Automation of the Principia Metaphysica in HOL: Part I

Christoph E. Benzmüller and Paul E. Oppenheimer and Edward N. Zalta

December 9, 2015

1 Introduction

We present a formalisation and partial automation of an initial part of the (third authors) Principia Metaphysica [6] in Isabelle/HOL [5].

The Principia Metaphysica, which is based on and extends the Theory of Abstract Objects [?], employs a modal relational type theory as logical foundation. Arguments defending this choice against a modal functional type theory have been presented before [7]. In a nutshell, the situation is this: functional type theory comes with strong comprehension principles, which, in the context of the Theory of Abstract Objects, have paradoxical implications. When starting off with a relational foundation, however, weaker comprehension principles are provided, and these obstacles can be avoided.

Isabelle/HOL is a proof assistant based on a functional type theory, more precisely, Church's theory of types [4]. Recently, it has been shown that Church's type theory can be elegantly utilized as a meta-logic to encode and automate various quantified non-classical logics, including modal functional type theory [2, 3]. This work has subsequently been employed in a case study in computational metaphysics, in which different variants of Kurt Gdel's ontological argument [1] were verified (respectively, falsified).

The motivating research questions for the work presented below include:

- Can functional type theory, despite the problems as pointed by Zalta and Oppenheimer, be utilized to encode the Theory of Abstract Objects when following the embeddings approach?
- How elegant and user-friendly is the resulting formalization? In other words, to what extend can Isabelle's user interface be facilitated to hide unpleasant technicalities of the (extended) embedding from the user?
- How far can automation be pushed in the approach? How much user interaction can be avoided in the formalization of the (first part) of the Principia Metaphysica?
- Can the consistency of the theory be validated with the available automated reasoning tools?
- Can the reasoners eventually even contribute some new knowledge? . . .

The encoding of modal functional type theory in functional type theory as explored in previous work [2, 3] is simple: modal logic formulas are identified with certain functional type theory

$$
\begin{array}{rrcl}
 & \delta & ::= & a_1, a_2, \ldots \\
 & \nu & ::= & x_1, x_2, \ldots \\
(n \geq 0) & \Sigma^n & ::= & P^n_1, P^n_2, \ldots \\
(n \geq 0) & \Omega^n & ::= & F^n_1, F^n_2, \ldots \\
 & \alpha & ::= & \nu \mid \Omega^n \ (n \geq 0) \\
 & \kappa & ::= & \delta \mid \nu \mid \iota\nu\varphi \\
(n \geq 1) & \Pi^n & ::= & \Sigma^n \mid \Omega^n \mid [\lambda\nu_1 \ldots \nu_n \ \varphi^*] \\
 & \Pi^0 & ::= & \Sigma^0 \mid \Omega^0 \mid [\lambda \ \varphi^*] \mid \varphi^* \\
 & \varphi^* & ::= & \Pi^n\kappa_1 \ldots \kappa_n \ (n \geq 1) \mid \Pi^0 \mid (\neg\varphi^*) \mid (\varphi^* \to \varphi^*) \mid \forall\alpha\varphi^* \mid \\
 & & & (\Box\varphi^*) \mid (\mathcal{A}\varphi^*) \\
 & \varphi & ::= & \kappa_1\Pi^1 \mid \varphi^* \mid (\neg\varphi) \mid (\varphi \to \varphi) \mid \forall\alpha\varphi \mid (\Box\varphi) \mid (\mathcal{A}\varphi) \\
 & \tau & ::= & \kappa \mid \Pi^n \ (n \geq 0)
\end{array}
$$

δ	individual constants
ν	individual variables
Σ^n	n-place relation constants ($n \geq 0$)
Ω^n	n-place relation variables ($n \geq 0$)
α	variables
κ	individual terms
Π^n	n-place relation terms ($n \geq 0$)
φ^*	propositional formulas
φ	formulas
τ	terms

Figure 1: Grammar of Modal Relational Type Theory. Note that two kinds of (complex) formulas are introduced: ones that may have encoding subformulas and ones that dont. The latter are designated as propositional formulas, the former ones simply as formulas.

formulas of predicate type *io*. Possible worlds are explicitly represented by terms of type *i*. A modal logic φ formula holds for a world w if and only if the application $\varphi\ w$ evaluates to true. The definition of the propositional modal logic connectives is then straightforward and it simply realizes the standard translation as a set of equations in functional type theory. The approach has been successfully extended for quantifiers. A crucial aspect thereby is that in simple type theory quantifiers can be treated as ordinary logical connectives. No extra binding mechanism is needed since the already existing lambda binding mechanism can be elegantly utilized.

The challenge in this work is to appropriately 'restrict' this embedding for modal relational type theory.

To achieve this we provide means to explicitly represents and maintain information and constraints on the syntactical structure of modal relational type theory, in particular, we provide means to distinguish between propositional formulas, formulas, terms and erreneous (disallowed) formations. This clearly creates some technical overhead. However, we exploit facilities in Isabelle/HOL's user interface, and other means, to hide most of these technicalities from the user in applications.

