Automation of the Principia Metaphysica in HOL: Part I

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

October 27, 2015

1 Introduction

This work is related to [1], which is significantly extends ...

2 Preliminaries

typedecl i
— the type possible worlds; the formalism explicitly encodes Kripke style semantics
type-synonym $io = (i \Rightarrow bool)$
— formulas are essentially of this type
— predicates on worlds
typedecl e
— the raw type of entities/objects (abstract or ordinary)
datatype $'a\ opt = Error\ 'a \mid Term\ 'a \mid Form\ 'a \mid PropForm\ 'a$

consts $cw :: i$
— the distinguished actual world
consts $dE{::}e\ dIO{::}io\ dEIO{::}e{\Rightarrow}io\ dEEIO{::}e{=>}e{\Rightarrow}io\ dEEEIO{::}e{=>}e{=>}e{\Rightarrow}io\ dA{::}'a$
— some fixed dummy symbols; we anyway assume that the domains are non-empty
— needed as dummy object in some cases below

Meta-logical predicates.

abbreviation $isWff :: 'a\ opt{\Rightarrow}bool$ **where** $isWff\ \varphi \equiv case\ \varphi\ of\ Error\ \psi \Rightarrow False \mid Term\ \psi \Rightarrow False \mid \text{-} \Rightarrow True$
abbreviation $isForm :: 'a\ opt{\Rightarrow}bool$ **where** $isForm\ \varphi \equiv case\ \varphi\ of\ Form\ \psi \Rightarrow True \mid \text{-} \Rightarrow False$
abbreviation $isPropForm :: 'a\ opt{\Rightarrow}bool$ **where** $isPropForm\ \varphi \equiv case\ \varphi\ of\ PropForm\ \psi \Rightarrow True \mid \text{-} \Rightarrow False$
abbreviation $isTerm :: 'a\ opt{\Rightarrow}bool$ **where** $isTerm\ \varphi \equiv case\ \varphi\ of\ Term\ \psi \Rightarrow True \mid \text{-} \Rightarrow False$
abbreviation $isError :: 'a\ opt{\Rightarrow}bool$ **where** $isError\ \varphi \equiv case\ \varphi\ of\ Error\ \psi \Rightarrow True \mid \text{-} \Rightarrow False$
abbreviation $valid :: io\ opt{\Rightarrow}bool$ **where** $[\varphi] \equiv case\ \varphi\ of$
$PropForm\ \psi \Rightarrow \forall w.(\psi\ w)$
$\mid Form\ \psi \Rightarrow \forall w.(\psi\ w)$
$\mid \text{-} \Rightarrow False$
abbreviation $satifiable :: io\ opt{\Rightarrow}bool$ **where** $[\varphi]^{sat} \equiv case\ \varphi\ of$
$PropForm\ \psi \Rightarrow \exists w.(\psi\ w)$
$\mid Form\ \psi \Rightarrow \exists w.(\psi\ w)$
$\mid \text{-} \Rightarrow False$
abbreviation $countersatifiable :: io\ opt{\Rightarrow}bool$ **where** $[\varphi]^{csat} \equiv\ case\ \varphi\ of$

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

3 Encoding of the language

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

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

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

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

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

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

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

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

abbreviation *z-not*::*io opt*$\Rightarrow$*io opt* **where** $\neg^z\ \varphi \equiv$ *case* φ *of*
Form $\psi \Rightarrow$ *Form* $(\lambda w.\ \neg\ \psi\ w)$
| *PropForm* $\psi \Rightarrow$ *PropForm* $(\lambda w.\ \neg\ \psi\ w)$
| - $\Rightarrow$ *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*$\Rightarrow$*io opt*$\Rightarrow$*io opt* **where** $\varphi \rightarrow^z \psi \equiv$ *case* (φ,ψ) *of*
(*PropForm* α,*PropForm* β) $\Rightarrow$ *PropForm* $(\lambda w.\ \alpha\ w \longrightarrow \beta\ w)$
| (*Form* α,*Form* β) $\Rightarrow$ *Form* $(\lambda w.\ \alpha\ w \longrightarrow \beta\ w)$
| - $\Rightarrow$ *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*::($'a \Rightarrow io\ opt) \Rightarrow io\ opt$ **where** $\forall\ \Phi \equiv case\ (\Phi\ dA)\ of$
$\quad PropForm\ \varphi \Rightarrow PropForm\ (\lambda w.\ \forall x.\ case\ (\Phi\ x)\ of\ PropForm\ \psi \Rightarrow \psi\ w)$
$\mid Form\ \varphi \Rightarrow Form\ (\lambda w.\ \forall x.\ case\ (\Phi\ x)\ of\ Form\ \psi \Rightarrow \psi\ w)$
$\mid - \Rightarrow Error\ dIO$

