form,

$$
\frac{A_1 \vdash \bot \quad \ldots \quad A_n \vdash \bot}{A \vdash \bot}
$$

or in a tableau view of the form,

$$
\frac{A}{A_1 \mid \ldots \mid A_n}
$$

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 $(\mathcal{F}, \mathcal{R})$ 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 $\text{Closed} \bot$, $\text{Closed} \neg \top$, $\text{Closed} \neq$, and $\text{ClosedSym}$ 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, $\text{DNeg}$ for eliminating of a double negation. For each other logical constant a positive and a negative rule is presented, for example, $\text{Imp}$ and $\text{NegImp}$ for implication, $\text{And}$ and $\text{NegAnd}$ 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 $\text{Bool}=\text{ and } \text{BoolExt}$ respectively, handle equality at base type $\circ$.

The functional equality and functional extensionality rules ($\text{Func}=\text{ and } \text{FuncExt}$) deal with equality at function types.

The Leibniz equality rule ($\text{Leibniz}$) 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 ($\text{Mat}$, $\text{Dec}$, $\text{Con}$) 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

$$((p \rightarrow q) \rightarrow p) \rightarrow p$$

where $p, q : \circ$. 

3.3. EXAMPLES

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 $\neg(((p \rightarrow q) \rightarrow p) \rightarrow p)$ since the outermost logical constants in this formula are negation and implication.

$$
\neg(((p \rightarrow q) \rightarrow p) \rightarrow p) \\
(p \rightarrow q) \rightarrow p \\
\neg p
$$

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

$$
\neg(((p \rightarrow q) \rightarrow p) \\
(p \rightarrow q) \rightarrow p \\
\neg p
$$

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 $\neg(p \rightarrow q)$ resulting in a $p$ on this branch also, so this branch is similarly closed.

$$
\neg p
$$

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

$$
\neg\exists\therefore g.\forall\therefore f.\exists\therefore u.gu = f
$$

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=.

$$
\neg\neg\exists g.\forall f.\exists u.gu = f \\
\exists g.\forall f.\exists u.gu = f \\
\forall f.\exists u.gu = f \\
\exists u.gu = \lambda x.\neg gxx \\
gd = \lambda x.\neg gxx \\
gdd = \neg gdd \\
\neg gdd
$$
<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>$A, \bot \vdash \bot$</td>
</tr>
<tr>
<td>$\neg$</td>
<td>$A \vdash \bot$</td>
</tr>
<tr>
<td>$\neg$</td>
<td>$A, s \vdash \bot$</td>
</tr>
<tr>
<td>Cut</td>
<td>$A, t \vdash \bot$</td>
</tr>
<tr>
<td>And</td>
<td>$A, t \vdash \bot$</td>
</tr>
<tr>
<td>NegOr</td>
<td>$A, \neg s \vdash \bot$</td>
</tr>
<tr>
<td>Forall</td>
<td>$A, [s t]^\beta \vdash \bot$</td>
</tr>
<tr>
<td>Exists</td>
<td>$A \vdash \bot$</td>
</tr>
<tr>
<td>Demorgan</td>
<td>$\exists s \in A, x \not\in FV(s)$</td>
</tr>
<tr>
<td>De Morgan</td>
<td>$A, \forall x, \neg s \vdash \bot$</td>
</tr>
<tr>
<td>Dneg</td>
<td>$A, s \vdash \bot$</td>
</tr>
<tr>
<td>NegImp</td>
<td>$A, s \vdash \bot$</td>
</tr>
<tr>
<td>NegAnd</td>
<td>$A, s \vdash \bot$</td>
</tr>
<tr>
<td>Or</td>
<td>$A, s \vdash \bot$</td>
</tr>
<tr>
<td>Imp</td>
<td>$A, t \vdash \bot$</td>
</tr>
<tr>
<td>NegOr</td>
<td>$A, \neg s \vdash \bot$</td>
</tr>
<tr>
<td>Forall</td>
<td>$A, [s t]^\beta \vdash \bot$</td>
</tr>
<tr>
<td>Exists</td>
<td>$A \vdash \bot$</td>
</tr>
<tr>
<td>Leibniz</td>
<td>$A, \forall p, p s = t \vdash \bot$</td>
</tr>
<tr>
<td>Func</td>
<td>$A, [s t]^{\beta} \vdash \bot$</td>
</tr>
<tr>
<td>FuncExt</td>
<td>$A, [s x \neq t x]^{\beta} \vdash \bot$</td>
</tr>
<tr>
<td>Mat</td>
<td>$A, s \neq t \vdash \bot$</td>
</tr>
<tr>
<td>Dec</td>
<td>$A, s \neq t \vdash \bot$</td>
</tr>
<tr>
<td>Con</td>
<td>$A, s \neq t \vdash \bot$</td>
</tr>
</tbody>
</table>

**Figure 3.1:** Tableau rules we will consider
### 3.3. EXAMPLES

<table>
<thead>
<tr>
<th>Rule</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closed</td>
<td>$\bot$</td>
</tr>
<tr>
<td>Closed</td>
<td>$\neg \top$</td>
</tr>
<tr>
<td>Closed</td>
<td>$s, \neg s$</td>
</tr>
<tr>
<td>Closed</td>
<td>$s \neq s$</td>
</tr>
<tr>
<td>ClosedSym</td>
<td>$(s = t), (t \neq s)$</td>
</tr>
<tr>
<td>Cut</td>
<td>$s \quad \neg s$</td>
</tr>
<tr>
<td>Dneg</td>
<td>$\neg s$</td>
</tr>
<tr>
<td>And</td>
<td>$s \quad t \quad s \land t$</td>
</tr>
<tr>
<td>Or</td>
<td>$s \quad t \quad s \lor t$</td>
</tr>
<tr>
<td>Imp</td>
<td>$s \quad t \quad \neg s \implies \neg t$</td>
</tr>
<tr>
<td>NegAnd</td>
<td>$\neg (s \land t)$</td>
</tr>
<tr>
<td>NegOr</td>
<td>$\neg (s \lor t)$</td>
</tr>
<tr>
<td>NegImp</td>
<td>$\neg (s \implies t)$</td>
</tr>
<tr>
<td>Forall</td>
<td>$\forall s \quad [s ; t]^{\beta}$</td>
</tr>
<tr>
<td>DeMorgan</td>
<td>$\neg \exists s \quad \forall s \quad \neg s \lor \neg s$</td>
</tr>
<tr>
<td>boolean</td>
<td>$\exists x \quad \neg s \quad x \not\in FV(s)$</td>
</tr>
<tr>
<td>DeMorgan</td>
<td>$\exists s \quad \forall x \quad \neg s \quad x \not\in FV(s)$</td>
</tr>
<tr>
<td>Bool</td>
<td>$s \equiv t \quad s, t \quad \neg s, \neg t$</td>
</tr>
<tr>
<td>BoolExt</td>
<td>$\neg (s \equiv t) \quad s, \neg t \quad t, \neg s$</td>
</tr>
<tr>
<td>Leibniz</td>
<td>$s =_\sigma t \quad \forall p. \quad p \quad s \rightarrow p ; t$</td>
</tr>
<tr>
<td>Func</td>
<td>$s_1 =_{\sigma \tau} s_2 \quad [s_1 ; t_1 ; s_2 ; t_2]^{\beta}$</td>
</tr>
<tr>
<td>FuncExt</td>
<td>$s \neq_{\sigma \tau} t \quad x \quad x \text{ is fresh}$</td>
</tr>
<tr>
<td>Mat</td>
<td>$x \quad s_1 \ldots s_n, \quad \neg x \quad t_1 \ldots t_n \quad s_1 \neq t_1 \ldots s_n \neq t_n$</td>
</tr>
<tr>
<td>Dec</td>
<td>$s_1 \neq t_1 \ldots s_n \neq t_n \quad x \quad s_1 \ldots s_n, \quad \neg x \quad t_1 \ldots t_n \quad s_1 \neq t_1 \ldots s_n \neq t_n$</td>
</tr>
</tbody>
</table>

**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

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

**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

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

where $x \in V$ and $\sigma \in T$.

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

**Definition 4.1.2** (Context). A *context* $\Gamma$ 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 $\langle (\Gamma, s), (\Gamma_1, s_1), \ldots, (\Gamma_n, s_n) \rangle$ where $\Gamma, \Gamma_1, \ldots, \Gamma_n$ are contexts and $s, s_1, \ldots, s_n$ are $\beta$-normal formulas that are elements of $\mathcal{N}$-terms. An ND rule is presented as follows:

$$\begin{array}{c}
\Gamma_1 \vdash_{\mathcal{N}} s_1 \\
\ldots \\
\Gamma_n \vdash_{\mathcal{N}} s_n \\
\hline
\Gamma \vdash_{\mathcal{N}} s
\end{array}$$

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

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

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

13
Note that if for some context $\Gamma$ and some formula $s$ we are given a derivation of $\Gamma \vdash_N s$ that uses the $wk$ rule, we can construct a derivation of $\Gamma \vdash_N s$ 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.

$\begin{align*}
\Gamma \vdash_N t \\
\Gamma \vdash_N \forall x.t \\
\Gamma \vdash_N s \\
\Gamma \vdash_N \exists x.t
\end{align*}$

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 correspondence of ND proofs and proof terms please refer to [29].

**Lemma 4.2.1.** $\vdash_N \forall p. p \rightarrow (p \rightarrow \bot) \rightarrow \bot$

**Proof.** By $\forall I$ and $\rightarrow I$ it is enough to prove $p, p \rightarrow \bot \vdash_N \bot$. We know this by $\rightarrow E$. \[\square\]

**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 $\Psi$ that map terms to $N$-terms. We would like the transformation to map tableau refutable formulas to ND-refutable $N$-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 $\Psi$ maps ground tableau refutable formulas to ND-refutable $N$-terms. Since a tableau calculus operates on branches instead of formulas, we will also need branch transformations $\Psi^*$ that maps branches to contexts. A branch $\Psi^*$ extends a logical transformation $\Psi$ if it agrees with $\Psi$ on ground formulas.

5.1 Definitions

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

Definition 5.1.2 (Compositional). A logical transformation $\Psi$ is compositional if it satisfies the following:

$\Psi(x) = x$ for $x \in \mathcal{V}$

$\Psi(st) = \Psi(s) \Psi(t)$

$\Psi(\lambda x.s) = \lambda x.\Psi(s)$

$\text{FV}(\Psi(c)) = \emptyset$ for all logical constants $c$

Definition 5.1.3 (Beta). A logical transformation $\Psi$ respects beta if

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

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

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

Definition 5.1.6 (Extends). Let $\Psi$ be a logical transformation. A branch transformation $\Psi^*$ extends $\Psi$ if for all ground branches $A$ we have $\Psi^*(A) = \{\Psi(s) | s \in A\}$. We say $\Psi^*$ trivially extends $\Psi$, if for all branches $A$ we have $\Psi^*(A) = \{\Psi(s) | s \in A\}$.
**Definition 5.1.7** (Branch Transformation Respects a Step). Let \( \Psi^* \) be a branch transformation and \( A, A_1, \ldots, A_n \) be branches. We say \( \Psi^* \) respects a step \( \langle A, A_1, \ldots, A_n \rangle \) if the following holds:

\[
\text{if } \Psi^*(A_1) \not\vdash_N \perp_N \ldots \Psi^*(A_n) \not\vdash_N \perp_N \text{ then } \Psi^*(A) \not\vdash_N \perp_N
\]

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 \( \Psi \) respects a tableau calculus if there exists a branch transformation \( \Psi^* \) which extends \( \Psi \) and \( \Psi^* \) respects \( \mathcal{T} \).

### 5.2 Translation

**Theorem 5.2.1** (Translation). Let \( \mathcal{T} = \langle \mathcal{F}, \mathcal{R} \rangle \) be a tableau calculus, \( \Psi^* \) be a branch transformation that respects \( \mathcal{T} \), and \( A \) be an \( \mathcal{F} \)-branch. If \( A \) is \( \mathcal{T} \)-refutable, then \( \Psi^*(A) \not\vdash_N \perp_N \).

**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, \( \Psi^*(A_1) \not\vdash_N \perp_N, \ldots, \Psi^*(A_n) \not\vdash_N \perp_N \). We want to show that \( \Psi^*(A) \not\vdash_N \perp_N \). We know that \( \Psi^* \) respects \( \mathcal{T} \). Hence, \( \Psi^*(A) \not\vdash_N \perp_N \).

**Corollary 5.2.2.** Let \( \mathcal{T} = \langle \mathcal{F}, \mathcal{R} \rangle \) be a tableau calculus, \( \Psi \) be a logical transformation that respects \( \mathcal{T} \), and \( s \) be a ground formula in \( \mathcal{F} \). If \( \{s\} \) is \( \mathcal{T} \)-refutable, then \( \{ \Psi(s) \} \not\vdash_N \perp_N \).

**Proof.** We know that \( \Psi \) respects \( \mathcal{T} \). Therefore, there exists a branch transformation \( \Psi^* \) which extends \( \Psi \) and \( \Psi^* \) respects \( \mathcal{T} \). By Theorem 5.2.1, we know \( \Psi^*(\{s\}) \not\vdash_N \perp_N \). Since \( s \) is ground therefore, \( \Psi^*(\{s\}) = \{ \Psi(s) \} \). Hence, \( \{ \Psi(s) \} \not\vdash_N \perp_N \).
Chapter 6

The Girard Transformation

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

6.1 The Girard Transformation $\Psi_G$

The Girard transformation $\Psi_G$ transforms terms as follows:

\[
\begin{align*}
\Psi_G(x) &= x \text{ for } x \in \mathcal{V} \\
\Psi_G(s \ t) &= \Psi_G(s) \ \Psi_G(t) \\
\Psi_G(\lambda x. s) &= \lambda x. \Psi_G(s) \\
\Psi_G(\rightarrow) &= \lambda x. \lambda y. x \rightarrow y \\
\Psi_G(\forall) &= \lambda f. \forall x. f x \\
\Psi_G(\forall) &= \lambda x. \lambda y. \forall p. p x \rightarrow p y \text{ (Leibniz Equality)} \\
\Psi_G(\top) &= \forall^\ast p. p \\
\Psi_G(\bot) &= \forall^\ast p. p \rightarrow p \\
\Psi_G(\wedge) &= \lambda x. \lambda y. \forall p. (x \rightarrow y \rightarrow p) \rightarrow p \\
\Psi_G(\lor) &= \lambda x. \lambda y. \forall p. (x \rightarrow p) \rightarrow (y \rightarrow p) \rightarrow p \\
\Psi_G(\exists) &= \lambda f. \forall p. (\forall x. f x \rightarrow p) \rightarrow p \\
\Psi_G(\neg) &= \lambda x. x \rightarrow \forall^\ast o. p. x \\
\Psi_G(\perp) &= \forall^\ast o. p. x \\
\Psi_G(\top) &= \forall^\ast o. p. (x \rightarrow p) \rightarrow p
\end{align*}
\]

We write $\neg_G$ and $\bot_G$ as shorthand for $\Psi_G(\neg)$ and $\Psi_G(\bot)$ respectively. We do the same for all other logical constants.

