Modal Relational Type Theory in Isabelle/HOL

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

May 28, 2016

1 Introduction

The Isabelle/HOL formalisation presented in this article is related to the ongoing Principia Metaphysica [6] project at Stanford University. This project, which exploits and extends Zalta's Theory of Abstract Objects [7], employs a modal relational type theory (MRTT) as logical foundation. Arguments defending this choice against a modal functional type theory (MFTT) have been presented before [8]. 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 extending Church's theory of types [5], and 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 MFTT [1, 3]. This work has subsequently been employed in a case study in computational metaphysics, in which different variants of Kurt Gdel's ontological argument were verified resp. falsified [3, 4].

The motivating research questions for the formalisation presented below include:

- Can functional type theory, despite the problems pointed at by Zalta and Oppenheimer, also be utilized to encode MRTT and subsequently the Theory of Abstract Objects when following the embeddings approach?
- How elegant and user-friendly is the resulting formalization? 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 to minimise user interaction in the formalization of the Theory of Abstract objects?
- Can the consistency of the theory eventually be validated with the available automated reasoning tools?
- Can the reasoners eventually even contribute some new knowledge?
- Are any suggestions for improvements in Isabelle arising? What are the particular problems detected in the course of the study?

$$
\begin{array}{rrcl}
 & \delta & ::= & a_1, a_2, \ldots \\
 & \nu & ::= & x_1, x_2, \ldots \\
(n \geq 0) & \Sigma^n & ::= & P_1^n, P_2^n, \ldots \\
(n \geq 0) & \Omega^n & ::= & F_1^n, F_2^n, \ldots \\
 & \alpha & ::= & \nu \mid \Omega^n \ (n \geq 0) \\
 & \kappa & ::= & \delta \mid \nu \mid \iota\nu\varphi \\
(n \geq 1) & \Pi^n & ::= & \Sigma^n \mid \Omega^n \mid [\lambda \nu_1 \ldots \nu_n \ \varphi^*] \\
 & \Pi^0 & ::= & \Sigma^0 \mid \Omega^0 \mid [\lambda \ \varphi^*] \mid \varphi^* \\
 & \varphi^* & ::= & \Pi^n \kappa_1 \ldots \kappa_n \ (n \geq 1) \mid \Pi^0 \mid (\neg\varphi^*) \mid (\varphi^* \rightarrow \varphi^*) \mid \forall\alpha\varphi^* \mid \\
 & & & (\Box\varphi^*) \mid (\mathcal{A}\varphi^*) \\
 & \varphi & ::= & \kappa_1\Pi^1 \mid \varphi^* \mid (\neg\varphi) \mid (\varphi \rightarrow \varphi) \mid \forall\alpha\varphi \mid (\Box\varphi) \mid (\mathcal{A}\varphi) \\
 & \tau & ::= & \kappa \mid \Pi^n \ (n \geq 0)
\end{array}
$$

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 MRTT, cf. [6] for further details in MRTT. Two kinds of (complex) formulas are introduced: ones that may have encoding subformulas and ones that do not. The latter are designated as propositional formulas, the former ones simply as formulas.

In this contribution to the Archive of Formal Proofs we focus on the encoding of MRTT in functional type theory. The idea is to reuse and adapt the fundamental ideas underlying the previous encoding of MFTT in functional type theory [1, 3], which has been realised inter alia in Isabelle/HOL [2]. In subsequent work we will then reuse and extend the foundations provided in this article.

The encoding of modal functional type theory in functional type theory as explored in previous work is simple: modal logic formulas are identified with certain functional type theory formulas of predicate type $i \Rightarrow bool$ (abbreviated as io below). Possible worlds are explicitly represented by terms of type i. A modal logic formula φ 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 for this work has been to suitably appropriately 'restrict' this embedding for modal relational type theory. The grammar of modal relational type theory is presented in Figure 1. Note that this grammar excludes terms such as $\lambda x.Rx \rightarrow xR$, where Rx represents the exemplification of property R by x and xR stands for the encoding of property R by x. The reason is that such kind of lambda-abstractions may lead to paradoxes in the theory of abstract objects [8].

To achieve our goal we provide means to explicitly represents and propagate information on the syntactical structure of MRTT in functional type theory. In particular, we provide means in form of annotations to explicitly distinguish between propositional formulas, formulas, terms and erroneous (ineligible/excluded) formations. Respective annotation information is propagated from the innermost constituents to the top level constructions. This clearly creates some significant technical overhead. However, we fruitfully exploit facilities in Isabelle/HOL's user interface, and other means, to hide most of these technicalities from the user in applications.

A note on using abbreviations versus definitions in our approach: We are aware that abbreviations should be used sparsingly in Isabelle/HOL, since they are automatically expanded and thus lead to a discrepancy between the internal and the external view of a term. However, here we deliberately deviate from this rule, since one aspect of the paper is to exactly illustrate this discrepancy and to emphasize the complexity of the embedding MRTT in functional type theory. In fact, this complexity makes pen and paper work with the proposed embedding pragmatically infeasible, as we believe, while in a proof assistant like Isabelle/HOL in can, at least to some degree, still be handled by an interactive user. Moreover, as we will also illustrate, the simplifier *simp* of Isabelle/HOL is well capable of effectively reducing this complexity again. In other words, the simplifier effectievely analyses and and rewrites the deeply annotated terms and propagates the annotation information to the top-level only as intended. It is exactly this effect which we want to illustrate and exploit here.[1]

2 Preliminaries

We start out with some type declarations and type abbreviations. Our formalism explicitly encodes possible world 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 more concise in the remainder 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. The Theory of Abstract Objects later introduces means to distinguish between abstract and ordinary entities.

typedecl e

To explicitly model the syntactical restrictions of MRTT we introduce a (polymorphic) datatype $'a\ opt$ ($'a$ is a type variable) based on four constructors: $ERR\ 'a$ (identifies ineligible/excluded constructions), $P\ 'a$ (identifies propositional formulas), $F\ 'a$ (identifies formulas), and $T\ 'a$ (identifies eligible terms, such as lambda abstractions). The embeddings approach (of MFTT in functional type theory) will be suitably adapted below so that for each language expression (in the embedded MRTT) the respective datatype is identified and appropriately propagated. The encapsulated expressions then correspond to the previous embedding of MRTT in functional type theory.

datatype $'a\ opt = ERR\ 'a \mid P\ 'a \mid F\ 'a \mid T\ 'a$

The following operators support a concise and elegant superscript annotation with these four syntactical categories for our language constructs.

abbreviation $mkP{::}io{\Rightarrow}io\ opt\ ({\text -}^P\ [109]\ 110)$ **where** $\varphi^P \equiv P\ \varphi$
abbreviation $mkF{::}io{\Rightarrow}io\ opt\ ({\text -}^F\ [109]\ 110)$ **where** $\varphi^F \equiv F\ \varphi$
abbreviation $mkT{::}'a{\Rightarrow}'a\ opt\ ({\text -}^T\ [109]\ 110)$ **where** $\varphi^T \equiv T\ \varphi$

[1] We have also experimented with the alternative use of definitions; the respective encodings can be requested from the authors.

abbreviation $mkE::'a \Rightarrow 'a\ opt$ $(\text{-}^E\ [109]\ 110)$ **where** $\varphi^E \equiv ERR\ \varphi$

Certain language constructs in the Theory of Abstract objects, such as the actuality operator $\boldsymbol{\mathcal{A}}$ ("it is actually the case that"), refer to a (fixed) designated world. To model such a rigid dependence we introduce a constant symbol (name) dw of world type i. Moreover, for technical reasons, which will be clarified below, we introduce further (dummy) constant symbols for the various other domains. Since we anyway assume that all domains are non-empty, introducing these constant symbols is obviously not harmful. [2]

consts $dw :: i$
consts $de::e\ dio::io\ deio::e \Rightarrow io\ da::'a$

3 Embedding of Modal Relational Type Theory

The various language constructs of MRTT (see Figure 1) are now introduced step by step.

