From Classical Extensional Higher-Order Tableau to Intuitionistic Intentional Natural Deduction

Chad E. Brown[1]
and Christine Rizkallah[2]

[1] Saarland University, Saarbrücken, Germany
cebrown@ps.uni-saarland.de
[2] Max-Planck-Institut für Informatik, Saarbrücken, Germany
crizkall@mpi-inf.mpg.de

Abstract

We define a translation that maps higher-order formulas provable in a classical extensional setting to semantically equivalent formulas provable in an intuitionistic intensional setting. For the classical extensional higher-order proof system we define a Henkin-complete tableau calculus. For the intuitionistic intensional higher-order proof system we give a natural deduction calculus. We prove that tableau provability of a formula implies provability of a translated formula in the natural deduction calculus. Implicit in this proof is a method for translating classical extensional tableau refutations into intuitionistic intensional natural deduction proofs.

1 Introduction

We describe a way of translating a simply-typed higher-order formula s into a semantically equivalent formula s' such that if s is provable in a classical extensional higher-order tableau calculus, then s' is provable in an intuitionistic intentional higher-order natural deduction calculus. A potential application of such a translation is mapping refutations found by a classical extensional tableau-based automated theorem prover like Satallax [4] to corresponding proof terms in an intuitionistic intensional natural deduction based system like Coq [15, 3].

The problem of translating classical logic into intuitionistic logic has a long history. A result by Glivenko [11] states that if s is a propositional tautology, then $\neg\neg s$ is intuitionistically provable. This result does not hold for first-order formulas. Kuroda [14] showed a similar result if one translates by double negating the formula and adding double negations to the bodies of universal quantifiers. We recently proved that the Kuroda translation generalizes to give a translation from classical intentional higher-order logic to intuitionistic intentional higher order logic [5]. The generalized Kuroda translation does not suffice in the presence of functional extensionality, for an example see [5]. The work of Gandy [9] provides a method to translate from a higher-order logic with extensionality into a higher-order logic without extensionality. The translation we give in this paper uses ideas similar to those of Kuroda and Gandy to deal with both issues at once. The Gandy translation translates equality with the help of a binary relation and a predicate that are defined by mutual recursion. Our translation is simpler in that it translates equality with the help of a single binary relation that is defined inductively on types.

Many logical systems of quite different character are commonly referred to as *higher-order logics*. Some forms of higher-order logic allow classical reasoning while others only allow intuitionistic reasoning. For example, formulas such as $\forall p.p \vee \neg p$ and $\forall p.\neg\neg p \to p$ are provable if and only if the higher-order logic is classical. Likewise, some forms of higher-order logic allow extensional reasoning while others do not. Formulas such as $\forall fg.(\forall x.fx = gx) \to f = g$

and $\forall pq.(p \to q) \wedge (q \to p) \to p = q$ are provable if the higher-order logic is extensional and are not provable otherwise. The formula $(\lambda x.\neg\neg x) = (\lambda x.x)$ is only provable if the logic is both classical and extensional. The extensionality properties can be factored into propositional extensionality (i.e., boolean extensionality) and functional extensionality (which can itself be factored into extensionality properties η and ξ) [1]. In this paper we consider a tableau system with full extensionality and a natural deduction system with no extensionality.

Many automated and interactive theorem provers (e.g., Isabelle/HOL [16], LEO-II [2] and Satallax [4]) are based on classical extensional versions of higher-order logic. Automated and interactive theorem provers are used to create proofs, while the task of a proof checker is checking the correctness of proofs. In order to check a proof, the proof must be represented as an explicit object. A well-known way of representing proofs is via the Curry-Howard-de Bruijn correspondence [13, 8]: a natural deduction proof of P can be encoded as a λ-term of type P. It is easy to write a natural deduction system for higher-order logic for which one can obtain proof terms in an obvious way. However, this creates a natural deduction system which is intuitionistic (not classical) and intentional (not extensional).

One way to resolve this mismatch is to add classical extensional axioms to the natural deduction system. Currently, Satallax produces Coq proof terms under the assumption that one has added such axioms to Coq [20]. Using the ideas in this paper, one can avoid assuming such axioms in Coq and still produce Coq proof terms from Satallax refutations (unless Satallax makes use of a choice operator).

The basic definitions and lemmas used in the translation have all been formalized and proven in Coq, as described in [18, 19]. However, there is currently no implementation of the translation as a whole. In particular, Satallax has not yet been extended to produce Coq proof terms using these definitions and lemmas.

Since the translation changes the formula, a Coq user could not typically call a classical extensional theorem prover like Satallax to request a proof term for an arbitrary formula in Coq. On the other hand, one could use the translation to automatically create a Coq development from a development in a classical extensional simple type theory.

The rest of the paper is organized as follows. In Section 2 we give a brief overview of simply typed λ-calculus and in Section 3 we introduce a tableau calculus $\mathcal{T}$. We then present the targeted natural deduction calculus $\mathcal{N}$ in Section 4. In Section 5 we present our translation. We prove that it maps $\mathcal{T}$-refutable branches to $\mathcal{N}$-refutable contexts in Section 6. We conclude in Section 7 and provide suggestions for future work.

More details about the translation in this paper and other translations can be found in the Master's thesis of one of the authors [18].

2 Simply Typed Lambda Calculus

2.1 Syntax

We describe simply typed λ-calculus in the style of Church [7]. The set of *types* $\mathbb{T}$ is given inductively: o and ι are types and $\sigma\tau$ is a type whenever σ and τ are types. The types o (the type of propositions) and ι (the type of individuals) are called *base types*. Types of the form $\sigma\tau$ are called *function types*. Often the function type $\sigma\tau$ is written as $\sigma \to \tau$, but we use the shorter notation since there are only base types and function types. We use σ and τ to range over types.

For each type σ, let $\mathcal{N}_\sigma$ be an infinite set of *names* of type σ. Some of the names are *logical constants*:

- $\top$ and $\bot$ are (distinct) logical constants in $\mathcal{N}_o$.
- $\wedge$, $\vee$, and $\to$ are (distinct) logical constants in $\mathcal{N}_{ooo}$.
- For each σ, $=^\sigma$ is a logical constant in $\mathcal{N}_{\sigma\sigma o}$.
- For each σ, $\forall^\sigma$ and $\exists^\sigma$ are (distinct) logical constants in $\mathcal{N}_{(\sigma o)o}$.

The remaining names are *variables* . Let $\mathcal{C}_\sigma$ be the set of logical constants of type σ and $\mathcal{V}_\sigma$ be the (infinite) set of all variables of type σ. Let $\mathcal{N} = \bigcup_\sigma \mathcal{N}_\sigma$, $\mathcal{C} = \bigcup_\sigma \mathcal{C}_\sigma$ and $\mathcal{V} = \bigcup_\sigma \mathcal{V}_\sigma$.

For any $\mathcal{C}' \subseteq \mathcal{C}$ we define a family of sets of terms $\Lambda_\sigma^{\mathcal{C}'}$ for each type σ by induction.

- For every $x \in \mathcal{V}_\sigma$, $x \in \Lambda_\sigma^{\mathcal{C}'}$.
- For every $c \in \mathcal{C}_\sigma \cap \mathcal{C}'$, $c \in \Lambda_\sigma^{\mathcal{C}'}$.
- For every $x \in \mathcal{V}_\sigma$ and $s \in \Lambda_\tau^{\mathcal{C}'}$, $(\lambda x.s) \in \Lambda_{\sigma\tau}^{\mathcal{C}'}$.
- For every $s \in \Lambda_{\sigma\tau}^{\mathcal{C}'}$ and $t \in \Lambda_\sigma^{\mathcal{C}'}$, $(st) \in \Lambda_\tau^{\mathcal{C}'}$.

We defined $\Lambda_\sigma^{\mathcal{C}'}$ relative to a set of logical constants. There are two particular sets of logical constants of interest in this paper: the full set $\mathcal{C}$ of logical constants and the set

$$\mathcal{C}^- := \{\to, \forall^\sigma | \sigma \text{ is a type}\}.$$

To simplify notation, we define Λ_σ to be $\Lambda_\sigma^{\mathcal{C}}$ and Λ_σ^- to be $\Lambda_\sigma^{\mathcal{C}^-}$. Note that $\Lambda_\sigma^{\mathcal{C}'} \subseteq \Lambda_\sigma$ for any $\mathcal{C}' \subseteq \mathcal{C}$. An element of Λ_σ is called a *term of type* σ. A *term* is an element of $\bigcup_\sigma \Lambda_\sigma$. Terms of the form $(\lambda x.s)$ are called λ*-abstractions*. Terms of the form (st) are called *applications*. A *formula* is a term of type o.

