Automation of the Principia Metaphysica in HOL: Part I

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

October 26, 2015

1 Introduction

This work significantly adapts and extends the embedding presented in [1]; say more ...

2 Preliminaries

typedecl *i*
— the type possible worlds; the formalism explicitly encodes Kripke style semantics

type-synonym *io* = (*i* ⇒ *bool*)
— modal logic formulas (or propositional formulas) are essentially of this type: predicates on
— worlds

typedecl *e*
— the raw type of entities/objects (abstract or ordinary)

datatype *'a opt* = *Error 'a* | *Term 'a* | *Form 'a* | *PropForm 'a*

consts *cw* :: *i*
— some fixed current world

consts *dE*::*e dIO*::*io dEIO*::*e*⇒*io dEEIO*::*e*=>*e*⇒*io dEEEIO*::*e*=>*e*=>*e*⇒*io dA*::*'a*
— some fixed dummy symbols; we anyway assume that the domains are on-empty
— needed as dummy object in some cases below

We consider an arbitrary but fixed accessibility relation r

consts *r* :: (*i*⇒*i*⇒*bool*)

Meta-logical predicates.

abbreviation *isWff* :: *'a opt*⇒*bool* **where** *isWff* φ ≡ *case* φ *of Error* ψ ⇒ *False* | - ⇒ *True*
abbreviation *isForm* :: *'a opt*⇒*bool* **where** *isForm* φ ≡ *case* φ *of Form* ψ ⇒ *True* | - ⇒ *False*
abbreviation *isPropForm* :: *'a opt*⇒*bool* **where** *isPropForm* φ ≡ *case* φ *of PropForm* ψ ⇒ *True* |
- ⇒ *False*
abbreviation *isTerm* :: *'a opt*⇒*bool* **where** *isTerm* φ ≡ *case* φ *of Term* ψ ⇒ *True* | - ⇒ *False*
abbreviation *isError* :: *'a opt*⇒*bool* **where** *isError* φ ≡ *case* φ *of Error* ψ ⇒ *True* | - ⇒ *False*
abbreviation *valid* :: *io opt*⇒*bool* **where** [φ] ≡ *case* φ *of*
PropForm ψ ⇒ ∀ *w*.(ψ *w*)
| *Form* ψ ⇒ ∀ *w*.(ψ *w*)

| - ⇒ *False*
abbreviation *satifiable* :: *io opt⇒bool* **where** $[\varphi]^{sat} \equiv$ *case* φ *of*
PropForm $\psi \Rightarrow \exists\, w.(\psi\ w)$
| *Form* $\psi \Rightarrow \exists\, w.(\psi\ w)$
| - ⇒ *False*
abbreviation *countersatifiable* :: *io opt⇒bool* **where** $[\varphi]^{csat} \equiv$ *case* φ *of*
PropForm $\psi \Rightarrow \exists\, w.\neg(\psi\ w)$
| *Form* $\psi \Rightarrow \exists\, w.\neg(\psi\ w)$
| - ⇒ *False*
abbreviation *invalid* :: *io opt⇒bool* **where** $[\varphi]^{inv} \equiv$ *case* φ *of*
PropForm $\psi \Rightarrow \forall\, w.\neg(\psi\ w)$
| *Form* $\psi \Rightarrow \forall\, w.\neg(\psi\ w)$
| - ⇒ *False*

3 Encoding of the language

abbreviation $\mathcal{A}$::*io opt* ⇒ *io opt* **where** $\mathcal{A}\ \varphi \equiv$ *case* φ *of*
Form $\psi \Rightarrow$ *Form* $(\lambda w.\ \psi\ cw)$
| *PropForm* $\psi \Rightarrow$ *PropForm* $(\lambda w.\ \psi\ cw)$
| - ⇒ *Error dIO*

actuality operator; φ is evaluated wrt the current world; Error is passed on

abbreviation *Enc*::*e opt⇒*(*e⇒io*) *opt⇒io opt* **where** $<x \circ P> \equiv$ *case* (*x*,*P*) *of*
(*Term y*,*Term Q*) ⇒ *Form* $(\lambda w.(Q\ y)\ w)$
| (-,-) ⇒ *Error dIO*