The actuality operator $\boldsymbol{\mathcal{A}}$, when being applied to a formula or propositional formula φ, evaluates φ wrt the fixed given world dw. The compound expression $\boldsymbol{\mathcal{A}}\varphi$ inherits its syntactical category F (formula) or P (propositional formula from φ. If the syntactical catagory of φ is ERR (error) or T (term), then the syntactical catagory of $\boldsymbol{\mathcal{A}}\varphi$ is ERR and a dummy entity of appropriate type is returned. This illustrates the core ideas of our explicit modeling of MRTT grammatical structure in functional type theory. This scheme will repeated below for all the other language constructs of MRTT.

abbreviation $Actual::io\ opt \Rightarrow io\ opt$ $(\boldsymbol{\mathcal{A}}\ \text{-}\ [64]\ 65)$ **where** $\boldsymbol{\mathcal{A}}\varphi \equiv case\ \varphi\ of$
$F(\psi) \Rightarrow F(\lambda w.\ \psi\ dw) \mid P(\psi) \Rightarrow P(\lambda w.\ \psi\ dw) \mid \text{-} \Rightarrow ERR(dio)$

The Theory of Abstract Objects distinguishes between encoding properties $\kappa_1\Pi^1$ and exemplifying properties $\Pi^n,\kappa_1,..,\kappa_n$ (for $n \geq 1$).
Encoding $\kappa_1\Pi^1$ is noted below as $\{\!|\kappa_1,\Pi^1|\!\}$. Encoding yields formulas and never propositional formulas. It is mapped to predicate application.

abbreviation $Enc::e\ opt \Rightarrow (e \Rightarrow io)\ opt \Rightarrow io\ opt$ $(\{\!|\text{-},\text{-}|\!\})$ **where** $\{\!|x,\Phi|\!\} \equiv case\ (x,\Phi)\ of$
$(T(y),T(Q)) \Rightarrow F(Q\ y) \mid \text{-} \Rightarrow ERR(dio)$

Unary exemplifying formulas $\Pi^1\kappa_1$ are noted below as $(\!|\Pi^1,\kappa_1|\!)$. Exemplification yields propositional formulas. Like encoding, it is then mapped to predicate application.

abbreviation $Exe1::(e \Rightarrow io)\ opt \Rightarrow e\ opt \Rightarrow io\ opt$ $((\!|\text{-},\text{-}|\!))$ **where** $(\!|\Phi,x|\!) \equiv case\ (\Phi,x)\ of$
$(T(Q),T(y)) \Rightarrow P(Q\ y) \mid \text{-} \Rightarrow ERR(dio)$

For pragmatical reasons we support exemplification formulas $\Pi^n,\kappa_1,..,\kappa_n$ here only for $1 \leq n \leq 3$. In addition to the unary case above, we thus introduce two further cases.

abbreviation $Exe2::(e \Rightarrow e \Rightarrow io)\ opt \Rightarrow e\ opt \Rightarrow e\ opt \Rightarrow io\ opt$ $((\!|\text{-},\text{-},\text{-}|\!))$
where $(\!|\Phi,x1,x2|\!) \equiv case\ (\Phi,x1,x2)\ of$
$(T(Q),T(y1),T(y2)) \Rightarrow P(Q\ y1\ y2) \mid \text{-} \Rightarrow ERR(dio)$
abbreviation $Exe3::(e \Rightarrow e \Rightarrow e \Rightarrow io)\ opt \Rightarrow e\ opt \Rightarrow e\ opt \Rightarrow e\ opt \Rightarrow io\ opt$ $((\!|\text{-},\text{-},\text{-},\text{-}|\!))$
where $(\!|\Phi,x1,x2,x3|\!) \equiv case\ (\Phi,x1,x2,x3)\ of$
$(T(Q),T(y1),T(y2),T(y3)) \Rightarrow P(Q\ y1\ y2\ y3) \mid \text{-} \Rightarrow ERR(dio)$

[2] The single polymorphic dummy $\mathbf{d}a::'a$, utilized e.g. in the definition of the universal quantifier of MRTT below, would actually cover all cases. However, to avoid type inference we actually prefer non-polymorphic dummy elements in all those cases where we can statically predetermine the required type.

Formations with negation and implication are supported for both, formulas and propositional formulas, and their embeddings are straightforward. In the case of implication, the compound formula is a propositional formula if both constituents are propositional formulas. If at least one constituent is a formula and the other one eligible, then the compound formula is a formula. In all other cases an ERR-Formula is returned.

abbreviation *not*::*io opt*⇒*io opt* (¬ - [*58*] *59*) **where** $\neg\ \varphi \equiv$ *case* φ *of*
$F(\psi) \Rightarrow F(\lambda w.\neg(\psi\ w))$ | $P(\psi) \Rightarrow P(\lambda w.\neg(\psi\ w))$ | - ⇒ *ERR*(*dio*)
abbreviation *implies*::*io opt*⇒*io opt*⇒*io opt* (**infixl** → *51*) **where** $\varphi \to \psi \equiv$ *case* (φ,ψ) *of*
$(P(\alpha),P(\beta)) \Rightarrow P(\lambda w.\ \alpha\ w \longrightarrow \beta\ w)$ | $(F(\alpha),F(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \longrightarrow \beta\ w)$ |
$(P(\alpha),F(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \longrightarrow \beta\ w)$ | $(F(\alpha),P(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \longrightarrow \beta\ w)$ |
- ⇒ *ERR*(*dio*)

Also universal quantification $\forall\,(\lambda x.\varphi)$ (first-order and higher-order) is supported for both, formulas and propositional formulas. Following previous work, the embedding maps $\forall\,(\lambda x.\varphi)$ to $(\lambda w.\forall\,x.\varphi w)$, where the latter ∀ is the universal quantifier from the HOL meta-logic. Note that ∀ is introduced as logical connective based on the existing λ-binder. To improve the presentation and intuitive use in the remainder we additionally introduce binder notation $\forall\,x.\varphi$ as syntactic sugar for $\forall\,(\lambda x.\varphi)$.

abbreviation *forall*::(*'a*⇒*io opt*)⇒*io opt* (∀) **where** $\forall\,\Phi \equiv$ *case* $(\Phi\ da)$ *of*
$F(\text{-}) \Rightarrow F(\lambda w.\forall\,x.$ *case* $(\Phi\ x)$ *of* $F(\psi) \Rightarrow \psi\ w)$
| $P(\text{-}) \Rightarrow P(\lambda w.\forall\,x.$ *case* $(\Phi\ x)$ *of* $P(\psi) \Rightarrow \psi\ w)$ | - ⇒ *ERR*(*dio*)
abbreviation *forallBinder*::(*'a*⇒*io opt*)⇒*io opt* (**binder** ∀ [*8*] *9*) **where** $\forall\,x.\ \varphi\ x \equiv \forall\,\varphi$