We write $\neg s$ for $((\to s)\bot)$. We write stu for $(st)u$, except that $\neg st$ means $\neg(st)$. We use the infix notations $s \wedge t$, $s \vee t$, $s \to t$ and $s =^\sigma t$ as shorthand for $\wedge st$, $\vee st$, $\to st$ and $=^\sigma st$, respectively. We also write $s \neq^\sigma t$ for $\neg(s =^\sigma t)$. Using infix notation note that $\neg s$ is the same term as $s \to \bot$. We sometimes write $\forall x : \sigma.s$ and $\exists x : \sigma.s$ for $\forall^\sigma(\lambda x.s)$ and $\exists^\sigma(\lambda x.s)$, respectively. We may also omit the type entirely from a quantified formula, equation or disequation and write $\forall x.s$, $\exists x.s$, $s = t$ or $s \neq t$ when the types are clear from the context.

If s is a term, $x \in \mathcal{N}_\sigma$ and t is a term of type σ, then $s[x := t]$ is defined to be the result of substituting t for x in s via a capture-avoiding substitution. A *simultaneous substitution* θ substitutes several variables simultaneously. We use the notation $\theta, [x := t]$ to mean the simultaneous substitution that agrees with θ on all variables except (possibly) x which is mapped to t. A term of the form $(\lambda x.s)t$ is called a β*-redex* with β*-reduct* $s[x := t]$. We say s β*-reduces to* t (and write $s \to_\beta t$) if a subterm of s is a β-redex such that t is the result of replacing this subterm by its β-reduct. We define $s \sim_\beta t$ to be the least equivalence relation containing $\to_\beta$. When $s \sim_\beta t$ holds, we say s and t are β*-equivalent*. A term is β*-normal* if it has no β-redexes. It is well-known that β-reduction is confluent and terminates on simply typed terms. Hence for every $s \in \Lambda_\sigma$ there is a β-normal form of s which is unique (up to names of bound variables). We write $\lceil s \rceil^\beta$ to denote the β-normal form of s.

The set of *free variables* of a term, written FV, is defined as follows.

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

A term s is *ground* if $FV(s) = \emptyset$. The set of *free variables* of a set of terms B is defined as $FV(B) := \bigcup_{s \in B} FV(s)$.

2.2 Semantics

Henkin proved completeness of a form of Church's simple theory of types [7] relative to a semantics now known as Henkin semantics [12]. We briefly describe Henkin semantics. The interested reader may find more details of a similar presentation in [6].

A *frame* is a function $\mathcal{D}$ defined on $\mathbb{T}$ such that $\mathcal{D}(o) = \{0,1\}$, $\forall \sigma \in \mathbb{T} :\ \mathcal{D}(\sigma) \neq \emptyset$ and $\forall \sigma,\tau \in \mathbb{T} :\ \mathcal{D}(\sigma\tau) \subseteq \{f \mid f : \mathcal{D}(\sigma) \to \mathcal{D}(\tau)\}$. An *assignment into a frame* $\mathcal{D}$ is a function $\mathcal{I}$ defined on $\mathbb{T} \cup \mathcal{V}$ such that $\mathcal{I}(x) \in \mathcal{D}(\sigma)$ for all types σ and variables $x : \sigma$. Let $\mathcal{I}$ be an assignment into a frame $\mathcal{D}$, $x : \sigma$ be a variable and $a \in \mathcal{D}(\sigma)$. We write $\mathcal{I}^x_a$ to denote the assignment into $\mathcal{D}$ that agrees everywhere with $\mathcal{I}$ except possibly on x where it yields a. We define a partial evaluation function $\widehat{\ }$ that maps assignments $\mathcal{I}$ and terms $s \in \Lambda_\sigma$ possibly to values $\hat{\mathcal{I}}(s)$ in $\mathcal{D}(\sigma)$ as follows:

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

Note that $\hat{\mathcal{I}}$ is a partial function from typed terms into the frame. An *interpretation* is a pair $(\mathcal{D},\mathcal{I})$ where $\mathcal{D}$ is a frame, $\mathcal{I}$ is an assignment into $\mathcal{D}$ and $\hat{\mathcal{I}}$ is a total function, i.e., $\hat{\mathcal{I}}s$ is defined for every $s \in \Lambda_\sigma$. We write $Interp$ for the set of all interpretations.

A *formula* is a term of type o. We say an interpretation $(\mathcal{D},\mathcal{I})$ *satisfies* a formula s if $\hat{\mathcal{I}}(s) = 1$. A set A of formulas is *satisfiable* if there is an interpretation $(\mathcal{D},\mathcal{I})$ simultaneously satisfying all the formulas in A. Otherwise, we say A is *unsatisfiable*. We say two terms $s,t \in \Lambda_\sigma$ are *semantically equivalent* (written $s \approx t$) if $\hat{\mathcal{I}}s = \hat{\mathcal{I}}t$ for all interpretations $(\mathcal{D},\mathcal{I})$.

3 Tableau Calculus $\mathcal{T}$

We briefly describe a tableau calculus for classical extensional higher-order logic which is complete relative to Henkin semantics. A similar tableau calculus is presented and proven complete in [6]. The calculus in [6] only uses the logical constants $=_\sigma$ and $\neg$ while here we include many more logical constants. Also, we include rules such as Cut and DeMorgan which are not needed for completeness. We include these rules because they are sound relative to Henkin semantics and the translation we give later is able to handle the extra rules.

A *branch* is a set of β-normal formulas. A *step* is an $n+1$-tuple $\langle A_1,\ldots,A_n,A\rangle$ of branches with $n \geq 0$. Given a set of steps T, one can inductively define the set of branches which are *T-refutable* as follows: If $\langle A_1,\ldots,A_n,A\rangle \in T$ and A_i is T-refutable for $i \in \{1,\ldots,n\}$, then A is T-refutable.

A *rule* is a set of steps. The rules are presented in the form

$$RuleName\ \frac{C}{B_1 \mid \cdots \mid B_n}$$

to indicate the set of steps of the form $\langle A_1,\ldots,A_n,A\rangle$ where $C \subseteq A$ and $A_i = A \cup B_i$ for each $i \in \{1,\ldots,n\}$. There are also sometimes side conditions on the branch A. For example if we say a variable y must be *fresh* in a rule, this means that for the step $\langle A_1,\ldots,A_n,A\rangle$ to be in the rule there is the additional requirement that $y \notin FV(A)$. In most cases C is a singleton set $\{s\}$ and in the remaining cases C is either empty (e.g, Cut) or contains two formulas (e.g., Mat).

Definition 3.1 (Tableau Calculus $\mathcal{T}$)**.** We define the *tableau calculus* $\mathcal{T}$ as the union of the rules in Figure 1. This also defines the corresponding notion of $\mathcal{T}$-refutability. We say a formula s is $\mathcal{T}$*-refutable* if the branch $\{\lceil s \rceil^\beta\}$ is $\mathcal{T}$-refutable. We say a formula s is $\mathcal{T}$*-provable* if the formula $\neg\lceil s \rceil^\beta$ is $\mathcal{T}$-refutable.

$$\textsf{Closed}\bot\ \frac{\bot}{}\qquad \textsf{Closed}\neg\top\ \frac{\neg\top}{}\qquad \textsf{Closed}\ \frac{s,\ \neg s}{}\qquad \textsf{Closed}\neq\ \frac{s \neq s}{}$$

$$\textsf{ClosedSym}\ \frac{(s = t),\ (t \neq s)}{}\qquad \textsf{Cut}\ \frac{}{s \mid \neg s}\qquad \textsf{Dneg}\ \frac{\neg\neg s}{s}\qquad \textsf{And}\ \frac{s \wedge t}{s, t}\qquad \textsf{Or}\ \frac{s \vee t}{s \mid t}$$

$$\textsf{Imp}\ \frac{s \to t}{\neg s \mid t}\qquad \textsf{NegAnd}\ \frac{\neg(s \wedge t)}{\neg s \mid \neg t}\qquad \textsf{NegOr}\ \frac{\neg(s \vee t)}{\neg s, \neg t}\qquad \textsf{NegImp}\ \frac{\neg(s \to t)}{s, \neg t}$$

$$\textsf{Forall}\ \frac{\forall s}{\lceil s\ t \rceil^\beta}\qquad \textsf{DeMorgan}\forall\ \frac{\neg\forall s}{\exists x.\neg\lceil s\ x \rceil^\beta}\ x \notin FV(s)\qquad \textsf{Exists}\ \frac{\exists s}{\lceil s\ y \rceil^\beta}\ y \text{ is fresh}$$

$$\textsf{DeMorgan}\exists\ \frac{\neg\exists s}{\forall x.\neg\lceil s\ x \rceil^\beta}\ x \notin FV(s)\qquad \textsf{Bool} = \frac{s =^o t}{s, t \mid \neg s, \neg t}\qquad \textsf{BoolExt}\ \frac{s \neq^o t}{s, \neg t \mid t, \neg s}$$

$$\textsf{Func} = \frac{s_1 =^{\sigma\tau} s_2}{\lceil s_1\ t =^\tau s_2\ t \rceil^\beta}\qquad \textsf{FuncExt}\ \frac{s \neq^{\sigma\tau} t}{\lceil s\ x \neq^\tau t\ x \rceil^\beta}\ x \text{ is fresh}\qquad \textsf{Mat}\ \frac{x\ s_1 \ldots s_n, \neg x\ t_1 \ldots t_n}{s_1 \neq t_1 \mid \ldots \mid s_n \neq t_n}$$

$$\textsf{Dec}\ \frac{x\ s_1 \ldots s_n \neq^\iota x\ t_1 \ldots t_n}{s_1 \neq t_1 \mid \ldots \mid s_n \neq t_n}\qquad \textsf{Con}\ \frac{s_1 =^\iota t_1,\ s_2 \neq^\iota t_2}{s_1 \neq s_2,\ t_1 \neq s_2 \mid s_1 \neq t_2,\ t_1 \neq t_2}$$

