Documentation of the Embedding

Daniel Kirchner

April 12, 2017

1 Embedding

1.1 Primitives

The background theory for the embedding is Isabelle/HOL, that provides a higher order logic that serves as our meta-logic. For a short overview of the extents of the background theory see [1].

The following primitive types are the basis of the embedding:

- Type i represents possible worlds in the Kripke semantics.
- Type j represents *states* that are used for different interpretations of relations and connectives to achieve a hyper-intensional logic (see below).
- Type *bool* represents meta-logical truth values (*True* or *False*) and is inherited from Isabelle/HOL.
- Type ω represents ordinary urelements.
- Type σ represents special urelements.

Two constants are introduced:

- The constant *dw* of type i represents the designated actual world.
- The constant *dj* of type j represents the designated actual state.

Based on the primitive types above the following types are defined:

- Type o is defined as the set of all functions of type $j \Rightarrow i \Rightarrow \mathit{bool}$ and represents truth values in the embedded logic.
- Type υ is defined as **datatype** $\upsilon = \omega\upsilon\ \omega \mid \sigma\upsilon\ \sigma$. This type represents urelements and an object of this type can be either an ordinary or a special urelement (with the respective type constructors $\omega\upsilon$ and $\sigma\upsilon$).
- Type Π_0 is defined as a synonym for type o and represents zero-place relations.

- Type Π_1 is defined as the set of all functions of type $\upsilon \Rightarrow j \Rightarrow i \Rightarrow \textit{bool}$ and represents one-place relations (for an urelement a one-place relation evaluates to a truth value in the embedded logic; for an urelement, a state and a possible world it evaluates to a meta-logical truth value).

- Type Π_2 is defined as the set of all functions of type $\upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow \textit{bool}$ and represents two-place relations.

- Type Π_3 is defined as the set of all functions of type $\upsilon \Rightarrow \upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow \textit{bool}$ and represents three-place relations.

- Type α is defined as a synonym of the type of sets of one-place relations Π_1 *set*, i.e. every set of one-place relations constitutes an object of type α. This type represents abstract objects.

- Type ν is defined as **datatype** $\nu = \omega\nu\ \omega \mid \alpha\nu\ \alpha$. This type represents individuals and can be either an ordinary urelement ω or an abstract object α (with the respective type constructors $\omega\nu$ and $\alpha\nu$.

- Type κ is defined as the set of all objects of type ν *option* and represents individual terms. The type *'a option* is part of Isabelle/HOL and consists of a type constructor *Some x* for an object x of type *'a* (in our case type ν) and an additional special element called *None*. *None* is used to represent individual terms that are definite descriptions that do not denote an individual.

Remark 1. *The Isabelle syntax typedef* $\mathrm{o} = \textit{UNIV}::(j \Rightarrow i \Rightarrow \textit{bool})$ *set morphisms evalo makeo .. introduces a new abstract type* o *that is represented by the full set (UNIV) of objects of type* $j \Rightarrow i \Rightarrow \textit{bool}$*. The morphism evalo maps an object of abstract type* o *to its representative of type* $j \Rightarrow i \Rightarrow \textit{bool}$*, whereas the morphism makeo maps an object of type* $j \Rightarrow i \Rightarrow \textit{bool}$ *to the object of type* o *that is represented by it. Defining these abstract types makes it possible to consider the defined types as primitives in later stages of the embedding, once their meta-logical properties are derived from the underlying representation. For a theoretical analysis of the representation layer the type* o *can be considered a synonym of* $j \Rightarrow i \Rightarrow \textit{bool}$*.*

The Isabelle syntax setup-lifting type-definition-o allows definitions for the abstract type o *to be stated directly for its representation type* $j \Rightarrow i \Rightarrow \textit{bool}$ *using the syntax lift-definition.*

In the remainder of this document these morphisms are omitted and definitions are stated directly for the representation types.