The modal □-operator is introduced here for logic S5. Since in an equivalence class of possible worlds each world is reachable from any other world, the guarding accessibility clause in the usual definition of the □-operator can be omitted. This is convenient and also improves the efficience of theorem provers, cf. [4]. In Section 6.3 we will actually demonstrate that the expected S5 properties are validated by our modeling of □. The □-operator can be applied to formulas and propositional formulas.

abbreviation *box*::*io opt*⇒*io opt* (□- [*62*] *63*) **where** $\Box\varphi \equiv$ *case* φ *of*
$F(\psi) \Rightarrow F(\lambda w.\forall\,v.\ \psi\ v)$ | $P(\psi) \Rightarrow P(\lambda w.\forall\,v.\ \psi\ v)$ | - ⇒ *ERR*(*dio*)

n-ary lambda abstraction $\boldsymbol{\lambda}^0$,$\boldsymbol{\lambda}$,$\boldsymbol{\lambda}^2$,$\boldsymbol{\lambda}^3$,..., for $n \geq 0$, is supported in the Theory of Abstract Objects only for propositional formulas. This way constructs such as beforehand mentioned $(\lambda x.Rx \to xR)$ (noted here as $(\boldsymbol{\lambda}x.\ (\!|R^T,x^T|\!) \to \{\!|x^T,R^T|\!\})$ are excluded, respectively identified as *ERR*-annotated terms in our framework. Their embedding is straightforward: $\boldsymbol{\lambda}^0$ is mapped to identity and $\boldsymbol{\lambda}$,$\boldsymbol{\lambda}^2$,$\boldsymbol{\lambda}^3$,... are mapped to n-ary lambda abstractions, that is, $\boldsymbol{\lambda}(\lambda x.\varphi)$ is mapped to $(\lambda x.\varphi)$ and $\boldsymbol{\lambda}^2(\lambda xy.\varphi)$ to $(\lambda xy.\varphi)$, etc. Similar to before, we support only the cases for $n \leq 3$. Binder notation is introduced for $\boldsymbol{\lambda}$.[3]

abbreviation *lam0*::*io opt*⇒*io opt* ($\boldsymbol{\lambda}^0$) **where** $\boldsymbol{\lambda}^0\varphi \equiv$ *case* φ *of*
$P(\psi) \Rightarrow P(\psi)$ | - ⇒ *ERR dio*
abbreviation *lam*::(*e*⇒*io opt*)⇒(*e*⇒*io*) *opt* ($\boldsymbol{\lambda}$) **where** $\boldsymbol{\lambda}\Phi \equiv$ *case* $(\Phi\ de)$ *of*
$P(\text{-}) \Rightarrow T(\lambda x.$ *case* $(\Phi\ x)$ *of* $P(\varphi) \Rightarrow \varphi)$ | - ⇒ $ERR(\lambda x.\ dio)$
abbreviation *lamBinder*::(*e*⇒*io opt*)⇒(*e*⇒*io*) *opt* (**binder** $\boldsymbol{\lambda}$ [*8*] *9*) **where** $\boldsymbol{\lambda}x.\ \varphi\ x \equiv \boldsymbol{\lambda}\ \varphi$
abbreviation *lam2*::(*e*⇒*e*⇒*io opt*)⇒(*e*⇒*e*⇒*io*) *opt* ($\boldsymbol{\lambda}^2$) **where** $\boldsymbol{\lambda}^2\Phi \equiv$ *case* $(\Phi\ de\ de)$ *of*
$P(\text{-}) \Rightarrow T(\lambda x\ y.$ *case* $(\Phi\ x\ y)$ *of* $P(\varphi) \Rightarrow \varphi)$ | - ⇒ $ERR(\lambda x\ y.\ dio)$
abbreviation *lam3*::(*e*⇒*e*⇒*e*⇒*io opt*)⇒(*e*⇒*e*⇒*e*⇒*io*) *opt* ($\boldsymbol{\lambda}^3$) **where** $\boldsymbol{\lambda}^3\Phi \equiv$ *case* $(\Phi\ de\ de\ de)$ *of*

[3]Unfortunately, we could not find out how binder notation could be analogously provided for $\boldsymbol{\lambda}^2$ and $\boldsymbol{\lambda}^3$

$P(\text{-}) \Rightarrow T(\lambda x\ y\ z.\ case\ (\Phi\ x\ y\ z)\ of\ P(\varphi) \Rightarrow \varphi) \mid \text{-} \Rightarrow ERR(\lambda x\ y\ z.\ dio)$

The Theory of Abstract Objects supports rigid definite descriptions. Our definition maps $\boldsymbol{\iota}(\lambda x.\varphi)$ to $(THE\ x.\ \varphi\ dw)$, that is, Isabelle's inbuilt definite description operator *THE* is utilized and evaluation is rigidly carried out with respect to the current world denoted by *dw*. We again introduce binder notation for $\boldsymbol{\iota}$.

abbreviation $that{::}(e{\Rightarrow}io\ opt){\Rightarrow}e\ opt\ (\boldsymbol{\iota})$ **where** $\boldsymbol{\iota}\Phi \equiv case\ (\Phi\ de)\ of$
$F(\text{-}) \Rightarrow T(THE\ x.\ case\ (\Phi\ x)\ of\ F\ \psi \Rightarrow \psi\ dw) \mid P(\text{-}) \Rightarrow T(THE\ x.\ case\ (\Phi\ x)\ of\ P\ \psi \Rightarrow \psi\ dw)$
$\mid \text{-} \Rightarrow ERR(de)$
abbreviation $thatBinder{::}(e{\Rightarrow}io\ opt){\Rightarrow}e\ opt$ (**binder** $\boldsymbol{\iota}$ $[8]\ 9$) **where** $\boldsymbol{\iota}x.\ \varphi\ x \equiv \boldsymbol{\iota}\ \varphi$

4 Further Logical Connectives

Further logical connectives can be defined as usual. For pragmatic reasons (to avoid the blow-up of abbreviation expansions) we prefer direct definitions in all cases.