2 Preliminaries

We start out with some type declarations and type abbreviations. Our formalism explicitly encodes Kripke style semantics. Hence, we introduce a distinguished type *i* to represent the set of possible worlds. Consequently, terms of this type denote possible worlds. Moreover, modal logic formulas are associated in our approach with predicates (resp. sets) on possible worlds. Hence, modal logic formulas have type $(i \Rightarrow bool)$. To make our representation in the remainder more concise we abbreviate this type as *io*.

typedecl *i*
type-synonym $io = (i \Rightarrow bool)$

Entities in the abstract theory of types are represented in our formalism by the type *e*. We call this the raw type of entities resp. objects. Later on we will introduce means to distinguish between abstract and ordinary entities.

typedecl e

To explicitly model the syntactical restrictions of modal relational type theory we introduce a datatype $'a\ opt$ based on four constructors: *Error* $'a$ (identifies erroneous term constructions), *PropForm* $'a$ (identifies propositional formulas), *Form* $'a$ (identifies formulas), and *Term* $'a$ (identifies terms, such as lambda abstractions). The embeddings approach will be suitably adapted below so that for each language expression (in the embedded modal relational type theory) the respective datatype is identified and appropriately propagated. The encapsulated expressions (the polymorphic type $'a$ will be instantiated below) realize the actual modeling of the logic embedding analogous to previous work for modal functional type theory.

datatype $'a\ opt = Error\ 'a \mid PropForm\ 'a \mid Form\ 'a \mid Term\ 'a$

Some language constructs in the theory of abstract types, e.g. the actuality operator $\mathcal{A}$ (for "it is actually the case that"), refer to a (fixed) given world. To model such a global world reference we introduce a constant symbol (name) *cw* of world type i. Moreover, for technical reasons, which will be clarified below, we introduce further (dummy) constant symbols for various domains. Since we assume that all domains are non-empty, introducing these constant symbols is obviously not harmful.

consts $cw :: i$
consts $de{::}e\ dio{::}io\ da{::}'a$

3 Embedding of Modal Relational Type Theory

The language constructs of modal relational type theory are introduced step by step.

The actuality operator $\mathcal{A}$ when applied to a formula or propositional formula φ evaluates φ wrt the fixed given world *cw*. The compound expression $\mathcal{A}\ \varphi$ inherits its syntactical category (*Form* or *PropForm*) from φ. If the grammatical catagory of φ is *Error* or *Term*, then the grammatical catagory of $\mathcal{A}\ \varphi$ is *Error* and a dummy entity of appropriate type is returned. This illustrates the very idea of our explicit structure and constraints and this scheme will repeated below for all the other language constructs of modal relational type theory.

abbreviation $\mathcal{A}{::}io\ opt \Rightarrow io\ opt$ **where** $\mathcal{A}\ \varphi \equiv case\ \varphi\ of$
$\quad Form\ \psi \Rightarrow Form\ (\lambda w.\ \psi\ cw) \mid PropForm\ \psi \Rightarrow PropForm\ (\lambda w.\ \psi\ cw) \mid \text{-} \Rightarrow Error\ dio$

The Principia Metaphysica distinguishes between encoding and exemplifying, ... say more ... Encoding $\kappa_1\Pi^1$ is noted here as $<\kappa_1\circ\Pi^1>$. Encoding yields formulas and never propositional formulas. In the embedding encoding is identified with predicate application.

abbreviation $Enc{::}e\ opt{\Rightarrow}(e{\Rightarrow}io)\ opt{\Rightarrow}io\ opt$ **where** $<x\circ P> \equiv case\ (x,P)\ of$
$\quad (Term\ y, Term\ Q) \Rightarrow Form\ (\lambda w.(Q\ y)\ w) \mid \text{-} \Rightarrow Error\ dio$

Exemplifying formulas $\Pi^1\kappa_1$ are noted here as $<\Pi^1{\cdot}\kappa_1>$. Exemplification yields propositional formulas and never formulas. In the embedding exemplification is identified with predicate application, just as encoding.

abbreviation $Exe1{::}(e{\Rightarrow}io)\ opt{\Rightarrow}e\ opt{\Rightarrow}io\ opt$ **where** $<P{\cdot}x> \equiv case\ (P,x)\ of$
$\quad (Term\ Q, Term\ y) \Rightarrow PropForm\ (\lambda w.(Q\ y)\ w) \mid \text{-} \Rightarrow Error\ dio$

The Principia Metaphysica supports n-ary exemplification constructions. For pragmatical reasons we consider here the cases only for $n \leq 3$.

abbreviation $\mathit{Exe2}{::}(e{\Rightarrow}e{\Rightarrow}io)\ \mathit{opt}{\Rightarrow}e\ \mathit{opt}{\Rightarrow}e\ \mathit{opt}{\Rightarrow}io\ \mathit{opt}$ **where** $<P{\cdot}x1,x2> \equiv \mathit{case}\ (P,x1,x2)\ \mathit{of}$
$(\mathit{Term}\ Q,\mathit{Term}\ y1,\mathit{Term}\ y2) \Rightarrow \mathit{PropForm}\ (\lambda w.(Q\ y1\ y2)\ w) \mid \text{-} \Rightarrow \mathit{Error}\ \mathit{dio}$
abbreviation $\mathit{Exe3}{::}(e{\Rightarrow}e{\Rightarrow}e{\Rightarrow}io)\ \mathit{opt}{\Rightarrow}e\ \mathit{opt}{\Rightarrow}e\ \mathit{opt}{\Rightarrow}e\ \mathit{opt}{\Rightarrow}io\ \mathit{opt}$ **where** $<P{\cdot}x1,x2,x3> \equiv \mathit{case}\ (P,x1,x2,x3)\ \mathit{of}$
$(\mathit{Term}\ Q,\mathit{Term}\ y1,\mathit{Term}\ y2,\mathit{Term}\ y3) \Rightarrow \mathit{PropForm}\ (\lambda w.(Q\ y1\ y2\ y3)\ w) \mid \text{-} \Rightarrow \mathit{Error}\ \mathit{dio}$