Proposition 6.1.1. The Girard transformation $\Psi_G$ is a logical transformation.

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

Proposition 6.1.2. The Girard transformation $\Psi_G$ respects beta.

Proof. Follows from Proposition 5.1.4 since $\Psi_G$ is compositional. \qed

Let $\Psi_G^*$ be the branch transformation that trivially extends $\Psi_G$.

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 $\mathcal{T}_{\text{sec}}$. In the following two sections we define these tableau calculi and state whether or not the Girard transformation $\Psi_G$ respects them.
6.2 The Tableau Calculus $\mathcal{T}_{\rightarrow, \bot}$

Let $\mathcal{F}_{\rightarrow, \bot}$ 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:

\[
\{\text{Closed}, \bot, \neg, \text{DNeg}, \text{Imp}, \text{NegImp}, \text{Cut}\}
\]

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

**Lemma 6.2.1 (NegImp).** $\vdash_{\bot}^{\neg} p. q. (p \rightarrow (\neg q) \rightarrow \bot_{\neg}) \rightarrow (\neg (p \rightarrow q)) \rightarrow \bot_{\neg}$

**Proof.** Figure 6.1 presents a derivation of $\vdash_{\bot}^{\neg} p. q. (p \rightarrow (\neg q) \rightarrow \bot_{\neg}) \rightarrow (\neg (p \rightarrow q)) \rightarrow \bot_{\neg}$. \square

**Lemma 6.2.2.** $\Psi_{G}$ 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 $\Psi_{G}^{\neg}(A), \Psi_{G}(p), \Psi_{G}(\neg q) \vdash_{\bot}^{\neg}$, then $\Psi_{G}^{\neg}(A) \vdash_{\bot}^{\neg}$. A proof is presented in Figure 6.2. Note that we implicitly make use of the fact that $\Psi_{G}(\neg(p \rightarrow q))$ is equivalent to $\neg(\Psi_{G}(p \rightarrow q))$ and to $\neg(\Psi_{G}(p) \rightarrow \Psi_{G}(q))$. In addition, this derivation depends on the transformation of the logical constants $\neg$ and $\rightarrow$ which are the ones used in the NegImp rule. \square

**Lemma 6.2.3 (Closed$\bot$).** $\vdash_{\bot}^{\neg} \Psi_{G}$ 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 $\Psi_{G}$ respects the lemma’s corresponding rule is straightforward. We presented a proof that $\Psi_{G}$ 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. \square

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

**Proof.** This follows directly from Lemma 6.2.8. \square

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

**Proof.** The proof follows directly from Corollary 5.2.2 and from Theorem 6.2.9. \square

1Note that we call this fragment $\mathcal{F}_{\rightarrow, \bot}$ because it includes the preimage of terms which only contain $\rightarrow$ and $\bot_{\neg}$ under $\Psi_{G}$. 
Figure 6.1: Derivation of proposition corresponding to $\text{NegImp}$ translated using $\Psi_{G}$.
Lemma 6.2.1

\[ \vdash \forall t_1, t_2, s_2 \rightarrow \neg \varphi(t_1 \rightarrow t_2) \rightarrow \bot.N \]

\[ \vdash \forall t_1, s_1 \rightarrow \neg \varphi(t_1) \rightarrow \varphi(q) \rightarrow \bot.N \]

\[ \Psi^G(A) \vdash \Psi^G(p), \neg \varphi \Psi^G(q) \rightarrow \bot.N \rightarrow \]

\[ \Psi^G(A) \vdash \neg \varphi(\Psi^G(q) \rightarrow \bot.N) \rightarrow \]

\[ \Psi^G(A) \vdash \neg \varphi(\Psi^G(q) \rightarrow \bot.N) \rightarrow \]

\[ \Psi^G(A) \vdash \neg \varphi(\Psi^G(q) \rightarrow \bot.N) \rightarrow \]

where \( s = (\Psi_G(p) \rightarrow \neg \varphi \Psi_G(q) \rightarrow \bot.N) \),

\[ s_1 = (t_1 \rightarrow \neg \varphi \Psi_G(q) \rightarrow \bot.N) \), and

\[ s_2 = (t_1 \rightarrow t_2 \rightarrow \bot.N) \).

Figure 6.2: \( \Psi^G \) respects NegImp
6.3  The Tableau Calculus $\mathcal{T}_{sec}$

Let $\mathcal{F}_{sec}$ be the fragment of terms containing the logical constants $\top, \bot, \neg, \land, \lor, \forall, \exists, \forall^0, \exists^0$, and $\exists^0$ and $\mathcal{R}_{sec}$ be the following set of rules:

$$\{ \text{Closed } \bot, \text{Closed } \neg \top, \text{Closed, DNeg, And, NegAnd, Or, NegOr, Imp, NegImp, Demorgan } \forall \text{, Demorgan } \exists \text{, Forall, Exists, Cut} \}$$

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

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

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

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

Lemma 6.3.2. $\forall_N (\forall x. \neg f(x)) \rightarrow \neg \forall x. f(x)$

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 intuitionistic ND system $N$. $\square$

Lemma 6.3.3. $\vdash_N (\Psi_G(\forall^0 f. \neg((\neg \forall^0 f. x) \land \neg \exists^0 f. x)) \rightarrow \perp_N) \rightarrow (\forall x. \neg \forall_0 f x) \rightarrow \neg \forall_0 \forall x. f x$

Proof. The Coq proof of this lemma is provided in Appendix A. $\square$

Lemma 6.3.4. $\{ \Psi_G(\forall^0 f. \neg((\neg \forall^0 f. x) \land \neg \exists^0 f. x)) \} \forall_N \perp_N$

Proof. Assume $\{ \Psi_G(\forall^0 f. \neg((\neg \forall^0 f. x) \land \neg \exists^0 f. x)) \} \vdash_N \perp_N$. By the $\rightarrow I$ rule we know $\vdash_N \Psi_G(\forall^0 f. \neg((\neg \forall^0 f. x) \land \neg \exists^0 f. x)) \rightarrow \perp_N$. Using Lemma 6.3.3 and the $\rightarrow E$ rule, we know $\vdash_N (\forall x. \neg_0 \neg_0 f x) \rightarrow \neg_0 \neg_0 \forall x. f x$. This leads to a contradiction with Lemma 6.3.2. Hence, $\{ \Psi_G(\forall^0 f. \neg((\neg \forall^0 f. x) \land \neg \exists^0 f. x)) \} \forall_N \perp_N$. $\square$

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

Proof. Consider the following formula $s$:

$$\forall^0 f. \neg((\neg \forall^0 f. x) \land \neg \exists^0 f. x)$$

The $\mathcal{F}_{sec}$-branch $\{ \neg \forall^0 f. \neg((\neg \forall^0 f. x) \land \neg \exists^0 f. x) \}$ 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 $\{ \Psi_G(\forall^0 f. \neg((\neg \forall^0 f. x) \land \neg \exists^0 f. x)) \} \forall_N \perp_N$. Since $s \in \mathcal{F}_{sec}$ and $\text{FV}(s) = \emptyset$, therefore using Corollary 5.2.2 it follows that the logical transformation $\Psi_G$ does not respect $\mathcal{T}_{sec}$. $\square$

If $\Psi_G$ 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 $\Psi_G$ respects and which rules it does not respect.
Theorem 6.3.14. The branch transformation 6.3.13, 6.3.11, 6.3.12, and 6.2.7 respectively.

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, (Lemma 6.3.13 (Lemma 6.3.12 (Lemma 6.3.11 (Lemma 6.3.10 (Lemma 6.3.9 (Lemma 6.3.8 (Lemma 6.3.7 (Lemma 6.3.6 (Lemma 6.3.5 (Lemma 6.3.4 (Lemma 6.3.3 (Lemma 6.3.2 (Lemma 6.3.1.

Figure 6.3: Tableau Proof of \( \forall^{\sigma}f.\neg((\forall^{\sigma}x.p x) \land \neg\exists^{\sigma}x.\neg p x) \)

6.3.2 Tableau Rules in \( T_{sec} \) that are Respected by \( \Psi^{*}_G \)

By Lemma 6.2.8 we know that \( \Psi^{*}_G \) respects the following rules:

Closed \( \bot \), Closed, DNeg, Imp, NegImp, Cut

Lemma 6.3.6 (Closed-\( \neg \)). \( \vdash_{\forall} (\neg^{\sigma} \top^{\sigma}) \rightarrow \bot_{\forall} \)

Lemma 6.3.7 (And). \( \vdash_{\forall} \forall p.q.(p \rightarrow q \rightarrow \bot_{\forall}) \rightarrow (p \land^{\sigma} q) \rightarrow \bot_{\forall} \)

Lemma 6.3.8 (NegAnd). \( \vdash_{\forall} \forall p.q.((\neg^{\sigma} p) \rightarrow \bot_{\forall}) \rightarrow ((\neg^{\sigma} q) \rightarrow \bot_{\forall}) \rightarrow \neg^{\sigma} (p \land^{\sigma} q) \rightarrow \bot_{\forall} \)

Lemma 6.3.9 (Or). \( \vdash_{\forall} \forall p.q.(p \rightarrow \bot_{\forall}) \rightarrow (q \rightarrow \bot_{\forall}) \rightarrow (p \lor^{\sigma} q) \rightarrow \bot_{\forall} \)

Lemma 6.3.10 (NegOr). \( \vdash_{\forall} \forall p.q.((\neg^{\sigma} p) \rightarrow (\neg^{\sigma} q) \rightarrow \bot_{\forall}) \rightarrow \neg^{\sigma} (p \lor^{\sigma} q) \rightarrow \bot_{\forall} \)

Lemma 6.3.11 (Forall). \( \vdash_{\forall} \forall^{\sigma} f.\forall^{\sigma} t.(f \rightarrow f \rightarrow \bot_{\forall}) \rightarrow ((\forall^{\sigma} f) \rightarrow \bot_{\forall}) \)

Lemma 6.3.12 (Exists). \( \vdash_{\forall} \forall^{\sigma} f. (\forall^{\sigma} x. f x \rightarrow \bot_{\forall}) \rightarrow (\exists^{\sigma} f) \rightarrow \bot_{\forall} \)

Lemma 6.3.13 (DeMorgan\( \exists \)). \( \vdash_{\forall} \forall^{\sigma} \neg^{\sigma} f.((\forall^{\sigma} f (\lambda x. \neg^{\sigma} f x)) \rightarrow \bot_{\forall}) \rightarrow (\neg^{\sigma} (\exists^{\sigma} f)) \rightarrow \bot_{\forall} \)

Theorem 6.3.14. The branch transformation \( \Psi^{*}_G \) respects the following rules:

Closed \( \bot \), Closed-\( \neg \), 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 \( T_{sec} \) that are not Respected by \( \Psi^{*}_G \)

Theorem 6.3.15. The branch transformation \( \Psi^{*}_G \) does not respect the DeMorgan\( \forall \) rule.

Proof. Let \( T \) be a tableau calculus containing exactly the following rules:

DeMorgan\( \forall \), Exists, DNeg, And, Closed

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

□
6.4 The Tableau Calculus $\mathcal{T}_{F\text{ull}}$

$\mathcal{T}_{F\text{ull}}$ 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 $\mathcal{T}_{F\text{ull}}$ 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 DeMorgan rules, but not respect the Leibniz equality does not follow from Boolean extensionality. Therefore we argue that $\Psi_{G\kappa\mathcal{F}}$ does not respect the Leibniz rule at type $o$. We already know from Lemma 6.3.15 that $\Psi^*_G$ does not respect the DeMorgan $\forall$ rule. Thus, $\Psi_G$ does not respect the full tableau calculus. However, we still check which rules in this full calculus are respected by $\Psi_G$. 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 $\mathcal{T}_{F\text{ull}}$ is complete.

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

6.4.1 $\Psi_G$ does not Respect $\mathcal{T}_{F\text{ull}}$

**Theorem 6.4.2.** The Girard transformation $\Psi_G$ does not respect $\mathcal{T}_{F\text{ull}}$.

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

6.4.2 Tableau Rules in $\mathcal{T}_{F\text{ull}}$ that are Respected by $\Psi^*_G$

Using Theorem 6.3.14 we know that the branch transformation $\Psi^*_G$ respects the following rules:

- Closed $\bot$, Closed $\neg T$, Closed DNeg, And, NegAnd, Or,
- NegOr, Imp, NegImp, DeMorgan $\exists$, Forall, Exists, Cut

**Lemma 6.4.3 (Closed$\neq$).** $\vdash_{\mathcal{N}} \forall^o s. \neg \psi(s \equiv \psi) s \rightarrow \bot_{\mathcal{N}}$

**Lemma 6.4.4 (ClosedSym).** $\vdash_{\mathcal{N}} \forall^o s t.(s \equiv \psi t) \rightarrow \psi(t \equiv s) \rightarrow \bot_{\mathcal{N}}$

**Lemma 6.4.5 (Bool=).** $\vdash_{\mathcal{N}} \forall^o p q.(p \rightarrow q \rightarrow \bot_{\mathcal{N}}) \rightarrow ((\neg \psi p) \rightarrow (\neg \psi q) \rightarrow \bot_{\mathcal{N}}) \rightarrow (p \equiv q) \rightarrow \bot_{\mathcal{N}}$

**Lemma 6.4.6 (Func=).** $\vdash_{\mathcal{N}} \forall^o r k h.\forall^o t.(k t = \psi h t) \rightarrow \bot_{\mathcal{N}}) \rightarrow (k = \psi h) \rightarrow \bot_{\mathcal{N}}$

**Lemma 6.4.7 (Mat).** $\vdash_{\mathcal{N}} \forall^o \sigma_1 \cdots \sigma_o p.\forall^o s_1 t_1.\forall^o s_2 t_2 \cdots \forall^o s_n t_n.(\neg \psi(s_1 = \psi t_1) \rightarrow \bot_{\mathcal{N}}) \rightarrow (\neg \psi(s_2 = \psi t_2) \rightarrow \bot_{\mathcal{N}}) \rightarrow \cdots \rightarrow (\neg \psi(s_n = \psi t_n) \rightarrow \bot_{\mathcal{N}}) \rightarrow (p s_1 s_2 \cdots s_n \rightarrow \neg \psi(p t_1 t_2 \cdots t_n) \rightarrow \bot_{\mathcal{N}}$

**Lemma 6.4.8 (Dec).** $\vdash_{\mathcal{N}} \forall^o \sigma_1 \cdots \sigma_o h.\forall^o s_1 t_1.\forall^o s_2 t_2 \cdots \forall^o s_n t_n.(\neg \psi(s_1 = \psi t_1) \rightarrow \bot_{\mathcal{N}}) \rightarrow (\neg \psi(s_2 = \psi t_2) \rightarrow \bot_{\mathcal{N}}) \rightarrow \cdots \rightarrow (\neg \psi(s_n = \psi t_n) \rightarrow \bot_{\mathcal{N}}) \rightarrow (h s_1 s_2 \cdots s_n = \psi h t_1 t_2 \cdots t_n) \rightarrow \bot_{\mathcal{N}}$

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 s_1 t_1 s_2 t_2. (\neg \gamma(s_1 = \varnothing s_2) \rightarrow \neg \gamma(t_1 = \varnothing s_2) \rightarrow \bot, \gamma(s_1 = \varnothing t_2) \rightarrow \neg \gamma(t_1 = \varnothing t_2) \rightarrow \bot, \gamma(s_1 = \varnothing t_2) \rightarrow \neg \gamma(t_1 = \varnothing t_2) \rightarrow \bot) \rightarrow (\neg \gamma(s_1 = \varnothing s_2) \rightarrow \bot, \gamma(s_1 = \varnothing t_2) \rightarrow \neg \gamma(t_1 = \varnothing t_2) \rightarrow \bot).