1.2 Individual Terms and Definite Descriptions

There are two basic types of individual terms: definite descriptions and individual variables. For any logically proper definite description there is an individual variable that denotes the same object.

In the embedding the type κ encompasses all individual terms, i.e. individual variables *and* definite descriptions. To use a pure individual variable (of type ν) as an object of type κ the decoration $\_^P$ is introduced:

$(x^P) = \mathit{Some}\ x$

The expression x^P (of type κ) is now marked to always be logically proper (it can only be substituted by objects that are internally of the form $\mathit{Some}\ x$) and to always denote the same object as the individual variable x.

It is now possible to define definite descriptions as follows:

$\iota x\ .\ \varphi\ x = (\mathit{if}\ \exists!x.\ \ (\varphi\ x)\ dj\ dw\ \mathbf{then}\ \mathit{Some}\ (\mathit{THE}\ x.\ \ (\varphi\ x)\ dj\ dw)\ \mathit{else}\ \mathit{None})$

If the propriety condition of a definite description $\exists!x.\ \varphi\ x\ dj\ dw$ holds, i.e. *there exists a unique* x*, such that* $\varphi\ x$ *holds for the actual state and the actual world*, the representing individual variable is set to $\mathit{Some}\ (\mathit{THE}\ x.\ \varphi\ x\ dj\ dw)$. Isabelle's *THE* operator evaluates to the unique object, for which the given condition holds, if there is a unique such object, and is undefined otherwise. If the propriety condition does not hold, the individual term is set to *None*.

The following meta-logical functions are defined to aid in handling individual terms:

$\mathit{proper}\ x = (\mathit{None} \neq\ \ x)$

$\mathit{rep}\ x = \mathit{the}\ (\ x)$

the maps an object of type *'a option* that is of the form $\mathit{Some}\ x$ to x and is undefined for *None*. For an object of type κ the expression $\mathit{proper}\ x$ is therefore true, if the term is logically proper, and if this is the case, the expression $\mathit{rep}\ x$ evaluates to the individual of type ν that the term denotes.

1.3 Mapping from abstract objects to special Urelements

To map abstract objects to urelements (for which relations are defined), a constant $\alpha\sigma$ of type $\alpha \Rightarrow \sigma$ is introduced, which maps abstract objects (of type α) to special urelements (of type σ).

To assure that every object in the full domain of urelements actually is an urelement for (one or more) individual objects, the constant $\alpha\sigma$ is axiomatized to be surjective.

1.4 Conversion between objects and Urelements

In order to represent relation exemplification as the function application of the meta-logical representative of a relation, individual variables have to be converted to urelements (see below). In order to define lambda expressions the inverse mapping is defined as well:

$\nu\upsilon \equiv \mathit{case}\text{-}\nu\ \omega\upsilon\ (\sigma\upsilon \circ \alpha\sigma)$

$\upsilon\nu \equiv \mathit{case}\text{-}\upsilon\ \omega\nu\ (\alpha\nu \circ \mathit{inv}\ \alpha\sigma)$

Remark 2. *The Isabelle notation* $\mathit{case}\text{-}\nu$ *is used to define a function acting on objects of type* ν *using the underlying types* ω *and* α*. Every object of type* ν

is (by definition) either of the form $\omega\nu\ x$ *or of the form* $\alpha\nu\ x$*. The expression* $\textit{case-}\nu\ \omega\upsilon\ (\sigma\upsilon \circ \alpha\sigma)$ *for an argument* y *now evaluates to* $\omega\upsilon\ x$ *if* y *is of the form* $\omega\nu\ x$ *and to* $(\sigma\upsilon \circ \alpha\sigma)\ x$ *(i.e.* $\sigma\upsilon\ (\alpha\sigma\ x)$*) if* y *is of the form* $\alpha\nu\ x$*.*