abbreviation $conj{::}io\ opt{\Rightarrow}io\ opt{\Rightarrow}io\ opt$ (**infixl** $\wedge$ 53) **where** $\varphi \wedge \psi \equiv case\ (\varphi,\psi)\ of$
$(P(\alpha),P(\beta)) \Rightarrow P(\lambda w.\ \alpha\ w \wedge \beta\ w) \mid (F(\alpha),F(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \wedge \beta\ w) \mid$
$(P(\alpha),F(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \wedge \beta\ w) \mid (F(\alpha),P(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \wedge \beta\ w) \mid$
$\text{-} \Rightarrow ERR(dio)$

abbreviation $disj{::}io\ opt{\Rightarrow}io\ opt{\Rightarrow}io\ opt$ (**infixl** $\vee$ 52) **where** $\varphi \vee \psi \equiv case\ (\varphi,\psi)\ of$
$(P(\alpha),P(\beta)) \Rightarrow P(\lambda w.\ \alpha\ w \vee \beta\ w) \mid (F(\alpha),F(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \vee \beta\ w) \mid$
$(P(\alpha),F(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \vee \beta\ w) \mid (F(\alpha),P(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \vee \beta\ w) \mid$
$\text{-} \Rightarrow ERR(dio)$

abbreviation $equiv{::}io\ opt{\Rightarrow}io\ opt{\Rightarrow}io\ opt$ (**infixl** $\equiv$ 51) **where** $\varphi \equiv \psi{\equiv}\ case\ (\varphi,\psi)\ of$
$(P(\alpha),P(\beta)) \Rightarrow P(\lambda w.\ \alpha\ w \longleftrightarrow \beta\ w) \mid (F(\alpha),F(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \longleftrightarrow \beta\ w) \mid$
$(P(\alpha),F(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \longleftrightarrow \beta\ w) \mid (F(\alpha),P(\beta)) \Rightarrow F(\lambda w.\ \alpha\ w \longleftrightarrow \beta\ w) \mid$
$\text{-} \Rightarrow ERR(dio)$

abbreviation $diamond{::}io\ opt{\Rightarrow}io\ opt\ (\Diamond\ \text{-}\ [62]\ 63)$ **where** $\Diamond\varphi \equiv case\ \varphi\ of$
$F(\psi) \Rightarrow F(\lambda w.\exists\, v.\ \psi\ v) \mid P(\psi) \Rightarrow P(\lambda w.\exists\, v.\ \psi\ v) \mid \text{-} \Rightarrow ERR(dio)$

abbreviation $exists{::}('a{\Rightarrow}io\ opt){\Rightarrow}io\ opt\ (\exists)$ **where** $\exists\,\Phi \equiv case\ (\Phi\ da)\ of$
$P(\text{-}) \Rightarrow P(\lambda w.\exists\, x.\ case\ (\Phi\ x)\ of\ P\ \psi \Rightarrow \psi\ w)$
$\mid F(\text{-}) \Rightarrow F(\lambda w.\ \exists\, x.\ case\ (\Phi\ x)\ of\ F\ \psi \Rightarrow \psi\ w) \mid \text{-} \Rightarrow ERR\ dio$
abbreviation $existsBinder{::}('a{\Rightarrow}io\ opt){\Rightarrow}io\ opt$ (**binder** $\exists$ $[8]\ 9$) **where** $\exists\, x.\ \varphi\ x \equiv \exists\,\varphi$

5 Meta-Logic

Our approach to rigorously distinguish between proper and improper language constructions and to explicitly maintain respective information is continued also at meta-level. For this we introduce three truth values *tt*, *ff* and *err*, representing truth, falsity and error. These values aro also noted as $\top$, $\bot$ and $*$. We could, of course, also introduce respective logical connectives for the meta-level, but in our applications (see below) this was not yet relevant.

datatype $mf = tt\ (\top) \mid ff\ (\bot) \mid err\ (*)$

Next we define the meta-logical notions of validity, satisfiability, countersatisfiability and invalidity for our embedded modal relational type theory. To support concise formula representations in the remainder we introduce the following notations: $[\varphi]$ (for φ is valid), $[\varphi]^{sat}$ (φ is satisfiability), $[\varphi]^{csat}$ (φ is countersatisfiability) and $[\varphi]^{inv}$ (φ is invalid). Actually, so far we only use validity.

abbreviation *valid* :: *io opt*$\Rightarrow$*mf* ([-] [*1*]) **where** $[\varphi] \equiv$ *case* φ *of*
$P(\psi) \Rightarrow$ *if* $\forall w.(\psi\ w) \longleftrightarrow$ *True then* $\top$ *else* $\bot$
$|\ F(\psi) \Rightarrow$ *if* $\forall w.(\psi\ w) \longleftrightarrow$ *True then* $\top$ *else* $\bot$ $|$ - $\Rightarrow *$
abbreviation *satisfiable* :: *io opt*$\Rightarrow$*mf* ($[\text{-}]^{sat}$ [*1*]) **where** $[\varphi]^{sat} \equiv$ *case* φ *of*
$P(\psi) \Rightarrow$ *if* $\exists w.(\psi\ w) \longleftrightarrow$ *True then* $\top$ *else* $\bot$
$|\ F(\psi) \Rightarrow$ *if* $\exists w.(\psi\ w) \longleftrightarrow$ *True then* $\top$ *else* $\bot$ $|$ - $\Rightarrow *$
abbreviation *countersatisfiable* :: *io opt*$\Rightarrow$*mf* ($[\text{-}]^{csat}$ [*1*]) **where** $[\varphi]^{csat} \equiv$ *case* φ *of*
$P(\psi) \Rightarrow$ *if* $\exists w.\neg(\psi\ w) \longleftrightarrow$ *True then* $\top$ *else* $\bot$
$|\ F(\psi) \Rightarrow$ *if* $\exists w.\neg(\psi\ w) \longleftrightarrow$ *True then* $\top$ *else* $\bot$ $|$ - $\Rightarrow *$
abbreviation *invalid* :: *io opt*$\Rightarrow$*mf* ($[\text{-}]^{inv}$ [*1*]) **where** $[\varphi]^{inv} \equiv$ *case* φ *of*
$P(\psi) \Rightarrow$ *if* $\forall w.\neg(\psi\ w) \longleftrightarrow$ *True then* $\top$ *else* $\bot$
$|\ F(\psi) \Rightarrow$ *if* $\forall w.\neg(\psi\ w) \longleftrightarrow$ *True then* $\top$ *else* $\bot$ $|$ - $\Rightarrow *$

6 Some Basic Tests

The next two statements are not theorems; Nitpick reports countermodels

lemma $[(\forall x.\ (\!|R^T,x^T|\!) \rightarrow \{\!|x^T,R^T|\!\})] = \top$ **apply** *simp* **nitpick oops** — Countermodel by Nitpick
lemma $[(\forall x.\ \{\!|x^T,R^T|\!\} \rightarrow (\!|R^T,x^T|\!))] = \top$ **apply** *simp* **nitpick oops** — Countermodel by Nitpick

lemma $[(\forall y.\ (\!|R^T,y^T|\!))] = \top$ **apply** *simp* **nitpick oops**

However, the next two statements are of course valid.

lemma $[(\forall x.\ (\!|R^T,x^T|\!) \rightarrow (\!|R^T,x^T|\!))] = \top$ **apply** *simp* **done**
lemma $[(\forall x.\ \{\!|x^T,R^T|\!\} \rightarrow \{\!|x^T,R^T|\!\})] = \top$ **apply** *simp* **done**

6.1 Verifying Necessitation

The next two lemmata show that neccessitation holds for arbitrary formulas and arbitrary propositional formulas. We present the lemma in both variants.

lemma *necessitationF*: $[\varphi^F] = \top \longrightarrow [\Box\varphi^F] = \top$ **apply** *simp* **done**
lemma *necessitationP*: $[\varphi^P] = \top \longrightarrow [\Box\varphi^P] = \top$ **apply** *simp* **done**

6.2 Modal Collapse is Countersatisfiable

The modelfinder Nitpick constructs a finite countermodel to the assertion that modal collaps holds.