$\kappa_1\Pi^1$ will be written here as $<\kappa_1 \circ \Pi^1>$; $\kappa_1\Pi^1$ is a Form; Error is passed on

abbreviation *Exe1*::(*e⇒io*) *opt⇒e opt⇒io opt* **where** $<P \cdot x> \equiv$ *case* (*P*,*x*) *of*
(*Term Q*,*Term y*) ⇒ *PropForm* $(\lambda w.(Q\ y)\ w)$
| - ⇒ *Error dIO*

$\Pi^1\kappa_1$ will be written here as $<\Pi^2 \cdot \kappa_1>$; $\Pi^1\kappa_1$ is a PropForm; Error is passed on

abbreviation *Exe2*::(*e⇒e⇒io*) *opt⇒e opt⇒e opt⇒io opt* **where** $<P \cdot x1,x2> \equiv$ *case* (*P*,*x1*,*x2*) *of*
(*Term Q*,*Term y1*,*Term y2*) ⇒ *PropForm* $(\lambda w.(Q\ y1\ y2)\ w)$
| - ⇒ *Error dIO*

$\Pi^2\kappa_1\kappa_2$ will be written here as $<\Pi^2 \cdot \kappa_1,\kappa_2>$; $\Pi^2\kappa_1\kappa_2$ is a PropForm; Error is passed on

abbreviation *Exe3*::(*e⇒e⇒e⇒io*) *opt⇒e opt⇒e opt⇒e opt⇒io opt* **where** $<P \cdot x1,x2,x3> \equiv$ *case* (*P*,*x1*,*x2*,*x3*) *of*
(*Term Q*,*Term y1*,*Term y2*,*Term y3*) ⇒ *PropForm* $(\lambda w.(Q\ y1\ y2\ y3)\ w)$
| - ⇒ *Error dIO*

${\Pi^3}_1\kappa_2\kappa_3$ will be written here as $<\Pi^2 \cdot \kappa_1,\kappa_2,\kappa_3>$; ${\Pi^3}_1\kappa_2\kappa_3$ is a PropForm; Error is passed on; we could, of course, introduce further operators: Exe4, Exe5, etc.

abbreviation *z-not*::*io opt⇒io opt* **where** $\neg^z\ \varphi \equiv$ *case* φ *of*
Form $\psi \Rightarrow$ *Form* $(\lambda w.\ \neg\ \psi\ w)$
| *PropForm* $\psi \Rightarrow$ *PropForm* $(\lambda w.\ \neg\ \psi\ w)$
| - ⇒ *Error dIO*

negation operator; $\neg^z\ \varphi$ inherits its type from φ if φ is Form or PropForm; Error is passed on

abbreviation *z-implies*::*io opt*⇒*io opt*⇒*io opt* **where** $\varphi \rightarrow^z \psi \equiv$ *case* (φ,ψ) *of*
  $(PropForm\ \alpha, PropForm\ \beta) \Rightarrow PropForm\ (\lambda w.\ \alpha\ w \longrightarrow \beta\ w)$
| $(Form\ \alpha, Form\ \beta) \Rightarrow Form\ (\lambda w.\ \alpha\ w \longrightarrow \beta\ w)$
| - ⇒ *Error dIO*

implication operator; $\varphi \rightarrow^z \psi$ returns returns a PropForm if both are PropForms, Form if both are Forms, otherwise it returns Error

abbreviation *z-forall*::(*'a*⇒*io opt*)⇒*io opt* **where** $\forall\ \Phi \equiv$ *case* $(\Phi\ dA)$ *of*
  $PropForm\ \varphi \Rightarrow PropForm\ (\lambda w.\ \forall x.\ case\ (\Phi\ x)\ of\ PropForm\ \psi \Rightarrow \psi\ w)$
| $Form\ \varphi \Rightarrow Form\ (\lambda w.\ \forall x.\ case\ (\Phi\ x)\ of\ Form\ \psi \Rightarrow \psi\ w)$
| - ⇒ *Error 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 *z-box*::*io opt*⇒*io opt* **where** $\Box^r\ \varphi \equiv$ *case* φ *of*
  $Form\ \psi \Rightarrow Form\ (\lambda w.\ \forall v.\ (r\ w\ v) \longrightarrow \psi\ v)$
| $PropForm\ \psi \Rightarrow PropForm\ (\lambda w.\ \forall v.\ (r\ w\ v) \longrightarrow \psi\ v)$
| - ⇒ *Error dIO*