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. \( \square \)

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 \( \mathcal{N}_{\mathcal{R}_3} \) 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. \( \exists M \in \mathcal{M}_\beta : M \vdash \Psi_{\mathcal{G}}(\exists^o x y. \exists^o f. \neg(x \rightarrow y \rightarrow f x \rightarrow f y)) \)

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 \( \Psi_{\mathcal{G}}(\exists^o x y. \exists^o f. \neg(x \rightarrow y \rightarrow f x \rightarrow f y)) \).

Lemma 6.4.12. \( \{ \Psi_{\mathcal{G}}(\exists^o x y. \exists^o f. \neg(x \rightarrow y \rightarrow f x \rightarrow f y)) \} \vdash_{\mathcal{N}} \bot \)

Proof. Assume \( \{ \Psi_{\mathcal{G}}(\exists^o x y. \exists^o f. \neg(x \rightarrow y \rightarrow f x \rightarrow f y)) \} \vdash_{\mathcal{N}} \bot \).

Our ND calculus \( \mathcal{N} \) is sound with respect to \( \mathcal{M}_\beta \).

Therefore, \( \forall M \in \mathcal{M}_\beta : M \nvdash \Psi_{\mathcal{G}}(\exists^o x y. \exists^o f. \neg(x \rightarrow y \rightarrow f x \rightarrow f y)) \).

This contradicts Lemma 6.4.11. \( \square \)

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 \( \{ \exists^o x y. \exists^o f. \neg(x \rightarrow y \rightarrow f x \rightarrow f y) \} \) 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 \( \{ \Psi_{\mathcal{G}}(\exists^o x y. \exists^o f. \neg(x \rightarrow y \rightarrow f x \rightarrow f y)) \} \vdash_{\mathcal{N}} \bot \). Since \( \exists x y. \exists^o f. \neg(x \rightarrow y \rightarrow f x \rightarrow f y) \) 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. \( \square \)

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

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

Lemma 6.4.15. \( \{ \Psi_{\mathcal{G}}(\exists^o f g. \neg((\neg \exists^o x. f x \neq g x) \rightarrow (f = g))) \} \vdash_{\mathcal{N}} \bot \)

Proof. Assume \( \{ \Psi_{\mathcal{G}}(\exists^o f g. \neg((\neg \exists^o x. f x \neq g x) \rightarrow (f = g))) \} \vdash_{\mathcal{N}} \bot \).

Our ND calculus \( \mathcal{N} \) is sound with respect to \( \mathcal{M}_\beta \).

Thus, \( \forall M \in \mathcal{M}_\beta : M \nvdash \Psi_{\mathcal{G}}(\exists^o f g. \neg((\neg \exists^o x. f x \neq g x) \rightarrow (f = g))) \).

This contradicts Lemma 6.4.14. \( \square \)
Theorem 6.4.16. The branch transformation $\Psi^*_G$ does not respect the FuncExt rule.

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

- $\text{Exists}$, $\text{NegImp}$, $\text{DeMorgan}$, $\text{FuncExt}$, $\text{Forall}$, $\text{Closed}$

A tableau refutation of the branch $\{\exists^o f.g.\neg((\neg\exists^o x.f x \neq g x) \rightarrow (f = g))\}$ that uses only the rules in $\mathcal{T}$ is given in Figure 6.5. Therefore, this formula is $\mathcal{T}$-refutable. We know by Lemma 6.4.15 that $\{\Psi_G(\exists^o f.g.\neg((\neg\exists^o x.f x \neq g x) \rightarrow (f = g)))\} \not\models \bot_N$. Since $\exists^o f.g.\neg((\neg\exists^o x.f x \neq g x) \rightarrow (f = g))$ is a ground formula, therefore using Corollary 5.2.2 it follows that the logical transformation $\Psi_G$ does not respect $\mathcal{T}$. We know that $\Psi^*_G$ respects the rules $\text{Exists}$, $\text{NegImp}$, $\text{DeMorgan}$, $\text{FuncExt}$, and $\text{Closed}$ using Theorem 6.4.10. Therefore, $\Psi^*_G$ does not respect the FuncExt rule. $\Box$

Figure 6.5: Tableau Refutation of $\exists^o f.g.\neg((\neg\exists^o x.f x \neq g x) \rightarrow (f = g))$

Theorem 6.4.17. The branch transformation $\Psi^*_G$ does not respect the following rules:

- $\text{DeMorgan}_\forall$, $\text{BoolExt}$, $\text{FuncExt}$

Proof. Follows directly from Theorems 6.3.15, 6.4.13, and 6.4.16. $\Box$

It may be a bit surprising that the only tableau rule which makes use of classical logic is the $\text{DeMorgan}_\forall$ rule. Thus, this rule is not respected by the straightforward transformation $\Psi_G^*$. 
On the other hand, it is not at all surprising that $\Psi^G$ does not respect the $\text{BoolExt}$ rule and the $\text{FuncExt}$ rule. This is because they explicitly make use of the concepts of Boolean and functional extensionality respectively. Note that $\Psi^G$ 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 $\Psi_{GK}$ is a classical non-extensional logical transformation which slightly modifies the Girard transformation $\Psi_G$. 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 $\text{BoolExt}$ and $\text{FuncExt}$ rules which introduce the principles of Boolean extensionality and functional extensionality respectively, the only rule that is not respected by the intuitionistic transformation $\Psi_G$ is the $\text{DeMorgan}^\forall$ rule. This is because the $\text{DeMorgan}^\forall$ 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 $\text{DeMorgan}^\forall$ 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 $\Psi_{GK}$ modifies the Girard transformation of the logical constant $\forall$ such that the $\text{DeMorgan}^\forall$ rule is respected.

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

$$\Psi_{GK}(\forall) = \lambda f. \forall x. \neg_G\neg_G f x$$

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

**Proposition 7.1.1.** The Girard-Kuroda transformation $\Psi_{GK}$ 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. \(\Box\)
Proposition 7.1.2. The Girard-Kuroda transformation $\Psi_{GK}$ respects beta.

Proof. Follows from Proposition 5.1.4 since $\Psi_{GK}$ is compositional. \hfill \Box

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

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

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

\begin{align*}
\{ & \text{Closed} \bot, \text{Closed} \neg \top, \text{Closed} \neg, \text{And}, \text{NegAnd}, \text{Or}, \text{NegOr}, \\
& \text{Imp}, \text{NegImp}, \text{DeMorgan} \exists, \text{DeMorgan} \forall, \text{Forall}, \text{Exists}, \text{Cut}, \\
& \text{Closed} \neq, \text{ClosedSym}, \text{Bool} =, \text{Leibniz}, \text{Func} =, \text{Mat}, \text{Dec}, \text{Con} \}\end{align*}

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

Note that the $\Psi_{GK}$ 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).** $\vdash^{\Psi \sigma_0}_{\forall} f. \forall \sigma. (f t \to \bot_{\forall}) \to (\forall \sigma_{\forall} f) \to \bot_{\forall}$

**Lemma 7.2.2 (DeMorgan $\exists$).** $\vdash^{\Psi \sigma_0}_{\forall} f. ((\forall \sigma_{\forall} (\lambda \sigma f. x)) \to \bot_{\forall}) \to (\exists \sigma_{\forall} f) \to \bot_{\forall}$

**Lemma 7.2.3 (DeMorgan $\forall$).** $\vdash^{\Psi \sigma_0}_{\forall} f. ((\exists \sigma_{\forall} (\lambda \sigma f. x)) \to \bot_{\forall}) \to (\forall \sigma_{\forall} f) \to \bot_{\forall}$

**Lemma 7.2.4 (Leibniz).** $\vdash^{\Psi \sigma_0}_{\forall} x f. ((\forall \sigma_{\forall} (\lambda \sigma f. x \to y)) \to \bot_{\forall}) \to (x = \sigma y) \to \bot_{\forall}$

**Lemma 7.2.5.** The branch transformation $\Psi^*_GK$ 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. \hfill \Box

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

**Proof.** This follows directly from Lemma 7.2.5. \hfill \Box

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

**Proof.** The proof follows directly from Corollary 5.2.2 and from Theorem 7.2.6. \hfill \Box

Using Lemma 7.2.5 we know that $\Psi_{GK}$ respects all the rules in $\mathcal{T}_{Full}$ except for the $\text{Bool} \neq$ and $\text{Func} \neq$ rules. Moreover, using similar proofs to those of Theorems 6.4.13 and 6.4.16 provided in Chapter 6 we know that $\Psi_{GK}$ does not respect the $\text{Bool} \neq$ and $\text{Func} \neq$ rules.
Chapter 8

The Girard-Kuroda-Per Transformation

The Girard-Kuroda-Per transformation $\Psi_{GKP}$ is a classical extensional logical transformation which modifies the Girard-Kuroda transformation $\Psi_{GK}$. It introduces the principles of Boolean and functional extensionality and therefore makes it possible to respect the rules $\text{BoolExt}$ and $\text{FuncExt}$. 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 $\Psi_{GKP}$ 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, $\Psi_{GKP}$ 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 $\Psi_{GKP}$ are the same up to double negations is still open for future work.

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

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

- $R^\sigma = \lambda x y. (x \to y) \land (y \to x)$
- $R^\iota = \lambda x y. \forall q. q x \to q y$
- $R^{\sigma\to\tau} = \lambda f g. \forall x y. R^\sigma x y \to \neg (R^\tau(f x)(g y))$

This term $R^\sigma$ corresponds to a binary relation on type $\sigma$. To give the reader a good intuition we sometimes speak $R^\sigma$ 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 $\Psi_{GKP}$ agrees with the Girard transformation $\Psi_G$ on all logical constants except for $\forall$, $\exists$ and $\equiv$, which it transforms as follows:

- $\Psi_{GKP}(\equiv^\sigma) = R^\sigma$
- $\Psi_{GKP}(\forall^\sigma) = \lambda f. \forall x. (R^\sigma x x) \to \neg \neg f x$
- $\Psi_{GKP}(\exists^\sigma) = \lambda f. \forall p. (\forall x. (R^\sigma x x) \to f x \to p) \to p$

29
We write $\forall_{GKP}$ and $\exists_{GKP}$ as shorthand for $\Psi_{GKP}(\forall)$ and $\Psi_{GKP}(\exists)$, respectively.

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

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

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 $(\hat{I}(R^\sigma) a b = 1) \iff a = b$.

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

- Case $\sigma = i$:
  We want to show $\hat{I}(R^i) a b = 1 \iff a = b$. We know that $R^i$ 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 $\hat{I}(R^{\sigma_1 \sigma_2}) a b = 1 \iff a = b$.
  - We show $\hat{I}(R^{\sigma_1 \sigma_2}) a b = 1 \implies a = b$.
    Assume $\hat{I}(R^{\sigma_1 \sigma_2}) a b = 1$. We need to show $a = b$. Let $c \in \mathcal{I}(\sigma_1)$ be given. Now we need to show $a(c) = b(c)$.
    
    $\begin{align*}
    \hat{I}(R^{\sigma_1 \sigma_2}) a b &= 1 \\
    \iff \hat{I}(\lambda f g. \forall x y. R^{\sigma_1} x y \to \neg \neg (R^{\sigma_2}(f x)(g y))) a b &= 1 \\
    \iff \hat{I}_{ab}^{\psi \psi}(\forall x y. R^{\sigma_1} x y \to \neg \neg (R^{\sigma_2}(f x)(g y))) &= 1 \\
    \iff \hat{I}_{ab}^{\psi \psi}(R^{\sigma_1} x y \to \neg \neg (R^{\sigma_2}(f x)(g y))) &= 1 \\
    \iff \hat{I}_{ab}^{\psi \psi}(R^{\sigma_1} x y \to (R^{\sigma_2}(f x)(g y))) &= 1 \\
    \iff \hat{I}_{ab}^{\psi \psi}(R^{\sigma_1} x y) &= 1 \implies \hat{I}_{ab}^{\psi \psi}(R^{\sigma_2}(f x)(g y))) &= 1 \\
    \iff (\hat{I}(R^{\sigma_1}) c c = 1 \implies \hat{I}(R^{\sigma_2})(a(c))(b(c)) &= 1) \\
    \iff (c = c \implies a(c) = b(c)) \\
    \iff a(c) = b(c) \\end{align*}$

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

\[
\hat{I}(R^{a\sigma_1}) \ a \ b = 1 \\
\iff \hat{I}(\lambda f. g. \forall x. y. R^{a\sigma_1}(x \ y \rightarrow \neg \phi \neg \psi (R^{a\sigma_2}(f \ x)(g \ y)))) \ a \ b = 1 \\
\iff \overline{\hat{I}_{ab}} (\forall x. y. R^{a\sigma_1}(x \ y \rightarrow \neg \phi \neg \psi (R^{a\sigma_2}(f \ x)(g \ y)))) = 1 \\
\iff \forall c, d \in \mathcal{I}(\sigma_1): \overline{\hat{I}_{abcd}} (R^{a\sigma_1}(x \ y \rightarrow \neg \phi \neg \psi (R^{a\sigma_2}(f \ x)(g \ y)))) = 1 \\
\iff \forall c, d \in \mathcal{I}(\sigma_1): \overline{\hat{I}_{abcd}} (R^{a\sigma_1}(x \ y \rightarrow (R^{a\sigma_2}(f \ x)(g \ y)))) = 1 \\
\iff \forall c, d \in \mathcal{I}(\sigma_1): \overline{\hat{I}_{abcd}} (R^{a\sigma_1}(x \ y \rightarrow (R^{a\sigma_2}(f \ x)(g \ y)))) = 1 \\
\iff \forall c, d \in \mathcal{I}(\sigma_1): \overline{\hat{I}_{abcd}} (R^{a\sigma_1}(c \ d = 1 \Rightarrow \hat{I}(R^{a\sigma_2})(a(c))\ (b(d)))) = 1 \\
\iff \forall c, d \in \mathcal{I}(\sigma_1): c = d \Rightarrow a(c) = b(d) \\
\iff a = b
\]

\[\square\]

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

**Proof.** Follows directly from Lemma 8.2.1. \[\square\]

**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. \[\square\]

### 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 $\sigma$ is **reflexive** if $\vdash_{\mathcal{N}} \forall \sigma. x. R^{\sigma \sigma} x x$.