Formations with negation and implication are supported for both, formulas and propositional formulas, and their embeddings are straightforward.

abbreviation $\mathit{not}{::}io\ \mathit{opt}{\Rightarrow}io\ \mathit{opt}$ **where** $\neg\ \varphi \equiv \mathit{case}\ \varphi\ \mathit{of}$
$\mathit{Form}\ \psi \Rightarrow \mathit{Form}\ (\lambda w.\neg(\psi\ w)) \mid \mathit{PropForm}\ \psi \Rightarrow \mathit{PropForm}\ (\lambda w.\neg(\psi\ w)) \mid \text{-} \Rightarrow \mathit{Error}\ \mathit{dio}$
abbreviation $\mathit{implies}{::}io\ \mathit{opt}{\Rightarrow}io\ \mathit{opt}{\Rightarrow}io\ \mathit{opt}$ **where** $\varphi \rightarrow \psi \equiv \mathit{case}\ (\varphi,\psi)\ \mathit{of}$
$(\mathit{Form}\ \alpha,\mathit{Form}\ \beta) \Rightarrow \mathit{Form}\ (\lambda w.\ \alpha\ w \longrightarrow \beta\ w) \mid (\mathit{PropForm}\ \alpha,\mathit{PropForm}\ \beta) \Rightarrow \mathit{PropForm}\ (\lambda w.\ \alpha\ w \longrightarrow \beta\ w) \mid \text{-} \Rightarrow \mathit{Error}\ \mathit{dio}$

universal quantification; $\forall\,(\lambda x.\varphi)$ inherits its kind (Form or PropForm) from φ; Error is passed on $\forall\,(\lambda x.\varphi)$ is mapped to $(\lambda w.\forall\,x.\varphi x w)$ as expected

abbreviation $\mathit{forall}{::}('a{\Rightarrow}io\ \mathit{opt}){\Rightarrow}io\ \mathit{opt}$ **where** $\forall\ \ \Phi \equiv \mathit{case}\ (\Phi\ da)\ \mathit{of}$
$\mathit{PropForm}\ \varphi \Rightarrow \mathit{PropForm}\ (\lambda w.\ \forall\,x.\ \mathit{case}\ (\Phi\ x)\ \mathit{of}\ \mathit{PropForm}\ \psi \Rightarrow \psi\ w)$
$\mid \mathit{Form}\ \varphi \Rightarrow \mathit{Form}\ (\lambda w.\ \forall\,x.\ \mathit{case}\ (\Phi\ x)\ \mathit{of}\ \mathit{Form}\ \psi \Rightarrow \psi\ w)$
$\mid \text{-} \Rightarrow \mathit{Error}\ \mathit{dio}$

abbreviation $\mathit{box}{::}io\ \mathit{opt}{\Rightarrow}io\ \mathit{opt}$ **where** $\Box\ \varphi \equiv \mathit{case}\ \varphi\ \mathit{of}$
$\mathit{Form}\ \psi \Rightarrow \mathit{Form}\ (\lambda w.\ \forall\,v.\ \psi\ v)$
$\mid \mathit{PropForm}\ \psi \Rightarrow \mathit{PropForm}\ (\lambda w.\ \forall\,v.\ \psi\ v)$
$\mid \text{-} \Rightarrow \mathit{Error}\ \mathit{dio}$

box operator; $\Box\ \varphi$ inherits its type (Form or PropForm) from φ; Error is passed on. Note that the $\Box$-operator is defined here without an accessibility relation; this is ok since we assume logic S5.

abbreviation $\mathit{lam0}{::}io\ \mathit{opt}{\Rightarrow}io\ \mathit{opt}$ **where** $\lambda^0\ \varphi \equiv \mathit{case}\ \varphi\ \mathit{of}$
$\mathit{PropForm}\ \psi \Rightarrow \mathit{PropForm}\ \psi$
$\mid \text{-} \Rightarrow \mathit{Error}\ \mathit{dio}$

0-arity lambda abstraction; $\lambda^0\ \varphi$ returns PropForm φ if φ is a PropForm, otherwise Error

abbreviation $\mathit{lam1}{::}(e{\Rightarrow}io\ \mathit{opt}){\Rightarrow}(e{\Rightarrow}io)\ \mathit{opt}$ **where** $\lambda^1\ \Phi \equiv \mathit{case}\ (\Phi\ de)\ \mathit{of}$
$\mathit{PropForm}\ \varphi \Rightarrow \mathit{Term}\ (\lambda x.\ \mathit{case}\ (\Phi\ x)\ \mathit{of}\ \mathit{PropForm}\ \varphi \Rightarrow \varphi)$
$\mid \text{-} \Rightarrow \mathit{Error}\ (\lambda x.\ \mathit{dio})$

1-arity lambda abstraction; $\lambda^1(\lambda x.\varphi)$ returns Term $(\lambda x.\varphi)$ if φ is a PropForm, otherwise Error

abbreviation $\mathit{lam2}{::}(e{\Rightarrow}e{\Rightarrow}io\ \mathit{opt}){\Rightarrow}(e{\Rightarrow}e{\Rightarrow}io)\ \mathit{opt}$ **where** $\lambda^2\ \Phi \equiv \mathit{case}\ (\Phi\ de\ de)\ \mathit{of}$
$\mathit{PropForm}\ \varphi \Rightarrow \mathit{Term}\ (\lambda x\ y.\ \mathit{case}\ (\Phi\ x\ y)\ \mathit{of}\ \mathit{PropForm}\ \varphi \Rightarrow \varphi)$
$\mid \text{-} \Rightarrow \mathit{Error}\ (\lambda x\ y.\ \mathit{dio})$