Figure 1: Tableau rules used to define the tableau calculus $\mathcal{T}$

Many variations of the tableau rules are possible. For example, instead of the rule

$$\textsf{DeMorgan}\exists\ \frac{\neg\exists s}{\forall x.\neg\lceil s\ x \rceil^\beta}\ x \notin FV(s)$$

we could have an alternative rule

$$\textsf{Neg}\exists\ \frac{\neg\exists s}{\neg\lceil s\ x \rceil^\beta}\ x \text{ is fresh.}$$

Note that if we use Neg$\exists$, then x must be fresh rather than the weaker requirement $x \notin FV(s)$ in DeMorgan$\exists$. By using the DeMorgan$\exists$ version, there is one fewer rule for which the freshness condition needs to be taken care of later. One could similarly modify the rule FuncExt so that the only rule with a freshness condition would be Exists. In this paper the form of the tableau rules are chosen to match those considered in [18].

We briefly consider two examples of $\mathcal{T}$-provable formulas.

Example 3.2. Let p be a variable of type o. We show the formula $p \neq \neg p$ is $\mathcal{T}$-provable, i.e., the branch $A_0 := \{p \neq \neg p\}$ is $\mathcal{T}$-refutable. The branch A_0 is $\mathcal{T}$-refutable because of the NegOr rule and the fact that $A_1 := A_0 \cup \{p, \neg\neg\neg p\}$ and $A_2 := A_0 \cup \{\neg p, \neg\neg p\}$ are $\mathcal{T}$-refutable. We know A_2 is $\mathcal{T}$-refutable using the Closed rule. We know A_1 is $\mathcal{T}$-refutable using the Dneg and

Closed rules. The reason we call this a *tableau refutation* is that one can display the refutation as a picture we refer to as a *tableau*. For this example the following is the corresponding tableau:

$$\begin{array}{c|c} p \neq^o \neg p & \\ p & \neg p \\ \neg\neg\neg p & \neg\neg p \\ \neg p & \end{array}$$

Example 3.3. Let p be a variable of type o. The formula $(\lambda p.\neg\neg p) =^{oo} (\lambda p.p)$ is $\mathcal{T}$-provable. In this case we simply show the tableau and note that the steps are justified by the FuncExt, BoolExt, Dneg and Closed rules.

$$\begin{array}{c|c} (\lambda p.\neg\neg p) \neq (\lambda p.p) & \\ (\neg\neg p) \neq p & \\ \neg\neg p & \neg\neg\neg p \\ \neg p & p \\ & \neg p \end{array}$$

4 Natural Deduction Calculus $\mathcal{N}$

We now present a natural deduction calculus for formulas in Λ_o^-. That is, we only consider formulas that use the logical constants $\to$ and $\forall^\sigma$. Such calculi were introduced by Gentzen in 1935 [10] and studied further by Prawitz [17].

A *context* Γ is a finite subset of Λ_o^-. $\Gamma \vdash_{\mathcal{N}} s$ holds when derivable using the rules in Figure 2.

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

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

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

$$\forall E\ \frac{\Gamma \vdash_{\mathcal{N}} \forall x.s}{\Gamma \vdash_{\mathcal{N}} s[x := t]} \qquad \to E\ \frac{\Gamma \vdash_{\mathcal{N}} s \to t \quad \Gamma \vdash_{\mathcal{N}} s}{\Gamma \vdash_{\mathcal{N}} t}$$

Figure 2: Rules in our ND calculus $\mathcal{N}$

We write $\vdash_{\mathcal{N}} s$ for $\emptyset \vdash_{\mathcal{N}} s$. We say a formula $s \in \Lambda_o^-$ is $\mathcal{N}$-*provable* if $\vdash_{\mathcal{N}} s$. We say a context Γ is $\mathcal{N}$-*refutable* if $\Gamma \vdash_{\mathcal{N}} \forall^o p.p$. Likewise, a formula $s \in \Lambda_o^-$ is $\mathcal{N}$-*refutable* if the context $\{s\}$ is $\mathcal{N}$-refutable.

Example 4.1. Let x, y, p and q be variables with $x, y \in \mathcal{V}_\iota$ and $p, q \in \mathcal{V}_{\iota o}$. We use the following diagram to show that the formula $(\forall p.px \to py) \to qy \to qx$ is $\mathcal{N}$-provable.

$$\to I\ \frac{\to E\ \dfrac{\beta\ \dfrac{\forall E\ \dfrac{hy\ \dfrac{}{\{\forall p.px \to py\} \vdash_{\mathcal{N}} \forall p.px \to py}}{\{\forall p.px \to py\} \vdash_{\mathcal{N}} (\lambda y.qy \to qx)x \to (\lambda y.qy \to qx)y}}{\{\forall p.px \to py\} \vdash_{\mathcal{N}} (qx \to qx) \to (qy \to qx)} \qquad wk\ \dfrac{\to I\ \dfrac{hy\ \dfrac{}{\{qx\} \vdash_{\mathcal{N}} qx}}{\vdash_{\mathcal{N}} qx \to qx}}{\{\forall p.px \to py\} \vdash_{\mathcal{N}} qx \to qx}}{\{\forall p.px \to py\} \vdash_{\mathcal{N}} qy \to qx}}{\vdash_{\mathcal{N}} (\forall p.px \to py) \to qy \to qx}$$

5 Translating Terms, Formulas and Branches

In this section we introduce a meaning preserving translation that maps tableau refutable formulas in Λ_o to $\mathcal{N}$-refutable formulas in Λ^-_o. Since a tableau calculus operates on branches instead of formulas, we will also need to define a branch translation Ψ^* that maps branches to contexts.

We begin by considering the most important issue: the translation of logical constants. We will associate with each logical constant $c \in \mathcal{C}_\sigma$ a term $\dot{c} \in \Lambda^-_\sigma$. For the propositional connectives we can use terms which are sometimes called the Russell-Prawitz definitions [17].

$$\begin{array}{lcl}
\dot{\rightarrow} & := & \lambda x.\lambda y.x \rightarrow y \\
\dot{\bot} & := & \forall p : o.p \\
\dot{\top} & := & \forall p : o.p \rightarrow p \\
\dot{\neg} & := & \lambda x.x \rightarrow \forall p : o.p \\
\dot{\wedge} & := & \lambda x.\lambda y.\forall p : o.(x \rightarrow y \rightarrow p) \rightarrow p \\
\dot{\vee} & := & \lambda x.\lambda y.\forall p : o.(x \rightarrow p) \rightarrow (y \rightarrow p) \rightarrow p
\end{array}$$

We also define $\doteq$ to be $\lambda xy : o.(x \rightarrow y)\dot{\wedge}(y \rightarrow x)$, even though we do not have a logical constant $\equiv$, since it will be useful below.

We will define $\doteq^\sigma$ to be a term $\mathbf{R}^\sigma$ of type $\sigma\sigma o$. This term $\mathbf{R}^\sigma$ will also be used in the definitions of $\dot{\forall}^\sigma$ and $\dot{\exists}^\sigma$. Since $\doteq^\sigma$ will be defined as $\mathbf{R}^\sigma$, the meaning of $\mathbf{R}^\sigma$ in every (classical extensional) interpretation must be equality. A simple way to satisfy this constraint is to define $\mathbf{R}^\sigma$ to be Leibniz equality, i.e., $\lambda xy.\forall q : \sigma o.qx \rightarrow qy$, at each type σ. For each $\mathcal{T}$-provable formula the translation should be $\mathcal{N}$-provable. If $\mathbf{R}^{\iota\iota}$ and $\mathbf{R}^\iota$ were both Leibniz equality, then even though $\forall fg : \iota\iota.(\forall x : \iota.fx = gx) \rightarrow f = g$ is clearly $\mathcal{T}$-provable its translation would not be $\mathcal{N}$-provable.