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

$$\vdash_{\mathcal{N}} \forall \sigma \ x \ y. (G \ x \ y) \rightarrow (G \ y \ x)$$

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

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

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

$$\vdash_{\mathcal{N}} \forall \sigma \ x \ y \ z. (G \ x \ y) \rightarrow \neg \psi (G \ x \ z) \rightarrow \neg \psi (G \ y \ z)$$

We show that for all types $\sigma$ the relation $R^{\sigma}$ that our transformation transforms equality into is both symmetric and transitive. Moreover, we show that not all types are reflexive. Hence, we can conclude that $R^{\sigma}$ is a partial equivalence relation. This is the reason why we call this transformation Girard-Kuroda-Per.
Lemma 8.3.5. For each type $\sigma$ the relation $R^\sigma$ is symmetric.

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

Lemma 8.3.6. For each type $\sigma$ the relation $R^\sigma$ is transitive.

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

Lemma 8.3.7. For each type $\sigma$ the relation $R^\sigma$ 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 $i$ are reflexive. Then we show some non-reflexive types, namely the types $oo$ and $oi$. This shows that not all types are reflexive.

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

Lemma 8.3.8. The basic type $i$ 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 $oi$ are not reflexive.

Proof. The model $M^f$ constructed in Example 5.4 of [4] is in $M_\beta$ and it is not difficult to verify that it satisfies the negation of both $\forall oo. f. R^{oo} ff$ and $\forall oo. f. R^{oo} ff$. Since $N$ is sound with respect to $M_\beta$, therefore the types $oo$ and $oi$ 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 $\Gamma \vdash N \neg R x x$ then we know $\Gamma \vdash \neg G \neg G (R (\Psi_{GKP}(t)) (\Psi_{GKP}(t)))$. 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 $N$-terms does not hold. This theorem is necessary to prove that Girard-Kuroda-Per respects the tableau calculus $TFull_{\neg \Pi}$, 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 $\vdash R c' c'$. The most interesting case is the $\exists_{GKP}$ 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. $\vdash R \neg R \neg R$

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

Lemma 8.3.12. $\vdash R \rightarrow \rightarrow$

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

Lemma 8.3.13. $\vdash R \wedge \wedge$

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

Lemma 8.3.14. $\vdash R \bigvee \bigvee$

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

Lemma 8.3.15. \( \vdash_R \exists_{\sigma : \text{gp}} \exists_{\sigma : \text{gp}} \) for any type \( \sigma \)

Proof. A Coq proof of this lemma is included in Appendix C. We also provide an explanation of the proof here.

We show that \( \vdash_{\text{gp}} R^{\sigma} (\exists_{\sigma : \text{gp}} g_1) (\exists_{\sigma : \text{gp}} g_2) \) for any \( \sigma \).

1. After unfolding the definition of \( R^{\sigma} \) we need to show \( \neg \neg \sigma \neg \neg \sigma \) under the assumption \( \forall \sigma \exists_{\sigma : \text{gp}} R^{\sigma} (\exists_{\sigma : \text{gp}} g_1) (\exists_{\sigma : \text{gp}} g_2) \).

2. So suppose \( \neg \neg \sigma \neg \neg \sigma \).

3. We need to show \( \exists_{\sigma : \text{gp}} g_2 \), of which we present only the implication from left to right here. (The other part is symmetric.)

4. So suppose \( \exists_{\sigma : \text{gp}} g_1 \), i.e., \( \forall \sigma p. (\forall \sigma x. R^{\sigma} x x \to \sigma g_1 x \to \sigma p) \to \sigma p \).

5. We need to show \( \exists_{\sigma : \text{gp}} g_2 \), i.e., \( \forall \sigma p. (\forall \sigma x. R^{\sigma} x x \to \sigma g_2 x \to \sigma p) \to \sigma p \).

6. It suffices to show \( p \). Note that we will not need to use the assumption \( \forall \sigma x. R^{\sigma} x x \to \sigma g_2 x \to \sigma p \).

7. Using (4) this reduces to showing \( \forall \sigma x. R^{\sigma} x x \to \sigma g_1 x \to \sigma p \).

8. So suppose \( R^{\sigma} x x \) and \( g_1 x \).

9. We need to show \( p \).

10. Instantiating the assumption in (1) yields \( \neg \sigma \neg \sigma (R^{\sigma} (g_1 x) (g_2 x)) \).

11. It thus suffices to show \( \neg \sigma (R^{\sigma} (g_1 x) (g_2 x)) \) (since from \( \bot \) we can conclude \( p \)).

12. So suppose \( R^{\sigma} (g_1 x) (g_2 x) \).

13. We need to show \( \bot \).

14. From \( g_1 x \) and \( g_1 x \to \sigma g_2 x \) we get \( g_2 x \).

15. By (2) it suffices to show \( R^{\sigma} (\exists_{\sigma : \text{gp}} g_1) (\exists_{\sigma : \text{gp}} g_2) \) (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 \( \exists_{\sigma : \text{gp}} g_2 \), i.e., \( \forall \sigma p. (\forall \sigma x. R^{\sigma} x x \to \sigma g_2 x \to \sigma p) \to \sigma p \).

18. So suppose \( \forall \sigma x. R^{\sigma} x x \to \sigma g_2 x \to \sigma q \).

19. We need to show \( q \), which follows from (18), (8), and (14).

\( \square \)

Lemma 8.3.16. \( \vdash_{\text{gp}} R \forall_{\sigma : \text{gp}} \forall_{\sigma : \text{gp}} \) for any type \( \sigma \)

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

Lemma 8.3.17. \( \vdash_{\text{gp}} R R^{\sigma} R^{\sigma} \) for any type \( \sigma \)
CHAPTER 8. THE GIRARD-KURODA-PER TRANSFORMATION

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 $R^\sigma$ 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 $R^\sigma$, we provide Coq proofs for the lemma generalized to any relation which is symmetric and transitive. Since we know $R^\sigma$ is symmetric and transitive on all types $\sigma$ thus using the Coq proofs we can obtain proofs for the corresponding lemma.

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

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

- Case $t = \top_j$ or $t = \bot_j$
  
  We need to show that $\vdash_N R^\sigma \top \top$ and $\vdash_N R^\sigma \bot \bot$. They both follow directly using Lemma 8.3.9.

- Case $t = \neg_j$, $t = \neg_j$, $t = \wedge_j$, $t = \vee_j$, $t = \exists_{\sigma_{GKP}}$, $t = \forall_{\sigma_{GKP}}$, 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. $\vdash_N \forall t_1 t_2 (\neg_j \neg_j R^{\sigma \tau}(t_1 t_1) \rightarrow (\neg_j \neg_j R^\sigma t_2 t_2) \rightarrow (\neg_j \neg_j R^\tau(t_1 t_2)(t_1 t_2)))$

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

Lemma 8.3.20. For all terms $t$, for all substitutions $\theta_1, \theta_2$ and, for all contexts $\Gamma$:

  if $\forall x \in FV(t) : \Gamma \vdash_N R(\theta_1(x))(\theta_2(x))$ then $\Gamma \vdash_N \neg_j \neg_j (R(\theta_1(\Psi_{GKP}(t)))(\theta_2(\Psi_{GKP}(t))))$

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

- Case $t$ is a variable $x$
  
  We know $\Psi_{GKP}(t) = x$ 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 $\Psi_{GKP}(t) = (\Psi_{GKP}(t_1)) (\Psi_{GKP}(t_2))$.
  
  Assume that the following holds:

  $\forall x \in FV(t) : \Gamma \vdash_N R(\theta_1(x))(\theta_2(x))$

  We want to show that

  $\Gamma \vdash_N \neg_j \neg_j (R(\theta_1(\Psi_{GKP}(t_1 t_2)))(\theta_2(\Psi_{GKP}(t_1 t_2))))$.

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

  $\forall x \in FV(t_1) : \Gamma \vdash_N R(\theta_1(x))(\theta_2(x))$
Similarly since $FV(t_2) \subseteq FV(t)$, therefore we also know:

$$\forall x \in FV(t_2) : \Gamma \vdash_R (\theta_1(x)) (\theta_2(x))$$

Thus, by applying the induction hypothesis with $\theta_1$, $\theta_2$, $\Gamma$ we know the following two facts:

$$\Gamma \vdash_R \neg \neg \phi (R (\theta_1(\Psi_{GKP}(t_1)))) (\theta_2(\Psi_{GKP}(t_2))))$$

and

$$\Gamma \vdash_R \neg \neg \phi (R (\theta_1(\Psi_{GKP}(t_2)))) (\theta_2(\Psi_{GKP}(t_2))))$$

By Lemma using 8.3.19 we directly get $\Gamma \vdash_R \neg \neg \phi (R (\theta_1(\Psi_{GKP}(t_1 t_2)))) (\theta_2(\Psi_{GKP}(t_1 t_2))))$.

- Case $t = \lambda x . t'$

We know $\Psi_{GKP}(t) = \lambda x . (\Psi_{GKP}(t'))$.

Assume that the following holds:

$$\forall y \in FV(t) : \Gamma \vdash_R (\theta_1(y)) (\theta_2(y))$$

We want to show

$$\Gamma \vdash_R \neg \neg \phi (R (\theta_1(\Psi_{GKP}(\lambda x . t')))) (\theta_2(\Psi_{GKP}(\lambda x . t'))))$$

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

$$\Gamma \vdash_R \neg \neg \phi (R (\lambda x . \theta_1(\Psi_{GKP}(t')))) (\lambda x . \theta_2(\Psi_{GKP}(t'))))$$

for which it suffices to show

$$\Gamma \vdash_R (\lambda x . \theta_1(\Psi_{GKP}(t'))) (\lambda x . \theta_2(\Psi_{GKP}(t')))$$

After unfolding the definition of $R$, we need to show

$$\Gamma,(R \ x_1 \ x_2) \vdash_R \neg \neg \phi (R ((\lambda x . \theta_1(\Psi_{GKP}(t')) \ x_1)) ((\lambda x . \theta_2(\Psi_{GKP}(t'))) \ x_2))$$

where $x_1$ and $x_2$ are two distinct fresh variables.

By $\beta$-reduction it remains to show

$$\Gamma,(R \ x_1 \ x_2) \vdash_R \neg \neg \phi (R (\theta_1(\Psi_{GKP}(t')))[x := x_1] (\theta_2(\Psi_{GKP}(t')))[x := x_2])$$

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

$$\forall y \in FV(t') : \Gamma' \vdash_R (\theta_1'(y)) (\theta_2'(y))$$

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

$$\Gamma' \vdash_R \neg \neg \phi (R (\theta_1'(\Psi_{GKP}(t'))) (\theta_2'(\Psi_{GKP}(t'))))$$

which is what we needed to show.

$\square$
For all terms $t$, and for all contexts $\Gamma$:

$$\text{if } \forall x \in FV(t) : \Gamma \vdash_N R \, x \, x \text{ then } \Gamma \vdash_N \neg \psi \neg \psi \, R \, (\Psi_{GKP}(t))(\Psi_{GKP}(t))$$

**Proof.** Follows directly from Lemma 8.3.20 by using the identity substitutions. \hfill \Box

**Corollary 8.3.22.** For all terms $t$, and for all contexts $\Gamma$:

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

$$\Gamma \vdash_N \neg \psi \neg \psi \, R \, (\Psi_{GKP}(t))(\Psi_{GKP}(t))$$

**Proof.** Follows directly from Theorem 8.3.21 because if $x$ has a reflexive type we know by the $wk$ rule that $\Gamma \vdash_N R \, x \, x$, and if $R \, x \, x \in \Gamma$ by $hy$ rule we know $\Gamma \vdash_N R \, x \, x$. \hfill \Box

### 8.4 The Tableau Calculus $\mathcal{T}_{\text{Full}}^{\text{res}}$

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

**Proposition 8.4.2.** The tableau calculus $\mathcal{T}_{\text{Full}}^{\text{res}}$ 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 $\mathcal{T}_{\text{Full}}$ is complete. Whenever we apply the Forall, Func=, or the Cut rule with a term $t$ we can apply the Forall$_{\text{res}}$, Func$_{\text{res}}$, or the Cut$_{\text{res}}$ rule respectively with a corresponding term $t'$ that is admissible for the branch. Hence, we can conclude that $\mathcal{T}_{\text{Full}}^{\text{res}}$ is complete. \hfill \Box

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

---

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

**Theorem 8.3.21.** For all terms $t$, and for all contexts $\Gamma$:

$$\text{if } \forall x \in FV(t) : \Gamma \vdash_N R \, x \, x \text{ then } \Gamma \vdash_N \neg \psi \neg \psi \, R \, (\Psi_{GKP}(t))(\Psi_{GKP}(t))$$

**Proof.** Follows directly from Lemma 8.3.20 by using the identity substitutions. \hfill \Box
8.5. \( \Psi_{GKP} \) RESPECTS \( T_{Full_{res}} \)

Lemma 8.5.1. \( \forall x \neg \neg \neg R_{x} x \)

Proof. The model \( M^{\beta / \gamma} \) constructed in Example 5.4 of [4] is in \( M_{\beta} \) and it is not difficult to verify that it satisfies \( \neg \neg \neg R_{x} x \). Thus, it does not satisfy \( \neg \neg \neg R_{x} x \). Since \( N \) is sound with respect to \( M_{\beta} \) we know \( \forall x \neg \neg \neg R_{x} x \).

Proposition 8.5.2. The trivial branch transformation \( \Psi_{GKP}^{\ast} \) that extends \( \Psi_{GKP} \) does not respect \( T_{Full_{res}} \).

Proof. This is because \( \Psi_{GKP}^{\ast} \) does not respect some of the rules in \( T_{Full_{res}} \).

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 \( Closed \neq \) rule. Of course, those added formulas have to be discharged at some point in the process of proving that this branch transformation respects \( T_{Full_{res}} \). We also illustrate how this works. Later on, we will introduce a branch transformation doing exactly this and indeed prove that it respects \( T_{Full_{res}} \).

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

\[
\exists z. z \neq z \\
y \neq y
\]

To show that the \( Closed \neq \) step in this refutation is respected we need to show that the following holds:

\[
\{ \neg \neg R_{y} y, R_{y} y \} \vdash_{N} \perp_{N}
\]

This is obvious.