lemma *modalCollapseF*: $[\varphi^F \rightarrow \Box\varphi^F] = \top$ **apply** *simp* **nitpick oops** — Countermodel by Nitpick

lemma *modalCollapseP*: $[\varphi^P \rightarrow \Box\varphi^P] = \top$ **apply** *simp* **nitpick oops** — Countermodel by Nitpick

6.3 Verifying S5 Principles

$\Box$ could have been modeled by employing an equivalence relation r in a guarding clause. This has been done in previous work. Our alternative, simpler definition of $\Box$ above omits this clause (since all worlds are reachable from any world in an equivalence relation). The following lemmata, which check various conditions for S5, confirm that we have indeed obtain a correct modeling of S5.

lemma *axiom-T-P*: $[\Box\varphi^P \to \varphi^P] = \top$ **apply** *simp* **done**
lemma *axiom-T-F*: $[\Box\varphi^F \to \varphi^F] = \top$ **apply** *simp* **done**

lemma *axiom-B-P*: $[\varphi^P \to \Box\Diamond\varphi^P] = \top$ **apply** *simp* **done**
lemma *axiom-B-F*: $[\varphi^F \to \Box\Diamond\varphi^F] = \top$ **apply** *simp* **done**

lemma *axiom-4-P*: $[\Box\varphi^P \to \Diamond\varphi^P] = \top$ **apply** *simp* **by** *auto*
lemma *axiom-4-F*: $[\Box\varphi^F \to \Diamond\varphi^F] = \top$ **apply** *simp* **by** *auto*

lemma *axiom-D-P*: $[\Box\varphi^P \to \Box\Box\varphi^P] = \top$ **apply** *simp* **done**
lemma *axiom-D-F*: $[\Box\varphi^F \to \Box\Box\varphi^F] = \top$ **apply** *simp* **done**

lemma *axiom-5-P*: $[\Diamond\varphi^P \to \Box\Diamond\varphi^P] = \top$ **apply** *simp* **done**
lemma *axiom-5-F*: $[\Diamond\varphi^F \to \Box\Diamond\varphi^F] = \top$ **apply** *simp* **done**

lemma *test-A-P*: $[\Box\Diamond\varphi^P \to \Diamond\varphi^P] = \top$ **apply** *simp* **done**
lemma *test-A-F*: $[\Box\Diamond\varphi^F \to \Diamond\varphi^F] = \top$ **apply** *simp* **done**

lemma *test-B-P*: $[\Diamond\Box\varphi^P \to \Diamond\varphi^P] = \top$ **apply** *simp* **by** *auto*
lemma *test-B-F*: $[\Diamond\Box\varphi^F \to \Diamond\varphi^F] = \top$ **apply** *simp* **by** *auto*

lemma *test-C-P*: $[\Box\Diamond\varphi^P \to \Box\varphi^P] = \top$ **apply** *simp* **nitpick oops** — Countermodel by Nitpick
lemma *test-C-F*: $[\Box\Diamond\varphi^F \to \Box\varphi^F] = \top$ **apply** *simp* **nitpick oops** — Countermodel by Nitpick

lemma *test-D-P*: $[\Diamond\Box\varphi^P \to \Box\varphi^P] = \top$ **apply** *simp* **done**
lemma *test-D-F*: $[\Diamond\Box\varphi^F \to \Box\varphi^F] = \top$ **apply** *simp* **done**

6.4 Relations between Meta-Logical Notions

lemma $[\varphi^P] = \top \longleftrightarrow [\varphi^P]^{csat} = \bot$ **apply** *simp* **done**
lemma $[\varphi^P]^{sat} = \top \longleftrightarrow [\varphi^P]^{inv} = \bot$ **apply** *simp* **done**
lemma $[\varphi^F] = \top \longleftrightarrow [\varphi^F]^{csat} = \bot$ **apply** *simp* **done**
lemma $[\varphi^F]^{sat} = \top \longleftrightarrow [\varphi^F]^{inv} = \bot$ **apply** *simp* **done**

However, for terms we have:

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

6.5 Testing the Propagation of Syntactical Category Information

lemma $\exists X.\ (\!|R^T,a^T|\!) = X^P \wedge \neg(\exists X.\ (\!|R^T,a^T|\!) = X^F) \wedge \neg(\exists X.\ (\!|R^T,a^T|\!) = X^T) \wedge \neg(\exists X.\ (\!|R^T,a^T|\!) = X^E)$ **apply** *simp* **done**
lemma $\exists X.\ \{\!|x^T,R^T|\!\} = X^F \wedge \neg(\exists X.\ \{\!|x^T,R^T|\!\} = X^P) \wedge \neg(\exists X.\ \{\!|x^T,R^T|\!\} = X^T) \wedge \neg(\exists X.\ \{\!|x^T,R^T|\!\} = X^E)$ **apply** *simp* **done**

Most importantly, we have that the following language construct is evaluated as ineligible at validity level; *error* ($*$) is returned.

lemma $(\boldsymbol{\lambda}x.\ (\!|R^T,x^T|\!) \to \{\!|x^T,R^T|\!\}) = X$ **apply** *simp* **oops**

lemma $[(\!|\boldsymbol{\lambda} x.\ (\!|R^T,x^T|\!) \rightarrow \{\!|x^T,R^T|\!\},a^T|\!)] = *$ **apply** *simp* **done**

This is also confirmed as follows in Isabelle: Isabelle simplifies the following expression to $dio^E = X$ (simply move the curse on *simp* to see this).

lemma $(\!|\boldsymbol{\lambda} x.\ (\!|R^T,x^T|\!) \rightarrow \{\!|x^T,R^T|\!\},a^T|\!) = X$ **apply** *simp* **oops** — X is dio^E
lemma $(\!|\boldsymbol{\lambda} x.\ (\!|R^T,x^T|\!) \wedge \neg\{\!|x^T,R^T|\!\},a^T|\!) = X$ **apply** *simp* **oops** — X is dio^E

6.6 Are Priorities Defined Correctly?

lemma $\varphi^P \wedge \psi^P \rightarrow \chi^P \equiv (\varphi^P \wedge \psi^P) \rightarrow \chi^P$ **apply** *simp* **done**
lemma $\varphi^P \wedge \psi^P \rightarrow \chi^P \equiv \varphi^P \wedge (\psi^P \rightarrow \chi^P)$ **apply** *simp* **nitpick oops** — Countermodel by Nitpick

lemma $(\varphi^P \wedge \psi^P \equiv \varphi^P \wedge \psi^P) \equiv ((\varphi^P \wedge \psi^P) \equiv (\varphi^P \wedge \psi^P))$ **apply** *simp* **done**
lemma $(\varphi^P \wedge \psi^P \equiv \varphi^P \wedge \psi^P) \equiv (\varphi^P \wedge (\psi^P \equiv \varphi^P) \wedge \psi^P)$ **apply** *simp* **nitpick oops** — Countermodel by Nitpick

7 E!, O!, A! and =E

We introduce the distinguished 1-place relation constant: E (read: being concrete or concreteness)

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

Being ordinary is defined as being possibly concrete.