This suggests that we should not define $\mathbf{R}^\sigma$ to be Leibniz equality for every type σ. Instead we will define $\mathbf{R}^\sigma$ to be Leibniz equality only when σ is the base type ι. We will define $\mathbf{R}^o$ to be logical equivalence (as expressed by $\doteq$) and on function types we will use functional equivalence modified by a double negation.

Definition 5.1. For every type σ we define inductively a term $\mathbf{R}^\sigma$ in $\Lambda^-_{\sigma\sigma o}$ as follows:

$$\begin{array}{l}
\mathbf{R}^o = \lambda x\, y.(x \rightarrow y)\dot{\wedge}(y \rightarrow x) \\
\mathbf{R}^\iota = \lambda x\, y.\forall q : \iota o.q\, x \rightarrow q\, y \\
\mathbf{R}^{\sigma\rightarrow\tau} = \lambda f\, g.\forall x\, y : \sigma.\mathbf{R}^\sigma\, x\, y \rightarrow \dot{\neg}\dot{\neg}\mathbf{R}^\tau(f\, x)(g\, y)
\end{array}$$

The term $\mathbf{R}^\sigma$ corresponds to a binary relation on type σ. We sometimes speak of $\mathbf{R}^\sigma$ as a relation rather than as a term.

It will turn out that we cannot generally prove (in $\mathcal{N}$) that $\mathbf{R}^\sigma$ is reflexive. As a consequence, we must restrict the binders in the definitions of $\dot{\forall}^\sigma$ and $\dot{\exists}^\sigma$ to x satisfying $\mathbf{R}^\sigma\ x\ x$. In terms of Henkin semantics, this will not make a difference since we will prove that $\hat{\mathcal{I}}(\mathbf{R}^\sigma)$ is equality on $\mathcal{D}(\sigma)$ in every interpretation $(\mathcal{D}, \mathcal{I})$. When considering provability of formulas in $\mathcal{N}$, the restriction to x satisfying $\mathbf{R}^\sigma\ x\ x$ will be important.

Now we define $\doteq^\sigma$, $\dot{\forall}^\sigma$, and $\dot{\exists}^\sigma$ as follows:

$$\begin{array}{lcl}
\doteq^\sigma & := & \mathbf{R}^\sigma \\
\dot{\forall}^\sigma & := & \lambda f.\forall^\sigma x.\mathbf{R}^\sigma xx \rightarrow \dot{\neg}\dot{\neg}fx \\
\dot{\exists}^\sigma & := & \lambda f.\forall^o p.(\forall^\sigma x.\mathbf{R}^\sigma xx \rightarrow fx \rightarrow p) \rightarrow p
\end{array}$$

Definition 5.2. We define our translation $\Psi : \Lambda_\sigma \to \Lambda^-_\sigma$ by recursion as follows.

$$\begin{array}{rcll} \Psi x & := & x & \text{for variables } x \\ \Psi st & := & (\Psi s)(\Psi t) & \\ \Psi \lambda x.s & := & \lambda x.\Psi s & \\ \Psi c & := & \dot{c} & \text{for constants } c \end{array}$$

We call Ψ *compositional* because it respects application and λ-abstraction. As a simple consequence of compositionality, we know that Ψ preserves β-equivalence.

Lemma 5.3. If s and t are β-equivalent, then Ψs and Ψt are also β-equivalent.

We now prove $\Psi s \approx s$ for all $s \in \Lambda_\sigma$. We first prove that $\mathbf{R}^\sigma$ behaves like equality.

Lemma 5.4. $\forall \sigma \in \mathbb{T} : \forall (\mathcal{D}, \mathcal{I}) \in Interp : \forall a, b \in \mathcal{D}(\sigma) : (\hat{\mathcal{I}}(\mathbf{R}^\sigma)\, a\, b = 1) \iff a = b$

Proof. We prove this lemma by induction on types. Let $(\mathcal{D}, \mathcal{I})$ be an arbitrary interpretation and let a and b be arbitrary elements in $\mathcal{D}(\sigma)$.

- Case $\sigma = o$:
  It is easy to check that $\widehat{\mathcal{I}^{xy}_{ab}}((x \to y)\dot{\wedge}(y \to x)) = 1 \iff a = b$.
- Case $\sigma = \iota$:
  We know $\hat{\mathcal{I}}(\mathbf{R}^\iota)\, a\, b = 1 \iff a = b$ since $\mathbf{R}^\iota$ is Leibniz equality.
- Case $\sigma = \sigma_1\sigma_2$:
  We want to show $\hat{\mathcal{I}}(\mathbf{R}^{\sigma_1\sigma_2})\, a\, b = 1 \iff a = b$.
  - Assume $\hat{\mathcal{I}}(\mathbf{R}^{\sigma_1\sigma_2})\, a\, b = 1$. We need to show $a = b$. Let $c \in \mathcal{D}(\sigma_1)$ be given. We prove $a(c) = b(c)$ as follows.

$$\begin{array}{ll} & \hat{\mathcal{I}}(\mathbf{R}^{\sigma_1\sigma_2})\, a\, b = 1 \\ \iff & \widehat{\mathcal{I}^{fg}_{ab}}(\forall x\, y.\mathbf{R}^{\sigma_1}\, x\, y \to \dot{\neg}\dot{\neg}\mathbf{R}^{\sigma_2}(f\, x)(g\, y)) = 1 \\ \implies & \widehat{\mathcal{I}^{fgxy}_{abcc}}(\mathbf{R}^{\sigma_1}\, x\, y \to \dot{\neg}\dot{\neg}\mathbf{R}^{\sigma_2}(f\, x)(g\, y)) = 1 \\ \iff & (\hat{\mathcal{I}}(\mathbf{R}^{\sigma_1})\, c\, c = 1 \implies \hat{\mathcal{I}}(\mathbf{R}^{\sigma_2})(a(c))(b(c)) = 1) \\ \iff & (c = c \implies a(c) = b(c)) \qquad \text{(IH)} \\ \iff & a(c) = b(c) \end{array}$$

  - Assume $a = b$. Let $c, d \in \mathcal{D}(\sigma_1)$ be such that $\hat{\mathcal{I}}(\mathbf{R}^{\sigma_1})\, c\, d = 1$. We know $c = d$ by the inductive hypothesis and so $ac = bd$. By the inductive hypothesis we have $\hat{\mathcal{I}}(\mathbf{R}^{\sigma_2})\, (ac)\, (bd) = 1$. Hence $\hat{\mathcal{I}}(\mathbf{R}^{\sigma_1\sigma_2})\, a\, b = 1$.

□

Lemma 5.5. For every interpretation $(\mathcal{D}, \mathcal{I})$ and every a in $\mathcal{D}(\sigma)$ we have $\mathcal{I}(\mathbf{R}^\sigma)\, a\, a = 1$.

Proof. Follows directly from Lemma 5.4. □

Lemma 5.6. For every $c \in \mathcal{C}_\sigma$, $\dot{c} \approx c$.

Proof. Let $(\mathcal{D},\mathcal{I})$ be an interpretation. In order to prove $\hat{\mathcal{I}}(\dot{c}) = \hat{\mathcal{I}}(c)$ it is enough to prove $\hat{\mathcal{I}}(\dot{c})$ has the usual classical meaning of c in $\mathcal{I}$ since at most one element of $\mathcal{D}(\sigma)$ can have this property. It is easy to check this for $\hat{\mathcal{I}}(\dot{c})$ when $c \in \{\to, \bot, \top, \neg, \wedge, \vee\}$. We know $\hat{\mathcal{I}}(\dot{=}^\tau)$ behaves like equality by Lemma 5.4. It remains to prove $\hat{\mathcal{I}}(\dot{\forall}^\tau)$ and $\hat{\mathcal{I}}(\dot{\exists}^\tau)$ behave like universal and existential quantification. Let $q \in \mathcal{D}(\tau o)$ be given. We use Lemma 5.4 to obtain

$$\begin{aligned}
&\hat{\mathcal{I}}(\dot{\forall}^\tau)\, q = 1\\
\iff &\widehat{\mathcal{I}^f_q}(\forall^\tau x.(\mathbf{R}^\tau xx) \to \dot{\neg}\dot{\neg} fx) = 1\\
\iff &\widehat{\mathcal{I}^f_q}(\forall^\tau x.fx) = 1\\
\iff &qa = 1 \text{ for every } a \in \mathcal{D}(\tau)
\end{aligned}$$

and

$$\begin{aligned}
&\hat{\mathcal{I}}(\dot{\exists}^\tau)\, q = 1\\
\iff &\widehat{\mathcal{I}^f_q}(\forall^o p.(\forall^\sigma x.(\mathbf{R}^\sigma xx) \to fx \to p) \to p) = 1\\
\iff &\widehat{\mathcal{I}^f_q}(\forall^o p.(\forall^\sigma x.fx \to p) \to p) = 1\\
\iff &qa = 1 \text{ for some } a \in \mathcal{D}(\tau).
\end{aligned}$$

□

Theorem 5.7. For every $s \in \Lambda_\sigma$, $\Psi s \approx s$.