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*::$io\ opt \Rightarrow io\ opt$ **where** $\Box^r\ \varphi \equiv case\ \varphi\ of$
$\quad Form\ \psi \Rightarrow Form\ (\lambda w.\ \forall v.\ \psi\ v)$
$\mid PropForm\ \psi \Rightarrow PropForm\ (\lambda w.\ \forall v.\ \psi\ v)$
$\mid - \Rightarrow Error\ dIO$

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 *lam0*::$io\ opt \Rightarrow io\ opt$ **where** $\lambda^0\ \varphi \equiv case\ \varphi\ of$
$\quad PropForm\ \psi \Rightarrow PropForm\ \psi$
$\mid - \Rightarrow Error\ dIO$

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

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

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

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

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

abbreviation *lam3*::$(e \Rightarrow e \Rightarrow e \Rightarrow io\ opt) \Rightarrow (e \Rightarrow e \Rightarrow e \Rightarrow io)\ opt$ **where** $\lambda^3\ \Phi \equiv case\ (\Phi\ dE\ dE\ dE)\ of$
$\quad PropForm\ \varphi \Rightarrow Term\ (\lambda x\ y\ z.\ case\ (\Phi\ x\ y\ z)\ of\ PropForm\ \varphi \Rightarrow \varphi)$
$\mid - \Rightarrow Error\ (\lambda x\ y\ z.\ dIO)$

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*::$(e \Rightarrow io\ opt) \Rightarrow e\ opt$ **where** $\varepsilon\ \Phi \equiv case\ (\Phi\ dE)\ of$
$\quad PropForm\ \varphi \Rightarrow Term\ (THE\ x.\ case\ (\Phi\ x)\ of\ PropForm\ \psi \Rightarrow \psi\ cw)$
$\mid - \Rightarrow Error\ dE$

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* (Φ *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 shortcuts for the constructors

abbreviation *mkPropForm* :: *io*⇒*io opt* **where** $,p, \equiv$ *PropForm p*
abbreviation *mkForm* :: *io*⇒*io opt* **where** $;p; \equiv$ *Form p*
abbreviation *mkTerm* :: '*a*⇒'*a opt* **where** $.t. \equiv$ *Term t*

6 Some basic tests

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 term and formula constructions

lemma $[<R1{\cdot}a>]$ **nitpick oops**
lemma *isPropForm* $<R1{\cdot}a>$ **apply** (*simp*) **done**
lemma $<R1{\cdot}a> = X$ **apply** (*simp*) **oops**

lemma $[<a{\circ}R1>]$ **nitpick oops**
lemma *isPropForm* $<a{\circ}R1>$ **apply** (*simp*) **oops**
lemma *isForm* $<a{\circ}R1>$ **apply** (*simp*) **done**
lemma $<a{\circ}R1> = X$ **apply** (*simp*) **oops**

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

lemma $\neg$ *isWff* $(<R1{\cdot}.x.> \rightarrow^z <.x.{\circ}R1>)$ **apply** (*simp*) **done**
lemma $\lambda^1(\lambda x.\ <R1{\cdot}.x.> \rightarrow^z <.x.{\circ}R1>) = X$ **apply** (*simp*) **oops**

lemma $[<\lambda^1(\lambda x.\ <R1{\cdot}.x.> \rightarrow^z <.x.{\circ}R1>){\cdot}a>]$ **apply** (*simp*) **oops**
lemma $<\lambda^1(\lambda x.\ <R1{\cdot}.x.> \rightarrow^z <.x.{\circ}R1>){\cdot}a> = X$ **apply** (*simp*) **oops**

lemma $[\forall(\lambda x.\ <R1{\cdot}.x.> \rightarrow^z <R1{\cdot}.x.>)]$ **apply** (*simp*) **done**
lemma $[\forall(\lambda R.\ \forall(\lambda x.\ <.R.{\cdot}.x.> \rightarrow^z <.R.{\cdot}.x.>))]$ **apply** (*simp*) **done**
lemma $\forall(\lambda x.\ <R1{\cdot}.x.> \rightarrow^z <R1{\cdot}.x.>) = X$ **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 Are the priorities set correctly?

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

Being ordinary is being possibly concrete.

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

Being abstract is not possibly being concrete.

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

9 Further test 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 $\mathit{isWff}\ ,(\lambda w.\ \mathit{True}),$ **apply** (simp) **done**

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

lemma $\lfloor(\exists\,(\lambda F.\ ;F;))\rfloor$ **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.