abbreviation *ordinaryObject*::($e \Rightarrow io$) *opt* ($O!$) **where** $O! \equiv \boldsymbol{\lambda} x.\ \Diamond(\!|E^T,x^T|\!)$

lemma $O! = X$ **apply** *simp* **oops** — X is $(\lambda x\ w.\ Ex\ (exe\ E\ x))^T$

Being abstract is is defined as not possibly being concrete.

abbreviation *abstractObject*::($e \Rightarrow io$) *opt* ($A!$) **where** $A! \equiv \boldsymbol{\lambda} x.\ \neg(\Diamond(\!|E^T,x^T|\!))$

lemma $A! = X$ **apply** *simp* **oops** — X is $(\lambda x\ w.\ \forall\, xa.\ \neg\ exe\ (E\ x)\ xa)^T$

Identity relations $=_E$ and $=$ are introduced.

abbreviation *identityE*::e *opt*$\Rightarrow e$ *opt*$\Rightarrow io$ *opt* (**infixl** $=_E$ *63*) **where** $x =_E y \equiv$
$(\!|O!,x|\!) \wedge (\!|O!,y|\!) \wedge \Box(\forall\, F.\ (\!|F^T,x|\!) \equiv (\!|F^T,y|\!))$

lemma $a^T =_E a^T = X$ **apply** *simp* **oops** — X is "(...)^P

7.0.1 Remark: Nested lambda-expressions

lemma $(\boldsymbol{\lambda}\ x.\ x^T =_E a^T) = X$ **apply** *simp* **oops**
lemma $(\boldsymbol{\lambda}\ x.\ x^T =_E a^T) = (\boldsymbol{\lambda}\ x.\ a^T =_E x^T)$ **apply** *simp* **by** *metis*

7.1 Identity on Individuals

abbreviation *identityI*::e *opt*$\Rightarrow e$ *opt*$\Rightarrow io$ *opt* (**infixl** $=$ *63*) **where** $x = y \equiv$
$x =_E y \vee ((\!|A!,x|\!) \wedge (\!|A!,y|\!) \wedge \Box(\forall\, F.\ \{\!|x,F^T|\!\} \equiv \{\!|y,F^T|\!\}))$

7.1.1 Remark: Tracing the propagation of annotations

lemma $a^T = a^T = X$ **apply** *simp* **oops** — X is $(...)^F$
lemma $((\!|A!,a^T|\!) \wedge (\!|A!,a^T|\!) \wedge \Box(\forall F.\ \{\!|a^T,F^T|\!\} \equiv \{\!|a^T,F^T|\!\})) = X$ **apply** *simp* **oops** — X is $(...)^F$
lemma $((\!|A!,a^T|\!) \wedge (\!|A!,a^T|\!)) = X$ **apply** *simp* **oops** — X is $(...)^P$
lemma $\Box(\forall F.\ \{\!|a^T,F^T|\!\} \equiv \{\!|a^T,F^T|\!\}) = X$ **apply** *simp* **oops** — X is $(...)^F$

As intended: the following two lambda-abstractions are not well-formed/eligible and their evaluation reports in ERR-terms.

lemma $\boldsymbol{\lambda}^2(\lambda x\ y.\ x^T = y^T) = X$ **apply** *simp* **oops** — X is $(\lambda x\ y.\ dio)^E$
lemma $(\boldsymbol{\lambda}x.\ x^T = y^T) = X$ **apply** *simp* **oops** — X is $(\lambda x.\ dio)^E$

7.2 Identitiy on Relations

abbreviation *identityRel1*:: $((e \Rightarrow io)\ opt) \Rightarrow ((e \Rightarrow io)\ opt) \Rightarrow io\ opt$ (**infixl** $=^1$ *63*)
where $F1 =^1 G1 \equiv \Box(\forall x.\ \{\!|x^T,F1|\!\} \equiv \{\!|x^T,G1|\!\})$

abbreviation *identityRel2*:: $((e \Rightarrow e \Rightarrow io)\ opt) \Rightarrow ((e \Rightarrow e \Rightarrow io)\ opt) \Rightarrow io\ opt$ (**infixl** $=^2$ *63*)
where $F2 =^2 G2 \equiv \forall x1.(\ (\boldsymbol{\lambda}y.(\!|F2,y^T,x1^T|\!)) =^1 (\boldsymbol{\lambda}y.(\!|G2,y^T,x1^T|\!))$
$\wedge\ (\boldsymbol{\lambda}y.(\!|F2,x1^T,y^T|\!)) =^1 (\boldsymbol{\lambda}y.(\!|G2,x1^T,y^T|\!)))$

abbreviation *identityRel3*:: $((e \Rightarrow e \Rightarrow e \Rightarrow io)\ opt) \Rightarrow ((e \Rightarrow e \Rightarrow e \Rightarrow io)\ opt) \Rightarrow io\ opt$ (**infixl** $=^3$ *63*)
where $F3 =^3 G3 \equiv \forall x1\ x2.(\ (\boldsymbol{\lambda}y.(\!|F3,y^T,x1^T,x2^T|\!)) =^1 (\boldsymbol{\lambda}y.(\!|G3,y^T,x1^T,x2^T|\!))$
$\wedge\ (\boldsymbol{\lambda}y.(\!|F3,x1^T,y^T,x2^T|\!)) =^1 (\boldsymbol{\lambda}y.(\!|G3,x1^T,y^T,x2^T|\!))$
$\wedge\ (\boldsymbol{\lambda}y.(\!|F3,x1^T,x2^T,y^T|\!)) =^1 (\boldsymbol{\lambda}y.(\!|G3,x1^T,x2^T,y^T|\!)))$

lemma $F1^T =^1 G1^T = X$ **apply** *simp* **oops** — X is $(...)^F$
lemma $F2^T =^2 G2^T = X$ **apply** *simp* **oops** — X is $(...)^F$
lemma $F3^T =^3 G3^T = X$ **apply** *simp* **oops** — X is $(...)^F$
lemma $\{\!|x^T,F1^T|\!\} \equiv \{\!|x^T,G1^T|\!\} = X$ **apply** *simp* **oops** — X is $(...)^F$
lemma $(\!|F1^T,x^T|\!) \equiv (\!|G1^T,x^T|\!) = X$ **apply** *simp* **oops** — X is $(...)^P$
lemma $(\boldsymbol{\lambda}y.(\!|F2^T,y^T,x1^T|\!))= X$ **apply** *simp* **oops** — X is $(...)^T$

abbreviation *equalityRel0*::$io\ opt \Rightarrow io\ opt \Rightarrow io\ opt$ (**infixl** $=^0$ *63*)
where $F0 =^0 G0 \equiv (\boldsymbol{\lambda}y\ .\ F0) =^1 (\boldsymbol{\lambda}y.\ G0)$