2-arity lambda abstraction; $\lambda^2(\lambda xy.\varphi)$ returns Term $(\lambda xy.\varphi)$ if φ is a PropForm, otherwise Error

abbreviation $\mathit{lam3}{::}(e{\Rightarrow}e{\Rightarrow}e{\Rightarrow}io\ \mathit{opt}){\Rightarrow}(e{\Rightarrow}e{\Rightarrow}e{\Rightarrow}io)\ \mathit{opt}$ **where** $\lambda^3\ \Phi \equiv \mathit{case}\ (\Phi\ de\ de\ de)\ \mathit{of}$
$\mathit{PropForm}\ \varphi \Rightarrow \mathit{Term}\ (\lambda x\ y\ z.\ \mathit{case}\ (\Phi\ x\ y\ z)\ \mathit{of}\ \mathit{PropForm}\ \varphi \Rightarrow \varphi)$

| - ⇒ *Error* $(\lambda x\ y\ z.\ dio)$

3-arity lambda abstraction; $\lambda^2(\lambda xyz.\varphi)$ returns Term $(\lambda xyz.\varphi)$ if φ is a PropForm, otherwise Error; we could, of course, introduce further operators: λ^4, λ^5, etc.

abbreviation *that*::(*e*⇒*io opt*)⇒*e opt* **where** $\varepsilon\ \Phi \equiv$ *case* $(\Phi\ de)$ *of*
  PropForm $\varphi \Rightarrow$ *Term* (*THE x. case* $(\Phi\ x)$ *of PropForm* $\psi \Rightarrow \psi$ *cw*)
  | - ⇒ *Error de*

that operator; that $(\lambda x.\varphi)$ returns Term (*THE x.* φ *x cw*), that is the inbuilt THE operator is used and evaluation is wrt to the current world cw; moreover, application of that is allowed if $(\Phi\ sRE)$ is a PropForm, otherwise Error is passed on for some someRawEntity

4 Further logical connectives

abbreviation *z-and*::*io opt*⇒*io opt*⇒*io opt* **where** $\varphi \wedge \psi \equiv \neg(\varphi \rightarrow \neg\psi)$
abbreviation *z-or*::*io opt*⇒*io opt*⇒*io opt* **where** $\varphi \vee \psi \equiv \neg\varphi \rightarrow \psi$
abbreviation *z-equiv*::*io opt*⇒*io opt*⇒*io opt* **where** $\varphi \equiv \psi \equiv (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$
abbreviation *z-exists*::(*'a*⇒*io opt*)⇒*io opt* **where** $\exists\ \Phi \equiv$ *case* $(\Phi\ da)$ *of*
  PropForm $\varphi \Rightarrow$ *PropForm* $(\lambda w.\ \exists x.$ *case* $(\Phi\ x)$ *of PropForm* $\psi \Rightarrow \psi\ w)$
  | *Form* $\varphi \Rightarrow$ *Form* $(\lambda w.\ \exists x.$ *case* $(\Phi\ x)$ *of Form* $\psi \Rightarrow \psi\ w)$
  | - ⇒ *Error dio*
abbreviation *z-dia*::*io opt*⇒*io opt* **where** $\Diamond\ \varphi \equiv \neg(\Box(\neg\ \varphi))$

5 Some shortcuts for the constructors

abbreviation *mkPropForm* :: *io*⇒*io opt* **where** ,*p*, ≡ *PropForm p*
abbreviation *mkForm* :: *io*⇒*io opt* **where** ;*p*; ≡ *Form p*
abbreviation *mkTerm* :: *'a*⇒*'a opt* **where** .*t*. ≡ *Term t*
abbreviation *mkError* :: *'a*⇒*'a opt* **where** ∗*t*∗ ≡ *Term t*

Three Valued Meta-Logic

datatype *mf* = *tt* | *ff* | *error*

abbreviation *valid* :: *io opt*⇒*mf* **where** $[\varphi] \equiv$ *case* φ *of*
  PropForm $\psi \Rightarrow$ *if* $\forall w.(\psi\ w) \longleftrightarrow$ *True then tt else ff*
  | *Form* $\psi \Rightarrow$ *if* $\forall w.(\psi\ w) \longleftrightarrow$ *True then tt else ff*
  | - ⇒ *error*
abbreviation *satisfiable* :: *io opt*⇒*mf* **where** $[\varphi]^{sat} \equiv$ *case* φ *of*
  PropForm $\psi \Rightarrow$ *if* $\exists w.(\psi\ w) \longleftrightarrow$ *True then tt else ff*
  | *Form* $\psi \Rightarrow$ *if* $\exists w.(\psi\ w) \longleftrightarrow$ *True then tt else ff*
  | - ⇒ *error*
abbreviation *countersatisfiable* :: *io opt*⇒*mf* **where** $[\varphi]^{csat} \equiv$ *case* φ *of*
  PropForm $\psi \Rightarrow$ *if* $\exists w.\neg(\psi\ w) \longleftrightarrow$ *True then tt else ff*
  | *Form* $\psi \Rightarrow$ *if* $\exists w.\neg(\psi\ w) \longleftrightarrow$ *True then tt else ff*
  | - ⇒ *error*
abbreviation *invalid* :: *io opt*⇒*mf* **where** $[\varphi]^{inv} \equiv$ *case* φ *of*
  PropForm $\psi \Rightarrow$ *if* $\forall w.\neg(\psi\ w) \longleftrightarrow$ *True then tt else ff*
  | *Form* $\psi \Rightarrow$ *if* $\forall w.\neg(\psi\ w) \longleftrightarrow$ *True then tt else ff*
  | - ⇒ *error*