In order to show that the \( Exists \) step in this refutation is respected we assume

\[
\{ \exists z R_{z} z, \neg \neg R_{y} y, R_{y} y \} \vdash_{N} \perp_{N}
\]

and want to show

\[
\{ \exists z R_{z} z \} \vdash_{N} \perp_{N}.
\]

This follows if we know

\[
\vdash_{N}(R_{y} y \rightarrow \neg \neg R_{y} y \rightarrow \perp_{N}) \rightarrow (\exists z R_{z} z) \rightarrow \perp_{N},
\]

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 \( Closed \neq \) rules looks as follows:

\[
\forall z. z \neq z \\
y \neq y
\]
As shown in the last example, proving that the Closed≠ step in this refutation is respected is obvious. In order to show that the Forall residential step in this refutation is respected we assume

\[ \forall_{\varphi^{GKP}} z. \neg \varphi R z z, \neg \varphi R y y, R y y \]  

and want to show

\[ \forall_{\varphi^{GKP}} z. \neg \varphi R z z \]

This follows if we know

\[ \{((\forall_{\varphi^{GKP}} z. \neg \varphi R z z) \rightarrow (\neg \varphi R y y) \rightarrow (R y y) \rightarrow \bot_N), (\forall_{\varphi^{GKP}} z. \neg \varphi R z z)\} \vdash_N \bot_N. \]

From the restriction on the Forall rule we can infer \( \vdash_N R y y \). Therefore it suffices to show

\[ \{R y y, ((\forall_{\varphi^{GKP}} z. \neg \varphi R z z) \rightarrow (\neg \varphi R y y) \rightarrow (R y y) \rightarrow \bot_N), (\forall_{\varphi^{GKP}} z. \neg \varphi R z z)\} \vdash_N \bot_N \]

We know \( \forall_{\varphi^{GKP}} z. \neg \varphi R z z = \forall z.R z z \rightarrow \neg \varphi R z z \). By instantiating this with \( y \) we get \( R y y \rightarrow \neg \varphi R y y \). By using the assumption \( R y y \) we know \( \neg \varphi R y y \). Using Lemma 6.2.1 with the assumption \( R y y \) we know \( \neg \varphi R y y \). By \( \neg \varphi R y y \) and \( \neg \varphi R y y \) we can infer \( \bot_N \).

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

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

**Lemma 8.5.6**: The branch transformation \( \Psi_{GKP}^* \) respects the following rules:

Closed, Closed¬\( T \), Closed, Cutres, DNeg, And, NegAnd, Or, NegOr, Imp, NegImp

**Proof**: Follows by Lemmas 6.2.3, 6.2.6, 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 \( \Psi_{GKP}^* \) respects the Cutres rule (using Lemma 6.2.7) relies on the restriction imposed by Cutres: any term \( s \) that it adds to a branch is admissible and therefore we know \( \vdash_N R x x \) for each \( x \) that is free in \( s \) but not in the branch. \( \square \)

**Lemma 8.5.7**: (ClosedSym). \( \vdash_N \forall^\sigma x y. (R^\sigma x y) \rightarrow \neg \varphi (R^\sigma y x) \rightarrow \bot_N \)

**Lemma 8.5.8**: (DeMorgan\( \forall \)). \( \vdash_N \forall^\sigma \forall^\sigma. f.(\forall_{\varphi^{GKP}} (\lambda x. \neg \varphi f x)) \rightarrow \bot_N \rightarrow (\neg \varphi (\forall_{\varphi^{GKP}} f)) \rightarrow \bot_N \)

**Lemma 8.5.9**: (DeMorgan\( \exists \)). \( \vdash_N \forall^\sigma \forall^\sigma. f.(\exists_{\varphi^{GKP}} (\lambda x. \neg \varphi f x)) \rightarrow \bot_N \rightarrow (\neg \varphi (\exists_{\varphi^{GKP}} f)) \rightarrow \bot_N \)

**Lemma 8.5.10**: (Bool=). \( \vdash_N \forall^\sigma p q.(p \rightarrow q) \rightarrow (\neg \varphi p) \rightarrow (\neg \varphi q) \rightarrow \bot_N \rightarrow (R^\sigma p q) \rightarrow \bot_N \)

**Lemma 8.5.11**: (BoolExt). \( \vdash_N \forall^\sigma p q.(p \rightarrow (\neg \varphi q)) \rightarrow (q \rightarrow (\neg \varphi p)) \rightarrow \bot_N \rightarrow (R^\sigma p q) \rightarrow \bot_N \)

**Lemma 8.5.12**: (Con). \( \vdash_N \forall^\sigma t u v. (\neg \varphi (R^\sigma t u) \rightarrow (\neg \varphi (R^\sigma t u) \rightarrow \bot_N) \rightarrow (\neg \varphi (R^\sigma t u) \rightarrow \bot_N) \rightarrow (\neg \varphi (R^\sigma t u) \rightarrow \bot_N) \rightarrow (\neg \varphi (R^\sigma t u) \rightarrow \bot_N) \rightarrow (\neg \varphi (R^\sigma t u) \rightarrow \bot_N) \rightarrow \bot_N \)

**Lemma 8.5.13**: (Exists). \( \vdash_N \forall^\sigma x. (R^\sigma x x) \rightarrow (f x) \rightarrow \bot_N \rightarrow (\exists_{\varphi^{GKP}} f) \rightarrow \bot_N \)

We provide a proof that \( \Psi_{GKP}^* \) respects the Exists rule using Lemma 8.5.13 because it is not straightforward.

**Lemma 8.5.14**: The branch transformation \( \Psi_{GKP}^* \) 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

$$\Psi^{\ast}_{GKP}(A \cup \{[s y]^\beta\}) \vdash_{\cdot} \bot_{\cdot}.$$  

We want to show that

$$\Psi^{\ast}_{GKP}(A) \vdash_{\cdot} \bot_{\cdot}.$$  

Note that if $y$ occurs free in $[s y]^\beta$ then

$$\Psi^{\ast}_{GKP}(A \cup \{[s y]^\beta\}) = \Psi^{\ast}_{GKP}(A) \cup \{\Psi_{GKP}([s y]^\beta)\} \cup \{R y y\},$$

otherwise

$$\Psi^{\ast}_{GKP}(A \cup \{[s y]^\beta\}) = \Psi^{\ast}_{GKP}(A) \cup \{\Psi_{GKP}([s y]^\beta)\}.$$  

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

$$\vdash_{\cdot} (\forall x. (R^x x x) \rightarrow ((\Psi_{GKP}(s)) x) \rightarrow \bot_{\cdot}) \rightarrow (\exists_{GKP}^\sigma (\Psi_{GKP}(s))) \rightarrow \bot_{\cdot}.$$  

By using the $wk$ rule we can get

$$\Psi^{\ast}_{GKP}(A) \vdash_{\cdot} (\forall x. (R^x x x) \rightarrow ((\Psi_{GKP}(s)) x) \rightarrow \bot_{\cdot}) \rightarrow (\exists_{GKP}^\sigma (\Psi_{GKP}(s))) \rightarrow \bot_{\cdot}.$$  

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

$$\Psi^{\ast}_{GKP}(A) \vdash_{\cdot} (\forall x. (R^x x x) \rightarrow ((\Psi_{GKP}(s)) x) \rightarrow \bot_{\cdot})$$

we obtain a proof of

$$\Psi^{\ast}_{GKP}(A) \vdash_{\cdot} (\exists_{GKP}^\sigma (\Psi_{GKP}(s))) \rightarrow \bot_{\cdot}.$$  

$\Psi_{GKP}$ is compositional, so

$$\exists_{GKP}^\sigma (\Psi_{GKP}(s)) = \Psi_{GKP}(\exists s).$$  

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

$$\Psi^{\ast}_{GKP}(A) \vdash_{\cdot} \bot_{\cdot}.$$  

It remains to show that

$$\Psi^{\ast}_{GKP}(A) \vdash_{\cdot} (\forall x. (R^x x x) \rightarrow ((\Psi_{GKP}(s)) x) \rightarrow \bot_{\cdot}).$$

By assumption and possibly the $wk$ rule we know

$$\Psi^{\ast}_{GKP}(A) \cup \{\Psi_{GKP}([s y]^\beta)\} \cup \{R y y\} \vdash_{\cdot} \bot_{\cdot}.$$  

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

$$\Psi^{\ast}_{GKP}(A) \vdash_{\cdot} (\forall x. (R^x x x) \rightarrow ((\Psi_{GKP}(s)) x) \rightarrow \bot_{\cdot}).$$

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

\[\blacksquare\]  

Lemma 8.5.15 (Forall$_{res}$). $\vdash_{\cdot} \forall x. (\forall x. f x t) \rightarrow (f t) \rightarrow \bot_{\cdot} \rightarrow (\forall x_{GKP} f) \rightarrow \bot_{\cdot}$
Applying Corollary 8.3.22 with \( \Gamma = \Psi \) and rule we obtain \( w_k \).

By assumption and possibly the \( \rightarrow \) rule we get
\[
\Psi_{GKP}(A \cup \{[s t]^{\beta}\}) \vdash_N \bot_N.
\]

We want to show that
\[
\Psi_{GKP}^*(A) \vdash_N \bot_N.
\]

Note that
\[
\Psi_{GKP}^*(A \cup \{[s t]^{\beta}\}) \subseteq \Psi_{GKP}(A) \cup \{\Psi_{GKP}([s t]^{\beta})\} \cup \{R x x \mid x \in FV(t)\}.
\]

By applying the \( \forall E \) rule twice to Lemma 8.5.15 with \( \Psi_{GKP}(s) \) and \( \Psi_{GKP}(t) \) respectively and then using the \( \omega k \) rule we get
\[
\Psi_{GKP}^*(A) \vdash_N \neg \neg \neg (\forall^\sigma_{GKP} \Psi_{GKP}(t)) \rightarrow ((\Psi_{GKP}(s) \Psi_{GKP}(t)) \rightarrow \bot_N) \rightarrow (\forall^\sigma_{GKP} \Psi_{GKP}(s)) \rightarrow \bot_N.
\]

By the \( \rightarrow E \) rule we know that given proofs of
\[
\Psi_{GKP}^*(A) \vdash_N \neg \neg \neg (\forall^\sigma_{GKP} \Psi_{GKP}(t))
\]
and
\[
\Psi_{GKP}^*(A) \vdash_N (\Psi_{GKP}(s) \Psi_{GKP}(t)) \rightarrow \bot_N
\]
we obtain a proof of
\[
\Psi_{GKP}^*(A) \vdash_N (\forall^\sigma_{GKP} \Psi_{GKP}(s)) \rightarrow \bot_N.
\]

Since \( \Psi_{GKP} \) is compositional,
\[
(\forall^\sigma_{GKP} \Psi_{GKP}(s)) = \Psi_{GKP}(\forall s).
\]

Since \( \forall s \in A \), we know \( \Psi_{GKP}(\forall s) \in \Psi_{GKP}^*(A) \) thus \( \Psi_{GKP}^*(A) \vdash_N \Psi_{GKP}(\forall s) \) by the \( hy \) rule. Hence, by applying the \( \rightarrow E \) rule we obtain
\[
\Psi_{GKP}^*(A) \vdash_N \bot_N.
\]

It remains to show two things, namely
\[
\Psi_{GKP}^*(A) \vdash_N \neg \neg \neg (\forall^\sigma_{GKP} \Psi_{GKP}(t))
\]
and
\[
\Psi_{GKP}^*(A) \vdash_N (\Psi_{GKP}(s) \Psi_{GKP}(t)) \rightarrow \bot_N.
\]

Applying Corollary 8.3.22 with \( \Gamma = \Psi_{GKP}^*(A) \) and making use of the restriction on the \( \forall all \) rule we obtain
\[
\Psi_{GKP}^*(A) \vdash_N \neg \neg \neg (\forall^\sigma_{GKP} \Psi_{GKP}(t)) \Psi_{GKP}(t)).
\]

By assumption and possibly the \( \omega k \) rule we know
\[
\Psi_{GKP}^*(A) \cup \{\Psi_{GKP}([s t]^{\beta})\} \cup \{R x x \mid x \in FV(t)\} \vdash_N \bot_N.
\]
We know that $\Psi_{GKP}$ is compositional and respects beta, therefore by applying the $\rightarrow I$ rule we obtain

$$\Psi_{GKP}(A) \cup \{R \ x \ x \ | \ x \in FV(t)\} \vdash_{\rightarrow} (\Psi_{GKP}(s) \ \Psi_{GKP}(t)) \rightarrow \bot_N.$$

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

$$\Psi_{GKP}(A) \vdash_{\rightarrow} (\Psi_{GKP}(s) \ \Psi_{GKP}(t)) \rightarrow \bot_N.$$

\[\square\]

**Lemma 8.5.17** (FuncExt). $\vdash_{\rightarrow} \forall s \tau k h. \neg \phi (\tau \s p \ h \ h) \rightarrow (\forall s \tau x . (\tau \s p \ x \ x) \rightarrow \neg \phi (\tau \s p (k \ x \ y) (h \ x)) \rightarrow \bot_N) \rightarrow \neg \phi (\tau \s p k \ h) \rightarrow \bot_N$.

The proof that $\Psi_{GKP}^*$ respects the FuncExt rule using Lemma 8.5.17 uses arguments similar to 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$_{res}$). $\vdash_{\rightarrow} \forall s \tau k h. \forall s \tau x . \neg \phi (\tau \s p x \ x) \rightarrow \neg \phi (\tau \s p t \ t) \rightarrow ((\tau \s p (k \ t \ y) (h \ t)) \rightarrow \bot_N) \rightarrow (\tau \s p k \ h) \rightarrow \bot_N$.

**Lemma 8.5.19** (Closed$\neq$). $\vdash_{\rightarrow} \forall s \tau x . \neg \phi (\tau \s p x \ x) \rightarrow \neg \phi (\tau \s p x \ x) \rightarrow \bot_N$.

**Lemma 8.5.20** (Mat). $\vdash_{\rightarrow} \forall s \tau_1 \ s_1 \ s_2 \ s_2 \ s_1 \ t_1 . \forall s \tau_2 \ s_2 \ t_2 . \ldots . \forall s \tau_n \ s_n \ t_n . \neg \phi (\tau \s p_1 s_1 t_1) \rightarrow \bot_N) \rightarrow (\neg \phi (\tau \s p_2 s_2 t_2) \rightarrow \bot_N) \rightarrow \ldots \rightarrow (\neg \phi (\tau \s p n s_n t_n) \rightarrow \bot_N) \rightarrow p s_1 s_2 \ldots s_n \rightarrow \neg \phi (p t_1 t_2 \ldots t_n) \rightarrow \bot_N$.

**Lemma 8.5.21** (Dec). $\vdash_{\rightarrow} \forall s \tau_1 \ s_1 \ s_2 \ s_2 \ s_2 \ s_2 \ s_1 \ t_1 . \forall s \tau_2 \ s_2 \ t_2 . \ldots . \forall s \tau_n \ s_n \ t_n . \neg \phi (\tau \s p_1 s_1 t_1) \rightarrow \bot_N) \rightarrow (\neg \phi (\tau \s p_2 s_2 t_2) \rightarrow \bot_N) \rightarrow \ldots \rightarrow (\neg \phi (\tau \s p n s_n t_n) \rightarrow \bot_N) \rightarrow \neg \phi (\tau \s p (h s_1 s_2 \ldots s_n) (h t_1 t_2 \ldots t_n)) \rightarrow \bot_N$.

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 $\Psi_{GKP}^*$ 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, 8.5.15, 8.5.18, 8.5.20, 8.5.21, 8.5.19, and 8.5.7. \[\square\]

**Theorem 8.5.23**. The Girard-Kuroda-Per transformation $\Psi_{GKP}$ 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 $\{\Psi_{GKP}(s)\} \vdash_{\rightarrow} \bot_N$.