Some tests: reflexity, symmetry, transitivity

lemma $F1^T =^1 F1^T = X$ **apply** *simp* **oops** — X is $(...)^F$
lemma $[F1^T =^1 F1^T] = \top$ **apply** *simp* **done**
lemma $[F2^T =^2 F2^T] = \top$ **apply** *simp* **done**
lemma $[F3^T =^3 F3^T] = \top$ **apply** *simp* **done**

lemma $[(F1^T =^1 G1^T) \equiv (G1^T =^1 F1^T)] = \top$ **apply** *simp* **by** *auto*
lemma $[(F2^T =^2 G2^T) \equiv (G2^T =^2 F2^T)] = \top$ **apply** *simp* **by** *auto*
lemma $[(F3^T =^3 G3^T) \equiv (G3^T =^3 F3^T)] = \top$ **apply** *simp* **by** *auto*

lemma $[(F1^T =^1 G1^T) \wedge (G1^T =^1 H1^T) \rightarrow (F1^T =^1 H1^T)] = \top$ **by** *simp*
lemma $[(F2^T =^2 G2^T) \wedge (G2^T =^2 H2^T) \rightarrow (F2^T =^2 H2^T)] = \top$ **by** *simp*
lemma $[(F3^T =^3 G3^T) \wedge (G3^T =^3 H3^T) \rightarrow (F3^T =^3 H3^T)] = \top$ **by** *simp*

The above examples are very resource intensive already

We discuss the example from [8, pp.365-366]:

lemma $(\boldsymbol{\lambda} x.\ \exists F.\ \{\!|x^T,F^T|\!\} \rightarrow (|F^T,x^T|)) = X$ **apply** *simp* **oops** — X is $(\lambda x.\ dio)^E$

abbreviation K **where** $K \equiv \boldsymbol{\lambda} x.\exists F.(\{\!|x^T,F^T|\!\} \rightarrow (|F^T,x^T|))$

lemma $K = X$ **apply** *simp* **oops** — X is $(\lambda x.\ dio)^E$

lemma $[(\exists x.\ (|A!,x^T|) \wedge (\forall F.\ (\{\!|x^T,F^T|\!\} \equiv F^T =^1 K)))] = *$ **apply** *simp* **done**
lemma $(\exists x.\ (|A!,x^T|) \wedge (\forall F.\ (\{\!|x^T,F^T|\!\} \equiv F^T =^1 K))) = X$ **apply** *simp* **oops** — X is $(dio)^E$

Tests on identity:

lemma $[a^T =_E a^T] = \top$ **apply** *simp* **nitpick oops** — Countermodel by Nitpick, as expected
lemma $[(|O!,a^T|) \rightarrow a^T =_E a^T] = \top$ **apply** *simp* **done**

lemma $[(\forall F.\ (|F^T,x^T|) \equiv (|F^T,x^T|))] = \top$ **apply** *simp* **done**
lemma $[(|O!,a^T|) \rightarrow (|\boldsymbol{\lambda} x.\ x^T =_E a^T,a^T|)] = \top$ **apply** *simp* **done**

lemma $[(a^T =_E a^T) \equiv (|\boldsymbol{\lambda} x.\ x^T =_E a^T,a^T|)] = \top$ **apply** *simp* **done**
lemma $[(a^T =_E a^T) \equiv \{\!|a^T,\boldsymbol{\lambda} x.\ x^T =_E a^T|\!\}] = \top$ **apply** *simp* **done**

lemma $[(\exists F.\ \{\!|a^T,F^T|\!\})] = \top$ **apply** *simp* **by** *auto*

lemma $[(\exists \varphi.\ \varphi^P)] = \top$ **apply** *simp* **by** *auto*
lemma $[(\exists \varphi.\ \varphi^F)] = \top$ **apply** *simp* **by** *auto*

7.3 Negation of Properties

abbreviation *notProp*::$((e \Rightarrow io)\ opt) \Rightarrow (e \Rightarrow io)\ opt$ ($\sim$- [58] 59) **where** $\sim \Phi \equiv$ *case* Φ *of* $T(\Psi) \Rightarrow \boldsymbol{\lambda} x.\neg(|\Phi,x^T|)$ | - $\Rightarrow ERR(deio)$

7.4 Individual Constant $\mathbf{a}_V$ and Function Term $\mathbf{a}_G$

abbreviation *a-V*::*e opt* ($\mathbf{a}_V$) **where** $\mathbf{a}_V \equiv \iota x.\ ((|A!,x^T|) \wedge (\forall F.\ \{\!|x^T,F^T|\!\} \equiv (F^T =^1 F^T)))$

abbreviation *a-G*::$(e \Rightarrow io)\ opt \Rightarrow e\ opt$ ($\mathbf{a}$_ [58] 59) **where** $\mathbf{a}_G \equiv \iota x.\ ((|A!,x^T|) \wedge (\forall F.\ \{\!|x^T,F^T|\!\} \equiv (F^T =^1 G)))$

8 Axioms

8.1 Axioms for Negations and Conditionals

lemma *a21-1-P*: $[\varphi^P \rightarrow (\varphi^P \rightarrow \varphi^P)] = \top$ **apply** *simp* **done**
lemma *a21-1-F*: $[\varphi^F \rightarrow (\varphi^F \rightarrow \varphi^F)] = \top$ **apply** *simp* **done**
lemma *a21-2-P*: $[(\varphi^P \rightarrow (\psi^P \rightarrow \chi^P)) \rightarrow ((\varphi^P \rightarrow \psi^P) \rightarrow (\varphi^P \rightarrow \chi^P))] = \top$ **apply** *simp* **done**
lemma *a21-2-F*: $[(\varphi^F \rightarrow (\psi^F \rightarrow \chi^F)) \rightarrow ((\varphi^F \rightarrow \psi^F) \rightarrow (\varphi^F \rightarrow \chi^F))] = \top$ **apply** *simp* **done**
lemma *a21-3-P*: $[(\neg\varphi^P \rightarrow \neg\psi^P) \rightarrow ((\neg\varphi^P \rightarrow \psi^P) \rightarrow \varphi^P)] = \top$ **apply** *simp* **done**
lemma *a21-3-F*: $[(\neg\varphi^F \rightarrow \neg\psi^F) \rightarrow ((\neg\varphi^F \rightarrow \psi^F) \rightarrow \varphi^F)] = \top$ **apply** *simp* **done**

8.2 Axioms of Identity

todo

8.3 Axioms of Quantification

todo

8.4 Axioms of Actuality

Here I have a big problem

lemma *a31-1-P*: $[\boldsymbol{\mathcal{A}}\varphi^P \equiv \varphi^P] = \top$ **apply** *simp* **nitpick oops**

8.5 Axioms of Necessity

lemma *a32-1-P*: $[(\Box(\varphi^P \to \varphi^P)) \to (\Box\varphi^P \to \Box\varphi^P)] = \top$ **apply** *simp* **done**
lemma *a32-1-F*: $[(\Box(\varphi^F \to \varphi^F)) \to (\Box\varphi^F \to \Box\varphi^F)] = \top$ **apply** *simp* **done**
lemma *a32-2-P*: $[\Box\varphi^P \to \varphi^P] = \top$ **apply** *simp* **done**
lemma *a32-2-F*: $[\Box\varphi^F \to \varphi^F] = \top$ **apply** *simp* **done**

lemma *a32-3-P*: $[\Box\Diamond\varphi^P \to \Diamond\varphi^P] = \top$ **apply** *simp* **done**
lemma *a32-3-F*: $[\Box\Diamond\varphi^F \to \Diamond\varphi^F] = \top$ **apply** *simp* **done**
lemma *a32-4-P*: $[(\forall\, x.\ \Box\varphi^P) \to \Box((\forall\, x.\ \varphi^P))] = \top$ **apply** *simp* **done**
lemma *a32-4-F*: $[(\forall\, x.\ \Box\varphi^F) \to \Box((\forall\, x.\ \varphi^F))] = \top$ **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-P: $[\Diamond(\exists\, x.\ (\!|E^T,x^T|\!) \wedge \Diamond(\neg(\!|E^T,x^T|\!))) \wedge \Diamond(\neg(\exists\, x.\ (\!|E^T,x^T|\!) \wedge \Diamond(\neg(\!|E^T,x^T|\!))))] = \top$