box operator; $\Box^r\ \varphi$ inherits its type (Form or PropForm) from φ; Error is passed on

abbreviation *lam0*::*io opt*⇒*io opt* **where** $\lambda^0\ \varphi \equiv$ *case* φ *of*
  $PropForm\ \psi \Rightarrow PropForm\ \psi$
| - ⇒ *Error dIO*

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

abbreviation *lam1*::(*e*⇒*io opt*)⇒(*e*⇒*io*) *opt* **where** $\lambda^1\ \Phi \equiv$ *case* $(\Phi\ dE)$ *of*
  $PropForm\ \varphi \Rightarrow Term\ (\lambda x.\ case\ (\Phi\ x)\ of\ PropForm\ \varphi \Rightarrow \varphi)$
| - ⇒ *Error* $(\lambda x.\ dIO)$

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

abbreviation *lam2*::(*e*⇒*e*⇒*io opt*)⇒(*e*⇒*e*⇒*io*) *opt* **where** $\lambda^2\ \Phi \equiv$ *case* $(\Phi\ dE\ dE)$ *of*
  $PropForm\ \varphi \Rightarrow Term\ (\lambda x\ y.\ case\ (\Phi\ x\ y)\ of\ PropForm\ \varphi \Rightarrow \varphi)$
| - ⇒ *Error* $(\lambda x\ y.\ dIO)$

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

abbreviation *lam3*::(*e*⇒*e*⇒*e*⇒*io opt*)⇒(*e*⇒*e*⇒*e*⇒*io*) *opt* **where** $\lambda^3\ \Phi \equiv$ *case* $(\Phi\ dE\ dE\ dE)$ *of*
  $PropForm\ \varphi \Rightarrow Term\ (\lambda x\ y\ z.\ case\ (\Phi\ x\ y\ z)\ of\ PropForm\ \varphi \Rightarrow \varphi)$
| - ⇒ *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 $(THE\ x.\ \varphi\ x\ cw)$, 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** $\varphi \wedge^z \psi \equiv \neg^z(\varphi \rightarrow^z \neg^z\psi)$
abbreviation *z-or*::*io opt*⇒*io opt*⇒*io opt* **where** $\varphi \vee^z \psi \equiv (\neg^z\varphi \rightarrow^z \psi)$
abbreviation *z-equiv*::*io opt*⇒*io opt*⇒*io opt* **where** $\varphi \equiv^z \psi \equiv (\varphi \rightarrow^z \psi) \wedge^z (\psi \rightarrow^z \varphi)$
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** $\Diamond^r\ \varphi \equiv \neg^z\ \Box^r\ (\neg^z\ \varphi)$

5 Some shorthands 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*

6 Some basic tests

An example signature; entities and relations

consts *a-0* :: *e* **abbreviation** *a* **where** *a* ≡ *.a-0.*
consts *b-0* :: *e* **abbreviation** *b* **where** *b* ≡ *.b-0.*
consts *c-0* :: *e* **abbreviation** *c* **where** *c* ≡ *.c-0.*

consts *R-0* :: *io* **abbreviation** *R0* **where** *R0* ≡ *.R-0.*
consts *R-1* :: *e*⇒*io* **abbreviation** *R1* **where** *R1* ≡ *.R-1.*
consts *R-2* :: *e*⇒*e*⇒*io* **abbreviation** *R2* **where** *R2* ≡ *.R-2.*
consts *R-3* :: *e*⇒*e*⇒*e*⇒*io* **abbreviation** *R3* **where** *R3* ≡ *.R-3.*

