Freie Universität Berlin

Master's thesis at the institute of mathematics at Freie Universität Berlin

An Embedding of the Theory of Abstract Objects in Isabelle/HOL

Daniel Kirchner

Matrikelnummer: 4387161

Supervisor:
PD Dr. Christoph Benzmüller

Berlin, May 27, 2017

Abstract

We present an embedding of the second order fragment of the Theory of Abstract Objects as described in Edward Zalta's upcoming work Principia Logico-Metaphysica (PLM[12]) in the automated reasoning framework Isabelle/HOL. The Theory of Abstract Objects is a metaphysical theory that reifies property patterns, as they for example occur in the abstract reasoning of mathematics, as *abstract objects* and provides an axiomatic framework that allows to reason about these objects. It thereby serves as a fundamental metaphysical theory that can be used to axiomatize and describe a wide range of philosophical objects, such as Platonic forms or Leibniz' concepts, and has the ambition to function as a foundational theory of mathematics. The target theory of our embedding as described in chapters 7-9 of PLM[12] employs a modal relational type theory as logical foundation for which a representation in functional type theory is known to be challenging[8].

Nevertheless we arrive at a functioning representation of the theory in the functional logic of Isabelle/HOL based on a semantical representation of an Aczel model of the theory. Based on this representation we construct an implementation of the deductive system of PLM ([12, Chap. 9]) which allows it to automatically and interactively find and verify theorems of PLM.

Our work thereby supports the concept of shallow semantical embeddings of logical systems in HOL as a universal tool for logical reasoning as promoted by Christoph Benzmüller[1].

The most notable result of the presented work is the discovery of a previously unknown paradox in the formulation of the Theory of Abstract Objects at the time of writing. The embedding of the theory in Isabelle/HOL played a vital part in this discovery. Furthermore it was possible to immediately offer several options to modify the theory to guarantee its consistency. Thereby our work could provide a significant contribution to the development of a proper grounding for object theory.

Contents

1. Introduction

Calculemus!

Leibniz

1.1. Universal Logical Reasoning[1]

The concept of understanding rational argumentation and reasoning using formal logical systems has a long tradition and can already be found in the study of syllogistic arguments by Aristotle. Since then a large variety of formal systems has evolved, each using different syntactical and semantical structures to capture specific aspects of logical reasoning (e.g. propositional logic, first-order/higher-order logic, modal logic, free logic, etc.). This diversity of formal systems gives rise to the question, whether a *universal* logic can be devised, that would be capable of expressing statements of all existing specialized logical systems and provide a basis for meta-logical considerations like the equivalence of or relations between those systems.

The idea of a universal logical framework is very prominent in the works of Gottfried Wilhelm Leibniz (1646-1716) with his concept of a *characteristica universalis*, i.e. a universal formal language able to express metaphysical, scientific and mathematical concepts. Based thereupon he envisioned the *calculus ratiocinator*, a universal logical calculus with which the truth of statements formulated in the characteristica universalis could be decided purely by formal calculation and thereby in an automated fashion, an idea that became famous under the slogan: *Calculemus!*

Nowadays with the rise of powerful computer systems such a universal logical framework could have repercussions throughout the sciences and may be a vital part of machine-computer interaction in the future. Leibniz' ideas have inspired recent efforts to use functional higher-order logic (HOL) as such a universal logical language and to represent various logical systems by the use of *shallow semantical embeddings*[1].

Notably this approach received attention due to the formalisation, validation and analysis of Gödel's ontological proof of the existence of God by Christoph Benzmüller[5], for which higher-order modal logic was embedded in the computerized logic framework Isabelle/HOL.

[1] This introductory section is based on the description of the topic in [1].

1.2. Shallow Semantical Embeddings in HOL

A semantic embedding of a target logical system defines the syntactical elements of the target language in a background logic (e.g. in a framework like Isabelle/HOL) based on their semantics. This way the background logic can be used to argue about the semantic truth of syntactic statements in the embedded logic.

A *deep* embedding represents the complete syntactical structure of the target language separately from the background logic, i.e. every term, variable symbol, connective, etc. of the target language is represented as a syntactical object and then the background logic is used to evaluate a syntactic expression by quantifying over all models that can be associated with the syntax. Variable symbols of the target logic for instance would be represented as constants in the background logic and a proposition would be considered semantically valid if it holds for all possible denotations an interpretation function can assign to these variables.

While this approach will work for most target logics, it has several drawbacks. It is likely that there are principles that are shared between the target logic and the background logic, such as α-conversion for λ-expressions or the equivalence of terms with renamed variables in general. In a deep embedding these principles usually have to be explicitly shown to hold for the syntactic representation of the target logic, which is usually connected with significant complexity. Furthermore if the framework used for the background logic allows automated reasoning, the degree of automation that can be achieved in the embedded logic is limited, as any reasoning in the target logic will have to consider the meta-logical evaluation process in the background logic that will usually be complex.

A *shallow* embedding uses a different approach based on the idea that most contemporary logical systems are semantically characterized by the means of set theory. A shallow embedding defines primitive syntactical objects of the target language such as variables or propositions using a set theoretic representation. For example propositions in a modal logic can be represented as functions from possible worlds to truth values in a non-modal logic.

The shallow embedding aims to equationally define only the syntactical elements of the target logic that are not already present in the background logic or whose semantics behaves differently than in the background logic, while preserving as much of the logical structure of the background logic as possible. The modal box operator for example can be represented as a quantification over all possible worlds satisfying an accessibility relation, while negation and quantification can be directly represented using the negation and quantification of the background logic (preserving the dependency on possible worlds).

This way basic principles of the background logic (such as alpha conversion) can often be directly applied to the embedded logic and the equational, definitional nature of the representation preserves a larger degree of automation. Furthermore axioms in the embedded logic can often be equivalently stated in the background logic, which makes the construction of models for the system easier and again increases the degree of automation that can be retained.

The shallow semantical embedding of modal logic was the basis for the analysis of Gödel's onthological argument[5] and the general concept has shown great potential as a universal tool for logical embeddings while retaining the existing infrastructure for automation as for example present in a framework like Isabelle/HOL[2].

1.3. Relational Type Theory vs. Functional Type Theory

The universality of this approach has since been challenged by Paul Oppenheimer and Edward Zalta who argue in the paper *Relations Versus Functions at the Foundations of Logic: Type-Theoretic Considerations*[8] that relational type theory is more general than functional type theory. In particular they argue that the Theory of Abstract Objects, which is founded in relational type theory, can not be properly characterized in functional type theory.

This has led to the question whether a shallow semantical embedding of the Theory of Abstract Objects in a functional logic framework like Isabelle/HOL is at all possible, which is the core question the work presented here attempts to examine and partially answer.

One of their main arguments is that unrestricted λ-expressions as present in functional type theory lead to an inconsistency when combined with one of the axioms of the theory and indeed it has been shown for early attempts on embedding the theory that despite significant efforts to avoid the aforementioned inconsistency by excluding problematic λ-expressions in the embedded logic, it could still be reproduced using an appropriate construction in the background logic[3].

The solution presented here circumvents this problem by identifying λ-expressions as one element of the target language that behaves differently than their counterparts in the background logic and consequently by representing λ-expressions of the target logic using a new *defined* kind of λ-expressions. This forces λ-expressions in the embedded logic to have a particular semantics that is inspired by the *Aczel-model* of the target theory (see 2.6) and avoids prior inconsistencies. The mentioned issue and the employed solution is discussed in more detail in sections 3.2 and 3.4.7.

[2]See [1] for an overview and an description of the ambitions of the approach.

[3] Early attempts of an embedding by Christoph Benzmüller (see https://github.com/cbenzmueller/PrincipiaMetaphysica) were discussed in his university lecture *Computational Metaphysics* (FU Berlin, SS2016) and the proof of their inconsistency in the author's final project for the course inspired the continued research in this master's thesis.

1.4. Overview of the following Chapters

The following chapters are structured as follows:

- The second chapter gives an overview of the motivation and structure of the target theory of the embedding, the Theory of Abstract Objects. It also introduces the *Aczel-model* of the theory, that was adapted as the basis for the embedding.
- The third chapter is a detailed documentation of the concepts and technical structure of the embedding. This chapter references the Isabelle theory that can be found in the appendix.
- The fourth chapter consists of a technical discussion about some of the issues encountered during the construction of the embedding due to limitations of the logical framework of Isabelle/HOL and the solutions that were employed.
- The last chapter discusses the relation between the embedding and the target theory of PLM and describes some of the results achieved using the embedding. Furthermore it states some open questions for future research.

This entire document is generated from an Isabelle theory file and thereby in particular all formal statements in the third chapter are well-formed terms, resp. verified valid theorems in the constructed embedding unless the contrary is stated explicitly.

2. The Theory of Abstract Objects

> It is widely supposed that every entity falls into one of two categories: Some are concrete; the rest abstract. The distinction is supposed to be of fundamental significance for metaphysics and epistemology.
>
> *Stanford Encyclopedia of Philosophy*[9]

2.1. Motivation

As the name suggests the Theory of Abstract Objects revolves around *abstract objects* and is thereby a metaphysical theory. As Zalta puts it: "Whereas physics attempts a systematic description of fundamental and complex concrete objects, metaphysics attempts a systematic description of fundamental and complex abstract objects. [...] The theory of abstract objects attempts to organize these objects within a systematic and axiomatic framework. [...] [We can] think of abstract objects as possible and actual property-patterns. [...] Our theory of abstract objects will *objectify* or *reify* the group of properties satisfying [such a] pattern."[13][1]

So what is the fundamental distinction between abstract and concrete objects? The analysis in the Theory of Abstract Objects is based on a distinction between two fundamental modes of predication that is based on the ideas of Ernst Mally. Whereas objects that are concrete (the Theory of Abstract Objects calls them *ordinary objects*) are characterized by the classical mode of predication, i.e. *exemplification*, a second mode of predication is introduced that is reserved for abstract objects. This new mode of predication is called *encoding* and formally written as xF (x *encodes* F) in contrast to Fx (x *exemplifies* F).

Mally informally introduces this second mode of predication in order to represent sentences about fictional objects. In his thinking only concrete objects, that for example have a fixed spatiotemporal location, a body and shape, etc., can *exemplify* their properties and are characterized by the properties they exemplify. Sentences about fictional objects such as "Sherlock Holmes is a detective" have a different meaning. Stating that "Sherlock Holmes is a detective" does not imply that there is some concrete object that is

[1] The introduction to the theory in this and the next section is based on the documentation of the theory in [13], which is paraphrased and summarized throughout the sections. Further references about the topic include [12], [11], [10].

Sherlock Holmes and this object exemplifies the property of being a detective - it rather states that the concept we have of the fictional character Sherlock Holmes includes the property of being a detective. Sherlock Holmes is not concrete, but an abstract object that is *determined* by the properties Sherlock Holmes is given by the fictional works involving him as character. This is expressed using the second mode of predication *Sherlock Holmes encodes the property of being a detective.*

To clarify the difference between the two concepts note that any object either exemplifies a property or its negation. The same is not true for encoding. For example it is not determinate whether Sherlock Holmes has a mole on his left foot. Therefore the abstract object Sherlock Holmes neither encodes the property of having a mole on his left foot, nor the property of not having a mole on his left foot.

The theory even allows for an abstract object to encode properties that no object could possibly exemplify and reason about them, for example the quadratic circle. In classical logic meaningful reasoning about a quadratic circle is impossible - as soon as I suppose that an object *exemplifies* the properties of being a circle and of being quadratic, this will lead to a contradiction and every statement becomes derivable.

In the Theory of Abstract Objects on the other hand there is an abstract object that encodes exactly these two properties and it is possible to reason about it. For example we can state that this object *exemplifies* the property of *being thought about by the reader of this paragraph.* This shows that the Theory of Abstract Objects provides the means to reason about processes of human thought in a much broader sense than classical logic would allow.

It turns out that by the means of abstract objects and encoding the Theory of Abstract Objects can be used to represent and reason about a large variety of concepts that regularly occur in philosophy, mathematics or linguistics.

In [13] the principal objectives of the theory are summerized as follows:

- To describe the logic underlying (scientific) thought and reasoning by extending classical propositional, predicate, and modal logic.
- To describe the laws governing universal entities such as properties, relations, and propositions (i.e., states of affairs).
- To identify *theoretical* mathematical objects and relations as well as the *natural* mathematical objects such as natural numbers and natural sets.
- To analyze the distinction between fact and fiction and systematize the various relationships between stories, characters, and other fictional objects.
- To systematize our modal thoughts about possible (actual, necessary) objects, states of affairs, situations and worlds.
- To account for the deviant logic of propositional attitude reports, explain the informativeness of identity statements, and give a general account of the objective and cognitive content of natural language.
- To axiomatize philosophical objects postulated by other philosophers, such as Forms (Plato), concepts (Leibniz), monads (Leibniz), possible worlds (Leibniz),

nonexistent objects (Meinong), senses (Frege), extensions of concepts (Frege), noematic senses (Husserl), the world as a state of affairs (early Wittgenstein), moments of time, etc.

The Theory of Abstract Objects has therefore the ambition and the potential to serve as a foundational theory of metaphysics as well as mathematics and can provide a simple unified axiomatic framework that allows reasoning about a huge variety of concepts throughout the sciences. This makes the attempt to represent the theory using the universal reasoning approach of shallow semantical embeddings outlined in the previous chapter particularly challenging and at the same time rewarding, if successful.

A successful implementation of the theory that allows it to utilize the existing sophisticated infrastructure for automated reasoning present in a framework like Isabelle/HOL would not only strongly support the applicability of shallow semantical embeddings as a universal reasoning tool, but could also aid in spreading the utilization of the theory itself as a foundational theory for various scientific fields by enabling convenient interactive and automated reasoning in a verified framework.

Although the embedding revealed certain challenges in this approach and there remain open questions for example about the precise relationship between the embedding and the target theory or its soundness and completeness, it is safe to say that it represents a significant step towards achieving this goal.

2.2. Basic Principles

Although the formal language of the theory is introduced only in the next section, some of the basic concepts of the theory are presented in advance to provide further motivation for the formalism.

The following are the two most important principles of the theory[13]:

- $\exists\, x(A!x \;\&\; \forall\, F(xF \equiv \varphi))$
- $x = y \equiv \Box\forall\, F(xF \equiv yF)$

The first statement asserts that for every condition on properties φ there exists an abstract object that encodes exactly those properties satisfying φ, whereas the second statement holds for two abstract objects x and y and states that they are equal, if and only if they necessarily encode the same properties.

Together these two principles clarify the notion of abstract objects as the reification of property patterns: Any set of properties is objectified as a distinct abstract object.

Note that these principles already allow it to postulate interesting abstract objects.

For example the Leibnizian concept of an (ordinary) individual u can be defined as *the (unique) abstract object that encodes all properties that* u *exemplifies*, formally: $\iota x\; A!x \;\&\; \forall\, F\; (xF \equiv Fu)$

Other interesting examples include possible worlds, Platonic Forms or even basic logical objects like truth values. Here it is important to note that the theory allows it to formulate a purely *syntactic* definition of objects like possible worlds and truth values and from these syntactic definitions it can be *derived* that there are two truth values or that the application of the modal box operator to a proposition is equivalent to the proposition being true in all possible worlds (where *being true in a possible world* is again defined syntactically).

This is an impressive property of the Theory of Abstract Objects: it can *syntactically* define objects that are usually only considered semantically.

2.3. The Language of PLM

The target of the embedding is the second order fragment of object theory as described in chapter 7 of Edward Zalta's upcoming *Principia Logico-Metaphysica* (PLM)[12]. The logical foundation of the theory uses a second-order modal logic (without primitive identity) formulated using relational type theory that is modified to admit *encoding* as a second mode of predication besides the traditional *exemplification*. In the following an informal description of the important aspects of the language is provided; for a detailed and fully formal description and the type-theoretic background refer to the respective chapters of PLM[12].

A compact description of the language can be given in Backus-Naur Form (BNF)[12, p. 170], as shown in figure 2.1, in which the following grammatical categories are used:

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

It is important to note that the language distinguishes between two types of basic formulas, namely (non-propositional) *formulas* that *may* contain encoding subformulas and *propositional formulas* that *may not* contain encoding subformulas. Only propositional formulas may be used in λ-expressions. The main reason for this distinction will be explained in section 3.2.

Note that there is a case in which propositional formulas *can* contain encoding expressions. This is due to the fact that *subformula* is defined in such a way[2] that xQ is *not* a subformula of $\iota x(xQ)$. Thereby $F\iota x(xQ)$ is a propositional formula and $[\lambda y\ F\iota x(xQ)]$ a well-formed λ-expression. On the other hand xF is not a propositional formula and therefore $[\lambda x\ xF]$ not a well-formed λ-expression. This fact will become relevant in the

[2] For a formal definition of subformula refer to definition (8) in [12].

Figure 2.1.: BNF grammar of the language of PLM[12, p. 170]

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

discussion in section 5.2, that describes a paradox in the formulation of the theory in the draft of PLM at the time of writing[3].

Furthermore the theory contains a designated relation constant $E!$ to be read as *being concrete*. Using this constant the distinction between ordinary and abstract objects is defined as follows:

- $O! =_{df} [\lambda x \; \Diamond E!x]$
- $A! =_{df} [\lambda x \; \neg\Diamond E!x]$

So ordinary objects are possibly concrete, whereas abstract objects cannot possibly be concrete.

It is important to note that the language does not contain the identity as primitive. Instead the language uses *defined* identities as follows:

ordinary objects	$x =_E y =_{df} O!x \;\&\; O!y \;\&\; \Box(\forall F \; Fx \equiv Fy)$
individuals	$x = y =_{df} x =_E y \vee (A!x \;\&\; A!y \;\&\; \Box(\forall F \; xF \equiv yF))$
one-place relations	$F^1 = G^1 =_{df} \Box(\forall x \; xF^1 \equiv xG^1)$
zero-place relations	$F^0 = G^0 =_{df} [\lambda y \; F^0] = [\lambda y \; G^0]$

The identity for n-place relations for $n \geq 2$ is defined in terms of the identity of one-place relations, see (16)[12] for the full details.

The identity for ordinary objects follows Leibniz' law of the identity of indiscernibles: Two ordinary objects that necessarily exemplify the same properties are identical. Abstract objects, however, are only identical if they necessarily *encode* the same properties. As mentioned in the previous section this goes along with the concept of abstract objects as the reification of property patterns. Notably the identity for properties has a different definition than one would expect from classical logic. Classically two properties are

[3] At the time of writing several options are being considered that can restore the consistency of the theory while retaining all theorems of PLM.

considered identical if and only if they necessarily are *exemplified* by the same objects. The Theory of Abstract Objects, however, defines two properties to be identical if and only if they are necessarily *encoded* by the same (abstract) objects. This has some interesting consequences that will be described in more detail in section 2.5 that describes the *hyperintensionality* of relations in the theory.

2.4. The Axioms

Based on the language above an axiom system is defined that constructs a S5 modal logic with an actuality operator, axioms for definite descriptions that go along with Russell's analysis of descriptions, the substitution of identicals as per the defined identity, α-, β-, η- and a special ι-conversion for λ-expressions, as well as dedicated axioms for encoding. A full accounting of the axioms in their representation in the embedding is found in section 3.10. For the original axioms refer to [12, Chap. 8]. At this point the axioms of encoding are the most relevant, namely:

- $xF \rightarrow \Box xF$
- $O!x \rightarrow \neg\exists\, F\; xF$
- $\exists\, x\; (A!x\; \&\; \forall\, F\; (xF \equiv \varphi))$,
  provided x doesn't occur free in φ

So encoding is modally rigid, ordinary objects do not encode properties and most importantly the comprehension axiom for abstract objects that was already mentioned above:
For every condition on properties φ there exists an abstract object, that encodes exactly those properties, that satisfy φ.

2.5. Hyperintensionality of Relations

An interesting property of the Theory of Abstract Objects results from the definition of identity for one-place relations. Recall that two properties are defined to be identical if and only if they are *encoded* by the same (abstract) objects. The theory imposes no restrictions whatsoever on which properties an abstract object encodes. Let for example F be the property *being the morning star* and G be the property *being the evening star*. Since the morning star and the evening star are actually both the planet Venus, every object that *exemplifies* F will also *exemplify* G and vice-versa: $\Box\forall\, x\; Fx \equiv Gx$. However the concept of being the morning star is different from the concept of being the evening star. The Theory of Abstract Object therefore does not prohibit the existence of an abstract object that *encodes* F, but does *not* encode G. Therefore by the definition of identity for properties it does *not* hold that $F = G$. As a matter of fact the Theory of Abstract Object does not force $F = G$ for any F and G. It rather stipulates what needs to be proven, if $F = G$ is to be established, namely that they are necessarily encoded by the same objects. Therefore if two properties *should* be equal in some context an axiom has to be added to the theory that allows it to prove that both properties are encoded by the same abstract objects.

To understand the extent of this *hyperintensionality* of the theory consider that the following are *not* necessarily equal in object theory:

$$[\lambda y\ p \vee \neg p] \text{ and } [\lambda y\ q \vee \neg q]$$
$$[\lambda y\ p\ \&\ q] \text{ and } [\lambda y\ q\ \&\ p]$$

Of course the theory can be extended in such a way that these properties are equal, namely by introducing a new axiom that requires that they are necessarily encoded by the same abstract objects. Without additional axioms, however, it is not provable that above properties are equal.

It is important to note that the theory is only hyperintensional in exactly the described sense (i.e. relations are intensional entities). Propositional reasoning is still governed by classical extensionality. For example properties that are necessarily exemplified by the same objects can be substituted for each other in an exemplification formula, the law of the excluded middle can be used in propositional reasoning, etc.

The Theory of Abstract Objects is an *extensional* theory of *intensional* entities[12, (130)].

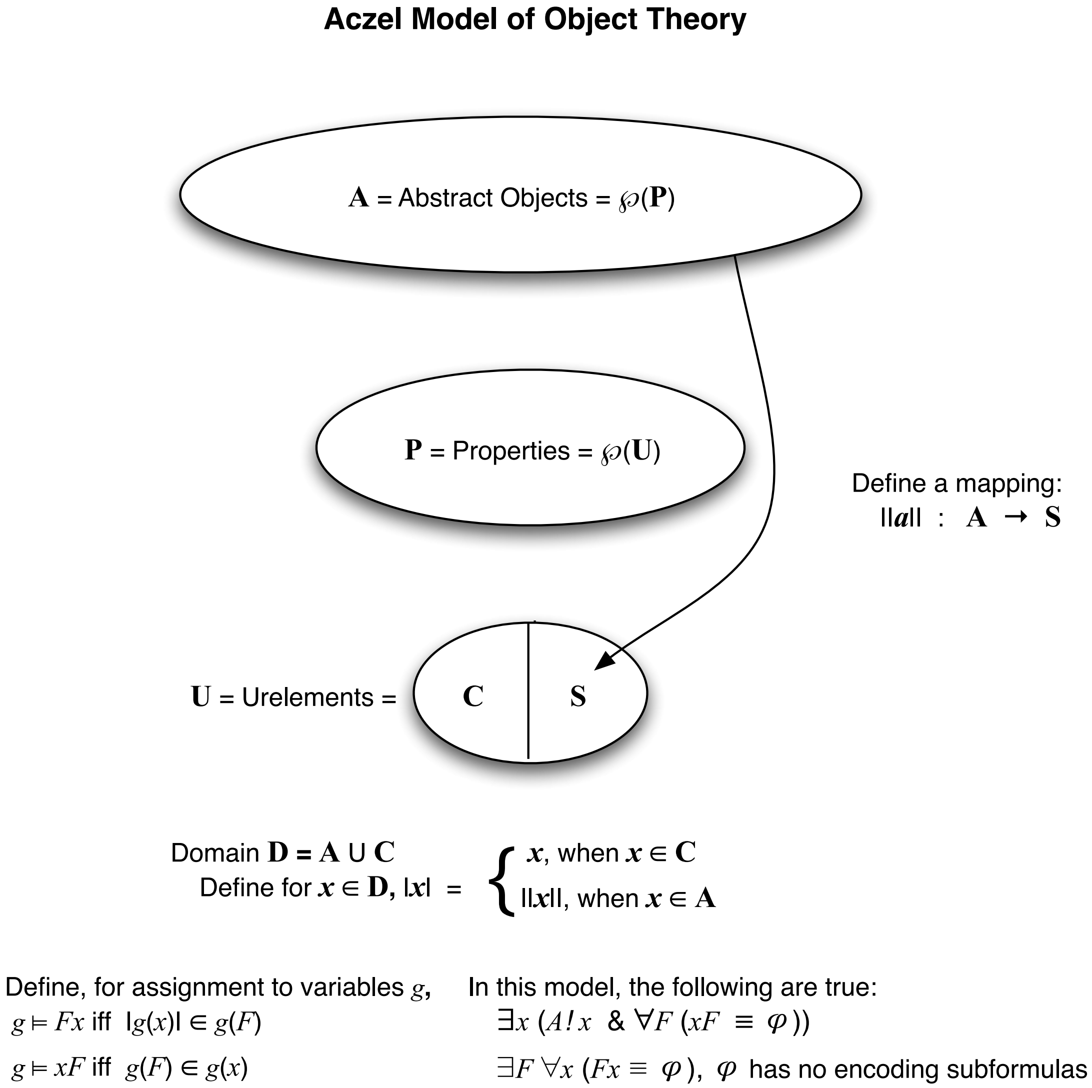
2.6. The Aczel-Model

When thinking about a model for the theory one will quickly notice the following problem: The comprehension axiom for abstract objects implies that for each set of properties there exists an abstract object encoding exactly those properties. Considering the definition of identity there therefore exists an injective map from the power set of properties to the set of abstract objects. On the other hand for an object y the term $[\lambda x\ Rxy]$ constitutes a property. If for distinct objects these properties were distinct, this would result in a violation of Cantor's theorem, since this would mean that there is an injective map from the power set of properties to the set of properties. So does the Theory of Abstract Objects as constructed above have a model? An answer to this question was provided by Peter Aczel[4] who proposed the model structure illustrated in figure 2.2.

In the Aczel-model abstract objects are represented by sets of properties. This of course validates the comprehension axiom of abstract objects. Properties on the other hand are not naively represented by sets of objects, which would lead to a violation of Cantor's theorem, but rather as the sets of *urelements*. Urelements are partitioned into two groups, ordinary urelements (C in the illustration) and special urelements (S in the illustration). Ordinary urelements can serve as the denotations of ordinary objects. Every abstract object on the other hand has a special urelement as its proxy. Which properties an abstract object exemplifies solely depends on its proxy. However, the map from abstract objects to special urelements is not injective; more than one abstract object

[4] In fact to our knowledge Dana Scott proposed a first model for the theory before Peter Aczel that we believe is a special case of an Aczel-model with only one *special urelement*.

Figure 2.2.: Illustration of the Aczel-Model, courtesy of Edward Zalta



can share the same proxy. This way a violation of Cantor's theorem is avoided. As a consequence there are abstract objects, that cannot be distinguished by the properties they exemplify. Interestingly the existence of abstract objects that are exemplification-indistinguishable is a theorem of PLM, see (197)[12].

Note that although the Aczel-model illustrated in figure 2.2 is non-modal, the extension to a modal version is straightforward by introducing primitive possible worlds as in the Kripke semantics of modal logic.

Further note that relations in the Aczel-model are *extensional*. Since properties are represented as the power set of urelements, two properties are in fact equal if they are exemplified by the same objects. This has no bearing on the soundness of the Aczel-model as a model for the Theory of Abstract Objects, but it has the consequence, that statements like $[\lambda\ p \vee \neg p] = [\lambda\ q \vee \neg q]$ are true in the model, although they are not derivable from the axioms of object theory as explained in the previous section.

For this reason an *intensional* variant of the Aczel-model is developed and used as the basis of the embedding. The technicalities of this model are described in the next chapter (see 3.3.1).

3. The Embedding

3.1. The Framework Isabelle/HOL

The embedding is implemented in Isabelle/HOL, that provides a functional higher-order logic that serves as meta-logic. An introduction to Isabelle/HOL can be found in [7][1]. For a general introduction to HOL and its automatization refer to [2].

The Isabelle theory containing the embedding is included in the appendix and documented in this chapter. Throughout the chapter references to the various sections of the appendix can be found.

This document itself is generated from a separate Isabelle theory that imports the complete embedding. The terms and theorems discussed throughout this chapter (starting from 3.4) are well-formed terms or valid theorems in the embedding, unless the contrary is stated explicitly. Furthermore the *pretty printing* facility of Isabelle's document generation has been utilized to make it easier to distinguish between the embedded logic and the meta-logic: all expressions that belong to the embedded logic are printed in blue color throughout the chapter.

For technical reasons this color coding could not be used for the raw Isabelle theory in the appendix. Still note the use of bold print for the quantifiers and connectives of the embedded logic.

3.2. A Russell-style Paradox

One of the major challenges of an implementation of the Theory of Abstract Objects in functional logic is the fact that a naive representation of the λ-expressions of the theory using the unrestricted, β-convertible λ-expressions of functional logic results in the following paradox (see [8, pp. 24-25]):

Assume $[\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]$ were a valid λ-expression denoting a relation. Now the comprehension axiom of abstract objects requires the following:

$$\exists x\ (A!x\ \&\ \forall F\ (xF \equiv F = [\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]))$$

So there is an abstract object that encodes only the property $[\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]$. Let b be such an object. Now first assume b exemplifies $[\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]$. By β-reduction this implies that there exists a property, that b encodes, but does not exemplify. Since

[1] An updated version is available at http://isabelle.in.tum.de/doc/tutorial.pdf or in the documentation of the current Isabelle release, see http://isabelle.in.tum.de/.

b only encodes $[\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]$, but does also exemplify it by assumption this is a contradiction.

Now assume b does not exemplify $[\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]$. By β-reduction it follows that there does not exist a property that b encodes, but does not exemplify. Since b encodes $[\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]$ by construction and does not exemplify it by assumption this is again a contradiction.

This paradox is prevented in the formulation of object theory by disallowing encoding subformulas in λ-expressions, so in particular $[\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]$ is not part of the language. However during the construction of the embedding it was discovered that this restriction is not sufficient to prevent paradoxes in general. This is discussed in section 5.2. The solution used in the embedding is described in section 3.4.7.

3.3. Basic Concepts

The introduction mentioned that shallow semantical embeddings were used to successfully represent different varieties of modal logic by implementing them using Kripke semantics. The advantage here is that Kripke semantics is well understood and there are extensive results about its completeness that can be utilized in the analysis of semantical embeddings[3].

For the Theory of Abstract Objects the situation is different. Although it is believed that Aczel-models are sound, section 2.6 already established that even a modal version of the traditional Aczel-model is extensional and therefore theorems are true in it, that are not derivable from the axioms of object theory. On the other hand the last section showed that care has to be taken to ensure the consistency of an embedding of the theory in functional logic.

For this reason the embedding first constructs a hyperintensional version of the Aczel-model as a consistent basis and then abstracts away from its technicalities using a layered reasoning approach. These concepts are described in more detail in the following sections.

3.3.1. Hyperintensional Aczel-model

As mentioned in section 2.6 it is straightforward to extend the traditional (non-modal) Aczel-model to a modal version by introducing primitive possible worlds following the Kripke semantics for a modal S5 logic.

Relations in the resulting Aczel-model are, however, still *extensional*. Two relations that are necessarily exemplified by the same objects are equal. The Aczel-model that is used as the basis for the embedding therefore introduces *states* as another primitive besides possible worlds. Truth values are represented as ternary functions from states and possible worlds to booleans; relations as functions from urelements, states and possible worlds to booleans.

Abstract objects are still defined as sets of one-place relations and the division of urelements into ordinary urelements and special urelements, that serve as proxies for abstract objects, is retained as well. Consequently encoding is still defined as set membership of a relation in an abstract object and exemplification still defined as function application of the relation to the urelement corresponding to an individual.

The semantic truth evaluation of a proposition in a given possible world is defined as its evaluation for a designated *actual state* and the possible world.

Logical connectives are defined to behave classically in the *actual state*, but have undefined behavior in other states.

The reason for this construction becomes apparent if one considers the definition of the identity of relations: relations are considered identical if they are *encoded* by the same abstract objects. In the constructed model encoding depends on the behavior of a relation in all states. Two relations can necessarily be *exemplified* by the same objects in the actual state, but still not be identical, since they can differ in other states. Therefore hyperintensionality of relations is achieved.

The dependency on states is not limited to relations, but introduced to propositions, connectives and quantifiers as well, although the semantic truth conditions of formulas only depend on the evaluation for the actual state. The reason for this is to be able to define λ-expressions (see section 3.4.7) and to extend the hyperintensionality of relations to them. Since the behavior of logical connectives is undefined in states other than the actual state, the behavior of λ-expressions - although classical in the actual state - remains undefined for different states.

In summery, since the semantic truth of a proposition solely depends on its evaluation for the designated actual state, in which the logical connectives are defined to behave classically, the reasoning about propositions remains classical, as desired. On the other hand the additional dependency on states allows a representation of the hyperintensionality of relations.

The technical details of the implementation are described in section 3.4.

3.3.2. Layered Structure

Although the constructed variant of the Aczel-model preserves the hyperintensionality of the theory at least to some degree, it is still known that there are true theorems in this model that are not derivable from the axioms of object theory (see 3.12).

Given this lack of a model with a well-understood degree of completeness, the embedding uses a different approach than other semantical embeddings, namely the embedding is divided into several *layers* as follows:

- The first layer represents the primitives of PLM using the described hyperintensional and modal variant of the Aczel-model.
- In a second layer the objects of the embedded logic constructed in the first layer are considered as primitives and some of their semantic properties are derived using the background logic as meta-logic.

- The third layer derives the axiom system of PLM mostly using the semantics of the second layer and partly using the meta-logic directly.
- Based on the third layer the deductive system PLM as described in [12, Chap. 9] is derived solely using the axiom system of the third layer and the fundamental meta-rules stated in PLM. The meta-logic and the properties of the representation layer are explicitly not used in any proofs. Thereby the reasoning in this last layer is independent of the first two layers.

The rationale behind this approach is the following: The first layer provides a representation of the embedded logic that is provably consistent. Only minimal axiomatization is necessary, whereas the main construction is purely definitional. Since the subsequent layers don't contain any additional axiomatization (the axiom system in the third layer is *derived*) their consistency is thereby guaranteed as well.

The second layer tries to abstract from the details of the representation by implementing an approximation of the formal semantics of PLM[2]. The long time goal would be to arrive at the representation of a complete semantics in this layer, that would be sufficient to derive the axiom system in the next layer and which any specific model structure would have to satisfy. Unfortunately this could not be achieved so far, but it was possible to lay some foundations for future work.

At the moment full abstraction from the representation layer is only achieved after deriving the axiom system in the third layer. Still it can be reasoned that in any model of object theory the axiom system has to be derivable and therefore by disallowing all further proofs to rely on the meta-logic and the model structure directly the derivation of the deductive system PLM is universal. The only exceptions are the primitive meta-rules of PLM: modus ponens, RN (necessitation) and GEN (universal generalisation), as well as the deduction rule. These rules do not follow from the axiom system itself, but are derived from the semantics in the second layer (see 3.11.2). Still as the corresponding semantical rules will again have to be derivable for *any* model, this does not have an impact on the universality of the subsequent reasoning.

There remains one issue, though. Since the logic of PLM is formulated in relational type theory, whereas Isabelle/HOL employs functional reasoning some formulations have to be adjusted to be representable and therefore there may still be some reservations about the accuracy of the representation of the axiom system[3].

The technical details of the constructed embedding are described in the following sections.

[2] Our thanks to Edward Zalta for supplying us with a preliminary version of the corresponding unpublished chapter of PLM.

[3] See for example the discussion in section 3.10.5.

3.4. The Representation Layer

The first layer of the embedding (see A.1) implements the variant of the Aczel-model described in section 3.3.1 and builds a representation of the language of PLM in the logic of Isabelle/HOL. This process is outlined step by step throughout this section.

3.4.1. Primitives

The following primitive types are the basis of the embedding (see A.1.1):

- Type i represents possible worlds in the Kripke semantics.
- Type j represents *states* as decribed in section 3.3.1.
- 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 (see A.1.2):

- Type o is defined as the set of all functions of type $j \Rightarrow i \Rightarrow bool$ 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 $\upsilon \Rightarrow j \Rightarrow i \Rightarrow bool$ 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 $\upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow bool$ and represents two-place relations.
- Type Π_3 is defined as the set of all functions of type $\upsilon \Rightarrow \upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow bool$ 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 of type ω or an abstract object of type 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 this case type ν) and an additional special element called *None*. *None* is used to represent individual terms that are definite descriptions that are not logically proper (i.e. they do not denote an individual).

Remark. *The Isabelle syntax* **typedef** o = *UNIV*::($j \Rightarrow i \Rightarrow$*bool*) *set* **morphisms** *eval*o *make*o **..** *found in the theory source in the appendix introduces a new abstract type* o *that is represented by the full set (UNIV) of objects of type* $j \Rightarrow i \Rightarrow$ *bool. The morphism eval*o *maps an object of abstract type* o *to its representative of type* $j \Rightarrow i \Rightarrow$ *bool, whereas the morphism make*o *maps an object of type* $j \Rightarrow i \Rightarrow$ *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* $j \Rightarrow i \Rightarrow$ *bool.*

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

For the sake of readability in the documentation of the embedding the morphisms are omitted and definitions are stated directly for the representation types[4].

3.4.2. Individual Terms and Definite Descriptions

There are two basic types of individual terms in PLM: definite descriptions and individual variables (and constants). Every logically proper definite description denotes an individual. A definite description is logically proper if its matrix is true for a unique individual.

In the embedding the type κ encompasses all individual terms, i.e. individual variables, constants *and* definite descriptions. To use an individual (i.e. a variable or constant of type ν) in place of an individual term of type κ the decoration $\_^P$ is introduced (see A.1.3):

$$x^P = \textit{Some } x$$

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 *Some* x) and to denote the individual x.

Definite descriptions are defined as follows:

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

[4]The omission of the morphisms is achieved using custom *pretty printing* rules for the document generation facility of Isabelle. The full technical details without these minor omissions can be found in the raw Isabelle theory in the appendix.

If the propriety condition of a definite description $\exists!x.\ \varphi\ x\ dj\ dw$ holds, i.e. *there exists a unique x, such that φ x holds for the actual state and the actual world*, the term $\iota x.\ \varphi\ x$ evaluates to *Some* $(THE\ x.\ \varphi\ x\ dj\ dw)$. 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 term evaluates to *None*.

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

- $proper\ x = (None \neq x)$
- $rep\ x = the\ x$

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

3.4.3. Mapping from Individuals to Urelements

To map abstract objects to urelements (for which relations can be evaluated), a constant $\alpha\sigma$ of type $\alpha \Rightarrow \sigma$ is introduced, which maps abstract objects (of type α) to special urelements (of type σ), see A.1.4.

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.

Now the mapping $\nu\upsilon$ of type $\nu \Rightarrow \upsilon$ can be defined as follows:

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

To clarify the syntax note that this is equivalent to the following:

$$(\forall x.\ \nu\upsilon\ (\omega\nu\ x) = \omega\upsilon\ x) \wedge (\forall x.\ \nu\upsilon\ (\alpha\nu\ x) = \sigma\upsilon\ (\alpha\sigma\ x))$$

So ordinary objects are simply converted to an urelements by the type constructor $\omega\upsilon$ for ordinary urelements, whereas for abstract objects the corresponding special urelement under $\alpha\sigma$ is converted to an urelement using the type constructor $\sigma\upsilon$ for special urelements.

Remark. *Note that future versions of the embedding may introduce a dependency of the mapping from individuals to urelements on states (see 3.12).*

3.4.4. 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 (see A.1.5):

- $(\!|p|\!) = p$
- $(\!|F,x|\!) = (\lambda s\ w.\ \mathit{proper}\ x \wedge F\ (\nu\upsilon\ (\mathit{rep}\ x))\ s\ w)$
- $(\!|F,x,y|\!) = (\lambda s\ w.\ \mathit{proper}\ x \wedge \mathit{proper}\ y \wedge F\ (\nu\upsilon\ (\mathit{rep}\ x))\ (\nu\upsilon\ (\mathit{rep}\ y))\ s\ w)$
- $(\!|F,x,y,z|\!) =$
  $(\lambda s\ w.\ \mathit{proper}\ x \wedge$
  $\mathit{proper}\ y \wedge \mathit{proper}\ z \wedge F\ (\nu\upsilon\ (\mathit{rep}\ x))\ (\nu\upsilon\ (\mathit{rep}\ y))\ (\nu\upsilon\ (\mathit{rep}\ z))\ s\ w)$

3.4.5. Encoding

Encoding is defined as follows (see A.1.6):

$$\{\!|x,F|\!\} = (\lambda s\ w.\ \mathit{proper}\ x \wedge (\textsf{case}\ \mathit{rep}\ x\ \textsf{of}\ \omega\nu\ \omega \Rightarrow \mathit{False} \mid \alpha\nu\ \alpha \Rightarrow F \in \alpha))$$

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 denoted individual $\mathit{rep}\ x$ is of the form $\alpha\nu\ \alpha$ for some object α (i.e. it is an abstract object) and F is contained in α (remember that abstract objects are defined to be sets of one-place relations).

Note that encoding is represented as a function of states and possible worlds to ensure type-correctness, but its evaluation does not depend on either. On the other hand whether F is contained in α does depend on the behavior of F in *all* states.

3.4.6. Connectives and Quantifiers

Following the model described in section 3.3.1 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[5].

For this purpose the following uninterpreted constants are introduced (see A.1.7):

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

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 defined as follows (see A.1.7):

- $\neg p = (\lambda s\ w.\ s = \mathit{dj} \wedge \neg\ p\ \mathit{dj}\ w \vee s \neq \mathit{dj} \wedge \mathit{I\text{-}NOT}\ s\ (p\ s)\ w)$
- $p \rightarrow q =$
  $(\lambda s\ w.\ s = \mathit{dj} \wedge (p\ \mathit{dj}\ w \longrightarrow q\ \mathit{dj}\ w) \vee s \neq \mathit{dj} \wedge \mathit{I\text{-}IMPL}\ s\ (p\ s)\ (q\ s)\ w)$
- $\forall_\nu\ x\ .\ \varphi\ x = (\lambda s\ w.\ \forall x.\ (\varphi\ x)\ s\ w)$
- $\forall_0\ p\ .\ \varphi\ p = (\lambda s\ w.\ \forall p.\ (\varphi\ p)\ s\ w)$
- $\forall_1\ F\ .\ \varphi\ F = (\lambda s\ w.\ \forall F.\ (\varphi\ F)\ s\ w)$
- $\forall_2\ F\ .\ \varphi\ F = (\lambda s\ w.\ \forall F.\ (\varphi\ F)\ s\ w)$

[5]Early attempts in using an intuitionistic version of connectives and quantifiers based on [6] were found to be insufficient to capture the full hyperintensionality of PLM, but served as inspiration for the current construction.

- $\forall_3\ F\ .\ \varphi\ F = (\lambda s\ w.\ \forall F.\ (\varphi\ F)\ s\ w)$
- $\Box p = (\lambda s\ w.\ \forall v.\ p\ s\ v)$
- $\boldsymbol{\mathcal{A}} p = (\lambda s\ w.\ p\ s\ dw)$

Note in particular that negation and implication behave classically if evaluated for the actual state $s = dj$, but are governed by the uninterpreted constants *I-NOT* and *I-IMPL* for $s \neq dj$:

- $s = dj \Longrightarrow \neg p\ s\ w = (\neg\ p\ s\ w)$
- $s \neq dj \Longrightarrow \neg p\ s\ w = \textit{I-NOT}\ s\ (p\ s)\ w$
- $s = dj \Longrightarrow p \rightarrow q\ s\ w = (p\ s\ w \longrightarrow q\ s\ w)$
- $s \neq dj \Longrightarrow p \rightarrow q\ s\ w = \textit{I-IMPL}\ s\ (p\ s)\ (q\ s)\ w$

Remark. *Future research may conclude that non-classical behavior in states $s \neq dj$ for negation and implication is not sufficient for achieving the desired level of hyperintensionality for λ-expressions. It would be trivial to introduce additional uninterpreted constants to govern the behavior of the remaining connectives and quantifiers in such states as well, though. The remainder of the embedding would not be affected, i.e. no assumption about the behavior of connectives and quantifiers in states other than dj is made in the subsequent reasoning. At the time of writing non-classical behavior for negation and implication is considered sufficient.*

3.4.7. λ-Expressions

The bound variables of the λ-expressions of the embedded logic are individual variables, whereas relations are represented as functions acting on urelements. Therefore the definition of the λ-expressions of the embedded logic is non-trivial. The embedding defines them as follows (see A.1.8):

- $\boldsymbol{\lambda}^0\ p = p$
- $\boldsymbol{\lambda} x.\ \varphi\ x = (\lambda u\ s\ w.\ \exists x.\ \nu\upsilon\ x = u \wedge (\varphi\ x)\ s\ w)$
- $\boldsymbol{\lambda}^2\ (\lambda x\ y.\ \varphi\ x\ y) = (\lambda u\ v\ s\ w.\ \exists x\ y.\ \nu\upsilon\ x = u \wedge \nu\upsilon\ y = v \wedge (\varphi\ x\ y)\ s\ w)$
- $\boldsymbol{\lambda}^3\ (\lambda x\ y\ z.\ \varphi\ x\ y\ z) =$
  $(\lambda u\ v\ r\ s\ w.\ \exists x\ y\ z.\ \nu\upsilon\ x = u \wedge \nu\upsilon\ y = v \wedge \nu\upsilon\ z = r \wedge (\varphi\ x\ y\ z)\ s\ w)$

Remark. *For technical reasons Isabelle only allows λ-expressions for one-place relations to use a nice binder notation. Although better workarounds may be possible, for now the issue is avoided by the use of the primitive λ-expressions of the background logic in combination with the constants $\boldsymbol{\lambda}^2$ and $\boldsymbol{\lambda}^3$ as shown above.*

The representation of zero-place λ-expressions as the identity is straight-forward[6]; the representation of n-place λ-expressions for $n \geq 1$ is illustrated for the case $n = 1$:

[6] Although this representation is technically straight-forward, see section 5.1.1 for a further discussion of the implications of this representation and the way it differs from the zero-place relations of PLM.

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

The result of the evaluation of a λ-expression $\boldsymbol{\lambda} x.\ \varphi\ x$ for an urelment u, a state s and a possible world w) is given by the following equation:

$$\boldsymbol{\lambda} x.\ \varphi\ x\ u\ s\ w = (\exists\, x.\ \nu\upsilon\ x = u \wedge \varphi\ x\ s\ w)$$

Note that $\nu\upsilon$ is bijective for ordinary objects and therefore:

$$\boldsymbol{\lambda} x.\ \varphi\ x\ (\omega\upsilon\ u)\ s\ w = (\varphi\ (\omega\nu\ u))\ s\ w$$

However in general $\nu\upsilon$ can map several abstract objects to the same special urelement, so an analog statement for abstract objects does not hold for arbitrary φ. As described in section 3.2 such a statement would in fact not be desirable, since it would lead to inconsistencies.

Instead the embedding introduces the concept of *proper maps*. A map from individuals to propositions is defined to be proper if its truth evaluation for the actual state only depends on the urelement corresponding to the individual (see A.1.9):

- $\mathit{IsProperInX}\ \varphi = (\forall\, x\ v.\ (\exists\, a.\ \nu\upsilon\ a = \nu\upsilon\ x \wedge (\varphi\ (a^P))\ dj\ v) = (\varphi\ (x^P))\ dj\ v)$
- $\mathit{IsProperInXY}\ \varphi =$
  $(\forall\, x\ y\ v.$
  $(\exists\, a\ b.\ \nu\upsilon\ a = \nu\upsilon\ x \wedge \nu\upsilon\ b = \nu\upsilon\ y \wedge (\varphi\ (a^P)\ (b^P))\ dj\ v) =$
  $(\varphi\ (x^P)\ (y^P))\ dj\ v)$
- $\mathit{IsProperInXYZ}\ \varphi =$
  $(\forall\, x\ y\ z\ v.$
  $(\exists\, a\ b\ c.$
  $\nu\upsilon\ a = \nu\upsilon\ x \wedge \nu\upsilon\ b = \nu\upsilon\ y \wedge \nu\upsilon\ c = \nu\upsilon\ z \wedge (\varphi\ (a^P)\ (b^P)\ (c^P))\ dj\ v) =$
  $(\varphi\ (x^P)\ (y^P)\ (z^P))\ dj\ v)$

Now by the definition of proper maps the evaluation of λ-expressions behaves as expected for proper φ:

$$\mathit{IsProperInX}\ \varphi = (\forall\, w\ x.\ \boldsymbol{\lambda} x.\ \varphi\ (x^P)\ (\nu\upsilon\ x)\ dj\ w = \varphi\ (x^P)\ dj\ w)$$

Remark. *Note that the right-hand side of the equation above does not quantify over all states, but is restricted to the actual state dj. This is sufficient given that truth evaluation only depends on the actual state and goes along with the desired semantics of λ-expressions (see 3.5.5).*

The concept behind this is that maps that contain encoding formulas in its argument are in general not proper and thereby the paradox mentioned in section 3.2 is avoided.

In fact proper maps are the most general kind of functions that may appear in a lambda-expression, such that β-conversion holds. In what way proper maps correspond to the formulas that PLM allows as the matrix of a λ-expression is a complex question and discussed seperately in section 5.1.1.

3.4.8. 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 (see A.1.10):

$$[\varphi \ in \ v] = \varphi \ dj \ v$$

This way the truth evaluation of a proposition only depends on the evaluation of its functional representative for the actual state *dj*. Recall 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 expressions and only in the sense that an encoding expression depends on the behavior of the contained relation in all states.

Remark. *The Isabelle Theory in the appendix defines the syntax* $v \models p$ *in the representation layer, following the syntax used in the formal semantics of PLM. The syntax* $[p \ in \ v]$ *that is easier to use in Isabelle due to bracketing the expression is only introduced after the semantics is derived in A.2.3. For simplicity only the latter syntax is used in this documentation.*

3.4.9. Concreteness

PLM 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)[12], see 3.10.7. 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 (see A.1.11):

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

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

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

Whether an ordinary object is concrete is governed by the introduced constant, whereas abstract objects are never concrete.

3.4.10. The Syntax of the Embedded Logic

The embedding aims to provide a readable syntax for the embedded logic that is as close as possible to the syntax of PLM and clearly distinguishes between the embedded logic and the meta-logic. Some concessions have to be made due to the limitations of definable syntax in Isabelle, though. Moreover exemplification and encoding have to use a dedicated syntax in order to be distinguishable from function application.

The syntax for the basic formulas of PLM used in the embedding is summarized in the following table:

PLM	syntax in words	embedded logic	type
φ	it holds that φ	φ	o
$\neg\varphi$	not φ	$\neg\varphi$	o
$\varphi \to \psi$	φ implies ψ	$\varphi \rightarrow \psi$	o
$\Box\varphi$	necessarily φ	$\Box\varphi$	o
$\mathcal{A}\varphi$	actually φ	$\boldsymbol{\mathcal{A}}\varphi$	o
$\Pi\upsilon$	υ (an individual term) exemplifies Π	$(\!\vert\Pi,\upsilon\vert\!)$	o
Πx	x (an individual variable) exemplifies Π	$(\!\vert\Pi,x^P\vert\!)$	o
$\Pi\upsilon_1\upsilon_2$	υ_1 and υ_2 exemplify Π	$(\!\vert\Pi,\upsilon_1,\upsilon_2\vert\!)$	o
Πxy	x and y exemplify Π	$(\!\vert\Pi,x^P,y^P\vert\!)$	o
$\Pi\upsilon_1\upsilon_2\upsilon_3$	υ_1, υ_2 and υ_3 exemplify Π	$(\!\vert\Pi,\upsilon_1,\upsilon_2,\upsilon_3\vert\!)$	o
Πxyz	x, y and z exemplify Π	$(\!\vert\Pi,x^P,y^P,z^P\vert\!)$	o
$\upsilon\Pi$	υ encodes Π	$\{\!\vert\upsilon,\Pi\vert\!\}$	o
$\iota x\varphi$	*the* x, such that φ	$\boldsymbol{\iota}x.\ \varphi\ x$	κ
$\forall x(\varphi)$	for all individuals x it holds that φ	$\forall_\nu x.\ \varphi\ x$	o
$\forall p(\varphi)$	for all propositions p it holds that φ	$\forall_0 p.\ \varphi\ p$	o
$\forall F(\varphi)$	for all relations F it holds that φ	$\forall_1 F.\ \varphi\ F$	o
		$\forall_2 F.\ \varphi\ F$	
		$\forall_3 F.\ \varphi\ F$	
$[\lambda\ p]$	being such that p	$\boldsymbol{\lambda}^0\ p$	Π_0
$[\lambda x\ \varphi]$	being x such that φ	$\boldsymbol{\lambda}x.\ \varphi\ x$	Π_1
$[\lambda xy\ \varphi]$	being x and y such that φ	$\boldsymbol{\lambda}^2\ (\lambda x\ y.\ \varphi\ x\ y)$	Π_2
$[\lambda xyz\ \varphi]$	being x, y and z such that φ	$\boldsymbol{\lambda}^3\ (\lambda x\ y\ z.\ \varphi\ x\ y\ z)$	Π_3

Several subtleties have to be considered:

- n-place relations are only represented for $n \leq 3$. As the resulting language is already expressive enough to represent the most interesting parts of the theory and it would be trivial to add analog implementations for $n > 3$, this is considered to be sufficient. Future work may attempt to construct a general representation for n-place relations for arbitrary n.
- There is a distinction between individual terms and variables. This circumstance was already mentioned in section 3.4.2: an individual term in PLM can either be an individual variable (or constant) or a definite description. Statements in PLM that use individual variables are represented using the decoration $\_^{P}$.
- In PLM conceptually a general term φ, as it occurs in definite descriptions, quantifications and λ-expressions above, can contain *free* variables. If such a term occurs within the scope of a variable binding operator, free occurrences of the variable are considered to be *bound* by the operator. In the embedding this concept is replaced by considering φ to be a *function* acting on the bound variables and using the native concept of binding operators in Isabelle.
- The representation layer of the embedding defines a separate quantifier for every type of variable in PLM. This is done to assure that only quantification ranging over these types are part of the embedded language. The definition of a general quantifier in the representation layer could for example be used to quantify over individual *terms* (of type κ), whereas only quantification ranging over individuals (of type ν) is part of the language of PLM. After the semantics is introduced in section 3.5, a *type class* is constructed that is characterized by the semantics of quantification and instantiated for all variable types. This way a general binder that can be used for all variable types can be defined. The details of this approach are explained in section 3.6.

The syntax used for stating that a proposition is semantically valid is the following:

$$[\varphi \textit{ in } v]$$

Here φ and v are free variables (in the meta-logic), therefore stating the expression as a lemma will implicitly be a quantified statement over all propositions φ and all possible worlds v (unless φ was explicitly declared as a constant in the global scope).

3.5. Semantical Abstraction

The second layer of the embedding (see A.2) abstracts away from the technicalities of the representation layer and states the truth conditions for formulas of the embedded logic in a similar way as the (at the time of writing unpublished) semantics of object theory.

3.5.1. Domains and Denotation Functions

In order to do so the abstract types introduced in the representation layer κ, o resp. Π_0, Π_1, Π_2 and Π_3 are considered as primitive types and assigned semantic domains: R_κ, R_0, R_1, R_2 and R_3 (see A.2.1.1).

For the embedding the definition of these semantic domains is trivial, since the abstract types of the representation layer are already modeled using representation sets. Therefore the semantic domains for each type can simply be defined as the type of its representatives.

As a next step denotation functions are defined that assign semantic denotations to the objects of each abstract type (see A.2.1.2). Note that the formal semantics of PLM does not a priori assume that every term has a denotation, therefore the denotation functions are represented as functions that map to the *option* type of the respective domain. This way they can either map a term to *Some* x, if the term denotes x, or to *None*, if the term does not denote.

In the embedding all relation terms always denote, therefore the denotation functions d_0, ..., d_3 for relations can simply be defined as the type constructor *Some*. Individual terms on the other hand are already represented by an *option* type, so the denotation function d_κ can be defined as the identity.

Moreover the primitive type of possible worlds i is used as the semantical domain of possible worlds W and the primitive actual world *dw* as the semantical actual world w_0 (see A.2.1.3).

Remark. *Although the definitions for semantical domains and denotations seem redundant, conceptually the abstract types of the representation layer now have the role of primitive types. Although for simplicity the last section regarded the type* o *as synonym of $j \Rightarrow i \Rightarrow$ bool, it was introduced as a distinct type for which the set of all functions of type $j \Rightarrow i \Rightarrow$ bool merely serves as the underlying set of representatives. An object of type* o *cannot directly be substituted for a variable of type $j \Rightarrow i \Rightarrow$ bool. To do so it first has to be mapped to its representative of type $j \Rightarrow i \Rightarrow$ bool by the use of the morphism* evalo *that was introduced in the type definition and omitted in the last section for the sake of readability. Therefore although the definitions of the semantic domains and denotation functions may seem superfluous, the domains are different types than the corresponding abstract type and the denotation functions are functions between distinct types (note the use of* **lift-definition** *rather than* **definition** *for the denotation functions in A.2.1.2 that allows to define functions on abstract types in the terms of the underlying representation types).*

3.5.2. Exemplification and Encoding Extensions

Semantic truth conditions for exemplification formulas are defined using *exemplification extensions*. Exemplification extensions are functions relative to semantic possible worlds that map objects in the domain of n-place relations to meta-logical truth conditions in the case $n = 0$ and sets of n-tuples of objects in the domain of individuals in the case $n \geq 1$. Formally they are defined as follows (see A.2.1.4):

- $ex0\ p\ w = p\ dj\ w$
- $ex1\ F\ w = \{x \mid F\ (\nu\upsilon\ x)\ dj\ w\}$
- $ex2\ R\ w = \{(x,\ y) \mid R\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ dj\ w\}$
- $ex3\ R\ w = \{(x,\ y,\ z) \mid R\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ (\nu\upsilon\ z)\ dj\ w\}$

The exemplification extension of a 0-place relation is its evaluation for the actual state and the given possible world. The exemplification extension of n-place relations ($n \geq 1$) in a possible world is the set of all (tuples of) *individuals* that are mapped to an *urelement* for which the relation evaluates to true for the given possible world and the actual state. This is in accordance with the constructed Aczel-model (see 3.3.1).

It is important to note that the concept of exemplification extensions as maps to sets of *individuals* is independent of the underlying model and in particular does not need the concept of *urelements* as they are present in an Aczel-model. The definition of truth conditions by the use of exemplification extensions is therefore an abstraction away from the technicalities of the representation layer.

Similarly to the exemplification extension for one-place relations an *encoding extension* is defined as follows (see A.2.1.5):

$$en\ F = \{x \mid \mathsf{case}\ x\ \mathsf{of}\ \omega\nu\ \omega \Rightarrow False \mid \alpha\nu\ y \Rightarrow F \in y\}$$

The encoding extension of a relation is defined as the set of all abstract objects that contain the relation. Since encoding is modally rigid the encoding extension does not need to be relativized for possible worlds.

3.5.3. Truth Conditons of Formulas

On the basis of the semantical definitions above it is now possible to define truth conditions for the atomic formulas of the language.

For exemplification formulas of n-place relations it suffices to consider the case of one-place relations, for which the truth condition is defined as follows (see A.2.1.7):

$$[(\!|\Pi,\kappa|\!)\ in\ w] = (\exists\, r\ o_1.\ Some\ r = d_1\ \Pi \wedge Some\ o_1 = d_\kappa\ \kappa \wedge o_1 \in ex1\ r\ w)$$

The relation term Π is exemplified by an individual term κ in a possible world w if both terms have a denotation and the denoted individual is contained in the exemplification extension of the denoted relation in w. The definitions for n-place relations ($n > 1$) and 0-place relations are analog.

The truth condition for encoding formulas is defined in a similar manner (see A.2.1.8):

$$[\{\!|\kappa,\Pi|\!\}\ in\ w] = (\exists\, r\ o_1.\ Some\ r = d_1\ \Pi \wedge Some\ o_1 = d_\kappa\ \kappa \wedge o_1 \in en\ r)$$

The only difference to exemplification formulas is that the encoding extension does not depend on the possible world w.

The definitions of truth conditions for complex formulas are straightforward (see A.2.1.9):

- $[\neg\psi \ \textit{in}\ w] = (\neg\ [\psi\ \textit{in}\ w])$
- $[\psi \rightarrow \chi\ \textit{in}\ w] = (\neg\ [\psi\ \textit{in}\ w] \vee [\chi\ \textit{in}\ w])$
- $[\Box\psi\ \textit{in}\ w] = (\forall\, v.\ [\psi\ \textit{in}\ v])$
- $[\boldsymbol{\mathcal{A}}\psi\ \textit{in}\ w] = [\psi\ \textit{in}\ dw]$
- $[\forall_{\nu} x.\ \psi\ x\ \textit{in}\ w] = (\forall\, x.\ [\psi\ x\ \textit{in}\ w])$
- $[\forall_{0} x.\ \psi\ x\ \textit{in}\ w] = (\forall\, x.\ [\psi\ x\ \textit{in}\ w])$
- $[\forall_{1} x.\ \psi\ x\ \textit{in}\ w] = (\forall\, x.\ [\psi\ x\ \textit{in}\ w])$
- $[\forall_{2} x.\ \psi\ x\ \textit{in}\ w] = (\forall\, x.\ [\psi\ x\ \textit{in}\ w])$
- $[\forall_{3} x.\ \psi\ x\ \textit{in}\ w] = (\forall\, x.\ [\psi\ x\ \textit{in}\ w])$

A negation formula $\neg\psi$ is semantically true in a possible world, if and only if ψ is not semantically true in the given possible world. Similarly truth conditions for implication formulas and quantification formulas are defined canonically.

The truth condition of the modal box operator $\Box\psi$ as ψ being true in all possible worlds, shows that modality follows a S5 logic. A formula involving the actuality operator $\boldsymbol{\mathcal{A}}\psi$ is defined to be semantically true, if and only if ψ is true in the designated actual world.

3.5.4. Denotation of Definite Descriptions

The definition of the denotation of description terms (see A.2.1.10) can be presented in a more readable form by splitting it into its two cases and by using the meta-logical quantifier for unique existence:

- $\exists!x.\ [\psi\ x\ \textit{in}\ w_0] \Longrightarrow d_\kappa\ \boldsymbol{\iota} x.\ \psi\ x = \textit{Some}\ (\textit{THE}\ x.\ [\psi\ x\ \textit{in}\ w_0])$
- $\nexists!x.\ [\psi\ x\ \textit{in}\ w_0] \Longrightarrow d_\kappa\ \boldsymbol{\iota} x.\ \psi\ x = \textit{None}$

If there exists a unique x, such that $\psi\ x$ is true in the actual world, the definite description denotes and its denotation is this unique x. Otherwise the definite description fails to denote.

It is important to consider what happens if a non-denoting definite description occurs in a formula: The only positions in which such a term could occur in a complex formula is in an exemplification expression or in an encoding expression. Given the above truth conditions it becomes clear, that the presence of non-denoting terms does *not* mean that there are formulas without truth conditions: Since exemplification and encoding formulas are defined to be true *only if* the contained individual term has a denotation, such formulas are *False* for non-denoting individual terms.

3.5.5. Denotation of λ-Expressions

The most complex part of the semantical abstraction is the definition of denotations for λ-expressions. The formal semantics of PLM is split into several cases and uses a special class of *Hilbert-Ackermann ε-terms* that are challenging to represent. Therefore a simplified formulation of the denotation criteria is used. Moreover the denotations of λ-expressions are coupled to syntactical conditions. This fact is represented using the notion of *proper maps* as a restriction for the matrix of a λ-expression that was introduced in section 3.4.7. The definitions are implemented as follows (see A.2.1.11):

- $d_1\ \boldsymbol{\lambda} x.\ (\!|\Pi, x^P|\!) = d_1\ \Pi$
- $\mathit{IsProperInX}\ \varphi \Longrightarrow$
  $\mathit{Some}\ r = d_1\ \boldsymbol{\lambda} x.\ \varphi\ (x^P) \wedge \mathit{Some}\ o_1 = d_\kappa\ x \longrightarrow (o_1 \in \mathit{ex1}\ r\ w) = [\varphi\ x\ \mathit{in}\ w]$
- $\mathit{Some}\ r = d_0\ \boldsymbol{\lambda}^0\ \varphi \longrightarrow \mathit{ex0}\ r\ w = [\varphi\ \mathit{in}\ w]$

The first condition for *elementary* λ-expressions is straightforward. The general case in the second condition is more complex: Given that the matrix φ is a proper map, the relation denoted by the λ-expression has the property, that for a denoting individual term x, the denoted individual is contained in its exemplification extension for a possible world w, if and only if $\varphi\ x$ holds in w. At a closer look this is the statement of β-conversion restricted to denoting individuals: the truth condition of the λ-expression being exemplified by some denoting individual term, is the same as the truth condition of the matrix of the term for the denoted individual. Therefore it is clear that the precondition that φ is a proper map is necessary and sufficient. Given this consideration the case for *0*-place relations is straightforward and the cases for $n \geq 2$ are analog to the case $n = 1$.

3.5.6. Properties of the Semantics

The formal semantics of PLM imposes several further restrictions some of which are derived as auxiliary lemmas. Furthermore some auxiliary statements that are specific to the underlying representation layer are proven. Future work may try to refrain from this second kind of statements.