A brief check that this axiom is well-formed, i.e. does not return error

lemma $[\Diamond(\exists\, x.\ (\!|E^T,x^T|\!) \wedge \Diamond(\neg(\!|E^T,x^T|\!))) \wedge \Diamond(\neg(\exists\, x.\ (\!|E^T,x^T|\!) \wedge \Diamond(\neg(\!|E^T,x^T|\!))))] \neq *$ **apply** *simp* **done**
lemma $\Diamond(\exists\, x.\ (\!|E^T,x^T|\!) \wedge \Diamond(\neg(\!|E^T,x^T|\!))) \wedge \Diamond(\neg(\exists\, x.\ (\!|E^T,x^T|\!) \wedge \Diamond(\neg(\!|E^T,x^T|\!)))) = X$ **apply** *simp* **oops** — X is $(...)^P$

8.6 (Instances of) Barcan Formula and Converse Barcan Formula

lemma *BF-inst*: $[(\forall\, \alpha.\ \Box(\!|R^T,\alpha^T|\!)) \to \Box(\forall\, \alpha.(\!|R^T,\alpha^T|\!))] = \top$ **by** *simp*
lemma *CBF-inst*: $[\Box(\forall\, \alpha.(\!|R^T,\alpha^T|\!)) \to (\forall\, \alpha.\ \Box(\!|R^T,\alpha^T|\!))] = \top$ **apply** *simp* **by** *auto*

8.7 Axioms of Necessity and Actuality

lemma *a33-1-P*: $[\boldsymbol{\mathcal{A}}\varphi^P \to \Box\boldsymbol{\mathcal{A}}\varphi^P] = \top$ **apply** *simp* **done**
lemma *a33-1-F*: $[\boldsymbol{\mathcal{A}}\varphi^F \to \Box\boldsymbol{\mathcal{A}}\varphi^F] = \top$ **apply** *simp* **done**
lemma *a33-2-P*: $[\Box\varphi^P \equiv \boldsymbol{\mathcal{A}}(\Box\varphi^P)] = \top$ **apply** *simp* **done**
lemma *a33-2-F*: $[\Box\varphi^F \equiv \boldsymbol{\mathcal{A}}(\Box\varphi^F)] = \top$ **apply** *simp* **done**

8.8 Axioms for Descriptions

lemma $(x^T = (\iota x.\{\!|x^T,R^T|\!\})) = X$ **apply** *simp* **oops** — X is $(...)^F$
lemma $(\forall\, z.\ (\boldsymbol{\mathcal{A}}(\{\!|x^T,R^T|\!\}) \equiv (z^T = x^T))) = X$ **apply** *simp* **oops** — X is $(...)^F$

For the following lemma cannot yet be automatically proved, since proof automation for definite descriptions is still not well enough developed in ATPs.

lemma *a34-Inst-1*: $[(x^T = (\iota x.\{\!|x^T,R^T|\!\})) \equiv (\forall\, z.\ (\boldsymbol{\mathcal{A}}(\{\!|z^T,R^T|\!\}) \equiv (z^T = x^T)))] = \top$ **apply** *simp* **oops**

8.9 Axioms for Complex relation Terms

We check for some α-renaming instances

lemma $(\boldsymbol{\lambda} z.(\!| R^T, z^T, (\iota y.(\!| Q^T, y^T |\!)) |\!)) = (\boldsymbol{\lambda} x.(\!| R^T, x^T, (\iota z.(\!| Q^T, z^T |\!)) |\!))$ **apply** *simp* **done**

lemma $((\forall F.(\!| F^T, a^T |\!)) \equiv (\forall G.(\!| G^T, b^T |\!))) = (\forall F.(\!| F^T, a^T |\!)) \equiv (\forall F.(\!| F^T, b^T |\!))$ **apply** *simp* **done**

Others are analogously valid, we omit them here

8.10 Axioms of Encoding

The following need to become an axioms; they are not implied by the embedding.

axiomatization where
a36: $[\{\!| x^T, G^T |\!\} \rightarrow \Box\{\!| x^T, G^T |\!\}] = \top$ **and**
a37: $[\boldsymbol{\mathcal{A}}\{\!| x^T, G^T |\!\} \rightarrow \{\!| x^T, G^T |\!\}] = \top$

The following however holds

lemma $[\Box(\boldsymbol{\mathcal{A}}\{\!| x^T, G^T |\!\} \rightarrow \{\!| x^T, G^T |\!\})] = \top$ **apply** *simp* **nitpick oops**

9 Leibniz Theory of Concepts

Below we don't get that far yet, a systematic bottom up development seems to be required first

abbreviation *LeibnizianConcept*$::(e \Rightarrow io)$ *opt* $(C!)$
where $C! \equiv \boldsymbol{\lambda} x.\ (\!| A!, x^T |\!)$
abbreviation *ConceptSummation* (**infix** $\bigoplus$ *100*)
where $x \bigoplus y \equiv \iota z.\ ((\!| C!, x |\!) \wedge (\forall F.\ (\{\!| z^T, F^T |\!\} \equiv \{\!| x, F^T |\!\} \vee \{\!| y, F^T |\!\})))$
abbreviation *ConceptInclusion* (**infix** $\preceq$ *100*)
where $x \preceq y \equiv \forall F.\ (\{\!| x, F^T |\!\} \rightarrow \{\!| y, F^T |\!\})$

lemma $[x^T \preceq y^T \equiv (\exists z.\ ((x^T \bigoplus z^T) = y^T))] = \top$ **apply** *simp* **oops**
lemma $[x^T \preceq y^T \equiv (x^T \bigoplus y^T = y^T)] = \top$ **apply** *simp* **oops**

lemma $[(\!| \boldsymbol{\lambda} x.\ (\!| R^T, x^T |\!), y^T |\!)] = X$ **apply** *simp* **oops**
lemma $[\{\!| y^T, \boldsymbol{\lambda} x.\ \{\!| x^T, R^T |\!\} |\!\}] = X$ **apply** *simp* **oops**
lemma $[\{\!| y^T, \boldsymbol{\lambda} x.\ (\!| R^T, x^T |\!) |\!\}] = X$ **apply** *simp* **oops**

References

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

[2] C. Benzmüller and B. Woltzenlogel-Paleo. Gödel's God in Isabelle/HOL. *Archive of Formal Proofs*, 2013, 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] C. Benzmüller and B. Woltzenlogel Paleo. The inconsistency in Gödels ontological argument: A success story for AI in metaphysics. In *IJCAI 2016*, 2016. Accepted for publication; to appear in 2016.

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

[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. *Abstract Objects: An Introduction to Axiomatic Metaphysics*. Dordrecht: D. Reidel, 1983.

[8] 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.