Proof. We argue by induction on s. If s is a variable, then the result is clear. If s is a logical constant, then the result follows by Lemma 5.6. Suppose s is tu. Let $(\mathcal{D},\mathcal{I})$ be an interpretation. By inductive hypothesis $\hat{\mathcal{I}}(\Psi t) = \hat{\mathcal{I}}(t)$ and $\hat{\mathcal{I}}(\Psi u) = \hat{\mathcal{I}}(u)$. Hence $\hat{\mathcal{I}}(\Psi(tu)) = \hat{\mathcal{I}}(tu)$.

Finally, suppose s is $\lambda x.t$ of type $\sigma_1\sigma_2$ and let $(\mathcal{D},\mathcal{I})$ be an interpretation. Let $a \in \mathcal{D}(\sigma_1)$ be given. By the inductive hypothesis $\widehat{\mathcal{I}^x_a}(\Psi t) = \widehat{\mathcal{I}^x_a}(t)$. Generalizing over a, we conclude $\hat{\mathcal{I}}(\Psi t) = \hat{\mathcal{I}}(t)$. □

Using Lemma 5.4 and Theorem 5.7 we easily obtain the following corollary.

Corollary 5.8. Let s be a formula such that $FV(s) = \{x_1, \ldots, x_n\}$. We know

$$s \approx (x_1 \dot{=} x_1 \to \cdots \to x_n \dot{=} x_n \to \dot{\neg}\dot{\neg}\Psi s)$$

We now consider which properties of $\mathbf{R}$ are provable in $\mathcal{N}$. In particular, $\mathbf{R}^\sigma$ is provably symmetric and transitive, i.e., a PER (partial equivalence relation). As we previously mentioned, we cannot generally prove $\mathbf{R}^\sigma$ is reflexive in $\mathcal{N}$ since $\mathcal{N}$ lacks extensionality, but we can prove that $\mathbf{R}^\sigma$ is reflexive when σ is o or ι. Proofs of the next two lemmas are straightforward and can be found as Coq proofs in [18, 19].

Lemma 5.9. For each type σ we have $\vdash_{\mathcal{N}} \forall^\sigma xyz.\mathbf{R}^\sigma xy \to \mathbf{R}^\sigma yz \to \mathbf{R}^\sigma xz$. We also have $\vdash_{\mathcal{N}} \forall^\sigma xy.\mathbf{R}^\sigma xy \to \mathbf{R}^\sigma yx$.

Definition 5.10. A type σ is *reflexive* if $\vdash_{\mathcal{N}} \forall^\sigma x.\mathbf{R}^\sigma xx$.

Lemma 5.11. The types ι and o are reflexive.

One can use the model constructed in Example 5.4 of [1] to demonstrate $\not\vdash_{\mathcal{N}} \forall^{oo}x.\mathbf{R}^{oo}xx$ and $\not\vdash_{\mathcal{N}} \forall^{o\iota}x.\mathbf{R}^{o\iota}xx$. For more details see [18].

When translating tableau refutations we will sometimes need to consider a term t of type σ and need to know that $\Gamma \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}^{\sigma}(\Psi t)(\Psi t)$. One might try to prove this by a simple induction on t, but such an attempt will fail at the variable case. In general, we cannot prove $\Gamma \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}^{\sigma}\ x\ x$. However, we are able to prove (see Theorem 5.15) $\Gamma \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}^{\sigma}(\Psi t)(\Psi t)$ under the assumption that $\Gamma \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}^{\sigma}\ x\ x$ for every $x \in FV(t)$. Establishing Theorem 5.15 requires generalizing the result using simultaneous substitutions (Lemma 5.14). In some sense, Lemma 5.14 and Theorem 5.15 encapsulate a central reason why the translation works.

Lemma 5.12. For each $c \in \mathcal{C}$ we have $\vdash_{\mathcal{N}} \mathbf{R}\dot{c}\dot{c}$.

Proof. One must consider each case. Details are in [18]. □

Lemma 5.13. $\vdash_{\mathcal{N}} \forall fgxy.\dot{\neg}\dot{\neg}\mathbf{R}^{\sigma\tau}fg \to \dot{\neg}\dot{\neg}\mathbf{R}^{\sigma}xy \to \dot{\neg}\dot{\neg}\mathbf{R}^{\tau}(fx)(gy)$

Proof. This is straightforward using the definition of $\mathbf{R}^{\sigma\tau}$. □

Lemma 5.14. For all terms t, for all substitutions θ_1, θ_2 and, for all contexts Γ:

$$\text{if } \forall x \in FV(t) : \Gamma \vdash_{\mathcal{N}} \mathbf{R}(\theta_1(x))(\theta_2(x)) \text{ then } \Gamma \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}(\theta_1(\Psi t))(\theta_2(\Psi t))$$

Proof. We prove this lemma by structural induction on t.

- If t is a variable, then the result holds by the assumption.
- If t is a logical constant, then the result hold by Lemma 5.12.
- Suppose t is t_1t_2. By the inductive hypothesis we know $\Gamma \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}(\theta_1(\Psi t_1))(\theta_2(\Psi t_1))$ and $\Gamma \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}(\theta_1(\Psi t_2))(\theta_2(\Psi t_2))$. Hence $\Gamma \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}(\theta_1(\Psi t_1 t_2))(\theta_2(\Psi t_1 t_2))$ by Lemma 5.13.
- Suppose t is $\lambda x.t'$. Renaming variables if necessary, we assume x is chosen to avoid capture so that $\theta_1(\Psi t) = \lambda x.(\theta_1(\Psi t'))$ and $\theta_2(\Psi t) = \lambda x.(\theta_2(\Psi t'))$. It suffices to prove

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

  Let x_1 and x_2 be distinct fresh variables. Let Γ' be $\Gamma, (\mathbf{R}x_1x_2)$. For each $i \in \{1, 2\}$ let θ'_i be $\theta_i, [x := x_i]$. It is easy to see that we have

  $$\forall y \in FV(t') : \Gamma' \vdash_{\mathcal{N}} \mathbf{R}(\theta'_1(y))(\theta'_2(y)),$$

  because of the assumption about $FV(t)$ and the fact that $FV(t') \subseteq FV(t) \cup \{x\}$. We apply the inductive hypothesis to obtain $\Gamma' \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}\theta'_1(\Psi t')\theta'_2(\Psi t')$. □

Theorem 5.15. For all terms t, and for all contexts Γ:

$$\text{if } \forall x \in FV(t) : \Gamma \vdash_{\mathcal{N}} \mathbf{R}xx \text{ then } \Gamma \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}(\Psi t)(\Psi t)$$

Proof. Follows directly from Lemma 5.14 by using the identity substitution. □

We need to extend Ψ to map branches to contexts. The most obvious extension that just maps Ψ to all the formulas in the branch to obtain a context does not have the properties we want. Using the model $\mathcal{M}^{\beta f}$ constructed in Example 5.4 of [1] one can prove $\not\vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}^{oo}xx$. Consequently, the branch $x \neq^{oo} x$ is $\mathcal{T}$-refutable but $\dot{\neg}\Psi(x =^{oo} x)$ is not $\mathcal{N}$-refutable.

In Definition 5.16 below we define a branch translation $\Psi^* A$ which includes $\mathbf{R}xx$ for each free variable x in A. Following the definition we give a detailed example to further illustrate why such extra formulas are desired.