In the definition of $\upsilon\nu$ *the expression* $\textit{inv}\ \alpha\sigma$ *is part of the logic of Isabelle/HOL and defined as some (arbitrary) object in the preimage under* $\alpha\sigma$*, i.e. it holds that* $\alpha\sigma\ (\textit{inv}\ \alpha\sigma\ x) = x$*, as* $\alpha\sigma$ *is axiomatized to be surjective.*

1.5 Exemplification of n-place relations

Exemplification of n-place relations can now be defined. Exemplification of zero-place relations is simply defined as the identity, whereas exemplification of n-place relations for $n \geq 1$ is defined to be true, if all individual terms are logically proper and the function application of the relation to the urelements corresponding to the individuals yields true for a given possible world and state:

- $\llparenthesis p \rrparenthesis = p$
- $\llparenthesis F,x \rrparenthesis = (\lambda w\ s.\ \textit{proper}\ x \wedge F\ (\nu\upsilon\ (\textit{rep}\ x))\ w\ s)$
- $\llparenthesis F,x,y \rrparenthesis = (\lambda w\ s.\ \textit{proper}\ x \wedge \textit{proper}\ y \wedge F\ (\nu\upsilon\ (\textit{rep}\ x))\ (\nu\upsilon\ (\textit{rep}\ y))\ w\ s)$
- $\llparenthesis F,x,y,z \rrparenthesis =$
  $(\lambda w\ s.\ \textit{proper}\ x \wedge$
  $\quad \textit{proper}\ y \wedge \textit{proper}\ z \wedge F\ (\nu\upsilon\ (\textit{rep}\ x))\ (\nu\upsilon\ (\textit{rep}\ y))\ (\nu\upsilon\ (\textit{rep}\ z))\ w\ s)$

1.6 Encoding

Encoding can now be defined as follows:
$\{\!| x,F |\!\} = (\lambda w\ s.\ \textit{proper}\ x \wedge (\mathsf{case}\ \textit{rep}\ x\ \mathsf{of}\ \omega\nu\ \omega \Rightarrow \textit{False} \mid \alpha\nu\ \alpha \Rightarrow F \in \alpha))$
That is for a given state s and a given possible world w it holds that an individual term x encodes F, if x is logically proper, the representative individual variable of x is of the form $\alpha\nu\ \alpha$ for some abstract object α and F is contained in α (remember that abstract objects are defined to be sets of one-place relations). Also note that encoding is a represented as a function of possible worlds and states to ensure type-correctness, but its evaluation does not depend on either.

1.7 Connectives and Quantifiers

The reason to make truth values depend on the additional primitive type of *states* is to achieve hyper-intensionality. The connectives and quantifiers are defined in such a way that they behave classically if evaluated for the designated actual state *dj*, whereas their behavior is governed by uninterpreted constants in any other state.

For this purpose the following uninterpreted constants are introduced:

- *I-NOT* of type $(j \Rightarrow i \Rightarrow \textit{bool}) \Rightarrow j \Rightarrow i \Rightarrow \textit{bool}$
- *I-IMPL* of type $(j \Rightarrow i \Rightarrow \textit{bool}) \Rightarrow (j \Rightarrow i \Rightarrow \textit{bool}) \Rightarrow j \Rightarrow i \Rightarrow \textit{bool}$

Modality is represented using the dependency on primitive possible worlds using a standard Kripke semantics for a S5 modal logic.

The basic connectives and quantifiers are now defined as follows:

- $(\neg p) = (\lambda s\ w.\ s = dj \wedge \neg\ \ p\ dj\ w \vee s \neq dj \wedge \textit{I-NOT}\ (\ p)\ s\ w)$
- $(p \rightarrow q) =$
  $(\lambda s\ w.\ s = dj \wedge (\ p\ dj\ w \longrightarrow\ \ q\ dj\ w) \vee s \neq dj \wedge \textit{I-IMPL}\ (\ p)\ (\ q)\ s\ w)$
- $\forall_{\nu}\ x\ .\ \varphi\ x = (\lambda s\ w.\ \forall x.\ \ (\varphi\ x)\ s\ w)$
- $\forall_{0}\ p\ .\ \varphi\ p = (\lambda s\ w.\ \forall p.\ \ (\varphi\ p)\ s\ w)$
- $\forall_{1}\ F\ .\ \varphi\ F = (\lambda s\ w.\ \forall F.\ \ (\varphi\ F)\ s\ w)$
- $\forall_{2}\ F\ .\ \varphi\ F = (\lambda s\ w.\ \forall F.\ \ (\varphi\ F)\ s\ w)$
- $\forall_{3}\ F\ .\ \varphi\ F = (\lambda s\ w.\ \forall F.\ \ (\varphi\ F)\ s\ w)$
- $(\Box p) = (\lambda s\ w.\ \forall v.\ \ p\ s\ v)$
- $(\boldsymbol{\mathcal{A}}p) = (\lambda s\ w.\ \ p\ dj\ dw)$

Note in particular that the definition of negation and implication behaves classically if evaluated for the actual state $s = dj$, but is governed by the uninterpreted constants *I-NOT* and *I-IMPL* for $s \neq dj$.

1.8 Lambda Expressions

The bound variables of the lambda expressions of the embedded logic are individual variables, whereas relations are represented as functions acting on urelements. Therefore the lambda expressions of the embedded logic are defined as follows:

- $(\boldsymbol{\lambda}^0\ p) =\ \ p$
- $\boldsymbol{\lambda} x.\ \varphi\ x = (\lambda u.\ \ (\varphi\ (\upsilon\nu\ \ u)))$
- $(\boldsymbol{\lambda}^2\ \varphi) = (\lambda u\ v.\ \ (\varphi\ (\upsilon\nu\ \ u)\ (\upsilon\nu\ \ v)))$
- $(\boldsymbol{\lambda}^3\ \varphi) = (\lambda u\ v\ w.\ \ (\varphi\ (\upsilon\nu\ \ u)\ (\upsilon\nu\ \ v)\ (\upsilon\nu\ \ w)))$

Remark 3. *For technical reasons Isabelle only allows lambda expressions for one-place relations to use a nice binder notation. For two- and three-place relations the following notation can be used instead:* $\boldsymbol{\lambda}^2\ (\lambda x\ y.\ \varphi\ x\ y)$, $\boldsymbol{\lambda}^3\ (\lambda x\ y\ z.\ \varphi\ x\ y\ z)$.

The representation of zero-place lambda expressions as the identity is straightforward, the representation of n-place lambda expressions for $n \geq \mathit{1}$ is illustrated for the case $n = \mathit{1}$:

The matrix of the lambda expression φ is a function from individual variables (of type ν) to truth values (of type o, resp. $j \Rightarrow i \Rightarrow \mathit{bool}$). One-place relations are represented as functions of type $\upsilon \Rightarrow j \Rightarrow i \Rightarrow \mathit{bool}$, though, where υ is the type of urelements.

The evaluation of a lambda expression $\boldsymbol{\lambda}x.\ \varphi\ x$ for an urelment u therefore has to be defined as $\varphi\ (\upsilon\nu\ u)$. Remember that $\upsilon\nu$ maps an urelement to some (arbitrary) individual variable in its preimage. Note that this mapping is injective only for ordinary objects, not for abstract objects. The expression $\boldsymbol{\lambda}x.\ \varphi\ x$ only implies *being x, such that there exists some y that is mapped to the same urelement as x, and it holds that $\varphi\ y$.* Conversely, only *for all y that are mapped to the same urelement as x it holds that $\varphi\ y$* is a sufficient condition to conclude that x exemplifies $\boldsymbol{\lambda}x.\ \varphi\ x$.