The following auxiliary statements are derived (see A.2.1.12):

1. All relations denote, e.g.
   $\exists\, r.\ \mathit{Some}\ r = d_1\ F$
2. An individual term of the form x^P denotes x:
   $d_\kappa\ x^P = \mathit{Some}\ x$
3. Every ordinary object is contained in the extension of the concreteness property for some possible world:
   $\mathit{Some}\ r = d_1\ E! \Longrightarrow \forall\, x.\ \exists\, w.\ \omega\nu\ x \in \mathit{ex1}\ r\ w$
4. An object that is contained in the extension of the concreteness property in any world is an ordinary object:
   $\mathit{Some}\ r = d_1\ E! \Longrightarrow \forall\, x.\ x \in \mathit{ex1}\ r\ w \longrightarrow (\exists\, y.\ x = \omega\nu\ y)$
5. The denotation functions for relation terms are injective, e.g.
   $d_1\ F = d_1\ G \Longrightarrow F = G$
6. The denotation function for individual terms is injective for denoting terms:

$$Some\ o_1 = d_\kappa\ x \wedge Some\ o_1 = d_\kappa\ y \Longrightarrow x = y$$

Especially the statements 5 and 6 are only derivable due to the specific construction of the representation layer: since the semantical domains were defined as the representation sets of the respective type of term and denotations were defined canonically, objects that have the same denotation are identical as objects of their abstract type. 3 and 4 are necessary to connect concreteness with the underlying distinction between ordinary and abstract objects in the model.

3.5.7. Proper Maps

The definition of *proper maps* as described in section 3.4.7 is formulated in terms of the meta-logic. Since denotation conditions in the semantics and later some of the axioms have to be restricted to proper maps, a method has to be devised by which the propriety of a map can easily be shown without using meta-logical concepts.

Therefore introduction rules for *IsProperInX*, *IsProperInXY* and *IsProperInXYZ* are derived and a proving method *show-proper* is defined that can be used to proof the propriety of a map using these introduction rules (see A.2.2).

The rules themselves rely on the power of the *unifier* of Isabelle/HOL: Any map acting on individuals that can be expressed by another map that solely acts on exemplification expressions involving the individuals, is shown to be proper. This effectively means that all maps whose arguments only appear in exemplification expressions are proper. For a discussion about the relation between this concept and admissible λ-expressions in PLM see section 5.1.1.

3.6. General All-Quantifier

Since the last section established the semantic truth conditions of the specific versions of the all quantifier for all variable types of PLM, it is now possible to define a binding symbol for general all quantification.

This is done using the concept of *type classes* in Isabelle/HOL. Type classes define constants that depend on a *type variable* and state assumptions about this constant. In subsequent reasoning the type of an object can be restricted to a type of the introduced type class. Thereby the reasoning can make use of all assumptions that have been stated about the constants of the type class. A priori it is not assumed that any type actually satisfies the requirements of the type class, so initially statements involving types restricted to a type class can not be applied to any specific type.

To allow this the type class has to be *instantiated* for the desired type. This is done by first providing a definition of the constants of the type class that is specific to the respective type and then providing proofs for each assumption made by the type class given the particular type and the provided definitions. Once this is done any statement that was proven for the general type class can be applied to this type.

In the case of general all quantification for the embedding this concept can be utilized by introducing the type class *quantifiable* that is equipped with a constant that is used as the general all quantification binder (see A.3.1). For this constant it can now be assumed that it satisfies the semantic property of all quantification: $[\forall\, x.\ \psi\ x\ in\ w] = (\forall\, x.\ [\psi\ x\ in\ w])$.

Since it was already shown in the last section that the specific all quantifier for each variable type satisfies this property, the type class can immediately be instantiated for the types ν, Π_0, Π_1, Π_2 and Π_3 (see A.3.2). The instantiation proofs only need to refer to the statements derived in the semantics section for the respective version of the quantifier and are thereby independent of the representation layer.

From this point onward therefore the general all quantifier replace the type specific quantifiers. This is true even if a quantification is meant to only range over objects of a particular type: In this case the desired type (if it can not implicitly be deduced from the context) can be stated explicitly while still using the general quantifier.

Remark. *Technically it would be possible to instantiate the type class quantifiable for any other type that satisfies the semantic criterion, thereby compromising the restriction of the all-quantifier to the primitive types of PLM. However, this is not done in the embedding and therefore the introduction of a general quantifier using a type class is considered a reasonable compromise.*

3.7. Derived Language Elements

The language of the embedded logic constructed so far is limited to the minimal set of primitive elements, e.g. only negation and implication are present and no notion of identity has been introduced so far.

Since section 3.5 has established the semantic properties of these basic elements of the language, it now makes sense to extend the language by introducing some basic definitions that can be expressed directly in the embedded logic.

3.7.1. Connectives

The remaining classical connectives are defined in the traditional manner (see A.4.1):

- $\varphi\ \&\ \psi = \neg(\varphi \rightarrow \neg\psi)$
- $\varphi \lor \psi = \neg\varphi \rightarrow \psi$
- $\varphi \equiv \psi = (\varphi \rightarrow \psi)\ \&\ (\psi \rightarrow \varphi)$
- $\Diamond\varphi = \neg\Box\neg\varphi$

Furthermore the general all quantifier is supplemented by an existence quantifier as follows:

- $\exists\ \alpha\ .\ \varphi\ \alpha = \neg(\forall\, \alpha.\ \neg\varphi\ \alpha)$

3.7.2. Identity

More importantly the definitions for identity are stated for each type of variable (see A.4.3):

- $x =_E y = (|\boldsymbol{\lambda}^2 (\lambda x\ y.\ (|O!,x^P|) \ \&\ (|O!,y^P|) \ \&\ \Box(\forall F.\ (|F,x^P|) \equiv (|F,y^P|))),x,y|)$
- $F =_1 G = \Box(\forall x.\ \{|x^P,F|\} \equiv \{|x^P,G|\})$
- $F =_2 G =$
  $\forall x.\ (\boldsymbol{\lambda} y.\ (|F,x^P,y^P|)) =_1 (\boldsymbol{\lambda} y.\ (|G,x^P,y^P|)) \ \&\ (\boldsymbol{\lambda} y.\ (|F,y^P,x^P|)) =_1 (\boldsymbol{\lambda} y.\ (|G,y^P,x^P|))$
- $F =_3 G = \forall x\ y.\ (\boldsymbol{\lambda} z.\ (|F,z^P,x^P,y^P|)) =_1 (\boldsymbol{\lambda} z.\ (|G,z^P,x^P,y^P|)) \ \&\ (\boldsymbol{\lambda} z.\ (|F,x^P,z^P,y^P|))$
  $=_1 (\boldsymbol{\lambda} z.\ (|G,x^P,z^P,y^P|)) \ \&\ (\boldsymbol{\lambda} z.\ (|F,x^P,y^P,z^P|)) =_1 (\boldsymbol{\lambda} z.\ (|G,x^P,y^P,z^P|))$

Similarly to the general all quantifier it makes sense to introduce a general identity relation for all types of terms (κ, o resp. Π_0, Π_1, Π_2, Π_3). In contrast to all quantification this is more challenging, though, since there is no semantic criterion that characterizes the identity for all these types. Therefore a general identity symbol will only be introduced after the next section, since it will then be possible to formulate and prove a reasonable property shared by the identity of all types of terms.

3.8. Proving Method meta_solver

3.8.1. Concept

Since the semantics in section 3.5 constructed a first abstraction layer on top of the representation layer (that focuses on a specific model structure), it makes sense to revisit the general concept of the layered structure of the embedding.

The idea behind this structure is that the reasoning in subsequent layers should - as far as possible - only rely on the previous layer. Automated reasoning in this context, however, can be cumbersome, since automated proving tools have to be restricted to only consider a specific subset of the theorems that are present in the global context. While this is possible it is still an interesting question whether the process of automated reasoning in the layered approach can be made easier.

To that end the embedding facilitates the possibility of defining custom proving methods that the Isabelle package *Eisbach* provides. This package allows it to conveniently define new proving methods that are based on the systematic application of existing methods.

Remark. *The Eisbach package even allows to construct more complex proving methods that involve pattern matching. This is for example utilized in the construction of a substitution method as described in section 3.11.5.*

The idea is to construct a simple resolution prover that can deconstruct complex formulas of the embedded logic to simpler formulas that are connected by some relation in the meta-logic as required by the semantics.

For example an implication formula can be deconstructed as follows:

$$[\varphi \rightarrow \psi \ in \ v] = ([\varphi \ in \ v] \longrightarrow [\psi \ in \ v])$$

Whereas the basic proving methods available in Isabelle cannot immediately prove $[\varphi \rightarrow \varphi \ in \ v]$ without any facts about the definitions of validity and implication, they *can* prove $[\varphi \ in \ v] \longrightarrow [\varphi \ in \ v]$ directly as an instance of $p \longrightarrow p$.

3.8.2. Implementation

Following this idea the method *meta-solver* is introduced (see A.5) that repeatedly applies rules like the above in order to translate a formula in the embedded logic to a meta-logical statement involving simpler formulas.

The statement of appropriate introduction, elimination and substitution rules for the logical connectives and quantifiers is straightforward, but beyond that the concept can be used to resolve exemplification and encoding formulas to their semantic truth conditions as well, e.g. (see A.5.10):

$$[(\!|F,x|\!) \ in \ v] \Longrightarrow \exists \, r \ o_1. \ Some \ r = d_1 \ F \wedge Some \ o_1 = d_\kappa \ x \wedge o_1 \in ex1 \ r \ v$$

This way a large set of formulas can be decomposed to semantic expressions that can be automatically proven without having to rely on the meta-logical definitions directly.

As mentioned before the concept of a strict separation between the layers is not yet achieved by the embedding. In particular the *meta-solver* is equipped with rules about being abstract and ordinary and most importantly about the defined identity that depend on the specific structure of the representation layer and are not solely derivable from the semantics.

Notably the representation layer has the property that the defined identities are equivalent to the identity in the meta-logic. Formally the following statements are true and defined as rules for the *meta-solver*:

- $[x =_E y \ in \ v] =$
  $(\exists \, o_1 \ X \ o_2. \ Some \ o_1 = d_\kappa \ x \wedge Some \ o_2 = d_\kappa \ y \wedge o_1 = o_2 \wedge o_1 = \omega\nu \ X)$
- $[x =_\kappa y \ in \ v] = (\exists \, o_1 \ o_2. \ Some \ o_1 = d_\kappa \ x \wedge Some \ o_2 = d_\kappa \ y \wedge o_1 = o_2)$
- $[F =_1 G \ in \ v] = (F = G)$
- $[F =_2 G \ in \ v] = (F = G)$
- $[F =_3 G \ in \ v] = (F = G)$
- $[F =_0 G \ in \ v] = (F = G)$

The proofs for these statements (see A.5.15) are complex and do not solely rely on the properties of the formal semantics of PLM.

Although future work may try to forgo these statements or to replace them with statements that are in fact based on the formal semantics alone, the fact that they are true in the constructed embedding has a distinct advantage: since identical terms in the sense of PLM are identical in the meta-logic, proving the axiom of substitution (see 3.10.4) is trivial.

Remark. *Instead of introducing a custom proving method using the Eisbach package, a similar effect could be achieved by instead supplying the derived introduction, elimination* *and substitution rules directly to one of the existing proving methods like auto or clarsimp. In practice, however, we found that the custom meta-solver produces more reliable results, especially in the case that a proving objective cannot be solved by the supplied rules completely. Moreover the construction of a custom proving method may serve as a proof of concept and inspire the development of further more complex proving methods that go beyond a simple resolution prover in the future.*

3.8.3. Applicability

Given the discussion above and keeping the layered structure of the embedding in mind, it is important to precisely determine for which purposes it is valid to use the constructed *meta-solver*.

The main application of the method in the embedding is its use in the derivation of the axiom system as described in section 3.10. Furthermore the *meta-solver* can aid in examining the meta-logical properties of the embedding. Care has been taken that the meta-solver only employs *reversible* transformations, thereby it is for example justified to use it to simplify a statement before employing a tool like **nitpick** in order to look for counter-models for a statement.

However it is *not* justified to assume that a theorem that can be proven with the aid of the *meta-solver* method can be considered to be derivable in the formal system of PLM, since the result still depends on the specific structure of the representation layer. Note, however, that the concept of the *meta-solver* inspired another proving method that is introduced in section 3.11.3, namely the *PLM-solver*. This proving method only employs rules that are derivable from the formal system of PLM itself. Thereby this method *can* be used in proves without sacrificing the universality of the result.

3.9. General Identity Relation

As already mentioned in section 3.6 similarly to the general quantification binder it is desirable to introduce a general identity relation.

Since the identity of PLM is not directly characterized by semantic truth conditions, but instead *defined* using specific complex formulas in the embedded logic for each type of terms, some other property has to be found that is shared by the specific definitions of identity and can reasonably be used as the condition of a type class.

A natural choice for such a condition is based on the axiom of the substitution of identicals (see 3.10.4). The axiom states that if two objects are identical (in the sense of the defined identity of PLM), then a formula involving the first object implies the formula resulting from substituting the second object for the first object. This inspires the following condition for the type class (see A.6.1):

$$[\alpha = \beta \ in \ v] \wedge [\varphi \ \alpha \ in \ v] \Longrightarrow [\varphi \ \beta \ in \ v]$$

Using the fact that in the last section it was already derived, that the defined identity in the embedded-logic for each term implies the primitive identity of the meta-logical objects, this type class can be instantiated for all types of terms: κ, Π_0 resp. o, Π_1, Π_2, Π_3 (see A.6.2).

Since now general quantification and general identity are available, an additional quantifier for unique existence can be introduced (such a quantifier involves both quantification and identity). To that end a derived type class is introduced that is the combination of the *quantifiable* and the *identifiable* classes. Although this is straightforward for the relation types, this reveals a subtlety involving the distinction between individuals of type ν and individual terms of type κ: The type ν belongs to the class *quantifiable*, the type κ on the other hand does not: no quantification over individual *terms* (that may not denote) was defined. On the other hand the class *identifiable* was only instantiated for the type κ, but not for the type ν. This issue can be solved by noticing that it is straightforward and justified to define an identity for ν as follows:

$$x = y = x^P = y^P$$

This way type ν is equipped with both the general all quantifier and the general identity relation and unique existence can be defined for all variable types as expected:

$$\exists\,!\ \alpha\ .\ \varphi\ \alpha = \exists\,\alpha.\ \varphi\ \alpha\ \&\ (\forall\,\beta.\ \varphi\ \beta \rightarrow \beta = \alpha)$$

Another subtlety has to be considered: at times it is necessary to expand the definitions of identity for a specific type to derive statements in PLM. Since the defined identities were introduced prior to the general identity symbol, such an expansion is therefore so far not possible for a statement that uses the general identity, even if the types are fixed in the context.

To allow such an expansion the definitions of identity are equivalently restated for the general identity symbol and each specific type (see A.6.3). This way the general identity can from this point onward completely replace the type-specific identity symbols.

3.10. The Axiom System of PLM

The last step in abstracting away from the representation layer is the derivation of the axiom system of PLM. Conceptionally the derivation of the axioms is the last moment in which it is deemed admissible to rely on the meta-logical properties of the underlying model structure. Future work may even restrict this further to only allow the use of the properties of the semantics in the proofs (if this is found to be possible).

To be able to distinguish between the axioms and other statements and theorems in the embedded logic they are stated using a dedicated syntax. Axioms are not stated relative to a specific possible world, but using the following definition (see A.7):

$$[[\ \varphi\ ]] = (\forall\, v.\ [\varphi\ in\ v])$$

Axioms are unconditionally true in all possible worlds. The only exceptions are *necessitation-averse*, resp. *modally fragile* axioms[7]. Such axioms are stated using the following definition:

$$[\varphi] = [\varphi \ in \ dw]$$

Remark. *Additionally a proving method axiom-meta-solver is introduced, that unfolds the definitions above, then applies the meta-solver method and if possible resolves the proof objective automatically.*

3.10.1. Axioms as Schemata

The axioms in PLM are stated as *axiom schemata*. They use variables that range over and can therefore be instantiated for any formula and term (potentially satisfying explicitly stated restrictions). Furthermore PLM introduces the notion of *closures*. Effectively this means that the statement of an axiom schema implies that the universal generalization of the schema, the actualization of the schema and the necessitation of the schema is also an axiom. The modally fragile Axiom 30 is the only exception: Its necessitations are not axioms as well.

Since in Isabelle/HOL free variables in a theorem already range over all terms of the same type no special measures have to be taken to allow instantiations for arbitrary terms. The concept of closures is introduced using the following rules (see A.7.1):

- $[[\ \varphi\ ]] \Longrightarrow [\varphi \ in \ v]$
- $(\bigwedge x.\ [[\ \varphi\ x\ ]]) \Longrightarrow [[\ \forall\, x.\ \varphi\ x\ ]]$
- $[[\ \varphi\ ]] \Longrightarrow [[\ \boldsymbol{\mathcal{A}}\varphi\ ]]$
- $[[\ \varphi\ ]] \Longrightarrow [[\ \Box\varphi\ ]]$

For modally fragile axioms only the following rules are introduced:

- $[\varphi] \Longrightarrow [\varphi \ in \ dw]$
- $(\bigwedge x.\ [\varphi\ x]) \Longrightarrow [\forall\, x.\ \varphi\ x]$

Remark. *To simplify the instantiation of the axioms in subsequent proofs, a set of* attributes *is defined that can be used to transform the statement of the axioms using the rules defined above.*

This way for example the axiom $[[\ \Box\varphi \rightarrow \varphi\ ]]$ *can be directly transformed to* $[\forall\, x.\ \Box\varphi\ x \rightarrow \varphi\ x \ in \ v]$ *by not referencing it directly as qml-2, but by applying the defined attributes to it: qml-2[axiom-universal, axiom-instance]*

3.10.2. Derivation of the Axioms

Most of the axioms can be derived by the *axiom-meta-solver* directly. Some axioms, however, require more verbose proofs or special attention has to be taken regarding

[7]Currently PLM uses only one such axiom, see 3.10.6.

their representation in the functional setting of Isabelle/HOL. Therefore in the following the complete axiom system is listed and discussed in detail where necessary. Additionally each axiom is associated with the numbering in the current draft of PLM[12].

3.10.3. Axioms for Negations and Conditionals

The axioms for negations and conditionals can be derived automatically and present no further issues (see A.7.2):

- $[\![\ \varphi \rightarrow (\psi \rightarrow \varphi)\ ]\!]$ (21.1)
- $[\![\ \varphi \rightarrow (\psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \psi \rightarrow (\varphi \rightarrow \chi))\ ]\!]$ (21.2)
- $[\![\ \neg\varphi \rightarrow \neg\psi \rightarrow (\neg\varphi \rightarrow \psi \rightarrow \varphi)\ ]\!]$ (21.3)

3.10.4. Axioms of Identity

The axiom of the substitution of identicals can be proven automatically, if additionally supplied with the defining assumption of the type class *identifiable*. The statement is the following (see A.7.3):

- $[\![\ \alpha = \beta \rightarrow (\varphi\ \alpha \rightarrow \varphi\ \beta)\ ]\!]$ (25)

3.10.5. Axioms of Quantification

The axioms of quantification are formulated in a way that differs from the statements in PLM, as follows (see A.7.4):

- $[\![\ (\forall\,\alpha.\ \varphi\ \alpha) \rightarrow \varphi\ \alpha\ ]\!]$ (29.1)
- $[\![\ (\forall\,\alpha.\ \varphi\ (\alpha^P)) \rightarrow ((\exists\,\beta.\ \beta^P = \alpha) \rightarrow \varphi\ \alpha)\ ]\!]$ (29.1)
- $[\![\ (\forall\,\alpha.\ \varphi\ \alpha \rightarrow \psi\ \alpha) \rightarrow ((\forall\,\alpha.\ \varphi\ \alpha) \rightarrow (\forall\,\alpha.\ \psi\ \alpha))\ ]\!]$ (29.3)
- $[\![\ \varphi \rightarrow (\forall\,\alpha.\ \varphi)\ ]\!]$ (29.4)
- $\mathit{SimpleExOrEnc}\ \psi \Longrightarrow [\![\ \psi\ (\iota x.\ \varphi\ x) \rightarrow (\exists\,\alpha.\ \alpha^P = (\iota x.\ \varphi\ x))\ ]\!]$ (29.5)
- $\mathit{SimpleExOrEnc}\ \psi \Longrightarrow [\![\ \psi\ \tau \rightarrow (\exists\,\alpha.\ \alpha^P = \tau)\ ]\!]$ (29.5)

The direct translation of the axioms of PLM would be the following:

- $[\![\ (\forall\,\alpha.\ \varphi\ \alpha) \rightarrow ((\exists\,\beta.\ \beta = \tau) \rightarrow \varphi\ \tau)\ ]\!]$ (29.1)
- $[\![\ \exists\,\beta.\ \beta = \tau\ ]\!]$ (29.2)
- $[\![\ (\forall\,\alpha.\ \varphi\ \alpha \rightarrow \psi\ \alpha) \rightarrow ((\forall\,\alpha.\ \varphi\ \alpha) \rightarrow (\forall\,\alpha.\ \psi\ \alpha))\ ]\!]$ (29.3)
- $[\![\ \varphi \rightarrow (\forall\,\alpha.\ \varphi)\ ]\!]$ (29.4)
- $\mathit{SimpleExOrEnc}\ \psi \Longrightarrow [\![\ \psi\ (\iota x.\ \varphi\ x) \rightarrow (\exists\,\alpha.\ \alpha^P = (\iota x.\ \varphi\ x))\ ]\!]$ (29.5)

Axiom (29.2) is furthermore restricted to τ not being a definite description. In the embedding definite descriptions have the type κ that is different from the type for individuals ν and quantification is only defined for ν, not for κ.

Thereby the restriction of (29.2) does not apply, since τ cannot be a definite description by construction. Since (29.2) would therefore hold in general, the additional restriction of (29.1) can be dropped - again since a quantifier is used in the formulation, the problematic case of definite descriptions (which would have type κ) is excluded already.

Now the modification of (29.5) can be explained: Since (29.2) already implies the right hand side for every term except definite descriptions, (29.5) can be stated for general terms instead of stating it specifically for definite descriptions.

What is left to be considered is how (29.1) can be applied to definite descriptions in the embedding. The modified version of (29.5) states that under the same condition that the unmodified version requires for a description to denote, the description (that has type κ) denotes an object of type ν and thereby (29.1) can be applied using the substitution of identicals.

Future work may want to reconsider the reformulation of the axioms, especially considering the most recent developments described in section 5.2. At the time of writing the fact that due to the type restrictions of the embedding, the reformulated version of the axioms is *derivable* from the original version the reformulation is a reasonable compromise.

Remark. *A formulation of the axioms as in PLM could be possible using the concept of domain restricted quantification as employed in embedding free logics (see [4]) to define a restricted quantification over the type κ. This would require some non-trivial restructuring of the embedding, though.*

The predicate *SimpleExOrEnc* used as the precondition for (29.5) is defined as an inductive predicate with the following introduction rules:

- *SimpleExOrEnc* $(\lambda x.\ (\!|F,x|\!))$
- *SimpleExOrEnc* $(\lambda x.\ (\!|F,x,\_|\!))$
- *SimpleExOrEnc* $(\lambda x.\ (\!|F,\_,x|\!))$
- *SimpleExOrEnc* $(\lambda x.\ (\!|F,x,\_,\_|\!))$
- *SimpleExOrEnc* $(\lambda x.\ (\!|F,\_,x,\_|\!))$
- *SimpleExOrEnc* $(\lambda x.\ (\!|F,\_,\_,x|\!))$
- *SimpleExOrEnc* $(\lambda x.\ \{\!|x,F|\!\})$

This corresponds exactly to the restriction of ψ to an exemplification or encoding formula in PLM.

3.10.6. Axioms of Actuality

As mentioned in the beginning of the section the modally-fragile axiom of actuality is stated using a different syntax (see A.7.5):

- $[\boldsymbol{\mathcal{A}}\varphi \equiv \varphi]$ (30)

Note that the model finding tool **nitpick** can find a counter-model for the formulation as a regular axiom, as expected.

The remaining axioms of actuality are not modally-fragile and therefore stated as regular axioms:

- $[[\ \mathcal{A}\neg\varphi \equiv \neg\mathcal{A}\varphi\ ]]$ (31.1)
- $[[\ \mathcal{A}(\varphi \to \psi) \equiv (\mathcal{A}\varphi \to \mathcal{A}\psi)\ ]]$ (31.2)
- $[[\ \mathcal{A}(\forall\,\alpha.\ \varphi\ \alpha) \equiv (\forall\,\alpha.\ \mathcal{A}\varphi\ \alpha)\ ]]$ (31.3)
- $[[\ \mathcal{A}\varphi \equiv \mathcal{A}\mathcal{A}\varphi\ ]]$ (31.4)

All of the above can be proven automatically by the *axiom-meta-solver* method.

3.10.7. Axioms of Necessity

The axioms of necessity are the following (see A.7.6):

- $[[\ \Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)\ ]]$ (32.1)
- $[[\ \Box\varphi \to \varphi\ ]]$ (32.2)
- $[[\ \Diamond\varphi \to \Box\Diamond\varphi\ ]]$ (32.3)
- $[[\ \Diamond(\exists\,x.\ (\!|E!,x^P|\!)\ \&\ \Diamond\neg(\!|E!,x^P|\!))\ \&\ \Diamond\neg(\exists\,x.\ (\!|E!,x^P|\!)\ \&\ \Diamond\neg(\!|E!,x^P|\!))\ ]]$ (32.4)

While the first three axioms can be derived automatically, the last axiom requires special attention. On a closer look the formulation may be familiar. The axiom was already mentioned in section 3.4.9 while constructing the representation of the constant $E!$. To be able to derive this axiom here the constant was specifically axiomatized. Consequently the derivation requires the use of these meta-logical axioms stated in the representation layer.

3.10.8. Axioms of Necessity and Actuality

The axioms of necessity and actuality can be derived automatically and require no further attention (see A.7.7):

- $[[\ \mathcal{A}\varphi \to \Box\mathcal{A}\varphi\ ]]$ (33.1)
- $[[\ \Box\varphi \equiv \mathcal{A}(\Box\varphi)\ ]]$ (33.2)

3.10.9. Axioms of Descriptions

There is only one axiom dedicated to descriptions only (note, however, that descriptions play a role in the axioms of quantification). The statement is the following (see A.7.8):

- $[[\ x^P = (\iota x.\ \varphi\ x) \equiv (\forall\,z.\ \mathcal{A}\varphi\ z \equiv z = x)\ ]]$ (34)

Given the technicalities of descriptions already discussed in section 3.10.5 it comes at no surprise that this statement requires a verbose proof.

3.10.10. Axioms of Complex Relation Terms

The axioms of complex relation terms deal with the properties of λ-expressions. Since the *meta-solver* was not equipped with explicit rules for λ-expressions, the statements rely on their semantic properties as described in section 3.5 directly.

The statements are the following (see A.7.9):

- $\boldsymbol{\lambda} x.\ \varphi\ x = \boldsymbol{\lambda} y.\ \varphi\ y$ (36.1)
- $\mathit{IsProperInX}\ \varphi \Longrightarrow [[\ (\!|\boldsymbol{\lambda} x.\ \varphi\ (x^P),x^P|\!) \equiv \varphi\ (x^P)\ ]]$ (36.2)
- $\mathit{IsProperInXY}\ \varphi \Longrightarrow [[\ (\!|\boldsymbol{\lambda}^2\ (\lambda x\ y.\ \varphi\ (x^P)\ (y^P)),x^P,y^P|\!) \equiv \varphi\ (x^P)\ (y^P)\ ]]$ (36.2)
- $\mathit{IsProperInXYZ}\ \varphi \Longrightarrow$
  $[[\ (\!|\boldsymbol{\lambda}^3\ (\lambda x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),x^P,y^P,z^P|\!) \equiv \varphi\ (x^P)\ (y^P)\ (z^P)\ ]]$ (36.2)
- $[[\ \boldsymbol{\lambda}^0\ \varphi = \varphi\ ]]$ (36.3)
- $[[\ (\boldsymbol{\lambda} x.\ (\!|F,x^P|\!)) = F\ ]]$ (36.3)
- $[[\ \boldsymbol{\lambda}^2\ (\lambda x\ y.\ (\!|F,x^P,y^P|\!)) = F\ ]]$ (36.3)
- $[[\ \boldsymbol{\lambda}^3\ (\lambda x\ y\ z.\ (\!|F,x^P,y^P,z^P|\!)) = F\ ]]$ (36.3)
- $(\bigwedge x.\ [\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x)\ \mathit{in}\ v]) \Longrightarrow [[\ \boldsymbol{\lambda}^0\ (\chi\ (\boldsymbol{\iota} x.\ \varphi\ x)) = \boldsymbol{\lambda}^0\ (\chi\ (\boldsymbol{\iota} x.\ \psi\ x))\ ]]$ (36.4)
- $(\bigwedge x.\ [\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x)\ \mathit{in}\ v]) \Longrightarrow [[\ (\boldsymbol{\lambda} x.\ \chi\ (\boldsymbol{\iota} x.\ \varphi\ x)\ x) = (\boldsymbol{\lambda} x.\ \chi\ (\boldsymbol{\iota} x.\ \psi\ x)\ x)\ ]]$ (36.4)
- $(\bigwedge x.\ [\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x)\ \mathit{in}\ v]) \Longrightarrow [[\ \boldsymbol{\lambda}^2\ (\chi\ (\boldsymbol{\iota} x.\ \varphi\ x)) = \boldsymbol{\lambda}^2\ (\chi\ (\boldsymbol{\iota} x.\ \psi\ x))\ ]]$ (36.4)
- $(\bigwedge x.\ [\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x)\ \mathit{in}\ v]) \Longrightarrow [[\ \boldsymbol{\lambda}^3\ (\chi\ (\boldsymbol{\iota} x.\ \varphi\ x)) = \boldsymbol{\lambda}^3\ (\chi\ (\boldsymbol{\iota} x.\ \psi\ x))\ ]]$ (36.4)

Note that the first axiom - α-conversion - could be omitted entirely, since the fact that lambda-expressions are modelled using functions with bound variables and α-conversion is part of the logic of Isabelle/HOL, it already holds implicitly.

Further note that the notion of *proper maps* is used as a necessary precondition for β-conversion, as explained in section 3.4.7.

Lastly note that the formulation of the last class of axioms ((36.4), ι-conversion) has to be adjusted to be representable in the functional setting:

The original axiom is stated as follows in the syntax of PLM:

$$\mathcal{A}(\varphi \equiv \psi) \to ([\lambda x_1 \cdots x_n\ \chi^*] = [\lambda x_1 \cdots x_n\ \chi^{*\prime}]$$

Here $\chi^{*\prime}$ is required to be the result of substituting $\iota x\psi$ for zero or more occurrences of $\iota x\varphi$ in χ^*. In the functional setting χ can be represented as function from individual terms of type κ to propositions of type o. Thereby substituting $\iota x\psi$ for occurences of $\iota x\varphi$ can be expressed as comparing the function application of χ to $\boldsymbol{\iota} x.\ \varphi\ x$ with that to $\boldsymbol{\iota} x.\ \psi\ x$.

Now since φ and ψ are again functions (this time from individuals of type ν to type o) the precondition has to be reformulated to hold for the application of φ and ψ to an arbitrary individual x to capture the concept of $\mathcal{A}(\varphi \equiv \psi)$ in PLM, where φ and ψ may contain x as a free variable.

3.10.11. Axioms of Encoding

The last class of axioms is comprised of the axioms of encoding (see A.7.10):

- $[[\ \{\!|x,F|\!\} \to \Box\{\!|x,F|\!\}\ ]]$ (37)
- $[[\ (\!|O!,x|\!) \to \neg(\exists\, F.\ \{\!|x,F|\!\})\ ]]$ (38)
- $[[\ \exists\, x.\ (\!|A!,x^P|\!)\ \&\ (\forall\, F.\ \{\!|x^P,F|\!\} \equiv \varphi\ F)\ ]]$ (39)

Whereas the first statement (encoding is modally rigid) is a direct consequence of the semantics (recall that the encoding extension of a property was not relativized to possible worlds; see section 3.5), the second axiom (ordinary objects do not encode) is only derivable by expanding the definition of the encoding extension and the meta-logical distinction between ordinary and abstract objects.

Similarly the comprehension axiom for abstract objects depends on the meta-logic and follows from the definition of abstract objects as the power set of relations and the representation of encoding as set membership.

It is again a requirement of the representation in the functional setting that φ in the comprehension axiom is a function and the condition it imposes on F is expressed as its application to F. The formulation in PLM on the other hand has to explicitly exclude a free occurence of x in φ. In the functional setting this is not necessary, since the only way the condition φ imposes could depend on the x bound by the existential quantifier would be that x was given to the function φ as an argument[8].

3.10.12. Summery

Although the formulation of some of the axioms has to be adjusted to work in the environment of functional type theory, it is possible to arrive at a formulation that faithfully represents the original axiom system of PLM. Future work may attempt to further align the representation with PLM and compare the extents of both systems in more detail.

Furthermore a large part of the axioms can be derived independently of the technicalities of the representation layer by only depending on the representation of the semantics described in section 3.5. Future work may explore possible options to further minimize the dependency on the underlying model structure.

To verify that the axiom system is a good representation of the reference system, as a next step the deductive system PLM as described in [12, Chap. 9] is derived solely based on the formulation of the axioms without falling back to the meta-logic.

[8] For that to be possible φ would of course also need to have a different type as a function.

3.11. The Deductive System PLM

The derivation of the deductive system PLM from the constructed axiom system constitutes a major part of the Isabelle Theory in the appendix (see A.9). Its extent of over one hundred pages makes it infeasible to discuss every aspect in full detail here.

Nevertheless it is worthwhile to have a look at the mechanics of the derivation and to highlight some interesting concepts.

3.11.1. Modally Strict Proofs

PLM distinguishes between two sets of theorems, the set of theorems, that are derivable from the complete axiom system, and the set of theorems, that have *modally strict* proofs.

A modally strict proof is a proof that does not use modally fragile axioms (namely the axiom of actuality described in section 3.10.6).

Although it is challenging to completely reproduce this distinction in the embedding (see 5.1.3), the following schema results in a reasonable representation:

Modally strict theorems are stated to be true for an *arbitrary* semantic possible world in the embedding as: $[\varphi \; in \; v]$

Here the variable v implicitly ranges over *any* semantic possible world of type i, including the designated actual world dw. Since modally fragile axioms only hold in dw, they therefore cannot be used to prove a statement formulated this way, as desired.

Non-modally strict theorems on the other hand are stated to be true only for the designated actual world: $[\varphi \; in \; dw]$

This way necessary axioms, as well as modally fragile axioms can be used in the proofs. However it is not possible to infer from a modally fragile theorem that the same statement holds as a modally strict theorem

It is important to note that the set of modally-strict theorems in PLM is in fact a *subset* of the theorems of the form $[\varphi \; in \; v]$ that are semantically true in the embedding. However, the rules introduced in PLM are stated in such a way that only this subset is in fact derivable without falling back to the semantics or the meta-logic. This is discussed in more detail in section 5.1.3.

3.11.2. Fundamental Metarules of PLM

The primitive rule of PLM is the modus ponens rule (see A.9.2):

- $[\varphi \; in \; v] \wedge [\varphi \rightarrow \psi \; in \; v] \Longrightarrow [\psi \; in \; v]$ (41)

In the embedding this rule is a direct consequence of the semantics of the implication.

Additionally two fundamental Metarules are derived in PLM, *GEN* and *RN* (see A.9.5):

- $(\bigwedge\alpha. \; [\varphi \; \alpha \; in \; v]) \Longrightarrow [\forall \alpha. \; \varphi \; \alpha \; in \; v]$ (49)

- $[\![\bigwedge w.\ [\varphi\ in\ w] \Longrightarrow [\psi\ in\ w];\ [\Box\varphi\ in\ v]]\!] \Longrightarrow [\Box\psi\ in\ v]$ (51)

Although in PLM these rules are derived by structural induction on the complexity of a formula, unfortunately this proving mechanism cannot be reproduced in Isabelle. However, the rules are direct consequences of the semantics described in section 3.5. The same is true for the deduction rule (see A.9.6):

- $([\varphi\ in\ v] \Longrightarrow [\psi\ in\ v]) \Longrightarrow [\varphi \rightarrow \psi\ in\ v]$ (54)

As a consequence this rule is derived from the semantics of the implication as well.

These rules are the *only* exceptions to the concept that the deductive system of PLM is completely derived from the axiom system and the primitive rule of inference (modus ponens).

3.11.3. PLM Solver

Similarly to the *meta-solver* described in section 3.8 another proving method is introduced, namely the *PLM-solver* (see A.9.1).

This proving method is initially not equipped with any rules. Throughout the derivation of the deductive system of PLM, appropriate rules of PLM are added to its set of rules. Furthermore at several instances further rules become trivially *derivable* from the statements of PLM proven so far. Such rules are added to the *PLM-solver* as well.

Additionally the *PLM-solver* can instantiate any rule of the deductive system PLM proven so far as well as any axiom, if this allows to resolve a proving objective.

By its construction the *PLM-solver* has the property, that it can *only* prove statements that are derivable from the deductive system PLM. Thereby it is safe to use to aid in any proof throughout the section. As a matter of fact a lot of the simple tautological statements derived in the beginning of [12, Chap. 9] can be proven completely automatically using this method.

Unfortunately some rules that would make the solving method more powerful and would make it more helpful in the derivation of more complex theorems, are *not* derivable from the deductive system itself (and consequently are *not* added to the set of admissible rules). Namely one example is the converse of the rule RN. This circumstance is discussed in more detail in section 5.1.3.

3.11.4. Additional Type Classes

There is one further subtlety one may notice in the derivation of the deductive system. In PLM it is possible to derive statements involving the general identity symbol by case distinction: if such a statement is derivable for all types of terms in the language separately, it can be concluded that it is derivable for the identity symbol in general. Such a case distinction cannot be directly reproduced in the embedding, since it cannot be assumed, that every instantiation of the type class *identifiable* is in fact one of the types of terms of PLM.

However, there is a simple way to still formulate such general statements. This is done by the introduction of additional type classes. A simple example is the type class *id-eq* (see A.9.7). This new type class assumes the following statements to be true:

- $[\alpha = \alpha \ in \ v]$ (71.1)
- $[\alpha = \beta \rightarrow \beta = \alpha \ in \ v]$ (71.2)
- $[\alpha = \beta \ \& \ \beta = \gamma \rightarrow \alpha = \gamma \ in \ v]$ (71.3)

Since these statements can be derived *separately* for the types ν, Π_0, Π_1, Π_2 and Π_3, the type class *id-eq* can now trivially be instantiated for each of these types.

3.11.5. The Rule of Substitution

A challenge in the derivation of the deductive system that is worth to examine in detail is the *rule of substitution*. The rule is stated in PLM as follows (see (113)[12]):

> If $\vdash_\Box \psi \equiv \chi$ and φ' is the result of substituting the formula χ for zero or more occurrences of ψ where the latter is a subformula of φ, then if $\Gamma \vdash \varphi$, then $\Gamma \vdash \varphi'$. [Variant: If $\vdash_\Box \psi \equiv \chi$, then $\varphi \vdash \varphi'$]

Naively one could try to express this in the functional setting as follows:

$$(\bigwedge v. \ [\psi \equiv \chi \ in \ v]) \Longrightarrow [\varphi \ \psi \ in \ v] = [\varphi \ \chi \ in \ v]$$

However this statement would *not* be derivable. The issue is connected to the restriction of ψ being a *subformula* of φ in PLM. The formulation above would allow the substitution for *any function* φ from formulas to formulas.

Formulas in the embedding have type o which is internally represented by functions of the type $j \Rightarrow i \Rightarrow bool$. Reasoning is only defined to be classically in the designated actual state dj, though. In the formulation above nothing would prevent φ from being a function with the following internal representation: $\lambda\psi \ s \ w. \ \forall s. \ \psi \ s \ w$

So nothing prevents φ from evaluating its argument for a state different from the designated actual state dj. The condition $\bigwedge v. \ [\psi \equiv \chi \ in \ v]$ on the other hand only requires ψ and χ to be equivalent in all possible worlds particularly in the *actual state* - no statement about other states is implied.

Another issue arises if one considers one of the example cases of legitimate uses of the rule of substitution in PLM (see [12, (113)]):

> If $\vdash \exists x \ A!x$ and $\vdash_\Box A!x \equiv \neg\Diamond E!x$, then $\vdash \exists x \ \neg\Diamond E!x$.

This does not follow from the naive formulation above. Since x is *bound* by the existential quantifier, in the functional representation φ has to have a different type: to be applicable in this example φ has to be $\varphi = (\lambda\psi. \ \exists x. \ \psi \ x)$, whereas ψ and χ themselves have to be functions as follows: $\psi = (\lambda x. \ (|A!,x|))$ and $\chi = (\lambda x. \ \neg\Diamond(|E!,x|))$. Furthermore now the equivalence condition has to be $\bigwedge x \ v. \ [\psi \ x \equiv \chi \ x \ in \ v]$. This is analog to the fact that x is a free variable in the condition $\vdash_\Box A!x \equiv \neg\Diamond E!x$ in PLM.

Solution

The embedding employs a solution that is complex, but can successfully address the issues described above.
First of all the following definition is introduced:

$$\textit{Substable cond } \varphi = (\forall \psi\ \chi\ v.\ \textit{cond } \psi\ \chi \longrightarrow [\varphi\ \psi \equiv \varphi\ \chi\ \textit{in } v])$$

Given a condition *cond* a function φ is considered *Substable*, if and only if for all ψ and χ (which may have an arbitrary type) it follows in each possible world v that $[\varphi\ \psi \equiv \varphi\ \chi\ \textit{in } v]$.
Now several introduction rules for this property are derived. The idea is to capture the notion of *subformula* in PLM. A few examples are:

- $\textit{Substable cond } (\lambda\varphi.\ \Theta)$
- $\textit{Substable cond } \psi \Longrightarrow \textit{Substable cond } (\lambda\varphi.\ \neg\psi\ \varphi)$
- $\textit{Substable cond } \psi \wedge \textit{Substable cond } \chi \Longrightarrow \textit{Substable cond } (\lambda\varphi.\ \psi\ \varphi \rightarrow \chi\ \varphi)$

These rules can be derived from the rules proven in PLM.
Now as mentioned above in the functional setting substitution has to be allowed not only for formulas, but also for *functions* to formulas. To that end the type class *Substable* is introduced that fixes a condition *Substable-Cond* to be used as *cond* in the definition above and assumes the following:

$$\textit{Substable Substable-Cond } \varphi \wedge \textit{Substable-Cond } \psi\ \chi \wedge \Theta\ [\varphi\ \psi\ \textit{in } v] \Longrightarrow \Theta\ [\varphi\ \chi\ \textit{in } v]$$

If φ is *Substable* (as per the definition above) under the condition *Substable-Cond* that was fixed in the type class, and ψ and χ satisfy the fixed condition *Substable-Cond*, then everything that is true for $[\varphi\ \psi\ \textit{in } v]$ is also true for $[\varphi\ \chi\ \textit{in } v]$.
Now as a base case this type class is *instantiated* for the type of formulas o with the following definition of *Substable-Cond*:

$$\textit{Substable-Cond } \psi\ \chi = (\forall v.\ [\psi \equiv \chi\ \textit{in } v])$$

Furthermore the type class is instantiated for *functions* from an arbitrary type to a type of the class *Substable* with the following definition of *Substable-Cond*:

$$\textit{Substable-Cond } \psi\ \chi = (\forall x.\ \textit{Substable-Cond } (\psi\ x)\ (\chi\ x))$$

This construction now allows substitutions in all cases required by the original formulation in PLM.

Proving Methods

Although the construction above covers exactly the cases in which PLM allows substitutions, it does not yet have a form that allows it to conveniently *apply* the rule of substitution. In order to apply the rule, it first has to be established that a formula can be decomposed into a function with the substituents as arguments and it further has to be shown that this function satisfies the appropriate *Substable* condition. This complexity prevents any reasonable use cases. This problem is mitigated by the introduction of proving methods, that are convenient to use. The main method is called *PLM-subst-method*.

This method uses a combination of pattern matching and automatic rule application to provide a convenient way to apply the rule of substitution in practice.

For example assume the current proof objective is $[\neg\neg\Diamond(\!|E!,x|\!)\ in\ v]$. Now it is possible to apply the *PLM-subst-method* method as follows:

$$\textbf{apply}\ (\mathit{PLM\text{-}subst\text{-}method}\ (\!|A!,x|\!)\ (\neg(\Diamond(\!|E!,x|\!))))$$

This will now automatically analyze the current proof goal, look for an appropriate choice of a function φ, apply the substitution rule and resolve the substitutability claim about φ. Consequently it can resolve the current proof objective by producing two new proving goals: $\forall\, v.\ [(\!|A!,x|\!) \equiv \neg\Diamond(\!|E!,x|\!)\ in\ v]$ and $[\neg(\!|A!,x|\!)\ in\ v]$, as expected. The complexity of the construction above is hidden away entirely.

Similarly assume the proof objective is $[\exists\, x.\ \neg\Diamond(\!|E!,x^P|\!)\ in\ v]$. Now the method *PLM-subst-method* can be invoked as follows:

$$\textbf{apply}\ (\mathit{PLM\text{-}subst\text{-}method}\ \lambda x\ .\ (\!|A!,x^P|\!)\ \lambda x\ .\ (\neg(\Diamond(\!|E!,x^P|\!))))$$

This will result in the new proving goals: $\forall\, x\ v.\ [(\!|A!,x^P|\!) \equiv \neg\Diamond(\!|E!,x^P|\!)\ in\ v]$ and $[\exists\, x.\ (\!|A!,x^P|\!)\ in\ v]$, as desired.

Conclusion

Although an adequate representation of the rule of substitution in the functional setting is challenging, the above construction allows a convenient use of the rule. Moreover it is important to note that despite the complexity of the representation no assumptions about the meta-logic or the underlying model structure were made. The construction is completely derivable from the rules of PLM itself, so the devised rule is safe to use without compromising the provability claim of the layered structure of the embedding.

All statements that are proven using the constructed substitution methods, remain derivable from the deductive system of PLM.

3.11.6. An Example Proof

To illustrate how the derivation of theorems in the embedding works in practice, consider the following example[9]:

lemma $[\Box(\varphi \rightarrow \Box\varphi) \rightarrow ((\neg\Box\varphi) \equiv (\Box(\neg\varphi)))\ \mathit{in}\ v]$
proof (*rule CP*)
  assume $[\Box(\varphi \rightarrow \Box\varphi)\ \mathit{in}\ v]$
  hence $[(\neg\Box(\neg\varphi)) \equiv \Box\varphi\ \mathit{in}\ v]$
    by (*metis sc-eq-box-box-1 diamond-def vdash-properties-10*)
  thus $[((\neg\Box\varphi) \equiv (\Box(\neg\varphi)))\ \mathit{in}\ v]$
    by (*meson CP* $\equiv$*I* $\equiv$*E* $\neg\neg$*I* $\neg\neg$*E*)
qed

Since the statement is an implication it is derived using a *conditional proof*. To that end the proof statement already applies the initial rule *CP*.

The proof objective inside the proof body is now $[\Box(\varphi \rightarrow \Box\varphi)\ \mathit{in}\ v] \Longrightarrow [\neg\Box\varphi \equiv \Box\neg\varphi\ \mathit{in}\ v]$, so $[\neg\Box\varphi \equiv \Box\neg\varphi\ \mathit{in}\ v]$ has to be shown under the assumption $[\Box(\varphi \rightarrow \Box\varphi)\ \mathit{in}\ v]$. Therefore the first step is to assume $[\neg\Box\varphi \equiv \Box\neg\varphi\ \mathit{in}\ v]$.

The second statement can now be automatically derived using the previously proven theorem *sc-eq-box-box-1*, the definition of the diamond operator and a deduction rule. The final proof objective follows from a combination of introduction and elimination rules.

Note that the automated reasoning tool **sledgehammer** can find proofs for the second and final statement automatically. It can even automatically find a proof for the entire theorem resulting in the following proof:

lemma $[\Box(\varphi \rightarrow \Box\varphi) \rightarrow ((\neg\Box\varphi) \equiv (\Box(\neg\varphi)))\ \mathit{in}\ v]$
  by (*metis* $\equiv$*I CP* $\equiv$*E(1)* $\equiv$*E(2) raa-cor-1 sc-eq-box-box-1 diamond-def*)

So it can be seen that the embedding can be used to interactively prove statements with the support of automated reasoning tools and often even complete proofs for complex statements can be found automatically. The dependencies of a proof are given explicitly in the proof statement.

3.11.7. Summery

Despite the presented challenges it was possible to derive a full representation of the deductive system PLM, as described in [12, Chap. 9] without sacrificing the layered structure of the embedding.

Although compromises affecting the degree of automation had to be made, the resulting representation can conveniently be used for the interactive construction of complex proofs while retaining the support of the automation facilities of Isabelle/HOL.

[9]Since the whole proof is stated as raw Isabelle code, unfortunately no color-coding can be applied.

3.12. Artificial Theorems

Although the layered approach of the embedding provides the means to derive theorems independently of the representation layer and model structure, it is interesting to consider some examples of theorems that are *not* part of PLM, but can be derived in the embedding using the meta-logic.

3.12.1. Non-Standard λ-Expressions

The following statement involves a λ-expressions that contains encoding subformulas and is consequently not part of PLM (see A.11):

$$[(\!|\boldsymbol{\lambda} x.\ \{\!|F^P, y|\!\}, x^P|\!) \equiv \{\!|F^P, y|\!\}\ in\ v]$$

In this case traditional β-conversion still holds, since the λ-expression does not contain encoding expressions involving their bound variable[10]. On the other hand the following is *not* a theorem in the embedding (the tool **nitpick** can find a counter-model):

$$[(\!|\boldsymbol{\lambda} x.\ \{\!|x^P, F|\!\}, x^P|\!) \rightarrow \{\!|x^P, F|\!\}\ in\ v]$$

Instead the following generalized versions of β-conversion are theorems in the embedding:

- $[(\!|\boldsymbol{\lambda} x.\ \{\!|x^P, F|\!\}, z^P|\!)\ in\ v] = (\exists\, y.\ \nu\upsilon\ y = \nu\upsilon\ z \wedge [\{\!|y^P, F|\!\}\ in\ v])$
- $[(\!|\boldsymbol{\lambda} x.\ \varphi\ (x^P), z^P|\!)\ in\ v] = (\exists\, y.\ \nu\upsilon\ y = \nu\upsilon\ z \wedge [\varphi\ (y^P)\ in\ v])$

These theorems can be equivalently stated purely in the embedded logic:

- $[(\!|\boldsymbol{\lambda} x.\ \{\!|x^P, F|\!\}, z^P|\!) \equiv (\exists\, y.\ (\forall\, F.\ (\!|F, z^P|\!) \equiv (\!|F, y^P|\!))\ \&\ \{\!|y^P, F|\!\})\ in\ v]$
- $[(\!|\boldsymbol{\lambda} x.\ \varphi\ (x^P), z^P|\!) \equiv (\exists\, y.\ (\forall\, F.\ (\!|F, z^P|\!) \equiv (\!|F, y^P|\!))\ \&\ \varphi\ (y^P))\ in\ v]$

Especially the second statement shows that in general λ-expressions in the embedding have a *non-standard* semantics. As a special case, however, the behavior of λ-expressions *is* classical if restricted to proper maps:

$$IsProperInX\ \varphi \Longrightarrow [(\exists\, y.\ (\forall\, F.\ (\!|F, x^P|\!) \equiv (\!|F, y^P|\!))\ \&\ \varphi\ (y^P)) \equiv \varphi\ (x^P)\ in\ v]$$

It is important to note that as a consequence of the generalized β-conversion there are theorems in the embedding involving λ-expressions that *do* contain encoding subformulas in the bound variable, e.g.:

$$[(\!|\boldsymbol{\lambda} x.\ \{\!|x^P, F|\!\} \equiv \{\!|x^P, F|\!\}, y^P|\!)\ in\ v]$$

A further discussion about this topic is found in section 5.1.1.

[10]Consequently the matrix is a *proper map*.

3.12.2. Consequences of the Aczel-model

Independently the following theorem is a consequence of the constructed Aczel-model:

$$[\forall F.\ (\!|F,a^P|\!) \equiv (\!|F,b^P|\!)\ \mathit{in}\ v] \Longrightarrow \boldsymbol{\lambda}x.\ (\!|R,x^P,a^P|\!) = \boldsymbol{\lambda}x.\ (\!|R,x^P,b^P|\!)$$

The reason for this theorem to hold is that the condition on a and b forces the embedding to map both objects to the same urelement. By the definition of exemplification the presented λ-expressions only depend on this urelement, therefore they are forced to be equal. Neither the deductive system of PLM nor its formal semantics require this equality.

Separate research suggests that this artificial theorem can be avoided by extending the embedding in the following way: the mapping from abstract objects to special urelements constructed in section 3.4.3 can be modified to depend on a *state*. This way the condition used in the theorem only implies that a and b are mapped to the same urelement in the *actual state*. Since they can still be mapped to different urelements in different states, the derived equality no longer holds.

This extension of the embedding increases the complexity of the representation layer slightly, but its preliminary analysis suggests that it presents no further issues, so future research and future versions of the embedding may want to include such a modification.

Note that there may still be further theorems that depend on the particular model structure that are yet to be discovered.

3.13. Sanity Tests

The consistency of the constructed embedding can be verified by the model-finding tool **nitpick** (see A.12.1). Since the main construction of the embedding is definitional and only a minimal set of meta-logical axioms is used, this is expected.

The hyperintensionality of the constructed model can be verified for some simple example cases. The following statements have counter-models (see A.12.2):

- $[(\boldsymbol{\lambda}y.\ q \vee \neg q) = (\boldsymbol{\lambda}y.\ p \vee \neg p)\ \mathit{in}\ v]$
- $[(\boldsymbol{\lambda}y.\ p \vee q) = (\boldsymbol{\lambda}y.\ q \vee p)\ \mathit{in}\ v]$

Furthermore the meta-logical axioms stated in section 3.4.9 can be justified (see A.12.4):

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

The first axiom is equivalent to the fact that concreteness matches the domains of ordinary, resp. abstract objects, whereas the second and third axiom corresponds to the conjuncts of axiom (32.4)[12].

Additionally some further desirable meta-logical properties of the embedding are verified in A.12.5 and A.12.6.

4. Technical Issues

Although the presented embedding shows that the generic proof assistant Isabelle/HOL offers a lot of flexibility in expressing even a very complex and challenging theory as the Theory of Abstract Objects, it has some limitations that required compromises in the formulation of the theory.

In this chapter some of these limitations and their consequences for the embedding are discussed. Future versions of Isabelle may allow a clearer implementation especially of the layered approach of the embedding.

4.1. Limitations of Type Classes and Locales

Isabelle provides a powerful tool for abstract reasoning called **locale**. Locales are used for *parametric* reasoning. Type classes as already described briefly in section 3.6 and further mentioned in sections 3.9 and 3.11.4 are in fact special cases of locales that are additionally connected to Isabelle's primitive type system.

The definition of a locale defines a set of constants that can use arbitrary type variables (type classes on the other hand are restricted to only one type variable). Furthermore assumptions about these constants can be postulated that can be used in the reasoning within the locale. Similarly to the instantiation of a type class a locale can be *interpreted* for specific definitions of the introduced constants, for which it has to be proven that they satisfy the postulated assumptions.

Thereby it is possible to reason about abstract structures that are solely characterized by a specific set of assumptions. Given that it can be shown that these assumptions are satisfied for a concrete case (i.e. a specific definition for the constants using specific types in place of the used type variables), an interpretation of the locale allows the use of all theorems shown for the abstract case in the concrete application.

Therefore in principle locales would be a perfect fit for the layered structure of the embedding: If the representation of the formal semantics and the axiom system could both be formulated as locales, it could first be shown that the axiom system is a *sublocale* of the formal semantics, i.e. every set of constants that satisfies the requirements of the formal semantics also satisfies the requirements of the axiom system, and further the formal semantics could be interpreted for the concrete model structure.

Since the reasoning within a locale cannot use further assumptions that are only satisfied by a specific interpretation, this way the universality of the reasoning based on the axiom system could be formally guaranteed - no proof that is solely based on the axiom locale could use any meta-logical statement tied to the underlying representation layer and model structure.

However, a major issue arises when trying to formulate the axiom system as a locale. The axioms of quantification and the substitution of identicals are restricted to only hold for specific sets of types. This already makes it impossible to introduce a general binder for all-quantification or a general identity symbol. A constant for identity would have to be introduced with a specific type. Although this type could use type variables, e.g. $'a \Rightarrow 'a \Rightarrow 'o$, the type variable $'a$ would be fixed throughout the locale.

Several solutions to this problem could be considered: identity could be introduced as a polymorphic constant *outside the locale* and the locale would assume some properties of this constant for specific type variables. Before interpreting the locale the polymorphic constant could then be *overloaded* for concrete types in order to be able to satisfy the assumptions. However, this way it would still be impossible to prove a general statement about identity: every statement would have to be restricted to a specific type, because in general no assumptions about the properties of identity could be made.

Another solution would be to refrain from using general quantifiers and identity relations altogether, but to introduce separate binders and identity symbols for the type of individuals and each relation type. This would, however, add a significant amount of notational complexity to the embedding and would require to duplicate all statements that hold for quantification and identity in general for every specific type. For this reason this option was not explored further.

It could also be considered to introduce the axioms of quantification and identity separately from the axiom locale in a type class. An interpretation of the complete axiom system would then have to interpret the axiom locale, as well as instantiate the respective type classes. Since type classes can only use one type variable, this would make it impossible to use a type variable for truth values in the definition of the respective type classes, though.

Several other concepts were considered during the construction of the embedding, but no solution was found that would both accurately represent the axiom system and still be notationally convenient. A complete account of the considered options would go beyond the scope of this discussion.

The most natural extension of Isabelle's locale system that would solve the described issues, would be the ability to introduce polymorphic constants in a locale that can be restricted to a type class (resp. a *sort*). The type class could potentially even be introduced simultaneously with the locale. However, such a construction is currently not possible in Isabelle and as of yet it is unknown whether the internal type system of Isabelle would allow such an extension in general.

4.2. Case Distinctions by Type

Although a general all-quantifier and identity relation can be approximated using type classes as described in sections 3.6 and 3.9, in fact this construction is conceptually different from the intention of PLM. The identity relation is not actually meant to be determined by some set of properties, but by their definition for specific concrete types.

However, currently Isabelle does not allow the restriction of a type variable in a statement to a specific set of types. Type variables can only be restricted to specific *sorts*, so effectively to type classes. As mentioned in section 3.11.4 this means that statements for example about the general identity relation that depend on the specific definitions of identity for the concrete types, cannot be proven as in PLM by case distinction on types, but another type class has to be introduced that *assumes* the statement, which then has to be instantiated for the concrete types.

Although the solution using type classes works for the embedding, it would be more natural to restrict such statements to the specific sets of types and to have an induction method that allows to prove the statement for the concrete types separately. Again as of yet it is unknown whether Isabelle could be extended in such a way given the limitations of its internal type system.

4.3. Structural Induction and Proof-Theoretic Reasoning

As mentioned in section 3.11.2 some of the meta-rules that PLM can *derive* by induction on the length of a derivation, e.g. the deduction theorem $([\varphi\ in\ v] \Longrightarrow [\psi\ in\ v]) \Longrightarrow [\varphi \rightarrow \psi\ in\ v]$, have to be proven using the semantics instead in the embedding.

While this is not considered a major problem, it would be interesting to investigate, whether some construction in Isabelle would in fact allow proof-theoretic reasoning similar to the proofs in PLM. This is related to the issue of accurately representing the concept of *modally-strict proofs* as described in sections 3.11.1 and 5.1.3.

5. Discussion and Results

5.1. Differences between the Embedding and PLM

Although the embedding attempts to represent the language and logic of PLM as precisely as possible, it is important to note that there remain differences between PLM and its representation in Isabelle/HOL. Some of the known differences are discussed in the following sections. A complete analysis of the precise relation between PLM and the embedding unfortunately goes beyond the scope of this thesis, but constitutes a highly interesting and relevant topic for future research.

5.1.1. Propositional Formulas and λ-Expressions

The main difference between the embedding and PLM is the fact that the embedding does not distinguish between propositional and non-propositional formulas.

This purely syntactical distinction is challenging to reproduce in a shallow embedding that does not introduce the complete term structure of the embedded language directly. Instead the embedding attempts to analyse the semantic reason for the syntactic distinction and to devise a semantic criterion that can be used as a replacement for the syntactic restriction.

The identified issue, that is addressed by the syntactic restriction in PLM, is described in section 3.2: Allowing non-propositional formulas in β-convertible λ-expressions without restriction leads to paradoxes.

Since the embedding is known to be consistent, the issue presents itself in a slightly different fashion: the paradox is constructed under the assumption that β-conversion holds unconditionally for all λ-expressions. In the embedding on the other hand in general λ-expressions have a *non-standard* semantics and β-conversion only follows as a special case (see 3.12.1). Thereby the consistency of the system is preserved.

With the definition of *proper maps* (see 3.4.7), the embedding constructs a necessary and sufficient condition on functions that may serve as matrix of a λ-expression while allowing β-conversion.

The idea of this concept was that every λ-expression that is syntactically well-formed in PLM would have a proper map as its matrix. Two subtleties have to be considered, though:

It was discovered that there are λ-expressions that are part of PLM, whose matrix does not correspond to a proper map in the embedding. The analysis of this issue led to the discovery of a paradox in the formulation of PLM and is discussed in more detail in section 5.2. As a consequence these cases will not constitute proper λ-expressions in future versions of PLM.

The remaining subtlety is the fact that there are proper maps, that do not correspond to propositional formulas. Some examples have already been mentioned in section 3.12.1. Therefore the embedding contains more proper, β-convertible λ-expressions than PLM. This effectively means that the theory constructed in the embedding is more general than the theory in PLM.

Consequently future research may on the one hand investigate available options to restrict the embedding further in order to arrive at a smaller set of relations that is in correspondence with PLM. On the other hand the embedding suggests that it is consistently possible to extend the theory of PLM by allowing a larger set of relations. Therefore it would be interesting to further analyse the philosophical implications of such an extension and to investigate the question, which new interesting theorems can be derived in such an extended theory.

5.1.2. Terms and Variables

In PLM an individual term can be an individual variable, an individual constant or a definite description. A large number of statements is formulated using specific variables. From such a statement its universal generalization can be derived (using the rule GEN), which then can be instantiated for any individual term, given that it denotes ($\exists\beta\ \beta = \tau$).

As already mentioned in sections 3.4.2 and 3.10.5 the embedding uses a slightly different approach. In the embedding individual variables and individual terms have different *types* and an individual variable (of type ν) has to be converted to an individual term (of type κ) using the decoration $\_^P$, so that it can be used for example in an exemplification formula (which is defined for the type κ).

The technicalities of this approach and a discussion about the accuracy of this representation were already given in the referenced sections, so at this point it suffices to summarize the resulting differences between the embedding and PLM:

- The individual variables of PLM are represented as variables of type ν in the embedding.
- Individual constants can be represented by declaring a constant of type ν.
- Meta-level variables (like τ) ranging over all individual terms in PLM can be represented as variables of type κ.
- Objects of type ν have to be explicitly converted to objects of type κ if they are to be used in a context that allows general individual terms.
- The axioms of quantification are adjusted to go along with this representation (see 3.10.5).

In PLM the situation for relation variables, constants and terms is analog. However the embedding uses the following simplification in order to avoid the additional complexity introduced for individuals:

Since at the time of writing PLM unconditionally asserts $\exists \beta\ \beta = \tau$ is for any relation term by an axiom, the embedding uses only one type Π_n for each arity of relations. Therefore no special type conversion between variables and terms is necessary and every relation term can immediately be instantiated for a variable of type Π_n. This hides the additional steps PLM employs for such instantiations (the generalization by GEN followed by an instantiation using quantification theory). Since $\exists \beta\ \beta = \tau$ holds unconditionally for relation terms, this simplification is justified.

The recent developments described in section 5.2, however, suggest that $\exists \beta\ \beta = \tau$ will likely no longer hold unconditionally for every relation term in future versions of PLM, so future versions of the embedding will have to include a distinction between relation terms and relation variables in a similar way as it is already done for individuals.

5.1.3. Modally-strict Proofs and the Converse of RN

As already mentioned in section 3.11.1, modally-strict theorems in the embedding are stated in the form $[\varphi\ in\ v]$ for arbitrary v. On the other hand more statements of the form $[\varphi\ in\ v]$ are semantically true in the embedding, than would be derivable using modally-strict proofs in PLM.

To understand this circumstance and the solution for this issue used in the embedding, the converse of the meta-rule RN has to be considered.

In PLM the metarule RN states in essence that if there is a modally strict proof for φ, then $\Box\varphi$ is derivable as a theorem. Therefore the converse of RN would be that if $\Box\varphi$ is derivable, then there is a modally strict proof for φ.

The discussion in remark (185)[12] shows that this is not true in PLM. However in the embedding the following is derivable from the semantics of the box operator:

$$[\Box\varphi\ in\ dw] \Longrightarrow \forall\, v.\ [\varphi\ in\ v]$$

So although the converse of RN is not true in PLM, an equivalent statement for theorems of the form $[\varphi\ in\ v]$ in the embedding can be derived from the semantics.

The reason for this is that modally-strict theorems are in fact a subset of a larger class of theorems, namely the theorems that are *necessarily true*. Semantically a statement of the form $[\varphi\ in\ v]$ in the embedding is derivable, whenever φ is a *necessary theorem*.

Modally-strict theorems in PLM on the other hand are defined as a proof-theoretic concept: modally-strict proofs are not allowed to use modally fragile axioms. Therefore they are solely derived from axioms whose necessitations are axioms as well (see 3.10.1). PLM now proves the fact that a modally strict derivation of φ implies that there is a derivation of $\Box\varphi$ by induction on the length of the proof. However, remark (185)[12] gives an example of a case in which the converse is false.

The problem for the embedding is that there is no semantic characterization of a statement that allows to decide whether it is a necessary theorem or a modally-strict theorem. Therefore the embedding has to express modally-strict theorems as necessary theorems. As seen above for this set of theorems the converse of RN is in fact true.

This still does not compromise the concept that any statement that is derived in A.9 is also derivable in PLM: the basis of this concept is that no proofs may rely on the meta-logic, but only the rules that are derived in PLM are allowed. To guarantee that no statement of the form $[\varphi \textit{ in } v]$ is derived that is *not* a modally-strict theorem of PLM, the converse of RN is not stated as an admissible rule for these proofs.