**Proof.** The proof follows directly from Corollary 5.2.2 and from Theorem 8.5.23. \[\square\]
CHAPTER 8. THE GIRARD-KURODA-PER TRANSFORMATION
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 $\Psi$ respects a tableau calculus $T$ then $\Psi$ maps any closed $T$-refutable formula $s$ to an ND-refutable formula $\Psi(s)$. Hence, there is a formula $s'$ that is semantically equivalent to $s$, namely $s' := (\Psi(s) \rightarrow \bot_N) \rightarrow \bot_N$, 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 $\Psi_G$, $\Psi_{GK}$, and $\Psi_{GKP}$ and corresponding branch transformations. We then showed which tableau rules are respected by each of the branch transformations and which are not.

If $T$ is a tableau calculus containing only rules that are respected by a branch transformation $\Psi^*$ that extends a logical transformation $\Psi$, then $\Psi$ respects $T$. We showed that $\Psi_G$ respects the tableau calculus $T_{\rightarrow \bot}$. Moreover, that the branch transformation trivially extending $\Psi_G$ respects all the tableau rules except for DeMorgan$\forall$, BoolExt, and FuncExt. Thus, $\Psi_G$ respects all tableau calculi that do not include those three rules. In addition we showed that $\Psi_{GK}$ respects the tableau calculus for elementary type theory $T_{elem}$. Finally, we showed that $\Psi_{GKP}$ respects a complete tableau calculus for higher-order logic $T_{Full_{fin}}$. 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 $T_{Full}$. 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 natural 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


Appendix A

Girard Transformation

A.1 Defining basic types i, o

Parameter i : Type.
Definition o := Prop.

A.2 Girard Transformation

Definition Neg := fun p : o ⇒ p → Bot.
Definition Imp := fun p q : o ⇒ p → q.
Definition Forall (σ : Type) := fun f : (σ → o) ⇒ (∀ x : σ, 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 (σ : Type) (M : σ) (N : σ)
  := ∀ f : (σ → o), (f M) → f N.
Definition Exists (σ : Type) (M : (σ → o))

A.2.1 Short Hand

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

A.3 Symmetry of equality

Definition Sym (σ : Type)
  := ∀ x y : σ, (Equal σ x y) → (Equal σ y x)
  := fun x y u v ⇒ u (fun z ⇒ f z → f x) (fun w ⇒ w) v.
A.4 Transitivity of equality

Definition \( \text{Tra} (\sigma : \text{Type}) \)
\[ \forall x y z : \sigma, (\text{Equal} \sigma x y) \rightarrow (\text{Equal} \sigma y z) \rightarrow (\text{Equal} \sigma x z) \]
\[ := \text{fun} \ x \ y \ z \ u \ v \ p \ w \Rightarrow v \ p \ (u \ p \ w). \]

A.5 Negative transitivity

Definition \( \text{TraNeg} (\sigma : \text{Type}) \)
\[ \forall x y z : \sigma, (\text{Equal} \sigma x y) \rightarrow \neg (\text{Equal} \sigma x z) \rightarrow \neg (\text{Equal} \sigma y z) \]
\[ := \text{fun} \ x \ y \ z \ u \ v \ w \Rightarrow v \ (\text{Tra} \sigma x y z u w). \]

A.6 Lemmas

A.6.1 Closed False Rule

Definition \( \text{closedfalserule} : \text{Bot} \rightarrow \text{Bot} := \text{fun} \ u \Rightarrow u. \)

A.6.2 Closed Not True Rule

Definition \( \text{closednottruerule} : (\neg \text{Top}) \rightarrow \text{Bot} := \text{fun} \ u \Rightarrow u \ (\text{fun} \ p \ v \Rightarrow v). \)

A.6.3 Closed Rule

Definition \( \text{closedrule} : \forall p : o, p \rightarrow (\neg p) \rightarrow \text{Bot} \)
\[ := \text{fun} \ p \ u \ v \Rightarrow (v \ u). \]

A.6.4 Closed Neg Equal Rule

Definition \( \text{closednegequalrule} (\sigma : \text{Type}) \)
\[ : \forall (s : \sigma), \neg (\text{Equal} \sigma s s) \rightarrow \text{Bot} \]
\[ := \text{fun} \ p \ u \Rightarrow u \ (\text{fun} \ f \ v \Rightarrow v). \]

A.6.5 Closed Symmetric Rule

Definition \( \text{closedsymrule} (\sigma : \text{Type}) \)
\[ : \forall (s t : \sigma), (\text{Equal} \sigma s t) \rightarrow \neg (\text{Equal} \sigma t s) \rightarrow \text{Bot} \]
\[ := \text{fun} \ s \ t \ u \ v \Rightarrow v \ (\text{Sym} \sigma s t u). \]

A.6.6 Double Negation Rule

Definition \( \text{dnegrule} \)
\[ : \forall p : o, (p \rightarrow \text{Bot}) \rightarrow (\neg (\neg p)) \rightarrow \text{Bot} \]
\[ := \text{fun} \ p \ u \ v \Rightarrow v \ u. \]
A.6.7 Cut Rule

Definition `cutrule`:
\[
\forall p : o, (p \rightarrow Bot) \rightarrow ((\neg p) \rightarrow Bot) \rightarrow Bot
\]
\[
:= \text{fun } p u v \Rightarrow v u.
\]

A.6.8 Implication Rule

Definition `imprule`:
\[
\forall p q : o, ((\neg p) \rightarrow Bot) \rightarrow (q \rightarrow Bot) \rightarrow (\text{Imp } p q) \rightarrow Bot
\]
\[
:= \text{fun } p q u v w \Rightarrow u (\text{fun } u1 : p \Rightarrow v (w u1)).
\]

A.6.9 Negative Implication Rule

Definition `negimprule`:
\[
\forall p q : o, (p \rightarrow (\neg q) \rightarrow Bot) \rightarrow (\neg (\text{Imp } p q)) \rightarrow Bot
\]
\[
:= \text{fun } p q u v \Rightarrow v (\text{fun } w1 : p \Rightarrow u w1 (\text{fun } z : q \Rightarrow v (\text{fun } w2 : p \Rightarrow z)) q).
\]

A.6.10 And Rule

Definition `andrule`:
\[
\forall p q : o, (p \rightarrow q \rightarrow Bot) \rightarrow (\text{And } p q) \rightarrow Bot
\]
\[
:= \text{fun } p q u v \Rightarrow v Bot u.
\]

A.6.11 Or Rule

Definition `orrule`:
\[
\forall p q : o, (p \rightarrow Bot) \rightarrow (q \rightarrow Bot) \rightarrow (\text{Or } p q) \rightarrow Bot
\]
\[
:= \text{fun } p q u v w \Rightarrow w Bot u v.
\]

A.6.12 Neg And Rule

Definition `negandrule`:
\[
\forall p q : o, ((\neg p) \rightarrow Bot) \rightarrow ((\neg q) \rightarrow Bot) \rightarrow (\neg (\text{And } p q)) \rightarrow Bot
\]
\[
:= \text{fun } p q u v \Rightarrow u (\text{fun } w : p \Rightarrow v (\text{fun } u1 w1 \Rightarrow w1 (\text{fun } u2 \Rightarrow w2 (\text{fun } p u3 \Rightarrow u3 u1 u2)))).
\]

A.6.13 Neg Or Rule

Definition `negorrule`:
\[
\forall p q : o, ((\neg p) \rightarrow (\neg q) \rightarrow Bot) \rightarrow ((\neg (\text{Or } p q)) \rightarrow Bot
\]
\[
:= \text{fun } p q u v w \Rightarrow u (\text{fun } w : u \Rightarrow v (\text{fun } r u1 u2 \Rightarrow w (\text{fun } u3 \Rightarrow u3 u1 u2))).
\]

A.6.14 Forall Rule

Definition `forallrule` `(sigma : Type)`:
\[
\forall f : sigma \rightarrow o, (t : sigma), ((f t) \rightarrow Bot) \rightarrow (\text{Forall } sigma f) \rightarrow Bot
\]
\[
:= \text{fun } f t u v \Rightarrow u (v t).
\]
A.6.15 Exists Rule

Definition existsrule (sigma : Type)

: \( \forall f : (\sigma \rightarrow o), (\forall x : \sigma, (f x) \rightarrow Bot) \rightarrow (Exists \sigma f) \rightarrow Bot \)

:= fun f u v ⇒ v Bot u.

A.6.16 DeMorgan Exists Rule

Definition demorganexistsrule (sigma : Type)

: \( \forall (f : \sigma \rightarrow o), ((\forall x : \sigma, f x) \rightarrow Bot) \rightarrow (Neg (Exists \sigma f)) \rightarrow Bot \)

:= fun f u v ⇒ u (fun w1 ⇒ v (fun w2 ⇒ w2 w1))

A.6.17 Boolean Equality Rule

Definition booleqrule

: \( \forall (p q : o), (p \rightarrow q \rightarrow Bot) \rightarrow ((\neg p) \rightarrow (\neg q) \rightarrow Bot) \rightarrow (Equal o p q) \rightarrow Bot \)

:= fun p q u1 u2 u3 ⇒ u2 (fun v1 ⇒ u1 v1 (fun w ⇒ w w u3 v)) (u3 (fun w ⇒ Neg w) (fun v ⇒ u1 v (u3 (fun w ⇒ w)))).

A.6.18 Leibniz Rule - not used in the Thesis

Definition leibnizrule (sigma : Type)

: \( \forall (x y : \sigma), ((\forall f : \sigma \rightarrow o, (f x) \rightarrow f y)) \rightarrow (Equal \sigma x y) \rightarrow Bot \)

:= fun x y u v ⇒ u v.

A.6.19 Functional Equality Rule

Definition funceqrule (sigma tau : Type)

: \( \forall (k h : (\sigma \rightarrow \tau \rightarrow i)), ((Equal \tau (k t) (h t)) \rightarrow Bot) \rightarrow (Equal (\sigma \rightarrow \tau) k h) \rightarrow Bot \)

:= 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)

: \( \forall (f : (\sigma \rightarrow \tau \rightarrow o)) (s1 t1 : \sigma) (s2 t2 : \tau), (Neg(Equal sigma s1 t1) \rightarrow Bot) \rightarrow (Neg(Equal tau s2 t2) \rightarrow Bot) \rightarrow (f s1 s2) \rightarrow Bot \)

:= 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)

: \( \forall (h : (\sigma \rightarrow \tau \rightarrow i)) (s1 t1 : \sigma) (s2 t2 : \tau), (Neg(Equal sigma s1 t1) \rightarrow Bot) \rightarrow (Neg(Equal tau s2 t2) \rightarrow Bot) \rightarrow...


A. 6. LEMMAS

\[ \text{Neg}(\text{Equal } i (h \ s1 \ s2) (h \ t1 \ t2)) \rightarrow \text{Bot} \]

:= 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:

\[ \forall (s1 \ t1 \ s2 \ t2 : i), \ (\text{Neg}(\text{Equal } i \ s1 \ s2) \rightarrow \text{Neg}(\text{Equal } i \ t1 \ s2) \rightarrow \text{Bot}) \rightarrow \]

(\text{Neg}(\text{Equal } i \ s1 \ t2) \rightarrow \text{Neg}(\text{Equal } i \ t1 \ t2) \rightarrow \text{Bot}) \rightarrow

:= 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 t2 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 Neg := fun p : o ⇒ p → Bot.
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))

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.
APPENDIX B. GIRARD-KURODA TRANSFORMATION

B.4 Transitivity of equality

Definition $\text{Tra} (\sigma : \text{Type})$

$\forall x y z : \sigma, (\text{Equal } \sigma x y) \rightarrow (\text{Equal } \sigma y z) \rightarrow (\text{Equal } \sigma x z)$

$:= \text{fun } x y z u v p w \Rightarrow v (u p w)$. 

B.5 Negative transitivity

Definition $\text{TraNeg} (\sigma : \text{Type})$

$\forall x y z : \sigma, (\text{Equal } \sigma x y) \rightarrow \text{Neg}(\text{Equal } \sigma x z) \rightarrow \text{Neg}(\text{Equal } \sigma y z)$

$:= \text{fun } x y z u v w \Rightarrow v (\text{Tra } \sigma x y z u w)$. 

B.6 Lemmas

B.6.1 Closed False Rule

Definition $\text{closedfalserule} : \text{Bot} \rightarrow \text{Bot} := \text{fun } u \Rightarrow u$. 

B.6.2 Closed Not True Rule

Definition $\text{closednottruerule} : (\text{Neg } \text{Top}) \rightarrow \text{Bot} := \text{fun } u \Rightarrow u (\text{fun } p v \Rightarrow v)$. 

B.6.3 Closed Rule

Definition $\text{closedrule} :$

$\forall p : o, p \rightarrow (\text{Neg } p) \rightarrow \text{Bot}$

$:= \text{fun } p u v \Rightarrow (v u)$. 

B.6.4 Closed Neg Equal Rule

Definition $\text{closednegequalrule} (\sigma : \text{Type})$

$\forall (s : \sigma), \text{Neg}(\text{Equal } \sigma s s) \rightarrow \text{Bot}$

$:= \text{fun } p u \Rightarrow u (\text{fun } f v \Rightarrow v)$. 

B.6.5 Closed Symmetric Rule

Definition $\text{closedsymrule} (\sigma : \text{Type})$

$\forall (s t : \sigma), (\text{Equal } \sigma s t) \rightarrow \text{Neg}(\text{Equal } \sigma t s) \rightarrow \text{Bot}$

$:= \text{fun } s t u v \Rightarrow v (\text{Sym } \sigma s t u)$. 

B.6.6 Double Negation Rule

Definition $\text{dnegrule}$

$\forall p : o, (p \rightarrow \text{Bot}) \rightarrow \text{Neg}(\text{Neg } p) \rightarrow \text{Bot}$

$:= \text{fun } p u v \Rightarrow v u$. 