6 Some Basic Tests

6.1 Verifying Modal Logic Principles

Necessitation holds

lemma *necessitation-PropForm*: $[,\varphi,] = tt \longrightarrow [\Box\ ,\varphi,] = tt$ **apply** *simp* **done**
lemma *necessitation-Form*: $[;\varphi;] = tt \longrightarrow [\Box\ ;\varphi;] = tt$ **apply** *simp* **done**

Modal Collapse does not hold

lemma *modalCollapse-PropForm*: $[,\varphi, \rightarrow \Box\ ,\varphi,] = tt$ **apply** *simp* **nitpick oops**
lemma *modalCollapse-Form*: $[;\varphi; \rightarrow \Box\ ;\varphi;] = tt$ **apply** *simp* **nitpick oops**

6.2 S5 Principles

lemma *axiom-T-PF*: $[(\Box\ ,\varphi,) \rightarrow ,\varphi,] = tt$ **apply** *simp* **done**
lemma *axiom-T-F*: $[(\Box\ ;\varphi;) \rightarrow ;\varphi;] = tt$ **apply** *simp* **done**

lemma *axiom-B-PF*: $[,\varphi, \rightarrow (\Box\ (\Diamond\ ,\varphi,))] = tt$ **apply** *simp* **done**
lemma *axiom-B-F*: $[;\varphi; \rightarrow (\Box\ (\Diamond\ ;\varphi;))] = tt$ **apply** *simp* **done**

lemma *axiom-D-PF*: $[\Box\ ,\varphi, \rightarrow \Box\ (\Box\ ,\varphi,)] = tt$ **apply** *simp* **done**
lemma *axiom-D-F*: $[\Box\ ;\varphi; \rightarrow \Box\ (\Box\ ;\varphi;)] = tt$ **apply** *simp* **done**

lemma *axiom-4-PF*: $[\Box\ ,\varphi, \rightarrow \Diamond\ ,\varphi,] = tt$ **apply** *simp* **by** *auto*
lemma *axiom-4-F*: $[\Box\ ;\varphi; \rightarrow \Diamond\ ;\varphi;] = tt$ **apply** *simp* **by** *auto*

lemma *axiom-5-PF*: $[\Diamond\ ,\varphi, \rightarrow \Box\ (\Diamond\ ,\varphi,)] = tt$ **apply** *simp* **done**
lemma *axiom-5-F*: $[\Diamond\ ;\varphi; \rightarrow \Box\ (\Diamond\ ;\varphi;)] = tt$ **apply** *simp* **done**

lemma *test-A-PF*: $[\Box\ (\Diamond\ ,\varphi,) \rightarrow \Diamond\ ,\varphi,] = tt$ **apply** *simp* **done**
lemma *test-A-F*: $[\Box\ (\Diamond\ ;\varphi;) \rightarrow \Diamond\ ;\varphi;] = tt$ **apply** *simp* **done**

lemma *test-B-PF*: $[\Diamond\ (\Box\ ,\varphi,) \rightarrow \Diamond\ ,\varphi,] = tt$ **apply** *simp* **by** *auto*
lemma *test-B-F*: $[\Diamond\ (\Box\ ;\varphi;) \rightarrow \Diamond\ ;\varphi;] = tt$ **apply** *simp* **by** *auto*

lemma *test-C-PF*: $[\Box\ (\Diamond\ ,\varphi,) \rightarrow \Box\ ,\varphi,] = tt$ **apply** *simp* **nitpick oops**
lemma *test-C-F*: $[\Box\ (\Diamond\ ;\varphi;) \rightarrow \Box\ ;\varphi;] = tt$ **apply** *simp* **nitpick oops**

lemma *test-D-PF*: $[\Diamond\ (\Box\ ,\varphi,) \rightarrow \Box\ ,\varphi,] = tt$ **apply** *simp* **done**
lemma *test-D-F*: $[\Diamond\ (\Box\ ;\varphi;) \rightarrow \Box\ ;\varphi;] = tt$ **apply** *simp* **done**

6.3 Validity, Satisfiabilty, Countersatisfiability and Invalidity

lemma $[,\varphi,] = tt \longleftrightarrow [,\varphi,]^{csat} = ff$ **apply** *simp* **done**
lemma $[,\varphi,]^{sat} = tt \longleftrightarrow [,\varphi,]^{inv} = ff$ **apply** *simp* **done**
lemma $[;\varphi;] = tt \longleftrightarrow [;\varphi;]^{csat} = ff$ **apply** *simp* **done**
lemma $[;\varphi;]^{sat} = tt \longleftrightarrow [;\varphi;]^{inv} = ff$ **apply** *simp* **done**