Unfortunately this has the consequence that the proving method *PLM-solver* cannot be equipped with a reversible elimination rule for the box-operator, which significantly reduces its power as an automated proving method. However, preserving the claim that theorems derived in the embedding are also theorems of PLM was considered to be more important.

5.2. A Paradox in PLM

During the analysis of the constructed embedding it was discovered, that the formulation of the theory in PLM at the time of writing allowed paradoxical constructions.

This section first describes the process that led to the discovery of the paradox and the role the embedding played in it, after which the construction of the paradox is outlined in the language of PLM.

The paradox has since been confirmed by Edward Zalta and a vivid discussion about the repercussions and possible solutions has developed. At the time of writing it has become clear that there are several options to recover from the paradox while in essence retaining the full set of theorems of PLM. So far no final decision has been reached about which option will be implemented in future versions of PLM.

5.2.1. Discovery of the Paradox

The discovery of the paradox originates in the analysis of the concept of *proper maps* in the embedding and its relation to propositional formulas allowed in the matrix of λ-expressions in PLM (see 5.1.1).

While verifying the concept, that the matrix of every λ-expression allowed in PLM corresponds to a proper map in the embedding, it was discovered, that λ-expressions of the form $[\lambda y\; F\iota x(y[\lambda z\; Rxz])]$ in which the bound variable y occurs in an encoding formula inside the matrix of a definite description, were part of PLM, but their matrix was *not* a proper map in the embedding and therefore β-conversion was not derivable for these terms.

In the attempt to proof that β-conversion would in fact follow for these expressions in the embedding as well, it first became clear, that this is not the case without a restriction of the Aczel-model. Namely without some kind of restriction of the map from abstract objects to urelements, β-conversion would not be derivable for these cases.

In order to better understand how the embedding could accommodate these circumstances, the consequences of allowing β-conversion in the mentioned cases *by assumption* were further studied in the embedding which led to the first proof of inconsistency (see A.13.4):

$$(\bigwedge G\ \varphi.\ \mathit{IsProperInX}\ (\lambda x.\ (\!|G,\iota y.\ \varphi\ y\ x|\!))) \Longrightarrow \mathit{False}$$

Under the assumption that $\lambda x.\ (\!|G,\iota y.\ \varphi\ y\ x|\!)$ is a proper map for arbitrary G and φ, *False* is derivable in the embedding. However λ-expressions with the equivalent of such maps as matrix were in fact part of PLM.

Since the meta-logic was not used in the proof, it was possible to reconstruct an equivalent proof in the language of PLM, which then served as the basis for further discussions with Edward Zalta.

Since then the issue leading to the paradox was identified as the *description backdoor* (see A.13.2) that can be used to construct a variety of paradoxical cases, e.g. the paradox mentioned in section 3.2 can be reconstructed. This refined version of the paradox is used in the inconsistency proof in A.13.3 and reconstructed in the language of PLM in the next section, while repeating the general situation leading to the paradox without referring to the particularities of the embedding.

5.2.2. Construction using the Language of PLM

Object theory distinguishes between propositional and non-propositional formulas. Propositional formulas are not allowed to contain encoding subformulas, so for example $\exists F\ xF$ is not propositional. Only propositional formulas can be the matrix of a λ-expression, so $[\lambda x\ \exists F\ xF]$ is not a valid term of the theory - it is excluded syntactically.

The reason for this is that considering $[\lambda x\ \exists F\ xF\ \&\ \neg Fx]$ a valid, denoting λ-expression for which β-conversion holds would result in a paradox as described in section 3.2.

The idea was that not allowing non-propositional formulas in λ-expressions would be sufficient to exclude *all* cases that lead to inconsistencies. This was shown to be incorrect, though.

The problem is the *description backdoor*. The syntactical definition of propositional formulas does allow expressions of the following form: $[\lambda y\ F\iota x\psi]$ where ψ does not have to be propositional itself, but can be *any* formula. This is due to the definition of *subformula*: by this definition ψ is *not* a subformula of $F\iota x\psi$, so ψ *may* contain encoding subformulas itself and $F\iota x\psi$ is still considered to be a propositional formula.

This was deemed to be no problem and for cases like $[\lambda y\ F\iota x(xG)]$ as they are mentioned in PLM this is indeed true.

It had not been considered that y may appear within the matrix of such a description and more so, it may appear in an encoding formula within the matrix of such a description, for example $[\lambda y\ F\iota x(xG\ \&\ yG)]$ is still a propositional formula. At least it had not been considered that this is a problem, but:

This way the following construction is possible:

$$[\lambda y\ [\lambda z\ \forall p(p \to p)]\iota x(x = y\ \&\ \psi)] \tag{1}$$

Here ψ can be an arbitrary non-propositional formula in which x and y may be free and 1 is still a valid, denoting λ-expression for which β-conversion holds.

Now it is possible to show that by β-conversion and description theory the following is derivable:

$$[\lambda y\ [\lambda z\ \forall p(p \to p)]\iota x(x = y\ \&\ \psi)]x \equiv \psi^x{}_y \tag{2}$$

Remark. *Using a modally-strict proof only the following is derivable:*
$[\lambda y\ [\lambda z\ \forall p(p \to p)]\iota x(x = y\ \&\ \psi)]x \equiv \mathcal{A}\psi^x{}_y$
For the construction of the paradox, the modally fragile statement is sufficient, though. Note, however, that it is also possible to construct similar paradoxical cases without appealing to any modally-fragile axioms or theorems.

This effectively undermines the intention of restricting λ-expressions to only propositional formulas:

Although $[\lambda x\ \exists F\ xF\ \&\ \neg Fx]$ is not part of the language, it is possible to formulate the following instead:

$$[\lambda y\ [\lambda z\ \forall p(p \to p)]\iota x(x = y\ \&\ (\exists F\ yF\ \&\ \neg Fy))] \tag{3}$$

If one considers 2 now, one can see that this λ-expressions behaves exactly the way that $[\lambda x\ \exists F\ xF\ \&\ \neg Fx]$ would, if it were part of the language, i.e. the result of β-reduction for $[\lambda x\ \exists F\ xF\ \&\ \neg Fx]$ would be the same as the right hand side of 2 when applied to 3. Therefore the λ-expression in 3 can be used to reproduce the paradox described in section 3.2.

5.2.3. Possible Solutions

Fortunately no theorems were derived in PLM, that actually depend on problematic λ-expressions as described above. Therefore it is possible to recover from the paradox without losing any theorems. At the time of writing it seems likely that a concept of *proper* λ-expressions will be introduced to the theory and only *proper* λ-expressions will be forced to have denotations and allow β-conversion. Problematic λ-expressions that would lead to paradoxes, would not be considered *proper*. Several options are available to define the propriety of *λ-expressions* and adjust PLM in detail.

As a consequence the purely syntactical distinction between propositional and non-propositional formulas is no longer sufficient to guarantee that every relation term has a denotation. The embedding of the theory supports the idea that an adequate definition of *proper λ-expressions* could replace this distinction entirely yielding a much broader set of relations. The philosophical implications of such a radical modification of the theory have not yet been analysed entirely, though, and at the time of writing it is an open question whether such a modification may be implemented in future versions of PLM.

5.3. A Meta-Conjecture about Possible Worlds

A conversation between Bruno Woltzenlogel Paleo and Edward Zalta about the Theory of Abstract Objects led to the following meta-conjecture:

" For every syntactic possible world w, there exists a semantic point p which is the denotation of w. "[1]

Since the embedding constructs a representation of the semantics of PLM, it was possible to formally analyse the relationship between syntactic and semantic possible worlds and arrive at the following theorems (see A.10):

- $\forall\, x.\ [\mathit{PossibleWorld}\ (x^P)\ \mathit{in}\ w] \longrightarrow (\exists\, v.\ \forall\, p.\ [x^P \models p\ \mathit{in}\ w] = [p\ \mathit{in}\ v])$
- $\forall\, v.\ \exists\, x.\ [\mathit{PossibleWorld}\ (x^P)\ \mathit{in}\ w] \wedge (\forall\, p.\ [p\ \mathit{in}\ v] = [x^P \models p\ \mathit{in}\ w])$

The first statement shows that for every *syntactic* possible world x there is a *semantic* possible world v, such that a proposition is syntactically true in x, if and only if it is semantically true in v.

The second statement shows that for every *semantic* possible world v there is a *syntactic* possible world x, such that a proposition is semantically true in v, if and only if it is *syntactically* true in x.

This result extends the following theorems already derived syntactically in PLM (w is restricted to only range over syntactic possible worlds):

- $\Diamond p \equiv \exists\, w(w \models p)$ (433.1)
- $\Box p \equiv \forall\, w(w \models p)$ (433.2)

Whereas the syntactic statements of PLM already show the relation between the modal operators and syntactic possible worlds, the semantic statements derived in the embedding show that there is in fact a natural bijection between syntactic and semantic possible worlds.

This example shows that a semantical embedding allows a detailed analysis of the semantical properties of a theory and to arrive at interesting meta-logical results.

5.4. Functional Object Theory

The first and foremost goal of the presented work was to show that the second order fragment of the Theory of Abstract Object as described in PLM can be represented in functional higher-order logic using a shallow semantical embedding.

[1]This formulation originates in the resulting e-mail correspondence between Bruno Woltzenlogel Paleo and Christoph Benzmüller.

As a result a theory was constructed in Isabelle/HOL that - although its soundness is yet to be formally verified - is most likely able to represent and verify all reasoning in the target theory. A formal analysis of the soundness of the embedding is unfortunately not possible at this time, since the theory of PLM first has to be adjusted to prevent the discovered paradox. Depending on the precise modifications of PLM the embedding will have to be adjusted accordingly, after which the question of its formal soundness can be revisited.

The embedding goes to great lengths to construct a restricted environment, in which it is possible to derive new theorems that can easily be translated back to the reference system of PLM. The fact that the construction of the paradox described in section 5.2 could be reproduced in the target logic, strongly indicates the merits and success of this approach.

What the embedding in its current form is not able to achieve, however, is completeness with respect to the theory of PLM: using the meta-logic theorems can be derived in the embedding, that are not derivable using the deductive system of PLM.

For this reason future research may want to address the following questions:

- Can the representation layer of the embedding be further restricted in such a way, that only theorems of PLM are derivable?
- If not, can the embedding be restricted in such a way, that all derivable theorems that are not part of PLM can be identified syntactically (resp. are not syntactically well-formed in PLM)?
- Is there an extension of the formal system constructed in PLM, for which the embedding is in fact complete, i.e. every theorem derivable in the embedding is a theorem in the extended system?

Independently of the relation between the embedding and the target system, a byproduct of the embedding is a working functional variant of object theory that deserves to be studied in it own right. To that end future research may want to drop the layered structure of the embedding and dismiss all constructions that solely serve to restrict reasoning in the embedding in order to more closely reproduce the language of PLM. Automated reasoning in the resulting theory will be significantly more powerful and the interesting properties of the original theory, that result from the introduction of abstract objects and encoding, can still be preserved.

5.5. Relations vs. Functions

As mentioned in the introduction, Openheimer and Zalta argue in [8] that relational type theory is more fundamental than functional type theory. One of their main arguments is that the Theory of Abstract Objects is not representable in functional type theory. Although there remain open questions and there may still be reservations about the accuracy of the representation constructed in the presented embedding, its initial success suggests that the topic has to be examined more closely.

First of all it is important to note that the result in [8] is actually supported by the presented work in the following sense: it is impossible to represent the Theory of Abstract Objects by representing its λ-expressions as primitive λ-expressions in functional logic. Furthermore the embedding supports the argument that exemplification cannot be represented classically as function application, while at the same time introducing encoding as a second mode of predication.

This already establishes that the traditional approach to translate relational type theory to functional type theory in fact fails for the Theory of Abstract Object. Furthermore the embedding suggests that a simple version of functional type theory that only involves two primitive types (for individuals and propositions), as used as a representative functional type theory in [8], is in fact not sufficient for a representation of object theory.

It is interesting to note that the embedding does not share several of the properties of the representative functional type theory constructed in [8, pp. 9-12]:

- Relations are *not* represented as functions from individuals to truth values.
- Exemplification is *not* represented as simple function application.
- The λ-expressions of object theory are *not* represented as primitive λ-expressions.

To illustrate the general schema that the embedding uses instead, assume that there is an additional primitive type for each arity of relations R_n. Let further ι be the type of individuals and o be the type of propositions. The general construct is now the following:

- Exemplification (of an n-place relation) is a function of type $R_n \Rightarrow \iota \Rightarrow \ldots \Rightarrow \iota \Rightarrow \mathrm{o}$.
- Encoding is a function of type $\iota \Rightarrow R_1 \Rightarrow \mathrm{o}$.
- To represent λ-expressions functions Λ_n of type $(\iota \Rightarrow \ldots \Rightarrow \iota \Rightarrow \mathrm{o}) \Rightarrow R_n$ are introduced. The λ-expression $[\lambda x_1 \ldots x_n\ \varphi]$ of object theory is represented as $\Lambda_n[\lambda x_1 \ldots x_n\ \varphi]$.

Interestingly one issue still remains that is in fact reflected in the constructed embedding as well: Not all functions of type $\iota \Rightarrow \ldots \Rightarrow \iota \Rightarrow \mathrm{o}$ are supposed to denote relations. However, in the proposed construction a concept used in the embedding of free logics can help[4]. The function Λ_n can map functions of type $\iota \Rightarrow \ldots \Rightarrow \iota \Rightarrow \mathrm{o}$ that do not correspond to propositional formulas to objects of type R_n that represent invalid (resp. non-existing) relations. The functions used to represent encoding and exemplification can be defined to map to an object of type o that represents invalid propositions if the involved relation is invalid.

In [8, pp. 30-31] it is argued that using a free logic and letting non-propositional formulas fail to denote is not an option, since it prevents classical reasoning for non-propositional formulas. Although this is true for the case of a simple functional type theory, it does not apply to the theory constructed above: since only objects of type R_n may fail to denote, non-propositional reasoning is unaffected by the introduction of the free logic.

Remark. *Although the constructed functional type theory is based on the general structure of the presented embedding, instead of introducing concepts of free logic λ-expressions involving non-propositional formulas are assigned* non-standard *denotations, i.e. they do denote, but β-conversion does only hold under certain conditions (see 5.1.1). How-*

ever, it is likely that future versions of the embedding will be able to utilize the concepts described in [4] to replace this construction by a free logic implementation that will more closely reflect the concepts of propositional formulas and λ-expressions present in object theory.

In summery it can be concluded that a representation of object theory in functional type theory is feasible, although it is connected with significant complexity (i.e. the introduction of additional primitive types and the usage of concepts of free logic). On the other hand, whether this result contradicts the philosophical claim that relations are more fundamental than functions, is still debatable considering the fact that the proposed construction has to introduce new primitive types for relations[2] and the construction is complex in general. Further it has to be noted that so far only the second order fragment of object theory has been considered and the full type-theoretic version of the theory may present further challenges.

5.6. Conclusion

The presented work shows that shallow semantical embeddings in HOL have the potential to represent even highly complex theories that originate in a fundamentally different tradition of logical reasoning (e.g. relational instead of functional type theory). The presented embedding represents the most ambitious project in this area so far; therefore it comes at no surprise that it faces unique challenges. Although further research is necessary to arrive at a comprehensive analysis of its implications, the presented embedding can already clearly show the merits of the approach.

Not only could the embedding uncover a previously unknown paradox in the formulation of its target theory, but it could contribute to the understanding of the relation between functional and relational type theory and provide further insights into the general structure of the target theory, its semantics and possible models. It can even show that a consistent extension of the theory seems possible that could increase its expressibility.

The presented work introduces novel concepts that can benefit future endeavors of semantical embeddings in general: a layered structure allows the representation of a target theory without extensive prior results about its model structure and provides the means to comprehensively study potential models. Custom proving methods may benefit automated reasoning in an embedded logic and provide the means to reproduce even complex deductive rules of the target system in a user-friendly manner.

The fact that the embedding can construct a verified environment that allows it to conveniently prove and verify theorems in the target logic while retaining the support of automated reasoning tools, shows the great potential of semantical embeddings in providing the means for a productive interaction between humans and computer systems.

[2]Note, however, that the embedding can represent relations as functions acting on urelements following the Aczel-model.

A. Isabelle Theory

A.1. Embedding

A.1.1. Primitives

typedecl i — possible worlds
typedecl j — states

consts $dw :: i$ — actual world
consts $dj :: j$ — actual state

typedecl ω — ordinary objects
typedecl σ — special urelements
datatype $\upsilon = \omega\upsilon\ \omega \mid \sigma\upsilon\ \sigma$ — urelements

A.1.2. Derived Types

typedef $\mathrm{o} = \mathit{UNIV}::(j \Rightarrow i \Rightarrow \mathit{bool})\ \mathit{set}$
morphisms $\mathit{eval}\mathrm{o}\ \mathit{make}\mathrm{o}$ **..** — truth values

type-synonym $\Pi_0 = \mathrm{o}$ — zero place relations
typedef $\Pi_1 = \mathit{UNIV}::(\upsilon \Rightarrow j \Rightarrow i \Rightarrow \mathit{bool})\ \mathit{set}$
morphisms $\mathit{eval}\Pi_1\ \mathit{make}\Pi_1$ **..** — one place relations
typedef $\Pi_2 = \mathit{UNIV}::(\upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow \mathit{bool})\ \mathit{set}$
morphisms $\mathit{eval}\Pi_2\ \mathit{make}\Pi_2$ **..** — two place relations
typedef $\Pi_3 = \mathit{UNIV}::(\upsilon \Rightarrow \upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow \mathit{bool})\ \mathit{set}$
morphisms $\mathit{eval}\Pi_3\ \mathit{make}\Pi_3$ **..** — three place relations

type-synonym $\alpha = \Pi_1\ \mathit{set}$ — abstract objects

datatype $\nu = \omega\nu\ \omega \mid \alpha\nu\ \alpha$ — individuals

typedef $\kappa = \mathit{UNIV}::(\nu\ \mathit{option})\ \mathit{set}$
morphisms $\mathit{eval}\kappa\ \mathit{make}\kappa$ **..** — individual terms

setup-lifting $\mathit{type\text{-}definition}\text{-}\mathrm{o}$
setup-lifting $\mathit{type\text{-}definition}\text{-}\kappa$
setup-lifting $\mathit{type\text{-}definition}\text{-}\Pi_1$
setup-lifting $\mathit{type\text{-}definition}\text{-}\Pi_2$
setup-lifting $\mathit{type\text{-}definition}\text{-}\Pi_3$

A.1.3. Individual Terms and Definite Descriptions

Remark. *Individual terms can be definite descriptions which may not denote. Therefore the type for individual terms κ is defined as ν option. Individuals are represented by Some x for an individual x of type ν, whereas non-denoting individual terms are represented by None. Note that relation terms on the other hand always denote, so there is no need for a similar distinction between relation terms and relations.*

lift-definition $\nu\kappa$ :: $\nu \Rightarrow \kappa$ (-^P [90] 90) **is** *Some* **.**
lift-definition *proper* :: $\kappa \Rightarrow$*bool* **is** *op*$\neq$ *None* **.**
lift-definition *rep* :: $\kappa \Rightarrow \nu$ **is** *the* **.**

Remark. *Individual terms can be explicitly marked to only range over logically proper objects (e.g. x^P). Their logical propriety and (in case they are logically proper) the represented individual can be extracted from the internal representation as ν option.*

lift-definition *that*::$(\nu \Rightarrow o) \Rightarrow \kappa$ (**binder** ι [8] 9) **is**
$\lambda\ \varphi$. *if* $(\exists!\ x\ .\ (\varphi\ x)\ dj\ dw)$
then Some (*THE* x . $(\varphi\ x)$ *dj dw*)
else None **.**

Remark. *Definite descriptions map conditions on individuals to individual terms. If no unique object satisfying the condition exists (and therefore the definite description is not logically proper), the individual term is set to None.*

A.1.4. Mapping from objects to urelements

consts $\alpha\sigma$ :: $\alpha \Rightarrow \sigma$
axiomatization where $\alpha\sigma$*-surj*: *surj* $\alpha\sigma$
definition $\nu\upsilon$ :: $\nu \Rightarrow \upsilon$ **where** $\nu\upsilon \equiv$ *case-*ν $\omega\upsilon$ $(\sigma\upsilon \circ \alpha\sigma)$

A.1.5. Exemplification of n-place relations.

lift-definition *exe0*::$\Pi_0 \Rightarrow o$ ($(\!|\text{-}|\!)$) **is** *id* **.**
lift-definition *exe1*::$\Pi_1 \Rightarrow \kappa \Rightarrow o$ ($(\!|\text{-},\text{-}|\!)$) **is**
$\lambda\ F\ x\ s\ w\ .\ (proper\ x) \wedge F\ (\nu\upsilon\ (rep\ x))\ s\ w$ **.**
lift-definition *exe2*::$\Pi_2 \Rightarrow \kappa \Rightarrow \kappa \Rightarrow o$ ($(\!|\text{-},\text{-},\text{-}|\!)$) **is**
$\lambda\ F\ x\ y\ s\ w\ .\ (proper\ x) \wedge (proper\ y) \wedge$
$F\ (\nu\upsilon\ (rep\ x))\ (\nu\upsilon\ (rep\ y))\ s\ w$ **.**
lift-definition *exe3*::$\Pi_3 \Rightarrow \kappa \Rightarrow \kappa \Rightarrow \kappa \Rightarrow o$ ($(\!|\text{-},\text{-},\text{-},\text{-}|\!)$) **is**
$\lambda\ F\ x\ y\ z\ s\ w\ .\ (proper\ x) \wedge (proper\ y) \wedge (proper\ z) \wedge$
$F\ (\nu\upsilon\ (rep\ x))\ (\nu\upsilon\ (rep\ y))\ (\nu\upsilon\ (rep\ z))\ s\ w$ **.**

Remark. *An exemplification formula can only be true if all individual terms are logically proper. Furthermore exemplification depends on the urelement corresponding to the individual, not the individual itself.*

A.1.6. Encoding

lift-definition *enc* :: $\kappa \Rightarrow \Pi_1 \Rightarrow o$ ($\{\!|\text{-},\text{-}|\!\}$) **is**
$\lambda\ x\ F\ s\ w\ .\ (proper\ x) \wedge case\text{-}\nu\ (\lambda\ \omega\ .\ False)\ (\lambda\ \alpha\ .\ F \in \alpha)\ (rep\ x)$ **.**

Remark. *An encoding formula can again only be true if the individual term is logically proper. Furthermore ordinary objects never encode, whereas abstract objects encode a property if and only if the property is contained in it as per the Aczel Model.*

A.1.7. Connectives and Quantifiers

consts *I-NOT* :: $j \Rightarrow (i \Rightarrow bool) \Rightarrow i \Rightarrow bool$
consts *I-IMPL* :: $j \Rightarrow (i \Rightarrow bool) \Rightarrow (i \Rightarrow bool) \Rightarrow (i \Rightarrow bool)$

lift-definition *not* :: $o \Rightarrow o$ ($\neg$- [*54*] *70*) **is**
$\lambda\ p\ s\ w\ .\ s = dj \wedge \neg p\ dj\ w \vee s \neq dj \wedge (\textit{I-NOT}\ s\ (p\ s)\ w)$ **.**
lift-definition *impl* :: $o \Rightarrow o \Rightarrow o$ (**infixl** $\rightarrow$ *51*) **is**
$\lambda\ p\ q\ s\ w\ .\ s = dj \wedge (p\ dj\ w \longrightarrow q\ dj\ w) \vee s \neq dj \wedge (\textit{I-IMPL}\ s\ (p\ s)\ (q\ s)\ w)$ **.**
lift-definition $forall_\nu$ :: $(\nu \Rightarrow o) \Rightarrow o$ (**binder** $\forall_\nu$ [*8*] *9*) **is**
$\lambda\ \varphi\ s\ w\ .\ \forall\ x :: \nu\ .\ (\varphi\ x)\ s\ w$ **.**
lift-definition $forall_0$ :: $(\Pi_0 \Rightarrow o) \Rightarrow o$ (**binder** $\forall_0$ [*8*] *9*) **is**
$\lambda\ \varphi\ s\ w\ .\ \forall\ x :: \Pi_0\ .\ (\varphi\ x)\ s\ w$ **.**
lift-definition $forall_1$ :: $(\Pi_1 \Rightarrow o) \Rightarrow o$ (**binder** $\forall_1$ [*8*] *9*) **is**
$\lambda\ \varphi\ s\ w\ .\ \forall\ x :: \Pi_1\ .\ (\varphi\ x)\ s\ w$ **.**
lift-definition $forall_2$ :: $(\Pi_2 \Rightarrow o) \Rightarrow o$ (**binder** $\forall_2$ [*8*] *9*) **is**
$\lambda\ \varphi\ s\ w\ .\ \forall\ x :: \Pi_2\ .\ (\varphi\ x)\ s\ w$ **.**
lift-definition $forall_3$ :: $(\Pi_3 \Rightarrow o) \Rightarrow o$ (**binder** $\forall_3$ [*8*] *9*) **is**
$\lambda\ \varphi\ s\ w\ .\ \forall\ x :: \Pi_3\ .\ (\varphi\ x)\ s\ w$ **.**
lift-definition $forall_o$ :: $(o \Rightarrow o) \Rightarrow o$ (**binder** $\forall_o$ [*8*] *9*) **is**
$\lambda\ \varphi\ s\ w\ .\ \forall\ x :: o\ .\ (\varphi\ x)\ s\ w$ **.**
lift-definition *box* :: $o \Rightarrow o$ ($\Box$- [*62*] *63*) **is**
$\lambda\ p\ s\ w\ .\ \forall\ v\ .\ p\ s\ v$ **.**
lift-definition *actual* :: $o \Rightarrow o$ ($\boldsymbol{\mathcal{A}}$- [*64*] *65*) **is**
$\lambda\ p\ s\ w\ .\ p\ s\ dw$ **.**

Remark. *The connectives behave classically if evaluated for the actual state dj, whereas their behavior is governed by uninterpreted constants for any other state.*

A.1.8. Lambda Expressions

Remark. *Lambda expressions have to convert maps from individuals to propositions to relations that are represented by maps from urelements to truth values.*

lift-definition *lambdabinder0* :: $o \Rightarrow \Pi_0$ ($\boldsymbol{\lambda}^0$) **is** *id* **.**
lift-definition *lambdabinder1* :: $(\nu \Rightarrow o) \Rightarrow \Pi_1$ (**binder** $\boldsymbol{\lambda}$ [*8*] *9*) **is**
$\lambda\ \varphi\ u\ s\ w\ .\ \exists\ x\ .\ \nu\upsilon\ x = u \wedge \varphi\ x\ s\ w$ **.**
lift-definition *lambdabinder2* :: $(\nu \Rightarrow \nu \Rightarrow o) \Rightarrow \Pi_2$ ($\boldsymbol{\lambda}^2$) **is**
$\lambda\ \varphi\ u\ v\ s\ w\ .\ \exists\ x\ y\ .\ \nu\upsilon\ x = u \wedge \nu\upsilon\ y = v \wedge \varphi\ x\ y\ s\ w$ **.**
lift-definition *lambdabinder3* :: $(\nu \Rightarrow \nu \Rightarrow \nu \Rightarrow o) \Rightarrow \Pi_3$ ($\boldsymbol{\lambda}^3$) **is**
$\lambda\ \varphi\ u\ v\ r\ s\ w\ .\ \exists\ x\ y\ z\ .\ \nu\upsilon\ x = u \wedge \nu\upsilon\ y = v \wedge \nu\upsilon\ z = r \wedge \varphi\ x\ y\ z\ s\ w$ **.**

A.1.9. Proper Maps from Individual Terms to Propositions

Remark. *The embedding introduces the notion of* proper *maps from individual terms to propositions.*
Such a map is proper if and only for all proper individual terms its truth evaluation in the actual state only depends on the urelement corresponding to the individual the term denotes.
Proper maps are exactly those maps that - when used in a lambda-expression - unconditionally allow beta-reduction.

lift-definition *IsProperInX* :: $(\kappa \Rightarrow o) \Rightarrow bool$ **is**
$\lambda\ \varphi\ .\ \forall\ x\ v\ .\ (\exists\ a\ .\ \nu\upsilon\ a = \nu\upsilon\ x \wedge (\varphi\ (a^P)\ dj\ v)) = (\varphi\ (x^P)\ dj\ v)$ **.**
lift-definition *IsProperInXY* :: $(\kappa \Rightarrow \kappa \Rightarrow o) \Rightarrow bool$ **is**

$\lambda\ \varphi\ .\ \forall\ \ x\ y\ v\ .\ (\exists\ \ a\ b\ .\ \nu\upsilon\ a = \nu\upsilon\ x \wedge \nu\upsilon\ b = \nu\upsilon\ y$
$\wedge\ (\varphi\ (a^P)\ (b^P)\ dj\ v)) = (\varphi\ (x^P)\ (y^P)\ dj\ v)$ **.**
lift-definition *IsProperInXYZ* :: $(\kappa{\Rightarrow}\kappa{\Rightarrow}\kappa{\Rightarrow}\mathrm{o}){\Rightarrow}bool$ **is**
$\lambda\ \varphi\ .\ \forall\ \ x\ y\ z\ v\ .\ (\exists\ \ a\ b\ c\ .\ \nu\upsilon\ a = \nu\upsilon\ x \wedge \nu\upsilon\ b = \nu\upsilon\ y \wedge \nu\upsilon\ c = \nu\upsilon\ z$
$\wedge\ (\varphi\ (a^P)\ (b^P)\ (c^P)\ dj\ v)) = (\varphi\ (x^P)\ (y^P)\ (z^P)\ dj\ v)$ **.**

A.1.10. Validity

lift-definition *valid-in* :: $i{\Rightarrow}\mathrm{o}{\Rightarrow}bool$ (**infixl** $\models$ *5*) **is**
$\lambda\ v\ \varphi\ .\ \varphi\ dj\ v$ **.**

Remark. *A formula is considered semantically valid for a possible world, if it evaluates to* True *for the actual state* dj *and the given possible world.*

A.1.11. Concreteness

consts *ConcreteInWorld* :: $\omega{\Rightarrow}i{\Rightarrow}bool$

abbreviation (*input*) *OrdinaryObjectsPossiblyConcrete* **where**
OrdinaryObjectsPossiblyConcrete $\equiv \forall\ \ x\ .\ \exists\ \ v\ .$ *ConcreteInWorld* $x\ v$
abbreviation (*input*) *PossiblyContingentObjectExists* **where**
PossiblyContingentObjectExists $\equiv \exists\ \ x\ v\ .$ *ConcreteInWorld* $x\ v$
$\wedge\ (\exists\ \ w\ .\ \neg$ *ConcreteInWorld* $x\ w)$
abbreviation (*input*) *PossiblyNoContingentObjectExists* **where**
PossiblyNoContingentObjectExists $\equiv \exists\ \ w\ .\ \forall\ \ x\ .$ *ConcreteInWorld* $x\ w$
$\longrightarrow (\forall\ \ v\ .$ *ConcreteInWorld* $x\ v)$
axiomatization where
OrdinaryObjectsPossiblyConcreteAxiom:
OrdinaryObjectsPossiblyConcrete
and *PossiblyContingentObjectExistsAxiom*:
PossiblyContingentObjectExists
and *PossiblyNoContingentObjectExistsAxiom*:
PossiblyNoContingentObjectExists

Remark. *In order to define concreteness, care has to be taken that the defined notion of concreteness coincides with the meta-logical distinction between abstract objects and ordinary objects. Furthermore the axioms about concreteness have to be satisfied. This is achieved by introducing an uninterpreted constant* ConcreteInWorld *that determines whether an ordinary object is concrete in a given possible world. This constant is axiomatized, such that all ordinary objects are possibly concrete, contingent objects possibly exist and possibly no contingent objects exist.*

lift-definition *Concrete*::Π_1 $(E!)$ **is**
$\lambda\ u\ s\ w\ .$ *case* u *of* $\omega\upsilon\ x \Rightarrow$ *ConcreteInWorld* $x\ w \mid \text{-} \Rightarrow$ *False* **.**

Remark. *Concreteness of ordinary objects is now defined using this axiomatized uninterpreted constant. Abstract objects on the other hand are never concrete.*

A.1.12. Collection of Meta-Definitions

The meta-logical definitions are collected with the theorem attribute *meta-defs*.

named-theorems *meta-defs*

declare *not-def*[*meta-defs*] *impl-def*[*meta-defs*] *forall*$_\nu$*-def*[*meta-defs*]
forall$_0$*-def*[*meta-defs*] *forall*$_1$*-def*[*meta-defs*]
forall$_2$*-def*[*meta-defs*] *forall*$_3$*-def*[*meta-defs*] *forall*$_o$*-def*[*meta-defs*]
box-def[*meta-defs*] *actual-def*[*meta-defs*] *that-def*[*meta-defs*]
lambdabinder0-def[*meta-defs*] *lambdabinder1-def*[*meta-defs*]
lambdabinder2-def[*meta-defs*] *lambdabinder3-def*[*meta-defs*]
exe0-def[*meta-defs*] *exe1-def*[*meta-defs*] *exe2-def*[*meta-defs*]
exe3-def[*meta-defs*] *enc-def*[*meta-defs*] *inv-def*[*meta-defs*]
that-def[*meta-defs*] *valid-in-def*[*meta-defs*] *Concrete-def*[*meta-defs*]

declare [[*smt-solver = cvc4*]]
declare [[*simp-depth-limit = 10*]]
declare [[*unify-search-bound = 40*]]

A.1.13. Auxiliary Lemmata

Some auxiliary lemmata are proven to make reasoning in the meta-logic easier. These auxiliary lemmata are collected using the theorem attribute *meta-aux*.

named-theorems *meta-aux*

declare *make*κ*-inverse*[*meta-aux*] *eval*κ*-inverse*[*meta-aux*]
make$o$*-inverse*[*meta-aux*] *eval*o*-inverse*[*meta-aux*]
makeΠ_1*-inverse*[*meta-aux*] *eval*Π_1*-inverse*[*meta-aux*]
makeΠ_2*-inverse*[*meta-aux*] *eval*Π_2*-inverse*[*meta-aux*]
makeΠ_3*-inverse*[*meta-aux*] *eval*Π_3*-inverse*[*meta-aux*]
lemma $\nu\upsilon$*-*$\omega\nu$*-is-*$\omega\upsilon$[*meta-aux*]: $\nu\upsilon\ (\omega\nu\ x) = \omega\upsilon\ x$ **by** (*simp add*: $\nu\upsilon$*-def*)
lemma *rep-proper-id*[*meta-aux*]: $rep\ (x^P) = x$
by (*simp add*: *meta-aux* $\nu\kappa$*-def rep-def*)
lemma $\nu\kappa$*-proper*[*meta-aux*]: $proper\ (x^P)$
by (*simp add*: *meta-aux* $\nu\kappa$*-def proper-def*)
lemma *no-*$\alpha\omega$[*meta-aux*]: $\neg(\nu\upsilon\ (\alpha\nu\ x) = \omega\upsilon\ y)$ **by** (*simp add*: $\nu\upsilon$*-def*)
lemma *no-*$\sigma\omega$[*meta-aux*]: $\neg(\sigma\upsilon\ x = \omega\upsilon\ y)$ **by** *blast*
lemma $\nu\upsilon$*-surj*[*meta-aux*]: *surj* $\nu\upsilon$
using $\alpha\sigma$*-surj* **unfolding** $\nu\upsilon$*-def surj-def*
by (*metis* ν*.simps*(*5*) ν*.simps*(*6*) υ*.exhaust comp-apply*)
lemma *lambda*Π_1*-aux*[*meta-aux*]:
$make\Pi_1\ (\lambda u\ s\ w.\ \exists x.\ \nu\upsilon\ x = u \wedge eval\Pi_1\ F\ (\nu\upsilon\ x)\ s\ w) = F$
proof –
have $\bigwedge\ u\ s\ w\ \varphi\ .\ (\exists\ x\ .\ \nu\upsilon\ x = u \wedge \varphi\ (\nu\upsilon\ x)\ (s{::}j)\ (w{::}i)) \longleftrightarrow \varphi\ u\ s\ w$
using $\nu\upsilon$*-surj* **unfolding** *surj-def* **by** *metis*
thus *?thesis* **apply** *transfer* **by** *simp*
qed
lemma *lambda*Π_2*-aux*[*meta-aux*]:
$make\Pi_2\ (\lambda u\ v\ s\ w.\ \exists x\ .\ \nu\upsilon\ x = u \wedge (\exists\ y\ .\ \nu\upsilon\ y = v \wedge eval\Pi_2\ F\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ s\ w)) = F$
proof –
have $\bigwedge\ u\ v\ (s\ {::}j)\ (w{::}i)\ \varphi\ .$
$(\exists\ x\ .\ \nu\upsilon\ x = u \wedge (\exists\ y\ .\ \nu\upsilon\ y = v \wedge \varphi\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ s\ w))$
$\longleftrightarrow \varphi\ u\ v\ s\ w$
using $\nu\upsilon$*-surj* **unfolding** *surj-def* **by** *metis*
thus *?thesis* **apply** *transfer* **by** *simp*
qed
lemma *lambda*Π_3*-aux*[*meta-aux*]:
$make\Pi_3\ (\lambda u\ v\ r\ s\ w.\ \exists x.\ \nu\upsilon\ x = u \wedge (\exists y.\ \nu\upsilon\ y = v \wedge$
$(\exists z.\ \nu\upsilon\ z = r \wedge eval\Pi_3\ F\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ (\nu\upsilon\ z)\ s\ w))) = F$
proof –
have $\bigwedge\ u\ v\ r\ (s{::}j)\ (w{::}i)\ \varphi\ .\ \exists x.\ \nu\upsilon\ x = u \wedge (\exists y.\ \nu\upsilon\ y = v$

$\wedge$ ($\exists z.\ \nu\upsilon\ z = r \wedge \varphi\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ (\nu\upsilon\ z)\ s\ w)) = \varphi\ u\ v\ r\ s\ w$
using *$\nu\upsilon$-surj* **unfolding** *surj-def* **by** *metis*
thus *?thesis* **apply** *transfer* **apply** (*rule ext*)+ **by** *metis*
qed

A.2. Semantics

A.2.1. Definition

locale *Semantics*
begin
named-theorems *semantics*

A.2.1.1. Semantical Domains

type-synonym $R_\kappa = \nu$
type-synonym $R_0 = j \Rightarrow i \Rightarrow bool$
type-synonym $R_1 = \upsilon \Rightarrow R_0$
type-synonym $R_2 = \upsilon \Rightarrow \upsilon \Rightarrow R_0$
type-synonym $R_3 = \upsilon \Rightarrow \upsilon \Rightarrow \upsilon \Rightarrow R_0$
type-synonym $W = i$

A.2.1.2. Denotation Functions

lift-definition $d_\kappa :: \kappa \Rightarrow R_\kappa$ *option* **is** *id* **.**
lift-definition $d_0 :: \Pi_0 \Rightarrow R_0$ *option* **is** *Some* **.**
lift-definition $d_1 :: \Pi_1 \Rightarrow R_1$ *option* **is** *Some* **.**
lift-definition $d_2 :: \Pi_2 \Rightarrow R_2$ *option* **is** *Some* **.**
lift-definition $d_3 :: \Pi_3 \Rightarrow R_3$ *option* **is** *Some* **.**

A.2.1.3. Actual World

definition w_0 **where** $w_0 \equiv dw$

A.2.1.4. Exemplification Extensions

definition *ex0* :: $R_0 \Rightarrow W \Rightarrow bool$
where *ex0* $\equiv \lambda\ F\ .\ F\ dj$
definition *ex1* :: $R_1 \Rightarrow W \Rightarrow (R_\kappa\ set)$
where *ex1* $\equiv \lambda\ F\ w\ .\ \{\ x\ .\ F\ (\nu\upsilon\ x)\ dj\ w\ \}$
definition *ex2* :: $R_2 \Rightarrow W \Rightarrow ((R_\kappa \times R_\kappa)\ set)$
where *ex2* $\equiv \lambda\ F\ w\ .\ \{\ (x,y)\ .\ F\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ dj\ w\ \}$
definition *ex3* :: $R_3 \Rightarrow W \Rightarrow ((R_\kappa \times R_\kappa \times R_\kappa)\ set)$
where *ex3* $\equiv \lambda\ F\ w\ .\ \{\ (x,y,z)\ .\ F\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ (\nu\upsilon\ z)\ dj\ w\ \}$

A.2.1.5. Encoding Extensions

definition *en* :: $R_1 \Rightarrow (R_\kappa\ set)$
where *en* $\equiv \lambda\ F\ .\ \{\ x\ .\ case\ x\ of\ \alpha\nu\ y \Rightarrow make\Pi_1\ (\lambda\ x\ .\ F\ x) \in y$
$|\ \text{-} \Rightarrow False\ \}$

A.2.1.6. Collection of Semantical Definitions

named-theorems *semantics-defs*
declare d_0*-def*[*semantics-defs*] d_1*-def*[*semantics-defs*]
d_2*-def*[*semantics-defs*] d_3*-def*[*semantics-defs*]
ex0-def[*semantics-defs*] *ex1-def*[*semantics-defs*]
ex2-def[*semantics-defs*] *ex3-def*[*semantics-defs*]
en-def[*semantics-defs*] d_κ*-def*[*semantics-defs*]
w_0*-def*[*semantics-defs*]

A.2.1.7. Truth Conditions of Exemplification Formulas

lemma *T1-1*[*semantics*]:
$(w \models (|F,x|)) = (\exists\ r\ o_1\ .\ Some\ r = d_1\ F \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in ex1\ r\ w)$
unfolding *semantics-defs*
apply (*simp add*: *meta-defs meta-aux rep-def proper-def*)
by (*metis option.discI option.exhaust option.sel*)

lemma *T1-2*[*semantics*]:
$(w \models (|F,x,y|)) = (\exists\ r\ o_1\ o_2\ .\ Some\ r = d_2\ F \wedge Some\ o_1 = d_\kappa\ x$
$\wedge Some\ o_2 = d_\kappa\ y \wedge (o_1,\ o_2) \in ex2\ r\ w)$
unfolding *semantics-defs*
apply (*simp add*: *meta-defs meta-aux rep-def proper-def*)
by (*metis option.discI option.exhaust option.sel*)

lemma *T1-3*[*semantics*]:
$(w \models (|F,x,y,z|)) = (\exists\ r\ o_1\ o_2\ o_3\ .\ Some\ r = d_3\ F \wedge Some\ o_1 = d_\kappa\ x$
$\wedge Some\ o_2 = d_\kappa\ y \wedge Some\ o_3 = d_\kappa\ z$
$\wedge (o_1,\ o_2,\ o_3) \in ex3\ r\ w)$
unfolding *semantics-defs*
apply (*simp add*: *meta-defs meta-aux rep-def proper-def*)
by (*metis option.discI option.exhaust option.sel*)

lemma *T3*[*semantics*]:
$(w \models (|F|)) = (\exists\ r\ .\ Some\ r = d_0\ F \wedge ex0\ r\ w)$
unfolding *semantics-defs*
by (*simp add*: *meta-defs meta-aux*)

A.2.1.8. Truth Conditions of Encoding Formulas

lemma *T2*[*semantics*]:
$(w \models \{|x,F|\}) = (\exists\ r\ o_1\ .\ Some\ r = d_1\ F \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in en\ r)$
unfolding *semantics-defs*
apply (*simp add*: *meta-defs meta-aux rep-def proper-def split*: ν*.split*)
by (*metis* ν*.exhaust* ν*.inject*(*2*) ν*.simps*(*4*) $\nu\kappa$*.rep-eq option.collapse*
option.discI rep.rep-eq rep-proper-id)

A.2.1.9. Truth Conditions of Complex Formulas

lemma *T4*[*semantics*]: $(w \models \neg\psi) = (\neg(w \models \psi))$
by (*simp add*: *meta-defs meta-aux*)

lemma *T5*[*semantics*]: $(w \models \psi \rightarrow \chi) = (\neg(w \models \psi) \vee (w \models \chi))$
by (*simp add*: *meta-defs meta-aux*)

lemma *T6*[*semantics*]: $(w \models \Box\psi) = (\forall\ v\ .\ (v \models \psi))$
by (*simp add*: *meta-defs meta-aux*)

lemma *T7*[*semantics*]: $(w \models \boldsymbol{\mathcal{A}}\psi) = (dw \models \psi)$
by (*simp add*: *meta-defs meta-aux*)

lemma *T8-ν*[*semantics*]: $(w \models \forall_\nu\ x.\ \psi\ x) = (\forall\ x\ .\ (w \models \psi\ x))$
by (*simp add*: *meta-defs meta-aux*)

lemma *T8-0*[*semantics*]: $(w \models \forall_0\ x.\ \psi\ x) = (\forall\ x\ .\ (w \models \psi\ x))$
by (*simp add*: *meta-defs meta-aux*)

lemma *T8-1*[*semantics*]: $(w \models \forall_1\ x.\ \psi\ x) = (\forall\ x\ .\ (w \models \psi\ x))$
by (*simp add*: *meta-defs meta-aux*)

lemma *T8-2*[*semantics*]: $(w \models \forall_2\ x.\ \psi\ x) = (\forall\ x\ .\ (w \models \psi\ x))$
by (*simp add*: *meta-defs meta-aux*)

lemma *T8-3*[*semantics*]: $(w \models \forall_3\ x.\ \psi\ x) = (\forall\ x\ .\ (w \models \psi\ x))$
by (*simp add*: *meta-defs meta-aux*)

lemma *T8-*o[*semantics*]: $(w \models \forall_{\mathrm{o}}\ x.\ \psi\ x) = (\forall\ x\ .\ (w \models \psi\ x))$
by (*simp add*: *meta-defs meta-aux*)

A.2.1.10. Denotations of Descriptions

lemma *D3*[*semantics*]:
$d_\kappa\ (\boldsymbol{\iota}x\ .\ \psi\ x) = (\mathit{if}\ (\exists x\ .\ (w_0 \models \psi\ x) \wedge (\forall\ y\ .\ (w_0\ \models \psi\ y) \longrightarrow y = x))$
$\mathit{then}\ (\mathit{Some}\ (\mathit{THE}\ x\ .\ (w_0 \models \psi\ x)))\ \mathit{else}\ \mathit{None})$
unfolding *semantics-defs*
by (*auto simp*: *meta-defs meta-aux*)

A.2.1.11. Denotations of Lambda Expressions

lemma *D4-1*[*semantics*]: $d_1\ (\boldsymbol{\lambda}\ x\ .\ (\!|F,\ x^P|\!)) = d_1\ F$
by (*simp add*: *meta-defs meta-aux*)

lemma *D4-2*[*semantics*]: $d_2\ (\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ (\!|F,\ x^P,\ y^P|\!))) = d_2\ F$
by (*simp add*: *meta-defs meta-aux*)

lemma *D4-3*[*semantics*]: $d_3\ (\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ (\!|F,\ x^P,\ y^P,\ z^P|\!))) = d_3\ F$
by (*simp add*: *meta-defs meta-aux*)

lemma *D5-1*[*semantics*]:
assumes *IsProperInX* φ
shows $\bigwedge\ w\ o_1\ r\ .\ \mathit{Some}\ r = d_1\ (\boldsymbol{\lambda}\ x\ .\ (\varphi\ (x^P))) \wedge \mathit{Some}\ o_1 = d_\kappa\ x$
$\longrightarrow (o_1 \in \mathit{ex1}\ r\ w) = (w \models \varphi\ x)$
using *assms* **unfolding** *IsProperInX-def semantics-defs*
by (*auto simp*: *meta-defs meta-aux rep-def proper-def* $\nu\kappa$*.abs-eq*)

lemma *D5-2*[*semantics*]:
assumes *IsProperInXY* φ
shows $\bigwedge\ w\ o_1\ o_2\ r\ .\ \mathit{Some}\ r = d_2\ (\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ \varphi\ (x^P)\ (y^P)))$
$\wedge\ \mathit{Some}\ o_1 = d_\kappa\ x \wedge \mathit{Some}\ o_2 = d_\kappa\ y$
$\longrightarrow ((o_1,o_2) \in \mathit{ex2}\ r\ w) = (w \models \varphi\ x\ y)$
using *assms* **unfolding** *IsProperInXY-def semantics-defs*
by (*auto simp*: *meta-defs meta-aux rep-def proper-def* $\nu\kappa$*.abs-eq*)

lemma *D5-3*[*semantics*]:
assumes *IsProperInXYZ* φ

shows $\bigwedge$ *w* o_1 o_2 o_3 *r* . *Some* $r = d_3$ ($\boldsymbol{\lambda}^3$ (λ *x y z* . φ (x^P) (y^P) (z^P)))
$\wedge$ *Some* $o_1 = d_\kappa$ *x* $\wedge$ *Some* $o_2 = d_\kappa$ *y* $\wedge$ *Some* $o_3 = d_\kappa$ *z*
$\longrightarrow$ ((o_1,o_2,o_3) $\in$ *ex3 r w*) = (*w* $\models$ φ *x y z*)
using *assms* **unfolding** *IsProperInXYZ-def semantics-defs*
by (*auto simp*: *meta-defs meta-aux rep-def proper-def* $\nu\kappa$*.abs-eq*)

lemma *D6*[*semantics*]: ($\bigwedge$ *w r* . *Some* $r = d_0$ ($\boldsymbol{\lambda}^0$ φ) $\longrightarrow$ *ex0 r w* = (*w* $\models$ φ))
by (*auto simp*: *meta-defs meta-aux semantics-defs*)

A.2.1.12. Auxiliary Lemmata

lemma *propex*$_0$: $\exists$ *r* . *Some* $r = d_0$ *F*
unfolding d_0*-def* **by** *simp*
lemma *propex*$_1$: $\exists$ *r* . *Some* $r = d_1$ *F*
unfolding d_1*-def* **by** *simp*
lemma *propex*$_2$: $\exists$ *r* . *Some* $r = d_2$ *F*
unfolding d_2*-def* **by** *simp*
lemma *propex*$_3$: $\exists$ *r* . *Some* $r = d_3$ *F*
unfolding d_3*-def* **by** *simp*
lemma d_κ*-proper*: d_κ (u^P) = *Some u*
unfolding d_κ*-def* **by** (*simp add*: $\nu\kappa$*-def meta-aux*)
lemma *ConcretenessSemantics1*:
Some $r = d_1$ *E*! $\Longrightarrow$ ($\exists$ *w* . $\omega\nu$ *x* $\in$ *ex1 r w*)
unfolding *semantics-defs* **apply** *transfer*
by (*simp add*: *OrdinaryObjectsPossiblyConcreteAxiom* $\nu\upsilon$-$\omega\nu$-*is*-$\omega\upsilon$)
lemma *ConcretenessSemantics2*:
Some $r = d_1$ *E*! $\Longrightarrow$ (*x* $\in$ *ex1 r w* $\longrightarrow$ ($\exists$ *y*. *x* = $\omega\nu$ *y*))
unfolding *semantics-defs* **apply** *transfer* **apply** *simp*
by (*metis* ν*.exhaust* υ*.exhaust* υ*.simps*(*6*) *no*-$\alpha\omega$)
lemma d_0*-inject*: $\bigwedge$*x y*. d_0 *x* = d_0 *y* $\Longrightarrow$ *x* = *y*
unfolding d_0*-def* **by** (*simp add*: *evalo-inject*)
lemma d_1*-inject*: $\bigwedge$*x y*. d_1 *x* = d_1 *y* $\Longrightarrow$ *x* = *y*
unfolding d_1*-def* **by** (*simp add*: *eval*Π_1*-inject*)
lemma d_2*-inject*: $\bigwedge$*x y*. d_2 *x* = d_2 *y* $\Longrightarrow$ *x* = *y*
unfolding d_2*-def* **by** (*simp add*: *eval*Π_2*-inject*)
lemma d_3*-inject*: $\bigwedge$*x y*. d_3 *x* = d_3 *y* $\Longrightarrow$ *x* = *y*
unfolding d_3*-def* **by** (*simp add*: *eval*Π_3*-inject*)
lemma d_κ*-inject*: $\bigwedge$*x y* o_1. *Some* $o_1 = d_\kappa$ *x* $\wedge$ *Some* $o_1 = d_\kappa$ *y* $\Longrightarrow$ *x* = *y*
proof −
fix *x* :: κ **and** *y* :: κ **and** o_1 :: ν
assume *Some* $o_1 = d_\kappa$ *x* $\wedge$ *Some* $o_1 = d_\kappa$ *y*
thus *x* = *y* **apply** *transfer* **by** *auto*
qed
end

A.2.2. Introduction Rules for Proper Maps

Remark. *Introduction rules for proper maps are derived. In particular every map whose argument only occurs in exemplification expressions is proper.*

named-theorems *IsProper-intros*

lemma *IsProperInX-intro*[*IsProper-intros*]:
IsProperInX (λ *x* . χ
(∗ *one place* ∗) (λ *F* . ⦇*F*,*x*⦈)
(∗ *two place* ∗) (λ *F* . ⦇*F*,*x*,*x*⦈) (λ *F a* . ⦇*F*,*x*,*a*⦈) (λ *F a* . ⦇*F*,*a*,*x*⦈)
(∗ *three place three x* ∗) (λ *F* . ⦇*F*,*x*,*x*,*x*⦈)

(∗ *three place two x* ∗) (λ F a . ⦇F,x,x,a⦈) (λ F a . ⦇F,x,a,x⦈)
(λ F a . ⦇F,a,x,x⦈)
(∗ *three place one x* ∗) (λ F a b. ⦇F,x,a,b⦈) (λ F a b. ⦇F,a,x,b⦈)
(λ F a b . ⦇F,a,b,x⦈))
unfolding *IsProperInX-def*
by (*auto simp*: *meta-defs meta-aux*)

lemma *IsProperInXY-intro*[*IsProper-intros*]:
IsProperInXY (λ x y . χ
(∗ *only x* ∗)
(∗ *one place* ∗) (λ F . ⦇F,x⦈)
(∗ *two place* ∗) (λ F . ⦇F,x,x⦈) (λ F a . ⦇F,x,a⦈) (λ F a . ⦇F,a,x⦈)
(∗ *three place three x* ∗) (λ F . ⦇F,x,x,x⦈)
(∗ *three place two x* ∗) (λ F a . ⦇F,x,x,a⦈) (λ F a . ⦇F,x,a,x⦈)
(λ F a . ⦇F,a,x,x⦈)
(∗ *three place one x* ∗) (λ F a b. ⦇F,x,a,b⦈) (λ F a b. ⦇F,a,x,b⦈)
(λ F a b . ⦇F,a,b,x⦈)
(∗ *only y* ∗)
(∗ *one place* ∗) (λ F . ⦇F,y⦈)
(∗ *two place* ∗) (λ F . ⦇F,y,y⦈) (λ F a . ⦇F,y,a⦈) (λ F a . ⦇F,a,y⦈)
(∗ *three place three y* ∗) (λ F . ⦇F,y,y,y⦈)
(∗ *three place two y* ∗) (λ F a . ⦇F,y,y,a⦈) (λ F a . ⦇F,y,a,y⦈)
(λ F a . ⦇F,a,y,y⦈)
(∗ *three place one y* ∗) (λ F a b. ⦇F,y,a,b⦈) (λ F a b. ⦇F,a,y,b⦈)
(λ F a b . ⦇F,a,b,y⦈)
(∗ *x and y* ∗)
(∗ *two place* ∗) (λ F . ⦇F,x,y⦈) (λ F . ⦇F,y,x⦈)
(∗ *three place* (x,y) ∗) (λ F a . ⦇F,x,y,a⦈) (λ F a . ⦇F,x,a,y⦈)
(λ F a . ⦇F,a,x,y⦈)
(∗ *three place* (y,x) ∗) (λ F a . ⦇F,y,x,a⦈) (λ F a . ⦇F,y,a,x⦈)
(λ F a . ⦇F,a,y,x⦈)
(∗ *three place* (x,x,y) ∗) (λ F . ⦇F,x,x,y⦈) (λ F . ⦇F,x,y,x⦈)
(λ F . ⦇F,y,x,x⦈)
(∗ *three place* (x,y,y) ∗) (λ F . ⦇F,x,y,y⦈) (λ F . ⦇F,y,x,y⦈)
(λ F . ⦇F,y,y,x⦈)
(∗ *three place* (x,x,x) ∗) (λ F . ⦇F,x,x,x⦈)
(∗ *three place* (y,y,y) ∗) (λ F . ⦇F,y,y,y⦈))
unfolding *IsProperInXY-def* **by** (*auto simp*: *meta-defs meta-aux*)

lemma *IsProperInXYZ-intro*[*IsProper-intros*]:
IsProperInXYZ (λ x y z . χ
(∗ *only x* ∗)
(∗ *one place* ∗) (λ F . ⦇F,x⦈)
(∗ *two place* ∗) (λ F . ⦇F,x,x⦈) (λ F a . ⦇F,x,a⦈) (λ F a . ⦇F,a,x⦈)
(∗ *three place three x* ∗) (λ F . ⦇F,x,x,x⦈)
(∗ *three place two x* ∗) (λ F a . ⦇F,x,x,a⦈) (λ F a . ⦇F,x,a,x⦈)
(λ F a . ⦇F,a,x,x⦈)
(∗ *three place one x* ∗) (λ F a b. ⦇F,x,a,b⦈) (λ F a b. ⦇F,a,x,b⦈)
(λ F a b . ⦇F,a,b,x⦈)
(∗ *only y* ∗)
(∗ *one place* ∗) (λ F . ⦇F,y⦈)
(∗ *two place* ∗) (λ F . ⦇F,y,y⦈) (λ F a . ⦇F,y,a⦈) (λ F a . ⦇F,a,y⦈)
(∗ *three place three y* ∗) (λ F . ⦇F,y,y,y⦈)
(∗ *three place two y* ∗) (λ F a . ⦇F,y,y,a⦈) (λ F a . ⦇F,y,a,y⦈)
(λ F a . ⦇F,a,y,y⦈)
(∗ *three place one y* ∗) (λ F a b. ⦇F,y,a,b⦈) (λ F a b. ⦇F,a,y,b⦈)
(λ F a b . ⦇F,a,b,y⦈)
(∗ *only z* ∗)

$(* \textit{ one place } *)\ (\lambda\ F\ .\ \llparenthesis F,z\rrparenthesis)$
$(* \textit{ two place } *)\ (\lambda\ F\ .\ \llparenthesis F,z,z\rrparenthesis)\ (\lambda\ F\ a\ .\ \llparenthesis F,z,a\rrparenthesis)\ (\lambda\ F\ a\ .\ \llparenthesis F,a,z\rrparenthesis)$
$(* \textit{ three place three } z *)\ (\lambda\ F\ .\ \llparenthesis F,z,z,z\rrparenthesis)$
$(* \textit{ three place two } z *)\ (\lambda\ F\ a\ .\ \llparenthesis F,z,z,a\rrparenthesis)\ (\lambda\ F\ a\ .\ \llparenthesis F,z,a,z\rrparenthesis)$
$(\lambda\ F\ a\ .\ \llparenthesis F,a,z,z\rrparenthesis)$
$(* \textit{ three place one } z *)\ (\lambda\ F\ a\ b.\ \llparenthesis F,z,a,b\rrparenthesis)\ (\lambda\ F\ a\ b.\ \llparenthesis F,a,z,b\rrparenthesis)$
$(\lambda\ F\ a\ b\ .\ \llparenthesis F,a,b,z\rrparenthesis)$
$(* \ x \textit{ and } y *)$
$(* \textit{ two place } *)\ (\lambda\ F\ .\ \llparenthesis F,x,y\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,y,x\rrparenthesis)$
$(* \textit{ three place } (x,y) *)\ (\lambda\ F\ a\ .\ \llparenthesis F,x,y,a\rrparenthesis)\ (\lambda\ F\ a\ .\ \llparenthesis F,x,a,y\rrparenthesis)$
$(\lambda\ F\ a\ .\ \llparenthesis F,a,x,y\rrparenthesis)$
$(* \textit{ three place } (y,x) *)\ (\lambda\ F\ a\ .\ \llparenthesis F,y,x,a\rrparenthesis)\ (\lambda\ F\ a\ .\ \llparenthesis F,y,a,x\rrparenthesis)$
$(\lambda\ F\ a\ .\ \llparenthesis F,a,y,x\rrparenthesis)$
$(* \textit{ three place } (x,x,y) *)\ (\lambda\ F\ .\ \llparenthesis F,x,x,y\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,x,y,x\rrparenthesis)$
$(\lambda\ F\ .\ \llparenthesis F,y,x,x\rrparenthesis)$
$(* \textit{ three place } (x,y,y) *)\ (\lambda\ F\ .\ \llparenthesis F,x,y,y\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,y,x,y\rrparenthesis)$
$(\lambda\ F\ .\ \llparenthesis F,y,y,x\rrparenthesis)$
$(* \textit{ three place } (x,x,x) *)\ (\lambda\ F\ .\ \llparenthesis F,x,x,x\rrparenthesis)$
$(* \textit{ three place } (y,y,y) *)\ (\lambda\ F\ .\ \llparenthesis F,y,y,y\rrparenthesis)$
$(* \ x \textit{ and } z *)$
$(* \textit{ two place } *)\ (\lambda\ F\ .\ \llparenthesis F,x,z\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,z,x\rrparenthesis)$
$(* \textit{ three place } (x,z) *)\ (\lambda\ F\ a\ .\ \llparenthesis F,x,z,a\rrparenthesis)\ (\lambda\ F\ a\ .\ \llparenthesis F,x,a,z\rrparenthesis)$
$(\lambda\ F\ a\ .\ \llparenthesis F,a,x,z\rrparenthesis)$
$(* \textit{ three place } (z,x) *)\ (\lambda\ F\ a\ .\ \llparenthesis F,z,x,a\rrparenthesis)\ (\lambda\ F\ a\ .\ \llparenthesis F,z,a,x\rrparenthesis)$
$(\lambda\ F\ a\ .\ \llparenthesis F,a,z,x\rrparenthesis)$
$(* \textit{ three place } (x,x,z) *)\ (\lambda\ F\ .\ \llparenthesis F,x,x,z\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,x,z,x\rrparenthesis)$
$(\lambda\ F\ .\ \llparenthesis F,z,x,x\rrparenthesis)$
$(* \textit{ three place } (x,z,z) *)\ (\lambda\ F\ .\ \llparenthesis F,x,z,z\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,z,x,z\rrparenthesis)$
$(\lambda\ F\ .\ \llparenthesis F,z,z,x\rrparenthesis)$
$(* \textit{ three place } (x,x,x) *)\ (\lambda\ F\ .\ \llparenthesis F,x,x,x\rrparenthesis)$
$(* \textit{ three place } (z,z,z) *)\ (\lambda\ F\ .\ \llparenthesis F,z,z,z\rrparenthesis)$
$(* \ y \textit{ and } z *)$
$(* \textit{ two place } *)\ (\lambda\ F\ .\ \llparenthesis F,y,z\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,z,y\rrparenthesis)$
$(* \textit{ three place } (y,z) *)\ (\lambda\ F\ a\ .\ \llparenthesis F,y,z,a\rrparenthesis)\ (\lambda\ F\ a\ .\ \llparenthesis F,y,a,z\rrparenthesis)$
$(\lambda\ F\ a\ .\ \llparenthesis F,a,y,z\rrparenthesis)$
$(* \textit{ three place } (z,y) *)\ (\lambda\ F\ a\ .\ \llparenthesis F,z,y,a\rrparenthesis)\ (\lambda\ F\ a\ .\ \llparenthesis F,z,a,y\rrparenthesis)$
$(\lambda\ F\ a\ .\ \llparenthesis F,a,z,y\rrparenthesis)$
$(* \textit{ three place } (y,y,z) *)\ (\lambda\ F\ .\ \llparenthesis F,y,y,z\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,y,z,y\rrparenthesis)$
$(\lambda\ F\ .\ \llparenthesis F,z,y,y\rrparenthesis)$
$(* \textit{ three place } (y,z,z) *)\ (\lambda\ F\ .\ \llparenthesis F,y,z,z\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,z,y,z\rrparenthesis)$
$(\lambda\ F\ .\ \llparenthesis F,z,z,y\rrparenthesis)$
$(* \textit{ three place } (y,y,y) *)\ (\lambda\ F\ .\ \llparenthesis F,y,y,y\rrparenthesis)$
$(* \textit{ three place } (z,z,z) *)\ (\lambda\ F\ .\ \llparenthesis F,z,z,z\rrparenthesis)$
$(* \ x\ y\ z *)$
$(* \textit{ three place } (x,...) *)\ (\lambda\ F\ .\ \llparenthesis F,x,y,z\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,x,z,y\rrparenthesis)$
$(* \textit{ three place } (y,...) *)\ (\lambda\ F\ .\ \llparenthesis F,y,x,z\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,y,z,x\rrparenthesis)$
$(* \textit{ three place } (z,...) *)\ (\lambda\ F\ .\ \llparenthesis F,z,x,y\rrparenthesis)\ (\lambda\ F\ .\ \llparenthesis F,z,y,x\rrparenthesis))$
unfolding *IsProperInXYZ-def*
by (*auto simp*: *meta-defs meta-aux*)

method *show-proper* = (*fast intro*: *IsProper-intros*)

The proving method *show-proper* is defined and is used in the subsequent theory whenever it is necessary to show that a map is proper.