B.6.7 Cut Rule

Definition cutrule:
\[ \forall p : o, (p \to \text{Bot}) \to ((\text{Neg } p) \to \text{Bot}) \to \text{Bot} \]
\[ := \text{fun } p u v \Rightarrow v u. \]

B.6.8 Implication Rule

Definition imprule:
\[ \forall p q : o, ((\text{Neg } p) \to \text{Bot}) \to (q \to \text{Bot}) \to (\text{Imp } p q) \to \text{Bot} \]
\[ := \text{fun } p q u v w \Rightarrow u (\text{fun } u1 : p \Rightarrow v (w u1)). \]

B.6.9 Negative Implication Rule

Definition negimprule:
\[ \forall p q : o, (p \to ((\text{Neg } q) \to \text{Bot}) \to (\text{Neg } (\text{Imp } p q)) \to \text{Bot} \]
\[ := \text{fun } p q u v \Rightarrow v (\text{fun } w1 : p \Rightarrow u w1 (\text{fun } z : q \Rightarrow v (\text{fun } w2 : p \Rightarrow z)) q). \]

B.6.10 And Rule

Definition andrule:
\[ \forall p q : o, (p \to q \to \text{Bot}) \to (\text{And } p q) \to \text{Bot} \]
\[ := \text{fun } p q u v \Rightarrow v \text{ Bot } u. \]

B.6.11 Or Rule

Definition orrule:
\[ \forall p q : o, (p \to \text{Bot}) \to (q \to \text{Bot}) \to (\text{Or } p q) \to \text{Bot} \]
\[ := \text{fun } p q u v w \Rightarrow w \text{ Bot } u v. \]

B.6.12 Neg And Rule

Definition negandrule:
\[ \forall p q : o, ((\text{Neg } p) \to \text{Bot}) \to ((\text{Neg } q) \to \text{Bot}) \to \text{Neg}(\text{And } p q) \to \text{Bot} \]
\[ := \text{fun } p q u v \Rightarrow u (\text{fun } w : p \Rightarrow \text{Bot } u w) (\text{fun } w : q \Rightarrow \text{Bot } w u) ((\text{fun } u : \text{Bot } w u) (\text{fun } u : \text{Bot } u w)). \]

B.6.13 Neg Or Rule

Definition negorrule:
\[ \forall p q : o, ((\text{Neg } p) \to (\text{Neg } q) \to \text{Bot}) \to \text{Neg}(\text{Or } p q) \to \text{Bot} \]
\[ := \text{fun } p q u v \Rightarrow u (\text{fun } w : \text{Bot } u w) (\text{fun } u : w \Rightarrow \text{Bot } u) (\text{fun } u : \text{Bot } w u) ((\text{fun } u : w \Rightarrow \text{Bot } u w) (\text{fun } u : w \Rightarrow \text{Bot } u w)). \]

B.6.14 Forall Rule

Definition forallrule (\sigma : \text{Type})
\[ \forall (f : \sigma \to o) (t : \sigma), ((f t) \to \text{Bot}) \to (\forall \sigma f) \to \text{Bot} \]
\[ := \text{fun } f t u v \Rightarrow v t u. \]
APPENDIX B. GIRARD-KURODA TRANSFORMATION

B.6.15 DeMorgan Forall Rule

Definition demorganforallrule (sigma : Type)
\[
\forall (f : sigma \to o), ((\exists \neg sigma f) \to Bot) \to (\neg (\forall sigma f)) \to Bot \\
\] := \(\text{fun } f \ u \ v \Rightarrow v \ (\text{fun } x \ w \Rightarrow u \ (\text{fun } z \ z \Rightarrow z \ x \ w))\).

B.6.16 Exists Rule

Definition existsrule (sigma : Type)
\[
\forall f : (sigma \to o), ((\forall x : sigma, (f x) \to Bot) \to (\exists sigma f)) \to Bot \\
\] := \(\text{fun } f \ u \ v \Rightarrow v \ Bot \ u\).

B.6.17 DeMorgan Exists Rule

Definition demorganexistsrule (sigma : Type)
\[
\forall (f : sigma \to o), ((\forall \neg sigma f) \to Bot) \to (\neg (\exists sigma f)) \to Bot \\
\] := \(\text{fun } f \ u \ v \Rightarrow u \ (\text{fun } y \ w1 \Rightarrow w1 \ (\text{fun } w2 \Rightarrow v \ (\text{fun } r \ w3 \Rightarrow w3 \ y \ w2)))\).

B.6.18 Boolean Equality Rule

Definition booleqrule :
\[
\forall (p q : o), ((p \to q \to Bot) \to ((\neg p) \to (\neg q) \to Bot) \to (\text{Equal } o \ p \ q) \to Bot \\
\] := \(\text{fun } p \ q \ u1 \ u2 \ u3 \Rightarrow u2 \ (\text{fun } v \Rightarrow u1 \ v \ (u3 \ (\text{fun } w \Rightarrow w))) \ (u3 \ (\text{fun } w \Rightarrow \neg w) \ (\text{fun } v \Rightarrow u1 \ v \ (u3 \ (\text{fun } w \Rightarrow w) \ v)))\).

B.6.19 Leibniz Rule

Definition leibnizrule (sigma : Type)
\[
\forall (x y : sigma), ((\forall sigma \to o) \ (\text{fun } f \Rightarrow f x \to f y)) \to Bot) \to (\text{Equal } sigma \ x \ y) \\
\] := \(\text{fun } x \ y \ u v \Rightarrow u \ (\text{fun } f u1 \Rightarrow u1 \ (v f))\).

B.6.20 Functional Equality Rule

Definition funcEqrule (sigma tau : Type)
\[
\forall (k h : (sigma \to tau)) (t : sigma), ((\text{Equal } tau \ (k t) \ (h t)) \to Bot) \to (\text{Equal } (sigma \to tau) \ k \ h) \\
\] := \(\text{fun } k \ h \ u1 \ u2 \Rightarrow u1 \ (u2 \ (\text{fun } r \Rightarrow \text{Equal } tau \ (k t) \ (r t)) \ (\text{fun } f v \Rightarrow v)))\).

B.6.21 Mating Rule - 2 arguments

Definition matingrule_2 (sigma tau : Type)
\[
\forall (f : (sigma \to tau \to o)) (s1 t1 : sigma) (s2 t2 : tau), \\
(\neg (\text{Equal } sigma \ s1 \ t1) \to Bot) \to (\neg (\text{Equal } tau \ s2 \ t2) \to Bot) \\
\] := \(\text{fun } f \ s1 \ t1 \ s2 \ t2 \ u1 \ u2 \ u3 \ u4 \Rightarrow u1 \ (\text{fun } v1 \Rightarrow u2 \ (\text{fun } v2 \Rightarrow u4 \ (v2 \ (\text{fun } f \Rightarrow f t1 x) \ (v1 \ (\text{fun } x \Rightarrow f x s2) \ u3))))\).
B.6.22  Decomposition Rule - 2 arguments

Definition decompositionrule₂ (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 \( i, o \)

Parameter \( i : \text{Type}. \)
Definition \( o := \text{Prop}. \)

C.2 Girard-Kuroda-Per Transformation

Definition \( \text{Bot} := \forall p : o, p \rightarrow p. \)
Definition \( \text{Imp} := \text{fun} p o \rightarrow p \rightarrow p. \)
Definition \( \text{And} (M : o) (N : o) := (\forall p : o, (M \rightarrow N \rightarrow p) \rightarrow p). \)
Definition \( \text{Or} (M : o) (N : o) := (\forall p : o, (M \rightarrow p) \rightarrow (N \rightarrow p) \rightarrow p). \)
Definition \( \text{R}_i (M : i) (N : i) := (\forall p : i \rightarrow o, (p M) \rightarrow (p N)). \)
Definition \( \text{R}_	ext{ar} (\sigma \tau : \text{Type}) (R_\sigma : \sigma \rightarrow \sigma \rightarrow o) \)
\( (R_\tau : \tau \rightarrow \tau \rightarrow o) (M : \sigma \rightarrow \tau) (N : \sigma \rightarrow \tau) \)
\( := \forall x y : \sigma, R_\sigma x y \rightarrow \text{Neg}(\text{Neg}(R_\tau (M x) (N y))). \)
Definition \( \text{Forall} (\sigma \text{ Type}) (R_\sigma : \sigma \rightarrow \sigma \rightarrow o) \)
\( := \text{fun} f : (\sigma \rightarrow o) \Rightarrow \forall t : \sigma, \text{Imp} (R_\sigma t t) (\text{Neg}(\text{Neg}(f t))). \)
Definition \( \text{Exists} (\sigma \text{ Type}) (R_\sigma : \sigma \rightarrow \sigma \rightarrow o) \)
\( := \text{fun} f : (\sigma \rightarrow o) \Rightarrow \forall p : o, (\forall x : \sigma, (R_\sigma x x) \rightarrow (f x) \rightarrow p) \rightarrow p. \)

C.2.1 Short Hand

Definition \( \text{ForallNeg} (\sigma : \text{Type}) (R_\sigma : \sigma \rightarrow \sigma \rightarrow o) \)
\( := \text{fun} f : (\sigma \rightarrow o) \Rightarrow \forall t : \sigma, \text{Imp} (R_\sigma t t) (\text{Neg}(\text{Neg}(f t))). \)
Definition \( \text{ExistsNeg} (\sigma : \text{Type}) (R_\sigma : \sigma \rightarrow \sigma \rightarrow o) (M : (\sigma \rightarrow o)) \)
\( := \text{Exists} \sigma R_\sigma (\text{fun} x : \sigma \Rightarrow (\text{Neg}(M x))). \)
C.3 Some Definitions

C.3.1 Definition of SymNeg

Definition $\text{SymNeg}(\sigma : \text{Type}) (R_{\sigma} : \sigma \to \sigma \to o)$

$(\forall x y : \sigma, (R_{\sigma x y} \to (R_{\sigma y x}))) 
\to

(\forall x y : \sigma, \text{Neg}(R_{\sigma x y}) \to \text{Neg}(R_{\sigma y x}))

:= \forall u x y v w \Rightarrow v (u x w).

C.3.2 Recursive definition of Sym

Definition $\text{Sym}_o$

$(\forall x y : o, (R_{o x y} \to (R_{o x y})))
\Rightarrow

(\forall x y : o, (R_{o x y} \to (R_{o y x})))$

:= \forall u x y v w \Rightarrow v (u y x w).

Definition $\text{Sym}_i$

$(\forall x y : i, (R_{i x y} \to (R_{i x y})))
\Rightarrow

(\forall x y : i, (R_{i x y} \to (R_{i y x})))$

:= \forall u x y v w \Rightarrow u (v y y u).

Definition $\text{Sym}_{\text{ar}}(\sigma \tau : \text{Type}) (R_{\sigma} : \sigma \to \sigma \to o)
(R_{\tau} : \tau \to \tau \to o)$

$(\forall u x y : \sigma, (\text{Sym}_{\sigma} x y v w))
\Rightarrow

(\forall u x y : \tau, (\text{Sym}_{\tau} x y v w))$

:= \forall u x y v w \Rightarrow v (u y y u).

C.3.3 Recursive definition of Tra

Definition $\text{Tra}_o$

$(\forall x y z : o, (R_{o x y} \to (R_{o x y})))
\Rightarrow

(\forall x y z : o, (R_{o x y} \to (R_{o y z})))$

:= \forall u x y v w \Rightarrow v (u y y u).

Definition $\text{Tra}_i$

$(\forall x y z : i, (R_{i x y} \to (R_{i x y})))
\Rightarrow

(\forall x y z : i, (R_{i x y} \to (R_{i y x})))$

:= \forall u x y v w \Rightarrow v (u y y u).

Definition $\text{Tra}_{\text{ar}}(\sigma \tau : \text{Type}) (R_{\sigma} : \sigma \to \sigma \to o)
(R_{\tau} : \tau \to \tau \to o)$

$(\forall u x y : \sigma, (\text{Tra}_{\sigma} x y v w))
\Rightarrow

(\forall u x y : \tau, (\text{Tra}_{\tau} x y v w))$

:= \forall u x y v w \Rightarrow v (u y y u).
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*

```
\forall (p : o), \neg (\neg p) \rightarrow (q \rightarrow Bot) \rightarrow (Imp p q) \rightarrow Bot
\quad := \text{fun } p q u v w \Rightarrow u (\text{fun } u1 : p \Rightarrow v (w u1)).
```

C.4.9 Negative Implication Rule

Definition *negimprule*

```
\forall (p : o), \neg (p \rightarrow (\neg q)) \rightarrow (Neg Imp p q) \rightarrow Bot
\quad := \text{fun } p q u v w \Rightarrow u (\text{fun } u1 : p \Rightarrow v (w u1)).
```

C.4.10 And Rule

Definition *andrule*

```
\forall (p : o), (p \rightarrow q) \rightarrow (And p q) \rightarrow Bot
\quad := \text{fun } p q u v w \Rightarrow u (\text{fun } u1 : p \Rightarrow v (w u1)).
```

C.4.11 Or Rule

Definition *orrule*

```
\forall (p : o), (p \rightarrow Bot) \rightarrow (q \rightarrow Bot) \rightarrow (Or p q) \rightarrow Bot
\quad := \text{fun } p q u v w \Rightarrow u (\text{fun } u1 : p \Rightarrow v (w u1)).
```

C.4.12 Neg And Rule

Definition *negandrule*

```
\forall (p : o), \neg (\neg p) \rightarrow (\neg q) \rightarrow Bot
\quad := \text{fun } p q u v w \Rightarrow u (\text{fun } u1 : p \Rightarrow v (w u1)).
```

C.4.13 Neg Or Rule

Definition *negorrule*

```
\forall (p : o), \neg (\neg p) \rightarrow (\neg q) \rightarrow Bot
\quad := \text{fun } p q u v w \Rightarrow u (\text{fun } u1 : p \Rightarrow v (w u1)).
```

C.4.14 Restricted Forall Rule

Definition *forallrule* (\sigma : Type) (R_{\sigma} : \sigma \times \sigma \rightarrow o)

```
\forall (f : \sigma \rightarrow o) (t : \sigma), \neg (\neg (R_{\sigma} \sigma t t)) \rightarrow (f t) \rightarrow Bot
\quad := \text{fun } f t u v w \Rightarrow u (\text{fun } u1 : t \Rightarrow v (w u1)).
```

C.4.15 DeMorgan Forall Rule

Definition *demorganforallrule* (\sigma : Type) (R_{\sigma} : \sigma \times \sigma \rightarrow o)

```
\forall (f : \sigma \rightarrow o), ((\exists \neg \sigma \sigma f) \rightarrow (f t) \rightarrow Bot)
\quad := \text{fun } f u v w \Rightarrow u (\text{fun } u1 : t \Rightarrow v (w u1)).
```
C.4. LEMMAS

C.4.16 Exists Rule

Definition existsrule (sigma : Type) (R_{\sigma} : \sigma \rightarrow \sigma \rightarrow o)
: \forall (f : \sigma \rightarrow o), (\forall x : \sigma, (R_{\sigma} x x) \rightarrow (f x) \rightarrow Bot) \rightarrow
  \exists (\exists (\exists \sigma R_{\sigma} f) \rightarrow Bot)
:= \text{fun} f u v \Rightarrow v \text{Bot} u.

C.4.17 DeMorgan Exists Rule

Definition demorganexistsrule (sigma : Type) (R_{\sigma} : \sigma \rightarrow \sigma \rightarrow o)
: \forall (f : \sigma \rightarrow o), ((\forall Neg (\exists \exists \sigma R_{\sigma} f) \rightarrow Bot) \rightarrow
  \neg (\exists (\exists \sigma R_{\sigma} f)) \rightarrow Bot)
:= \text{fun} f u v \Rightarrow u (\text{fun} x u1 u2 \Rightarrow u2 (\text{fun} u3 \Rightarrow v (\text{fun} p v1 \Rightarrow v1 x u1 u3))).

C.4.18 Boolean Equality Rule

Definition boolextrule
: \forall (p q : o), (p \rightarrow q \rightarrow Bot) \rightarrow ((\neg p) \rightarrow (\neg q) \rightarrow Bot) \rightarrow (R_{\neg o} p q) \rightarrow Bot
:= \text{fun} p q u v w \Rightarrow
  w \text{Bot} (\text{fun} u1 u2 \Rightarrow v (\text{fun} v1 \Rightarrow u v1 (u1 v1)) (\text{fun} v2 \Rightarrow u (u2 v2) v2)).