Testing some term and formula constructions

lemma [<*R1*•*a*>] **nitpick oops**
lemma *isPropForm* <*R1*•*a*> **apply** (*simp*) **done**
lemma <*R1*•*a*> = *X* **apply** (*simp*) **oops**

lemma [<*a*∘*R1*>] **nitpick oops**
lemma *isPropForm* <*a*∘*R1*> **apply** (*simp*) **oops**
lemma *isForm* <*a*∘*R1*> **apply** (*simp*) **done**
lemma <*a*∘*R1*> = *X* **apply** (*simp*) **oops**

lemma [<λ^1($\lambda x.$ <*R1*•*.x.*> →^{z} <*R1*•*.x.*>)•*a*>] **apply** (*simp*) **done**
lemma <λ^1($\lambda x.$ <*R1*•*.x.*> →^{z} <*R1*•*.x.*>)•*a*> = *X* **apply** (*simp*) **oops**

lemma ¬ *isWff* (<*R1*•*.x.*> →^{z} <*.x.*∘*R1*>) **apply** (*simp*) **done**
lemma λ^1($\lambda x.$ <*R1*•*.x.*> →^{z} <*.x.*∘*R1*>) = *X* **apply** (*simp*) **oops**

lemma [<λ^1($\lambda x.$ <*R1*•*.x.*> →^{z} <*.x.*∘*R1*>)•*a*>] **apply** (*simp*) **oops**
lemma <λ^1($\lambda x.$ <*R1*•*.x.*> →^{z} <*.x.*∘*R1*>)•*a*> = *X* **apply** (*simp*) **oops**

lemma [∀ ($\lambda x.$ <*R1*•*.x.*> →^{z} <*R1*•*.x.*>)] **apply** (*simp*) **done**
lemma [∀ ($\lambda R.$ ∀ ($\lambda x.$ <*.R.*•*.x.*> →^{z} <*.R.*•*.x.*>))] **apply** (*simp*) **done**
lemma ∀ ($\lambda x.$ <*R1*•*.x.*> →^{z} <*R1*•*.x.*>) = *X* **apply** (*simp*) **oops**

lemma $[\forall\,(\lambda x.\ <.x.\circ R1.> \rightarrow^z\ <.x.\circ R1.>)]$ **apply** (*simp*) **done**
lemma $\forall\,(\lambda x.\ <.x.\circ R1.> \rightarrow^z\ <.x.\circ R1.>) = X$ **apply** (*simp*) **oops**

lemma $[\forall\,(\lambda x.\ <R1\cdot.x.> \rightarrow^z\ <.x.\circ R1.>)]$ **apply** (*simp*) **oops**
lemma $\forall\,(\lambda x.\ <R1\cdot.x.> \rightarrow^z\ <.x.\circ R1.>) = X$ **apply** (*simp*) **oops**
lemma $[\forall\,(\lambda R.\ <.R.\cdot.x.> \rightarrow^z\ <.x.\circ.R.>)]$ **apply** (*simp*) **oops**
lemma $\forall\,(\lambda R.\ <.R.\cdot.x.> \rightarrow^z\ <.x.\circ.R.>) = X$ **apply** (*simp*) **oops**

7 Get the priorities right

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

lemma $(,\varphi,\ \wedge^z\ ,\psi,\ \equiv^z\ ,\varphi,\ \wedge^z\ ,\psi,) \equiv ((,\varphi,\ \wedge^z\ ,\psi,)\ \equiv^z\ (,\varphi,\ \wedge^z\ ,\psi,))$ **apply** (*simp*) **done**
lemma $(,\varphi,\ \wedge^z\ ,\psi,\ \equiv^z\ ,\varphi,\ \wedge^z\ ,\psi,) \equiv (,\varphi,\ \wedge^z\ (,\psi,\ \equiv^z\ ,\varphi,)\ \wedge^z\ ,\psi,)$ **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** $O^! \equiv \lambda^1(\lambda x.\ \Diamond^r\ <.E.\cdot.x.>)$

Being ordinary is being possibly concrete.

Question: is the term above a Form or a PropForm?

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

Being abstract is not possibly being concrete.

Question: is the term above a Form or a PropForm?

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

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

9 Some further examples

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

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

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

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

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

lemma *isWff* ,$(\lambda w.\ True)$, **apply** $(simp)$ **done**

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

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.