For Terms and Error we have

lemma $[.\varphi.] = error$ **apply** *simp* **done**
lemma $[.\varphi.]^{sat} = error$ **apply** *simp* **done**

lemma $[.\varphi.]^{csat} = \mathit{error}$ **apply** *simp* **done**
lemma $[.\varphi.]^{inv} = \mathit{error}$ **apply** *simp* **done**
lemma $[*\varphi*] = \mathit{error}$ **apply** *simp* **done**
lemma $[*\varphi*]^{sat} = \mathit{error}$ **apply** *simp* **done**
lemma $[*\varphi*]^{csat} = \mathit{error}$ **apply** *simp* **done**
lemma $[*\varphi*]^{inv} = \mathit{error}$ **apply** *simp* **done**

6.4 Example signature; entities and relations

consts $\mathit{a\text{-}0} :: e$ **abbreviation** a **where** $a \equiv .\mathit{a\text{-}0}.$
consts $\mathit{b\text{-}0} :: e$ **abbreviation** b **where** $b \equiv .\mathit{b\text{-}0}.$
consts $\mathit{c\text{-}0} :: e$ **abbreviation** c **where** $c \equiv .\mathit{c\text{-}0}.$

consts $\mathit{R\text{-}0} :: \mathit{io}$ **abbreviation** $\mathit{R0}$ **where** $\mathit{R0} \equiv .\mathit{R\text{-}0}.$
consts $\mathit{R\text{-}1} :: e \Rightarrow \mathit{io}$ **abbreviation** $\mathit{R1}$ **where** $\mathit{R1} \equiv .\mathit{R\text{-}1}.$
consts $\mathit{R\text{-}2} :: e \Rightarrow e \Rightarrow \mathit{io}$ **abbreviation** $\mathit{R2}$ **where** $\mathit{R2} \equiv .\mathit{R\text{-}2}.$
consts $\mathit{R\text{-}3} :: e \Rightarrow e \Rightarrow e \Rightarrow \mathit{io}$ **abbreviation** $\mathit{R3}$ **where** $\mathit{R3} \equiv .\mathit{R\text{-}3}.$

Testing term and formula constructions

lemma $[<\mathit{R1} \bullet a>] = \mathit{tt}$ **apply** *simp* **nitpick** **oops**
lemma $<\mathit{R1} \bullet a> = X$ **apply** *simp* **oops**

lemma $[<a \circ \mathit{R1}>] = \mathit{tt}$ **nitpick** **oops**
lemma $<a \circ \mathit{R1}> = X$ **apply** *simp* **oops**

lemma $[<\lambda^1(\lambda x.\ <\mathit{R1} \bullet .x.> \to <\mathit{R1} \bullet .x.>) \bullet a>] = \mathit{tt}$ **apply** *simp* **done**
lemma $<\lambda^1(\lambda x.\ <\mathit{R1} \bullet .x.> \to <\mathit{R1} \bullet .x.>) \bullet a> = X$ **apply** *simp* **oops**

lemma $\lambda^1(\lambda x.\ <\mathit{R1} \bullet .x.> \to <.x. \circ \mathit{R1}>) = X$ **apply** *simp* **oops**

lemma $[<\lambda^1(\lambda x.\ <\mathit{R1} \bullet .x.> \to <.x. \circ \mathit{R1}>) \bullet a>] = \mathit{error}$ **apply** *simp* **done**
lemma $[<\lambda^1(\lambda x.\ <\mathit{R1} \bullet .x.> \to <.x. \circ \mathit{R1}>) \bullet a>] = X$ **apply** *simp* **oops**
lemma $<\lambda^1(\lambda x.\ <\mathit{R1} \bullet .x.> \to <.x. \circ \mathit{R1}>) \bullet a> = X$ **apply** *simp* **oops**

lemma $[\forall (\lambda x.\ <\mathit{R1} \bullet .x.> \to <\mathit{R1} \bullet .x.>)] = \mathit{tt}$ **apply** *simp* **done**
lemma $[\forall (\lambda R.\ \forall (\lambda x.\ <.R. \bullet .x.> \to <.R. \bullet .x.>))] = \mathit{tt}$ **apply** *simp* **done**
lemma $\forall (\lambda x.\ <\mathit{R1} \bullet .x.> \to <\mathit{R1} \bullet .x.>) = X$ **apply** *simp* **oops**

lemma $[\forall (\lambda x.\ <.x. \circ \mathit{R1}> \to <.x. \circ \mathit{R1}>)] = \mathit{tt}$ **apply** *simp* **done**
lemma $\forall (\lambda x.\ <.x. \circ \mathit{R1}> \to <.x. \circ \mathit{R1}>) = X$ **apply** *simp* **oops**

lemma $[\forall (\lambda x.\ <\mathit{R1} \bullet .x.> \to <.x. \circ \mathit{R1}>)] = \mathit{error}$ **apply** *simp* **done**
lemma $[\forall (\lambda x.\ <\mathit{R1} \bullet .x.> \to <.x. \circ \mathit{R1}>)] = X$ **apply** *simp* **oops**
lemma $\forall (\lambda x.\ <\mathit{R1} \bullet .x.> \to <.x. \circ \mathit{R1}>) = X$ **apply** *simp* **oops**
lemma $[\forall (\lambda R.\ <.R. \bullet .x.> \to <.x. \circ .R.>)] = \mathit{error}$ **apply** *simp* **done**
lemma $\forall (\lambda R.\ <.R. \bullet .x.> \to <.x. \circ .R.>) = X$ **apply** *simp* **oops**