A.2.3. Validity Syntax

abbreviation *validity-in* :: o⇒*i*⇒*bool* ([- *in* -] [*1*]) **where**
validity-in ≡ $\lambda\ \varphi\ v\ .\ v \models \varphi$
definition *actual-validity* :: o⇒*bool* ([-] [*1*]) **where**
actual-validity ≡ $\lambda\ \varphi\ .\ dw \models \varphi$
definition *necessary-validity* :: o⇒*bool* (□[-] [*1*]) **where**
necessary-validity ≡ $\lambda\ \varphi\ .\ \forall\ v\ .\ (v \models \varphi)$

A.3. General Quantification

Remark. *In order to define general quantifiers that can act on individuals as well as relations a type class is introduced which assumes the semantics of the all quantifier. This type class is then instantiated for individuals and relations.*

A.3.1. Type Class

Type class for quantifiable types:

class *quantifiable* = **fixes** *forall* :: ('*a*⇒o)⇒o (**binder** ∀ [*8*] *9*)
assumes *quantifiable-T8*: $(w \models (\forall\ x\ .\ \psi\ x)) = (\forall\ x\ .\ (w \models (\psi\ x)))$
begin
end

Semantics for the general all quantifier:

lemma (**in** *Semantics*) *T8*: **shows** $(w \models \forall\ x\ .\ \psi\ x) = (\forall\ x\ .\ (w \models \psi\ x))$
using *quantifiable-T8* **.**

A.3.2. Instantiations

instantiation ν :: *quantifiable*
begin
definition *forall*-ν :: (ν⇒o)⇒o **where** *forall*-ν ≡ $forall_\nu$
instance proof
fix w :: i **and** ψ :: ν⇒o
show $(w \models \forall x.\ \psi\ x) = (\forall x.\ (w \models \psi\ x))$
unfolding *forall*-ν-*def* **using** *Semantics.T8*-ν **.**
qed
end

instantiation o :: *quantifiable*
begin
definition *forall*-o :: (o⇒o)⇒o **where** *forall*-o ≡ $forall_o$
instance proof
fix w :: i **and** ψ :: o⇒o
show $(w \models \forall x.\ \psi\ x) = (\forall x.\ (w \models \psi\ x))$
unfolding *forall*-o-*def* **using** *Semantics.T8*-o **.**
qed
end

instantiation Π_1 :: *quantifiable*
begin
definition *forall*-Π_1 :: (Π_1⇒o)⇒o **where** *forall*-Π_1 ≡ $forall_1$
instance proof
fix w :: i **and** ψ :: Π_1⇒o
show $(w \models \forall x.\ \psi\ x) = (\forall x.\ (w \models \psi\ x))$

    unfolding *forall*-Π_1-*def* **using** *Semantics.T8-1* **.**
  qed
end

instantiation Π_2 :: *quantifiable*
begin
  definition *forall*-Π_2 :: $(\Pi_2{\Rightarrow}o){\Rightarrow}o$ **where** *forall*-Π_2 $\equiv$ $forall_2$
  instance proof
    fix w :: i **and** ψ :: $\Pi_2{\Rightarrow}o$
    show $(w \models \forall\, x.\ \psi\ x) = (\forall\, x.\ (w \models \psi\ x))$
      unfolding *forall*-Π_2-*def* **using** *Semantics.T8-2* **.**
  qed
end

instantiation Π_3 :: *quantifiable*
begin
  definition *forall*-Π_3 :: $(\Pi_3{\Rightarrow}o){\Rightarrow}o$ **where** *forall*-Π_3 $\equiv$ $forall_3$
  instance proof
    fix w :: i **and** ψ :: $\Pi_3{\Rightarrow}o$
    show $(w \models \forall\, x.\ \psi\ x) = (\forall\, x.\ (w \models \psi\ x))$
      unfolding *forall*-Π_3-*def* **using** *Semantics.T8-3* **.**
  qed
end

A.4. Basic Definitions

A.4.1. Derived Connectives

definition *conj*::o⇒o⇒o (**infixl &** *53*) **where**
  $conj \equiv \lambda\ x\ y\ .\ \neg(x \rightarrow \neg y)$
definition *disj*::o⇒o⇒o (**infixl** $\vee$ *52*) **where**
  $disj \equiv \lambda\ x\ y\ .\ \neg x \rightarrow y$
definition *equiv*::o⇒o⇒o (**infixl** $\equiv$ *51*) **where**
  $equiv \equiv \lambda\ x\ y\ .\ (x \rightarrow y)\ \&\ (y \rightarrow x)$
definition *diamond*::o⇒o ($\Diamond$- [*62*] *63*) **where**
  $diamond \equiv \lambda\ \varphi\ .\ \neg\Box\neg\varphi$
definition (**in** *quantifiable*) *exists* :: $('a{\Rightarrow}o){\Rightarrow}o$ (**binder** $\exists$ [*8*] *9*) **where**
    $exists \equiv \lambda\ \varphi\ .\ \neg(\forall\ \ x\ .\ \neg\varphi\ x)$

named-theorems *conn-defs*
declare *diamond-def*[*conn-defs*] *conj-def*[*conn-defs*]
        disj-def[*conn-defs*] *equiv-def*[*conn-defs*]
        exists-def[*conn-defs*]

A.4.2. Abstract and Ordinary Objects

definition *Ordinary* :: Π_1 $(O!)$ **where** *Ordinary* $\equiv \boldsymbol{\lambda} x.\ \Diamond(|E!,x^P|)$
definition *Abstract* :: Π_1 $(A!)$ **where** *Abstract* $\equiv \boldsymbol{\lambda} x.\ \neg\Diamond(|E!,x^P|)$

A.4.3. Identity Definitions

definition $basic\text{-}identity_E$::Π_2 **where**
  $basic\text{-}identity_E \equiv \boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ (|O!,x^P|)\ \&\ (|O!,y^P|)$
                $\&\ \Box(\forall\ \ F.\ (|F,x^P|) \equiv (|F,y^P|)))$

definition *basic-identity*$_E$*-infix*::$\kappa \Rightarrow \kappa \Rightarrow o$ (**infixl** $=_E$ *63*) **where**
  $x =_E y \equiv (|basic\text{-}identity_E,\ x,\ y|)$

definition *basic-identity*$_\kappa$ (**infixl** $=_\kappa$ *63*) **where**
  $basic\text{-}identity_\kappa \equiv \lambda\ x\ y\ .\ (x =_E y) \lor (|A!,x|)\ \&\ (|A!,y|)$
  $\&\ \Box(\forall\ F.\ \{|x,F|\} \equiv \{|y,F|\})$

definition *basic-identity*$_1$ (**infixl** $=_1$ *63*) **where**
  $basic\text{-}identity_1 \equiv \lambda\ F\ G\ .\ \Box(\forall\ x.\ \{|x^P,F|\} \equiv \{|x^P,G|\})$

definition *basic-identity*$_2$ :: $\Pi_2 \Rightarrow \Pi_2 \Rightarrow o$ (**infixl** $=_2$ *63*) **where**
  $basic\text{-}identity_2 \equiv \lambda\ F\ G\ .\ \forall\ x.\ ((\boldsymbol{\lambda} y.\ (|F,x^P,y^P|)) =_1 (\boldsymbol{\lambda} y.\ (|G,x^P,y^P|)))$
  $\&\ ((\boldsymbol{\lambda} y.\ (|F,y^P,x^P|)) =_1 (\boldsymbol{\lambda} y.\ (|G,y^P,x^P|)))$

definition *basic-identity*$_3$::$\Pi_3 \Rightarrow \Pi_3 \Rightarrow o$ (**infixl** $=_3$ *63*) **where**
  $basic\text{-}identity_3 \equiv \lambda\ F\ G\ .\ \forall\ x\ y.\ (\boldsymbol{\lambda} z.\ (|F,z^P,x^P,y^P|)) =_1 (\boldsymbol{\lambda} z.\ (|G,z^P,x^P,y^P|))$
  $\&\ (\boldsymbol{\lambda} z.\ (|F,x^P,z^P,y^P|)) =_1 (\boldsymbol{\lambda} z.\ (|G,x^P,z^P,y^P|))$
  $\&\ (\boldsymbol{\lambda} z.\ (|F,x^P,y^P,z^P|)) =_1 (\boldsymbol{\lambda} z.\ (|G,x^P,y^P,z^P|))$

definition *basic-identity*$_0$::$o \Rightarrow o \Rightarrow o$ (**infixl** $=_0$ *63*) **where**
  $basic\text{-}identity_0 \equiv \lambda\ F\ G\ .\ (\boldsymbol{\lambda} y.\ F) =_1 (\boldsymbol{\lambda} y.\ G)$

A.5. MetaSolver

Remark. *meta-solver is a resolution prover that translates expressions in the embedded logic to expressions in the meta-logic, resp. semantic expressions. The rules for connectives, quantifiers, exemplification and encoding are easy to prove. Futhermore rules for the defined identities are derived using more verbose proofs. By design the defined identities in the embedded logic coincide with the meta-logical equality.*

locale *MetaSolver*
begin
  interpretation *Semantics* **.**

  named-theorems *meta-intro*
  named-theorems *meta-elim*
  named-theorems *meta-subst*
  named-theorems *meta-cong*

  method *meta-solver* = (*assumption* | *rule meta-intro*
    | *erule meta-elim* | *drule meta-elim* | *subst meta-subst*
    | *subst* (*asm*) *meta-subst* | (*erule notE*; (*meta-solver*; *fail*))
)+

A.5.1. Rules for Implication

  lemma *ImplI*[*meta-intro*]: $([\varphi\ in\ v] \Longrightarrow [\psi\ in\ v]) \Longrightarrow ([\varphi \rightarrow \psi\ in\ v])$
    by (*simp add*: *Semantics.T5*)
  lemma *ImplE*[*meta-elim*]: $([\varphi \rightarrow \psi\ in\ v]) \Longrightarrow ([\varphi\ in\ v] \longrightarrow [\psi\ in\ v])$
    by (*simp add*: *Semantics.T5*)
  lemma *ImplS*[*meta-subst*]: $([\varphi \rightarrow \psi\ in\ v]) = ([\varphi\ in\ v] \longrightarrow [\psi\ in\ v])$
    by (*simp add*: *Semantics.T5*)

A.5.2. Rules for Negation

lemma *NotI*[*meta-intro*]: $\neg[\varphi \ in \ v] \Longrightarrow [\neg\varphi \ in \ v]$
by (*simp add*: *Semantics.T4*)
lemma *NotE*[*meta-elim*]: $[\neg\varphi \ in \ v] \Longrightarrow \neg[\varphi \ in \ v]$
by (*simp add*: *Semantics.T4*)
lemma *NotS*[*meta-subst*]: $[\neg\varphi \ in \ v] = (\neg[\varphi \ in \ v])$
by (*simp add*: *Semantics.T4*)

A.5.3. Rules for Conjunction

lemma *ConjI*[*meta-intro*]: $([\varphi \ in \ v] \wedge [\psi \ in \ v]) \Longrightarrow [\varphi \ \& \ \psi \ in \ v]$
by (*simp add*: *conj-def NotS ImplS*)
lemma *ConjE*[*meta-elim*]: $[\varphi \ \& \ \psi \ in \ v] \Longrightarrow ([\varphi \ in \ v] \wedge [\psi \ in \ v])$
by (*simp add*: *conj-def NotS ImplS*)
lemma *ConjS*[*meta-subst*]: $[\varphi \ \& \ \psi \ in \ v] = ([\varphi \ in \ v] \wedge [\psi \ in \ v])$
by (*simp add*: *conj-def NotS ImplS*)

A.5.4. Rules for Equivalence

lemma *EquivI*[*meta-intro*]: $([\varphi \ in \ v] \longleftrightarrow [\psi \ in \ v]) \Longrightarrow [\varphi \equiv \psi \ in \ v]$
by (*simp add*: *equiv-def NotS ImplS ConjS*)
lemma *EquivE*[*meta-elim*]: $[\varphi \equiv \psi \ in \ v] \Longrightarrow ([\varphi \ in \ v] \longleftrightarrow [\psi \ in \ v])$
by (*auto simp*: *equiv-def NotS ImplS ConjS*)
lemma *EquivS*[*meta-subst*]: $[\varphi \equiv \psi \ in \ v] = ([\varphi \ in \ v] \longleftrightarrow [\psi \ in \ v])$
by (*auto simp*: *equiv-def NotS ImplS ConjS*)

A.5.5. Rules for Disjunction

lemma *DisjI*[*meta-intro*]: $([\varphi \ in \ v] \vee [\psi \ in \ v]) \Longrightarrow [\varphi \lor \psi \ in \ v]$
by (*auto simp*: *disj-def NotS ImplS*)
lemma *DisjE*[*meta-elim*]: $[\varphi \lor \psi \ in \ v] \Longrightarrow ([\varphi \ in \ v] \vee [\psi \ in \ v])$
by (*auto simp*: *disj-def NotS ImplS*)
lemma *DisjS*[*meta-subst*]: $[\varphi \lor \psi \ in \ v] = ([\varphi \ in \ v] \vee [\psi \ in \ v])$
by (*auto simp*: *disj-def NotS ImplS*)

A.5.6. Rules for Necessity

lemma *BoxI*[*meta-intro*]: $(\bigwedge v.[\varphi \ in \ v]) \Longrightarrow [\Box\varphi \ in \ v]$
by (*simp add*: *Semantics.T6*)
lemma *BoxE*[*meta-elim*]: $[\Box\varphi \ in \ v] \Longrightarrow (\bigwedge v.[\varphi \ in \ v])$
by (*simp add*: *Semantics.T6*)
lemma *BoxS*[*meta-subst*]: $[\Box\varphi \ in \ v] = (\forall \ v.[\varphi \ in \ v])$
by (*simp add*: *Semantics.T6*)

A.5.7. Rules for Possibility

lemma *DiaI*[*meta-intro*]: $(\exists \, v.[\varphi \ in \ v]) \Longrightarrow [\Diamond\varphi \ in \ v]$
by (*metis BoxS NotS diamond-def*)
lemma *DiaE*[*meta-elim*]: $[\Diamond\varphi \ in \ v] \Longrightarrow (\exists \, v.[\varphi \ in \ v])$
by (*metis BoxS NotS diamond-def*)
lemma *DiaS*[*meta-subst*]: $[\Diamond\varphi \ in \ v] = (\exists \ v.[\varphi \ in \ v])$
by (*metis BoxS NotS diamond-def*)

A.5.8. Rules for Quantification

lemma $AllI[meta\text{-}intro]$: $(\bigwedge x.\ [\varphi\ x\ in\ v]) \Longrightarrow [\forall\ x.\ \varphi\ x\ in\ v]$
by $(auto\ simp: T8)$
lemma $AllE[meta\text{-}elim]$: $[\forall\, x.\ \varphi\ x\ in\ v] \Longrightarrow (\bigwedge x.[\varphi\ x\ in\ v])$
by $(auto\ simp: T8)$
lemma $AllS[meta\text{-}subst]$: $[\forall\, x.\ \varphi\ x\ in\ v] = (\forall\, x.[\varphi\ x\ in\ v])$
by $(auto\ simp: T8)$

A.5.8.1. Rules for Existence

lemma $ExIRule$: $([\varphi\ y\ in\ v]) \Longrightarrow [\exists\, x.\ \varphi\ x\ in\ v]$
by $(auto\ simp: exists\text{-}def\ Semantics.T8\ Semantics.T4)$
lemma $ExI[meta\text{-}intro]$: $(\exists\ y\ .\ [\varphi\ y\ in\ v]) \Longrightarrow [\exists\, x.\ \varphi\ x\ in\ v]$
by $(auto\ simp: exists\text{-}def\ Semantics.T8\ Semantics.T4)$
lemma $ExE[meta\text{-}elim]$: $[\exists\, x.\ \varphi\ x\ in\ v] \Longrightarrow (\exists\ y\ .\ [\varphi\ y\ in\ v])$
by $(auto\ simp: exists\text{-}def\ Semantics.T8\ Semantics.T4)$
lemma $ExS[meta\text{-}subst]$: $[\exists\, x.\ \varphi\ x\ in\ v] = (\exists\ y\ .\ [\varphi\ y\ in\ v])$
by $(auto\ simp: exists\text{-}def\ Semantics.T8\ Semantics.T4)$
lemma $ExERule$: **assumes** $[\exists\, x.\ \varphi\ x\ in\ v]$ **obtains** x **where** $[\varphi\ x\ in\ v]$
using $ExE\ assms$ **by** $auto$

A.5.9. Rules for Actuality

lemma $ActualI[meta\text{-}intro]$: $[\varphi\ in\ dw] \Longrightarrow [\boldsymbol{\mathcal{A}}\varphi\ in\ v]$
by $(auto\ simp: Semantics.T7)$
lemma $ActualE[meta\text{-}elim]$: $[\boldsymbol{\mathcal{A}}\varphi\ in\ v] \Longrightarrow [\varphi\ in\ dw]$
by $(auto\ simp: Semantics.T7)$
lemma $ActualS[meta\text{-}subst]$: $[\boldsymbol{\mathcal{A}}\varphi\ in\ v] = [\varphi\ in\ dw]$
by $(auto\ simp: Semantics.T7)$

A.5.10. Rules for Encoding

lemma $EncI[meta\text{-}intro]$:
assumes $\exists\ r\ o_1\ .\ Some\ r = d_1\ F \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in en\ r$
shows $[\{\!|x,F|\!\}\ in\ v]$
using $assms$ **by** $(auto\ simp: Semantics.T2)$
lemma $EncE[meta\text{-}elim]$:
assumes $[\{\!|x,F|\!\}\ in\ v]$
shows $\exists\ r\ o_1\ .\ Some\ r = d_1\ F \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in en\ r$
using $assms$ **by** $(auto\ simp: Semantics.T2)$
lemma $EncS[meta\text{-}subst]$:
$[\{\!|x,F|\!\}\ in\ v] = (\exists\ r\ o_1\ .\ Some\ r = d_1\ F \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in en\ r)$
by $(auto\ simp: Semantics.T2)$

A.5.11. Rules for Exemplification

A.5.11.1. Zero-place Relations

lemma $Exe0I[meta\text{-}intro]$:
assumes $\exists\ r\ .\ Some\ r = d_0\ p \wedge ex0\ r\ v$
shows $[(\!|p|\!)\ in\ v]$
using $assms$ **by** $(auto\ simp: Semantics.T3)$
lemma $Exe0E[meta\text{-}elim]$:
assumes $[(\!|p|\!)\ in\ v]$
shows $\exists\ r\ .\ Some\ r = d_0\ p \wedge ex0\ r\ v$
using $assms$ **by** $(auto\ simp: Semantics.T3)$
lemma $Exe0S[meta\text{-}subst]$:

$[\llparenthesis p \rrparenthesis\ in\ v] = (\exists\ r\ .\ Some\ r = d_0\ p \wedge ex0\ r\ v)$
by (*auto simp*: *Semantics.T3*)

A.5.11.2. One-Place Relations

lemma *Exe1I*[*meta-intro*]:
assumes $\exists\ r\ o_1\ .\ Some\ r = d_1\ F \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in ex1\ r\ v$
shows $[\llparenthesis F,x \rrparenthesis\ in\ v]$
using *assms* **by** (*auto simp*: *Semantics.T1-1*)
lemma *Exe1E*[*meta-elim*]:
assumes $[\llparenthesis F,x \rrparenthesis\ in\ v]$
shows $\exists\ r\ o_1\ .\ Some\ r = d_1\ F \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in ex1\ r\ v$
using *assms* **by** (*auto simp*: *Semantics.T1-1*)
lemma *Exe1S*[*meta-subst*]:
$[\llparenthesis F,x \rrparenthesis\ in\ v] = (\exists\ r\ o_1\ .\ Some\ r = d_1\ F \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in ex1\ r\ v)$
by (*auto simp*: *Semantics.T1-1*)

A.5.11.3. Two-Place Relations

lemma *Exe2I*[*meta-intro*]:
assumes $\exists\ r\ o_1\ o_2\ .\ Some\ r = d_2\ F \wedge Some\ o_1 = d_\kappa\ x$
$\wedge\ Some\ o_2 = d_\kappa\ y \wedge (o_1,\ o_2) \in ex2\ r\ v$
shows $[\llparenthesis F,x,y \rrparenthesis\ in\ v]$
using *assms* **by** (*auto simp*: *Semantics.T1-2*)
lemma *Exe2E*[*meta-elim*]:
assumes $[\llparenthesis F,x,y \rrparenthesis\ in\ v]$
shows $\exists\ r\ o_1\ o_2\ .\ Some\ r = d_2\ F \wedge Some\ o_1 = d_\kappa\ x$
$\wedge\ Some\ o_2 = d_\kappa\ y \wedge (o_1,\ o_2) \in ex2\ r\ v$
using *assms* **by** (*auto simp*: *Semantics.T1-2*)
lemma *Exe2S*[*meta-subst*]:
$[\llparenthesis F,x,y \rrparenthesis\ in\ v] = (\exists\ r\ o_1\ o_2\ .\ Some\ r = d_2\ F \wedge Some\ o_1 = d_\kappa\ x$
$\wedge\ Some\ o_2 = d_\kappa\ y \wedge (o_1,\ o_2) \in ex2\ r\ v)$
by (*auto simp*: *Semantics.T1-2*)

A.5.11.4. Three-Place Relations

lemma *Exe3I*[*meta-intro*]:
assumes $\exists\ r\ o_1\ o_2\ o_3\ .\ Some\ r = d_3\ F \wedge Some\ o_1 = d_\kappa\ x$
$\wedge\ Some\ o_2 = d_\kappa\ y \wedge Some\ o_3 = d_\kappa\ z$
$\wedge\ (o_1,\ o_2,\ o_3) \in ex3\ r\ v$
shows $[\llparenthesis F,x,y,z \rrparenthesis\ in\ v]$
using *assms* **by** (*auto simp*: *Semantics.T1-3*)
lemma *Exe3E*[*meta-elim*]:
assumes $[\llparenthesis F,x,y,z \rrparenthesis\ in\ v]$
shows $\exists\ r\ o_1\ o_2\ o_3\ .\ Some\ r = d_3\ F \wedge Some\ o_1 = d_\kappa\ x$
$\wedge\ Some\ o_2 = d_\kappa\ y \wedge Some\ o_3 = d_\kappa\ z$
$\wedge\ (o_1,\ o_2,\ o_3) \in ex3\ r\ v$
using *assms* **by** (*auto simp*: *Semantics.T1-3*)
lemma *Exe3S*[*meta-subst*]:
$[\llparenthesis F,x,y,z \rrparenthesis\ in\ v] = (\exists\ r\ o_1\ o_2\ o_3\ .\ Some\ r = d_3\ F \wedge Some\ o_1 = d_\kappa\ x$
$\wedge\ Some\ o_2 = d_\kappa\ y \wedge Some\ o_3 = d_\kappa\ z$
$\wedge\ (o_1,\ o_2,\ o_3) \in ex3\ r\ v)$
by (*auto simp*: *Semantics.T1-3*)

A.5.12. Rules for Being Ordinary

lemma *OrdI*[*meta-intro*]:
assumes $\exists\ o_1\ y.\ Some\ o_1 = d_\kappa\ x \wedge o_1 = \omega\nu\ y$

shows $[(\!|O!,x|\!)\ \mathit{in}\ v]$
proof –
have *IsProperInX* $(\lambda x.\ \Diamond(\!|E!,x|\!))$
by *show-proper*
moreover have $[\Diamond(\!|E!,x|\!)\ \mathit{in}\ v]$
apply *meta-solver*
using *ConcretenessSemantics1* propex_1 *assms* **by** *fast*
ultimately show $[(\!|O!,x|\!)\ \mathit{in}\ v]$
unfolding *Ordinary-def*
using *D5-1* propex_1 *assms ConcretenessSemantics1 Exe1S*
by *blast*
qed
lemma *OrdE*[*meta-elim*]:
assumes $[(\!|O!,x|\!)\ \mathit{in}\ v]$
shows $\exists\ o_1\ y.\ \mathit{Some}\ o_1 = d_\kappa\ x \wedge o_1 = \omega\nu\ y$
proof –
have $\exists\, r\ o_1.\ \mathit{Some}\ r = d_1\ O! \wedge \mathit{Some}\ o_1 = d_\kappa\ x \wedge o_1 \in \mathit{ex1}\ r\ v$
using *assms Exe1E* **by** *simp*
moreover have *IsProperInX* $(\lambda x.\ \Diamond(\!|E!,x|\!))$
by *show-proper*
ultimately have $[\Diamond(\!|E!,x|\!)\ \mathit{in}\ v]$
using *D5-1* **unfolding** *Ordinary-def* **by** *fast*
thus *?thesis*
apply – **apply** *meta-solver*
using *ConcretenessSemantics2* **by** *blast*
qed
lemma *OrdS*[*meta-cong*]:
$[(\!|O!,x|\!)\ \mathit{in}\ v] = (\exists\ o_1\ y.\ \mathit{Some}\ o_1 = d_\kappa\ x \wedge o_1 = \omega\nu\ y)$
using *OrdI OrdE* **by** *blast*

A.5.13. Rules for Being Abstract

lemma *AbsI*[*meta-intro*]:
assumes $\exists\ o_1\ y.\ \mathit{Some}\ o_1 = d_\kappa\ x \wedge o_1 = \alpha\nu\ y$
shows $[(\!|A!,x|\!)\ \mathit{in}\ v]$
proof –
have *IsProperInX* $(\lambda x.\ \neg\Diamond(\!|E!,x|\!))$
by *show-proper*
moreover have $[\neg\Diamond(\!|E!,x|\!)\ \mathit{in}\ v]$
apply *meta-solver*
using *ConcretenessSemantics2* propex_1 *assms*
by (*metis* ν*.distinct*(*1*) *option.sel*)
ultimately show $[(\!|A!,x|\!)\ \mathit{in}\ v]$
unfolding *Abstract-def*
using *D5-1* propex_1 *assms ConcretenessSemantics1 Exe1S*
by *blast*
qed
lemma *AbsE*[*meta-elim*]:
assumes $[(\!|A!,x|\!)\ \mathit{in}\ v]$
shows $\exists\ o_1\ y.\ \mathit{Some}\ o_1 = d_\kappa\ x \wedge o_1 = \alpha\nu\ y$
proof –
have *1*: *IsProperInX* $(\lambda x.\ \neg\Diamond(\!|E!,x|\!))$
by *show-proper*
have $\exists\, r\ o_1.\ \mathit{Some}\ r = d_1\ A! \wedge \mathit{Some}\ o_1 = d_\kappa\ x \wedge o_1 \in \mathit{ex1}\ r\ v$
using *assms Exe1E* **by** *simp*
moreover hence $[\neg\Diamond(\!|E!,x|\!)\ \mathit{in}\ v]$
using *D5-1*[*OF 1*]
unfolding *Abstract-def* **by** *fast*

ultimately show *?thesis*
apply − **apply** *meta-solver*
using *ConcretenessSemantics1* $propex_1$
by (*metis* ν*.exhaust*)
qed
lemma *AbsS*[*meta-cong*]:
$[(|A!,x|)\ in\ v] = (\exists\ o_1\ y.\ Some\ o_1 = d_\kappa\ x \wedge o_1 = \alpha\nu\ y)$
using *AbsI AbsE* **by** *blast*

A.5.14. Rules for Definite Descriptions

lemma *TheEqI*:
assumes $\bigwedge x.\ [\varphi\ x\ in\ dw] = [\psi\ x\ in\ dw]$
shows $(\iota x.\ \varphi\ x) = (\iota x.\ \psi\ x)$
proof −
have *1*: $d_\kappa\ (\iota x.\ \varphi\ x) = d_\kappa\ (\iota x.\ \psi\ x)$
using *assms D3* **unfolding** w_0*-def* **by** *simp*
{
assume $\exists\ o_1\ .\ Some\ o_1 = d_\kappa\ (\iota x.\ \varphi\ x)$
hence *?thesis* **using** *1* d_κ*-inject* **by** *force*
}
moreover {
assume $\neg(\exists\ o_1\ .\ Some\ o_1 = d_\kappa\ (\iota x.\ \varphi\ x))$
hence *?thesis* **using** *1 D3*
by (*metis* d_κ*.rep-eq eval*κ*-inverse*)
}
ultimately show *?thesis* **by** *blast*
qed

A.5.15. Rules for Identity

A.5.15.1. Ordinary Objects

lemma $Eq_E I$[*meta-intro*]:
assumes $\exists\ o_1\ X\ o_2.\ Some\ o_1 = d_\kappa\ x \wedge Some\ o_2 = d_\kappa\ y \wedge o_1 = o_2 \wedge o_1 = \omega\nu\ X$
shows $[x =_E y\ in\ v]$
proof −
obtain $o_1\ X\ o_2$ **where** *1*:
$Some\ o_1 = d_\kappa\ x \wedge Some\ o_2 = d_\kappa\ y \wedge o_1 = o_2 \wedge o_1 = \omega\nu\ X$
using *assms* **by** *auto*
obtain r **where** *2*:
$Some\ r = d_2\ basic\text{-}identity_E$
using $propex_2$ **by** *auto*
have $[(|O!,x|)\ \&\ (|O!,y|)\ \&\ \Box(\forall F.\ (|F,x|) \equiv (|F,y|))\ in\ v]$
proof −
have $[(|O!,x|)\ in\ v] \wedge [(|O!,y|)\ in\ v]$
using *OrdI 1* **by** *blast*
moreover have $[\Box(\forall F.\ (|F,x|) \equiv (|F,y|))\ in\ v]$
apply *meta-solver* **using** *1* **by** *force*
ultimately show *?thesis* **using** *ConjI* **by** *simp*
qed
moreover have *IsProperInXY* $(\lambda\ x\ y\ .\ (|O!,x|)\ \&\ (|O!,y|)\ \&\ \Box(\forall F.\ (|F,x|) \equiv (|F,y|)))$
by *show-proper*
ultimately have $(o_1,\ o_2) \in ex2\ r\ v$
using *D5-2 1 2*
unfolding $basic\text{-}identity_E$*-def* **by** *fast*
thus $[x =_E y\ in\ v]$

```
      using Exe2I 1 2
      unfolding basic-identity_E-infix-def basic-identity_E-def
      by blast
  qed
lemma Eq_E E[meta-elim]:
  assumes [x =_E y in v]
  shows ∃ o1 X o2. Some o1 = d_κ x ∧ Some o2 = d_κ y ∧ o1 = o2 ∧ o1 = ων X
proof −
  have IsProperInXY (λ x y . (|O!,x|) & (|O!,y|) & □(∀ F. (|F,x|) ≡ (|F,y|)))
    by show-proper
  hence 1: [(|O!,x|) & (|O!,y|) & □(∀ F. (|F,x|) ≡ (|F,y|)) in v]
    using assms unfolding basic-identity_E-def basic-identity_E-infix-def
    using D4-2 T1-2 D5-2 by meson
  hence 2: ∃ o1 o2 X Y . Some o1 = d_κ x ∧ o1 = ων X
                    ∧ Some o2 = d_κ y ∧ o2 = ων Y
    apply (subst (asm) ConjS)
    apply (subst (asm) ConjS)
    using OrdE by auto
  then obtain o1 o2 X Y where 3:
    Some o1 = d_κ x ∧ o1 = ων X ∧ Some o2 = d_κ y ∧ o2 = ων Y
    by auto
  have ∃ r . Some r = d1 (λ z . makeo (λ w s . d_κ (z^P) = Some o1))
    using propex1 by auto
  then obtain r where 4:
    Some r = d1 (λ z . makeo (λ w s . d_κ (z^P) = Some o1))
    by auto
  hence 5: r = (λu s w. ∃ x . νυ x = u ∧ Some x = Some o1)
    unfolding lambdabinder1-def d1-def d_κ-proper
    apply transfer
    by simp
  have [□(∀ F. (|F,x|) ≡ (|F,y|)) in v]
    using 1 using ConjE by blast
  hence 6: ∀ v F . [(|F,x|) in v] ⟷ [(|F,y|) in v]
    using BoxE EquivE AllE by fast
  hence 7: ∀ v . (o1 ∈ ex1 r v) = (o2 ∈ ex1 r v)
    using 2 4 unfolding valid-in-def
    by (metis 3 6 d1.rep-eq d_κ-inject d_κ-proper ex1-def evalo-inverse exe1.rep-eq
        mem-Collect-eq option.sel rep-proper-id νκ-proper valid-in.abs-eq)
  have o1 ∈ ex1 r v
    using 5 3 unfolding ex1-def by (simp add: meta-aux)
  hence o2 ∈ ex1 r v
    using 7 by auto
  hence o1 = o2
    unfolding ex1-def 5 using 3 by (auto simp: meta-aux)
  thus ?thesis
    using 3 by auto
qed
lemma Eq_E S[meta-subst]:
  [x =_E y in v] = (∃ o1 X o2. Some o1 = d_κ x ∧ Some o2 = d_κ y
                          ∧ o1 = o2 ∧ o1 = ων X)
  using Eq_E I Eq_E E by blast
```

A.5.15.2. Individuals

```
lemma EqκI[meta-intro]:
  assumes ∃ o1 o2. Some o1 = d_κ x ∧ Some o2 = d_κ y ∧ o1 = o2
  shows [x =_κ y in v]
proof −
```

```
  have x = y using assms dκ-inject by meson
  moreover have [x =κ x in v]
    unfolding basic-identityκ-def
    apply meta-solver
    by (metis (no-types, lifting) assms AbsI Exe1E ν.exhaust)
  ultimately show ?thesis by auto
qed
lemma Eqκ-prop:
  assumes [x =κ y in v]
  shows [φ x in v] = [φ y in v]
proof −
  have [x =E y ∨ (|A!,x|) & (|A!,y|) & □(∀ F. {|x,F|} ≡ {|y,F|}) in v]
    using assms unfolding basic-identityκ-def by simp
  moreover {
    assume [x =E y in v]
    hence (∃ o1 o2. Some o1 = dκ x ∧ Some o2 = dκ y ∧ o1 = o2)
      using EqE E by fast
  }
  moreover {
    assume 1: [(|A!,x|) & (|A!,y|) & □(∀ F. {|x,F|} ≡ {|y,F|}) in v]
    hence 2: (∃ o1 o2 X Y. Some o1 = dκ x ∧ Some o2 = dκ y
                         ∧ o1 = αν X ∧ o2 = αν Y)
      using AbsE ConjE by meson
    moreover then obtain o1 o2 X Y where 3:
      Some o1 = dκ x ∧ Some o2 = dκ y ∧ o1 = αν X ∧ o2 = αν Y
      by auto
    moreover have 4: [□(∀ F. {|x,F|} ≡ {|y,F|}) in v]
      using 1 ConjE by blast
    hence 6: ∀ v F . [{|x,F|} in v] ⟷ [{|y,F|} in v]
      using BoxE AllE EquivE by fast
    hence 7: ∀ v r. (∃ o1. Some o1 = dκ x ∧ o1 ∈ en r)
              = (∃ o1. Some o1 = dκ y ∧ o1 ∈ en r)
      apply − apply meta-solver
      using propex1 d1-inject apply simp
      apply transfer by simp
    hence 8: ∀ r. (o1 ∈ en r) = (o2 ∈ en r)
      using 3 dκ-inject dκ-proper apply simp
      by (metis option.inject)
    hence ∀ r. (o1 ∈ r) = (o2 ∈ r)
      unfolding en-def using 3
      by (metis Collect-cong Collect-mem-eq ν.simps(6)
             mem-Collect-eq makeΠ1-cases)
    hence (o1 ∈ { x . o1 = x }) = (o2 ∈ { x . o1 = x })
      by metis
    hence o1 = o2 by simp
    hence (∃ o1 o2. Some o1 = dκ x ∧ Some o2 = dκ y ∧ o1 = o2)
      using 3 by auto
  }
  ultimately have x = y
    using DisjS using Semantics.dκ-inject by auto
  thus (v ⊨ (φ x)) = (v ⊨ (φ y)) by simp
qed
lemma EqκE[meta-elim]:
  assumes [x =κ y in v]
  shows ∃ o1 o2. Some o1 = dκ x ∧ Some o2 = dκ y ∧ o1 = o2
proof −
  have ∀ φ . (v ⊨ φ x) = (v ⊨ φ y)
    using assms Eqκ-prop by blast
```

moreover obtain φ **where** *φ-prop*:
$\varphi = (\lambda\ \alpha\ .\ makeo\ (\lambda\ w\ s\ .\ (\exists\ o_1\ o_2.\ Some\ o_1 = d_\kappa\ x$
$\wedge\ Some\ o_2 = d_\kappa\ \alpha \wedge o_1 = o_2)))$
by *auto*
ultimately have $(v \models \varphi\ x) = (v \models \varphi\ y)$ **by** *metis*
moreover have $(v \models \varphi\ x)$
using *assms* **unfolding** *φ-prop basic-identity$_\kappa$-def*
by (*metis* (*mono-tags*, *lifting*) *AbsS ConjE DisjS*
Eq$_E$S valid-in.abs-eq)
ultimately have $(v \models \varphi\ y)$ **by** *auto*
thus *?thesis*
unfolding *φ-prop*
by (*simp add*: *valid-in-def meta-aux*)
qed
lemma *EqκS*[*meta-subst*]:
$[x =_\kappa y\ in\ v] = (\exists\ o_1\ o_2.\ Some\ o_1 = d_\kappa\ x \wedge Some\ o_2 = d_\kappa\ y \wedge o_1 = o_2)$
using *EqκI EqκE* **by** *blast*

A.5.15.3. One-Place Relations

lemma *Eq$_1$I*[*meta-intro*]: $F = G \Longrightarrow [F =_1 G\ in\ v]$
unfolding *basic-identity$_1$-def*
apply (*rule BoxI*, *rule AllI*, *rule EquivI*)
by *simp*
lemma *Eq$_1$E*[*meta-elim*]: $[F =_1 G\ in\ v] \Longrightarrow F = G$
unfolding *basic-identity$_1$-def*
apply (*drule BoxE*, *drule-tac* x=($\alpha\nu$ { F }) **in** *AllE*, *drule EquivE*)
apply (*simp add*: *Semantics.T2*)
unfolding *en-def d$_\kappa$-def d$_1$-def*
using *νκ-proper rep-proper-id*
by (*simp add*: *rep-def proper-def meta-aux νκ.rep-eq*)
lemma *Eq$_1$S*[*meta-subst*]: $[F =_1 G\ in\ v] = (F = G)$
using *Eq$_1$I Eq$_1$E* **by** *auto*
lemma *Eq$_1$-prop*: $[F =_1 G\ in\ v] \Longrightarrow [\varphi\ F\ in\ v] = [\varphi\ G\ in\ v]$
using *Eq$_1$E* **by** *blast*

A.5.15.4. Two-Place Relations

lemma *Eq$_2$I*[*meta-intro*]: $F = G \Longrightarrow [F =_2 G\ in\ v]$
unfolding *basic-identity$_2$-def*
apply (*rule AllI*, *rule ConjI*, (*subst Eq$_1$S*)+)
by *simp*
lemma *Eq$_2$E*[*meta-elim*]: $[F =_2 G\ in\ v] \Longrightarrow F = G$
proof −
assume $[F =_2 G\ in\ v]$
hence *1*: $[\forall\ x.\ (\boldsymbol{\lambda} y.\ (\!|F,x^P,y^P|\!)) =_1 (\boldsymbol{\lambda} y.\ (\!|G,x^P,y^P|\!))\ in\ v]$
unfolding *basic-identity$_2$-def*
apply − **apply** *meta-solver* **by** *auto*
{
fix $u\ v\ s\ w$
obtain x **where** *x-def*: $\nu\upsilon\ x = v$ **by** (*metis νυ-surj surj-def*)
obtain a **where** *a-def*:
$a = (\lambda u\ s\ w.\ \exists xa.\ \nu\upsilon\ xa = u \wedge eval\Pi_2\ F\ (\nu\upsilon\ x)\ (\nu\upsilon\ xa)\ s\ w)$
by *auto*
obtain b **where** *b-def*:
$b = (\lambda u\ s\ w.\ \exists xa.\ \nu\upsilon\ xa = u \wedge eval\Pi_2\ G\ (\nu\upsilon\ x)\ (\nu\upsilon\ xa)\ s\ w)$
by *auto*
have $a = b$ **unfolding** *a-def b-def*

using *1* **apply** − **apply** *meta-solver*
by (*auto simp*: *meta-defs meta-aux make*Π_1*-inject*)
hence $a\ u\ s\ w = b\ u\ s\ w$ **by** *auto*
hence $(eval\Pi_2\ F\ (\nu\upsilon\ x)\ u\ s\ w) = (eval\Pi_2\ G\ (\nu\upsilon\ x)\ u\ s\ w)$
unfolding *a-def b-def*
by (*metis* (*no-types*, *hide-lams*) *νυ-surj surj-def*)
hence $(eval\Pi_2\ F\ v\ u\ s\ w) = (eval\Pi_2\ G\ v\ u\ s\ w)$
unfolding *x-def* **by** *auto*
}
hence $(eval\Pi_2\ F) = (eval\Pi_2\ G)$ **by** *blast*
thus $F = G$ **by** (*simp add*: *eval*Π_2*-inject*)
qed
lemma *Eq*$_2$*S*[*meta-subst*]: $[F =_2 G\ in\ v] = (F = G)$
using *Eq*$_2$*I Eq*$_2$*E* **by** *auto*
lemma *Eq*$_2$*-prop*: $[F =_2 G\ in\ v] \Longrightarrow [\varphi\ F\ in\ v] = [\varphi\ G\ in\ v]$
using *Eq*$_2$*E* **by** *blast*

A.5.15.5. Three-Place Relations

lemma *Eq*$_3$*I*[*meta-intro*]: $F = G \Longrightarrow [F =_3 G\ in\ v]$
apply (*simp add*: *meta-defs meta-aux conn-defs forall-ν-def basic-identity*$_3$*-def*)
using *MetaSolver.Eq*$_1$*I valid-in.rep-eq* **by** *auto*
lemma *Eq*$_3$*E*[*meta-elim*]: $[F =_3 G\ in\ v] \Longrightarrow F = G$
proof −

assume $[F =_3 G\ in\ v]$
hence *1*: $[\forall\ x\ y.\ (\boldsymbol{\lambda} z.\ (\!|F,x^P,y^P,z^P|\!)) =_1 (\boldsymbol{\lambda} z.\ (\!|G,x^P,y^P,z^P|\!))\ in\ v]$
unfolding *basic-identity*$_3$*-def*
apply − **apply** *meta-solver* **by** *auto*
{
fix $u\ v\ r\ s\ w$
obtain x **where** *x-def*: $\nu\upsilon\ x = v$ **by** (*metis νυ-surj surj-def*)
obtain y **where** *y-def*: $\nu\upsilon\ y = r$ **by** (*metis νυ-surj surj-def*)
obtain a **where** *a-def*:
$a = (\lambda u\ s\ w.\ \exists xa.\ \nu\upsilon\ xa = u \wedge eval\Pi_3\ F\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ (\nu\upsilon\ xa)\ s\ w)$
by *auto*
obtain b **where** *b-def*:
$b = (\lambda u\ s\ w.\ \exists xa.\ \nu\upsilon\ xa = u \wedge eval\Pi_3\ G\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ (\nu\upsilon\ xa)\ s\ w)$
by *auto*
have $a = b$ **unfolding** *a-def b-def*
using *1* **apply** − **apply** *meta-solver*
by (*auto simp*: *meta-defs meta-aux make*Π_1*-inject*)
hence $a\ u\ s\ w = b\ u\ s\ w$ **by** *auto*
hence $(eval\Pi_3\ F\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ u\ s\ w) = (eval\Pi_3\ G\ (\nu\upsilon\ x)\ (\nu\upsilon\ y)\ u\ s\ w)$
unfolding *a-def b-def*
by (*metis* (*no-types*, *hide-lams*) *νυ-surj surj-def*)
hence $(eval\Pi_3\ F\ v\ r\ u\ s\ w) = (eval\Pi_3\ G\ v\ r\ u\ s\ w)$
unfolding *x-def y-def* **by** *auto*
}
hence $(eval\Pi_3\ F) = (eval\Pi_3\ G)$ **by** *blast*
thus $F = G$ **by** (*simp add*: *eval*Π_3*-inject*)
qed
lemma *Eq*$_3$*S*[*meta-subst*]: $[F =_3 G\ in\ v] = (F = G)$
using *Eq*$_3$*I Eq*$_3$*E* **by** *auto*
lemma *Eq*$_3$*-prop*: $[F =_3 G\ in\ v] \Longrightarrow [\varphi\ F\ in\ v] = [\varphi\ G\ in\ v]$
using *Eq*$_3$*E* **by** *blast*

A.5.15.6. Propositions

```
  lemma Eq_0 I[meta-intro]: x = y ⟹ [x =_0 y in v]
    unfolding basic-identity_0-def by (simp add: Eq_1 S)
  lemma Eq_0 E[meta-elim]: [F =_0 G in v] ⟹ F = G
    proof −
      assume [F =_0 G in v]
      hence [(𝛌y. F) =_1 (𝛌y. G) in v]
        unfolding basic-identity_0-def by simp
      hence (𝛌y. F) = (𝛌y. G)
        using Eq_1 S by simp
      hence (λu s w. (∃x. νυ x = u) ∧ evalo F s w)
           = (λu s w. (∃x. νυ x = u) ∧ evalo G s w)
        apply (simp add: meta-defs meta-aux)
        by (metis (no-types, lifting) UNIV-I makeΠ_1-inverse)
      hence ⋀s w.(evalo F s w) = (evalo G s w)
        by metis
      hence (evalo F) = (evalo G) by blast
      thus F = G
      by (metis evalo-inverse)
    qed
  lemma Eq_0 S[meta-subst]: [F =_0 G in v] = (F = G)
    using Eq_0 I Eq_0 E by auto
  lemma Eq_0-prop: [F =_0 G in v] ⟹ [φ F in v] = [φ G in v]
    using Eq_0 E by blast

end
```

A.6. General Identity

Remark. *In order to define a general identity symbol that can act on all types of terms a type class is introduced which assumes the substitution property which is needed to state the axioms later. This type class is then instantiated for all applicable types.*

A.6.1. Type Classes

```
class identifiable =
fixes identity :: 'a⇒'a⇒o (infixl = 63)
assumes l-identity:
  w ⊨ x = y ⟹ w ⊨ φ x ⟹ w ⊨ φ y
begin
  abbreviation notequal (infixl ≠ 63) where
    notequal ≡ λ x y . ¬(x = y)
end

class quantifiable-and-identifiable = quantifiable + identifiable
begin
  definition exists-unique::('a⇒o)⇒o (binder ∃! [8] 9) where
    exists-unique ≡ λ φ . ∃ α . φ α & (∀β. φ β → β = α)

  declare exists-unique-def[conn-defs]
end
```

A.6.2. Instantiations

instantiation κ :: *identifiable*
begin
  definition *identity-*κ **where** *identity-*$\kappa \equiv$ *basic-identity*$_\kappa$
  instance proof
    fix $x\ y :: \kappa$ **and** $w\ \varphi$
    show $[x = y\ in\ w] \Longrightarrow [\varphi\ x\ in\ w] \Longrightarrow [\varphi\ y\ in\ w]$
      unfolding *identity-*κ*-def*
      using *MetaSolver.Eq*κ*-prop* **..**
  qed
end

instantiation ν :: *identifiable*
begin
  definition *identity-*ν **where** *identity-*$\nu \equiv \lambda\ x\ y\ .\ x^P = y^P$
  instance proof
    fix $\alpha :: \nu$ **and** $\beta :: \nu$ **and** $v\ \varphi$
    assume $v \models \alpha = \beta$
    hence $v \models \alpha^P = \beta^P$
      unfolding *identity-*ν*-def* **by** *auto*
    hence $\bigwedge\varphi.(v \models \varphi\ (\alpha^P)) \Longrightarrow (v \models \varphi\ (\beta^P))$
      using *l-identity* **by** *auto*
    hence $(v \models \varphi\ (rep\ (\alpha^P))) \Longrightarrow (v \models \varphi\ (rep\ (\beta^P)))$
      by *meson*
    thus $(v \models \varphi\ \alpha) \Longrightarrow (v \models \varphi\ \beta)$
      by (*simp only*: *rep-proper-id*)
  qed
end

instantiation Π_1 :: *identifiable*
begin
  definition *identity-*Π_1 **where** *identity-*$\Pi_1 \equiv$ *basic-identity*$_1$
  instance proof
    fix $F\ G :: \Pi_1$ **and** $w\ \varphi$
    show $(w \models F = G) \Longrightarrow (w \models \varphi\ F) \Longrightarrow (w \models \varphi\ G)$
      unfolding *identity-*Π_1*-def* **using** *MetaSolver.Eq*$_1$*-prop* **..**
  qed
end

instantiation Π_2 :: *identifiable*
begin
  definition *identity-*Π_2 **where** *identity-*$\Pi_2 \equiv$ *basic-identity*$_2$
  instance proof
    fix $F\ G :: \Pi_2$ **and** $w\ \varphi$
    show $(w \models F = G) \Longrightarrow (w \models \varphi\ F) \Longrightarrow (w \models \varphi\ G)$
      unfolding *identity-*Π_2*-def* **using** *MetaSolver.Eq*$_2$*-prop* **..**
  qed
end

instantiation Π_3 :: *identifiable*
begin
  definition *identity-*Π_3 **where** *identity-*$\Pi_3 \equiv$ *basic-identity*$_3$
  instance proof
    fix $F\ G :: \Pi_3$ **and** $w\ \varphi$
    show $(w \models F = G) \Longrightarrow (w \models \varphi\ F) \Longrightarrow (w \models \varphi\ G)$
      unfolding *identity-*Π_3*-def* **using** *MetaSolver.Eq*$_3$*-prop* **..**
  qed

end

instantiation o :: *identifiable*
begin
  definition *identity*-o **where** *identity*-o $\equiv$ $basic\text{-}identity_0$
  instance proof
    fix F G :: o **and** w φ
    show $(w \models F = G) \Longrightarrow (w \models \varphi\ F) \Longrightarrow (w \models \varphi\ G)$
      unfolding *identity*-o-*def* **using** $MetaSolver.Eq_0$-*prop* **..**
  qed
end

instance ν :: *quantifiable-and-identifiable* **..**
instance Π_1 :: *quantifiable-and-identifiable* **..**
instance Π_2 :: *quantifiable-and-identifiable* **..**
instance Π_3 :: *quantifiable-and-identifiable* **..**
instance o :: *quantifiable-and-identifiable* **..**

A.6.3. New Identity Definitions

Remark. *The basic definitions of identity used the type specific quantifiers and identities. We now introduce equivalent definitions that use the general identity and general quantifiers.*

named-theorems *identity-defs*
lemma $identity_E$-*def*[*identity-defs*]:
  $basic\text{-}identity_E \equiv \boldsymbol{\lambda}^2\ (\lambda x\ y.\ (|O!,x^P|)\ \&\ (|O!,y^P|)\ \&\ \Box(\forall\, F.\ (|F,x^P|) \equiv (|F,y^P|)))$
  unfolding $basic\text{-}identity_E$-*def* *forall*-Π_1-*def* **by** *simp*
lemma $identity_E$-*infix-def*[*identity-defs*]:
  $x =_E y \equiv (|basic\text{-}identity_E,x,y|)$ **using** $basic\text{-}identity_E$-*infix-def* **.**
lemma $identity_\kappa$-*def*[*identity-defs*]:
  $op = \equiv \lambda x\ y.\ x =_E y \vee (|A!,x|)\ \&\ (|A!,y|)\ \&\ \Box(\forall\ F.\ \{|x,F|\} \equiv \{|y,F|\})$
  unfolding *identity*-κ-*def* $basic\text{-}identity_\kappa$-*def* *forall*-$\Pi_1$-*def* **by** *simp*
lemma $identity_\nu$-*def*[*identity-defs*]:
  $op = \equiv \lambda x\ y.\ (x^P) =_E (y^P) \vee (|A!,x^P|)\ \&\ (|A!,y^P|)\ \&\ \Box(\forall\ F.\ \{|x^P,F|\} \equiv \{|y^P,F|\})$
  unfolding *identity*-ν-*def* $identity_\kappa$-*def* **by** *simp*
lemma $identity_1$-*def*[*identity-defs*]:
  $op = \equiv \lambda F\ G.\ \Box(\forall\ x\ .\ \{|x^P,F|\} \equiv \{|x^P,G|\})$
  unfolding *identity*-Π_1-*def* $basic\text{-}identity_1$-*def* *forall*-ν-*def* **by** *simp*
lemma $identity_2$-*def*[*identity-defs*]:
  $op = \equiv \lambda F\ G.\ \forall\ x.\ (\boldsymbol{\lambda} y.\ (|F,x^P,y^P|)) = (\boldsymbol{\lambda} y.\ (|G,x^P,y^P|))$
        $\&\ (\boldsymbol{\lambda} y.\ (|F,y^P,x^P|)) = (\boldsymbol{\lambda} y.\ (|G,y^P,x^P|))$
  unfolding *identity*-Π_2-*def* *identity*-Π_1-*def* $basic\text{-}identity_2$-*def* *forall*-ν-*def* **by** *simp*
lemma $identity_3$-*def*[*identity-defs*]:
  $op = \equiv \lambda F\ G.\ \forall\ x\ y.\ (\boldsymbol{\lambda} z.\ (|F,z^P,x^P,y^P|)) = (\boldsymbol{\lambda} z.\ (|G,z^P,x^P,y^P|))$
        $\&\ (\boldsymbol{\lambda} z.\ (|F,x^P,z^P,y^P|)) = (\boldsymbol{\lambda} z.\ (|G,x^P,z^P,y^P|))$
        $\&\ (\boldsymbol{\lambda} z.\ (|F,x^P,y^P,z^P|)) = (\boldsymbol{\lambda} z.\ (|G,x^P,y^P,z^P|))$
  unfolding *identity*-Π_3-*def* *identity*-Π_1-*def* $basic\text{-}identity_3$-*def* *forall*-ν-*def* **by** *simp*
lemma $identity_o$-*def*[*identity-defs*]: $op = \equiv \lambda F\ G.\ (\boldsymbol{\lambda} y.\ F) = (\boldsymbol{\lambda} y.\ G)$
  unfolding *identity*-o-*def* *identity*-Π_1-*def* $basic\text{-}identity_0$-*def* **by** *simp*

A.7. The Axioms of Principia Metaphysica

Remark. *The axioms of PM can now be derived from the Semantics and the meta-logic.*

locale *Axioms*
begin
interpretation *MetaSolver* **.**
interpretation *Semantics* **.**
named-theorems *axiom*

Remark. *The special syntax* [[-]] *is introduced for axioms. This allows to formulate special rules resembling the concepts of closures in PM. To simplify the instantiation of axioms later, special attributes are introduced to automatically resolve the special axiom syntax. Necessitation averse axioms are stated with the syntax for actual validity* [-].

definition *axiom* :: o⇒*bool* ([[-]]) **where** *axiom* ≡ λ φ . ∀ v . [φ *in* v]

method *axiom-meta-solver* = ((((*unfold axiom-def*)?, *rule allI*) | (*unfold actual-validity-def*)?), *meta-solver*,
(*simp* | (*auto*; *fail*))?)

A.7.1. Closures

lemma *axiom-instance*[*axiom*]: [[φ]] ⟹ [φ *in* v]
unfolding *axiom-def* **by** *simp*
lemma *closures-universal*[*axiom*]: (⋀x.[[φ x]]) ⟹ [[∀ x. φ x]]
by *axiom-meta-solver*
lemma *closures-actualization*[*axiom*]: [[φ]] ⟹ [[𝓐 φ]]
by *axiom-meta-solver*
lemma *closures-necessitation*[*axiom*]: [[φ]] ⟹ [[□ φ]]
by *axiom-meta-solver*
lemma *necessitation-averse-axiom-instance*[*axiom*]: [φ] ⟹ [φ *in dw*]
by *axiom-meta-solver*
lemma *necessitation-averse-closures-universal*[*axiom*]: (⋀x.[φ x]) ⟹ [∀ x. φ x]
by *axiom-meta-solver*

```
attribute-setup axiom-instance = ‹‹
  Scan.succeed (Thm.rule-attribute []
    (fn - => fn thm => thm RS @{thm axiom-instance}))
››

attribute-setup necessitation-averse-axiom-instance = ‹‹
  Scan.succeed (Thm.rule-attribute []
    (fn - => fn thm => thm RS @{thm necessitation-averse-axiom-instance}))
››

attribute-setup axiom-necessitation = ‹‹
  Scan.succeed (Thm.rule-attribute []
    (fn - => fn thm => thm RS @{thm closures-necessitation}))
››

attribute-setup axiom-actualization = ‹‹
  Scan.succeed (Thm.rule-attribute []
    (fn - => fn thm => thm RS @{thm closures-actualization}))
››

attribute-setup axiom-universal = ‹‹
  Scan.succeed (Thm.rule-attribute []
    (fn - => fn thm => thm RS @{thm closures-universal}))
››
```

A.7.2. Axioms for Negations and Conditionals

lemma *pl-1*[*axiom*]:
  $[[\varphi \to (\psi \to \varphi)]]$
  by *axiom-meta-solver*
lemma *pl-2*[*axiom*]:
  $[[(\varphi \to (\psi \to \chi)) \to ((\varphi \to \psi) \to (\varphi \to \chi))]]$
  by *axiom-meta-solver*
lemma *pl-3*[*axiom*]:
  $[[(\neg\varphi \to \neg\psi) \to ((\neg\varphi \to \psi) \to \varphi)]]$
  by *axiom-meta-solver*

A.7.3. Axioms of Identity

lemma *l-identity*[*axiom*]:
  $[[\alpha = \beta \to (\varphi\ \alpha \to \varphi\ \beta)]]$
  using *l-identity* **apply** − **by** *axiom-meta-solver*

A.7.4. Axioms of Quantification

Remark. *The axioms of quantification differ from the axioms in Principia Metaphysica. The differences can be justified, though.*

- *Axiom cqt-2 is omitted, as the embedding does not distinguish between terms and variables for relations. Instead it is combined with cqt-1, in which the corresponding condition is omitted, and with cqt-5 in its modified form cqt-5-mod.*
- *Note that the all quantifier for individuals only ranges over type ν, which is always a denoting term and not a definite description in the embedding.*
- *The case of definite descriptions is handled separately in axiom cqt-1-κ: If a formula involving an object of type κ holds for all denoting terms ($\forall\ \alpha.\ \varphi\ (\alpha^P)$) then the formula holds for an individual term $\varphi\ \alpha$, if α denotes, i.e. $\exists\ \beta\ .\ (\beta^P) = \alpha$.*
- *Although axiom cqt-5 can be stated without modification, it is not a suitable formulation for the embedding. Instead the seemingly stronger version cqt-5-mod is stated as well. On a closer look, though, cqt-5-mod immediately follows from the original cqt-5 together with the omitted cqt-2.*

lemma *cqt-1*[*axiom*]:
  $[[(\forall\ \alpha.\ \varphi\ \alpha) \to \varphi\ \alpha]]$
  by *axiom-meta-solver*
lemma *cqt-1-κ*[*axiom*]:
  $[[(\forall\ \alpha.\ \varphi\ (\alpha^P)) \to ((\exists\ \beta\ .\ (\beta^P) = \alpha) \to \varphi\ \alpha)]]$
  proof −
    {
      fix v
      assume *1*: $[(\forall\ \alpha.\ \varphi\ (\alpha^P))\ in\ v]$
      assume $[(\exists\ \beta\ .\ (\beta^P) = \alpha)\ in\ v]$
      then obtain β **where** *2*:
        $[(\beta^P) = \alpha\ in\ v]$ **by** (*rule ExERule*)
      hence $[\varphi\ (\beta^P)\ in\ v]$ **using** *1 AllE* **by** *fast*
      hence $[\varphi\ \alpha\ in\ v]$
        using *l-identity*[**where** φ=φ, *axiom-instance*]
        ImplS 2 **by** *simp*
    }
    thus $[[(\forall\ \alpha.\ \varphi\ (\alpha^P)) \to ((\exists\ \beta\ .\ (\beta^P) = \alpha) \to \varphi\ \alpha)]]$
      unfolding *axiom-def* **using** *ImplI* **by** *blast*

qed
lemma *cqt-3*[*axiom*]:
[[($\forall\,\alpha.\ \varphi\ \alpha \rightarrow \psi\ \alpha$) $\rightarrow$ (($\forall\,\alpha.\ \varphi\ \alpha$) $\rightarrow$ ($\forall\,\alpha.\ \psi\ \alpha$))]]
by *axiom-meta-solver*
lemma *cqt-4*[*axiom*]:
[[$\varphi \rightarrow (\forall\,\alpha.\ \varphi)$]]
by *axiom-meta-solver*

inductive *SimpleExOrEnc*
where *SimpleExOrEnc* ($\lambda\ x$. ⦇F,x⦈)
| *SimpleExOrEnc* ($\lambda\ x$. ⦇F,x,y⦈)
| *SimpleExOrEnc* ($\lambda\ x$. ⦇F,y,x⦈)
| *SimpleExOrEnc* ($\lambda\ x$. ⦇F,x,y,z⦈)
| *SimpleExOrEnc* ($\lambda\ x$. ⦇F,y,x,z⦈)
| *SimpleExOrEnc* ($\lambda\ x$. ⦇F,y,z,x⦈)
| *SimpleExOrEnc* ($\lambda\ x$. ⦃x,F⦄)

lemma *cqt-5*[*axiom*]:
assumes *SimpleExOrEnc* ψ
shows [[(ψ (ιx . φ x)) $\rightarrow$ ($\exists$ α. (α^P) = (ιx . φ x))]]
proof −
have $\forall$ w . ([(ψ (ιx . φ x)) *in* w] $\longrightarrow$ ($\exists$ o_1 . *Some* o_1 = d_κ (ιx . φ x)))
using *assms* **apply** *induct* **by** (*meta-solver*;*metis*)+
thus *?thesis*
apply − **unfolding** *identity-κ-def*
apply *axiom-meta-solver*
using *d_κ-proper* **by** *auto*
qed

lemma *cqt-5-mod*[*axiom*]:
assumes *SimpleExOrEnc* ψ
shows [[ψ τ $\rightarrow$ ($\exists$ α . (α^P) = τ)]]
proof −
have $\forall$ w . ([(ψ τ) *in* w] $\longrightarrow$ ($\exists$ o_1 . *Some* o_1 = d_κ τ))
using *assms* **apply** *induct* **by** (*meta-solver*;*metis*)+
thus *?thesis*
apply − **unfolding** *identity-κ-def*
apply *axiom-meta-solver*
using *d_κ-proper* **by** *auto*
qed

A.7.5. Axioms of Actuality

Remark. *The necessitation averse axiom of actuality is stated to be actually true; for the statement as a proper axiom (for which necessitation would be allowed) nitpick can find a counter-model as desired.*

lemma *logic-actual*[*axiom*]: [($\boldsymbol{\mathcal{A}}\varphi$) $\equiv$ φ]
by *axiom-meta-solver*
lemma [[($\boldsymbol{\mathcal{A}}\varphi$) $\equiv$ φ]]
nitpick[*user-axioms*, *expect* = *genuine*, *card* = *1*, *card i* = *2*]
oops — Counter-model by nitpick

lemma *logic-actual-nec-1*[*axiom*]:
[[$\boldsymbol{\mathcal{A}}\neg\varphi \equiv \neg\boldsymbol{\mathcal{A}}\varphi$]]
by *axiom-meta-solver*
lemma *logic-actual-nec-2*[*axiom*]:

$[[(\boldsymbol{\mathcal{A}}(\varphi \rightarrow \psi)) \equiv (\boldsymbol{\mathcal{A}}\varphi \rightarrow \boldsymbol{\mathcal{A}}\psi)]]$
by *axiom-meta-solver*
lemma *logic-actual-nec-3*[*axiom*]:
$[[\boldsymbol{\mathcal{A}}(\forall\, \alpha.\ \varphi\ \alpha) \equiv (\forall\, \alpha.\ \boldsymbol{\mathcal{A}}(\varphi\ \alpha))]]$
by *axiom-meta-solver*
lemma *logic-actual-nec-4*[*axiom*]:
$[[\boldsymbol{\mathcal{A}}\varphi \equiv \boldsymbol{\mathcal{A}}\boldsymbol{\mathcal{A}}\varphi]]$
by *axiom-meta-solver*

A.7.6. Axioms of Necessity

lemma *qml-1*[*axiom*]:
$[[\Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi)]]$
by *axiom-meta-solver*
lemma *qml-2*[*axiom*]:
$[[\Box\varphi \rightarrow \varphi]]$
by *axiom-meta-solver*
lemma *qml-3*[*axiom*]:
$[[\Diamond\varphi \rightarrow \Box\Diamond\varphi]]$
by *axiom-meta-solver*
lemma *qml-4*[*axiom*]:
$[[\Diamond(\exists\, x.\ (\!|E!,x^P|\!) \ \&\ \Diamond\neg(\!|E!,x^P|\!)) \ \&\ \Diamond\neg(\exists\, x.\ (\!|E!,x^P|\!) \ \&\ \Diamond\neg(\!|E!,x^P|\!))]]$
unfolding *axiom-def*
using *PossiblyContingentObjectExistsAxiom*
PossiblyNoContingentObjectExistsAxiom
apply (*simp add*: *meta-defs meta-aux conn-defs forall-ν-def*
split: *ν.split υ.split*)
by (*metis νυ-ων-is-ωυ υ.distinct*(*1*) *υ.inject*(*1*))

A.7.7. Axioms of Necessity and Actuality

lemma *qml-act-1*[*axiom*]:
$[[\boldsymbol{\mathcal{A}}\varphi \rightarrow \Box\boldsymbol{\mathcal{A}}\varphi]]$
by *axiom-meta-solver*
lemma *qml-act-2*[*axiom*]:
$[[\Box\varphi \equiv \boldsymbol{\mathcal{A}}(\Box\varphi)]]$
by *axiom-meta-solver*

A.7.8. Axioms of Descriptions

lemma *descriptions*[*axiom*]:
$[[x^P = (\boldsymbol{\iota}x.\ \varphi\ x) \equiv (\forall\, z.(\boldsymbol{\mathcal{A}}(\varphi\ z) \equiv z = x))]]$
unfolding *axiom-def*
proof (*rule allI*, *rule EquivI*; *rule*)
fix *v*
assume $[x^P = (\boldsymbol{\iota}x.\ \varphi\ x)\ in\ v]$
moreover hence *1*:
$\exists\, o_1\ o_2.\ Some\ o_1 = d_\kappa\ (x^P) \wedge Some\ o_2 = d_\kappa\ (\boldsymbol{\iota}x.\ \varphi\ x) \wedge o_1 = o_2$
apply – **unfolding** *identity-κ-def* **by** *meta-solver*
then obtain $o_1\ o_2$ **where** *2*:
$Some\ o_1 = d_\kappa\ (x^P) \wedge Some\ o_2 = d_\kappa\ (\boldsymbol{\iota}x.\ \varphi\ x) \wedge o_1 = o_2$
by *auto*
hence *3*:
$(\exists\ x\ .((w_0 \models \varphi\ x) \wedge (\forall\, y.\ (w_0 \models \varphi\ y) \longrightarrow y = x)))$
$\wedge\ d_\kappa\ (\boldsymbol{\iota}x.\ \varphi\ x) = Some\ (THE\ x.\ (w_0 \models \varphi\ x))$
using *D3* **by** (*metis option.distinct*(*1*))
then obtain *X* **where** *4*:

$((w_0 \models \varphi\ X) \wedge (\forall y.\ (w_0 \models \varphi\ y) \longrightarrow y = X))$
by *auto*
moreover have $o_1 = (\mathit{THE}\ x.\ (w_0 \models \varphi\ x))$
using *2 3* **by** *auto*
ultimately have *5*: $X = o_1$
by (*metis* (*mono-tags*) *theI*)
have $\forall\ z\ .\ [\boldsymbol{\mathcal{A}}\varphi\ z\ \mathit{in}\ v] = [(z^P) = (x^P)\ \mathit{in}\ v]$
proof
fix z
have $[\boldsymbol{\mathcal{A}}\varphi\ z\ \mathit{in}\ v] \Longrightarrow [(z^P) = (x^P)\ \mathit{in}\ v]$
unfolding *identity-κ-def* **apply** *meta-solver*
using *4 5 2* d_κ*-proper* w_0*-def* **by** *auto*
moreover have $[(z^P) = (x^P)\ \mathit{in}\ v] \Longrightarrow [\boldsymbol{\mathcal{A}}\varphi\ z\ \mathit{in}\ v]$
unfolding *identity-κ-def* **apply** *meta-solver*
using *2 4 5*
by (*simp add*: d_κ*-proper* w_0*-def*)
ultimately show $[\boldsymbol{\mathcal{A}}\varphi\ z\ \mathit{in}\ v] = [(z^P) = (x^P)\ \mathit{in}\ v]$
by *auto*
qed
thus $[\forall z.\ \boldsymbol{\mathcal{A}}\varphi\ z \equiv (z) = (x)\ \mathit{in}\ v]$
unfolding *identity-ν-def*
by (*simp add*: *AllI EquivS*)
next
fix v
assume $[\forall z.\ \boldsymbol{\mathcal{A}}\varphi\ z \equiv (z) = (x)\ \mathit{in}\ v]$
hence $\bigwedge z.\ (dw \models \varphi\ z) = (\exists\, o_1\ o_2.\ \mathit{Some}\ o_1 = d_\kappa\ (z^P)$
$\wedge\ \mathit{Some}\ o_2 = d_\kappa\ (x^P) \wedge o_1 = o_2)$
apply − **unfolding** *identity-ν-def identity-κ-def* **by** *meta-solver*
hence $\forall\ z\ .\ (dw \models \varphi\ z) = (z = x)$
by (*simp add*: d_κ*-proper*)
moreover hence $x = (\mathit{THE}\ z\ .\ (dw \models \varphi\ z))$ **by** *simp*
ultimately have $x^P = (\boldsymbol{\iota} x.\ \varphi\ x)$
using *D3* d_κ*-inject* d_κ*-proper* w_0*-def* **by** *presburger*
thus $[x^P = (\boldsymbol{\iota} x.\ \varphi\ x)\ \mathit{in}\ v]$
using *EqκS* **unfolding** *identity-κ-def* **by** (*metis* d_κ*-proper*)
qed

A.7.9. Axioms for Complex Relation Terms

lemma *lambda-predicates-1* [*axiom*]:
$(\boldsymbol{\lambda}\ x\ .\ \varphi\ x) = (\boldsymbol{\lambda}\ y\ .\ \varphi\ y)$ **..**

lemma *lambda-predicates-2-1* [*axiom*]:
assumes *IsProperInX* φ
shows $[[(\!|\boldsymbol{\lambda}\ x\ .\ \varphi\ (x^P),\ x^P|\!) \equiv \varphi\ (x^P)]]$
apply *axiom-meta-solver*
using *D5-1* [*OF assms*] d_κ*-proper* propex_1
by *metis*

lemma *lambda-predicates-2-2* [*axiom*]:
assumes *IsProperInXY* φ
shows $[[(\!|(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ \varphi\ (x^P)\ (y^P))),\ x^P,\ y^P|\!) \equiv \varphi\ (x^P)\ (y^P)]]$
apply *axiom-meta-solver*
using *D5-2* [*OF assms*] d_κ*-proper* propex_2
by *metis*

lemma *lambda-predicates-2-3* [*axiom*]:
assumes *IsProperInXYZ* φ

shows $[[(\!|(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \varphi\ (x^P)\ (y^P)\ (z^P))),x^P,y^P,z^P|\!) \equiv \varphi\ (x^P)\ (y^P)\ (z^P)]]$
proof $-$
have $[[(\!|(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \varphi\ (x^P)\ (y^P)\ (z^P))),x^P,y^P,z^P|\!) \rightarrow \varphi\ (x^P)\ (y^P)\ (z^P)]]$
apply *axiom-meta-solver* **using** *D5-3*[*OF assms*] **by** *auto*
moreover have
$[[\varphi\ (x^P)\ (y^P)\ (z^P) \rightarrow (\!|(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \varphi\ (x^P)\ (y^P)\ (z^P))),x^P,y^P,z^P|\!)]]$
apply *axiom-meta-solver*
using *D5-3*[*OF assms*] d_κ*-proper* $propex_3$
by (*metis* (*no-types*, *lifting*))
ultimately show *?thesis* **unfolding** *axiom-def equiv-def ConjS* **by** *blast*
qed

lemma *lambda-predicates-3-0*[*axiom*]:
$[[(\boldsymbol{\lambda}^0\ \varphi) = \varphi]]$
unfolding *identity-defs*
apply *axiom-meta-solver*
by (*simp add*: *meta-defs meta-aux*)

lemma *lambda-predicates-3-1*[*axiom*]:
$[[(\boldsymbol{\lambda}\ x\ .\ (\!|F,\ x^P|\!)) = F]]$
unfolding *axiom-def*
apply (*rule allI*)
unfolding *identity*-Π_1*-def* **apply** (*rule* Eq_1I)
using *D4-1* d_1*-inject* **by** *simp*

lemma *lambda-predicates-3-2*[*axiom*]:
$[[(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ (\!|F,\ x^P,\ y^P|\!))) = F]]$
unfolding *axiom-def*
apply (*rule allI*)
unfolding *identity*-Π_2*-def* **apply** (*rule* Eq_2I)
using *D4-2* d_2*-inject* **by** *simp*

lemma *lambda-predicates-3-3*[*axiom*]:
$[[(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ (\!|F,\ x^P,\ y^P,\ z^P|\!))) = F]]$
unfolding *axiom-def*
apply (*rule allI*)
unfolding *identity*-Π_3*-def* **apply** (*rule* Eq_3I)
using *D4-3* d_3*-inject* **by** *simp*

lemma *lambda-predicates-4-0*[*axiom*]:
assumes $\bigwedge x.[(\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x))\ in\ v]$
shows $[[(\boldsymbol{\lambda}^0\ (\chi\ (\iota x.\ \varphi\ x)) = \boldsymbol{\lambda}^0\ (\chi\ (\iota x.\ \psi\ x)))]]$
unfolding *axiom-def identity*-o*-def* **apply** $-$ **apply** (*rule allI*; *rule* Eq_0I)
using *TheEqI*[*OF assms*[*THEN ActualE*, *THEN EquivE*]] **by** *auto*

lemma *lambda-predicates-4-1*[*axiom*]:
assumes $\bigwedge x.[(\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x))\ in\ v]$
shows $[[((\boldsymbol{\lambda}\ x\ .\ \chi\ (\iota x.\ \varphi\ x)\ x) = (\boldsymbol{\lambda}\ x\ .\ \chi\ (\iota x.\ \psi\ x)\ x))]]$
unfolding *axiom-def identity*-Π_1*-def* **apply** $-$ **apply** (*rule allI*; *rule* Eq_1I)
using *TheEqI*[*OF assms*[*THEN ActualE*, *THEN EquivE*]] **by** *auto*

lemma *lambda-predicates-4-2*[*axiom*]:
assumes $\bigwedge x.[(\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x))\ in\ v]$
shows $[[((\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ \chi\ (\iota x.\ \varphi\ x)\ x\ y)) = (\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ \chi\ (\iota x.\ \psi\ x)\ x\ y)))]]$
unfolding *axiom-def identity*-Π_2*-def* **apply** $-$ **apply** (*rule allI*; *rule* Eq_2I)
using *TheEqI*[*OF assms*[*THEN ActualE*, *THEN EquivE*]] **by** *auto*

lemma *lambda-predicates-4-3*[*axiom*]:

assumes $\bigwedge x.[(\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x))\ in\ v]$
shows $[[(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \chi\ (\boldsymbol{\iota}x.\ \varphi\ x)\ x\ y\ z)) = (\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \chi\ (\boldsymbol{\iota}x.\ \psi\ x)\ x\ y\ z))]]$
unfolding *axiom-def identity-*Π_3*-def* **apply** − **apply** (*rule allI*; *rule* Eq_3I)
using *TheEqI*[*OF assms*[*THEN ActualE*, *THEN EquivE*]] **by** *auto*

A.7.10. Axioms of Encoding

lemma *encoding*[*axiom*]:
$[[\{\!|x,F|\!\} \to \Box\{\!|x,F|\!\}]]$
by *axiom-meta-solver*
lemma *nocoder*[*axiom*]:
$[[(\!|O!,x|\!) \to \neg(\exists\ F\ .\ \{\!|x,F|\!\})]]$
unfolding *axiom-def*
apply (*rule allI*, *rule ImplI*, *subst* (*asm*) *OrdS*)
apply *meta-solver* **unfolding** *en-def*
by (*metis* ν*.simps*(*5*) *mem-Collect-eq option.sel*)
lemma *A-objects*[*axiom*]:
$[[\exists\, x.\ (\!|A!,x^P|\!)\ \&\ (\forall\ \ F\ .\ (\{\!|x^P,F|\!\} \equiv \varphi\ F))]]$
unfolding *axiom-def*
proof (*rule allI*, *rule ExIRule*)
fix v
let $?x = \alpha\nu\ \{\ F\ .\ [\varphi\ F\ in\ v]\}$
have $[(\!|A!,?x^P|\!)\ in\ v]$ **by** (*simp add*: *AbsS* d_κ*-proper*)
moreover have $[(\forall\, F.\ \{\!|?x^P,F|\!\} \equiv \varphi\ F)\ in\ v]$
apply *meta-solver* **unfolding** *en-def*
using d_1*.rep-eq* d_κ*-def* d_κ*-proper eval*Π_1*-inverse* **by** *auto*
ultimately show $[(\!|A!,?x^P|\!)\ \&\ (\forall\, F.\ \{\!|?x^P,F|\!\} \equiv \varphi\ F)\ in\ v]$
by (*simp only*: *ConjS*)
qed

end

A.8. Definitions

Various definitions needed throughout PLM.

A.8.1. Property Negations

consts *propnot* :: $'a \Rightarrow 'a$ (-$^-$ [*90*] *90*)
overloading $propnot_0 \equiv propnot :: \Pi_0 \Rightarrow \Pi_0$
$propnot_1 \equiv propnot :: \Pi_1 \Rightarrow \Pi_1$
$propnot_2 \equiv propnot :: \Pi_2 \Rightarrow \Pi_2$
$propnot_3 \equiv propnot :: \Pi_3 \Rightarrow \Pi_3$
begin
definition $propnot_0 :: \Pi_0 \Rightarrow \Pi_0$ **where**
$propnot_0 \equiv \lambda\ p\ .\ \boldsymbol{\lambda}^0\ (\neg p)$
definition $propnot_1$ **where**
$propnot_1 \equiv \lambda\ F\ .\ \boldsymbol{\lambda}\ x\ .\ \neg(\!|F,\ x^P|\!)$
definition $propnot_2$ **where**
$propnot_2 \equiv \lambda\ F\ .\ \boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ \neg(\!|F,\ x^P,\ y^P|\!))$
definition $propnot_3$ **where**
$propnot_3 \equiv \lambda\ F\ .\ \boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \neg(\!|F,\ x^P,\ y^P,\ z^P|\!))$
end

named-theorems *propnot-defs*

declare *propnot*$_0$*-def*[*propnot-defs*] *propnot*$_1$*-def*[*propnot-defs*]
propnot$_2$*-def*[*propnot-defs*] *propnot*$_3$*-def*[*propnot-defs*]

A.8.2. Noncontingent and Contingent Relations

consts *Necessary* :: $'a \Rightarrow o$
overloading $Necessary_0 \equiv Necessary :: \Pi_0 \Rightarrow o$
$Necessary_1 \equiv Necessary :: \Pi_1 \Rightarrow o$
$Necessary_2 \equiv Necessary :: \Pi_2 \Rightarrow o$
$Necessary_3 \equiv Necessary :: \Pi_3 \Rightarrow o$
begin
definition $Necessary_0$ **where**
$Necessary_0 \equiv \lambda\ p\ .\ \Box p$
definition $Necessary_1 :: \Pi_1 \Rightarrow o$ **where**
$Necessary_1 \equiv \lambda\ F\ .\ \Box(\forall\ x\ .\ (|F,x^P|))$
definition $Necessary_2$ **where**
$Necessary_2 \equiv \lambda\ F\ .\ \Box(\forall\ x\ y\ .\ (|F,x^P,y^P|))$
definition $Necessary_3$ **where**
$Necessary_3 \equiv \lambda\ F\ .\ \Box(\forall\ x\ y\ z\ .\ (|F,x^P,y^P,z^P|))$
end

named-theorems *Necessary-defs*
declare *Necessary*$_0$*-def*[*Necessary-defs*] *Necessary*$_1$*-def*[*Necessary-defs*]
Necessary$_2$*-def*[*Necessary-defs*] *Necessary*$_3$*-def*[*Necessary-defs*]

consts *Impossible* :: $'a \Rightarrow o$
overloading $Impossible_0 \equiv Impossible :: \Pi_0 \Rightarrow o$
$Impossible_1 \equiv Impossible :: \Pi_1 \Rightarrow o$
$Impossible_2 \equiv Impossible :: \Pi_2 \Rightarrow o$
$Impossible_3 \equiv Impossible :: \Pi_3 \Rightarrow o$
begin
definition $Impossible_0$ **where**
$Impossible_0 \equiv \lambda\ p\ .\ \Box\neg p$
definition $Impossible_1$ **where**
$Impossible_1 \equiv \lambda\ F\ .\ \Box(\forall\ x.\ \neg(|F,x^P|))$
definition $Impossible_2$ **where**
$Impossible_2 \equiv \lambda\ F\ .\ \Box(\forall\ x\ y.\ \neg(|F,x^P,y^P|))$
definition $Impossible_3$ **where**
$Impossible_3 \equiv \lambda\ F\ .\ \Box(\forall\ x\ y\ z.\ \neg(|F,x^P,y^P,z^P|))$
end

named-theorems *Impossible-defs*
declare *Impossible*$_0$*-def*[*Impossible-defs*] *Impossible*$_1$*-def*[*Impossible-defs*]
Impossible$_2$*-def*[*Impossible-defs*] *Impossible*$_3$*-def*[*Impossible-defs*]

definition *NonContingent* **where**
$NonContingent \equiv \lambda\ F\ .\ (Necessary\ F) \lor (Impossible\ F)$
definition *Contingent* **where**
$Contingent \equiv \lambda\ F\ .\ \neg(Necessary\ F \lor Impossible\ F)$

definition *ContingentlyTrue* :: $o \Rightarrow o$ **where**
$ContingentlyTrue \equiv \lambda\ p\ .\ p\ \&\ \Diamond\neg p$
definition *ContingentlyFalse* :: $o \Rightarrow o$ **where**
$ContingentlyFalse \equiv \lambda\ p\ .\ \neg p\ \&\ \Diamond p$

definition *WeaklyContingent* **where**
$WeaklyContingent \equiv \lambda\ F\ .\ Contingent\ F\ \&\ (\forall\ x.\ \Diamond(|F,x^P|) \rightarrow \Box(|F,x^P|))$

A.8.3. Null and Universal Objects

definition *Null* :: $\kappa \Rightarrow o$ **where**
  $Null \equiv \lambda\ x\ .\ (\!|A!,x|\!)\ \&\ \neg(\exists\ F\ .\ \{\!|x,\ F|\!\})$
definition *Universal* :: $\kappa \Rightarrow o$ **where**
  $Universal \equiv \lambda\ x\ .\ (\!|A!,x|\!)\ \&\ (\forall\ F\ .\ \{\!|x,\ F|\!\})$

definition *NullObject* :: $\kappa\ (\mathbf{a}_\emptyset)$ **where**
  $NullObject \equiv (\iota x\ .\ Null\ (x^P))$
definition *UniversalObject* :: $\kappa\ (\mathbf{a}_V)$ **where**
  $UniversalObject \equiv (\iota x\ .\ Universal\ (x^P))$

A.8.4. Propositional Properties

definition *Propositional* **where**
  $Propositional\ F \equiv \exists\ \ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)$

A.8.5. Indiscriminate Properties

definition *Indiscriminate* :: $\Pi_1 \Rightarrow o$ **where**
  $Indiscriminate \equiv \lambda\ F\ .\ \Box((\exists\ x\ .\ (\!|F,x^P|\!)) \rightarrow (\forall\ x\ .\ (\!|F,x^P|\!)))$

A.8.6. Miscellaneous

definition *not-identical*$_E$ :: $\kappa \Rightarrow \kappa \Rightarrow o$ (**infixl** $\neq_E$ *63*)
  where $not\text{-}identical_E \equiv \lambda\ x\ y\ .\ (\!|(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ x^P =_E y^P))^-,\ x,\ y|\!)$

A.9. The Deductive System PLM

declare *meta-defs*[*no-atp*] *meta-aux*[*no-atp*]

locale *PLM* = *Axioms*
begin

A.9.1. Automatic Solver

named-theorems *PLM*
named-theorems *PLM-intro*
named-theorems *PLM-elim*
named-theorems *PLM-dest*
named-theorems *PLM-subst*

method *PLM-solver* **declares** *PLM-intro PLM-elim PLM-subst PLM-dest PLM*
  = ((*assumption* | (*match axiom* **in** *A*: $[[\varphi]]$ **for** $\varphi \Rightarrow$ ‹*fact A*[*axiom-instance*]›)
    | *fact PLM* | *rule PLM-intro* | *subst PLM-subst* | *subst* (*asm*) *PLM-subst*
    | *fastforce* | *safe* | *drule PLM-dest* | *erule PLM-elim*); (*PLM-solver*)?)

A.9.2. Modus Ponens

lemma *modus-ponens*[*PLM*]:
  $[\![[\varphi\ in\ v];\ [\varphi \rightarrow \psi\ in\ v]]\!] \Longrightarrow [\psi\ in\ v]$
  by (*simp add*: *Semantics.T5*)

A.9.3. Axioms

interpretation *Axioms* **.**
declare *axiom*[*PLM*]
declare *conn-defs*[*PLM*]

A.9.4. (Modally Strict) Proofs and Derivations

lemma *vdash-properties-6*[*no-atp*]:
$[\![[\varphi\ in\ v]; [\varphi \rightarrow \psi\ in\ v]]\!] \Longrightarrow [\psi\ in\ v]$
using *modus-ponens* **.**
lemma *vdash-properties-9*[*PLM*]:
$[\varphi\ in\ v] \Longrightarrow [\psi \rightarrow \varphi\ in\ v]$
using *modus-ponens pl-1*[*axiom-instance*] **by** *blast*
lemma *vdash-properties-10*[*PLM*]:
$[\varphi \rightarrow \psi\ in\ v] \Longrightarrow ([\varphi\ in\ v] \Longrightarrow [\psi\ in\ v])$
using *vdash-properties-6* **.**

attribute-setup *deduction* = ⟪
Scan.succeed (*Thm.rule-attribute* []
(*fn* - => *fn thm* => *thm RS* @{*thm vdash-properties-10*}))
⟫

A.9.5. GEN and RN

lemma *rule-gen*[*PLM*]:
$[\![\bigwedge\alpha\ .\ [\varphi\ \alpha\ in\ v]]\!] \Longrightarrow [\forall\,\alpha\ .\ \varphi\ \alpha\ in\ v]$
by (*simp add*: *Semantics.T8*)

lemma *RN-2*[*PLM*]:
$(\bigwedge\ v\ .\ [\psi\ in\ v] \Longrightarrow [\varphi\ in\ v]) \Longrightarrow ([\Box\psi\ in\ v] \Longrightarrow [\Box\varphi\ in\ v])$
by (*simp add*: *Semantics.T6*)

lemma *RN*[*PLM*]:
$(\bigwedge\ v\ .\ [\varphi\ in\ v]) \Longrightarrow [\Box\varphi\ in\ v]$
using *qml-3*[*axiom-necessitation*, *axiom-instance*] *RN-2* **by** *blast*

A.9.6. Negations and Conditionals

lemma *if-p-then-p*[*PLM*]:
$[\varphi \rightarrow \varphi\ in\ v]$
using *pl-1 pl-2 vdash-properties-10 axiom-instance* **by** *blast*

lemma *deduction-theorem*[*PLM*,*PLM-intro*]:
$[\![[\varphi\ in\ v] \Longrightarrow [\psi\ in\ v]]\!] \Longrightarrow [\varphi \rightarrow \psi\ in\ v]$
by (*simp add*: *Semantics.T5*)
lemmas *CP* = *deduction-theorem*

lemma *ded-thm-cor-3*[*PLM*]:
$[\![[\varphi \rightarrow \psi\ in\ v]; [\psi \rightarrow \chi\ in\ v]]\!] \Longrightarrow [\varphi \rightarrow \chi\ in\ v]$
by (*meson pl-2 vdash-properties-10 vdash-properties-9 axiom-instance*)
lemma *ded-thm-cor-4*[*PLM*]:
$[\![[\varphi \rightarrow (\psi \rightarrow \chi)\ in\ v]; [\psi\ in\ v]]\!] \Longrightarrow [\varphi \rightarrow \chi\ in\ v]$
by (*meson pl-2 vdash-properties-10 vdash-properties-9 axiom-instance*)

lemma *useful-tautologies-1*[*PLM*]:
$[\neg\neg\varphi \rightarrow \varphi\ in\ v]$

by (*meson pl-1 pl-3 ded-thm-cor-3 ded-thm-cor-4 axiom-instance*)
lemma *useful-tautologies-2*[*PLM*]:
$[\varphi \rightarrow \neg\neg\varphi \ in \ v]$
by (*meson pl-1 pl-3 ded-thm-cor-3 useful-tautologies-1 vdash-properties-10 axiom-instance*)
lemma *useful-tautologies-3*[*PLM*]:
$[\neg\varphi \rightarrow (\varphi \rightarrow \psi) \ in \ v]$
by (*meson pl-1 pl-2 pl-3 ded-thm-cor-3 ded-thm-cor-4 axiom-instance*)
lemma *useful-tautologies-4*[*PLM*]:
$[(\neg\psi \rightarrow \neg\varphi) \rightarrow (\varphi \rightarrow \psi) \ in \ v]$
by (*meson pl-1 pl-2 pl-3 ded-thm-cor-3 ded-thm-cor-4 axiom-instance*)
lemma *useful-tautologies-5*[*PLM*]:
$[(\varphi \rightarrow \psi) \rightarrow (\neg\psi \rightarrow \neg\varphi) \ in \ v]$
by (*metis CP useful-tautologies-4 vdash-properties-10*)
lemma *useful-tautologies-6*[*PLM*]:
$[(\varphi \rightarrow \neg\psi) \rightarrow (\psi \rightarrow \neg\varphi) \ in \ v]$
by (*metis CP useful-tautologies-4 vdash-properties-10*)
lemma *useful-tautologies-7*[*PLM*]:
$[(\neg\varphi \rightarrow \psi) \rightarrow (\neg\psi \rightarrow \varphi) \ in \ v]$
using *ded-thm-cor-3 useful-tautologies-4 useful-tautologies-5 useful-tautologies-6* **by** *blast*
lemma *useful-tautologies-8*[*PLM*]:
$[\varphi \rightarrow (\neg\psi \rightarrow \neg(\varphi \rightarrow \psi)) \ in \ v]$
by (*meson ded-thm-cor-3 CP useful-tautologies-5*)
lemma *useful-tautologies-9*[*PLM*]:
$[(\varphi \rightarrow \psi) \rightarrow ((\neg\varphi \rightarrow \psi) \rightarrow \psi) \ in \ v]$
by (*metis CP useful-tautologies-4 vdash-properties-10*)
lemma *useful-tautologies-10*[*PLM*]:
$[(\varphi \rightarrow \neg\psi) \rightarrow ((\varphi \rightarrow \psi) \rightarrow \neg\varphi) \ in \ v]$
by (*metis ded-thm-cor-3 CP useful-tautologies-6*)

lemma *modus-tollens-1*[*PLM*]:
$[\![[\varphi \rightarrow \psi \ in \ v]; [\neg\psi \ in \ v]]\!] \Longrightarrow [\neg\varphi \ in \ v]$
by (*metis ded-thm-cor-3 ded-thm-cor-4 useful-tautologies-3 useful-tautologies-7 vdash-properties-10*)
lemma *modus-tollens-2*[*PLM*]:
$[\![[\varphi \rightarrow \neg\psi \ in \ v]; [\psi \ in \ v]]\!] \Longrightarrow [\neg\varphi \ in \ v]$
using *modus-tollens-1 useful-tautologies-2 vdash-properties-10* **by** *blast*

lemma *contraposition-1*[*PLM*]:
$[\varphi \rightarrow \psi \ in \ v] = [\neg\psi \rightarrow \neg\varphi \ in \ v]$
using *useful-tautologies-4 useful-tautologies-5 vdash-properties-10* **by** *blast*
lemma *contraposition-2*[*PLM*]:
$[\varphi \rightarrow \neg\psi \ in \ v] = [\psi \rightarrow \neg\varphi \ in \ v]$
using *contraposition-1 ded-thm-cor-3 useful-tautologies-1* **by** *blast*

lemma *reductio-aa-1*[*PLM*]:
$[\![[\neg\varphi \ in \ v] \Longrightarrow [\neg\psi \ in \ v]; [\neg\varphi \ in \ v] \Longrightarrow [\psi \ in \ v]]\!] \Longrightarrow [\varphi \ in \ v]$
using *CP modus-tollens-2 useful-tautologies-1 vdash-properties-10* **by** *blast*
lemma *reductio-aa-2*[*PLM*]:
$[\![[\varphi \ in \ v] \Longrightarrow [\neg\psi \ in \ v]; [\varphi \ in \ v] \Longrightarrow [\psi \ in \ v]]\!] \Longrightarrow [\neg\varphi \ in \ v]$
by (*meson contraposition-1 reductio-aa-1*)
lemma *reductio-aa-3*[*PLM*]:
$[\![[\neg\varphi \rightarrow \neg\psi \ in \ v]; [\neg\varphi \rightarrow \psi \ in \ v]]\!] \Longrightarrow [\varphi \ in \ v]$

using *reductio-aa-1 vdash-properties-10* **by** *blast*
lemma *reductio-aa-4*[*PLM*]:
$[\![[\varphi \to \neg\psi\ in\ v]; [\varphi \to \psi\ in\ v]]\!] \Longrightarrow [\neg\varphi\ in\ v]$
using *reductio-aa-2 vdash-properties-10* **by** *blast*

lemma *raa-cor-1*[*PLM*]:
$[\![[\varphi\ in\ v]; [\neg\psi\ in\ v] \Longrightarrow [\neg\varphi\ in\ v]]\!] \Longrightarrow ([\varphi\ in\ v] \Longrightarrow [\psi\ in\ v])$
using *reductio-aa-1 vdash-properties-9* **by** *blast*
lemma *raa-cor-2*[*PLM*]:
$[\![[\neg\varphi\ in\ v]; [\neg\psi\ in\ v] \Longrightarrow [\varphi\ in\ v]]\!] \Longrightarrow ([\neg\varphi\ in\ v] \Longrightarrow [\psi\ in\ v])$
using *reductio-aa-1 vdash-properties-9* **by** *blast*
lemma *raa-cor-3*[*PLM*]:
$[\![[\varphi\ in\ v]; [\neg\psi \to \neg\varphi\ in\ v]]\!] \Longrightarrow ([\varphi\ in\ v] \Longrightarrow [\psi\ in\ v])$
using *raa-cor-1 vdash-properties-10* **by** *blast*
lemma *raa-cor-4*[*PLM*]:
$[\![[\neg\varphi\ in\ v]; [\neg\psi \to \varphi\ in\ v]]\!] \Longrightarrow ([\neg\varphi\ in\ v] \Longrightarrow [\psi\ in\ v])$
using *raa-cor-2 vdash-properties-10* **by** *blast*

Remark. *The classical introduction and elimination rules are proven earlier than in PM. The statements proven so far are sufficient for the proofs and using these rules Isabelle can prove the tautologies automatically.*

lemma *intro-elim-1*[*PLM*]:
$[\![[\varphi\ in\ v]; [\psi\ in\ v]]\!] \Longrightarrow [\varphi\ \&\ \psi\ in\ v]$
unfolding *conj-def* **using** *ded-thm-cor-4 if-p-then-p modus-tollens-2* **by** *blast*
lemmas $\&I$ = *intro-elim-1*
lemma *intro-elim-2-a*[*PLM*]:
$[\varphi\ \&\ \psi\ in\ v] \Longrightarrow [\varphi\ in\ v]$
unfolding *conj-def* **using** *CP reductio-aa-1* **by** *blast*
lemma *intro-elim-2-b*[*PLM*]:
$[\varphi\ \&\ \psi\ in\ v] \Longrightarrow [\psi\ in\ v]$
unfolding *conj-def* **using** *pl-1 CP reductio-aa-1 axiom-instance* **by** *blast*
lemmas $\&E$ = *intro-elim-2-a intro-elim-2-b*
lemma *intro-elim-3-a*[*PLM*]:
$[\varphi\ in\ v] \Longrightarrow [\varphi \lor \psi\ in\ v]$
unfolding *disj-def* **using** *ded-thm-cor-4 useful-tautologies-3* **by** *blast*
lemma *intro-elim-3-b*[*PLM*]:
$[\psi\ in\ v] \Longrightarrow [\varphi \lor \psi\ in\ v]$
by (*simp only*: *disj-def vdash-properties-9*)
lemmas $\lor I$ = *intro-elim-3-a intro-elim-3-b*
lemma *intro-elim-4-a*[*PLM*]:
$[\![[\varphi \lor \psi\ in\ v]; [\varphi \to \chi\ in\ v]; [\psi \to \chi\ in\ v]]\!] \Longrightarrow [\chi\ in\ v]$
unfolding *disj-def* **by** (*meson reductio-aa-2 vdash-properties-10*)
lemma *intro-elim-4-b*[*PLM*]:
$[\![[\varphi \lor \psi\ in\ v]; [\neg\varphi\ in\ v]]\!] \Longrightarrow [\psi\ in\ v]$
unfolding *disj-def* **using** *vdash-properties-10* **by** *blast*
lemma *intro-elim-4-c*[*PLM*]:
$[\![[\varphi \lor \psi\ in\ v]; [\neg\psi\ in\ v]]\!] \Longrightarrow [\varphi\ in\ v]$
unfolding *disj-def* **using** *raa-cor-2 vdash-properties-10* **by** *blast*
lemma *intro-elim-4-d*[*PLM*]:
$[\![[\varphi \lor \psi\ in\ v]; [\varphi \to \chi\ in\ v]; [\psi \to \Theta\ in\ v]]\!] \Longrightarrow [\chi \lor \Theta\ in\ v]$
unfolding *disj-def* **using** *contraposition-1 ded-thm-cor-3* **by** *blast*
lemma *intro-elim-4-e*[*PLM*]:
$[\![[\varphi \lor \psi\ in\ v]; [\varphi \equiv \chi\ in\ v]; [\psi \equiv \Theta\ in\ v]]\!] \Longrightarrow [\chi \lor \Theta\ in\ v]$
unfolding *equiv-def* **using** $\&E(1)$ *intro-elim-4-d* **by** *blast*
lemmas $\lor E$ = *intro-elim-4-a intro-elim-4-b intro-elim-4-c intro-elim-4-d*
lemma *intro-elim-5*[*PLM*]:
$[\![[\varphi \to \psi\ in\ v]; [\psi \to \varphi\ in\ v]]\!] \Longrightarrow [\varphi \equiv \psi\ in\ v]$

by (*simp only*: *equiv-def* **&**I)
lemmas $\equiv I$ = *intro-elim-5*
lemma *intro-elim-6-a*[*PLM*]:
$[\![[\varphi \equiv \psi\ in\ v]; [\varphi\ in\ v]]\!] \Longrightarrow [\psi\ in\ v]$
unfolding *equiv-def* **using** **&**$E(1)$ *vdash-properties-10* **by** *blast*
lemma *intro-elim-6-b*[*PLM*]:
$[\![[\varphi \equiv \psi\ in\ v]; [\psi\ in\ v]]\!] \Longrightarrow [\varphi\ in\ v]$
unfolding *equiv-def* **using** **&**$E(2)$ *vdash-properties-10* **by** *blast*
lemma *intro-elim-6-c*[*PLM*]:
$[\![[\varphi \equiv \psi\ in\ v]; [\neg\varphi\ in\ v]]\!] \Longrightarrow [\neg\psi\ in\ v]$
unfolding *equiv-def* **using** **&**$E(2)$ *modus-tollens-1* **by** *blast*
lemma *intro-elim-6-d*[*PLM*]:
$[\![[\varphi \equiv \psi\ in\ v]; [\neg\psi\ in\ v]]\!] \Longrightarrow [\neg\varphi\ in\ v]$
unfolding *equiv-def* **using** **&**$E(1)$ *modus-tollens-1* **by** *blast*
lemma *intro-elim-6-e*[*PLM*]:
$[\![[\varphi \equiv \psi\ in\ v]; [\psi \equiv \chi\ in\ v]]\!] \Longrightarrow [\varphi \equiv \chi\ in\ v]$
by (*metis equiv-def ded-thm-cor-3* **&**E $\equiv I$)
lemma *intro-elim-6-f*[*PLM*]:
$[\![[\varphi \equiv \psi\ in\ v]; [\varphi \equiv \chi\ in\ v]]\!] \Longrightarrow [\chi \equiv \psi\ in\ v]$
by (*metis equiv-def ded-thm-cor-3* **&**E $\equiv I$)
lemmas $\equiv E$ = *intro-elim-6-a intro-elim-6-b intro-elim-6-c*
intro-elim-6-d intro-elim-6-e intro-elim-6-f
lemma *intro-elim-7*[*PLM*]:
$[\varphi\ in\ v] \Longrightarrow [\neg\neg\varphi\ in\ v]$
using *if-p-then-p modus-tollens-2* **by** *blast*
lemmas $\neg\neg I$ = *intro-elim-7*
lemma *intro-elim-8*[*PLM*]:
$[\neg\neg\varphi\ in\ v] \Longrightarrow [\varphi\ in\ v]$
using *if-p-then-p raa-cor-2* **by** *blast*
lemmas $\neg\neg E$ = *intro-elim-8*

context
begin
private lemma *NotNotI*[*PLM-intro*]:
$[\varphi\ in\ v] \Longrightarrow [\neg(\neg\varphi)\ in\ v]$
by (*simp add*: $\neg\neg I$)
private lemma *NotNotD*[*PLM-dest*]:
$[\neg(\neg\varphi)\ in\ v] \Longrightarrow [\varphi\ in\ v]$
using $\neg\neg E$ **by** *blast*

private lemma *ImplI*[*PLM-intro*]:
$([\varphi\ in\ v] \Longrightarrow [\psi\ in\ v]) \Longrightarrow [\varphi \rightarrow \psi\ in\ v]$
using *CP* **.**
private lemma *ImplE*[*PLM-elim*, *PLM-dest*]:
$[\varphi \rightarrow \psi\ in\ v] \Longrightarrow ([\varphi\ in\ v] \Longrightarrow [\psi\ in\ v])$
using *modus-ponens* **.**
private lemma *ImplS*[*PLM-subst*]:
$[\varphi \rightarrow \psi\ in\ v] = ([\varphi\ in\ v] \longrightarrow [\psi\ in\ v])$
using *ImplI ImplE* **by** *blast*

private lemma *NotI*[*PLM-intro*]:
$([\varphi\ in\ v] \Longrightarrow (\bigwedge\psi\ .[\psi\ in\ v])) \Longrightarrow [\neg\varphi\ in\ v]$
using *CP modus-tollens-2* **by** *blast*
private lemma *NotE*[*PLM-elim*,*PLM-dest*]:
$[\neg\varphi\ in\ v] \Longrightarrow ([\varphi\ in\ v] \longrightarrow (\forall\,\psi\ .[\psi\ in\ v]))$
using $\vee I(2)$ $\vee E(3)$ **by** *blast*
private lemma *NotS*[*PLM-subst*]:
$[\neg\varphi\ in\ v] = ([\varphi\ in\ v] \longrightarrow (\forall\,\psi\ .[\psi\ in\ v]))$

using *NotI NotE* **by** *blast*

private lemma *ConjI*[*PLM-intro*]:
$[\![[\varphi \ in \ v]; [\psi \ in \ v]]\!] \Longrightarrow [\varphi \ \& \ \psi \ in \ v]$
using $\&I$ **by** *blast*
private lemma *ConjE*[*PLM-elim*,*PLM-dest*]:
$[\varphi \ \& \ \psi \ in \ v] \Longrightarrow (([\varphi \ in \ v] \wedge [\psi \ in \ v]))$
using *CP* $\&E$ **by** *blast*
private lemma *ConjS*[*PLM-subst*]:
$[\varphi \ \& \ \psi \ in \ v] = (([\varphi \ in \ v] \wedge [\psi \ in \ v]))$
using *ConjI ConjE* **by** *blast*

private lemma *DisjI*[*PLM-intro*]:
$[\varphi \ in \ v] \vee [\psi \ in \ v] \Longrightarrow [\varphi \vee \psi \ in \ v]$
using $\vee I$ **by** *blast*
private lemma *DisjE*[*PLM-elim*,*PLM-dest*]:
$[\varphi \vee \psi \ in \ v] \Longrightarrow [\varphi \ in \ v] \vee [\psi \ in \ v]$
using *CP* $\vee E(1)$ **by** *blast*
private lemma *DisjS*[*PLM-subst*]:
$[\varphi \vee \psi \ in \ v] = ([\varphi \ in \ v] \vee [\psi \ in \ v])$
using *DisjI DisjE* **by** *blast*

private lemma *EquivI*[*PLM-intro*]:
$[\![[\varphi \ in \ v] \Longrightarrow [\psi \ in \ v];[\psi \ in \ v] \Longrightarrow [\varphi \ in \ v]]\!] \Longrightarrow [\varphi \equiv \psi \ in \ v]$
using *CP* $\equiv I$ **by** *blast*
private lemma *EquivE*[*PLM-elim*,*PLM-dest*]:
$[\varphi \equiv \psi \ in \ v] \Longrightarrow (([\varphi \ in \ v] \longrightarrow [\psi \ in \ v]) \wedge ([\psi \ in \ v] \longrightarrow [\varphi \ in \ v]))$
using $\equiv E(1)$ $\equiv E(2)$ **by** *blast*
private lemma *EquivS*[*PLM-subst*]:
$[\varphi \equiv \psi \ in \ v] = ([\varphi \ in \ v] \longleftrightarrow [\psi \ in \ v])$
using *EquivI EquivE* **by** *blast*

private lemma *NotOrD*[*PLM-dest*]:
$\neg[\varphi \vee \psi \ in \ v] \Longrightarrow \neg[\varphi \ in \ v] \wedge \neg[\psi \ in \ v]$
using $\vee I$ **by** *blast*
private lemma *NotAndD*[*PLM-dest*]:
$\neg[\varphi \ \& \ \psi \ in \ v] \Longrightarrow \neg[\varphi \ in \ v] \vee \neg[\psi \ in \ v]$
using $\&I$ **by** *blast*
private lemma *NotEquivD*[*PLM-dest*]:
$\neg[\varphi \equiv \psi \ in \ v] \Longrightarrow [\varphi \ in \ v] \neq [\psi \ in \ v]$
by (*meson NotI contraposition-1* $\equiv I$ *vdash-properties-9*)

private lemma *BoxI*[*PLM-intro*]:
$(\bigwedge \ v \ . \ [\varphi \ in \ v]) \Longrightarrow [\Box\varphi \ in \ v]$
using *RN* **by** *blast*
private lemma *NotBoxD*[*PLM-dest*]:
$\neg[\Box\varphi \ in \ v] \Longrightarrow (\exists \ v \ . \ \neg[\varphi \ in \ v])$
using *BoxI* **by** *blast*

private lemma *AllI*[*PLM-intro*]:
$(\bigwedge \ x \ . \ [\varphi \ x \ in \ v]) \Longrightarrow [\forall \ x \ . \ \varphi \ x \ in \ v]$
using *rule-gen* **by** *blast*
lemma *NotAllD*[*PLM-dest*]:
$\neg[\forall \ x \ . \ \varphi \ x \ in \ v] \Longrightarrow (\exists \ x \ . \ \neg[\varphi \ x \ in \ v])$
using *AllI* **by** *fastforce*
end

lemma *oth-class-taut-1-a*[*PLM*]:

[$\neg(\varphi$ **&** $\neg\varphi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-1-b*[*PLM*]:
[$\neg(\varphi \equiv \neg\varphi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-2*[*PLM*]:
[$\varphi \vee \neg\varphi$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-3-a*[*PLM*]:
[$(\varphi$ **&** $\varphi) \equiv \varphi$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-3-b*[*PLM*]:
[$(\varphi$ **&** $\psi) \equiv (\psi$ **&** $\varphi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-3-c*[*PLM*]:
[$(\varphi$ **&** $(\psi$ **&** $\chi)) \equiv ((\varphi$ **&** $\psi)$ **&** $\chi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-3-d*[*PLM*]:
[$(\varphi \vee \varphi) \equiv \varphi$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-3-e*[*PLM*]:
[$(\varphi \vee \psi) \equiv (\psi \vee \varphi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-3-f*[*PLM*]:
[$(\varphi \vee (\psi \vee \chi)) \equiv ((\varphi \vee \psi) \vee \chi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-3-g*[*PLM*]:
[$(\varphi \equiv \psi) \equiv (\psi \equiv \varphi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-3-i*[*PLM*]:
[$(\varphi \equiv (\psi \equiv \chi)) \equiv ((\varphi \equiv \psi) \equiv \chi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-4-a*[*PLM*]:
[$\varphi \equiv \varphi$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-4-b*[*PLM*]:
[$\varphi \equiv \neg\neg\varphi$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-5-a*[*PLM*]:
[$(\varphi \rightarrow \psi) \equiv \neg(\varphi$ **&** $\neg\psi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-5-b*[*PLM*]:
[$\neg(\varphi \rightarrow \psi) \equiv (\varphi$ **&** $\neg\psi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-5-c*[*PLM*]:
[$(\varphi \rightarrow \psi) \rightarrow ((\psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \chi))$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-5-d*[*PLM*]:
[$(\varphi \equiv \psi) \equiv (\neg\varphi \equiv \neg\psi)$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-5-e*[*PLM*]:
[$(\varphi \equiv \psi) \rightarrow ((\varphi \rightarrow \chi) \equiv (\psi \rightarrow \chi))$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-5-f*[*PLM*]:
[$(\varphi \equiv \psi) \rightarrow ((\chi \rightarrow \varphi) \equiv (\chi \rightarrow \psi))$ *in v*]
by *PLM-solver*
lemma *oth-class-taut-5-g*[*PLM*]:
[$(\varphi \equiv \psi) \rightarrow ((\varphi \equiv \chi) \equiv (\psi \equiv \chi))$ *in v*]

by *PLM-solver*
lemma *oth-class-taut-5-h*[*PLM*]:
$[(\varphi \equiv \psi) \rightarrow ((\chi \equiv \varphi) \equiv (\chi \equiv \psi))\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-5-i*[*PLM*]:
$[(\varphi \equiv \psi) \equiv ((\varphi\ \&\ \psi) \vee (\neg\varphi\ \&\ \neg\psi))\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-5-j*[*PLM*]:
$[(\neg(\varphi \equiv \psi)) \equiv ((\varphi\ \&\ \neg\psi) \vee (\neg\varphi\ \&\ \psi))\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-5-k*[*PLM*]:
$[(\varphi \rightarrow \psi) \equiv (\neg\varphi \vee \psi)\ \mathit{in}\ v]$
by *PLM-solver*

lemma *oth-class-taut-6-a*[*PLM*]:
$[(\varphi\ \&\ \psi) \equiv \neg(\neg\varphi \vee \neg\psi)\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-6-b*[*PLM*]:
$[(\varphi \vee \psi) \equiv \neg(\neg\varphi\ \&\ \neg\psi)\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-6-c*[*PLM*]:
$[\neg(\varphi\ \&\ \psi) \equiv (\neg\varphi \vee \neg\psi)\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-6-d*[*PLM*]:
$[\neg(\varphi \vee \psi) \equiv (\neg\varphi\ \&\ \neg\psi)\ \mathit{in}\ v]$
by *PLM-solver*

lemma *oth-class-taut-7-a*[*PLM*]:
$[(\varphi\ \&\ (\psi \vee \chi)) \equiv ((\varphi\ \&\ \psi) \vee (\varphi\ \&\ \chi))\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-7-b*[*PLM*]:
$[(\varphi \vee (\psi\ \&\ \chi)) \equiv ((\varphi \vee \psi)\ \&\ (\varphi \vee \chi))\ \mathit{in}\ v]$
by *PLM-solver*

lemma *oth-class-taut-8-a*[*PLM*]:
$[((\varphi\ \&\ \psi) \rightarrow \chi) \rightarrow (\varphi \rightarrow (\psi \rightarrow \chi))\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-8-b*[*PLM*]:
$[(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi\ \&\ \psi) \rightarrow \chi)\ \mathit{in}\ v]$
by *PLM-solver*

lemma *oth-class-taut-9-a*[*PLM*]:
$[(\varphi\ \&\ \psi) \rightarrow \varphi\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-9-b*[*PLM*]:
$[(\varphi\ \&\ \psi) \rightarrow \psi\ \mathit{in}\ v]$
by *PLM-solver*

lemma *oth-class-taut-10-a*[*PLM*]:
$[\varphi \rightarrow (\psi \rightarrow (\varphi\ \&\ \psi))\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-10-b*[*PLM*]:
$[(\varphi \rightarrow (\psi \rightarrow \chi)) \equiv (\psi \rightarrow (\varphi \rightarrow \chi))\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-10-c*[*PLM*]:
$[(\varphi \rightarrow \psi) \rightarrow ((\varphi \rightarrow \chi) \rightarrow (\varphi \rightarrow (\psi\ \&\ \chi)))\ \mathit{in}\ v]$
by *PLM-solver*
lemma *oth-class-taut-10-d*[*PLM*]:

$[(\varphi \to \chi) \to ((\psi \to \chi) \to ((\varphi \vee \psi) \to \chi))\ in\ v]$
by *PLM-solver*
lemma *oth-class-taut-10-e*[*PLM*]:
$[(\varphi \to \psi) \to ((\chi \to \Theta) \to ((\varphi\ \&\ \chi) \to (\psi\ \&\ \Theta)))\ in\ v]$
by *PLM-solver*
lemma *oth-class-taut-10-f*[*PLM*]:
$[((\varphi\ \&\ \psi) \equiv (\varphi\ \&\ \chi)) \equiv (\varphi \to (\psi \equiv \chi))\ in\ v]$
by *PLM-solver*
lemma *oth-class-taut-10-g*[*PLM*]:
$[((\varphi\ \&\ \psi) \equiv (\chi\ \&\ \psi)) \equiv (\psi \to (\varphi \equiv \chi))\ in\ v]$
by *PLM-solver*

attribute-setup *equiv-lr* = ⟪
Scan.succeed (*Thm.rule-attribute* []
(*fn* - => *fn thm* => *thm RS* @{*thm* ≡*E*(*1*)}))
⟫

attribute-setup *equiv-rl* = ⟪
Scan.succeed (*Thm.rule-attribute* []
(*fn* - => *fn thm* => *thm RS* @{*thm* ≡*E*(*2*)}))
⟫

attribute-setup *equiv-sym* = ⟪
Scan.succeed (*Thm.rule-attribute* []
(*fn* - => *fn thm* => *thm RS* @{*thm oth-class-taut-3-g*[*equiv-lr*]}))
⟫

attribute-setup *conj1* = ⟪
Scan.succeed (*Thm.rule-attribute* []
(*fn* - => *fn thm* => *thm RS* @{*thm* &*E*(*1*)}))
⟫

attribute-setup *conj2* = ⟪
Scan.succeed (*Thm.rule-attribute* []
(*fn* - => *fn thm* => *thm RS* @{*thm* &*E*(*2*)}))
⟫

attribute-setup *conj-sym* = ⟪
Scan.succeed (*Thm.rule-attribute* []
(*fn* - => *fn thm* => *thm RS* @{*thm oth-class-taut-3-b*[*equiv-lr*]}))
⟫

A.9.7. Identity

Remark. *For the following proofs first the definitions for the respective identities have to be expanded. They are defined directly in the embedded logic, though, so the proofs are still independent of the meta-logic.*

lemma *id-eq-prop-prop-1*[*PLM*]:
$[(F{::}\Pi_1) = F\ in\ v]$
unfolding *identity-defs* **by** *PLM-solver*
lemma *id-eq-prop-prop-2*[*PLM*]:
$[((F{::}\Pi_1) = G) \to (G = F)\ in\ v]$
by (*meson id-eq-prop-prop-1 CP ded-thm-cor-3 l-identity*[*axiom-instance*])
lemma *id-eq-prop-prop-3*[*PLM*]:
$[(((F{::}\Pi_1) = G)\ \&\ (G = H)) \to (F = H)\ in\ v]$
by (*metis l-identity*[*axiom-instance*] *ded-thm-cor-4 CP* &*E*)

lemma *id-eq-prop-prop-4-a*[*PLM*]:
$[(F::\Pi_2) = F \ in \ v]$
unfolding *identity-defs* **by** *PLM-solver*
lemma *id-eq-prop-prop-4-b*[*PLM*]:
$[(F::\Pi_3) = F \ in \ v]$
unfolding *identity-defs* **by** *PLM-solver*
lemma *id-eq-prop-prop-5-a*[*PLM*]:
$[((F::\Pi_2) = G) \rightarrow (G = F) \ in \ v]$
by (*meson id-eq-prop-prop-4-a CP ded-thm-cor-3 l-identity*[*axiom-instance*])
lemma *id-eq-prop-prop-5-b*[*PLM*]:
$[((F::\Pi_3) = G) \rightarrow (G = F) \ in \ v]$
by (*meson id-eq-prop-prop-4-b CP ded-thm-cor-3 l-identity*[*axiom-instance*])
lemma *id-eq-prop-prop-6-a*[*PLM*]:
$[(((F::\Pi_2) = G) \ \& \ (G = H)) \rightarrow (F = H) \ in \ v]$
by (*metis l-identity*[*axiom-instance*] *ded-thm-cor-4 CP* &*E*)
lemma *id-eq-prop-prop-6-b*[*PLM*]:
$[(((F::\Pi_3) = G) \ \& \ (G = H)) \rightarrow (F = H) \ in \ v]$
by (*metis l-identity*[*axiom-instance*] *ded-thm-cor-4 CP* &*E*)
lemma *id-eq-prop-prop-7*[*PLM*]:
$[(p::\Pi_0) = p \ in \ v]$
unfolding *identity-defs* **by** *PLM-solver*
lemma *id-eq-prop-prop-7-b*[*PLM*]:
$[(p::\mathrm{o}) = p \ in \ v]$
unfolding *identity-defs* **by** *PLM-solver*
lemma *id-eq-prop-prop-8*[*PLM*]:
$[((p::\Pi_0) = q) \rightarrow (q = p) \ in \ v]$
by (*meson id-eq-prop-prop-7 CP ded-thm-cor-3 l-identity*[*axiom-instance*])
lemma *id-eq-prop-prop-8-b*[*PLM*]:
$[((p::\mathrm{o}) = q) \rightarrow (q = p) \ in \ v]$
by (*meson id-eq-prop-prop-7-b CP ded-thm-cor-3 l-identity*[*axiom-instance*])
lemma *id-eq-prop-prop-9*[*PLM*]:
$[(((p::\Pi_0) = q) \ \& \ (q = r)) \rightarrow (p = r) \ in \ v]$
by (*metis l-identity*[*axiom-instance*] *ded-thm-cor-4 CP* &*E*)
lemma *id-eq-prop-prop-9-b*[*PLM*]:
$[(((p::\mathrm{o}) = q) \ \& \ (q = r)) \rightarrow (p = r) \ in \ v]$
by (*metis l-identity*[*axiom-instance*] *ded-thm-cor-4 CP* &*E*)

lemma *eq-E-simple-1*[*PLM*]:
$[(x =_E y) \equiv ((\!|O!,x|\!) \ \& \ (\!|O!,y|\!) \ \& \ \Box(\forall F \ . \ (\!|F,x|\!) \equiv (\!|F,y|\!))) \ in \ v]$
proof (*rule* $\equiv I$; *rule CP*)
assume *1*: $[x =_E y \ in \ v]$
have $[\forall \ x \ y \ . \ ((x^P) =_E (y^P)) \equiv ((\!|O!,x^P|\!) \ \& \ (\!|O!,y^P|\!)$
$\& \ \Box(\forall F \ . \ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))) \ in \ v]$
unfolding *identity*$_E$*-infix-def identity*$_E$*-def*
apply (*rule lambda-predicates-2-2*[*axiom-universal*, *axiom-universal*, *axiom-instance*])
by *show-proper*
moreover have $[\exists \ \alpha \ . \ (\alpha^P) = x \ in \ v]$
apply (*rule cqt-5-mod*[**where** $\psi = \lambda \ x \ . \ x =_E y$, *axiom-instance*,*deduction*])
unfolding *identity*$_E$*-infix-def*
apply (*rule SimpleExOrEnc.intros*)
using *1* **unfolding** *identity*$_E$*-infix-def* **by** *auto*
moreover have $[\exists \ \beta \ . \ (\beta^P) = y \ in \ v]$
apply (*rule cqt-5-mod*[**where** $\psi = \lambda \ y \ . \ x =_E y$,*axiom-instance*,*deduction*])
unfolding *identity*$_E$*-infix-def*
apply (*rule SimpleExOrEnc.intros*) **using** *1*
unfolding *identity*$_E$*-infix-def* **by** *auto*
ultimately have $[(x =_E y) \equiv ((\!|O!,x|\!) \ \& \ (\!|O!,y|\!)$
$\& \ \Box(\forall F \ . \ (\!|F,x|\!) \equiv (\!|F,y|\!))) \ in \ v]$

```
      using cqt-1-κ[axiom-instance,deduction, deduction] by meson
    thus [((⦇O!,x⦈ & ⦇O!,y⦈ & □(∀ F . ⦇F,x⦈ ≡ ⦇F,y⦈))) in v]
      using 1 ≡E(1) by blast
  next
    assume 1: [(⦇O!,x⦈ & ⦇O!,y⦈ & □(∀ F. ⦇F,x⦈ ≡ ⦇F,y⦈)) in v]
    have [∀ x y . ((x^P) =_E (y^P)) ≡ ((⦇O!,x^P⦈ & ⦇O!,y^P⦈
            & □(∀ F . ⦇F,x^P⦈ ≡ ⦇F,y^P⦈))) in v]
      unfolding identity_E-def identity_E-infix-def
      apply (rule lambda-predicates-2-2[axiom-universal, axiom-universal, axiom-instance])
      by show-proper
    moreover have [∃ α . (α^P) = x in v]
      apply (rule cqt-5-mod[where ψ=λ x . ⦇O!,x⦈,axiom-instance,deduction])
      apply (rule SimpleExOrEnc.intros)
      using 1[conj1,conj1] by auto
    moreover have [∃ β . (β^P) = y in v]
      apply (rule cqt-5-mod[where ψ=λ y . ⦇O!,y⦈,axiom-instance,deduction])
       apply (rule SimpleExOrEnc.intros)
      using 1[conj1,conj2] by auto
    ultimately have [(x =_E y) ≡ (⦇O!,x⦈ & ⦇O!,y⦈
                    & □(∀ F . ⦇F,x⦈ ≡ ⦇F,y⦈)) in v]
    using cqt-1-κ[axiom-instance,deduction, deduction] by meson
    thus [(x =_E y) in v] using 1 ≡E(2) by blast
  qed
lemma eq-E-simple-2[PLM]:
  [(x =_E y) → (x = y) in v]
  unfolding identity-defs by PLM-solver
lemma eq-E-simple-3[PLM]:
  [(x = y) ≡ (((⦇O!,x⦈ & ⦇O!,y⦈ & □(∀ F . ⦇F,x⦈ ≡ ⦇F,y⦈))
          ∨ (⦇A!,x⦈ & ⦇A!,y⦈ & □(∀ F. ⦃x,F⦄ ≡ ⦃y,F⦄))) in v]
  using eq-E-simple-1
  apply − unfolding identity-defs
  by PLM-solver

lemma id-eq-obj-1[PLM]: [(x^P) = (x^P) in v]
  proof −
    have [(◇⦇E!, x^P⦈) ∨ (¬◇⦇E!, x^P⦈) in v]
      using PLM.oth-class-taut-2 by simp
    hence [(◇⦇E!, x^P⦈) in v] ∨ [(¬◇⦇E!, x^P⦈) in v]
      using CP ∨E(1) by blast
    moreover {
      assume [(◇⦇E!, x^P⦈) in v]
      hence [⦇λx. ◇⦇E!,x^P⦈,x^P⦈ in v]
        apply (rule lambda-predicates-2-1[axiom-instance, equiv-rl, rotated])
        by show-proper
      hence [⦇λx. ◇⦇E!,x^P⦈,x^P⦈ & ⦇λx. ◇⦇E!,x^P⦈,x^P⦈
            & □(∀ F. ⦇F,x^P⦈ ≡ ⦇F,x^P⦈) in v]
        apply − by PLM-solver
      hence [(x^P) =_E (x^P) in v]
        using eq-E-simple-1[equiv-rl] unfolding Ordinary-def by fast
    }
    moreover {
      assume [(¬◇⦇E!, x^P⦈) in v]
      hence [⦇λx. ¬◇⦇E!,x^P⦈,x^P⦈ in v]
        apply (rule lambda-predicates-2-1[axiom-instance, equiv-rl, rotated])
        by show-proper
      hence [⦇λx. ¬◇⦇E!,x^P⦈,x^P⦈ & ⦇λx. ¬◇⦇E!,x^P⦈,x^P⦈
            & □(∀ F. ⦃x^P,F⦄ ≡ ⦃x^P,F⦄) in v]
        apply − by PLM-solver
```

```
      }
      ultimately show ?thesis unfolding identity-defs Ordinary-def Abstract-def
        using ∨I by blast
    qed
  lemma id-eq-obj-2[PLM]:
    [((x^P) = (y^P)) → ((y^P) = (x^P)) in v]
    by (meson l-identity[axiom-instance] id-eq-obj-1 CP ded-thm-cor-3)
  lemma id-eq-obj-3[PLM]:
    [((x^P) = (y^P)) & ((y^P) = (z^P)) → ((x^P) = (z^P)) in v]
    by (metis l-identity[axiom-instance] ded-thm-cor-4 CP &E)
end
```

Remark. *To unify the statements of the properties of equality a type class is introduced.*

```
class id-eq = quantifiable-and-identifiable +
  assumes id-eq-1: [(x :: 'a) = x in v]
  assumes id-eq-2: [((x :: 'a) = y) → (y = x) in v]
  assumes id-eq-3: [((x :: 'a) = y) & (y = z) → (x = z) in v]

instantiation ν :: id-eq
begin
  instance proof
    fix x :: ν and v
    show [x = x in v]
      using PLM.id-eq-obj-1
      by (simp add: identity-ν-def)
  next
    fix x y::ν and v
    show [x = y → y = x in v]
      using PLM.id-eq-obj-2
      by (simp add: identity-ν-def)
  next
    fix x y z::ν and v
    show [((x = y) & (y = z)) → x = z in v]
      using PLM.id-eq-obj-3
      by (simp add: identity-ν-def)
  qed
end

instantiation o :: id-eq
begin
  instance proof
    fix x :: o and v
    show [x = x in v]
      using PLM.id-eq-prop-prop-7 .
  next
    fix x y :: o and v
    show [x = y → y = x in v]
      using PLM.id-eq-prop-prop-8 .
  next
    fix x y z :: o and v
    show [((x = y) & (y = z)) → x = z in v]
      using PLM.id-eq-prop-prop-9 .
  qed
end

instantiation Π₁ :: id-eq
begin
```

```
  instance proof
    fix x :: Π1 and v
    show [x = x in v]
      using PLM.id-eq-prop-prop-1 .
  next
    fix x y :: Π1 and v
    show [x = y → y = x in v]
      using PLM.id-eq-prop-prop-2 .
  next
    fix x y z :: Π1 and v
    show [((x = y) & (y = z)) → x = z in v]
      using PLM.id-eq-prop-prop-3 .
  qed
end

instantiation Π2 :: id-eq
begin
  instance proof
    fix x :: Π2 and v
    show [x = x in v]
      using PLM.id-eq-prop-prop-4-a .
  next
    fix x y :: Π2 and v
    show [x = y → y = x in v]
      using PLM.id-eq-prop-prop-5-a .
  next
    fix x y z :: Π2 and v
    show [((x = y) & (y = z)) → x = z in v]
      using PLM.id-eq-prop-prop-6-a .
  qed
end

instantiation Π3 :: id-eq
begin
  instance proof
    fix x :: Π3 and v
    show [x = x in v]
      using PLM.id-eq-prop-prop-4-b .
  next
    fix x y :: Π3 and v
    show [x = y → y = x in v]
      using PLM.id-eq-prop-prop-5-b .
  next
    fix x y z :: Π3 and v
    show [((x = y) & (y = z)) → x = z in v]
      using PLM.id-eq-prop-prop-6-b .
  qed
end

context PLM
begin
  lemma id-eq-1[PLM]:
    [(x::'a::id-eq) = x in v]
    using id-eq-1 .
  lemma id-eq-2[PLM]:
    [((x::'a::id-eq) = y) → (y = x) in v]
    using id-eq-2 .
  lemma id-eq-3[PLM]:
```

```
  [((x::'a::id-eq) = y) & (y = z) → (x = z) in v]
  using id-eq-3 .

attribute-setup eq-sym = ‹
  Scan.succeed (Thm.rule-attribute []
    (fn - => fn thm => thm RS @{thm id-eq-2[deduction]}))
›


lemma all-self-eq-1[PLM]:
  [□(∀ α :: 'a::id-eq . α = α) in v]
  by PLM-solver
lemma all-self-eq-2[PLM]:
  [∀ α :: 'a::id-eq . □(α = α) in v]
  by PLM-solver

lemma t-id-t-proper-1[PLM]:
  [τ = τ' → (∃ β . (β^P) = τ) in v]
  proof (rule CP)
    assume [τ = τ' in v]
    moreover {
      assume [τ =_E τ' in v]
      hence [∃ β . (β^P) = τ in v]
        apply −
        apply (rule cqt-5-mod[where ψ=λ τ . τ =_E τ', axiom-instance, deduction])
         subgoal unfolding identity-defs by (rule SimpleExOrEnc.intros)
        by simp
    }
    moreover {
      assume [(|A!,τ|) & (|A!,τ'|) & □(∀ F. {|τ,F|} ≡ {|τ',F|}) in v]
      hence [∃ β . (β^P) = τ in v]
        apply −
        apply (rule cqt-5-mod[where ψ=λ τ . (|A!,τ|), axiom-instance, deduction])
         subgoal unfolding identity-defs by (rule SimpleExOrEnc.intros)
        by PLM-solver
    }
    ultimately show [∃ β . (β^P) = τ in v] unfolding identity_κ-def
      using intro-elim-4-b reductio-aa-1 by blast
  qed

lemma t-id-t-proper-2[PLM]: [τ = τ' → (∃ β . (β^P) = τ') in v]
proof (rule CP)
  assume [τ = τ' in v]
  moreover {
    assume [τ =_E τ' in v]
    hence [∃ β . (β^P) = τ' in v]
      apply −
      apply (rule cqt-5-mod[where ψ=λ τ' . τ =_E τ', axiom-instance, deduction])
       subgoal unfolding identity-defs by (rule SimpleExOrEnc.intros)
      by simp
  }
  moreover {
    assume [(|A!,τ|) & (|A!,τ'|) & □(∀ F. {|τ,F|} ≡ {|τ',F|}) in v]
    hence [∃ β . (β^P) = τ' in v]
      apply −
      apply (rule cqt-5-mod[where ψ=λ τ . (|A!,τ|), axiom-instance, deduction])
       subgoal unfolding identity-defs by (rule SimpleExOrEnc.intros)
      by PLM-solver
```

```
  }
  ultimately show [∃ β . (β^P) = τ' in v] unfolding identity_κ-def
    using intro-elim-4-b reductio-aa-1 by blast
qed

lemma id-nec[PLM]: [((α::'a::id-eq) = (β)) ≡ □((α) = (β)) in v]
  apply (rule ≡I)
   using l-identity[where φ = (λ β .  □((α) = (β))), axiom-instance]
        id-eq-1 RN ded-thm-cor-4 unfolding identity-ν-def
   apply blast
  using qml-2[axiom-instance] by blast

lemma id-nec-desc[PLM]:
  [((ιx. φ x) = (ιx. ψ x)) ≡ □((ιx. φ x) = (ιx. ψ x)) in v]
  proof (cases [(∃ α. (α^P) = (ιx . φ x)) in v] ∧ [(∃ β. (β^P) = (ιx . ψ x)) in v])
    assume [(∃ α. (α^P) = (ιx . φ x)) in v] ∧ [(∃ β. (β^P) = (ιx . ψ x)) in v]
    then obtain α and β where
      [(α^P) = (ιx . φ x) in v] ∧ [(β^P) = (ιx . ψ x) in v]
      apply – unfolding conn-defs by PLM-solver
    moreover {
      moreover have [(α) = (β) ≡ □((α) = (β)) in v] by PLM-solver
      ultimately have [((ιx. φ x) = (β^P) ≡ □((ιx. φ x) = (β^P))) in v]
        using l-identity[where φ=λ α . (α) = (β^P) ≡ □((α) = (β^P)), axiom-instance]
        modus-ponens unfolding identity-ν-def by metis
    }
    ultimately show ?thesis
      using l-identity[where φ=λ α . (ιx . φ x) = (α)
                              ≡ □((ιx . φ x) = (α)), axiom-instance]
      modus-ponens by metis
  next
    assume ¬([(∃ α. (α^P) = (ιx . φ x)) in v] ∧ [(∃ β. (β^P) = (ιx . ψ x)) in v])
    hence ¬[⦇A!,(ιx . φ x)⦈ in v] ∧ ¬[(ιx . φ x) =_E (ιx . ψ x) in v]
         ∨ ¬[⦇A!,(ιx . ψ x)⦈ in v] ∧ ¬[(ιx . φ x) =_E (ιx . ψ x) in v]
    unfolding identity_E-infix-def
    using cqt-5[axiom-instance] PLM.contraposition-1 SimpleExOrEnc.intros
          vdash-properties-10 by meson
    hence ¬[(ιx . φ x) = (ιx . ψ x) in v]
      apply – unfolding identity-defs by PLM-solver
    thus ?thesis apply – apply PLM-solver
      using qml-2[axiom-instance, deduction] by auto
  qed
```