Definition 5.16 (The Branch Translation Ψ^*)**.** The branch translation Ψ^* maps a branch to a context as follows:

$$\Psi^* A := \{\Psi s | s \in A\} \cup \{\mathbf{R}^\sigma x\, x | x : \sigma \text{ and } x \in FV(A)\}$$

Example 5.17. Consider the formula $x \neq^{oo} x$. A tableau refutation of this formula starts with the branch $\{x \neq^{oo} x\}$. This branch is directly $\mathcal{T}$-refutable using the Closed$\neq$ rule. To mimic the Closed$\neq$ step in $\mathcal{N}$ we need to prove that $\Psi^*\{x \neq^{oo} x\}$ is $\mathcal{N}$-refutable. If $\Psi^*\{x \neq^{oo} x\}$ were defined as simply $\{\Psi(x \neq^{oo} x)\}$, then it would not be $\mathcal{N}$-refutable as mentioned above. Our definition of Ψ^* maps the branch $\{x \neq^{oo} x\}$ to the context $\{\dot\neg\mathbf{R}yy, \mathbf{R}yy\}$ which is obviously $\mathcal{N}$-refutable.

Example 5.18. We consider the translation of the formula $(\lambda p.\neg\neg p) = (\lambda p.p)$ proven in Example 3.3.

$$\begin{array}{lll} \Psi((\lambda p.\neg\neg p) = (\lambda p.p)) & \text{is} & \mathbf{R}^{oo}(\lambda p.\dot\neg\dot\neg p)(\lambda p.p) \\ & \text{is} & \forall pq : o.(p \doteq q \to \dot\neg\dot\neg((\dot\neg\dot\neg p) \doteq q)). \end{array}$$

A consequence of the final corollary in the next section is that the formula

$$\dot\neg\dot\neg\forall pq : o.(p \doteq q \to \dot\neg\dot\neg((\dot\neg\dot\neg p) \doteq q))$$

is $\mathcal{N}$-provable.

6 Translating Proofs

We prove that every $\mathcal{T}$-refutable branch A maps to an $\mathcal{N}$-refutable context $\Psi^* A$ (see Theorem 6.14). Implicitly this gives a translation from tableau refutations to natural deduction refutations. One may attempt to prove this by an induction on the $\mathcal{T}$-refutability of A, but a problem arises. Namely, a rule such as Forall may introduce a term t with free variables that prevent us from applying Theorem 5.15. In order to avoid this problem, we define a restricted tableau calculus $\mathcal{T}_r$.[1] We prove that if A is $\mathcal{T}$-refutable, then it is also $\mathcal{T}_r$-refutable. We then prove that if a branch A is $\mathcal{T}_r$-refutable, then $\Psi^* A$ is $\mathcal{N}$-refutable.

Definition 6.1 (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 is of type ι or o.

Definition 6.2 (Tableau Calculus $\mathcal{T}_r$)**.** The *tableau calculus* $\mathcal{T}_r$ contains all the tableau rules that are in $\mathcal{T}$ (see Definition 3.1) except for Forall, Func=, and Cut, for which it contains restricted forms as shown in Figure 3. This also defines the corresponding notion of $\mathcal{T}_r$-refutability.

In Proposition 6.7 below we prove that every $\mathcal{T}$-refutable branch is $\mathcal{T}_r$-refutable. The proof depends on a few simple lemmas.

Lemma 6.3 (Weakening)**.** If a branch A is $\mathcal{T}_r$-refutable, then every branch A' such that $A \subseteq A'$ is $\mathcal{T}_r$-refutable.

Proof. This is proven by induction on $\mathcal{T}_r$-refutability, taking care to rename variables from the Exists and FuncExt rules so they remain fresh. □

[1] The proof of Lemma 6.11 will show how using $\mathcal{T}_r$ resolves the problem.

$$\mathsf{Forall}_r \frac{\forall s}{\lceil s\ t \rceil^\beta}\ t \text{ is admissible for the branch} \qquad \mathsf{Cut}_r \frac{}{s \mid \neg s}\ s \text{ is admissible for the branch}$$

$$\mathsf{Func} =_r \frac{s_1 =^{\sigma\tau} s_2}{\lceil s_1\ t =^\tau s_2\ t \rceil^\beta}\ t \text{ is admissible for the branch}$$

Figure 3: Restricted Forall, Func=, and Cut Rules

Lemma 6.4. If $\neg\exists^\sigma x.x = x$ is in a branch A, then A is $\mathcal{T}_r$-refutable.

Proof. Using DeMorgan$\exists$ it suffices to prove $A \cup \{\forall^\sigma x.x \neq x\}$ is $\mathcal{T}_r$-refutable. The type σ has the form $\sigma_1 \cdots \sigma_n\alpha$ where $\alpha \in \{o, \iota\}$. Choose a variable y of type α and for each $i \in \{1, \ldots, n\}$ choose a variable z_i of type σ_i. Let t be the term $\lambda z_1 \cdots z_n.y$. We know $A \cup \{(\forall^\sigma x.x \neq x), t \neq t\}$ is $\mathcal{T}_r$-refutable using the Closed $\neq$ rule. The term t is of type σ and is admissible for the branch $A \cup \{\forall^\sigma x.x \neq x\}$ since y has type ι or o. Hence the rule Forall_r justifies that $A \cup \{\forall^\sigma x.x \neq x\}$ is $\mathcal{T}_r$-refutable. □

Definition 6.5. Let X be a finite set of variables. We define $\mathcal{E}^X$ to be the branch $\bigcup_{x \in X}\{(\exists x.x = x), (x = x)\}$.

Lemma 6.6. Let A be a branch and X be a finite set of variables such that $X \cap FV(A) = \emptyset$. If $A \cup \mathcal{E}^X$ is $\mathcal{T}_r$-refutable, then A is $\mathcal{T}_r$-refutable.

Proof. We prove this by induction on the number of variables in X. If X is empty, then the result is trivial since $\mathcal{E}^\emptyset$ is empty. Suppose X is $Y \cup \{x\}$ where $x \notin Y$. In this case $\mathcal{E}^X$ is $\mathcal{E}^Y \cup \{(\exists x.x = x), (x = x)\}$. Since x is not free in $A \cup \mathcal{E}^Y \cup \{(\exists x.x = x)\}$ and $A \cup \mathcal{E}^X$ is $\mathcal{T}_r$-refutable, we know $A \cup \mathcal{E}^Y \cup \{\exists x.x = x\}$ is $\mathcal{T}_r$-refutable via the Exists rule. By Lemma 6.4 we also know $A \cup \mathcal{E}^Y \cup \{\neg\exists x.x = x\}$ is $\mathcal{T}_r$-refutable. By Cut_r we know $A \cup \mathcal{E}^Y$ is $\mathcal{T}_r$-refutable. Finally, we conclude A is $\mathcal{T}_r$-refutable using the inductive hypothesis. □

Now we are in a position to prove that if a branch is $\mathcal{T}$-refutable, then it is $\mathcal{T}_r$-refutable. The proof implicitly describes an algorithm for modifying a tableau refutation so that it only uses the restricted rules.

Proposition 6.7. If a branch A is $\mathcal{T}$-refutable, then A is $\mathcal{T}_r$-refutable.

Proof. The proof is by induction on $\mathcal{T}$-refutability. Suppose $\mathcal{T}$-refutability of A follows from the step $\langle A_1, \ldots, A_n, A\rangle$ in $\mathcal{T}$ where A_i is $\mathcal{T}$-refutable for each $i \in \{1, \ldots, n\}$. By the inductive hypothesis, we know A_i is $\mathcal{T}_r$-refutable for each $i \in \{1, \ldots, n\}$. If $\langle A_1, \ldots, A_n, A\rangle$ is a step in $\mathcal{T}_r$, then we are done. Otherwise, $\langle A_1, \ldots, A_n, A\rangle$ must be a step in one of the Forall, Func=, or Cut rules and the new term used in the rule contains free variables not in A. We consider the Forall rule; the others are similar. Suppose $\forall^\sigma s \in A$, $n = 1$ and A_1 is $A, \lceil st \rceil^\beta$. Let X be $FV(t) \setminus FV(A)$. Clearly X is finite and $X \cap FV(A) = \emptyset$. By weakening (Lemma 6.3) and $\mathcal{T}_r$-refutability of A_1, we know $A, \lceil st \rceil^\beta \cup \mathcal{E}^X$ is $\mathcal{T}_r$-refutable. Clearly every free variable of t is free in $A, \lceil st \rceil^\beta \cup \mathcal{E}^X$. Hence, we can use the Forall_r rule to conclude $A \cup \mathcal{E}^X$ is $\mathcal{T}_r$-refutable. By Lemma 6.6 we know A is $\mathcal{T}_r$-refutable. □

Corollary 6.8. The tableau calculus $\mathcal{T}_r$ is complete.

Proof. This is a direct consequence of Proposition 6.7 and the completeness of $\mathcal{T}$. □

We can now turn to the main part of the translation from restricted tableau refutations to natural deduction refutations. We prove that if A is $\mathcal{T}_r$-refutable, then $\Psi^* A$ is $\mathcal{N}$-refutable. We can reduce this to a local property – the property of a rule being respected.