Remark 4. *Formally the following statements hold, where $[p\ in\ v]$ is the evaluation of the formula p in the embedded logic to its meta-logical representation for a possible world v (and the actual state dj, for details refer to the next subsection):*

- $[\llparenthesis\boldsymbol{\lambda}x.\ \varphi\ x, x^P\rrparenthesis\ in\ v] \longrightarrow (\exists\, y.\ \nu\upsilon\ y = \nu\upsilon\ x \longrightarrow\ (\varphi\ y)\ dj\ v)$
- $(\forall\, y.\ \nu\upsilon\ y = \nu\upsilon\ x \longrightarrow\ (\varphi\ y)\ dj\ v) \longrightarrow [\llparenthesis\boldsymbol{\lambda}x.\ \varphi\ x, x^P\rrparenthesis\ in\ v]$

Principia defines lambda expressions only for propositional formulas, though, i.e. for formulas that do *not* contain encoding subformulas. The only other kind of formulas in which the bound variable x could be used in the matrix φ, however, are exemplication subformulas, which are defined to only depend on urelmements. Consider the following simple lambda-expression and the evaluation to its meta-logical representation:

$$\boldsymbol{\lambda}x.\ \llparenthesis F, x^P\rrparenthesis\ =\ (\lambda u.\ \ F\ (\nu\upsilon\ (\upsilon\nu\ u)))$$

Further note that the following identity holds: $\nu\upsilon\ (\upsilon\nu\ u) = u$ and therefore $\boldsymbol{\lambda}x.\ \llparenthesis F, x^P\rrparenthesis = F$, as desired.

Therefore the defined lambda-expressions can accurately represent the lambda-expressions of the Principia. However the embedding still allows for lambda expressions that contain encoding subformulas. $\llparenthesis\boldsymbol{\lambda}x.\ \lbrace\!|x^P, F|\!\rbrace, y^P\rrparenthesis$ does *not* imply $\lbrace\!|y^P, F|\!\rbrace$, but only that there exists an abstract object z, that is mapped to the same urelement as x and it holds that *embedded-style* $\lbrace\!|z^P,\ F|\!\rbrace$. The former would lead to well-known inconsistencies, which the latter avoids.

Remark 5. *Formally the following statements are true:*

- $[\llparenthesis\boldsymbol{\lambda}x.\ \lbrace\!|x^P, F|\!\rbrace, x^P\rrparenthesis\ in\ v] \longrightarrow (\exists\, y.\ \nu\upsilon\ y = \nu\upsilon\ x \wedge [\lbrace\!|y^P, F|\!\rbrace\ in\ v])$
- $(\forall\, y.\ \nu\upsilon\ y = \nu\upsilon\ x \longrightarrow [\lbrace\!|y^P, F|\!\rbrace\ in\ v]) \longrightarrow [\llparenthesis\boldsymbol{\lambda}x.\ \lbrace\!|x^P, F|\!\rbrace, x^P\rrparenthesis\ in\ v]$

An example of a statement containing lambda-expressions that contain encoding subformulas that becomes derivable using the meta-logic is the following: $[\forall\ F\ y.\ \llparenthesis\boldsymbol{\lambda}x.\ \lbrace\!|x^P, F|\!\rbrace \equiv \lbrace\!|x^P, F|\!\rbrace, y^P\rrparenthesis\ in\ v]$

1.9 Validity

A formula is considered semantically valid for a possible world v if it evaluates to *True* for the actual state dj and the given possible world v. Semantic validity is defined as follows:

$[\varphi \ \mathit{in} \ v] = \ \varphi \ \mathit{dj} \ v$

This way the truth evaluation of a proposition only depends on the evaluation of its representation for the actual state *dj*. Remember that for the actual state the connectives and quantifiers are defined to behave classically. In fact the only formulas of the embedded logic whose truth evaluation *does* depend on all states are formulas containing encoding subformulas.