A.9.8. Quantification

```
lemma rule-ui[PLM,PLM-elim,PLM-dest]:
  [∀ α . φ α in v] ⟹ [φ β in v]
  by (meson cqt-1[axiom-instance, deduction])
lemmas ∀ E = rule-ui

lemma rule-ui-2[PLM,PLM-elim,PLM-dest]:
  ⟦[∀ α . φ (α^P) in v]; [∃ α . (α)^P = β in v]⟧ ⟹ [φ β in v]
  using cqt-1-κ[axiom-instance, deduction, deduction] by blast

lemma cqt-orig-1[PLM]:
  [(∀ α. φ α) → φ β in v]
  by PLM-solver
lemma cqt-orig-2[PLM]:
  [(∀ α. φ → ψ α) → (φ → (∀ α. ψ α)) in v]
```

by *PLM-solver*

lemma *universal*[*PLM*]:
$(\bigwedge\alpha\ .\ [\varphi\ \alpha\ in\ v]) \Longrightarrow [\forall\ \ \alpha\ .\ \varphi\ \alpha\ in\ v]$
using *rule-gen* **.**
lemmas $\forall\, I$ = *universal*

lemma *cqt-basic-1*[*PLM*]:
$[(\forall\,\alpha.\ (\forall\,\beta\ .\ \varphi\ \alpha\ \beta)) \equiv (\forall\,\beta.\ (\forall\,\alpha.\ \varphi\ \alpha\ \beta))\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-2*[*PLM*]:
$[(\forall\,\alpha.\ \varphi\ \alpha \equiv \psi\ \alpha) \equiv ((\forall\,\alpha.\ \varphi\ \alpha \rightarrow \psi\ \alpha)\ \&\ (\forall\,\alpha.\ \psi\ \alpha \rightarrow \varphi\ \alpha))\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-3*[*PLM*]:
$[(\forall\,\alpha.\ \varphi\ \alpha \equiv \psi\ \alpha) \rightarrow ((\forall\,\alpha.\ \varphi\ \alpha) \equiv (\forall\,\alpha.\ \psi\ \alpha))\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-4*[*PLM*]:
$[(\forall\,\alpha.\ \varphi\ \alpha\ \&\ \psi\ \alpha) \equiv ((\forall\,\alpha.\ \varphi\ \alpha)\ \&\ (\forall\,\alpha.\ \psi\ \alpha))\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-6*[*PLM*]:
$[(\forall\,\alpha.\ (\forall\,\alpha.\ \varphi\ \alpha)) \equiv (\forall\,\alpha.\ \varphi\ \alpha)\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-7*[*PLM*]:
$[(\varphi \rightarrow (\forall\,\alpha\ .\ \psi\ \alpha)) \equiv (\forall\,\alpha.(\varphi \rightarrow \psi\ \alpha))\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-8*[*PLM*]:
$[((\forall\,\alpha.\ \varphi\ \alpha) \vee (\forall\,\alpha.\ \psi\ \alpha)) \rightarrow (\forall\,\alpha.\ (\varphi\ \alpha \vee \psi\ \alpha))\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-9*[*PLM*]:
$[((\forall\,\alpha.\ \varphi\ \alpha \rightarrow \psi\ \alpha)\ \&\ (\forall\,\alpha.\ \psi\ \alpha \rightarrow \chi\ \alpha)) \rightarrow (\forall\,\alpha.\ \varphi\ \alpha \rightarrow \chi\ \alpha)\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-10*[*PLM*]:
$[((\forall\,\alpha.\ \varphi\ \alpha \equiv \psi\ \alpha)\ \&\ (\forall\,\alpha.\ \psi\ \alpha \equiv \chi\ \alpha)) \rightarrow (\forall\,\alpha.\ \varphi\ \alpha \equiv \chi\ \alpha)\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-11*[*PLM*]:
$[(\forall\,\alpha.\ \varphi\ \alpha \equiv \psi\ \alpha) \equiv (\forall\,\alpha.\ \psi\ \alpha \equiv \varphi\ \alpha)\ in\ v]$
by *PLM-solver*
lemma *cqt-basic-12*[*PLM*]:
$[(\forall\,\alpha.\ \varphi\ \alpha) \equiv (\forall\,\beta.\ \varphi\ \beta)\ in\ v]$
by *PLM-solver*

lemma *existential*[*PLM*,*PLM-intro*]:
$[\varphi\ \alpha\ in\ v] \Longrightarrow [\exists\ \ \alpha.\ \varphi\ \alpha\ in\ v]$
unfolding *exists-def* **by** *PLM-solver*
lemmas $\exists\, I$ = *existential*
lemma *instantiation-*[*PLM*,*PLM-elim*,*PLM-dest*]:
$[\![[\exists\,\alpha\ .\ \varphi\ \alpha\ in\ v];\ (\bigwedge\alpha.[\varphi\ \alpha\ in\ v] \Longrightarrow [\psi\ in\ v])]\!] \Longrightarrow [\psi\ in\ v]$
unfolding *exists-def* **by** *PLM-solver*

lemma *Instantiate*:
assumes $[\exists\ \ x\ .\ \varphi\ x\ \ in\ v]$
obtains x **where** $[\varphi\ x\ in\ v]$
apply (*insert assms*) **unfolding** *exists-def* **by** *PLM-solver*
lemmas $\exists\, E$ = *Instantiate*

lemma *cqt-further-1*[*PLM*]:
$[(\forall\,\alpha.\ \varphi\ \alpha) \rightarrow (\exists\,\alpha.\ \varphi\ \alpha)\ in\ v]$
by *PLM-solver*

```
lemma cqt-further-2[PLM]:
  [(¬(∀ α. φ α)) ≡ (∃ α. ¬φ α) in v]
  unfolding exists-def by PLM-solver
lemma cqt-further-3[PLM]:
  [(∀ α. φ α) ≡ ¬(∃ α. ¬φ α) in v]
  unfolding exists-def by PLM-solver
lemma cqt-further-4[PLM]:
  [(¬(∃ α. φ α)) ≡ (∀ α. ¬φ α) in v]
  unfolding exists-def by PLM-solver
lemma cqt-further-5[PLM]:
  [(∃ α. φ α & ψ α) → ((∃ α. φ α) & (∃ α. ψ α)) in v]
    unfolding exists-def by PLM-solver
lemma cqt-further-6[PLM]:
  [(∃ α. φ α ∨ ψ α) ≡ ((∃ α. φ α) ∨ (∃ α. ψ α)) in v]
  unfolding exists-def by PLM-solver
lemma cqt-further-10[PLM]:
  [(φ (α::'a::id-eq) & (∀ β . φ β → β = α)) ≡ (∀ β . φ β ≡ β = α) in v]
  apply PLM-solver
   using l-identity[axiom-instance, deduction, deduction] id-eq-2[deduction]
   apply blast
  using id-eq-1 by auto
lemma cqt-further-11[PLM]:
  [((∀ α. φ α) & (∀ α. ψ α)) → (∀ α. φ α ≡ ψ α) in v]
  by PLM-solver
lemma cqt-further-12[PLM]:
  [((¬(∃ α. φ α)) & (¬(∃ α. ψ α))) → (∀ α. φ α ≡ ψ α) in v]
  unfolding exists-def by PLM-solver
lemma cqt-further-13[PLM]:
  [((∃ α. φ α) & (¬(∃ α. ψ α))) → (¬(∀ α. φ α ≡ ψ α)) in v]
  unfolding exists-def by PLM-solver
lemma cqt-further-14[PLM]:
  [(∃ α. ∃ β. φ α β) ≡ (∃ β. ∃ α. φ α β) in v]
  unfolding exists-def by PLM-solver

lemma nec-exist-unique[PLM]:
  [(∀ x. φ x → □(φ x)) → ((∃ !x. φ x) → (∃ !x. □(φ x))) in v]
  proof (rule CP)
    assume a: [∀ x. φ x → □φ x in v]
    show [(∃ !x. φ x) → (∃ !x. □φ x) in v]
    proof (rule CP)
      assume [(∃ !x. φ x) in v]
      hence [∃ α. φ α & (∀ β. φ β → β = α) in v]
        by (simp only: exists-unique-def)
      then obtain α where 1:
        [φ α & (∀ β. φ β → β = α) in v]
        by (rule ∃ E)
      {
        fix β
        have [□φ β → β = α in v]
          using 1 &E(2) qml-2[axiom-instance]
            ded-thm-cor-3 ∀ E by fastforce
      }
      hence [∀ β. □φ β → β = α in v] by (rule ∀ I)
      moreover have [□(φ α) in v]
        using 1 &E(1) a vdash-properties-10 cqt-orig-1[deduction]
        by fast
      ultimately have [∃ α. □(φ α) & (∀ β. □φ β → β = α) in v]
        using &I ∃ I by fast
```

```
    thus [(∃!x. □φ x) in v]
      unfolding exists-unique-def by assumption
  qed
qed
```

A.9.9. Actuality and Descriptions

```
lemma nec-imp-act[PLM]: [□φ → 𝓐φ in v]
  apply (rule CP)
  using qml-act-2[axiom-instance, equiv-lr]
        qml-2[axiom-actualization, axiom-instance]
        logic-actual-nec-2[axiom-instance, equiv-lr, deduction]
  by blast
lemma act-conj-act-1[PLM]:
  [𝓐(𝓐φ → φ) in v]
  using equiv-def logic-actual-nec-2[axiom-instance]
        logic-actual-nec-4[axiom-instance] &E(2) ≡E(2)
  by metis
lemma act-conj-act-2[PLM]:
  [𝓐(φ → 𝓐φ) in v]
  using logic-actual-nec-2[axiom-instance] qml-act-1[axiom-instance]
        ded-thm-cor-3 ≡E(2) nec-imp-act
  by blast
lemma act-conj-act-3[PLM]:
  [(𝓐φ & 𝓐ψ) → 𝓐(φ & ψ) in v]
  unfolding conn-defs
  by (metis logic-actual-nec-2[axiom-instance]
            logic-actual-nec-1[axiom-instance]
            ≡E(2) CP ≡E(4) reductio-aa-2
            vdash-properties-10)
lemma act-conj-act-4[PLM]:
  [𝓐(𝓐φ ≡ φ) in v]
  unfolding equiv-def
  by (PLM-solver PLM-intro: act-conj-act-3[where φ=𝓐φ → φ
                            and ψ=φ → 𝓐φ, deduction])
lemma closure-act-1a[PLM]:
  [𝓐𝓐(𝓐φ ≡ φ) in v]
  using logic-actual-nec-4[axiom-instance]
        act-conj-act-4 ≡E(1)
  by blast
lemma closure-act-1b[PLM]:
  [𝓐𝓐𝓐(𝓐φ ≡ φ) in v]
  using logic-actual-nec-4[axiom-instance]
        act-conj-act-4 ≡E(1)
  by blast
lemma closure-act-1c[PLM]:
  [𝓐𝓐𝓐𝓐(𝓐φ ≡ φ) in v]
  using logic-actual-nec-4[axiom-instance]
        act-conj-act-4 ≡E(1)
  by blast
lemma closure-act-2[PLM]:
  [∀α. 𝓐(𝓐(φ α) ≡ φ α) in v]
  by PLM-solver

lemma closure-act-3[PLM]:
  [𝓐(∀α. 𝓐(φ α) ≡ φ α) in v]
  by (PLM-solver PLM-intro: logic-actual-nec-3[axiom-instance, equiv-rl])
lemma closure-act-4[PLM]:
```

```
  [𝓐(∀ α1 α2. 𝓐(φ α1 α2) ≡ φ α1 α2) in v]
  by (PLM-solver PLM-intro: logic-actual-nec-3[axiom-instance, equiv-rl])
lemma closure-act-4-b[PLM]:
  [𝓐(∀ α1 α2 α3. 𝓐(φ α1 α2 α3) ≡ φ α1 α2 α3) in v]
  by (PLM-solver PLM-intro: logic-actual-nec-3[axiom-instance, equiv-rl])
lemma closure-act-4-c[PLM]:
  [𝓐(∀ α1 α2 α3 α4. 𝓐(φ α1 α2 α3 α4) ≡ φ α1 α2 α3 α4) in v]
  by (PLM-solver PLM-intro: logic-actual-nec-3[axiom-instance, equiv-rl])

lemma RA[PLM,PLM-intro]:
  ([φ in dw]) ⟹ [𝓐φ in dw]
  using logic-actual[necessitation-averse-axiom-instance, equiv-rl] .

lemma RA-2[PLM,PLM-intro]:
  ([ψ in dw] ⟹ [φ in dw]) ⟹ ([𝓐ψ in dw] ⟹ [𝓐φ in dw])
  using RA logic-actual[necessitation-averse-axiom-instance] intro-elim-6-a by blast

context
begin
  private lemma ActualE[PLM,PLM-elim,PLM-dest]:
    [𝓐φ in dw] ⟹ [φ in dw]
    using logic-actual[necessitation-averse-axiom-instance, equiv-lr] .

  private lemma NotActualD[PLM-dest]:
    ¬[𝓐φ in dw] ⟹ ¬[φ in dw]
    using RA by metis

  private lemma ActualImplI[PLM-intro]:
    [𝓐φ → 𝓐ψ in v] ⟹ [𝓐(φ → ψ) in v]
    using logic-actual-nec-2[axiom-instance, equiv-rl] .
  private lemma ActualImplE[PLM-dest, PLM-elim]:
    [𝓐(φ → ψ) in v] ⟹ [𝓐φ → 𝓐ψ in v]
    using logic-actual-nec-2[axiom-instance, equiv-lr] .
  private lemma NotActualImplD[PLM-dest]:
    ¬[𝓐(φ → ψ) in v] ⟹ ¬[𝓐φ → 𝓐ψ in v]
    using ActualImplI by blast

  private lemma ActualNotI[PLM-intro]:
    [¬𝓐φ in v] ⟹ [𝓐¬φ in v]
    using logic-actual-nec-1[axiom-instance, equiv-rl] .
  lemma ActualNotE[PLM-elim,PLM-dest]:
    [𝓐¬φ in v] ⟹ [¬𝓐φ in v]
    using logic-actual-nec-1[axiom-instance, equiv-lr] .
  lemma NotActualNotD[PLM-dest]:
    ¬[𝓐¬φ in v] ⟹ ¬[¬𝓐φ in v]
    using ActualNotI by blast

  private  lemma ActualConjI[PLM-intro]:
    [𝓐φ & 𝓐ψ in v] ⟹ [𝓐(φ & ψ) in v]
    unfolding equiv-def
    by (PLM-solver PLM-intro: act-conj-act-3[deduction])
  private lemma ActualConjE[PLM-elim,PLM-dest]:
    [𝓐(φ & ψ) in v] ⟹ [𝓐φ & 𝓐ψ in v]
    unfolding conj-def by PLM-solver

  private lemma ActualEquivI[PLM-intro]:
    [𝓐φ ≡ 𝓐ψ in v] ⟹ [𝓐(φ ≡ ψ) in v]
    unfolding equiv-def
```

by (*PLM-solver PLM-intro*: *act-conj-act-3*[*deduction*])
private lemma *ActualEquivE*[*PLM-elim*, *PLM-dest*]:
$[\mathcal{A}(\varphi \equiv \psi)\ in\ v] \Longrightarrow [\mathcal{A}\varphi \equiv \mathcal{A}\psi\ in\ v]$
unfolding *equiv-def* **by** *PLM-solver*

private lemma *ActualBoxI*[*PLM-intro*]:
$[\Box\varphi\ in\ v] \Longrightarrow [\mathcal{A}(\Box\varphi)\ in\ v]$
using *qml-act-2*[*axiom-instance*, *equiv-lr*] **.**
private lemma *ActualBoxE*[*PLM-elim*, *PLM-dest*]:
$[\mathcal{A}(\Box\varphi)\ in\ v] \Longrightarrow [\Box\varphi\ in\ v]$
using *qml-act-2*[*axiom-instance*, *equiv-rl*] **.**
private lemma *NotActualBoxD*[*PLM-dest*]:
$\neg[\mathcal{A}(\Box\varphi)\ in\ v] \Longrightarrow \neg[\Box\varphi\ in\ v]$
using *ActualBoxI* **by** *blast*

private lemma *ActualDisjI*[*PLM-intro*]:
$[\mathcal{A}\varphi \vee \mathcal{A}\psi\ in\ v] \Longrightarrow [\mathcal{A}(\varphi \vee \psi)\ in\ v]$
unfolding *disj-def* **by** *PLM-solver*
private lemma *ActualDisjE*[*PLM-elim*,*PLM-dest*]:
$[\mathcal{A}(\varphi \vee \psi)\ in\ v] \Longrightarrow [\mathcal{A}\varphi \vee \mathcal{A}\psi\ in\ v]$
unfolding *disj-def* **by** *PLM-solver*
private lemma *NotActualDisjD*[*PLM-dest*]:
$\neg[\mathcal{A}(\varphi \vee \psi)\ in\ v] \Longrightarrow \neg[\mathcal{A}\varphi \vee \mathcal{A}\psi\ in\ v]$
using *ActualDisjI* **by** *blast*

private lemma *ActualForallI*[*PLM-intro*]:
$[\forall\ x\ .\ \mathcal{A}(\varphi\ x)\ in\ v] \Longrightarrow [\mathcal{A}(\forall\ x\ .\ \varphi\ x)\ in\ v]$
using *logic-actual-nec-3*[*axiom-instance*, *equiv-rl*] **.**
lemma *ActualForallE*[*PLM-elim*,*PLM-dest*]:
$[\mathcal{A}(\forall\ x\ .\ \varphi\ x)\ in\ v] \Longrightarrow [\forall\ x\ .\ \mathcal{A}(\varphi\ x)\ in\ v]$
using *logic-actual-nec-3*[*axiom-instance*, *equiv-lr*] **.**
lemma *NotActualForallD*[*PLM-dest*]:
$\neg[\mathcal{A}(\forall\ x\ .\ \varphi\ x)\ in\ v] \Longrightarrow \neg[\forall\ x\ .\ \mathcal{A}(\varphi\ x)\ in\ v]$
using *ActualForallI* **by** *blast*

lemma *ActualActualI*[*PLM-intro*]:
$[\mathcal{A}\varphi\ in\ v] \Longrightarrow [\mathcal{A}\mathcal{A}\varphi\ in\ v]$
using *logic-actual-nec-4*[*axiom-instance*, *equiv-lr*] **.**
lemma *ActualActualE*[*PLM-elim*,*PLM-dest*]:
$[\mathcal{A}\mathcal{A}\varphi\ in\ v] \Longrightarrow [\mathcal{A}\varphi\ in\ v]$
using *logic-actual-nec-4*[*axiom-instance*, *equiv-rl*] **.**
lemma *NotActualActualD*[*PLM-dest*]:
$\neg[\mathcal{A}\mathcal{A}\varphi\ in\ v] \Longrightarrow \neg[\mathcal{A}\varphi\ in\ v]$
using *ActualActualI* **by** *blast*
end

lemma *ANeg-1*[*PLM*]:
$[\neg\mathcal{A}\varphi \equiv \neg\varphi\ in\ dw]$
by *PLM-solver*
lemma *ANeg-2*[*PLM*]:
$[\neg\mathcal{A}\neg\varphi \equiv \varphi\ in\ dw]$
by *PLM-solver*
lemma *Act-Basic-1*[*PLM*]:
$[\mathcal{A}\varphi \vee \mathcal{A}\neg\varphi\ in\ v]$
by *PLM-solver*
lemma *Act-Basic-2*[*PLM*]:
$[\mathcal{A}(\varphi\ \&\ \psi) \equiv (\mathcal{A}\varphi\ \&\ \mathcal{A}\psi)\ in\ v]$
by *PLM-solver*

lemma *Act-Basic-3*[*PLM*]:
$[\mathcal{A}(\varphi \equiv \psi) \equiv ((\mathcal{A}(\varphi \to \psi))\ \&\ (\mathcal{A}(\psi \to \varphi)))\ in\ v]$
by *PLM-solver*
lemma *Act-Basic-4*[*PLM*]:
$[(\mathcal{A}(\varphi \to \psi)\ \&\ \mathcal{A}(\psi \to \varphi)) \equiv (\mathcal{A}\varphi \equiv \mathcal{A}\psi)\ in\ v]$
by *PLM-solver*
lemma *Act-Basic-5*[*PLM*]:
$[\mathcal{A}(\varphi \equiv \psi) \equiv (\mathcal{A}\varphi \equiv \mathcal{A}\psi)\ in\ v]$
by *PLM-solver*
lemma *Act-Basic-6*[*PLM*]:
$[\Diamond\varphi \equiv \mathcal{A}(\Diamond\varphi)\ in\ v]$
unfolding *diamond-def* **by** *PLM-solver*
lemma *Act-Basic-7*[*PLM*]:
$[\mathcal{A}\varphi \equiv \Box\mathcal{A}\varphi\ in\ v]$
by (*simp add*: *qml-2*[*axiom-instance*] *qml-act-1*[*axiom-instance*] $\equiv I$)
lemma *Act-Basic-8*[*PLM*]:
$[\mathcal{A}(\Box\varphi) \to \Box\mathcal{A}\varphi\ in\ v]$
by (*metis qml-act-2*[*axiom-instance*] *CP Act-Basic-7* $\equiv E(1)$
$\equiv E(2)$ *nec-imp-act vdash-properties-10*)
lemma *Act-Basic-9*[*PLM*]:
$[\Box\varphi \to \Box\mathcal{A}\varphi\ in\ v]$
using *qml-act-1*[*axiom-instance*] *ded-thm-cor-3 nec-imp-act* **by** *blast*
lemma *Act-Basic-10*[*PLM*]:
$[\mathcal{A}(\varphi \lor \psi) \equiv \mathcal{A}\varphi \lor \mathcal{A}\psi\ in\ v]$
by *PLM-solver*

lemma *Act-Basic-11*[*PLM*]:
$[\mathcal{A}(\exists\,\alpha.\ \varphi\ \alpha) \equiv (\exists\,\alpha.\mathcal{A}(\varphi\ \alpha))\ in\ v]$
proof −
have $[\mathcal{A}(\forall\ \alpha\ .\ \neg\varphi\ \alpha) \equiv (\forall\ \alpha\ .\ \mathcal{A}\neg\varphi\ \alpha)\ in\ v]$
using *logic-actual-nec-3*[*axiom-instance*] **by** *blast*
hence $[\neg\mathcal{A}(\forall\ \alpha\ .\ \neg\varphi\ \alpha) \equiv \neg(\forall\ \alpha\ .\ \mathcal{A}\neg\varphi\ \alpha)\ in\ v]$
using *oth-class-taut-5-d*[*equiv-lr*] **by** *blast*
moreover have $[\mathcal{A}\neg(\forall\ \alpha\ .\ \neg\varphi\ \alpha) \equiv \neg\mathcal{A}(\forall\ \alpha\ .\ \neg\varphi\ \alpha)\ in\ v]$
using *logic-actual-nec-1*[*axiom-instance*] **by** *blast*
ultimately have $[\mathcal{A}\neg(\forall\ \alpha\ .\ \neg\varphi\ \alpha) \equiv \neg(\forall\ \alpha\ .\ \mathcal{A}\neg\varphi\ \alpha)\ in\ v]$
using $\equiv E(5)$ **by** *auto*
moreover {
have $[\forall\ \alpha\ .\ \mathcal{A}\neg\varphi\ \alpha \equiv \neg\mathcal{A}\varphi\ \alpha\ in\ v]$
using *logic-actual-nec-1*[*axiom-universal*, *axiom-instance*] **by** *blast*
hence $[(\forall\ \alpha\ .\ \mathcal{A}\neg\varphi\ \alpha) \equiv (\forall\ \alpha\ .\ \neg\mathcal{A}\varphi\ \alpha)\ in\ v]$
using *cqt-basic-3*[*deduction*] **by** *fast*
hence $[(\neg(\forall\ \alpha\ .\ \mathcal{A}\neg\varphi\ \alpha)) \equiv \neg(\forall\ \alpha\ .\ \neg\mathcal{A}\varphi\ \alpha)\ in\ v]$
using *oth-class-taut-5-d*[*equiv-lr*] **by** *blast*
}
ultimately show *?thesis* **unfolding** *exists-def* **using** $\equiv E(5)$ **by** *auto*
qed

lemma *act-quant-uniq*[*PLM*]:
$[(\forall\ z\ .\ \mathcal{A}\varphi\ z \equiv z = x) \equiv (\forall\ z\ .\ \varphi\ z \equiv z = x)\ in\ dw]$
by *PLM-solver*

lemma *fund-cont-desc*[*PLM*]:
$[(x^P = (\iota x.\ \varphi\ x)) \equiv (\forall\ z\ .\ \varphi\ z \equiv (z = x))\ in\ dw]$
using *descriptions*[*axiom-instance*] *act-quant-uniq* $\equiv E(5)$ **by** *fast*

lemma *hintikka*[*PLM*]:
$[(x^P = (\iota x.\ \varphi\ x)) \equiv (\varphi\ x\ \&\ (\forall\ z\ .\ \varphi\ z \to z = x))\ in\ dw]$

```
  proof -
    have [(∀ z . φ z ≡ z = x) ≡ (φ x & (∀ z . φ z → z = x)) in dw]
      unfolding identity-ν-def apply PLM-solver using id-eq-obj-1 apply simp
      using l-identity[where φ=λ x . φ x, axiom-instance,
                          deduction, deduction]
      using id-eq-obj-2[deduction] unfolding identity-ν-def by fastforce
    thus ?thesis using ≡E(5) fund-cont-desc by blast
  qed

lemma russell-axiom-a[PLM]:
  [((|F, ιx. φ x|)) ≡ (∃ x . φ x & (∀ z . φ z → z = x) & (|F, x^P|)) in dw]
  (is [?lhs ≡ ?rhs in dw])
  proof -
    {
      assume 1: [?lhs in dw]
      hence [∃ α. α^P = (ιx. φ x) in dw]
      using cqt-5[axiom-instance, deduction]
            SimpleExOrEnc.intros
      by blast
      then obtain α where 2:
        [α^P = (ιx. φ x) in dw]
        using ∃ E by auto
      hence 3: [φ α & (∀ z . φ z → z = α) in dw]
        using hintikka[equiv-lr] by simp
      from 2 have [(ιx. φ x) = (α^P)  in dw]
        using l-identity[where α=α^P and β=ιx. φ x and φ=λ x . x = α^P,
              axiom-instance, deduction, deduction]
              id-eq-obj-1[where x=α] by auto
      hence [(|F, α^P|) in dw]
      using 1 l-identity[where β=α^P and α=ιx. φ x and φ=λ x . (|F,x|),
                           axiom-instance, deduction, deduction] by auto
      with 3 have [φ α & (∀ z . φ z → z = α) & (|F, α^P|) in dw] by (rule &I)
      hence [?rhs in dw] using ∃ I[where α=α] by simp
    }
    moreover {
      assume [?rhs in dw]
      then obtain α where 4:
        [φ α & (∀ z . φ z → z = α) & (|F, α^P|) in dw]
        using ∃ E by auto
      hence [α^P = (ιx . φ x) in dw] ∧ [(|F, α^P|) in dw]
        using hintikka[equiv-rl] &E by blast
      hence [?lhs in dw]
        using l-identity[axiom-instance, deduction, deduction]
        by blast
    }
    ultimately show ?thesis by PLM-solver
  qed

lemma russell-axiom-g[PLM]:
  [{|ιx. φ x,F|} ≡ (∃ x . φ x & (∀ z . φ z → z = x) & {|x^P, F|}) in dw]
  (is [?lhs ≡ ?rhs in dw])
  proof -
    {
      assume 1: [?lhs in dw]
      hence [∃ α. α^P = (ιx. φ x) in dw]
      using cqt-5[axiom-instance, deduction] SimpleExOrEnc.intros by blast
      then obtain α where 2: [α^P = (ιx. φ x) in dw] by (rule ∃ E)
      hence 3: [(φ α & (∀ z . φ z → z = α)) in dw]
```

using *hintikka*[*equiv-lr*] **by** *simp*
from *2* **have** $[(\iota x.\ \varphi\ x) = \alpha^P\ \ in\ dw]$
using *l-identity*[**where** $\alpha{=}\alpha^P$ **and** $\beta{=}\iota x.\ \varphi\ x$ **and** $\varphi{=}\lambda\ x\ .\ x = \alpha^P$,
axiom-instance, *deduction*, *deduction*]
id-eq-obj-1[**where** $x{=}\alpha$] **by** *auto*
hence $[\{\!|\alpha^P,\ F|\!\}\ in\ dw]$
using *1 l-identity*[**where** $\beta{=}\alpha^P$ **and** $\alpha{=}\iota x.\ \varphi\ x$ **and** $\varphi{=}\lambda\ x\ .\ \{\!|x,F|\!\}$,
axiom-instance, *deduction*, *deduction*] **by** *auto*
with *3* **have** $[(\varphi\ \alpha\ \&\ (\forall\ \ z\ .\ \varphi\ z \rightarrow z = \alpha))\ \&\ \{\!|\alpha^P,\ F|\!\}\ in\ dw]$
using $\&I$ **by** *auto*
hence $[?rhs\ in\ dw]$ **using** $\exists\,I$[**where** $\alpha{=}\alpha$] **by** (*simp add*: *identity-defs*)
}
moreover {
assume $[?rhs\ in\ dw]$
then obtain α **where** *4*:
$[\varphi\ \alpha\ \&\ (\forall\ \ z\ .\ \varphi\ z \rightarrow z = \alpha)\ \&\ \{\!|\alpha^P,\ F|\!\}\ in\ dw]$
using $\exists\,E$ **by** *auto*
hence $[\alpha^P = (\iota x\ .\ \varphi\ x)\ in\ dw] \wedge [\{\!|\alpha^P,\ F|\!\}\ in\ dw]$
using *hintikka*[*equiv-rl*] $\&E$ **by** *blast*
hence $[?lhs\ in\ dw]$
using *l-identity*[*axiom-instance*, *deduction*, *deduction*]
by *fast*
}
ultimately show *?thesis* **by** *PLM-solver*
qed

lemma *russell-axiom*[*PLM*]:
assumes *SimpleExOrEnc* ψ
shows $[\psi\ (\iota x.\ \varphi\ x) \equiv (\exists\ \ x\ .\ \varphi\ x\ \&\ (\forall\ \ z\ .\ \varphi\ z \rightarrow z = x)\ \&\ \psi\ (x^P))\ in\ dw]$
(**is** $[?lhs \equiv\ ?rhs\ in\ dw]$)
proof –
{
assume *1*: $[?lhs\ in\ dw]$
hence $[\exists\,\alpha.\ \alpha^P = (\iota x.\ \varphi\ x)\ in\ dw]$
using *cqt-5*[*axiom-instance*, *deduction*] *assms* **by** *blast*
then obtain α **where** *2*: $[\alpha^P = (\iota x.\ \varphi\ x)\ in\ dw]$ **by** (*rule* $\exists\,E$)
hence *3*: $[(\varphi\ \alpha\ \&\ (\forall\ \ z\ .\ \varphi\ z \rightarrow z = \alpha))\ in\ dw]$
using *hintikka*[*equiv-lr*] **by** *simp*
from *2* **have** $[(\iota x.\ \varphi\ x) = (\alpha^P)\ \ in\ dw]$
using *l-identity*[**where** $\alpha{=}\alpha^P$ **and** $\beta{=}\iota x.\ \varphi\ x$ **and** $\varphi{=}\lambda\ x\ .\ x = \alpha^P$,
axiom-instance, *deduction*, *deduction*]
id-eq-obj-1[**where** $x{=}\alpha$] **by** *auto*
hence $[\psi\ (\alpha^P)\ in\ dw]$
using *1 l-identity*[**where** $\beta{=}\alpha^P$ **and** $\alpha{=}\iota x.\ \varphi\ x$ **and** $\varphi{=}\lambda\ x\ .\ \psi\ x$,
axiom-instance, *deduction*, *deduction*] **by** *auto*
with *3* **have** $[\varphi\ \alpha\ \&\ (\forall\ \ z\ .\ \varphi\ z \rightarrow z = \alpha)\ \&\ \psi\ (\alpha^P)\ in\ dw]$
using $\&I$ **by** *auto*
hence $[?rhs\ in\ dw]$ **using** $\exists\,I$[**where** $\alpha{=}\alpha$] **by** (*simp add*: *identity-defs*)
}
moreover {
assume $[?rhs\ in\ dw]$
then obtain α **where** *4*:
$[\varphi\ \alpha\ \&\ (\forall\ \ z\ .\ \varphi\ z \rightarrow z = \alpha)\ \&\ \psi\ (\alpha^P)\ in\ dw]$
using $\exists\,E$ **by** *auto*
hence $[\alpha^P = (\iota x\ .\ \varphi\ x)\ in\ dw] \wedge [\psi\ (\alpha^P)\ in\ dw]$
using *hintikka*[*equiv-rl*] $\&E$ **by** *blast*
hence $[?lhs\ in\ dw]$
using *l-identity*[*axiom-instance*, *deduction*, *deduction*]

```
      by fast
    }
    ultimately show ?thesis by PLM-solver
  qed

lemma unique-exists[PLM]:
  [(∃ y . y^P = (ιx. φ x)) ≡ (∃!x . φ x) in dw]
  proof((rule ≡I, rule CP, rule-tac[2] CP))
    assume [∃ y. y^P = (ιx. φ x) in dw]
    then obtain α where
      [α^P = (ιx. φ x) in dw]
      by (rule ∃ E)
    hence [φ α & (∀ β. φ β → β = α) in dw]
      using hintikka[equiv-lr] by auto
    thus [∃!x . φ x in dw]
      unfolding exists-unique-def using ∃ I by fast
  next
    assume [∃!x . φ x in dw]
    then obtain α where
      [φ α & (∀ β. φ β → β = α) in dw]
      unfolding exists-unique-def by (rule ∃ E)
    hence [α^P = (ιx. φ x) in dw]
      using hintikka[equiv-rl] by auto
    thus [∃ y. y^P = (ιx. φ x) in dw]
      using ∃ I by fast
  qed

lemma y-in-1[PLM]:
  [x^P = (ιx . φ) → φ in dw]
  using hintikka[equiv-lr, conj1] by (rule CP)

lemma y-in-2[PLM]:
  [z^P = (ιx . φ x) → φ z in dw]
  using hintikka[equiv-lr, conj1] by (rule CP)

lemma y-in-3[PLM]:
  [(∃ y . y^P = (ιx . φ (x^P))) → φ (ιx . φ (x^P)) in dw]
  proof (rule CP)
    assume [(∃ y . y^P = (ιx . φ (x^P))) in dw]
    then obtain y where 1:
      [y^P = (ιx. φ (x^P)) in dw]
      by (rule ∃ E)
    hence [φ (y^P) in dw]
      using y-in-2[deduction] unfolding identity-ν-def by blast
    thus [φ (ιx. φ (x^P)) in dw]
      using l-identity[axiom-instance, deduction,
                        deduction] 1 by fast
  qed

lemma act-quant-nec[PLM]:
  [(∀ z . (𝓐φ z ≡ z = x)) ≡ (∀ z. 𝓐𝓐φ z ≡ z = x) in v]
  by PLM-solver

lemma equi-desc-descA-1[PLM]:
  [(x^P = (ιx . φ x)) ≡ (x^P = (ιx . 𝓐φ x)) in v]
  using descriptions[axiom-instance] apply (rule ≡E(5))
  using act-quant-nec apply (rule ≡E(5))
  using descriptions[axiom-instance]
```

by $(meson \equiv E(6)\ oth\text{-}class\text{-}taut\text{-}4\text{-}a)$

lemma *equi-desc-descA-2*[*PLM*]:
$[(\exists\ y\ .\ y^P = (\boldsymbol{\iota}x.\ \varphi\ x)) \rightarrow ((\boldsymbol{\iota}x\ .\ \varphi\ x) = (\boldsymbol{\iota}x\ .\ \boldsymbol{\mathcal{A}}\varphi\ x))\ in\ v]$
proof (*rule CP*)
assume $[\exists\, y.\ y^P = (\boldsymbol{\iota}x.\ \varphi\ x)\ in\ v]$
then obtain y **where**
$[y^P = (\boldsymbol{\iota}x.\ \varphi\ x)\ in\ v]$
by $(rule\ \exists\, E)$
moreover hence $[y^P = (\boldsymbol{\iota}x.\ \boldsymbol{\mathcal{A}}\varphi\ x)\ in\ v]$
using *equi-desc-descA-1*[*equiv-lr*] **by** *auto*
ultimately show $[(\boldsymbol{\iota}x.\ \varphi\ x) = (\boldsymbol{\iota}x.\ \boldsymbol{\mathcal{A}}\varphi\ x)\ in\ v]$
using *l-identity*[*axiom-instance*, *deduction*, *deduction*]
by *fast*
qed

lemma *equi-desc-descA-3*[*PLM*]:
assumes *SimpleExOrEnc* ψ
shows $[\psi\ (\boldsymbol{\iota}x.\ \varphi\ x) \rightarrow (\exists\ y\ .\ y^P = (\boldsymbol{\iota}x.\ \boldsymbol{\mathcal{A}}\varphi\ x))\ in\ v]$
proof (*rule CP*)
assume $[\psi\ (\boldsymbol{\iota}x.\ \varphi\ x)\ in\ v]$
hence $[\exists\, \alpha.\ \alpha^P = (\boldsymbol{\iota}x.\ \varphi\ x)\ in\ v]$
using *cqt-5*[*OF assms*, *axiom-instance*, *deduction*] **by** *auto*
then obtain α **where** $[\alpha^P = (\boldsymbol{\iota}x.\ \varphi\ x)\ in\ v]$ **by** $(rule\ \exists\, E)$
hence $[\alpha^P = (\boldsymbol{\iota}x\ .\ \boldsymbol{\mathcal{A}}\varphi\ x)\ in\ v]$
using *equi-desc-descA-1*[*equiv-lr*] **by** *auto*
thus $[\exists\, y.\ y^P = (\boldsymbol{\iota}x.\ \boldsymbol{\mathcal{A}}\varphi\ x)\ in\ v]$
using $\exists\, I$ **by** *fast*
qed

lemma *equi-desc-descA-4*[*PLM*]:
assumes *SimpleExOrEnc* ψ
shows $[\psi\ (\boldsymbol{\iota}x.\ \varphi\ x) \rightarrow ((\boldsymbol{\iota}x.\ \varphi\ x) = (\boldsymbol{\iota}x.\ \boldsymbol{\mathcal{A}}\varphi\ x))\ in\ v]$
proof (*rule CP*)
assume $[\psi\ (\boldsymbol{\iota}x.\ \varphi\ x)\ in\ v]$
hence $[\exists\, \alpha.\ \alpha^P = (\boldsymbol{\iota}x.\ \varphi\ x)\ in\ v]$
using *cqt-5*[*OF assms*, *axiom-instance*, *deduction*] **by** *auto*
then obtain α **where** $[\alpha^P = (\boldsymbol{\iota}x.\ \varphi\ x)\ in\ v]$ **by** $(rule\ \exists\, E)$
moreover hence $[\alpha^P\ = (\boldsymbol{\iota}x\ .\ \boldsymbol{\mathcal{A}}\varphi\ x)\ in\ v]$
using *equi-desc-descA-1*[*equiv-lr*] **by** *auto*
ultimately show $[(\boldsymbol{\iota}x.\ \varphi\ x)\ = (\boldsymbol{\iota}x\ .\ \boldsymbol{\mathcal{A}}\varphi\ x)\ in\ v]$
using *l-identity*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
qed

lemma *nec-hintikka-scheme*[*PLM*]:
$[(x^P = (\boldsymbol{\iota}x.\ \varphi\ x)) \equiv (\boldsymbol{\mathcal{A}}\varphi\ x\ \&\ (\forall\ z\ .\ \boldsymbol{\mathcal{A}}\varphi\ z \rightarrow z = x))\ in\ v]$
using *descriptions*[*axiom-instance*]
apply $(rule \equiv E(5))$
apply *PLM-solver*
using *id-eq-obj-1* **apply** *simp*
using *id-eq-obj-2*[*deduction*]
l-identity[**where** $\alpha{=}x$, *axiom-instance*, *deduction*, *deduction*]
unfolding *identity-*ν*-def*
apply *blast*
using *l-identity*[**where** $\alpha{=}x$, *axiom-instance*, *deduction*, *deduction*]
id-eq-2[**where** $'a{=}\nu$, *deduction*] **unfolding** *identity-*ν*-def* **by** *meson*

lemma *equiv-desc-eq*[*PLM*]:

```
assumes ⋀x.[𝓐(φ x ≡ ψ x) in v]
shows [(∀ x . ((x^P = (ιx . φ x)) ≡ (x^P = (ιx . ψ x)))) in v]
proof(rule ∀I)
  fix x
  {
    assume [x^P = (ιx . φ x) in v]
    hence 1: [𝓐φ x & (∀z. 𝓐φ z → z = x) in v]
      using nec-hintikka-scheme[equiv-lr] by auto
    hence 2: [𝓐φ x in v] ∧ [(∀z. 𝓐φ z → z = x) in v]
      using &E by blast
    {
      fix z
      {
        assume [𝓐ψ z in v]
        hence [𝓐φ z in v]
         using assms[where x=z] apply − by PLM-solver
        moreover have [𝓐φ z → z = x in v]
          using 2 cqt-1[axiom-instance,deduction] by auto
        ultimately have [z = x in v]
         using vdash-properties-10 by auto
      }
      hence [𝓐ψ z → z = x in v] by (rule CP)
    }
    hence [(∀ z . 𝓐ψ z → z = x) in v] by (rule ∀I)
    moreover have [𝓐ψ x in v]
      using 1[conj1] assms[where x=x]
      apply − by PLM-solver
    ultimately have [𝓐ψ x & (∀z. 𝓐ψ z → z = x) in v]
      by PLM-solver
    hence [x^P = (ιx. ψ x) in v]
     using nec-hintikka-scheme[where φ=ψ, equiv-rl] by auto
  }
  moreover {
    assume [x^P = (ιx . ψ x) in v]
    hence 1: [𝓐ψ x & (∀z. 𝓐ψ z → z = x) in v]
      using nec-hintikka-scheme[equiv-lr] by auto
    hence 2: [𝓐ψ x in v] ∧ [(∀z. 𝓐ψ z → z = x) in v]
      using &E by blast
    {
      fix z
      {
        assume [𝓐φ z in v]
        hence [𝓐ψ z in v]
          using assms[where x=z]
          apply − by PLM-solver
        moreover have [𝓐ψ z → z = x in v]
          using 2 cqt-1[axiom-instance,deduction] by auto
        ultimately have [z = x in v]
          using vdash-properties-10 by auto
      }
      hence [𝓐φ z → z = x in v] by (rule CP)
    }
    hence [(∀z. 𝓐φ z → z = x) in v] by (rule ∀I)
    moreover have [𝓐φ x in v]
      using 1[conj1] assms[where x=x]
      apply − by PLM-solver
    ultimately have [𝓐φ x & (∀z. 𝓐φ z → z = x) in v]
      by PLM-solver
```

```
    hence [x^P = (ιx. φ x) in v]
      using nec-hintikka-scheme[where φ=φ,equiv-rl]
      by auto
  }
  ultimately show [x^P = (ιx. φ x) ≡ (x^P) = (ιx. ψ x) in v]
    using ≡I CP by auto
qed

lemma UniqueAux:
  assumes [(𝓐φ (α::ν) & (∀ z . 𝓐(φ z) → z = α)) in v]
  shows [(∀ z . (𝓐(φ z) ≡ (z = α))) in v]
  proof –
    {
      fix z
      {
        assume [𝓐(φ z) in v]
        hence [z = α in v]
          using assms[conj2, THEN cqt-1[where α=z,
                     axiom-instance, deduction],
                   deduction] by auto
      }
      moreover {
        assume [z = α in v]
        hence [α = z in v]
          unfolding identity-ν-def
          using id-eq-obj-2[deduction] by fast
        hence [𝓐(φ z) in v] using assms[conj1]
          using l-identity[axiom-instance, deduction,
                        deduction] by fast
      }
      ultimately have [(𝓐(φ z) ≡ (z = α)) in v]
        using ≡I CP by auto
    }
    thus [(∀ z . (𝓐(φ z) ≡ (z = α))) in v]
    by (rule ∀I)
  qed

lemma nec-russell-axiom[PLM]:
  assumes SimpleExOrEnc ψ
  shows [(ψ (ιx. φ x)) ≡ (∃ x . (𝓐φ x & (∀ z . 𝓐(φ z) → z = x))
                            & ψ (x^P)) in v]
  (is [?lhs ≡ ?rhs in v])
  proof –
    {
      assume 1: [?lhs in v]
      hence [∃ α. (α^P) = (ιx. φ x) in v]
        using cqt-5[axiom-instance, deduction] assms by blast
      then obtain α where 2: [(α^P) = (ιx. φ x) in v] by (rule ∃E)
      hence [(∀ z . (𝓐(φ z) ≡ (z = α))) in v]
        using descriptions[axiom-instance, equiv-lr] by auto
      hence 3: [(𝓐φ α) & (∀ z . (𝓐(φ z) → (z = α))) in v]
        using cqt-1[where α=α and φ=λ z . (𝓐(φ z) ≡ (z = α)),
                  axiom-instance, deduction, equiv-rl]
        using id-eq-obj-1[where x=α] unfolding identity-ν-def
        using hintikka[equiv-lr] cqt-basic-2[equiv-lr,conj1]
        &I by fast
      from 2 have [(ιx. φ x) = (α^P)  in v]
        using l-identity[where β=(ιx. φ x) and φ=λ x . x = (α^P),
```

axiom-instance, *deduction*, *deduction*]
id-eq-obj-1[**where** $x{=}\alpha$] **by** *auto*
hence [$\psi\ (\alpha^P)$ *in v*]
using *1 l-identity*[**where** $\alpha{=}(\iota x.\ \varphi\ x)$ **and** $\varphi{=}\lambda\ x\ .\ \psi\ x$,
axiom-instance, *deduction*,
deduction] **by** *auto*
with *3* **have** [$(\mathcal{A}\varphi\ \alpha\ \&\ (\forall\ z\ .\ \mathcal{A}(\varphi\ z) \rightarrow (z = \alpha)))\ \&\ \psi\ (\alpha^P)$ *in v*]
using &*I* **by** *simp*
hence [*?rhs in v*]
using $\exists$ *I*[**where** $\alpha{=}\alpha$]
by (*simp add*: *identity-defs*)
}
moreover {
assume [*?rhs in v*]
then obtain α **where** *4*:
[$(\mathcal{A}\varphi\ \alpha\ \&\ (\forall\ z\ .\ \mathcal{A}(\varphi\ z) \rightarrow z = \alpha))\ \&\ \psi\ (\alpha^P)$ *in v*]
using $\exists$ *E* **by** *auto*
hence [$(\forall\ z\ .\ (\mathcal{A}(\varphi\ z) \equiv (z = \alpha)))$ *in v*]
using *UniqueAux* &*E*(*1*) **by** *auto*
hence [$(\alpha^P) = (\iota x\ .\ \varphi\ x)$ *in v*] $\wedge$ [$\psi\ (\alpha^P)$ *in v*]
using *descriptions*[*axiom-instance*, *equiv-rl*]
4[*conj2*] **by** *blast*
hence [*?lhs in v*]
using *l-identity*[*axiom-instance*, *deduction*,
deduction]
by *fast*
}
ultimately show *?thesis* **by** *PLM-solver*
qed

lemma *actual-desc-1*[*PLM*]:
[$(\exists\ y\ .\ (y^P) = (\iota x.\ \varphi\ x)) \equiv (\exists!\ x\ .\ \mathcal{A}(\varphi\ x))$ *in v*] (**is** [*?lhs* $\equiv$ *?rhs in v*])
proof –
{
assume [*?lhs in v*]
then obtain α **where**
[$((\alpha^P) = (\iota x.\ \varphi\ x))$ *in v*]
by (*rule* $\exists$ *E*)
hence [$(\!|A!,(\iota x.\ \varphi\ x)|\!)$ *in v*] $\vee$ [$(\alpha^P) =_E (\iota x.\ \varphi\ x)$ *in v*]
apply – **unfolding** *identity-defs* **by** *PLM-solver*
then obtain x **where**
[$((\mathcal{A}\varphi\ x\ \&\ (\forall\ z\ .\ \mathcal{A}(\varphi\ z) \rightarrow z = x)))$ *in v*]
using *nec-russell-axiom*[**where** $\psi{=}\lambda x\ .\ (\!|A!,x|\!)$, *equiv-lr*, *THEN* $\exists$ *E*]
using *nec-russell-axiom*[**where** $\psi{=}\lambda x\ .\ (\alpha^P) =_E x$, *equiv-lr*, *THEN* $\exists$ *E*]
using *SimpleExOrEnc.intros* **unfolding** *identity$_E$-infix-def*
by (*meson* &*E*)
hence [*?rhs in v*] **unfolding** *exists-unique-def* **by** (*rule* $\exists$ *I*)
}
moreover {
assume [*?rhs in v*]
then obtain x **where**
[$((\mathcal{A}\varphi\ x\ \&\ (\forall\ z\ .\ \mathcal{A}(\varphi\ z) \rightarrow z = x)))$ *in v*]
unfolding *exists-unique-def* **by** (*rule* $\exists$ *E*)
hence [$\forall z.\ \mathcal{A}\varphi\ z \equiv z = x$ *in v*]
using *UniqueAux* **by** *auto*
hence [$(x^P) = (\iota x.\ \varphi\ x)$ *in v*]
using *descriptions*[*axiom-instance*, *equiv-rl*] **by** *auto*
hence [*?lhs in v*] **by** (*rule* $\exists$ *I*)

```
    }
    ultimately show ?thesis
      using ≡I CP by auto
  qed

lemma actual-desc-2[PLM]:
  [(x^P) = (ιx. φ) → 𝓐φ in v]
  using nec-hintikka-scheme[equiv-lr, conj1]
  by (rule CP)

lemma actual-desc-3[PLM]:
  [(z^P) = (ιx. φ x) → 𝓐(φ z) in v]
  using nec-hintikka-scheme[equiv-lr, conj1]
  by (rule CP)

lemma actual-desc-4[PLM]:
  [(∃ y . ((y^P) = (ιx. φ (x^P)))) → 𝓐(φ (ιx. φ (x^P))) in v]
  proof (rule CP)
    assume [(∃ y . (y^P) = (ιx . φ (x^P))) in v]
    then obtain y where 1:
      [y^P = (ιx. φ (x^P)) in v]
      by (rule ∃E)
    hence [𝓐(φ (y^P)) in v] using actual-desc-3[deduction] by fast
    thus [𝓐(φ (ιx. φ (x^P))) in v]
      using l-identity[axiom-instance, deduction,
                      deduction] 1 by fast
  qed

lemma unique-box-desc-1[PLM]:
  [(∃!x . □(φ x)) → (∀ y . (y^P) = (ιx. φ x) → φ y) in v]
  proof (rule CP)
    assume [(∃!x . □(φ x)) in v]
    then obtain α where 1:
      [□φ α & (∀β. □(φ β) → β = α) in v]
      unfolding exists-unique-def by (rule ∃E)
    {
      fix y
      {
        assume [(y^P) = (ιx. φ x) in v]
        hence [𝓐φ α → α = y in v]
          using nec-hintikka-scheme[where x=y and φ=φ, equiv-lr, conj2,
                    THEN cqt-1[where α=α,axiom-instance, deduction]] by simp
        hence [α = y in v]
          using 1[conj1] nec-imp-act vdash-properties-10 by blast
        hence [φ y in v]
          using 1[conj1] qml-2[axiom-instance, deduction]
                l-identity[axiom-instance, deduction, deduction]
          by fast
      }
      hence [(y^P) = (ιx. φ x) → φ y in v]
        by (rule CP)
    }
    thus [∀ y . (y^P) = (ιx. φ x) → φ y in v]
      by (rule ∀I)
  qed

lemma unique-box-desc[PLM]:
  [(∀ x . (φ x → □(φ x))) → ((∃!x . φ x)
```

$\rightarrow (\forall\ y\ .\ (y^P = (\iota x\ .\ \varphi\ x)) \rightarrow \varphi\ y))\ in\ v]$
apply (*rule CP*, *rule CP*)
using *nec-exist-unique*[*deduction*, *deduction*]
unique-box-desc-1[*deduction*] **by** *blast*

A.9.10. Necessity

lemma *RM-1*[*PLM*]:
$(\bigwedge v.[\varphi \rightarrow \psi\ in\ v]) \Longrightarrow [\Box\varphi \rightarrow \Box\psi\ in\ v]$
using *RN qml-1*[*axiom-instance*] *vdash-properties-10* **by** *blast*

lemma *RM-1-b*[*PLM*]:
$(\bigwedge v.[\chi\ in\ v] \Longrightarrow [\varphi \rightarrow \psi\ in\ v]) \Longrightarrow ([\Box\chi\ in\ v] \Longrightarrow [\Box\varphi \rightarrow \Box\psi\ in\ v])$
using *RN-2 qml-1*[*axiom-instance*] *vdash-properties-10* **by** *blast*

lemma *RM-2*[*PLM*]:
$(\bigwedge v.[\varphi \rightarrow \psi\ in\ v]) \Longrightarrow [\Diamond\varphi \rightarrow \Diamond\psi\ in\ v]$
unfolding *diamond-def*
using *RM-1 contraposition-1* **by** *auto*

lemma *RM-2-b*[*PLM*]:
$(\bigwedge v.[\chi\ in\ v] \Longrightarrow [\varphi \rightarrow \psi\ in\ v]) \Longrightarrow ([\Box\chi\ in\ v] \Longrightarrow [\Diamond\varphi \rightarrow \Diamond\psi\ in\ v])$
unfolding *diamond-def*
using *RM-1-b contraposition-1* **by** *blast*

lemma *KBasic-1*[*PLM*]:
$[\Box\varphi \rightarrow \Box(\psi \rightarrow \varphi)\ in\ v]$
by (*simp only*: *pl-1*[*axiom-instance*] *RM-1*)
lemma *KBasic-2*[*PLM*]:
$[\Box(\neg\varphi) \rightarrow \Box(\varphi \rightarrow \psi)\ in\ v]$
by (*simp only*: *RM-1 useful-tautologies-3*)
lemma *KBasic-3*[*PLM*]:
$[\Box(\varphi\ \&\ \psi) \equiv \Box\varphi\ \&\ \Box\psi\ in\ v]$
apply (*rule* $\equiv I$)
apply (*rule CP*)
apply (*rule* $\& I$)
using *RM-1 oth-class-taut-9-a vdash-properties-6* **apply** *blast*
using *RM-1 oth-class-taut-9-b vdash-properties-6* **apply** *blast*
using *qml-1*[*axiom-instance*] *RM-1 ded-thm-cor-3 oth-class-taut-10-a*
oth-class-taut-8-b vdash-properties-10
by *blast*
lemma *KBasic-4*[*PLM*]:
$[\Box(\varphi \equiv \psi) \equiv (\Box(\varphi \rightarrow \psi)\ \&\ \Box(\psi \rightarrow \varphi))\ in\ v]$
apply (*rule* $\equiv I$)
unfolding *equiv-def* **using** *KBasic-3 PLM.CP* $\equiv E(1)$
apply *blast*
using *KBasic-3 PLM.CP* $\equiv E(2)$
by *blast*
lemma *KBasic-5*[*PLM*]:
$[(\Box(\varphi \rightarrow \psi)\ \&\ \Box(\psi \rightarrow \varphi)) \rightarrow (\Box\varphi \equiv \Box\psi)\ in\ v]$
by (*metis qml-1*[*axiom-instance*] *CP* $\& E \equiv I$ *vdash-properties-10*)
lemma *KBasic-6*[*PLM*]:
$[\Box(\varphi \equiv \psi) \rightarrow (\Box\varphi \equiv \Box\psi)\ in\ v]$
using *KBasic-4 KBasic-5* **by** (*metis equiv-def ded-thm-cor-3* $\& E(1)$)
lemma $[(\Box\varphi \equiv \Box\psi) \rightarrow \Box(\varphi \equiv \psi)\ in\ v]$
nitpick[*expect=genuine*, *user-axioms*, *card = 1*, *card i = 2*]
oops — countermodel as desired
lemma *KBasic-7*[*PLM*]:

$[(\Box\varphi \;\&\; \Box\psi) \rightarrow \Box(\varphi \equiv \psi)\ in\ v]$
proof (*rule CP*)
assume $[\Box\varphi \;\&\; \Box\psi\ in\ v]$
hence $[\Box(\psi \rightarrow \varphi)\ in\ v] \wedge [\Box(\varphi \rightarrow \psi)\ in\ v]$
using **&***E KBasic-1 vdash-properties-10* **by** *blast*
thus $[\Box(\varphi \equiv \psi)\ in\ v]$
using *KBasic-4* $\equiv$*E(2) intro-elim-1* **by** *blast*
qed

lemma *KBasic-8*[*PLM*]:
$[\Box(\varphi \;\&\; \psi) \rightarrow \Box(\varphi \equiv \psi)\ in\ v]$
using *KBasic-7 KBasic-3*
by (*metis equiv-def PLM.ded-thm-cor-3* **&***E(1)*)
lemma *KBasic-9*[*PLM*]:
$[\Box((\neg\varphi) \;\&\; (\neg\psi)) \rightarrow \Box(\varphi \equiv \psi)\ in\ v]$
proof (*rule CP*)
assume $[\Box((\neg\varphi) \;\&\; (\neg\psi))\ in\ v]$
hence $[\Box((\neg\varphi) \equiv (\neg\psi))\ in\ v]$
using *KBasic-8 vdash-properties-10* **by** *blast*
moreover have $\bigwedge v.[((\neg\varphi) \equiv (\neg\psi)) \rightarrow (\varphi \equiv \psi)\ in\ v]$
using *CP* $\equiv$*E(2) oth-class-taut-5-d* **by** *blast*
ultimately show $[\Box(\varphi \equiv \psi)\ in\ v]$
using *RM-1 PLM.vdash-properties-10* **by** *blast*
qed

lemma *rule-sub-lem-1-a*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [(\neg\psi) \equiv (\neg\chi)\ in\ v]$
using *qml-2*[*axiom-instance*] $\equiv$*E(1) oth-class-taut-5-d*
vdash-properties-10
by *blast*
lemma *rule-sub-lem-1-b*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [(\psi \rightarrow \Theta) \equiv (\chi \rightarrow \Theta)\ in\ v]$
by (*metis equiv-def contraposition-1 CP* **&***E(2)* $\equiv$*I*
$\equiv$*E(1) rule-sub-lem-1-a*)
lemma *rule-sub-lem-1-c*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [(\Theta \rightarrow \psi) \equiv (\Theta \rightarrow \chi)\ in\ v]$
by (*metis CP* $\equiv$*I* $\equiv$*E(3)* $\equiv$*E(4)* $\neg\neg$*I*
$\neg\neg$*E rule-sub-lem-1-a*)
lemma *rule-sub-lem-1-d*[*PLM*]:
$(\bigwedge x.[\Box(\psi\ x \equiv \chi\ x)\ in\ v]) \Longrightarrow [(\forall\,\alpha.\ \psi\ \alpha) \equiv (\forall\,\alpha.\ \chi\ \alpha)\ in\ v]$
by (*metis equiv-def* $\forall$ *I CP* **&***E* $\equiv$*I raa-cor-1*
vdash-properties-10 rule-sub-lem-1-a $\forall$ *E*)
lemma *rule-sub-lem-1-e*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [\boldsymbol{\mathcal{A}}\psi \equiv \boldsymbol{\mathcal{A}}\chi\ in\ v]$
using *Act-Basic-5* $\equiv$*E(1) nec-imp-act*
vdash-properties-10
by *blast*
lemma *rule-sub-lem-1-f*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [\Box\psi \equiv \Box\chi\ in\ v]$
using *KBasic-6* $\equiv$*I* $\equiv$*E(1) vdash-properties-9*
by *blast*

named-theorems *Substable-intros*

definition *Substable* :: $('a{\Rightarrow}'a{\Rightarrow}bool){\Rightarrow}('a{\Rightarrow}\mathrm{o}) \Rightarrow bool$
where *Substable* $\equiv (\lambda\ cond\ \varphi\ .\ \forall\ \psi\ \chi\ v\ .\ (cond\ \psi\ \chi) \longrightarrow [\varphi\ \psi \equiv \varphi\ \chi\ in\ v])$

```
  lemma Substable-intro-const[Substable-intros]:
    Substable cond (λ φ . Θ)
    unfolding Substable-def using oth-class-taut-4-a by blast

  lemma Substable-intro-not[Substable-intros]:
    assumes Substable cond ψ
    shows Substable cond (λ φ . ¬(ψ φ))
    using assms unfolding Substable-def
    using rule-sub-lem-1-a RN-2 ≡E oth-class-taut-5-d by metis
  lemma Substable-intro-impl[Substable-intros]:
    assumes Substable cond ψ
        and Substable cond χ
    shows Substable cond (λ φ . ψ φ → χ φ)
    using assms unfolding Substable-def
    by (metis ≡I CP intro-elim-6-a intro-elim-6-b)
  lemma Substable-intro-box[Substable-intros]:
    assumes Substable cond ψ
    shows Substable cond (λ φ . □(ψ φ))
    using assms unfolding Substable-def
    using rule-sub-lem-1-f RN by meson
  lemma Substable-intro-actual[Substable-intros]:
    assumes Substable cond ψ
    shows Substable cond (λ φ . 𝓐(ψ φ))
    using assms unfolding Substable-def
    using rule-sub-lem-1-e RN by meson
  lemma Substable-intro-all[Substable-intros]:
    assumes ∀ x . Substable cond (ψ x)
    shows Substable cond (λ φ . ∀ x . ψ x φ)
    using assms unfolding Substable-def
    by (simp add: RN rule-sub-lem-1-d)

  named-theorems Substable-Cond-defs
end

class Substable =
  fixes Substable-Cond :: 'a⇒'a⇒bool
  assumes rule-sub-nec:
    ⋀ φ ψ χ Θ v . ⟦PLM.Substable Substable-Cond φ; Substable-Cond ψ χ⟧
      ⟹ Θ [φ ψ in v] ⟹ Θ [φ χ in v]

instantiation o :: Substable
begin
  definition Substable-Cond-o where [PLM.Substable-Cond-defs]:
    Substable-Cond-o ≡ λ φ ψ . ∀ v . [φ ≡ ψ in v]
  instance proof
    interpret PLM .
    fix φ :: o ⇒ o and  ψ χ :: o and Θ :: bool ⇒ bool and v::i
    assume Substable Substable-Cond φ
    moreover assume Substable-Cond ψ χ
    ultimately have [φ ψ ≡ φ χ in v]
    unfolding Substable-def by blast
    hence [φ ψ in v] = [φ χ in v] using ≡E by blast
    moreover assume Θ [φ ψ in v]
    ultimately show Θ [φ χ in v] by simp
  qed
end

instantiation fun :: (type, Substable) Substable
```

```
begin
  definition Substable-Cond-fun where [PLM.Substable-Cond-defs]:
    Substable-Cond-fun ≡ λ φ ψ . ∀ x . Substable-Cond (φ x) (ψ x)
  instance proof
    interpret PLM .
    fix φ:: ('a ⇒ 'b) ⇒ o and  ψ χ :: 'a ⇒ 'b and Θ v
    assume Substable Substable-Cond φ
    moreover assume Substable-Cond ψ χ
    ultimately have [φ ψ ≡ φ χ in v]
      unfolding Substable-def by blast
    hence [φ ψ in v] = [φ χ in v] using ≡E by blast
    moreover assume Θ [φ ψ in v]
    ultimately show Θ [φ χ in v] by simp
  qed
end

context PLM
begin

  lemma Substable-intro-equiv[Substable-intros]:
    assumes Substable cond ψ
        and Substable cond χ
    shows Substable cond (λ φ . ψ φ ≡ χ φ)
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-conj[Substable-intros]:
    assumes Substable cond ψ
        and Substable cond χ
    shows Substable cond (λ φ . ψ φ & χ φ)
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-disj[Substable-intros]:
    assumes Substable cond ψ
        and Substable cond χ
    shows Substable cond (λ φ . ψ φ ∨ χ φ)
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-diamond[Substable-intros]:
    assumes Substable cond ψ
    shows Substable cond (λ φ . ◇(ψ φ))
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-exist[Substable-intros]:
    assumes ∀ x . Substable cond (ψ x)
    shows Substable cond (λ φ . ∃ x . ψ x φ)
    unfolding conn-defs by (simp add: assms Substable-intros)

  lemma Substable-intro-id-o[Substable-intros]:
    Substable Substable-Cond (λ φ . φ)
    unfolding Substable-def Substable-Cond-o-def by blast
  lemma Substable-intro-id-fun[Substable-intros]:
    assumes Substable Substable-Cond ψ
    shows Substable Substable-Cond (λ φ . ψ (φ x))
    using assms unfolding Substable-def Substable-Cond-fun-def
    by blast

  method PLM-subst-method for ψ::'a::Substable and χ::'a::Substable =
    (match conclusion in Θ [φ χ in v] for Θ and φ and v ⇒
      ‹(rule rule-sub-nec[where Θ=Θ and χ=χ and ψ=ψ and φ=φ and v=v],
        ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
        unfold Substable-Cond-defs)›)
```

```
method PLM-autosubst =
  (match premises in ⋀v . [ψ ≡ χ in v] for ψ and χ ⇒
    ‹ match conclusion in Θ [φ χ in v] for Θ φ and v ⇒
      ‹(rule rule-sub-nec[where Θ=Θ and χ=χ and ψ=ψ and φ=φ and v=v],
        ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
        unfold Substable-Cond-defs)› ›)

method PLM-autosubst1 =
  (match premises in ⋀v x . [ψ x ≡ χ x in v]
    for ψ::'a::type⇒o and χ::'a⇒o ⇒
    ‹ match conclusion in Θ [φ χ in v] for Θ φ and v ⇒
      ‹(rule rule-sub-nec[where Θ=Θ and χ=χ and ψ=ψ and φ=φ and v=v],
        ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
        unfold Substable-Cond-defs)› ›)

method PLM-autosubst2 =
  (match premises in ⋀v x y . [ψ x y ≡ χ x y in v]
    for ψ::'a::type⇒'a⇒o and χ::'a::type⇒'a⇒o ⇒
    ‹ match conclusion in Θ [φ χ in v] for Θ φ and v ⇒
      ‹(rule rule-sub-nec[where Θ=Θ and χ=χ and ψ=ψ and φ=φ and v=v],
        ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
        unfold Substable-Cond-defs)› ›)

method PLM-subst-goal-method for φ::'a::Substable⇒o and ψ::'a =
  (match conclusion in Θ [φ χ in v] for Θ and χ and v ⇒
    ‹(rule rule-sub-nec[where Θ=Θ and χ=χ and ψ=ψ and φ=φ and v=v],
      ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
      unfold Substable-Cond-defs)›)


lemma rule-sub-nec[PLM]:
  assumes Substable Substable-Cond φ
  shows (⋀v.[(ψ ≡ χ) in v]) ⟹ Θ [φ ψ in v] ⟹ Θ [φ χ in v]
  proof –
    assume (⋀v.[(ψ ≡ χ) in v])
    hence [φ ψ in v] = [φ χ in v]
      using assms RN unfolding Substable-def Substable-Cond-defs
      using ≡I CP ≡E(1) ≡E(2) by meson
    thus Θ [φ ψ in v] ⟹ Θ [φ χ in v] by auto
  qed

lemma rule-sub-nec1[PLM]:
  assumes Substable Substable-Cond φ
  shows (⋀v x .[(ψ x ≡ χ x) in v]) ⟹ Θ [φ ψ in v] ⟹ Θ [φ χ in v]
  proof –
    assume (⋀v x.[(ψ x ≡ χ x) in v])
    hence [φ ψ in v] = [φ χ in v]
      using assms RN unfolding Substable-def Substable-Cond-defs
      using ≡I CP ≡E(1) ≡E(2) by metis
    thus Θ [φ ψ in v] ⟹ Θ [φ χ in v] by auto
  qed

lemma rule-sub-nec2[PLM]:
  assumes Substable Substable-Cond φ
  shows (⋀v x y .[ψ x y ≡ χ x y in v]) ⟹ Θ [φ ψ in v] ⟹ Θ [φ χ in v]
  proof –
    assume (⋀v x y .[ψ x y ≡ χ x y in v])
```

hence $[\varphi\ \psi\ in\ v] = [\varphi\ \chi\ in\ v]$
using *assms RN* **unfolding** *Substable-def Substable-Cond-defs*
using $\equiv I$ *CP* $\equiv E(1)$ $\equiv E(2)$ **by** *metis*
thus $\Theta\ [\varphi\ \psi\ in\ v] \Longrightarrow \Theta\ [\varphi\ \chi\ in\ v]$ **by** *auto*
qed

lemma *rule-sub-remark-1-autosubst*:
assumes $(\bigwedge v.[(\!|A!,x|\!) \equiv (\neg(\Diamond(\!|E!,x|\!)))\ in\ v])$
and $[\neg(\!|A!,x|\!)\ in\ v]$
shows$[\neg\neg\Diamond(\!|E!,x|\!)\ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-1*:
assumes $(\bigwedge v.[(\!|A!,x|\!) \equiv (\neg(\Diamond(\!|E!,x|\!)))\ in\ v])$
and $[\neg(\!|A!,x|\!)\ in\ v]$
shows$[\neg\neg\Diamond(\!|E!,x|\!)\ in\ v]$
apply (*PLM-subst-method* $(\!|A!,x|\!)$ $(\neg(\Diamond(\!|E!,x|\!)))$)
apply (*simp add*: *assms*(*1*))
by (*simp add*: *assms*(*2*))

lemma *rule-sub-remark-2*:
assumes $(\bigwedge v.[(\!|R,x,y|\!) \equiv ((\!|R,x,y|\!)\ \&\ ((\!|Q,a|\!) \vee (\neg(\!|Q,a|\!))))\ in\ v])$
and $[p \rightarrow (\!|R,x,y|\!)\ in\ v]$
shows$[p \rightarrow ((\!|R,x,y|\!)\ \&\ ((\!|Q,a|\!) \vee (\neg(\!|Q,a|\!))))\ \ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-3-autosubst*:
assumes $(\bigwedge v\ x.[(\!|A!,x^P|\!) \equiv (\neg(\Diamond(\!|E!,x^P|\!)))\ in\ v])$
and $[\exists\ \ x\ .\ (\!|A!,x^P|\!)\ in\ v]$
shows$[\exists\ \ x\ .\ (\neg(\Diamond(\!|E!,x^P|\!)))\ \ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst1* **by** *auto*

lemma *rule-sub-remark-3*:
assumes $(\bigwedge v\ x.[(\!|A!,x^P|\!) \equiv (\neg(\Diamond(\!|E!,x^P|\!)))\ in\ v])$
and $[\exists\ \ x\ .\ (\!|A!,x^P|\!)\ in\ v]$
shows $[\exists\ \ x\ .\ (\neg(\Diamond(\!|E!,x^P|\!)))\ \ in\ v]$
apply (*PLM-subst-method* $\lambda x\ .\ (\!|A!,x^P|\!)\ \lambda x\ .\ (\neg(\Diamond(\!|E!,x^P|\!)))$)
apply (*simp add*: *assms*(*1*))
by (*simp add*: *assms*(*2*))

lemma *rule-sub-remark-4*:
assumes $\bigwedge v\ x.[(\neg(\neg(\!|P,x^P|\!))) \equiv (\!|P,x^P|\!)\ in\ v]$
and $[\boldsymbol{\mathcal{A}}(\neg(\neg(\!|P,x^P|\!)))\ in\ v]$
shows $[\boldsymbol{\mathcal{A}}(\!|P,x^P|\!)\ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst1* **by** *auto*

lemma *rule-sub-remark-5*:
assumes $\bigwedge v.[(\varphi \rightarrow \psi) \equiv ((\neg\psi) \rightarrow (\neg\varphi))\ in\ v]$
and $[\Box(\varphi \rightarrow \psi)\ in\ v]$
shows $[\Box((\neg\psi) \rightarrow (\neg\varphi))\ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-6*:
assumes $\bigwedge v.[\psi \equiv \chi\ in\ v]$
and $[\Box(\varphi \rightarrow \psi)\ in\ v]$
shows $[\Box(\varphi \rightarrow \chi)\ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-7*:
  assumes $\bigwedge v.[\varphi \equiv (\neg(\neg\varphi))$ *in* $v]$
    and $[\Box(\varphi \to \varphi)$ *in* $v]$
  shows $[\Box((\neg(\neg\varphi)) \to \varphi)$ *in* $v]$
  apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-8*:
  assumes $\bigwedge v.[\boldsymbol{\mathcal{A}}\varphi \equiv \varphi$ *in* $v]$
    and $[\Box(\boldsymbol{\mathcal{A}}\varphi)$ *in* $v]$
  shows $[\Box(\varphi)$ *in* $v]$
  apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-9*:
  assumes $\bigwedge v.[(\!|P,a|\!) \equiv ((\!|P,a|\!)\ \&\ ((\!|Q,b|\!) \lor (\neg(\!|Q,b|\!))))$ *in* $v]$
    and $[(\!|P,a|\!) = (\!|P,a|\!)$ *in* $v]$
  shows $[(\!|P,a|\!) = ((\!|P,a|\!)\ \&\ ((\!|Q,b|\!) \lor (\neg(\!|Q,b|\!))))$ *in* $v]$
   unfolding *identity-defs* **apply** (*insert assms*)
   apply *PLM-autosubst* **oops** — no match as desired

— *dr-alphabetic-rules* implicitly holds
— *dr-alphabetic-thm* implicitly holds

lemma *KBasic2-1*[*PLM*]:
  $[\Box\varphi \equiv \Box(\neg(\neg\varphi))$ *in* $v]$
  apply (*PLM-subst-method* φ $(\neg(\neg\varphi))$)
   by *PLM-solver+*

lemma *KBasic2-2*[*PLM*]:
  $[(\neg(\Box\varphi)) \equiv \Diamond(\neg\varphi)$ *in* $v]$
  unfolding *diamond-def*
  apply (*PLM-subst-method* φ $\neg(\neg\varphi)$)
   by *PLM-solver+*

lemma *KBasic2-3*[*PLM*]:
  $[\Box\varphi \equiv (\neg(\Diamond(\neg\varphi)))$ *in* $v]$
  unfolding *diamond-def*
  apply (*PLM-subst-method* φ $\neg(\neg\varphi)$)
   apply *PLM-solver*
  by (*simp add*: *oth-class-taut-4-b*)
lemmas *Df*$\Box$ = *KBasic2-3*

lemma *KBasic2-4*[*PLM*]:
  $[\Box(\neg(\varphi)) \equiv (\neg(\Diamond\varphi))$ *in* $v]$
  unfolding *diamond-def*
  by (*simp add*: *oth-class-taut-4-b*)

lemma *KBasic2-5*[*PLM*]:
  $[\Box(\varphi \to \psi) \to (\Diamond\varphi \to \Diamond\psi)$ *in* $v]$
  by (*simp only*: *CP RM-2-b*)
lemmas *K*$\Diamond$ = *KBasic2-5*

lemma *KBasic2-6*[*PLM*]:
  $[\Diamond(\varphi \lor \psi) \equiv (\Diamond\varphi \lor \Diamond\psi)$ *in* $v]$
  proof −
   have $[\Box((\neg\varphi)\ \&\ (\neg\psi)) \equiv (\Box(\neg\varphi)\ \&\ \Box(\neg\psi))$ *in* $v]$
    using *KBasic-3* **by** *blast*
   hence $[(\neg(\Diamond(\neg((\neg\varphi)\ \&\ (\neg\psi))))) \equiv (\Box(\neg\varphi)\ \&\ \Box(\neg\psi))$ *in* $v]$
    using *Df*$\Box$ **by** (*rule* $\equiv E(6)$)

```
    hence [(¬(◊(¬((¬φ) & (¬ψ))))) ≡ ((¬(◊φ)) & (¬(◊ψ))) in v]
      apply − apply (PLM-subst-method □(¬φ) ¬(◊φ))
       apply (simp add: KBasic2-4)
      apply (PLM-subst-method □(¬ψ) ¬(◊ψ))
       apply (simp add: KBasic2-4)
      unfolding diamond-def by assumption
    hence [(¬(◊(φ ∨ ψ))) ≡ ((¬(◊φ)) & (¬(◊ψ))) in v]
      apply − apply (PLM-subst-method ¬((¬φ) & (¬ψ)) φ ∨ ψ)
      using oth-class-taut-6-b[equiv-sym] by auto
    hence [(¬(¬(◊(φ ∨ ψ)))) ≡ (¬((¬(◊φ))&(¬(◊ψ)))) in v]
      by (rule oth-class-taut-5-d[equiv-lr])
    hence [◊(φ ∨ ψ) ≡ (¬((¬(◊φ)) & (¬(◊ψ)))) in v]
      apply − apply (PLM-subst-method ¬(¬(◊(φ ∨ ψ))) ◊(φ ∨ ψ))
      using oth-class-taut-4-b[equiv-sym] by auto
    thus ?thesis
      apply − apply (PLM-subst-method ¬((¬(◊φ)) & (¬(◊ψ))) (◊φ) ∨ (◊ψ))
      using oth-class-taut-6-b[equiv-sym] by auto
  qed

lemma KBasic2-7[PLM]:
  [(□φ ∨ □ψ) → □(φ ∨ ψ) in v]
  proof −
    have ⋀ v . [φ → (φ ∨ ψ) in v]
      by (metis contraposition-1 contraposition-2 useful-tautologies-3 disj-def)
    hence [□φ → □(φ ∨ ψ) in v] using RM-1 by auto
    moreover {
        have ⋀ v . [ψ → (φ ∨ ψ) in v]
          by (simp only: pl-1[axiom-instance] disj-def)
        hence [□ψ → □(φ ∨ ψ) in v]
          using RM-1 by auto
    }
    ultimately show ?thesis
      using oth-class-taut-10-d vdash-properties-10 by blast
  qed

lemma KBasic2-8[PLM]:
  [◊(φ & ψ) → (◊φ & ◊ψ) in v]
  by (metis CP RM-2 &I oth-class-taut-9-a
          oth-class-taut-9-b vdash-properties-10)

lemma KBasic2-9[PLM]:
  [◊(φ → ψ) ≡ (□φ → ◊ψ) in v]
  apply (PLM-subst-method (¬(□φ)) ∨ (◊ψ) □φ → ◊ψ)
   using oth-class-taut-5-k[equiv-sym] apply simp
  apply (PLM-subst-method (¬φ) ∨ ψ φ → ψ)
   using oth-class-taut-5-k[equiv-sym] apply simp
  apply (PLM-subst-method ◊(¬φ) ¬(□φ))
   using KBasic2-2[equiv-sym] apply simp
  using KBasic2-6 .

lemma KBasic2-10[PLM]:
  [◊(□φ) ≡ (¬(□◊(¬φ))) in v]
  unfolding diamond-def apply (PLM-subst-method φ ¬¬φ)
  using oth-class-taut-4-b oth-class-taut-4-a by auto

lemma KBasic2-11[PLM]:
  [◊◊φ ≡ (¬(□□(¬φ))) in v]
  unfolding diamond-def
```

```
  apply (PLM-subst-method □(¬φ) ¬(¬(□(¬φ))))
  using oth-class-taut-4-b oth-class-taut-4-a by auto

lemma KBasic2-12[PLM]: [□(φ ∨ ψ) → (□φ ∨ ◊ψ) in v]
  proof −
    have [□(ψ ∨ φ) → (□(¬ψ) → □φ) in v]
      using CP RM-1-b ∨E(2) by blast
    hence [□(ψ ∨ φ) → (◊ψ ∨ □φ) in v]
      unfolding diamond-def disj-def
      by (meson CP ¬¬E vdash-properties-6)
    thus ?thesis apply −
      apply (PLM-subst-method (◊ψ ∨ □φ) (□φ ∨ ◊ψ))
       apply (simp add: PLM.oth-class-taut-3-e)
      apply (PLM-subst-method (ψ ∨ φ) (φ ∨ ψ))
       apply (simp add: PLM.oth-class-taut-3-e)
      by assumption
  qed

lemma TBasic[PLM]:
  [φ → ◊φ in v]
  unfolding diamond-def
  apply (subst contraposition-1)
  apply (PLM-subst-method □¬φ ¬¬□¬φ)
   apply (simp add: PLM.oth-class-taut-4-b)
  using qml-2[where φ=¬φ, axiom-instance]
  by simp
lemmas T◊ = TBasic

lemma S5Basic-1[PLM]:
  [◊□φ → □φ in v]
  proof (rule CP)
    assume [◊□φ in v]
    hence [¬□◊¬φ in v]
      using KBasic2-10[equiv-lr] by simp
    moreover have [◊(¬φ) → □◊(¬φ) in v]
      by (simp add: qml-3[axiom-instance])
    ultimately have [¬◊¬φ in v]
      by (simp add: PLM.modus-tollens-1)
    thus [□φ in v]
      unfolding diamond-def apply −
      apply (PLM-subst-method ¬¬φ φ)
       using oth-class-taut-4-b[equiv-sym] apply simp
      unfolding diamond-def using oth-class-taut-4-b[equiv-rl]
      by simp
  qed
lemmas 5◊ = S5Basic-1

lemma S5Basic-2[PLM]:
  [□φ ≡ ◊□φ in v]
  using 5◊ T◊ ≡I by blast

lemma S5Basic-3[PLM]:
  [◊φ ≡ □◊φ in v]
  using qml-3[axiom-instance] qml-2[axiom-instance] ≡I by blast

lemma S5Basic-4[PLM]:
  [φ → □◊φ in v]
  using T◊[deduction, THEN S5Basic-3[equiv-lr]]
```

by (*rule CP*)

lemma *S5Basic-5*[*PLM*]:
[$\Diamond\Box\varphi \rightarrow \varphi$ *in v*]
using *S5Basic-2*[*equiv-rl*, *THEN qml-2*[*axiom-instance*, *deduction*]]
by (*rule CP*)
lemmas *B*$\Diamond$ = *S5Basic-5*

lemma *S5Basic-6*[*PLM*]:
[$\Box\varphi \rightarrow \Box\Box\varphi$ *in v*]
using *S5Basic-4*[*deduction*] *RM-1*[*OF S5Basic-1*, *deduction*] *CP* **by** *auto*
lemmas *4*$\Box$ = *S5Basic-6*

lemma *S5Basic-7*[*PLM*]:
[$\Box\varphi \equiv \Box\Box\varphi$ *in v*]
using *4*$\Box$ *qml-2*[*axiom-instance*] **by** (*rule* $\equiv$*I*)

lemma *S5Basic-8*[*PLM*]:
[$\Diamond\Diamond\varphi \rightarrow \Diamond\varphi$ *in v*]
using *S5Basic-6*[**where** φ=$\neg\varphi$, *THEN contraposition-1*[*THEN iffD1*], *deduction*]
KBasic2-11[*equiv-lr*] *CP* **unfolding** *diamond-def* **by** *auto*
lemmas *4*$\Diamond$ = *S5Basic-8*

lemma *S5Basic-9*[*PLM*]:
[$\Diamond\Diamond\varphi \equiv \Diamond\varphi$ *in v*]
using *4*$\Diamond$ *T*$\Diamond$ **by** (*rule* $\equiv$*I*)

lemma *S5Basic-10*[*PLM*]:
[$\Box(\varphi \vee \Box\psi) \equiv (\Box\varphi \vee \Box\psi)$ *in v*]
apply (*rule* $\equiv$*I*)
apply (*PLM-subst-goal-method* $\lambda\ \chi\ .\ \Box(\varphi \vee \Box\psi) \rightarrow (\Box\varphi \vee \chi)\ \Diamond\Box\psi$)
using *S5Basic-2*[*equiv-sym*] **apply** *simp*
using *KBasic2-12* **apply** *assumption*
apply (*PLM-subst-goal-method* $\lambda\ \chi\ .(\Box\varphi \vee \chi) \rightarrow \Box(\varphi \vee \Box\psi)\ \Box\Box\psi$)
using *S5Basic-7*[*equiv-sym*] **apply** *simp*
using *KBasic2-7* **by** *auto*

lemma *S5Basic-11*[*PLM*]:
[$\Box(\varphi \vee \Diamond\psi) \equiv (\Box\varphi \vee \Diamond\psi)$ *in v*]
apply (*rule* $\equiv$*I*)
apply (*PLM-subst-goal-method* $\lambda\ \chi\ .\ \Box(\varphi \vee \Diamond\psi) \rightarrow (\Box\varphi \vee \chi)\ \Diamond\Diamond\psi$)
using *S5Basic-9* **apply** *simp*
using *KBasic2-12* **apply** *assumption*
apply (*PLM-subst-goal-method* $\lambda\ \chi\ .(\Box\varphi \vee \chi) \rightarrow \Box(\varphi \vee \Diamond\psi)\ \Box\Diamond\psi$)
using *S5Basic-3*[*equiv-sym*] **apply** *simp*
using *KBasic2-7* **by** *assumption*

lemma *S5Basic-12*[*PLM*]:
[$\Diamond(\varphi\ \&\ \Diamond\psi) \equiv (\Diamond\varphi\ \&\ \Diamond\psi)$ *in v*]
proof –
have [$\Box((\neg\varphi) \vee \Box(\neg\psi)) \equiv (\Box(\neg\varphi) \vee \Box(\neg\psi))$ *in v*]
using *S5Basic-10* **by** *auto*
hence *1*: [$(\neg\Box((\neg\varphi) \vee \Box(\neg\psi))) \equiv \neg(\Box(\neg\varphi) \vee \Box(\neg\psi))$ *in v*]
using *oth-class-taut-5-d*[*equiv-lr*] **by** *auto*
have *2*: [$(\Diamond(\neg((\neg\varphi) \vee (\neg(\Diamond\psi))))) \equiv (\neg((\neg(\Diamond\varphi)) \vee (\neg(\Diamond\psi))))$ *in v*]
apply (*PLM-subst-method* $\Box\neg\psi\ \neg\Diamond\psi$)
using *KBasic2-4* **apply** *simp*
apply (*PLM-subst-method* $\Box\neg\varphi\ \neg\Diamond\varphi$)

```
    using KBasic2-4 apply simp
   apply (PLM-subst-method (¬□((¬φ) ∨ □(¬ψ))) (◇(¬((¬φ) ∨ (□(¬ψ))))))
    unfolding diamond-def
    apply (simp add: RN oth-class-taut-4-b rule-sub-lem-1-a rule-sub-lem-1-f)
   using 1 by assumption
  show ?thesis
   apply (PLM-subst-method ¬((¬φ) ∨ (¬◇ψ)) φ & ◇ψ)
    using oth-class-taut-6-a[equiv-sym] apply simp
   apply (PLM-subst-method ¬((¬(◇φ)) ∨ (¬◇ψ)) ◇φ & ◇ψ)
    using oth-class-taut-6-a[equiv-sym] apply simp
   using 2 by assumption
 qed

lemma S5Basic-13[PLM]:
 [◇(φ & (□ψ)) ≡ (◇φ & (□ψ)) in v]
 apply (PLM-subst-method ◇□ψ □ψ)
  using S5Basic-2[equiv-sym] apply simp
 using S5Basic-12 by simp

lemma S5Basic-14[PLM]:
 [□(φ → (□ψ)) ≡ □(◇φ → ψ) in v]
 proof (rule ≡I; rule CP)
  assume [□(φ → □ψ) in v]
  moreover {
   have ⋀v.[□(φ → □ψ) → (◇φ → ψ) in v]
    proof (rule CP)
     fix v
     assume [□(φ → □ψ) in v]
     hence [◇φ → ◇□ψ in v]
      using K◇[deduction] by auto
     thus [◇φ → ψ in v]
      using B◇ ded-thm-cor-3 by blast
    qed
   hence [□(□(φ → □ψ) → (◇φ → ψ)) in v]
    by (rule RN)
   hence [□(□(φ → □ψ)) → □((◇φ → ψ)) in v]
    using qml-1[axiom-instance, deduction] by auto
  }
  ultimately show [□(◇φ → ψ) in v]
   using S5Basic-6 CP vdash-properties-10 by meson
 next
  assume [□(◇φ → ψ) in v]
  moreover {
   fix v
   {
    assume [□(◇φ → ψ) in v]
    hence 1: [□◇φ → □ψ in v]
     using qml-1[axiom-instance, deduction] by auto
    assume [φ in v]
    hence [□◇φ in v]
     using S5Basic-4[deduction] by auto
    hence [□ψ in v]
     using 1[deduction] by auto
   }
   hence [□(◇φ → ψ) in v] ⟹ [φ → □ψ in v]
    using CP by auto
  }
  ultimately show [□(φ → □ψ) in v]
```

```
    using S5Basic-6 RN-2 vdash-properties-10 by blast
  qed

lemma sc-eq-box-box-1[PLM]:
  [□(φ → □φ) → (◇φ ≡ □φ) in v]
  proof(rule CP)
    assume 1: [□(φ → □φ) in v]
    hence [□(◇φ → φ) in v]
      using S5Basic-14[equiv-lr] by auto
    hence [◇φ → φ in v]
      using qml-2[axiom-instance, deduction] by auto
    moreover from 1 have [φ → □φ in v]
      using qml-2[axiom-instance, deduction] by auto
    ultimately have [◇φ → □φ in v]
      using ded-thm-cor-3 by auto
    moreover have [□φ → ◇φ in v]
      using qml-2[axiom-instance] T◇
      by (rule ded-thm-cor-3)
    ultimately show [◇φ ≡ □φ in v]
      by (rule ≡I)
  qed

lemma sc-eq-box-box-2[PLM]:
  [□(φ → □φ) → ((¬□φ) ≡ (□(¬φ))) in v]
  proof (rule CP)
    assume [□(φ → □φ) in v]
    hence [(¬□(¬φ)) ≡ □φ in v]
      using sc-eq-box-box-1[deduction] unfolding diamond-def by auto
    thus [((¬□φ) ≡ (□(¬φ))) in v]
      by (meson CP ≡I ≡E(3)
              ≡E(4) ¬¬I ¬¬E)
  qed

lemma sc-eq-box-box-3[PLM]:
  [(□(φ → □φ) & □(ψ → □ψ)) → ((□φ ≡ □ψ) → □(φ ≡ ψ)) in v]
  proof (rule CP)
    assume 1: [(□(φ → □φ) & □(ψ → □ψ)) in v]
    {
      assume [□φ ≡ □ψ in v]
      hence [(□φ & □ψ) ∨ ((¬(□φ)) & (¬(□ψ))) in v]
        using oth-class-taut-5-i[equiv-lr] by auto
      moreover {
        assume [□φ & □ψ in v]
        hence [□(φ ≡ ψ) in v]
          using KBasic-7[deduction] by auto
      }
      moreover {
        assume [(¬(□φ)) & (¬(□ψ)) in v]
        hence [□(¬φ) & □(¬ψ) in v]
           using 1 &E &I sc-eq-box-box-2[deduction, equiv-lr]
           by metis
        hence [□((¬φ) & (¬ψ)) in v]
          using KBasic-3[equiv-rl] by auto
        hence [□(φ ≡ ψ) in v]
          using KBasic-9[deduction] by auto
      }
      ultimately have [□(φ ≡ ψ) in v]
        using CP ∨E(1) by blast
```

```
    }
    thus [□φ ≡ □ψ → □(φ ≡ ψ) in v]
      using CP by auto
  qed

lemma derived-S5-rules-1-a[PLM]:
  assumes ⋀v. [χ in v] ⟹ [◇φ → ψ in v]
  shows [□χ in v] ⟹ [φ → □ψ in v]
  proof –
    have [□χ in v] ⟹ [□◇φ → □ψ in v]
      using assms RM-1-b by metis
    thus [□χ in v] ⟹ [φ → □ψ in v]
      using S5Basic-4 vdash-properties-10 CP by metis
  qed

lemma derived-S5-rules-1-b[PLM]:
  assumes ⋀v. [◇φ → ψ in v]
  shows [φ → □ψ in v]
  using derived-S5-rules-1-a all-self-eq-1 assms by blast

lemma derived-S5-rules-2-a[PLM]:
  assumes ⋀v. [χ in v] ⟹ [φ → □ψ in v]
  shows [□χ in v] ⟹ [◇φ → ψ in v]
  proof –
    have [□χ in v] ⟹ [◇φ → ◇□ψ in v]
      using RM-2-b assms by metis
    thus [□χ in v] ⟹ [◇φ → ψ in v]
      using B◇ vdash-properties-10 CP by metis
  qed

lemma derived-S5-rules-2-b[PLM]:
  assumes ⋀v. [φ → □ψ in v]
  shows [◇φ → ψ in v]
  using assms derived-S5-rules-2-a all-self-eq-1 by blast

lemma BFs-1[PLM]: [(∀α. □(φ α)) → □(∀α. φ α) in v]
  proof (rule derived-S5-rules-1-b)
    fix v
    {
      fix α
      have ⋀v.[(∀α . □(φ α)) → □(φ α) in v]
        using cqt-orig-1 by metis
      hence [◇(∀α. □(φ α)) → ◇□(φ α) in v]
        using RM-2 by metis
      moreover have [◇□(φ α) → (φ α) in v]
        using B◇ by auto
      ultimately have [◇(∀α. □(φ α)) → (φ α) in v]
        using ded-thm-cor-3 by auto
    }
    hence [∀ α . ◇(∀α. □(φ α)) → (φ α) in v]
      using ∀I by metis
    thus [◇(∀α. □(φ α)) → (∀α. φ α) in v]
      using cqt-orig-2[deduction] by auto
  qed
lemmas BF = BFs-1

lemma BFs-2[PLM]:
  [□(∀α. φ α) → (∀α. □(φ α)) in v]
```

```
  proof -
    {
      fix α
      {
        fix v
        have [(∀ α. φ α) → φ α in v] using cqt-orig-1 by metis
      }
      hence [□(∀ α . φ α) → □(φ α) in v] using RM-1 by auto
    }
    hence [∀ α . □(∀ α . φ α) → □(φ α) in v] using ∀ I by metis
    thus ?thesis using cqt-orig-2[deduction] by metis
  qed
lemmas CBF = BFs-2

lemma BFs-3[PLM]:
  [◊(∃ α. φ α) → (∃ α . ◊(φ α)) in v]
  proof -
    have [(∀ α. □(¬(φ α))) → □(∀ α. ¬(φ α)) in v]
      using BF by metis
    hence 1: [(¬(□(∀ α. ¬(φ α)))) → (¬(∀ α. □(¬(φ α)))) in v]
      using contraposition-1 by simp
    have 2: [◊(¬(∀ α. ¬(φ α))) → (¬(∀ α. □(¬(φ α)))) in v]
      apply (PLM-subst-method ¬□(∀ α . ¬(φ α)) ◊(¬(∀ α. ¬(φ α))))
      using KBasic2-2 1 by simp+
    have [◊(¬(∀ α. ¬(φ α))) → (∃ α . ¬(□(¬(φ α)))) in v]
      apply (PLM-subst-method ¬(∀ α. □(¬(φ α))) ∃ α. ¬(□(¬(φ α))))
       using cqt-further-2 apply metis
      using 2 by metis
    thus ?thesis
      unfolding exists-def diamond-def by auto
  qed
lemmas BF◊ = BFs-3

lemma BFs-4[PLM]:
  [(∃ α . ◊(φ α)) → ◊(∃ α. φ α) in v]
  proof -
    have 1: [□(∀ α . ¬(φ α)) → (∀ α. □(¬(φ α))) in v]
      using CBF by auto
    have 2: [(∃ α . (¬(□(¬(φ α))))) → (¬(□(∀ α. ¬(φ α)))) in v]
      apply (PLM-subst-method ¬(∀ α. □(¬(φ α))) (∃ α . (¬(□(¬(φ α))))))
       using cqt-further-2 apply blast
      using 1 using contraposition-1 by metis
    have [(∃ α . (¬(□(¬(φ α))))) → ◊(¬(∀ α . ¬(φ α))) in v]
      apply (PLM-subst-method ¬(□(∀ α. ¬(φ α))) ◊(¬(∀ α. ¬(φ α))))
       using KBasic2-2 apply blast
      using 2 by assumption
    thus ?thesis
      unfolding diamond-def exists-def by auto
  qed
lemmas CBF◊ = BFs-4

lemma sign-S5-thm-1[PLM]:
  [(∃ α. □(φ α)) → □(∃ α. φ α) in v]
  proof (rule CP)
    assume [∃ α . □(φ α) in v]
    then obtain τ where [□(φ τ) in v]
      by (rule ∃ E)
    moreover {
```

```
      fix v
      assume [φ τ in v]
      hence [∃ α . φ α in v]
        by (rule ∃I)
    }
    ultimately show [□(∃ α . φ α) in v]
      using RN-2 by blast
  qed
lemmas Buridan = sign-S5-thm-1

lemma sign-S5-thm-2[PLM]:
  [◇(∀ α . φ α) → (∀ α . ◇(φ α)) in v]
  proof –
    {
      fix α
      {
        fix v
        have [(∀ α . φ α) → φ α in v]
          using cqt-orig-1 by metis
      }
      hence [◇(∀ α . φ α) → ◇(φ α) in v]
        using RM-2 by metis
    }
    hence [∀ α . ◇(∀ α . φ α) → ◇(φ α) in v]
      using ∀I by metis
    thus ?thesis
      using cqt-orig-2[deduction] by metis
  qed
lemmas Buridan◇ = sign-S5-thm-2

lemma sign-S5-thm-3[PLM]:
  [◇(∃ α . φ α & ψ α) → ◇((∃ α . φ α) & (∃ α . ψ α)) in v]
  by (simp only: RM-2 cqt-further-5)

lemma sign-S5-thm-4[PLM]:
  [((□(∀ α. φ α → ψ α)) & (□(∀ α . ψ α → χ α))) → □(∀α. φ α → χ α) in v]
  proof (rule CP)
    assume [□(∀α. φ α → ψ α) & □(∀α. ψ α → χ α) in v]
    hence [□((∀α. φ α → ψ α) & (∀α. ψ α → χ α)) in v]
      using KBasic-3[equiv-rl] by blast
    moreover {
      fix v
      assume [((∀α. φ α → ψ α) & (∀α. ψ α → χ α)) in v]
      hence [(∀ α . φ α → χ α) in v]
        using cqt-basic-9[deduction] by blast
    }
    ultimately show [□(∀α. φ α → χ α) in v]
      using RN-2 by blast
  qed

lemma sign-S5-thm-5[PLM]:
  [((□(∀α. φ α ≡ ψ α)) & (□(∀α. ψ α ≡ χ α))) → (□(∀α. φ α ≡ χ α)) in v]
  proof (rule CP)
    assume [□(∀α. φ α ≡ ψ α) & □(∀α. ψ α ≡ χ α) in v]
    hence [□((∀α. φ α ≡ ψ α) & (∀α. ψ α ≡ χ α)) in v]
      using KBasic-3[equiv-rl] by blast
    moreover {
      fix v
```

```
    assume [((∀α. φ α ≡ ψ α) & (∀α. ψ α ≡ χ α)) in v]
    hence [(∀ α . φ α ≡ χ α) in v]
      using cqt-basic-10[deduction] by blast
  }
  ultimately show [□(∀α. φ α ≡ χ α) in v]
    using RN-2 by blast
qed

lemma id-nec2-1[PLM]:
  [◊((α::'a::id-eq) = β) ≡ (α = β) in v]
  apply (rule ≡I; rule CP)
   using id-nec[equiv-lr] derived-S5-rules-2-b CP modus-ponens apply blast
  using T◊[deduction] by auto

lemma id-nec2-2-Aux:
  [(◊φ) ≡ ψ in v] ⟹ [(¬ψ) ≡ □(¬φ) in v]
  proof –
    assume [(◊φ) ≡ ψ in v]
    moreover have ⋀φ ψ. [(¬φ) ≡ ψ in v] ⟹ [(¬ψ) ≡ φ in v]
      by PLM-solver
    ultimately show ?thesis
      unfolding diamond-def by blast
  qed

lemma id-nec2-2[PLM]:
  [((α::'a::id-eq) ≠ β) ≡ □(α ≠ β) in v]
  using id-nec2-1[THEN id-nec2-2-Aux] by auto

lemma id-nec2-3[PLM]:
  [(◊((α::'a::id-eq) ≠ β)) ≡ (α ≠ β) in v]
  using T◊ ≡I id-nec2-2[equiv-lr]
        CP derived-S5-rules-2-b by metis

lemma exists-desc-box-1[PLM]:
  [(∃ y . (y^P) = (ιx. φ x)) → (∃ y . □((y^P) = (ιx. φ x))) in v]
  proof (rule CP)
    assume [∃y. (y^P) = (ιx. φ x) in v]
    then obtain y where [(y^P) = (ιx. φ x) in v]
      by (rule ∃E)
    hence [□(y^P = (ιx. φ x)) in v]
      using l-identity[axiom-instance, deduction, deduction]
            cqt-1[axiom-instance] all-self-eq-2[where 'a=ν]
            modus-ponens unfolding identity-ν-def by fast
    thus [∃y. □((y^P) = (ιx. φ x)) in v]
      by (rule ∃I)
  qed

lemma exists-desc-box-2[PLM]:
  [(∃ y . (y^P) = (ιx. φ x)) → □(∃ y .((y^P) = (ιx. φ x))) in v]
  using exists-desc-box-1 Buridan ded-thm-cor-3 by fast

lemma en-eq-1[PLM]:
  [◊⦃x,F⦄ ≡ □⦃x,F⦄ in v]
  using encoding[axiom-instance] RN
        sc-eq-box-box-1 modus-ponens by blast
lemma en-eq-2[PLM]:
  [⦃x,F⦄ ≡ □⦃x,F⦄ in v]
  using encoding[axiom-instance] qml-2[axiom-instance] by (rule ≡I)
```

lemma *en-eq-3*[*PLM*]:
$[\Diamond\{\!\{x,F\}\!\} \equiv \{\!\{x,F\}\!\}\ in\ v]$
using *encoding*[*axiom-instance*] *derived-S5-rules-2-b* $\equiv I$ $T\Diamond$ **by** *auto*
lemma *en-eq-4*[*PLM*]:
$[(\{\!\{x,F\}\!\} \equiv \{\!\{y,G\}\!\}) \equiv (\Box\{\!\{x,F\}\!\} \equiv \Box\{\!\{y,G\}\!\})\ in\ v]$
by (*metis CP en-eq-2* $\equiv I$ $\equiv E(1)$ $\equiv E(2)$)
lemma *en-eq-5*[*PLM*]:
$[\Box(\{\!\{x,F\}\!\} \equiv \{\!\{y,G\}\!\}) \equiv (\Box\{\!\{x,F\}\!\} \equiv \Box\{\!\{y,G\}\!\})\ in\ v]$
using $\equiv I$ *KBasic-6 encoding*[*axiom-necessitation*, *axiom-instance*]
sc-eq-box-box-3[*deduction*] **&***I* **by** *simp*
lemma *en-eq-6*[*PLM*]:
$[(\{\!\{x,F\}\!\} \equiv \{\!\{y,G\}\!\}) \equiv \Box(\{\!\{x,F\}\!\} \equiv \{\!\{y,G\}\!\})\ in\ v]$
using *en-eq-4 en-eq-5 oth-class-taut-4-a* $\equiv E(6)$ **by** *meson*
lemma *en-eq-7*[*PLM*]:
$[(\neg\{\!\{x,F\}\!\}) \equiv \Box(\neg\{\!\{x,F\}\!\})\ in\ v]$
using *en-eq-3*[*THEN id-nec2-2-Aux*] **by** *blast*
lemma *en-eq-8*[*PLM*]:
$[\Diamond(\neg\{\!\{x,F\}\!\}) \equiv (\neg\{\!\{x,F\}\!\})\ in\ v]$
unfolding *diamond-def* **apply** (*PLM-subst-method* $\{\!\{x,F\}\!\}$ $\neg\neg\{\!\{x,F\}\!\}$)
using *oth-class-taut-4-b* **apply** *simp*
apply (*PLM-subst-method* $\{\!\{x,F\}\!\}$ $\Box\{\!\{x,F\}\!\}$)
using *en-eq-2* **apply** *simp*
using *oth-class-taut-4-a* **by** *assumption*
lemma *en-eq-9*[*PLM*]:
$[\Diamond(\neg\{\!\{x,F\}\!\}) \equiv \Box(\neg\{\!\{x,F\}\!\})\ in\ v]$
using *en-eq-8 en-eq-7* $\equiv E(5)$ **by** *blast*
lemma *en-eq-10*[*PLM*]:
$[\boldsymbol{\mathcal{A}}\{\!\{x,F\}\!\} \equiv \{\!\{x,F\}\!\}\ in\ v]$
apply (*rule* $\equiv I$)
using *encoding*[*axiom-actualization*, *axiom-instance*,
THEN logic-actual-nec-2[*axiom-instance*, *equiv-lr*],
deduction, *THEN qml-act-2*[*axiom-instance*, *equiv-rl*],
THEN en-eq-2[*equiv-rl*]] *CP*
apply *simp*
using *encoding*[*axiom-instance*] *nec-imp-act ded-thm-cor-3* **by** *blast*

A.9.11. The Theory of Relations

lemma *beta-equiv-eq-1-1*[*PLM*]:
assumes *IsProperInX* φ
and *IsProperInX* ψ
and $\bigwedge x.[\varphi\ (x^P) \equiv \psi\ (x^P)\ in\ v]$
shows $[(\!|\boldsymbol{\lambda}\ y.\ \varphi\ (y^P),\ x^P|\!) \equiv (\!|\boldsymbol{\lambda}\ y.\ \psi\ (y^P),\ x^P|\!)\ in\ v]$
using *lambda-predicates-2-1*[*OF assms*(*1*), *axiom-instance*]
using *lambda-predicates-2-1*[*OF assms*(*2*), *axiom-instance*]
using *assms*(*3*) **by** (*meson* $\equiv E(6)$ *oth-class-taut-4-a*)

lemma *beta-equiv-eq-1-2*[*PLM*]:
assumes *IsProperInXY* φ
and *IsProperInXY* ψ
and $\bigwedge x\ y.[\varphi\ (x^P)\ (y^P) \equiv \psi\ (x^P)\ (y^P)\ in\ v]$
shows $[(\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \varphi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \psi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)\ in\ v]$
using *lambda-predicates-2-2*[*OF assms*(*1*), *axiom-instance*]
using *lambda-predicates-2-2*[*OF assms*(*2*), *axiom-instance*]
using *assms*(*3*) **by** (*meson* $\equiv E(6)$ *oth-class-taut-4-a*)

lemma *beta-equiv-eq-1-3*[*PLM*]:

assumes *IsProperInXYZ* φ
and *IsProperInXYZ* ψ
and $\bigwedge x\ y\ z.[\varphi\ (x^P)\ (y^P)\ (z^P) \equiv \psi\ (x^P)\ (y^P)\ (z^P)\ \mathit{in}\ v]$
shows $[(\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \psi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)\ \mathit{in}\ v]$
using *lambda-predicates-2-3*[*OF assms*(*1*), *axiom-instance*]
using *lambda-predicates-2-3*[*OF assms*(*2*), *axiom-instance*]
using *assms*(*3*) **by** (*meson* $\equiv$*E*(*6*) *oth-class-taut-4-a*)

lemma *beta-equiv-eq-2-1*[*PLM*]:
assumes *IsProperInX* φ
and *IsProperInX* ψ
shows $[(\Box(\forall\ \ x\ .\ \varphi\ (x^P) \equiv \psi\ (x^P))) \rightarrow$
$(\Box(\forall\ \ x\ .\ (\!|\boldsymbol{\lambda}\ y.\ \varphi\ (y^P),\ x^P|\!) \equiv (\!|\boldsymbol{\lambda}\ y.\ \psi\ (y^P),\ x^P|\!)))\ \mathit{in}\ v]$
apply (*rule qml-1*[*axiom-instance*, *deduction*])
apply (*rule RN*)
proof (*rule CP*, *rule* $\forall$*I*)
fix *v x*
assume $[\forall x.\ \varphi\ (x^P) \equiv \psi\ (x^P)\ \mathit{in}\ v]$
hence $\bigwedge x.[\varphi\ (x^P) \equiv \psi\ (x^P)\ \mathit{in}\ v]$
by *PLM-solver*
thus $[(\!|\boldsymbol{\lambda}\ y.\ \varphi\ (y^P),\ x^P|\!) \equiv (\!|\boldsymbol{\lambda}\ y.\ \psi\ (y^P),\ x^P|\!)\ \mathit{in}\ v]$
using *assms beta-equiv-eq-1-1* **by** *auto*
qed

lemma *beta-equiv-eq-2-2*[*PLM*]:
assumes *IsProperInXY* φ
and *IsProperInXY* ψ
shows $[(\Box(\forall\ \ x\ y\ .\ \varphi\ (x^P)\ (y^P) \equiv \psi\ (x^P)\ (y^P))) \rightarrow$
$(\Box(\forall\ \ x\ y\ .\ (\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \varphi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \psi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)))\ \mathit{in}\ v]$
apply (*rule qml-1*[*axiom-instance*, *deduction*])
apply (*rule RN*)
proof (*rule CP*, *rule* $\forall$*I*, *rule* $\forall$*I*)
fix *v x y*
assume $[\forall x\ y.\ \varphi\ (x^P)\ (y^P) \equiv \psi\ (x^P)\ (y^P)\ \mathit{in}\ v]$
hence $(\bigwedge x\ y.[\varphi\ (x^P)\ (y^P) \equiv \psi\ (x^P)\ (y^P)\ \mathit{in}\ v])$
by (*meson* $\forall$*E*)
thus $[(\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \varphi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \psi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)\ \mathit{in}\ v]$
using *assms beta-equiv-eq-1-2* **by** *auto*
qed

lemma *beta-equiv-eq-2-3*[*PLM*]:
assumes *IsProperInXYZ* φ
and *IsProperInXYZ* ψ
shows $[(\Box(\forall\ \ x\ y\ z\ .\ \varphi\ (x^P)\ (y^P)\ (z^P) \equiv \psi\ (x^P)\ (y^P)\ (z^P))) \rightarrow$
$(\Box(\forall\ \ x\ y\ z\ .\ (\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \psi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)))\ \mathit{in}\ v]$
apply (*rule qml-1*[*axiom-instance*, *deduction*])
apply (*rule RN*)
proof (*rule CP*, *rule* $\forall$*I*, *rule* $\forall$*I*, *rule* $\forall$*I*)
fix *v x y z*
assume $[\forall x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P) \equiv \psi\ (x^P)\ (y^P)\ (z^P)\ \mathit{in}\ v]$
hence $(\bigwedge x\ y\ z.[\varphi\ (x^P)\ (y^P)\ (z^P) \equiv \psi\ (x^P)\ (y^P)\ (z^P)\ \mathit{in}\ v])$
by (*meson* $\forall$*E*)
thus $[(\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \psi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)\ \mathit{in}\ v]$

using *assms beta-equiv-eq-1-3* **by** *auto*
qed

lemma *beta-C-meta-1*[*PLM*]:
assumes *IsProperInX* φ
shows $[\llparenthesis\boldsymbol{\lambda}\ y.\ \varphi\ (y^P),\ x^P\rrparenthesis \equiv \varphi\ (x^P)\ \mathit{in}\ v]$
using *lambda-predicates-2-1*[*OF assms*, *axiom-instance*] **by** *auto*

lemma *beta-C-meta-2*[*PLM*]:
assumes *IsProperInXY* φ
shows $[\llparenthesis\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \varphi\ (x^P)\ (y^P)),\ x^P,\ y^P\rrparenthesis \equiv \varphi\ (x^P)\ (y^P)\ \mathit{in}\ v]$
using *lambda-predicates-2-2*[*OF assms*, *axiom-instance*] **by** *auto*

lemma *beta-C-meta-3*[*PLM*]:
assumes *IsProperInXYZ* φ
shows $[\llparenthesis\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P\rrparenthesis \equiv \varphi\ (x^P)\ (y^P)\ (z^P)\ \mathit{in}\ v]$
using *lambda-predicates-2-3*[*OF assms*, *axiom-instance*] **by** *auto*

lemma *relations-1*[*PLM*]:
assumes *IsProperInX* φ
shows $[\exists\ F.\ \Box(\forall\ x.\ \llparenthesis F,x^P\rrparenthesis \equiv \varphi\ (x^P))\ \mathit{in}\ v]$
using *assms* **apply** − **by** *PLM-solver*

lemma *relations-2*[*PLM*]:
assumes *IsProperInXY* φ
shows $[\exists\ F.\ \Box(\forall\ x\ y.\ \llparenthesis F,x^P,y^P\rrparenthesis \equiv \varphi\ (x^P)\ (y^P))\ \mathit{in}\ v]$
using *assms* **apply** − **by** *PLM-solver*

lemma *relations-3*[*PLM*]:
assumes *IsProperInXYZ* φ
shows $[\exists\ F.\ \Box(\forall\ x\ y\ z.\ \llparenthesis F,x^P,y^P,z^P\rrparenthesis \equiv \varphi\ (x^P)\ (y^P)\ (z^P))\ \mathit{in}\ v]$
using *assms* **apply** − **by** *PLM-solver*

lemma *prop-equiv*[*PLM*]:
shows $[(\forall\ x\ .\ (\{\!\!\{x^P,F\}\!\!\} \equiv \{\!\!\{x^P,G\}\!\!\})) \to F = G\ \mathit{in}\ v]$
proof (*rule CP*)
assume *1*: $[\forall\, x.\ \{\!\!\{x^P,F\}\!\!\} \equiv \{\!\!\{x^P,G\}\!\!\}\ \mathit{in}\ v]$
{
fix x
have $[\{\!\!\{x^P,F\}\!\!\} \equiv \{\!\!\{x^P,G\}\!\!\}\ \mathit{in}\ v]$
using *1* **by** (*rule* $\forall\,E$)
hence $[\Box(\{\!\!\{x^P,F\}\!\!\} \equiv \{\!\!\{x^P,G\}\!\!\})\ \mathit{in}\ v]$
using *PLM.en-eq-6* $\equiv E(1)$ **by** *blast*
}
hence $[\forall\, x.\ \Box(\{\!\!\{x^P,F\}\!\!\} \equiv \{\!\!\{x^P,G\}\!\!\})\ \mathit{in}\ v]$
by (*rule* $\forall\,I$)
thus $[F = G\ \mathit{in}\ v]$
unfolding *identity-defs*
by (*rule BF*[*deduction*])
qed

lemma *propositions-lemma-1*[*PLM*]:
$[\boldsymbol{\lambda}^0\ \varphi = \varphi\ \mathit{in}\ v]$
using *lambda-predicates-3-0*[*axiom-instance*] **.**

lemma *propositions-lemma-2*[*PLM*]:
$[\boldsymbol{\lambda}^0\ \varphi \equiv \varphi\ \mathit{in}\ v]$
using *lambda-predicates-3-0*[*axiom-instance*, *THEN id-eq-prop-prop-8-b*[*deduction*]]

```
  apply (rule l-identity[axiom-instance, deduction, deduction])
  by PLM-solver

lemma propositions-lemma-4[PLM]:
  assumes ⋀x.[𝓐(φ x ≡ ψ x) in v]
  shows [(χ::κ⇒o) (ιx. φ x) = χ (ιx. ψ x) in v]
  proof –
    have [λ⁰ (χ (ιx. φ x)) = λ⁰ (χ (ιx. ψ x)) in v]
      using assms lambda-predicates-4-0[axiom-instance]
      by blast
    hence [(χ (ιx. φ x)) = λ⁰ (χ (ιx. ψ x)) in v]
      using propositions-lemma-1[THEN id-eq-prop-prop-8-b[deduction]]
            id-eq-prop-prop-9-b[deduction] &I
      by blast
    thus ?thesis
      using propositions-lemma-1 id-eq-prop-prop-9-b[deduction] &I
      by blast
  qed

lemma propositions[PLM]:
  [∃ p . □(p ≡ p') in v]
  by PLM-solver

lemma pos-not-equiv-then-not-eq[PLM]:
  [◇(¬(∀ x. ⦇F,x^P⦈ ≡ ⦇G,x^P⦈)) → F ≠ G in v]
  unfolding diamond-def
  proof (subst contraposition-1[symmetric], rule CP)
    assume [F = G in v]
    thus [□(¬(¬(∀ x. ⦇F,x^P⦈ ≡ ⦇G,x^P⦈)))) in v]
      apply (rule l-identity[axiom-instance, deduction, deduction])
      by PLM-solver
  qed

lemma thm-relation-negation-1-1[PLM]:
  [⦇F⁻, x^P⦈ ≡ ¬⦇F, x^P⦈ in v]
  unfolding propnot-defs
  apply (rule lambda-predicates-2-1[axiom-instance])
  by show-proper

lemma thm-relation-negation-1-2[PLM]:
  [⦇F⁻, x^P, y^P⦈ ≡ ¬⦇F, x^P, y^P⦈ in v]
  unfolding propnot-defs
  apply (rule lambda-predicates-2-2[axiom-instance])
  by show-proper

lemma thm-relation-negation-1-3[PLM]:
  [⦇F⁻, x^P, y^P, z^P⦈ ≡ ¬⦇F, x^P, y^P, z^P⦈ in v]
  unfolding propnot-defs
  apply (rule lambda-predicates-2-3[axiom-instance])
  by show-proper

lemma thm-relation-negation-2-1[PLM]:
  [(¬⦇F⁻, x^P⦈) ≡ ⦇F, x^P⦈ in v]
  using thm-relation-negation-1-1[THEN oth-class-taut-5-d[equiv-lr]]
  apply – by PLM-solver

lemma thm-relation-negation-2-2[PLM]:
  [(¬⦇F⁻, x^P, y^P⦈) ≡ ⦇F, x^P, y^P⦈ in v]
```

using *thm-relation-negation-1-2*[*THEN oth-class-taut-5-d*[*equiv-lr*]]
apply − **by** *PLM-solver*

lemma *thm-relation-negation-2-3*[*PLM*]:
$[(\neg(\!|F^-,\ x^P,\ y^P,\ z^P|\!)) \equiv (\!|F,\ x^P,\ y^P,\ z^P|\!)\ in\ v]$
using *thm-relation-negation-1-3*[*THEN oth-class-taut-5-d*[*equiv-lr*]]
apply − **by** *PLM-solver*

lemma *thm-relation-negation-3*[*PLM*]:
$[(p)^- \equiv \neg p\ in\ v]$
unfolding *propnot-defs*
using *propositions-lemma-2* **by** *simp*

lemma *thm-relation-negation-4*[*PLM*]:
$[(\neg((p{::}\mathrm{o})^-)) \equiv p\ in\ v]$
using *thm-relation-negation-3*[*THEN oth-class-taut-5-d*[*equiv-lr*]]
apply − **by** *PLM-solver*

lemma *thm-relation-negation-5-1*[*PLM*]:
$[(F{::}\Pi_1) \neq (F^-)\ in\ v]$
using *id-eq-prop-prop-2*[*deduction*]
l-identity[**where** $\varphi{=}\lambda\ G\ .\ (\!|G{,}x^P|\!) \equiv (\!|F^-{,}x^P|\!)$, *axiom-instance*,
deduction, *deduction*]
oth-class-taut-4-a thm-relation-negation-1-1 $\equiv E(5)$
oth-class-taut-1-b modus-tollens-1 CP
by *meson*

lemma *thm-relation-negation-5-2*[*PLM*]:
$[(F{::}\Pi_2) \neq (F^-)\ in\ v]$
using *id-eq-prop-prop-5-a*[*deduction*]
l-identity[**where** $\varphi{=}\lambda\ G\ .\ (\!|G{,}x^P{,}y^P|\!) \equiv (\!|F^-{,}x^P{,}y^P|\!)$, *axiom-instance*,
deduction, *deduction*]
oth-class-taut-4-a thm-relation-negation-1-2 $\equiv E(5)$
oth-class-taut-1-b modus-tollens-1 CP
by *meson*

lemma *thm-relation-negation-5-3*[*PLM*]:
$[(F{::}\Pi_3) \neq (F^-)\ in\ v]$
using *id-eq-prop-prop-5-b*[*deduction*]
l-identity[**where** $\varphi{=}\lambda\ G\ .\ (\!|G{,}x^P{,}y^P{,}z^P|\!) \equiv (\!|F^-{,}x^P{,}y^P{,}z^P|\!)$,
axiom-instance, *deduction*, *deduction*]
oth-class-taut-4-a thm-relation-negation-1-3 $\equiv E(5)$
oth-class-taut-1-b modus-tollens-1 CP
by *meson*

lemma *thm-relation-negation-6*[*PLM*]:
$[(p{::}\mathrm{o}) \neq (p^-)\ in\ v]$
using *id-eq-prop-prop-8-b*[*deduction*]
l-identity[**where** $\varphi{=}\lambda\ G\ .\ G \equiv (p^-)$, *axiom-instance*,
deduction, *deduction*]
oth-class-taut-4-a thm-relation-negation-3 $\equiv E(5)$
oth-class-taut-1-b modus-tollens-1 CP
by *meson*

lemma *thm-relation-negation-7*[*PLM*]:
$[((p{::}\mathrm{o})^-) = \neg p\ in\ v]$
unfolding *propnot-defs* **using** *propositions-lemma-1* **by** *simp*

```
lemma thm-relation-negation-8[PLM]:
  [(p::o) ≠ ¬p in v]
  unfolding propnot-defs
  using id-eq-prop-prop-8-b[deduction]
      l-identity[where φ=λ G . G ≡ ¬(p), axiom-instance,
                deduction, deduction]
      oth-class-taut-4-a oth-class-taut-1-b
      modus-tollens-1 CP
  by meson

lemma thm-relation-negation-9[PLM]:
  [((p::o) = q) → ((¬p) = (¬q)) in v]
  using l-identity[where α=p and β=q and φ=λ x . (¬p) = (¬x),
                axiom-instance, deduction]
      id-eq-prop-prop-7-b using CP modus-ponens by blast

lemma thm-relation-negation-10[PLM]:
  [((p::o) = q) → ((p⁻) = (q⁻)) in v]
  using l-identity[where α=p and β=q and φ=λ x . (p⁻) = (x⁻),
                axiom-instance, deduction]
      id-eq-prop-prop-7-b using CP modus-ponens by blast

lemma thm-cont-prop-1[PLM]:
  [NonContingent (F::Π₁) ≡ NonContingent (F⁻) in v]
  proof (rule ≡I; rule CP)
    assume [NonContingent F in v]
    hence [□(∀ x.⦇F,xᴾ⦈) ∨ □(∀ x.¬⦇F,xᴾ⦈) in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs .
    hence [□(∀ x. ¬⦇F⁻,xᴾ⦈) ∨ □(∀ x. ¬⦇F,xᴾ⦈) in v]
      apply −
      apply (PLM-subst-method λ x . ⦇F,xᴾ⦈ λ x . ¬⦇F⁻,xᴾ⦈)
      using thm-relation-negation-2-1[equiv-sym] by auto
    hence [□(∀ x. ¬⦇F⁻,xᴾ⦈) ∨ □(∀ x. ⦇F⁻,xᴾ⦈) in v]
      apply −
      apply (PLM-subst-goal-method
            λ φ . □(∀ x. ¬⦇F⁻,xᴾ⦈) ∨ □(∀ x. φ x) λ x . ¬⦇F,xᴾ⦈)
      using thm-relation-negation-1-1[equiv-sym] by auto
    hence [□(∀ x. ⦇F⁻,xᴾ⦈) ∨ □(∀ x. ¬⦇F⁻,xᴾ⦈) in v]
      by (rule oth-class-taut-3-e[equiv-lr])
    thus [NonContingent (F⁻) in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs .
  next
    assume [NonContingent (F⁻) in v]
    hence [□(∀ x. ¬⦇F⁻,xᴾ⦈) ∨ □(∀ x. ⦇F⁻,xᴾ⦈) in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs
      by (rule oth-class-taut-3-e[equiv-lr])
    hence [□(∀ x.⦇F,xᴾ⦈) ∨ □(∀ x.⦇F⁻,xᴾ⦈) in v]
      apply −
      apply (PLM-subst-method  λ x . ¬⦇F⁻,xᴾ⦈ λ x . ⦇F,xᴾ⦈)
      using thm-relation-negation-2-1 by auto
    hence [□(∀ x. ⦇F,xᴾ⦈) ∨ □(∀ x. ¬⦇F,xᴾ⦈) in v]
      apply −
      apply (PLM-subst-method λ x . ⦇F⁻,xᴾ⦈ λ x . ¬⦇F,xᴾ⦈)
      using thm-relation-negation-1-1 by auto
    thus [NonContingent F in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs .
  qed
```

```
lemma thm-cont-prop-2[PLM]:
  [Contingent F ≡ ◇(∃ x . ⦇F,x^P⦈) & ◇(∃ x . ¬⦇F,x^P⦈) in v]
  proof (rule ≡I; rule CP)
    assume [Contingent F in v]
    hence [¬(□(∀ x.⦇F,x^P⦈) ∨ □(∀ x.¬⦇F,x^P⦈)) in v]
      unfolding Contingent-def Necessary-defs Impossible-defs .
    hence [(¬□(∀ x.⦇F,x^P⦈)) & (¬□(∀ x.¬⦇F,x^P⦈)) in v]
      by (rule oth-class-taut-6-d[equiv-lr])
    hence [(◇¬(∀ x.¬⦇F,x^P⦈)) & (◇¬(∀ x.⦇F,x^P⦈)) in v]
      using KBasic2-2[equiv-lr] &I &E by meson
    thus [(◇(∃ x.⦇F,x^P⦈)) & (◇(∃ x. ¬⦇F,x^P⦈)) in v]
      unfolding exists-def apply –
      apply (PLM-subst-method λ x . ⦇F,x^P⦈ λ x . ¬¬⦇F,x^P⦈)
      using oth-class-taut-4-b by auto
  next
    assume [(◇(∃ x.⦇F,x^P⦈)) & (◇(∃ x. ¬⦇F,x^P⦈)) in v]
    hence [(◇¬(∀ x.¬⦇F,x^P⦈)) & (◇¬(∀ x.⦇F,x^P⦈)) in v]
      unfolding exists-def apply –
      apply (PLM-subst-goal-method
             λ φ . (◇¬(∀ x.¬⦇F,x^P⦈)) & (◇¬(∀ x. φ x)) λ x . ¬¬⦇F,x^P⦈)
      using oth-class-taut-4-b[equiv-sym] by auto
    hence [(¬□(∀ x.⦇F,x^P⦈)) & (¬□(∀ x.¬⦇F,x^P⦈)) in v]
      using KBasic2-2[equiv-rl] &I &E by meson
    hence [¬(□(∀ x.⦇F,x^P⦈) ∨ □(∀ x.¬⦇F,x^P⦈)) in v]
      by (rule oth-class-taut-6-d[equiv-rl])
    thus [Contingent F in v]
      unfolding Contingent-def Necessary-defs Impossible-defs .
  qed

lemma thm-cont-prop-3[PLM]:
  [Contingent (F::Π₁) ≡ Contingent (F⁻) in v]
  using thm-cont-prop-1
  unfolding NonContingent-def Contingent-def
  by (rule oth-class-taut-5-d[equiv-lr])

lemma lem-cont-e[PLM]:
  [◇(∃ x . ⦇F,x^P⦈ & (◇(¬⦇F,x^P⦈))) ≡ ◇(∃ x . ((¬⦇F,x^P⦈) & ◇⦇F,x^P⦈))in v]
  proof –
    have [◇(∃ x . ⦇F,x^P⦈ & (◇(¬⦇F,x^P⦈))) in v]
          = [(∃ x . ◇(⦇F,x^P⦈ & ◇(¬⦇F,x^P⦈))) in v]
      using BF◇[deduction] CBF◇[deduction] by fast
    also have ... = [∃ x . (◇⦇F,x^P⦈ & ◇(¬⦇F,x^P⦈)) in v]
      apply (PLM-subst-method
             λ x . ◇(⦇F,x^P⦈ & ◇(¬⦇F,x^P⦈))
             λ x . ◇⦇F,x^P⦈ & ◇(¬⦇F,x^P⦈))
      using S5Basic-12 by auto
    also have ... = [∃ x . ◇(¬⦇F,x^P⦈) & ◇⦇F,x^P⦈ in v]
      apply (PLM-subst-method
             λ x . ◇⦇F,x^P⦈ & ◇(¬⦇F,x^P⦈)
             λ x . ◇(¬⦇F,x^P⦈) & ◇⦇F,x^P⦈)
      using oth-class-taut-3-b by auto
    also have ... = [∃ x . ◇((¬⦇F,x^P⦈) & ◇⦇F,x^P⦈) in v]
      apply (PLM-subst-method
             λ x . ◇(¬⦇F,x^P⦈) & ◇⦇F,x^P⦈
             λ x . ◇((¬⦇F,x^P⦈) & ◇⦇F,x^P⦈))
      using S5Basic-12[equiv-sym] by auto
    also have ... = [◇ (∃ x . ((¬⦇F,x^P⦈) & ◇⦇F,x^P⦈)) in v]
      using CBF◇[deduction] BF◇[deduction] by fast
```

finally show *?thesis* **using** $\equiv I$ *CP* **by** *blast*
qed

lemma *lem-cont-e-2*[*PLM*]:
$[\Diamond(\exists\ x\ .\ (|F,x^P|)\ \&\ \Diamond(\neg(|F,x^P|))) \equiv \Diamond(\exists\ x\ .\ (|F^-,x^P|)\ \&\ \Diamond(\neg(|F^-,x^P|)))\ \mathit{in}\ v]$
apply (*PLM-subst-method* $\lambda\ x\ .\ (|F,x^P|)\ \lambda\ x\ .\ \neg(|F^-,x^P|)$)
using *thm-relation-negation-2-1*[*equiv-sym*] **apply** *simp*
apply (*PLM-subst-method* $\lambda\ x\ .\ \neg(|F,x^P|)\ \lambda\ x\ .\ (|F^-,x^P|)$)
using *thm-relation-negation-1-1*[*equiv-sym*] **apply** *simp*
using *lem-cont-e* **by** *simp*

lemma *thm-cont-e-1*[*PLM*]:
$[\Diamond(\exists\ x\ .\ ((\neg(|E!,x^P|))\ \&\ (\Diamond(|E!,x^P|))))\ \mathit{in}\ v]$
using *lem-cont-e*[**where** *F*=*E*!, *equiv-lr*] *qml-4*[*axiom-instance*,*conj1*]
by *blast*