7 Are the priorities set correctly?

lemma $,\varphi, \wedge ,\psi, \to ,\chi, \equiv (,\varphi, \wedge ,\psi,) \to ,\chi,$ **apply** *simp* **done**
lemma $,\varphi, \wedge ,\psi, \to ,\chi, \equiv ,\varphi, \wedge (,\psi, \to ,\chi,)$ **apply** *simp* **nitpick** **oops**

lemma $(,\varphi, \wedge ,\psi, \equiv ,\varphi, \wedge ,\psi,) \equiv ((,\varphi, \wedge ,\psi,) \equiv (,\varphi, \wedge ,\psi,))$ **apply** *simp* **done**
lemma $(,\varphi, \wedge ,\psi, \equiv ,\varphi, \wedge ,\psi,) \equiv (,\varphi, \wedge (,\psi, \equiv ,\varphi,) \wedge ,\psi,)$ **apply** *simp* **nitpick oops**

8 E!, O!, A! and =E

consts $E::(e \Rightarrow io)$

Distinguished 1-place relation constant: E! (read: being concrete or concreteness)

abbreviation *z-ordinary*$::(e \Rightarrow io)$ *opt* **where** $O^! \equiv \lambda^1(\lambda x.\ \Diamond <.E.\cdot.x.>)$

Being ordinary is being possibly concrete.

abbreviation *z-abstract*$::(e \Rightarrow io)$ *opt* **where** $A^! \equiv \lambda^1(\lambda x.\ \neg\ (\Diamond <.E.\cdot.x.>))$

Being abstract is not possibly being concrete.

abbreviation *z-identity*$::(e \Rightarrow e \Rightarrow io)$ *opt* **where** $=_e\ \equiv$
$\lambda^2(\lambda x\ y.\ ((<O^!\cdot.x.> \wedge <O^!\cdot.y.>) \wedge \Box\ (\forall\,(\lambda F.\ <.F.\cdot.x.> \equiv <.F.\cdot.y.>))))$

abbreviation *z-identityE*$::(e\ opt \Rightarrow e\ opt \Rightarrow io\ opt)$ **where** $x =_E y \equiv (Exe2 =_e x\ y)$

9 Further test examples

lemma $[\forall\,(\lambda x.\ \exists\,(\lambda R.\ (<.x.\circ.R.> \rightarrow <.x.\circ R1>)))] = tt$ **apply** *simp* **by** *auto*
lemma $[\forall\,(\lambda x.\ \forall\,(\lambda R.\ (<.x.\circ.R.> \rightarrow <.x.\circ R1>)))] = tt$ **apply** *simp* **nitpick oops**

lemma $[a =_E a] = tt$ **apply** *simp* **nitpick oops**

lemma $[<O^!\cdot a> \rightarrow a =_E a] = tt$ **apply** *simp* **done**

lemma $[(\forall\,(\lambda F.\ <.F.\cdot.x.> \equiv <.F.\cdot.x.>))] = tt$ **apply** *simp* **done**
lemma $[<O^!\cdot a> \rightarrow <\lambda^1(\lambda x.\ .x. =_E a)\cdot a>] = tt$ **apply** *simp* **done**

lemma $[(\exists\,(\lambda F.\ <a\circ.F.>))] = tt$ **apply** *simp* **by** *auto*

lemma $[\exists\,(\lambda\varphi.\ ,\varphi,)] = tt$ **apply** *simp* **by** *auto*
lemma $[\exists\,(\lambda\varphi.\ ;\varphi;)] = tt$ **apply** *simp* **by** *auto*

10 Axioms

10.1 Axioms for Negations and Conditionals

lemma *a21-1-PF*: $[,\varphi, \rightarrow (,\varphi, \rightarrow ,\varphi,)] = tt$ **apply** *simp* **done**
lemma *a21-1-F*: $[;\varphi; \rightarrow (;\varphi; \rightarrow ;\varphi;)] = tt$ **apply** *simp* **done**
lemma *a21-2-PF*: $[(,\varphi, \rightarrow (,\psi, \rightarrow ,\chi,)) \rightarrow ((,\varphi, \rightarrow ,\psi,) \rightarrow (,\varphi, \rightarrow ,\chi,))] = tt$ **apply** *simp* **done**
lemma *a21-2-F*: $[(;\varphi; \rightarrow (;\psi; \rightarrow ;\chi;)) \rightarrow ((;\varphi; \rightarrow ;\psi;) \rightarrow (;\varphi; \rightarrow ;\chi;))] = tt$ **apply** *simp* **done**
lemma *a21-3-PF*: $[(\neg\ ,\varphi, \rightarrow \neg\ ,\psi,) \rightarrow (\neg\ ,\varphi, \rightarrow ,\psi,) \rightarrow ,\varphi,] = tt$ **apply** *simp* **done**
lemma *a21-3-F*: $[(\neg\ ;\varphi; \rightarrow \neg\ ;\psi;) \rightarrow (\neg\ ;\varphi; \rightarrow ;\psi;) \rightarrow ;\varphi;] = tt$ **apply** *simp* **done**