Definition 6.9. We say a rule (a set of steps) is *respected* if the following holds for every step $\langle A_1, \ldots, A_n, A\rangle$ in the rule: If $\Psi^* A_i \vdash_{\mathcal{N}} \dot{\bot}$ holds for all $i \in \{1, \ldots, n\}$, then $\Psi^* A \vdash_{\mathcal{N}} \dot{\bot}$ holds.

We prove that every rule defining $\mathcal{T}_r$ is respected. The fact that the rules are respected follows from the $\mathcal{N}$-provability of certain formulas. We prove this for the rules Exists and Forall_r in some detail. For the remaining rules we mainly give corresponding $\mathcal{N}$-provable formulas.

Lemma 6.10. The Exists rule is respected.

Proof. Let s be a term, A be a branch containing $\exists^\sigma s$, and x be a fresh variable.
Assume that $\Psi^*(A \cup \{\lceil sx \rceil^\beta\}) \vdash_{\mathcal{N}} \dot{\bot}$. We want to show that $\Psi^* A \vdash_{\mathcal{N}} \dot{\bot}$. Note that if y occurs free in $\lceil sx \rceil^\beta$ then

$$\Psi^*(A \cup \{\lceil sx \rceil^\beta\}) = \Psi^*(A) \cup \{\Psi(\lceil sx \rceil^\beta)\} \cup \{\mathbf{R}xx\},$$

otherwise

$$\Psi^*(A \cup \{\lceil sx \rceil^\beta\}) = \Psi^*(A) \cup \{\Psi(\lceil sx \rceil^\beta)\}.$$

In either case we know $\Psi^*(A) \cup \{\Psi(\lceil sx \rceil^\beta)\} \cup \{\mathbf{R}xx\} \vdash_{\mathcal{N}} \dot{\bot}$ by assumption and possibly the wk rule. By Lemma 5.3 we know $\Psi(\lceil sx \rceil^\beta)$ is β-equivalent to $\Psi(sx)$, i.e., $(\Psi s)x$. Using $\to I$ and β we have $\Psi^* A \vdash_{\mathcal{N}} \mathbf{R}^\sigma xx \to (\Psi s)x \to \dot{\bot}$. Since $x \notin FV(A)$ we can use $\forall I$ to obtain

$$\Psi^* A \vdash_{\mathcal{N}} \forall^\sigma x.\mathbf{R}^\sigma xx \to (\Psi s)x \to \dot{\bot}.$$

Since $\exists s \in A$, we know $\Psi(\exists s) \in \Psi^* A$ and thus $\Psi^* A \vdash_{\mathcal{N}} \dot{\exists}^\sigma(\Psi s)$ by the hy rule. The following is easy to verify (for a Coq proof term see [18, 19]) :

$$\vdash_{\mathcal{N}} \forall^{\sigma o} f.(\forall^\sigma x.\mathbf{R}^\sigma xx \to fx \to \dot{\bot}) \to (\dot{\exists}^\sigma f) \to \dot{\bot}.$$

Hence we have $\Psi^* A \vdash_{\mathcal{N}} \dot{\bot}$ as desired. □

The argument for the Forall_r rule is similar. In this case we must make use of Theorem 5.15.

Lemma 6.11. The Forall_r rule is respected.

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

$$\Psi^*(A \cup \{\lceil st \rceil^\beta\}) \subseteq \Psi^*(A) \cup \{\Psi(\lceil st \rceil^\beta)\} \cup \{\mathbf{R}xx | x \in FV(t)\}.$$

By assumption and possibly the wk rule we know

$$\Psi^*(A) \cup \{\Psi(\lceil st \rceil^\beta)\} \cup \{\mathbf{R}xx | x \in FV(t)\} \vdash_{\mathcal{N}} \dot{\bot}.$$

We know t is admissible. Thus for each $x \in FV(t)$ either $x \in FV(A)$ or x has the reflexive type ι or o. Hence for each $x \in FV(t)$ we know $\Psi^* A \vdash_{\mathcal{N}} \mathbf{R}xx$. Consequently, we have $\Psi^*(A) \cup \{\Psi(\lceil st \rceil^\beta)\} \vdash_{\mathcal{N}} \dot{\bot}$. Using Lemma 5.3 we know

$$\Psi^* A \vdash_{\mathcal{N}} (\Psi s)(\Psi t) \to \dot{\bot}.$$

Applying Theorem 5.15 we also have

$$\Psi^* A \vdash_{\mathcal{N}} \dot{\neg}\dot{\neg}\mathbf{R}^\sigma(\Psi t)(\Psi t).$$

Closed: $\forall^o p.p \to (\dot\neg p) \to \dot\bot$

Closed$\bot$: $\dot\bot \to \dot\bot$

Closed$\neg\top$: $(\dot\neg\dot\top) \to \dot\bot$

Closed$\neq$: $\forall^\sigma x.\dot\neg\dot\neg(\mathbf{R}^\sigma xx) \to \dot\neg(\mathbf{R}^\sigma xx) \to \dot\bot$

ClosedSym: $\forall^\sigma xy.(\mathbf{R}^\sigma xy) \to \dot\neg(\mathbf{R}^\sigma yx) \to \dot\bot$

Cut$_r$: $\forall^o p.(p \to \dot\bot) \to ((\dot\neg p) \to \dot\bot) \to \dot\bot$

DNeg: $\forall^o p.(p \to \dot\bot) \to (\dot\neg\dot\neg p) \to \dot\bot$

And: $\forall p\, q.(p \to q \to \dot\bot) \to (p \dot\wedge q) \to \dot\bot$

Or: $\forall p\, q.(p \to \dot\bot) \to (q \to \dot\bot) \to (p \dot\vee q) \to \dot\bot$

Imp: $\forall^o p\, q.((\dot\neg p) \to \dot\bot) \to (q \to \dot\bot) \to (p \to q) \to \dot\bot$

NegAnd: $\forall p\, q.((\dot\neg p) \to \dot\bot) \to ((\dot\neg q) \to \dot\bot) \to \dot\neg(p \dot\wedge q) \to \dot\bot$

NegOr: $\forall p\, q.((\dot\neg p) \to (\dot\neg q) \to \dot\bot) \to \dot\neg(p \dot\vee q) \to \dot\bot$

NegImp: $\forall^o p\, q.(p \to (\dot\neg q) \to \dot\bot) \to (\dot\neg(p \to q)) \to \dot\bot$

DeMorgan$\forall$: $\forall^{\sigma\to o} f.((\dot\exists^\sigma(\lambda x.\dot\neg fx)) \to \dot\bot) \to (\dot\neg(\dot\forall^\sigma f)) \to \dot\bot$

DeMorgan$\exists$: $\forall^{\sigma\to o} f.((\dot\forall^\sigma(\lambda x.\dot\neg fx)) \to \dot\bot) \to (\dot\neg(\dot\exists^\sigma f)) \to \dot\bot$

Bool$=$: $\forall^o pq.(p \to q \to \dot\bot) \to ((\dot\neg p) \to (\dot\neg q) \to \dot\bot) \to (\mathbf{R}^o pq) \to \dot\bot$

BoolExt: $\forall^o pq.(p \to (\dot\neg q) \to \dot\bot) \to (q \to (\dot\neg p) \to \dot\bot) \to \dot\neg(\mathbf{R}^o pq) \to \dot\bot$

FuncExt: $\forall^{\sigma\tau} kh.\dot\neg\dot\neg(\mathbf{R}^{\sigma\tau} hh) \to (\forall^\sigma x.(\mathbf{R}^\sigma xx) \to \dot\neg(\mathbf{R}^\tau(kx)(hx)) \to \dot\bot) \to \dot\neg(\mathbf{R}^{\sigma\tau} kh) \to \dot\bot$

Func$=_r$: $\forall^{\sigma\tau} kh.\forall^\sigma t.\dot\neg\dot\neg(\mathbf{R}^\sigma tt) \to ((\mathbf{R}^\tau(kt)(ht)) \to \dot\bot) \to (\mathbf{R}^{\sigma\tau} kh) \to \dot\bot$

Mat: $\forall^{\sigma_1\sigma_2\dots\sigma_n o} p.\forall^{\sigma_1} x_1 y_1.\forall^{\sigma_2} x_2 y_2.\dots.\forall^{\sigma_n} x_n y_n.\dot\neg\dot\neg(\mathbf{R}^{\sigma_1\sigma_2\dots\sigma_n o} pp) \to$
$(\dot\neg(\mathbf{R}^{\sigma_1} x_1 y_1) \to \dot\bot) \to (\dot\neg(\mathbf{R}^{\sigma_2} x_2 y_2) \to \dot\bot) \to \dots \to (\dot\neg(\mathbf{R}^{\sigma_n} x_n y_n) \to \dot\bot) \to$
$px_1x_2\dots x_n \to \dot\neg(py_1y_2\dots y_n) \to \dot\bot$