C.4.19 Boolean Extensionality Rule

Definition boolextrule
: \forall (p q : o), (p \rightarrow q \rightarrow Bot) \rightarrow (q \rightarrow (\neg p) \rightarrow Bot) \rightarrow (\neg (\exists (\exists o p q) \rightarrow Bot)
:= \text{fun} p q u v w \Rightarrow
  w (\text{fun} r w1 \Rightarrow
    w1 (\text{fun} z1 \Rightarrow
      u z1 (\text{fun} z2 \Rightarrow
        w (\text{fun} r2 w2 \Rightarrow w2 (\text{fun} z3 \Rightarrow z2) (\text{fun} z4 \Rightarrow z1))))) q)
    (\text{fun} z1 \Rightarrow
      v z1 (\text{fun} z2 \Rightarrow
        w (\text{fun} r2 w2 \Rightarrow w2 (\text{fun} z3 \Rightarrow z1) (\text{fun} z4 \Rightarrow z2))))) p).

C.4.20 Restricted Functional Equality Rule

Definition funcceqrule (sigma tau : Type) (R_{\sigma} : \sigma \rightarrow \sigma \rightarrow o)
(R_{\tau} : \tau \rightarrow \tau \rightarrow o)
: \forall (k h : (\sigma \rightarrow \tau)) (t : \sigma), (\neg (\exists \neg (R_{\sigma} t t)) \rightarrow
  ((R_{\tau} (k t)(h t)) \rightarrow Bot) \rightarrow (R_{\neg \alpha} sigma tau R_{\sigma} sigma R_{\tau} tau k h) \rightarrow Bot
:= \text{fun} k h t u v w \Rightarrow u (\text{fun} u1 \Rightarrow w t t u1 v).

C.4.21 Functional Extensionality Rule

Definition funcextrule (sigma tau : Type) (R_{\sigma} : \sigma \rightarrow \sigma \rightarrow o)
(R_{\tau} : \tau \rightarrow \tau \rightarrow o)
(Sym_{\sigma} : \forall (x : \sigma), (R_{\sigma} x x) \rightarrow (R_{\sigma} x x))
(Tra_{\sigma} : \forall x y z : \sigma, (R_{\sigma} x y) \rightarrow (R_{\sigma} x y) \rightarrow (R_{\sigma} x z) \rightarrow (R_{\sigma} x z))
(Tra_{\tau} : \forall x y z : \tau, (R_{\tau} x y) \rightarrow (R_{\tau} x y) \rightarrow (R_{\tau} y z) \rightarrow (R_{\tau} x z))
: \forall (k h : (\sigma \rightarrow \tau)), (\neg (\exists \neg \alpha sigma tau R_{\sigma} sigma R_{\tau} tau h h)) \rightarrow
\( \forall x : \text{sigma}, (R_{\text{sigma}} x x) \rightarrow \text{Neg}(R_{\tau\text{au}} (k x) (h x)) \rightarrow \text{Bot} \rightarrow \text{Neg}(R_{\tau\text{ar}} \text{sigma} \tau\text{au} R_{\text{tau}} k h) \rightarrow \text{Bot} \)

\[ \begin{align*}
&:= \text{fun} \; k \; h \; u \; v \; w \Rightarrow \\
&w \; (\text{fun} \; x \; y \; w1 \; w2 \Rightarrow \\
&\quad v \; x \; (\text{Tra}_{\text{sigma}} x y x \; w1 \; (\text{Sym}_{\text{sigma}} x y \; w1)) \\
&\quad (\text{fun} \; v1 \Rightarrow \\
&\quad \quad u \; (\text{fun} \; u1 \Rightarrow \\
&\quad \quad u1 \; x \; y \; w1 \; (\text{TraNeg} \; \text{tau} \; R_{\text{tau}} \; \text{Tra}_x (k x) (h x) (h y) \; v1 \; w2)))).
\end{align*} \]

**C.4.22 Mating Rule - 1 argument**

Definition matingrule1 \( (\text{sigma} : \text{Type}) (R_{\text{sigma}} : \text{sigma} \rightarrow \text{sigma} \rightarrow o) \)

\[ \forall (f : (\text{sigma} \rightarrow o)) (s \; t : \text{sigma}), \text{Neg}(\text{Neg}(R_{\tau\text{ar}} \text{sigma} \; o \; R_{\text{sigma}} \; R_{\text{sigma}} \; o \; f \; f)) \rightarrow \text{Bot} \rightarrow \text{Neg}(f \; s) \rightarrow \text{Bot} \]

\[ := \text{fun} \; f \; s \; t \; u0 \; u1 \; u2 \; u3 \Rightarrow \]

\[ u1 \; (\text{fun} \; v1 \Rightarrow \\
&\quad u0 \; (\text{fun} \; u00 \Rightarrow u00 \; s \; t \; v1 \; (\text{fun} \; v2 \Rightarrow v2 \; \text{Bot} \; (\text{fun} \; w1 \; w2 \Rightarrow \text{w3} \; (w1 \; u2))).) \]

**C.4.23 Mating Rule - 2 arguments**

Definition matingrule2 \( (\text{sigma tau} : \text{Type}) (R_{\text{sigma}} : \text{sigma} \rightarrow \text{sigma} \rightarrow o) \)

\[ (R_{\tau\text{au}} : \tauau \rightarrow o) \]

\[ \forall (p : (\text{sigma} \rightarrow \tauau \rightarrow o)) (s \; t \; u \; v : \text{sigma}), \text{Neg}(\text{Neg}(R_{\tau\text{ar}} \text{sigma} \; i \; R_{\text{sigma}} \; R_{\text{tau}} \; R_{\text{tau}} \; i \; h \; h)) \rightarrow \text{Bot} \rightarrow \text{Neg}(R_{\tau\text{au}} s \; t) \rightarrow \text{Bot} \rightarrow \text{Neg}(R_{\tau\text{au}} (h \; s) (h \; t)) \rightarrow \text{Bot} \]

\[ := \text{fun} \; p \; s \; t \; u0 \; u1 \; u2 \; u3 \; u4 \Rightarrow \]

\[ u1 \; (\text{fun} \; v1 \Rightarrow \\
&\quad u2 \; (\text{fun} \; v2 \Rightarrow \\
&\quad u0 \; (\text{fun} \; u00 \Rightarrow \\
&\quad u00 \; s \; t \; v1 \; (\text{fun} \; w \Rightarrow \\
&\quad w \; s \; t \; v2 \; (\text{fun} \; z \Rightarrow \\
&\quad z \; \text{Bot} \; (\text{fun} \; z1 \; z2 \Rightarrow \\
&\quad u4 \; (z1 \; u3)))))))). \]

**C.4.24 Decomposition Rule - 1 argument**

Definition decompositionrule1 \( (\text{sigma} : \text{Type}) (R_{\text{sigma}} : \text{sigma} \rightarrow \text{sigma} \rightarrow o) \)

\[ \forall (h : (\text{sigma} \rightarrow i)) (s \; t : \text{sigma}), \text{Neg}(\text{Neg}(R_{\tau\text{ar}} \text{sigma} \; i \; R_{\text{sigma}} \; R_{\text{tau}} \; i \; h \; h)) \rightarrow \text{Bot} \rightarrow \text{Neg}(R_{\tau\text{au}} (h \; s) (h \; t)) \rightarrow \text{Bot} \]

\[ := \text{fun} \; h \; s \; t \; u \; v \; w \Rightarrow u \; (\text{fun} \; u1 \Rightarrow v \; (\text{fun} \; v1 \Rightarrow (u1 \; s \; t \; v1 \; w))). \]

**C.4.25 Decomposition Rule - 2 arguments**

Definition decompositionrule2 \( (\text{sigma tau} : \text{Type}) (R_{\text{sigma}} : \text{sigma} \rightarrow \text{sigma} \rightarrow o) \)

\[ (R_{\tau\text{au}} : \tauau \rightarrow o) \]

\[ \forall (h : (\text{sigma} \rightarrow \tauau \rightarrow i)) (s \; t \; u : \text{sigma}), \text{Neg}(\text{Neg}(R_{\tau\text{ar}} \text{sigma} \; i \; R_{\text{sigma}} \; R_{\tau\text{ar}} \; i \; h \; h)) \rightarrow \text{Bot} \rightarrow \text{Neg}(R_{\tau\text{au}} (h \; s) (h \; t)) \rightarrow \text{Bot} \rightarrow \text{Neg}(R_{\tau\text{au}} (h \; s \; t \; (h \; t \; t1 \; t2)) \rightarrow \text{Bot} \rightarrow \text{Bot} \]

\[ := \text{fun} \; h \; s \; t \; u \; v \; w \Rightarrow u \; (\text{fun} \; u1 \Rightarrow v \; (\text{fun} \; v1 \Rightarrow (u1 \; s \; t \; v1 \; w))). \]
C.5. OTHER LEMMAS

:= 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 Roref1 :
∀ (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 Rirefl :
∀ (x : i), R_i x x
:= fun x p u ⇒ u.

C.5.3 Lemma 8.3.11

Definition negrefl :
R_o o o R_o 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 imprefl :
R_o o (o → o) R_o (R_o o o R_o R_o) Imp Imp
APPENDIX C. GIRARD-KURODA-PER TRANSFORMATION

\[ \text{APPEndix C. GIRARD-KURODA-PER TRANsFORMA\text{TION}} \]

C.5.5 Lemma 8.3.13

Definition andrefl

\[ \begin{align*}
\text{andrefl} & : R_{\text{ar o o}} (R_{\text{o o}} R_{\text{o o}}) \text{ And And} \\
\text{andrefl} & := \begin{cases} 
\text{fun } x1 x2 u1 v1 \Rightarrow \\
\text{v1 (fun y1 y2 u2 v2 \Rightarrow} \\
v2 \text{ (fun p u3 \Rightarrow} \\
u1 p \text{ (fun w1 w2 \Rightarrow} \\
u2 p \text{ (fun w3 w4 \Rightarrow} \\
u3 \text{ (fun w5 w6 \Rightarrow} \\
\text{w3 \text{ (w5 \text{ (w2 w6))})}} \\
\text{w4 \text{ (w5 \text{ (w1 w6))})}} \\
\text{))}) \\
\end{cases} 
\end{align*} \]

C.5.6 Lemma 8.3.14

Definition orrefl

\[ \begin{align*}
\text{orrefl} & : R_{\text{ar o o}} (R_{\text{o o}} R_{\text{o o}}) \text{ Or Or} \\
orrefl & := \begin{cases} 
\text{fun } x1 x2 u1 v1 \Rightarrow \\
v1 \text{ (fun y1 y2 u2 v2 \Rightarrow} \\
v2 \text{ (fun p u3 \Rightarrow} \\
u1 p \text{ (fun w1 w2 \Rightarrow} \\
u2 p \text{ (fun w3 w4 \Rightarrow} \\
u3 \text{ (fun w5 q w6 \Rightarrow} \\
w5 q \text{ (fun w7 w8 \Rightarrow} \\
w6 \text{ (w1 w7) (w3 w8))}} \\
\text{w4 \text{ (w5 \text{ (w2 w7))})}} \\
\text{w7 \text{ (w4 \text{ (w8))})}} \\
\end{cases} 
\end{align*} \]

C.5.7 Lemma 8.3.17

Definition Rrefl (sigma tau : Type) (R_{\text{ar sigma : sigma \rightarrow sigma \rightarrow o}})

\[ \begin{align*}
\text{Rrefl} & (sigma \ tau : \text{Type}) (R_{\text{ar sigma : sigma \rightarrow sigma \rightarrow o}}) \\
\text{(Sym}_{\text{sigma}} & : \forall (x y : sigma),(R_{\text{sigma x y}} \rightarrow (R_{\text{sigma x y}})) \\
\text{(Trans}_{\text{sigma}} & : \forall x y z : \text{sigma}, (R_{\text{sigma x y}} \rightarrow (R_{\text{sigma x z}})) \\
\text{Rrefl} & := \begin{cases} 
\text{fun } x1 x2 u1 v1 \Rightarrow \\
v1 \text{ (fun y1 y2 u2 v2 \Rightarrow} \\
v2 \text{ (fun p u3 \Rightarrow} 
\end{cases} 
\end{align*} \]
C.5. OTHER LEMMAS

C.5.8 Lemma 8.3.15

Definition existsrefl (sigma : Type) (R_sigma : sigma \to sigma \to o)
: R_ar (sigma \to 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 \Rightarrow
  u2 (fun p u3 \Rightarrow
    u3 (fun v1 q v2 \Rightarrow
      v1 q (fun x v3 v4 \Rightarrow
        u1 x x v3 (fun v5 \Rightarrow
          v5 Bot (fun v6 v7 \Rightarrow
            u2 (fun r v8 \Rightarrow
              v8 (fun v9 r1 r10 \Rightarrow
                v10 x v3 (v6 v4))
              (fun v9 \Rightarrow v1)))
            (fun v9 \Rightarrow v1)))))
  (fun v1 q v2 \Rightarrow
    v1 q (fun x v3 v4 \Rightarrow
      u1 x x v3 (fun v5 \Rightarrow
        v5 Bot (fun v6 v7 \Rightarrow
          u2 (fun r v8 \Rightarrow
            v8 (fun v9 r1 v10 \Rightarrow
              v10 x v3 (v7 v4)))
            (fun v9 \Rightarrow v1))))).

C.5.9 Lemma 8.3.16

Definition forallrefl (sigma : Type) (R_sigma : sigma \to sigma \to o)
: R_ar (sigma \to 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 \Rightarrow
  u2 (fun p u3 \Rightarrow
    u3 (fun v1 x v2 v3 \Rightarrow
      v1 x v2 (fun v4 \Rightarrow
        u1 x x v2 (fun v5 \Rightarrow
          v5 Bot (fun v6 v7 \Rightarrow v3 (v6 v4))))
      (fun v1 x v2 v3 \Rightarrow
        v1 x v2 (fun v4 \Rightarrow
          u1 x x v2 (fun v5 \Rightarrow
            v5 Bot (fun v6 v7 \Rightarrow v3 (v7 v4))))))).
**C.5.10 Lemma 8.3.19**

Definition \( \text{apppra} (\sigma \tau : \text{Type}) (R_{\sigma} : \sigma \rightarrow \sigma \rightarrow \sigma) (R_{\tau} : \tau \rightarrow \tau \rightarrow \tau) \) : \( \forall (f g : (\sigma \rightarrow \tau)) (x y : \sigma), \) 
\( \text{Neg}(\text{Neg}(R_{\sigma} \sigma \tau R_{\sigma} f g)) \rightarrow \text{Neg}(\text{Neg}(R_{\sigma} \sigma x y)) \rightarrow \text{Neg}(\text{Neg}(R_{\tau} f x (g y)))) \) := fun f g x y u v w ⇒ v (fun v1 ⇒ u (fun u1 ⇒ u1 x y v1 w)).