10.2 Axioms of Identity

todo

10.3 Axioms of Quantification

todo

10.4 Axioms of Actuality

lemma *a31-1-PF*: $[\mathcal{A}\ (\neg\ ,\varphi,) \equiv (\neg\ (\mathcal{A}\ ,\varphi,))] = tt$ **apply** *simp* **done**
lemma *a31-1-F*: $[\mathcal{A}\ (\neg\ ;\varphi;) \equiv (\neg\ (\mathcal{A}\ ;\varphi;))] = tt$ **apply** *simp* **done**
lemma *a31-2-PF*: $[\mathcal{A}\ (,\varphi, \rightarrow ,\psi,) \equiv (\mathcal{A}\ ,\varphi, \rightarrow \mathcal{A}\ ,\psi,)] = tt$ **apply** *simp* **done**
lemma *a31-2-F*: $[\mathcal{A}\ (;\varphi; \rightarrow ;\psi;) \equiv (\mathcal{A}\ ;\varphi; \rightarrow \mathcal{A}\ ;\psi;)] = tt$ **apply** *simp* **done**
lemma *a31-3-PF*: $[(\mathcal{A}\ (\forall\, (\lambda x.\ ,\varphi,)) \equiv \forall\, (\lambda x.\ \mathcal{A}\ ,\varphi,))] = tt$ **apply** *simp* **done**
lemma *a31-3-F*: $[(\mathcal{A}\ (\forall\, (\lambda x.\ ;\varphi;)) \equiv \forall\, (\lambda x.\ \mathcal{A}\ ;\varphi;))] = tt$ **apply** *simp* **done**
lemma *a31-4-PF*: $[\mathcal{A}\ ,\varphi, \equiv \mathcal{A}\ (\mathcal{A}\ ,\varphi,)] = tt$ **apply** *simp* **done**
lemma *a31-4-F*: $[\mathcal{A}\ ;\varphi; \equiv \mathcal{A}\ (\mathcal{A}\ ;\varphi;)] = tt$ **apply** *simp* **done**

10.5 Axioms of Necessity

lemma *a32-1-PF*: $[\Box\ (,\varphi, \rightarrow ,\varphi,) \rightarrow (\Box\ ,\varphi, \rightarrow \Box\ ,\varphi,)] = tt$ **apply** *simp* **done**
lemma *a32-1-F*: $[\Box\ (;\varphi; \rightarrow ;\varphi;) \rightarrow (\Box\ ;\varphi; \rightarrow \Box\ ;\varphi;)] = tt$ **apply** *simp* **done**
lemma *a32-2-PF*: $[\Box\ ,\varphi, \rightarrow ,\varphi,] = tt$ **apply** *simp* **done**
lemma *a32-2-F*: $[\Box\ ;\varphi; \rightarrow ;\varphi;] = tt$ **apply** *simp* **done**
lemma *a32-3-PF*: $[\Box\ (\Diamond\ ,\varphi,) \rightarrow (\Diamond\ ,\varphi,)] = tt$ **apply** *simp* **done**
lemma *a32-3-F*: $[\Box\ (\Diamond\ ;\varphi;) \rightarrow (\Diamond\ ;\varphi;)] = tt$ **apply** *simp* **done**
lemma *a32-4-PF*: $[(\forall\, (\lambda x.\ \Box\ ,\varphi,)) \rightarrow \Box\ (\forall\, (\lambda x.\ ,\varphi,))] = tt$ **apply** *simp* **done**
lemma *a32-4-F*: $[(\forall\, (\lambda x.\ \Box\ ;\varphi;)) \rightarrow \Box\ (\forall\, (\lambda x.\ ;\varphi;))] = tt$ **apply** *simp* **done**

The following needs to be an axiom; it does not follow for free: it is possible that there are contingently concrete individuals and it is possible that there are not:

axiomatization where
a32-5-PF: $[\Diamond\ (\exists\, (\lambda x.\ <.E.\cdot.x.> \wedge (\Diamond\ (\neg\ <.E.\cdot.x.>)))) \wedge \Diamond\ (\neg\ (\exists\, (\lambda x.\ <.E.\cdot.x.> \wedge (\Diamond\ (\neg\ <.E.\cdot.x.>)))))] = tt$

10.6 Axioms of Necessity and Actuality

lemma *a33-1-PF*: $[\mathcal{A}\ ,\varphi, \rightarrow \Box\ (\mathcal{A}\ ,\varphi,)] = tt$ **apply** *simp* **done**
lemma *a33-1-F*: $[\mathcal{A}\ ;\varphi; \rightarrow \Box\ (\mathcal{A}\ ;\varphi;)] = tt$ **apply** *simp* **done**
lemma *a33-2-PF*: $[\Box\ ,\varphi, \equiv (\mathcal{A}\ (\Box\ ,\varphi,))] = tt$ **apply** *simp* **done**
lemma *a33-2-F*: $[\Box\ ;\varphi; \equiv (\mathcal{A}\ (\Box\ ;\varphi;))] = tt$ **apply** *simp* **done**

References

[1] C. Benzmüller and B. W. Paleo. Automating Gödel's ontological proof of God's existence with higher-order automated theorem provers. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 93 – 98. IOS Press, 2014.

[2] C. Benzmüller and L. Paulson. Quantified multimodal logics in simple type theory. *Logica Universalis (Special Issue on Multimodal Logics)*, 7(1):7–20, 2013.

[3] C. Benzmüller and B. Woltzenlogel Paleo. Automating Gödel's ontological proof of God's existence with higher-order automated theorem provers. In T. Schaub, G. Friedrich, and B. O'Sullivan, editors, *ECAI 2014*, volume 263 of *Frontiers in Artificial Intelligence and Applications*, pages 93 – 98. IOS Press, 2014.

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

[5] T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*. Number 2283 in LNCS. Springer, 2002.

[6] E. N. Zalta. Principia metaphysica, a compilation of the theorems of the theory of abstract objects. Available at `https://mally.stanford.edu/publications.html`.

[7] E. N. Zalta and P. E. Oppenheimer. Relations versus functions at the foundations of logic: Type-theoretic considerations. *Journal of Logic and Computation*, (21):351374, 2011.