Dec: $\forall^{\sigma_1\sigma_2\dots\sigma_n\iota} h.\forall^{\sigma_1} x_1 y_1.\forall^{\sigma_2} x_2 y_2.\dots.\forall^{\sigma_n} x_n y_n.\dot\neg\dot\neg(\mathbf{R}^{\sigma_1\sigma_2\dots\sigma_n\iota} hh) \to$
$(\dot\neg(\mathbf{R}^{\sigma_1} x_1 y_1) \to \dot\bot) \to (\dot\neg(\mathbf{R}^{\sigma_2} x_2 y_2) \to \dot\bot) \to \dots \to (\dot\neg(\mathbf{R}^{\sigma_n} x_n y_n) \to \dot\bot) \to$
$\dot\neg(\mathbf{R}^\iota(hx_1x_2\dots x_n)(hy_1y_2\dots y_n)) \to \dot\bot$

Con: $\forall^\iota xyzw.(\dot\neg(\mathbf{R}^\iota xz) \to \dot\neg(\mathbf{R}^\iota yz) \to \dot\bot) \to (\dot\neg(\mathbf{R}^\iota xw) \to \dot\neg(\mathbf{R}^\iota yw) \to \dot\bot) \to$
$(\mathbf{R}^\iota xy) \to \dot\neg(\mathbf{R}^\iota zw) \to \dot\bot$

Figure 4: Formulas provable in $\mathcal{N}$ used in the proof of Lemma 6.12

Since $\forall s \in A$, we know $\Psi(\forall s) \in \Psi^* A$ thus $\Psi^* A \vdash_{\mathcal{N}} \dot\forall^\sigma(\Psi s)$ by the *hy* rule. The following is easy to verify (for a Coq proof term see [18, 19]) :

$$\vdash_{\mathcal{N}} \forall^{\sigma o} f.\forall^\sigma x.\dot\neg\dot\neg\mathbf{R}^\sigma xx \to (fx \to \dot\bot) \to (\dot\forall^\sigma f) \to \dot\bot$$

Hence we have $\Psi^* A \vdash_{\mathcal{N}} \dot\bot$ as desired. □

The proofs of Lemmas 6.10 and 6.11 illustrate how one proves that a rule is respected. For the remaining rules we will simply indicate the formulas whose $\mathcal{N}$-provability implies the rule is respected.

Lemma 6.12. All of the rules in $\mathcal{T}_r$ are respected.

Proof. We have already proven this for Exist in Lemma 6.10 and Forall$_r$ in Lemma 6.11. For the remaining rules one can argue similarly making use of formulas which are provable in $\mathcal{N}$ and correspond to the structure of the rule. We display formulas corresponding to the remaining

rules in Figure 4. Proof terms (in Coq) for these formulas are available in [18, 19] with the exception of the lemmas for Mat and Dec. The lemmas for Mat and Dec have been formulated and proven in Coq for the cases with 1 and 2 arguments. □

Proposition 6.13. If A is $\mathcal{T}_r$-refutable, then $\Psi^* A$ is $\mathcal{N}$-refutable.

Proof. The proof is by an easy induction on the $\mathcal{T}_r$-refutation using Lemma 6.12 at each step. □

We finally conclude similar results for $\mathcal{T}$-refutability.

Theorem 6.14. If A is $\mathcal{T}$-refutable, then $\Psi^* A$ is $\mathcal{N}$-refutable. Also, if s is a ground formula and s is $\mathcal{T}$-refutable, then Ψs is $\mathcal{N}$-refutable.

Proof. This follows from Propositions 6.7 and 6.13. □

We can also conclude the following using Theorem 6.14 and Corollary 5.8.

Corollary 6.15. Let s be a formula such that $FV(s) = \{x_1, \ldots, x_n\}$. If s is $\mathcal{T}$-provable, then $s \approx (x_1 \doteq x_1 \to \cdots \to x_n \doteq x_n \to \dot{\neg}\dot{\neg}\Psi s)$ and $(x_1 \doteq x_1 \to \cdots \to x_n \doteq x_n \to \dot{\neg}\dot{\neg}\Psi s)$ is $\mathcal{N}$-provable.

7 Conclusion

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 intentional natural deduction proof of s'. We defined two tableau calculi $\mathcal{T}$ and $\mathcal{T}_r$ and proved that whenever a branch is $\mathcal{T}$-refutable, it is also $\mathcal{T}_r$-refutable (Proposition 6.7). Moreover, we gave a translation Ψ and proved that it maps higher-order formulas to semantically equivalent formulas in the sense of Henkin semantics (Theorem 5.7). Furthermore, we defined a branch translation Ψ^* that maps branches to contexts and proved that for any $\mathcal{T}_r$-refutable branch A, $\Psi^* A$ is $\mathcal{N}$-refutable (Proposition 6.13). We concluded that for any $\mathcal{T}$-provable formula s with free variables $x_1, \ldots, x_n$, the formula

$$\Psi(x_1 = x_1) \to \cdots \to \Psi(x_n = x_n) \to \Psi(\neg\neg s)$$

is semantically equivalent to s and is $\mathcal{N}$-provable (Corollary 6.15). Hence for any ground formula s that is $\mathcal{T}$-provable, $\Psi(\neg\neg s)$ is $\mathcal{N}$-provable.

Several issues are still open for future work. One may want to determine the precise relationship between our translation Ψ and the translation given by Gandy [9]. One could also investigate to what extent the translation can be extended to handle a choice operator.

A choice operator is a logical constant $\varepsilon^\sigma : (\sigma o)\sigma$ satisfying $\forall p : \sigma o. \forall x : \sigma. px \to p(\varepsilon^\sigma p)$. Such operators are supported by Satallax. To extend the translation in this paper to handle ε^σ one would need to find a term $\Psi\varepsilon^\sigma$ satisfying

$$\vdash_{\mathcal{N}} \Psi(\neg\neg(\varepsilon^\sigma = \varepsilon^\sigma))$$

and

$$\vdash_{\mathcal{N}} \Psi(\neg\neg\forall p : \sigma o. \forall x : \sigma. px \to p(\varepsilon^\sigma p)).$$

This will not generally be possible, but might be possible in special situations. For example, one might restrict to having the choice operator only at the base type ι and assume that the base type ι is finite.

On the more practical side, one can implement a mapping from tableau proofs to natural deduction proof terms. This would enable proof checking the tableau proofs that Satallax outputs using Coq. This implementation could make use of the Coq lemmas that are provided in [18, 19].

References

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

[2] C. Benzmüller, L. Paulson, F. Theiss, and A. Fietzke. `LEO-II` — A cooperative automatic theorem prover for classical higher-order logic. In *Fourth International Joint Conference on Automated Reasoning (IJCAR'08)*, volume 5195 of *LNCS (LNAI)*, pages 162–170. Springer, 2008.

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

[4] C. E. Brown. Satallax: An automated higher-order prover. In U. S. Bernhard Gramlich, Dale Miller, editor, *6th International Joint Conference on Automated Reasoning (IJCAR 2012)*, pages 111 – 117. Springer, 2012.

[5] C. E. Brown and C. Rizkallah. Glivenko and Kuroda for simple type theory. Technical report, Saarland University, Dec 2011. Article to be published in *Journal of Symbolic Logic*.

[6] C. E. Brown and G. Smolka. Analytic tableaux for simple type theory and its first-order fragment. *Logical Methods in Computer Science*, 6(2), Jun 2010.

[7] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5(1):56–68, 1940.

[8] N. G. de Bruijn. A survey of the project AUTOMATH. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*, pages 579–606. Academic Press, 1980.

[9] R. O. Gandy. On the axiom of extensionality–part I. *Journal of Symbolic Logic*, 21(1):36–48, 1956.

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

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

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

[13] W. A. Howard. The formula-as-types notion of construction. In J. P. Seldin and J. R. Hindley, editors, *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism*, pages 480–490. Academic Press, 1980.

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

[15] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

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

[17] D. Prawitz. *Natural deduction: a proof-theoretical study*. PhD thesis, Almqvist & Wiksell, 1965.

[18] C. Rizkallah. Proof representations for higher-order logic. Master's thesis, Saarland University, Saarbruecken, Germany, Dec 2009.

[19] C. Rizkallah. Proof representations for higher-order logic: Coq proofs, 2009. `http://www.mpi-inf.mpg.de/~crizkall/Full_Tableau_ND_Translation.v`.

[20] A. Teucke. Translating a Satallax refutation to a tableau refutation encoded in Coq. Bachelor's thesis, Universität des Saarlandes, 2011.