1.10 Concreteness

Principia defines concreteness as a one-place relation constant. For the embedding care has to be taken that concreteness actually matches the primitive distinction between ordinary and abstract objects. The following requirements have to be satisfied by the introduced notion of concreteness:

- Ordinary objects are possibly concrete. In the meta-logic this means that for every ordinary object there exists at least one possible world, in which the object is concrete.
- Abstract objects are never concrete.

An additional requirement is enforced by axiom (32.4)[2]. To satisfy this axiom the following has to be assured:

- Possibly contingent objects exist. In the meta-logic this means that there exists an ordinary object and two possible worlds, such that the ordinary object is concrete in one of the worlds, but not concrete in the other.
- Possibly no contingent objects exist. In the meta-logic this means that there exists a possible world, such that all objects that are concrete in this world, are concrete in all possible worlds.

In order to satisfy these requirements a constant *ConcreteInWorld* is introduced, that maps ordinary objects (of type ω) and possible worlds (of type i) to meta-logical truth values (of type *bool*). This constant is axiomatized in the following way:

- $\forall x. \ \exists v. \ \mathit{ConcreteInWorld} \ x \ v$
- $\exists x \ v. \ \mathit{ConcreteInWorld} \ x \ v \wedge (\exists w. \ \neg \ \mathit{ConcreteInWorld} \ x \ w)$
- $\exists w. \ \forall x. \ \mathit{ConcreteInWorld} \ x \ w \longrightarrow (\forall v. \ \mathit{ConcreteInWorld} \ x \ v)$

Concreteness can now be defined as a one-place relation:

$E! = (\lambda u \ s \ w. \ \mathsf{case} \ u \ \mathsf{of} \ \omega\upsilon \ x \Rightarrow \mathit{ConcreteInWorld} \ x \ w \mid \sigma\upsilon \ \sigma \Rightarrow \mathit{False})$

The equivalence of the axioms stated in the meta-logic and the notion of concreteness in Principia can now be verified:

- $(\forall x. \ \exists v. \ \mathit{ConcreteInWorld} \ x \ v) =$ $(\forall y. \ [(\!|\boldsymbol{\lambda} u. \ \neg\Box\neg(\!|E!,u^P|\!),y^P|\!) \ \mathit{in} \ v] = (\mathsf{case} \ y \ \mathsf{of} \ \omega\nu \ z \Rightarrow \mathit{True} \mid \alpha\nu \ z \Rightarrow \mathit{False}))$

- $(\forall x.\ \exists v.\ \mathit{ConcreteInWorld}\ x\ v) =$
  $(\forall y.\ [(\!|\boldsymbol{\lambda} u.\ \Box\neg(\!|E!,u^P|\!),y^P|\!)\ \mathit{in}\ v] = (\mathsf{case}\ y\ \mathsf{of}\ \omega\nu\ z \Rightarrow \mathit{False}\ |\ \alpha\nu\ z \Rightarrow \mathit{True}))$
- $(\exists x\ v.\ \mathit{ConcreteInWorld}\ x\ v \wedge (\exists w.\ \neg\ \mathit{ConcreteInWorld}\ x\ w)) =$
  $[\neg\Box(\forall x.\ (\!|E!,x^P|\!) \rightarrow \Box(\!|E!,x^P|\!))\ \mathit{in}\ v]$
- $(\exists w.\ \forall x.\ \mathit{ConcreteInWorld}\ x\ w \longrightarrow (\forall v.\ \mathit{ConcreteInWorld}\ x\ v)) =$
  $[\neg\Box\neg(\forall x.\ (\!|E!,x^P|\!) \rightarrow \Box(\!|E!,x^P|\!))\ \mathit{in}\ v]$

References

[1] T. Nipkow. What's in main. http://isabelle.in.tum.de/doc/main.pdf. [accessed: March 13, 2017].

[2] E. N. Zalta. Principia logico-metaphysica. http://mally.stanford.edu/principia.pdf. [Draft/Excerpt; accessed: October 28, 2016].