lemma *thm-cont-e-2*[*PLM*]:
[*Contingent* (*E*!) *in v*]
using *thm-cont-prop-2*[*equiv-rl*] &*I* *qml-4*[*axiom-instance*, *conj1*]
KBasic2-8[*deduction*, *OF sign-S5-thm-3*[*deduction*], *conj1*]
KBasic2-8[*deduction*, *OF sign-S5-thm-3*[*deduction*, *OF thm-cont-e-1*], *conj1*]
by *fast*

lemma *thm-cont-e-3*[*PLM*]:
[*Contingent* ($E!^-$) *in v*]
using *thm-cont-e-2 thm-cont-prop-3*[*equiv-lr*] **by** *blast*

lemma *thm-cont-e-4*[*PLM*]:
$[\exists\ (F{::}\Pi_1)\ G\ .\ (F \neq G\ \&\ \mathit{Contingent}\ F\ \&\ \mathit{Contingent}\ G)\ \mathit{in}\ v]$
apply (*rule-tac* α=*E*! **in** $\exists I$, *rule-tac* $\alpha=E!^-$ **in** $\exists I$)
using *thm-cont-e-2 thm-cont-e-3 thm-relation-negation-5-1* &*I* **by** *auto*

context
begin
qualified definition *L* **where** $L \equiv (\boldsymbol{\lambda}\ x\ .\ (|E!,\ x^P|) \rightarrow (|E!,\ x^P|))$

lemma *thm-noncont-e-e-1*[*PLM*]:
[*Necessary L in v*]
unfolding *Necessary-defs L-def* **apply** (*rule RN*, *rule* $\forall I$)
apply (*rule lambda-predicates-2-1*[*axiom-instance*, *equiv-rl*])
apply *show-proper*
using *if-p-then-p* **.**

lemma *thm-noncont-e-e-2*[*PLM*]:
[*Impossible* (L^-) *in v*]
unfolding *Impossible-defs L-def* **apply** (*rule RN*, *rule* $\forall I$)
apply (*rule thm-relation-negation-2-1*[*equiv-rl*])
apply (*rule lambda-predicates-2-1*[*axiom-instance*, *equiv-rl*])
apply *show-proper*
using *if-p-then-p* **.**

lemma *thm-noncont-e-e-3*[*PLM*]:
[*NonContingent* (*L*) *in v*]
unfolding *NonContingent-def* **using** *thm-noncont-e-e-1*
by (*rule* $\vee I(1)$)

lemma *thm-noncont-e-e-4*[*PLM*]:
[*NonContingent* (L^-) *in v*]

unfolding *NonContingent-def* **using** *thm-noncont-e-e-2*
by (*rule* $\vee I(2)$)

lemma *thm-noncont-e-e-5*[*PLM*]:
[$\exists$ (F::Π_1) G . $F \neq G$ & *NonContingent F* & *NonContingent G in v*]
apply (*rule-tac* α=L **in** $\exists I$, *rule-tac* α=L^- **in** $\exists I$)
using $\exists I$ *thm-relation-negation-5-1 thm-noncont-e-e-3*
thm-noncont-e-e-4 &I
by *simp*

lemma *four-distinct-1*[*PLM*]:
[*NonContingent* (F::Π_1) $\rightarrow \neg$($\exists$ G . (*Contingent G* & $G = F$)) *in v*]
proof (*rule CP*)
assume [*NonContingent F in v*]
hence [$\neg$(*Contingent F*) *in v*]
unfolding *NonContingent-def Contingent-def*
apply – **by** *PLM-solver*
moreover {
assume [$\exists$ G . *Contingent G* & $G = F$ *in v*]
then obtain P **where** [*Contingent P* & $P = F$ *in v*]
by (*rule* $\exists E$)
hence [*Contingent F in v*]
using &E *l-identity*[*axiom-instance*, *deduction*, *deduction*]
by *blast*
}
ultimately show [$\neg$($\exists G$. *Contingent G* & $G = F$) *in v*]
using *modus-tollens-1 CP* **by** *blast*
qed

lemma *four-distinct-2*[*PLM*]:
[*Contingent* (F::Π_1) $\rightarrow \neg$($\exists$ G . (*NonContingent G* & $G = F$)) *in v*]
proof (*rule CP*)
assume [*Contingent F in v*]
hence [$\neg$(*NonContingent F*) *in v*]
unfolding *NonContingent-def Contingent-def*
apply – **by** *PLM-solver*
moreover {
assume [$\exists$ G . *NonContingent G* & $G = F$ *in v*]
then obtain P **where** [*NonContingent P* & $P = F$ *in v*]
by (*rule* $\exists E$)
hence [*NonContingent F in v*]
using &E *l-identity*[*axiom-instance*, *deduction*, *deduction*]
by *blast*
}
ultimately show [$\neg$($\exists G$. *NonContingent G* & $G = F$) *in v*]
using *modus-tollens-1 CP* **by** *blast*
qed

lemma *four-distinct-3*[*PLM*]:
[$L \neq (L^-)$ & $L \neq E!$ & $L \neq (E!^-)$ & $(L^-) \neq E!$
& $(L^-) \neq (E!^-)$ & $E! \neq (E!^-)$ *in v*]
proof (*rule* &I)+
show [$L \neq (L^-)$ *in v*]
by (*rule thm-relation-negation-5-1*)
next
{
assume [$L = E!$ *in v*]

```
      hence [NonContingent L & L = E! in v]
        using thm-noncont-e-e-3 &I by auto
      hence [∃ G . NonContingent G & G = E! in v]
        using thm-noncont-e-e-3 &I ∃ I by fast
    }
    thus [L ≠ E! in v]
      using four-distinct-2[deduction, OF thm-cont-e-2]
            modus-tollens-1 CP
      by blast
  next
    {
      assume [L = (E!⁻) in v]
      hence [NonContingent L & L = (E!⁻) in v]
        using thm-noncont-e-e-3 &I by auto
      hence [∃ G . NonContingent G & G = (E!⁻) in v]
        using thm-noncont-e-e-3 &I ∃ I by fast
    }
    thus [L ≠ (E!⁻) in v]
      using four-distinct-2[deduction, OF thm-cont-e-3]
            modus-tollens-1 CP
      by blast
  next
    {
      assume [(L⁻) = E! in v]
      hence [NonContingent (L⁻) & (L⁻) = E! in v]
        using thm-noncont-e-e-4 &I by auto
      hence [∃ G . NonContingent G & G = E! in v]
        using thm-noncont-e-e-3 &I ∃ I by fast
    }
    thus [(L⁻) ≠ E! in v]
      using four-distinct-2[deduction, OF thm-cont-e-2]
            modus-tollens-1 CP
      by blast
  next
    {
      assume [(L⁻) = (E!⁻) in v]
      hence [NonContingent (L⁻) & (L⁻) = (E!⁻) in v]
        using thm-noncont-e-e-4 &I by auto
      hence [∃ G . NonContingent G & G = (E!⁻) in v]
        using thm-noncont-e-e-3 &I ∃ I by fast
    }
    thus [(L⁻) ≠ (E!⁻) in v]
      using four-distinct-2[deduction, OF thm-cont-e-3]
            modus-tollens-1 CP
      by blast
  next
    show [E! ≠ (E!⁻) in v]
      by (rule thm-relation-negation-5-1)
  qed
end

lemma thm-cont-propos-1[PLM]:
  [NonContingent (p::o) ≡ NonContingent (p⁻) in v]
  proof (rule ≡I; rule CP)
    assume [NonContingent p in v]
    hence [□p ∨ □¬p in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs .
    hence [□(¬(p⁻)) ∨ □(¬p) in v]
```

```
      apply −
      apply (PLM-subst-method p ¬(p^-))
      using thm-relation-negation-4[equiv-sym] by auto
    hence [□(¬(p^-)) ∨ □(p^-) in v]
      apply −
      apply (PLM-subst-goal-method λφ . □(¬(p^-)) ∨ □(φ) ¬p)
      using thm-relation-negation-3[equiv-sym] by auto
    hence [□(p^-) ∨ □(¬(p^-)) in v]
      by (rule oth-class-taut-3-e[equiv-lr])
    thus [NonContingent (p^-) in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs .
  next
    assume [NonContingent (p^-) in v]
    hence [□(¬(p^-)) ∨ □(p^-) in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs
      by (rule oth-class-taut-3-e[equiv-lr])
    hence [□(p) ∨ □(p^-) in v]
      apply −
      apply (PLM-subst-goal-method λφ . □φ ∨ □(p^-) ¬(p^-))
      using thm-relation-negation-4 by auto
    hence [□(p) ∨ □(¬p) in v]
      apply −
      apply (PLM-subst-method p^- ¬p)
      using thm-relation-negation-3 by auto
    thus [NonContingent p in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs .
  qed

lemma thm-cont-propos-2[PLM]:
  [Contingent p ≡ ◇p & ◇(¬p) in v]
  proof (rule ≡I; rule CP)
    assume [Contingent p in v]
    hence [¬(□p ∨ □(¬p)) in v]
      unfolding Contingent-def Necessary-defs Impossible-defs .
    hence [(¬□p) & (¬□(¬p)) in v]
      by (rule oth-class-taut-6-d[equiv-lr])
    hence [(◇¬(¬p)) & (◇¬p) in v]
      using KBasic2-2[equiv-lr] &I &E by meson
    thus [(◇p) & (◇(¬p)) in v]
      apply − apply PLM-solver
      apply (PLM-subst-method ¬¬p p)
      using oth-class-taut-4-b[equiv-sym] by auto
  next
    assume [(◇p) & (◇¬(p)) in v]
    hence [(◇¬(¬p)) & (◇¬(p)) in v]
      apply − apply PLM-solver
      apply (PLM-subst-method p ¬¬p)
      using oth-class-taut-4-b by auto
    hence [(¬□p) & (¬□(¬p)) in v]
      using KBasic2-2[equiv-rl] &I &E by meson
    hence [¬(□(p) ∨ □(¬p)) in v]
      by (rule oth-class-taut-6-d[equiv-rl])
    thus [Contingent p in v]
      unfolding Contingent-def Necessary-defs Impossible-defs .
  qed

lemma thm-cont-propos-3[PLM]:
  [Contingent (p::o) ≡ Contingent (p^-) in v]
```

```
  using thm-cont-propos-1
  unfolding NonContingent-def Contingent-def
  by (rule oth-class-taut-5-d[equiv-lr])

context
begin
  private definition p₀ where
    p₀ ≡ ∀ x. ⦇E!,x^P⦈ → ⦇E!,x^P⦈

  lemma thm-noncont-propos-1[PLM]:
    [Necessary p₀ in v]
    unfolding Necessary-defs p₀-def
    apply (rule RN, rule ∀ I)
    using if-p-then-p .

  lemma thm-noncont-propos-2[PLM]:
    [Impossible (p₀⁻) in v]
    unfolding Impossible-defs
    apply (PLM-subst-method ¬p₀ p₀⁻)
     using thm-relation-negation-3[equiv-sym] apply simp
    apply (PLM-subst-method p₀ ¬¬p₀)
     using oth-class-taut-4-b apply simp
    using thm-noncont-propos-1 unfolding Necessary-defs
    by simp

  lemma thm-noncont-propos-3[PLM]:
    [NonContingent (p₀) in v]
    unfolding NonContingent-def using thm-noncont-propos-1
    by (rule ∨I(1))

  lemma thm-noncont-propos-4[PLM]:
    [NonContingent (p₀⁻) in v]
    unfolding NonContingent-def using thm-noncont-propos-2
    by (rule ∨I(2))

  lemma thm-noncont-propos-5[PLM]:
    [∃ (p::o) q . p ≠ q & NonContingent p & NonContingent q in v]
    apply (rule-tac α=p₀ in ∃ I, rule-tac α=p₀⁻ in ∃ I)
    using ∃ I thm-relation-negation-6 thm-noncont-propos-3
          thm-noncont-propos-4 &I by simp

  private definition q₀ where
    q₀ ≡ ∃ x . ⦇E!,x^P⦈ & ◊(¬⦇E!,x^P⦈)

  lemma basic-prop-1[PLM]:
    [∃ p . ◊p & ◊(¬p) in v]
    apply (rule-tac α=q₀ in ∃ I) unfolding q₀-def
    using qml-4[axiom-instance] by simp

  lemma basic-prop-2[PLM]:
    [Contingent q₀ in v]
    unfolding Contingent-def Necessary-defs Impossible-defs
    apply (rule oth-class-taut-6-d[equiv-rl])
    apply (PLM-subst-goal-method λ φ . (¬□(φ)) & ¬□¬q₀ ¬¬q₀)
     using oth-class-taut-4-b[equiv-sym] apply simp
    using qml-4[axiom-instance,conj-sym]
    unfolding q₀-def diamond-def by simp
```

lemma *basic-prop-3*[*PLM*]:
  [*Contingent* (q_0^-) *in* v]
  apply (*rule thm-cont-propos-3*[*equiv-lr*])
  using *basic-prop-2* **.**

lemma *basic-prop-4*[*PLM*]:
  [$\exists$ (p::o) q . $p \neq q$ **&** *Contingent* p **&** *Contingent* q *in* v]
  apply (*rule-tac* α=q_0 **in** $\exists I$, *rule-tac* α=q_0^- **in** $\exists I$)
  using *thm-relation-negation-6 basic-prop-2 basic-prop-3* **&***I* **by** *simp*

lemma *four-distinct-props-1*[*PLM*]:
  [*NonContingent* (p::Π_0) $\rightarrow$ ($\neg$($\exists$ q . *Contingent* q **&** $q = p$)) *in* v]
  proof (*rule CP*)
    assume [*NonContingent* p *in* v]
    hence [$\neg$(*Contingent* p) *in* v]
      unfolding *NonContingent-def Contingent-def*
      apply $-$ **by** *PLM-solver*
    moreover {
       assume [$\exists$ q . *Contingent* q **&** $q = p$ *in* v]
       then obtain r **where** [*Contingent* r **&** $r = p$ *in* v]
        by (*rule* $\exists E$)
       hence [*Contingent* p *in* v]
         using **&***E l-identity*[*axiom-instance*, *deduction*, *deduction*]
         by *blast*
    }
    ultimately show [$\neg$($\exists q$. *Contingent* q **&** $q = p$) *in* v]
      using *modus-tollens-1 CP* **by** *blast*
  qed

lemma *four-distinct-props-2*[*PLM*]:
  [*Contingent* (p::o) $\rightarrow$ $\neg$($\exists$ q . (*NonContingent* q **&** $q = p$)) *in* v]
  proof (*rule CP*)
    assume [*Contingent* p *in* v]
    hence [$\neg$(*NonContingent* p) *in* v]
      unfolding *NonContingent-def Contingent-def*
      apply $-$ **by** *PLM-solver*
    moreover {
       assume [$\exists$ q . *NonContingent* q **&** $q = p$ *in* v]
       then obtain r **where** [*NonContingent* r **&** $r = p$ *in* v]
        by (*rule* $\exists E$)
       hence [*NonContingent* p *in* v]
         using **&***E l-identity*[*axiom-instance*, *deduction*, *deduction*]
         by *blast*
    }
    ultimately show [$\neg$($\exists q$. *NonContingent* q **&** $q = p$) *in* v]
      using *modus-tollens-1 CP* **by** *blast*
  qed

lemma *four-distinct-props-4*[*PLM*]:
  [$p_0 \neq (p_0^-)$ **&** $p_0 \neq q_0$ **&** $p_0 \neq (q_0^-)$ **&** $(p_0^-) \neq q_0$
   & $(p_0^-) \neq (q_0^-)$ **&** $q_0 \neq (q_0^-)$ *in* v]
  proof (*rule* **&***I*)+
    show [$p_0 \neq (p_0^-)$ *in* v]
      by (*rule thm-relation-negation-6*)
    next
      {
        assume [$p_0 = q_0$ *in* v]
        hence [$\exists$ q . *NonContingent* q **&** $q = q_0$ *in* v]

```
        using &I thm-noncont-propos-3 ∃ I[where α=p₀]
        by simp
      }
      thus [p₀ ≠ q₀ in v]
        using four-distinct-props-2[deduction, OF basic-prop-2]
              modus-tollens-1 CP
        by blast
    next
      {
        assume [p₀ = (q₀⁻) in v]
        hence [∃ q . NonContingent q & q = (q₀⁻) in v]
          using thm-noncont-propos-3 &I ∃ I[where α=p₀] by simp
      }
      thus [p₀ ≠ (q₀⁻) in v]
        using four-distinct-props-2[deduction, OF basic-prop-3]
              modus-tollens-1 CP
      by blast
    next
      {
        assume [(p₀⁻) = q₀ in v]
        hence [∃ q . NonContingent q & q = q₀ in v]
          using thm-noncont-propos-4 &I ∃ I[where α=p₀⁻] by auto
      }
      thus [(p₀⁻) ≠ q₀ in v]
        using four-distinct-props-2[deduction, OF basic-prop-2]
              modus-tollens-1 CP
        by blast
    next
      {
        assume [(p₀⁻) = (q₀⁻) in v]
        hence [∃ q . NonContingent q & q = (q₀⁻) in v]
          using thm-noncont-propos-4 &I ∃ I[where α=p₀⁻] by auto
      }
      thus [(p₀⁻) ≠ (q₀⁻) in v]
        using four-distinct-props-2[deduction, OF basic-prop-3]
              modus-tollens-1 CP
        by blast
    next
      show [q₀ ≠ (q₀⁻) in v]
        by (rule thm-relation-negation-6)
    qed

lemma cont-true-cont-1[PLM]:
  [ContingentlyTrue p → Contingent p in v]
  apply (rule CP, rule thm-cont-propos-2[equiv-rl])
  unfolding ContingentlyTrue-def
  apply (rule &I, drule &E(1))
   using T◊[deduction] apply simp
  by (rule &E(2))

lemma cont-true-cont-2[PLM]:
  [ContingentlyFalse p → Contingent p in v]
  apply (rule CP, rule thm-cont-propos-2[equiv-rl])
  unfolding ContingentlyFalse-def
  apply (rule &I, drule &E(2))
   apply simp
  apply (drule &E(1))
  using T◊[deduction] by simp
```

```
lemma cont-true-cont-3[PLM]:
  [ContingentlyTrue p ≡ ContingentlyFalse (p⁻) in v]
  unfolding ContingentlyTrue-def ContingentlyFalse-def
  apply (PLM-subst-method ¬p p⁻)
   using thm-relation-negation-3[equiv-sym] apply simp
  apply (PLM-subst-method p ¬¬p)
  by PLM-solver+

lemma cont-true-cont-4[PLM]:
  [ContingentlyFalse p ≡ ContingentlyTrue (p⁻) in v]
  unfolding ContingentlyTrue-def ContingentlyFalse-def
  apply (PLM-subst-method ¬p p⁻)
   using thm-relation-negation-3[equiv-sym] apply simp
  apply (PLM-subst-method p ¬¬p)
  by PLM-solver+

lemma cont-tf-thm-1[PLM]:
  [ContingentlyTrue q0 ∨ ContingentlyFalse q0 in v]
  proof −
    have [q0 ∨ ¬q0 in v]
      by PLM-solver
    moreover {
      assume [q0 in v]
      hence [q0 & ◇¬q0 in v]
        unfolding q0-def
        using qml-4[axiom-instance,conj2] &I
        by auto
    }
    moreover {
      assume [¬q0 in v]
      hence [(¬q0) & ◇q0 in v]
        unfolding q0-def
        using qml-4[axiom-instance,conj1] &I
        by auto
    }
    ultimately show ?thesis
      unfolding ContingentlyTrue-def ContingentlyFalse-def
      using ∨E(4) CP by auto
  qed

lemma cont-tf-thm-2[PLM]:
  [ContingentlyFalse q0 ∨ ContingentlyFalse (q0⁻) in v]
  using cont-tf-thm-1 cont-true-cont-3[where p=q0]
        cont-true-cont-4[where p=q0]
  apply − by PLM-solver

lemma cont-tf-thm-3[PLM]:
  [∃ p . ContingentlyTrue p in v]
  proof (rule ∨E(1); (rule CP)?)
    show [ContingentlyTrue q0 ∨ ContingentlyFalse q0 in v]
      using cont-tf-thm-1 .
  next
    assume [ContingentlyTrue q0 in v]
    thus ?thesis
      using ∃I by metis
  next
    assume [ContingentlyFalse q0 in v]
```

```
    hence [ContingentlyTrue (q0^-) in v]
      using cont-true-cont-4[equiv-lr] by simp
    thus ?thesis
      using ∃ I by metis
  qed

lemma cont-tf-thm-4[PLM]:
  [∃ p . ContingentlyFalse p in v]
  proof (rule ∨E(1); (rule CP)?)
    show [ContingentlyTrue q0 ∨ ContingentlyFalse q0 in v]
      using cont-tf-thm-1 .
  next
    assume [ContingentlyTrue q0 in v]
    hence [ContingentlyFalse (q0^-) in v]
      using cont-true-cont-3[equiv-lr] by simp
    thus ?thesis
      using ∃ I by metis
  next
    assume [ContingentlyFalse q0 in v]
    thus ?thesis
      using ∃ I by metis
  qed

lemma cont-tf-thm-5[PLM]:
  [ContingentlyTrue p & Necessary q → p ≠ q in v]
  proof (rule CP)
    assume [ContingentlyTrue p & Necessary q in v]
    hence 1: [◊(¬p) & □ q in v]
      unfolding ContingentlyTrue-def Necessary-defs
      using &E &I by blast
    hence [¬□p in v]
      apply − apply (drule &E(1))
      unfolding diamond-def
      apply (PLM-subst-method ¬¬p p)
      using oth-class-taut-4-b[equiv-sym] by auto
    moreover {
      assume [p = q in v]
      hence [□p in v]
        using l-identity[where α=q and β=p and φ=λ x . □ x,
                         axiom-instance, deduction, deduction]
              1[conj2] id-eq-prop-prop-8-b[deduction]
        by blast
    }
    ultimately show [p ≠ q in v]
      using modus-tollens-1 CP by blast
  qed

lemma cont-tf-thm-6[PLM]:
  [(ContingentlyFalse p & Impossible q) → p ≠ q in v]
  proof (rule CP)
    assume [ContingentlyFalse p & Impossible q in v]
    hence 1: [◊p & □(¬q) in v]
      unfolding ContingentlyFalse-def Impossible-defs
      using &E &I by blast
    hence [¬◊q in v]
      unfolding diamond-def apply − by PLM-solver
    moreover {
      assume [p = q in v]
```

```
      hence [◇q in v]
        using l-identity[axiom-instance, deduction, deduction] 1[conj1]
              id-eq-prop-prop-8-b[deduction]
        by blast
    }
    ultimately show [p ≠ q in v]
      using modus-tollens-1 CP by blast
  qed
end

lemma oa-contingent-1[PLM]:
  [O! ≠ A! in v]
  proof −
    {
      assume [O! = A! in v]
      hence [(λx. ◇⦇E!,x^P⦈) = (λx. ¬◇⦇E!,x^P⦈) in v]
        unfolding Ordinary-def Abstract-def .
      moreover have [⦇(λx. ◇⦇E!,x^P⦈), x^P⦈ ≡ ◇⦇E!,x^P⦈ in v]
        apply (rule beta-C-meta-1)
        by show-proper
      ultimately have [⦇(λx. ¬◇⦇E!,x^P⦈), x^P⦈ ≡ ◇⦇E!,x^P⦈ in v]
        using l-identity[axiom-instance, deduction, deduction] by fast
      moreover have [⦇(λx. ¬◇⦇E!,x^P⦈), x^P⦈ ≡ ¬◇⦇E!,x^P⦈ in v]
        apply (rule beta-C-meta-1)
        by show-proper
      ultimately have [◇⦇E!,x^P⦈ ≡ ¬◇⦇E!,x^P⦈ in v]
        apply − by PLM-solver
    }
    thus ?thesis
      using oth-class-taut-1-b modus-tollens-1 CP
      by blast
  qed

lemma oa-contingent-2[PLM]:
  [⦇O!,x^P⦈ ≡ ¬⦇A!,x^P⦈ in v]
  proof −
      have [⦇(λx. ¬◇⦇E!,x^P⦈), x^P⦈ ≡ ¬◇⦇E!,x^P⦈ in v]
        apply (rule beta-C-meta-1)
        by show-proper
      hence [(¬⦇(λx. ¬◇⦇E!,x^P⦈), x^P⦈) ≡ ◇⦇E!,x^P⦈ in v]
        using oth-class-taut-5-d[equiv-lr] oth-class-taut-4-b[equiv-sym]
              ≡E(5) by blast
      moreover have [⦇(λx. ◇⦇E!,x^P⦈), x^P⦈ ≡ ◇⦇E!,x^P⦈ in v]
        apply (rule beta-C-meta-1)
        by show-proper
      ultimately show ?thesis
        unfolding Ordinary-def Abstract-def
        apply − by PLM-solver
  qed

lemma oa-contingent-3[PLM]:
  [⦇A!,x^P⦈ ≡ ¬⦇O!,x^P⦈ in v]
  using oa-contingent-2
  apply − by PLM-solver

lemma oa-contingent-4[PLM]:
  [Contingent O! in v]
  apply (rule thm-cont-prop-2[equiv-rl], rule &I)
```

```
  subgoal
    unfolding Ordinary-def
    apply (PLM-subst-method λ x . ◇⦇E!,x^P⦈ λ x . ⦇λx. ◇⦇E!,x^P⦈,x^P⦈)
     apply (safe intro!: beta-C-meta-1[equiv-sym])
      apply show-proper
    using BF◇[deduction, OF thm-cont-prop-2[equiv-lr, OF thm-cont-e-2, conj1]]
    by (rule T◇[deduction])
  subgoal
    apply (PLM-subst-method λ x . ⦇A!,x^P⦈ λ x . ¬⦇O!,x^P⦈)
     using oa-contingent-3 apply simp
    using cqt-further-5[deduction,conj1, OF A-objects[axiom-instance]]
    by (rule T◇[deduction])
  done

lemma oa-contingent-5[PLM]:
  [Contingent A! in v]
  apply (rule thm-cont-prop-2[equiv-rl], rule &I)
  subgoal
    using cqt-further-5[deduction,conj1, OF A-objects[axiom-instance]]
    by (rule T◇[deduction])
  subgoal
    unfolding Abstract-def
    apply (PLM-subst-method λ x . ¬◇⦇E!,x^P⦈ λ x . ⦇λx. ¬◇⦇E!,x^P⦈,x^P⦈)
     apply (safe intro!: beta-C-meta-1[equiv-sym])
      apply show-proper
    apply (PLM-subst-method λ x . ◇⦇E!,x^P⦈ λ x . ¬¬◇⦇E!,x^P⦈)
     using oth-class-taut-4-b apply simp
    using BF◇[deduction, OF thm-cont-prop-2[equiv-lr, OF thm-cont-e-2, conj1]]
    by (rule T◇[deduction])
  done

lemma oa-contingent-6[PLM]:
  [(O!⁻) ≠ (A!⁻) in v]
  proof -
   {
    assume [(O!⁻) = (A!⁻) in v]
    hence [(λx. ¬⦇O!,x^P⦈) = (λx. ¬⦇A!,x^P⦈) in v]
      unfolding propnot-defs .
    moreover have [⦇(λx. ¬⦇O!,x^P⦈), x^P⦈ ≡ ¬⦇O!,x^P⦈ in v]
      apply (rule beta-C-meta-1)
      by show-proper
    ultimately have [⦇λx. ¬⦇A!,x^P⦈,x^P⦈ ≡  ¬⦇O!,x^P⦈ in v]
      using l-identity[axiom-instance, deduction, deduction]
      by fast
    hence [(¬⦇A!,x^P⦈) ≡ ¬⦇O!,x^P⦈ in v]
      apply -
      apply (PLM-subst-method ⦇λx. ¬⦇A!,x^P⦈,x^P⦈ (¬⦇A!,x^P⦈))
       apply (safe intro!: beta-C-meta-1)
      by show-proper
    hence [⦇O!,x^P⦈ ≡ ¬⦇O!,x^P⦈ in v]
      using oa-contingent-2 apply - by PLM-solver
   }
   thus ?thesis
    using oth-class-taut-1-b modus-tollens-1 CP
    by blast
  qed

lemma oa-contingent-7[PLM]:
```

$[(\!|O!^-,x^P|\!) \equiv \neg(\!|A!^-,x^P|\!)\ in\ v]$
proof −
have $[(\neg(\!|\boldsymbol{\lambda}x.\ \neg(\!|A!,x^P|\!),x^P|\!)) \equiv (\!|A!,x^P|\!)\ in\ v]$
apply ($PLM\text{-}subst\text{-}method$ $(\neg(\!|A!,x^P|\!))$ $(\!|\boldsymbol{\lambda}x.\ \neg(\!|A!,x^P|\!),x^P|\!)$)
apply (*safe intro*!: *beta-C-meta-1*[*equiv-sym*])
apply *show-proper*
using *oth-class-taut-4-b*[*equiv-sym*] **by** *auto*
moreover have $[(\!|\boldsymbol{\lambda}x.\ \neg(\!|O!,x^P|\!),x^P|\!) \equiv \neg(\!|O!,x^P|\!)\ in\ v]$
apply (*rule beta-C-meta-1*)
by *show-proper*
ultimately show *?thesis*
unfolding *propnot-defs*
using *oa-contingent-3*
apply − **by** *PLM-solver*
qed

lemma *oa-contingent-8*[*PLM*]:
$[Contingent\ (O!^-)\ in\ v]$
using *oa-contingent-4 thm-cont-prop-3*[*equiv-lr*] **by** *auto*

lemma *oa-contingent-9*[*PLM*]:
$[Contingent\ (A!^-)\ in\ v]$
using *oa-contingent-5 thm-cont-prop-3*[*equiv-lr*] **by** *auto*

lemma *oa-facts-1*[*PLM*]:
$[(\!|O!,x^P|\!) \rightarrow \Box(\!|O!,x^P|\!)\ in\ v]$
proof (*rule CP*)
assume $[(\!|O!,x^P|\!)\ in\ v]$
hence $[\Diamond(\!|E!,x^P|\!)\ in\ v]$
unfolding *Ordinary-def* **apply** −
apply (*rule beta-C-meta-1*[*equiv-lr*])
by *show-proper*
hence $[\Box\Diamond(\!|E!,x^P|\!)\ in\ v]$
using *qml-3*[*axiom-instance*, *deduction*] **by** *auto*
thus $[\Box(\!|O!,x^P|\!)\ in\ v]$
unfolding *Ordinary-def*
apply −
apply ($PLM\text{-}subst\text{-}method$ $\Diamond(\!|E!,x^P|\!)$ $(\!|\boldsymbol{\lambda}x.\ \Diamond(\!|E!,x^P|\!),x^P|\!)$)
apply (*safe intro*!: *beta-C-meta-1*[*equiv-sym*])
by *show-proper*
qed

lemma *oa-facts-2*[*PLM*]:
$[(\!|A!,x^P|\!) \rightarrow \Box(\!|A!,x^P|\!)\ in\ v]$
proof (*rule CP*)
assume $[(\!|A!,x^P|\!)\ in\ v]$
hence $[\neg\Diamond(\!|E!,x^P|\!)\ in\ v]$
unfolding *Abstract-def* **apply** −
apply (*rule beta-C-meta-1*[*equiv-lr*])
by *show-proper*
hence $[\Box\Box\neg(\!|E!,x^P|\!)\ in\ v]$
using *KBasic2-4*[*equiv-rl*] *4*$\Box$[*deduction*] **by** *auto*
hence $[\Box\neg\Diamond(\!|E!,x^P|\!)\ in\ v]$
apply −
apply ($PLM\text{-}subst\text{-}method$ $\Box\neg(\!|E!,x^P|\!)$ $\neg\Diamond(\!|E!,x^P|\!)$)
using *KBasic2-4* **by** *auto*
thus $[\Box(\!|A!,x^P|\!)\ in\ v]$
unfolding *Abstract-def*

apply –
apply (*PLM-subst-method* $\neg\Diamond(|E!,x^P|)$ $(|\boldsymbol{\lambda}x.\ \neg\Diamond(|E!,x^P|),x^P|)$)
apply (*safe intro*!: *beta-C-meta-1*[*equiv-sym*])
by *show-proper*
qed

lemma *oa-facts-3*[*PLM*]:
$[\Diamond(|O!,x^P|) \rightarrow (|O!,x^P|)$ *in v*$]$
using *oa-facts-1* **by** (*rule derived-S5-rules-2-b*)

lemma *oa-facts-4*[*PLM*]:
$[\Diamond(|A!,x^P|) \rightarrow (|A!,x^P|)$ *in v*$]$
using *oa-facts-2* **by** (*rule derived-S5-rules-2-b*)

lemma *oa-facts-5*[*PLM*]:
$[\Diamond(|O!,x^P|) \equiv \Box(|O!,x^P|)$ *in v*$]$
using *oa-facts-1*[*deduction*, *OF oa-facts-3*[*deduction*]]
T$\Diamond$[*deduction*, *OF qml-2*[*axiom-instance*, *deduction*]]
$\equiv$*I CP* **by** *blast*

lemma *oa-facts-6*[*PLM*]:
$[\Diamond(|A!,x^P|) \equiv \Box(|A!,x^P|)$ *in v*$]$
using *oa-facts-2*[*deduction*, *OF oa-facts-4*[*deduction*]]
T$\Diamond$[*deduction*, *OF qml-2*[*axiom-instance*, *deduction*]]
$\equiv$*I CP* **by** *blast*

lemma *oa-facts-7*[*PLM*]:
$[(|O!,x^P|) \equiv \boldsymbol{\mathcal{A}}(|O!,x^P|)$ *in v*$]$
apply (*rule* $\equiv$*I*; *rule CP*)
apply (*rule nec-imp-act*[*deduction*, *OF oa-facts-1*[*deduction*]]; *assumption*)
proof –
assume $[\boldsymbol{\mathcal{A}}(|O!,x^P|)$ *in v*$]$
hence $[\boldsymbol{\mathcal{A}}(\Diamond(|E!,x^P|))$ *in v*$]$
unfolding *Ordinary-def* **apply** –
apply (*PLM-subst-method* $(|\boldsymbol{\lambda}x.\ \Diamond(|E!,x^P|),x^P|)$ $\Diamond(|E!,x^P|)$)
apply (*safe intro*!: *beta-C-meta-1*)
by *show-proper*
hence $[\Diamond(|E!,x^P|)$ *in v*$]$
using *Act-Basic-6*[*equiv-rl*] **by** *auto*
thus $[(|O!,x^P|)$ *in v*$]$
unfolding *Ordinary-def* **apply** –
apply (*PLM-subst-method* $\Diamond(|E!,x^P|)$ $(|\boldsymbol{\lambda}x.\ \Diamond(|E!,x^P|),x^P|)$)
apply (*safe intro*!: *beta-C-meta-1*[*equiv-sym*])
by *show-proper*
qed

lemma *oa-facts-8*[*PLM*]:
$[(|A!,x^P|) \equiv \boldsymbol{\mathcal{A}}(|A!,x^P|)$ *in v*$]$
apply (*rule* $\equiv$*I*; *rule CP*)
apply (*rule nec-imp-act*[*deduction*, *OF oa-facts-2*[*deduction*]]; *assumption*)
proof –
assume $[\boldsymbol{\mathcal{A}}(|A!,x^P|)$ *in v*$]$
hence $[\boldsymbol{\mathcal{A}}(\neg\Diamond(|E!,x^P|))$ *in v*$]$
unfolding *Abstract-def* **apply** –
apply (*PLM-subst-method* $(|\boldsymbol{\lambda}x.\ \neg\Diamond(|E!,x^P|),x^P|)$ $\neg\Diamond(|E!,x^P|)$)
apply (*safe intro*!: *beta-C-meta-1*)
by *show-proper*
hence $[\boldsymbol{\mathcal{A}}(\Box\neg(|E!,x^P|))$ *in v*$]$

```
      apply −
      apply (PLM-subst-method (¬◇⦇E!,x^P⦈) (□¬⦇E!,x^P⦈))
      using KBasic2-4[equiv-sym] by auto
    hence [¬◇⦇E!,x^P⦈ in v]
      using qml-act-2[axiom-instance, equiv-rl] KBasic2-4[equiv-lr] by auto
    thus [⦇A!,x^P⦈ in v]
      unfolding Abstract-def apply −
      apply (PLM-subst-method ¬◇⦇E!,x^P⦈ ⦇λx. ¬◇⦇E!,x^P⦈,x^P⦈)
      apply (safe intro!: beta-C-meta-1[equiv-sym])
      by show-proper
  qed

lemma cont-nec-fact1-1[PLM]:
  [WeaklyContingent F ≡ WeaklyContingent (F⁻) in v]
  proof (rule ≡I; rule CP)
    assume [WeaklyContingent F in v]
    hence wc-def: [Contingent F & (∀ x . (◇⦇F,x^P⦈ → □⦇F,x^P⦈)) in v]
      unfolding WeaklyContingent-def .
    have [Contingent (F⁻) in v]
      using wc-def[conj1] by (rule thm-cont-prop-3[equiv-lr])
    moreover {
      {
        fix x
        assume [◇⦇F⁻,x^P⦈ in v]
        hence [¬□⦇F,x^P⦈ in v]
          unfolding diamond-def apply −
          apply (PLM-subst-method ¬⦇F⁻,x^P⦈ ⦇F,x^P⦈)
           using thm-relation-negation-2-1 by auto
        moreover {
          assume [¬□⦇F⁻,x^P⦈ in v]
          hence [¬□⦇λx. ¬⦇F,x^P⦈,x^P⦈ in v]
            unfolding propnot-defs .
          hence [◇⦇F,x^P⦈ in v]
            unfolding diamond-def
            apply − apply (PLM-subst-method ⦇λx. ¬⦇F,x^P⦈,x^P⦈ ¬⦇F,x^P⦈)
            apply (safe intro!: beta-C-meta-1)
            by show-proper
          hence [□⦇F,x^P⦈ in v]
            using wc-def[conj2] cqt-1[axiom-instance, deduction]
                  modus-ponens by fast
        }
        ultimately have [□⦇F⁻, x^P⦈ in v]
          using ¬¬E modus-tollens-1 CP by blast
      }
      hence [∀ x . ◇⦇F⁻,x^P⦈ → □⦇F⁻, x^P⦈ in v]
        using ∀ I CP by fast
    }
    ultimately show [WeaklyContingent (F⁻) in v]
      unfolding WeaklyContingent-def by (rule &I)
  next
    assume [WeaklyContingent (F⁻) in v]
    hence wc-def: [Contingent (F⁻) & (∀ x . (◇⦇F⁻,x^P⦈ → □⦇F⁻,x^P⦈)) in v]
      unfolding WeaklyContingent-def .
    have [Contingent F in v]
      using wc-def[conj1] by (rule thm-cont-prop-3[equiv-rl])
    moreover {
      {
        fix x
```

```
      assume [◇⦇F,x^P⦈ in v]
      hence [¬□⦇F⁻,x^P⦈ in v]
        unfolding diamond-def apply –
        apply (PLM-subst-method ¬⦇F,x^P⦈ ⦇F⁻,x^P⦈)
        using thm-relation-negation-1-1[equiv-sym] by auto
      moreover {
        assume [¬□⦇F,x^P⦈ in v]
        hence [◇⦇F⁻,x^P⦈ in v]
          unfolding diamond-def
          apply – apply (PLM-subst-method ⦇F,x^P⦈ ¬⦇F⁻,x^P⦈)
          using thm-relation-negation-2-1[equiv-sym] by auto
        hence [□⦇F⁻,x^P⦈ in v]
          using wc-def[conj2] cqt-1[axiom-instance, deduction]
                modus-ponens by fast
      }
      ultimately have [□⦇F, x^P⦈ in v]
        using ¬¬E modus-tollens-1 CP by blast
    }
    hence [∀ x . ◇⦇F,x^P⦈ → □⦇F, x^P⦈ in v]
      using ∀I CP by fast
  }
  ultimately show [WeaklyContingent (F) in v]
    unfolding WeaklyContingent-def by (rule &I)
qed

lemma cont-nec-fact1-2[PLM]:
  [(WeaklyContingent F & ¬(WeaklyContingent G)) → (F ≠ G) in v]
  using l-identity[axiom-instance,deduction,deduction] &E &I
        modus-tollens-1 CP by metis

lemma cont-nec-fact2-1[PLM]:
  [WeaklyContingent (O!) in v]
  unfolding WeaklyContingent-def
  apply (rule &I)
   using oa-contingent-4 apply simp
  using oa-facts-5 unfolding equiv-def
  using &E(1) ∀I by fast

lemma cont-nec-fact2-2[PLM]:
  [WeaklyContingent (A!) in v]
  unfolding WeaklyContingent-def
  apply (rule &I)
   using oa-contingent-5 apply simp
  using oa-facts-6 unfolding equiv-def
  using &E(1) ∀I by fast

lemma cont-nec-fact2-3[PLM]:
  [¬(WeaklyContingent (E!)) in v]
  proof (rule modus-tollens-1, rule CP)
    assume [WeaklyContingent E! in v]
    thus [∀ x . ◇⦇E!,x^P⦈ → □⦇E!,x^P⦈ in v]
    unfolding WeaklyContingent-def using &E(2) by fast
  next
    {
      assume 1: [∀ x . ◇⦇E!,x^P⦈ → □⦇E!,x^P⦈ in v]
      have [∃ x . ◇(⦇E!,x^P⦈ & ◇(¬⦇E!,x^P⦈)) in v]
        using qml-4[axiom-instance,conj1, THEN BFs-3[deduction]] .
      then obtain x where [◇(⦇E!,x^P⦈ & ◇(¬⦇E!,x^P⦈)) in v]
```

```
      by (rule ∃E)
    hence [◇⦇E!,x^P⦈ & ◇(¬⦇E!,x^P⦈) in v]
      using KBasic2-8[deduction] S5Basic-8[deduction]
            &I &E by blast
    hence [□⦇E!,x^P⦈ & (¬□⦇E!,x^P⦈) in v]
      using 1[THEN ∀E, deduction] &E &I
            KBasic2-2[equiv-rl] by blast
    hence [¬(∀ x . ◇⦇E!,x^P⦈ → □⦇E!,x^P⦈) in v]
      using oth-class-taut-1-a modus-tollens-1 CP by blast
  }
  thus [¬(∀ x . ◇⦇E!,x^P⦈ → □⦇E!,x^P⦈) in v]
    using reductio-aa-2 if-p-then-p CP by meson
qed

lemma cont-nec-fact2-4[PLM]:
  [¬(WeaklyContingent (PLM.L)) in v]
  proof –
    {
      assume [WeaklyContingent PLM.L in v]
      hence [Contingent PLM.L in v]
        unfolding WeaklyContingent-def using &E(1) by blast
    }
    thus ?thesis
      using thm-noncont-e-e-3
      unfolding Contingent-def NonContingent-def
      using modus-tollens-2 CP by blast
  qed

lemma cont-nec-fact2-5[PLM]:
  [O! ≠ E! & O! ≠ (E!⁻) & O! ≠ PLM.L & O! ≠ (PLM.L⁻) in v]
  proof ((rule &I)+)
    show [O! ≠ E! in v]
      using cont-nec-fact2-1 cont-nec-fact2-3
            cont-nec-fact1-2[deduction] &I by simp
  next
    have [¬(WeaklyContingent (E!⁻)) in v]
      using cont-nec-fact1-1[THEN oth-class-taut-5-d[equiv-lr], equiv-lr]
            cont-nec-fact2-3 by auto
    thus [O! ≠ (E!⁻) in v]
      using cont-nec-fact2-1 cont-nec-fact1-2[deduction] &I by simp
  next
    show [O! ≠ PLM.L in v]
      using cont-nec-fact2-1 cont-nec-fact2-4
            cont-nec-fact1-2[deduction] &I by simp
  next
    have [¬(WeaklyContingent (PLM.L⁻)) in v]
      using cont-nec-fact1-1[THEN oth-class-taut-5-d[equiv-lr], equiv-lr]
            cont-nec-fact2-4 by auto
    thus [O! ≠ (PLM.L⁻) in v]
      using cont-nec-fact2-1 cont-nec-fact1-2[deduction] &I by simp
  qed

lemma cont-nec-fact2-6[PLM]:
  [A! ≠ E! & A! ≠ (E!⁻) & A! ≠ PLM.L & A! ≠ (PLM.L⁻) in v]
  proof ((rule &I)+)
    show [A! ≠ E! in v]
      using cont-nec-fact2-2 cont-nec-fact2-3
            cont-nec-fact1-2[deduction] &I by simp
```

```
next
  have [¬(WeaklyContingent (E!⁻)) in v]
    using cont-nec-fact1-1[THEN oth-class-taut-5-d[equiv-lr], equiv-lr]
          cont-nec-fact2-3 by auto
  thus [A! ≠ (E!⁻) in v]
    using cont-nec-fact2-2 cont-nec-fact1-2[deduction] &I by simp
next
  show [A! ≠ PLM.L in v]
    using cont-nec-fact2-2 cont-nec-fact2-4
          cont-nec-fact1-2[deduction] &I by simp
next
  have [¬(WeaklyContingent (PLM.L⁻)) in v]
    using cont-nec-fact1-1[THEN oth-class-taut-5-d[equiv-lr],
            equiv-lr] cont-nec-fact2-4 by auto
  thus [A! ≠ (PLM.L⁻) in v]
    using cont-nec-fact2-2 cont-nec-fact1-2[deduction] &I by simp
qed

lemma id-nec3-1[PLM]:
 [((x^P) =_E (y^P)) ≡ (□((x^P) =_E (y^P)))  in v]
 proof (rule ≡I; rule CP)
  assume [(x^P) =_E (y^P) in v]
  hence [⦇O!,x^P⦈ in v] ∧ [⦇O!,y^P⦈ in v] ∧ [□(∀ F . ⦇F,x^P⦈ ≡ ⦇F,y^P⦈) in v]
    using eq-E-simple-1[equiv-lr] using &E by blast
  hence [□⦇O!,x^P⦈ in v] ∧ [□⦇O!,y^P⦈ in v]
        ∧ [□□(∀ F . ⦇F,x^P⦈ ≡ ⦇F,y^P⦈) in v]
    using oa-facts-1[deduction] S5Basic-6[deduction] by blast
  hence [□(⦇O!,x^P⦈ & ⦇O!,y^P⦈ & □(∀ F. ⦇F,x^P⦈ ≡ ⦇F,y^P⦈)) in v]
    using &I KBasic-3[equiv-rl] by presburger
  thus [□((x^P) =_E (y^P)) in v]
    apply −
    apply (PLM-subst-method
           (⦇O!,x^P⦈ & ⦇O!,y^P⦈ & □(∀ F. ⦇F,x^P⦈ ≡ ⦇F,y^P⦈))
           (x^P) =_E (y^P))
    using eq-E-simple-1[equiv-sym] by auto
 next
  assume [□((x^P) =_E (y^P)) in v]
  thus [((x^P) =_E (y^P)) in v]
  using qml-2[axiom-instance,deduction] by simp
 qed

lemma id-nec3-2[PLM]:
 [◊((x^P) =_E (y^P)) ≡ ((x^P) =_E (y^P)) in v]
 proof (rule ≡I; rule CP)
  assume [◊((x^P) =_E (y^P)) in v]
  thus [(x^P) =_E (y^P) in v]
    using derived-S5-rules-2-b[deduction] id-nec3-1[equiv-lr]
          CP modus-ponens by blast
 next
  assume [(x^P) =_E (y^P) in v]
  thus [◊((x^P) =_E (y^P)) in v]
    by (rule TBasic[deduction])
 qed

lemma thm-neg-eqE[PLM]:
 [((x^P) ≠_E (y^P)) ≡ (¬((x^P) =_E (y^P))) in v]
 proof −
  have [(x^P) ≠_E (y^P) in v] = [⦇(λ² (λ x y . (x^P) =_E (y^P)))⁻, x^P, y^P⦈ in v]
```

unfolding *not-identical*$_E$*-def* **by** *simp*
also have ... = $[\neg(\!|(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ (x^P) =_E (y^P))),\ x^P,\ y^P|\!)\ in\ v]$
unfolding *propnot-defs*
apply (*safe intro*!: *beta-C-meta-2*[*equiv-lr*] *beta-C-meta-2*[*equiv-rl*])
by *show-proper*+
also have ... = $[\neg((x^P) =_E (y^P))\ in\ v]$
apply (*PLM-subst-method*
$(\!|(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ (x^P) =_E (y^P))),\ x^P,\ y^P|\!)$
$(x^P) =_E (y^P))$
apply (*safe intro*!: *beta-C-meta-2*)
unfolding *identity-defs* **by** *show-proper*
finally show *?thesis*
using $\equiv I$ *CP* **by** *presburger*
qed

lemma *id-nec4-1*[*PLM*]:
$[((x^P) \neq_E (y^P)) \equiv \Box((x^P) \neq_E (y^P))\ in\ v]$
proof −
have $[(\neg((x^P) =_E (y^P))) \equiv \Box(\neg((x^P) =_E (y^P)))\ in\ v]$
using *id-nec3-2*[*equiv-sym*] *oth-class-taut-5-d*[*equiv-lr*]
KBasic2-4[*equiv-sym*] *intro-elim-6-e* **by** *fast*
thus *?thesis*
apply −
apply (*PLM-subst-method* $(\neg((x^P) =_E (y^P)))$ $(x^P) \neq_E (y^P))$
using *thm-neg-eqE*[*equiv-sym*] **by** *auto*
qed

lemma *id-nec4-2*[*PLM*]:
$[\Diamond((x^P) \neq_E (y^P)) \equiv ((x^P) \neq_E (y^P))\ in\ v]$
using $\equiv I$ *id-nec4-1*[*equiv-lr*] *derived-S5-rules-2-b* *CP* $T\Diamond$ **by** *simp*

lemma *id-act-1*[*PLM*]:
$[((x^P) =_E (y^P)) \equiv (\boldsymbol{\mathcal{A}}((x^P) =_E (y^P)))\ in\ v]$
proof (*rule* $\equiv I$; *rule CP*)
assume $[(x^P) =_E (y^P)\ in\ v]$
hence $[\Box((x^P) =_E (y^P))\ in\ v]$
using *id-nec3-1*[*equiv-lr*] **by** *auto*
thus $[\boldsymbol{\mathcal{A}}((x^P) =_E (y^P))\ in\ v]$
using *nec-imp-act*[*deduction*] **by** *fast*
next
assume $[\boldsymbol{\mathcal{A}}((x^P) =_E (y^P))\ in\ v]$
hence $[\boldsymbol{\mathcal{A}}((\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)\ \&\ \Box(\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!)))\ in\ v]$
apply −
apply (*PLM-subst-method*
$(x^P) =_E (y^P)$
$((\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)\ \&\ \Box(\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))))$
using *eq-E-simple-1* **by** *auto*
hence $[\boldsymbol{\mathcal{A}}(\!|O!,x^P|\!)\ \&\ \boldsymbol{\mathcal{A}}(\!|O!,y^P|\!)\ \&\ \boldsymbol{\mathcal{A}}(\Box(\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!)))\ in\ v]$
using *Act-Basic-2*[*equiv-lr*] $\&I$ $\&E$ **by** *meson*
thus $[(x^P) =_E (y^P)\ in\ v]$
apply − **apply** (*rule eq-E-simple-1*[*equiv-rl*])
using *oa-facts-7*[*equiv-rl*] *qml-act-2*[*axiom-instance*, *equiv-rl*]
$\&I$ $\&E$ **by** *meson*
qed

lemma *id-act-2*[*PLM*]:
$[((x^P) \neq_E (y^P)) \equiv (\boldsymbol{\mathcal{A}}((x^P) \neq_E (y^P)))\ in\ v]$
apply (*PLM-subst-method* $(\neg((x^P) =_E (y^P)))$ $((x^P) \neq_E (y^P)))$

```
      using thm-neg-eqE[equiv-sym] apply simp
    using id-act-1 oth-class-taut-5-d[equiv-lr] thm-neg-eqE intro-elim-6-e
          logic-actual-nec-1[axiom-instance,equiv-sym] by meson

end

class id-act = id-eq +
  assumes id-act-prop: [𝓐(α = β) in v] ⟹ [(α = β) in v]

instantiation ν :: id-act
begin
  instance proof
    interpret PLM .
    fix x::ν and y::ν and v::i
    assume [𝓐(x = y) in v]
    hence [𝓐(((x^P) =_E (y^P)) ∨ (⦇A!,x^P⦈ & ⦇A!,y^P⦈
            & □(∀ F . ⦃x^P,F⦄ ≡ ⦃y^P,F⦄))) in v]
      unfolding identity-defs by auto
    hence [𝓐(((x^P) =_E (y^P))) ∨ 𝓐((⦇A!,x^P⦈ & ⦇A!,y^P⦈
            & □(∀ F . ⦃x^P,F⦄ ≡ ⦃y^P,F⦄))) in v]
      using Act-Basic-10[equiv-lr] by auto
    moreover {
       assume [𝓐(((x^P) =_E (y^P))) in v]
       hence [(x^P) = (y^P) in v]
        using id-act-1[equiv-rl] eq-E-simple-2[deduction] by auto
    }
    moreover {
       assume [𝓐(⦇A!,x^P⦈ & ⦇A!,y^P⦈ & □(∀ F . ⦃x^P,F⦄ ≡ ⦃y^P,F⦄)) in v]
       hence [𝓐⦇A!,x^P⦈ & 𝓐⦇A!,y^P⦈ & 𝓐(□(∀ F . ⦃x^P,F⦄ ≡ ⦃y^P,F⦄)) in v]
         using Act-Basic-2[equiv-lr] &I &E by meson
       hence [⦇A!,x^P⦈ & ⦇A!,y^P⦈ & (□(∀ F . ⦃x^P,F⦄ ≡ ⦃y^P,F⦄)) in v]
         using oa-facts-8[equiv-rl] qml-act-2[axiom-instance,equiv-rl]
           &I &E by meson
       hence [(x^P) = (y^P) in v]
        unfolding identity-defs using ∨I by auto
    }
    ultimately have [(x^P) = (y^P) in v]
      using intro-elim-4-a CP by meson
    thus [x = y in v]
      unfolding identity-defs by auto
  qed
end

instantiation Π₁ :: id-act
begin
  instance proof
    interpret PLM .
    fix F::Π₁ and G::Π₁ and v::i
    show [𝓐(F = G) in v] ⟹ [(F = G) in v]
      unfolding identity-defs
      using qml-act-2[axiom-instance,equiv-rl] by auto
  qed
end

instantiation o :: id-act
begin
  instance proof
    interpret PLM .
```

fix p :: o **and** q :: o **and** v::i
show $[\boldsymbol{\mathcal{A}}(p = q)\ in\ v] \Longrightarrow [p = q\ in\ v]$
unfolding *identity*$_\text{o}$*-def* **using** *id-act-prop* **by** *blast*
qed
end

instantiation Π_2 :: *id-act*
begin
instance proof
interpret *PLM* **.**
fix F::Π_2 **and** G::Π_2 **and** v::i
assume a: $[\boldsymbol{\mathcal{A}}(F = G)\ in\ v]$
{
fix x
have $[\boldsymbol{\mathcal{A}}((\boldsymbol{\lambda} y.\ (|F,x^P,y^P|)) = (\boldsymbol{\lambda} y.\ (|G,x^P,y^P|))$
& $(\boldsymbol{\lambda} y.\ (|F,y^P,x^P|)) = (\boldsymbol{\lambda} y.\ (|G,y^P,x^P|)))\ in\ v]$
using *a logic-actual-nec-3*[*axiom-instance*, *equiv-lr*] *cqt-basic-4*[*equiv-lr*] $\forall E$
unfolding *identity*$_2$*-def* **by** *fast*
hence $[((\boldsymbol{\lambda} y.\ (|F,x^P,y^P|)) = (\boldsymbol{\lambda} y.\ (|G,x^P,y^P|)))$
& $((\boldsymbol{\lambda} y.\ (|F,y^P,x^P|)) = (\boldsymbol{\lambda} y.\ (|G,y^P,x^P|)))\ in\ v]$
using &I **&**E *id-act-prop Act-Basic-2*[*equiv-lr*] **by** *metis*
}
thus $[F = G\ in\ v]$ **unfolding** *identity-defs* **by** (*rule* $\forall I$)
qed
end

instantiation Π_3 :: *id-act*
begin
instance proof
interpret *PLM* **.**
fix F::Π_3 **and** G::Π_3 **and** v::i
assume a: $[\boldsymbol{\mathcal{A}}(F = G)\ in\ v]$
let $?p = \lambda\ x\ y\ .\ (\boldsymbol{\lambda} z.\ (|F,z^P,x^P,y^P|)) = (\boldsymbol{\lambda} z.\ (|G,z^P,x^P,y^P|))$
& $(\boldsymbol{\lambda} z.\ (|F,x^P,z^P,y^P|)) = (\boldsymbol{\lambda} z.\ (|G,x^P,z^P,y^P|))$
& $(\boldsymbol{\lambda} z.\ (|F,x^P,y^P,z^P|)) = (\boldsymbol{\lambda} z.\ (|G,x^P,y^P,z^P|))$
{
fix x
{
fix y
have $[\boldsymbol{\mathcal{A}}(?p\ x\ y)\ in\ v]$
using *a logic-actual-nec-3*[*axiom-instance*, *equiv-lr*]
cqt-basic-4[*equiv-lr*] $\forall E$[**where** $'a$=ν]
unfolding *identity*$_3$*-def* **by** *blast*
hence $[?p\ x\ y\ in\ v]$
using &I **&**E *id-act-prop Act-Basic-2*[*equiv-lr*] **by** *metis*
}
hence $[\forall\ y\ .\ ?p\ x\ y\ in\ v]$
by (*rule* $\forall I$)
}
thus $[F = G\ in\ v]$
unfolding *identity*$_3$*-def* **by** (*rule* $\forall I$)
qed
end

context *PLM*
begin
lemma *id-act-3*[*PLM*]:
$[((\alpha::('a::id\text{-}act)) = \beta) \equiv \boldsymbol{\mathcal{A}}(\alpha = \beta)\ in\ v]$

using $\equiv I$ *CP id-nec*[*equiv-lr*, *THEN nec-imp-act*[*deduction*]]
id-act-prop **by** *metis*

lemma *id-act-4*[*PLM*]:
$[((\alpha::('a::id\text{-}act)) \neq \beta) \equiv \boldsymbol{\mathcal{A}}(\alpha \neq \beta)\ in\ v]$
using *id-act-3*[*THEN oth-class-taut-5-d*[*equiv-lr*]]
logic-actual-nec-1[*axiom-instance*, *equiv-sym*]
intro-elim-6-e **by** *blast*

lemma *id-act-desc*[*PLM*]:
$[(y^P) = (\boldsymbol{\iota}x\ .\ x = y)\ in\ v]$
using *descriptions*[*axiom-instance*,*equiv-rl*]
id-act-3[*equiv-sym*] $\forall I$ **by** *fast*

lemma *eta-conversion-lemma-1*[*PLM*]:
$[(\boldsymbol{\lambda}\ x\ .\ (\!|F,x^P|\!)) = F\ in\ v]$
using *lambda-predicates-3-1*[*axiom-instance*] **.**

lemma *eta-conversion-lemma-0*[*PLM*]:
$[(\boldsymbol{\lambda}^0\ p) = p\ in\ v]$
using *lambda-predicates-3-0*[*axiom-instance*] **.**

lemma *eta-conversion-lemma-2*[*PLM*]:
$[(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ (\!|F,x^P,y^P|\!))) = F\ in\ v]$
using *lambda-predicates-3-2*[*axiom-instance*] **.**

lemma *eta-conversion-lemma-3*[*PLM*]:
$[(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ (\!|F,x^P,y^P,z^P|\!))) = F\ in\ v]$
using *lambda-predicates-3-3*[*axiom-instance*] **.**

lemma *lambda-p-q-p-eq-q*[*PLM*]:
$[((\boldsymbol{\lambda}^0\ p) = (\boldsymbol{\lambda}^0\ q)) \equiv (p = q)\ in\ v]$
using *eta-conversion-lemma-0*
l-identity[*axiom-instance*, *deduction*, *deduction*]
eta-conversion-lemma-0[*eq-sym*] $\equiv I$ *CP*
by *metis*

A.9.12. The Theory of Objects

lemma *partition-1*[*PLM*]:
$[\forall\ x\ .\ (\!|O!,x^P|\!) \lor (\!|A!,x^P|\!)\ in\ v]$
proof (*rule* $\forall I$)
fix x
have $[\Diamond(\!|E!,x^P|\!) \lor \neg\Diamond(\!|E!,x^P|\!)\ in\ v]$
by *PLM-solver*
moreover have $[\Diamond(\!|E!,x^P|\!) \equiv (\!|\boldsymbol{\lambda}\ y\ .\ \Diamond(\!|E!,y^P|\!),\ x^P|\!)\ in\ v]$
apply (*rule beta-C-meta-1*[*equiv-sym*])
by *show-proper*
moreover have $[(\neg\Diamond(\!|E!,x^P|\!)) \equiv (\!|\boldsymbol{\lambda}\ y\ .\ \neg\Diamond(\!|E!,y^P|\!),\ x^P|\!)\ in\ v]$
apply (*rule beta-C-meta-1*[*equiv-sym*])
by *show-proper*
ultimately show $[(\!|O!,\ x^P|\!) \lor (\!|A!,\ x^P|\!)\ in\ v]$
unfolding *Ordinary-def Abstract-def* **by** *PLM-solver*
qed

lemma *partition-2*[*PLM*]:
$[\neg(\exists\ x\ .\ (\!|O!,x^P|\!)\ \&\ (\!|A!,x^P|\!))\ in\ v]$
proof −

```
    {
      assume [∃ x . ⦇O!,x^P⦈ & ⦇A!,x^P⦈ in v]
      then obtain b where [⦇O!,b^P⦈ & ⦇A!,b^P⦈ in v]
        by (rule ∃E)
      hence ?thesis
        using &E oa-contingent-2[equiv-lr]
              reductio-aa-2 by fast
    }
    thus ?thesis
      using reductio-aa-2 by blast
  qed

lemma ord-eq-Eequiv-1[PLM]:
  [⦇O!,x⦈ → (x =E x) in v]
  proof (rule CP)
    assume [⦇O!,x⦈ in v]
    moreover have [□(∀ F . ⦇F,x⦈ ≡ ⦇F,x⦈) in v]
      by PLM-solver
    ultimately show [(x) =E (x) in v]
      using &I eq-E-simple-1[equiv-rl] by blast
  qed

lemma ord-eq-Eequiv-2[PLM]:
  [(x =E y) → (y =E x) in v]
  proof (rule CP)
    assume [x =E y in v]
    hence 1: [⦇O!,x⦈ & ⦇O!,y⦈ & □(∀ F . ⦇F,x⦈ ≡ ⦇F,y⦈) in v]
      using eq-E-simple-1[equiv-lr] by simp
    have [□(∀ F . ⦇F,y⦈ ≡ ⦇F,x⦈) in v]
      apply (PLM-subst-method
             λ F . ⦇F,x⦈ ≡ ⦇F,y⦈
             λ F . ⦇F,y⦈ ≡ ⦇F,x⦈)
      using oth-class-taut-3-g 1[conj2] by auto
    thus [y =E x in v]
      using eq-E-simple-1[equiv-rl] 1[conj1]
            &E &I by meson
  qed

lemma ord-eq-Eequiv-3[PLM]:
  [((x =E y) & (y =E z)) → (x =E z) in v]
  proof (rule CP)
    assume a: [(x =E y) & (y =E z) in v]
    have [□((∀ F . ⦇F,x⦈ ≡ ⦇F,y⦈) & (∀ F . ⦇F,y⦈ ≡ ⦇F,z⦈)) in v]
      using KBasic-3[equiv-rl] a[conj1, THEN eq-E-simple-1[equiv-lr,conj2]]
            a[conj2, THEN eq-E-simple-1[equiv-lr,conj2]] &I by blast
    moreover {
      {
        fix w
        have [((∀ F . ⦇F,x⦈ ≡ ⦇F,y⦈) & (∀ F . ⦇F,y⦈ ≡ ⦇F,z⦈))
                → (∀ F . ⦇F,x⦈ ≡ ⦇F,z⦈) in w]
          by PLM-solver
      }
      hence [□(((∀ F . ⦇F,x⦈ ≡ ⦇F,y⦈) & (∀ F . ⦇F,y⦈ ≡ ⦇F,z⦈))
              → (∀ F . ⦇F,x⦈ ≡ ⦇F,z⦈)) in v]
        by (rule RN)
    }
    ultimately have [□(∀ F . ⦇F,x⦈ ≡ ⦇F,z⦈) in v]
      using qml-1[axiom-instance,deduction,deduction] by blast
```

thus $[x =_E z \; in \; v]$
using $a[conj1, \; THEN \; eq\text{-}E\text{-}simple\text{-}1[equiv\text{-}lr, conj1, conj1]]$
using $a[conj2, \; THEN \; eq\text{-}E\text{-}simple\text{-}1[equiv\text{-}lr, conj1, conj2]]$
$eq\text{-}E\text{-}simple\text{-}1[equiv\text{-}rl]$ **&**I
by *presburger*
qed

lemma $ord\text{-}eq\text{-}E\text{-}eq[PLM]$:
$[((\!|O!,x^P|\!) \vee (\!|O!,y^P|\!)) \rightarrow ((x^P = y^P) \equiv (x^P =_E y^P)) \; in \; v]$
proof $(rule \; CP)$
assume $[(\!|O!,x^P|\!) \vee (\!|O!,y^P|\!) \; in \; v]$
moreover {
assume $[(\!|O!,x^P|\!) \; in \; v]$
hence $[(x^P = y^P) \equiv (x^P =_E y^P) \; in \; v]$
using $\equiv I \; CP \; l\text{-}identity[axiom\text{-}instance, \; deduction, \; deduction]$
$ord\text{-}eq\text{-}Eequiv\text{-}1[deduction] \; eq\text{-}E\text{-}simple\text{-}2[deduction]$ **by** *metis*
}
moreover {
assume $[(\!|O!,y^P|\!) \; in \; v]$
hence $[(x^P = y^P) \equiv (x^P =_E y^P) \; in \; v]$
using $\equiv I \; CP \; l\text{-}identity[axiom\text{-}instance, \; deduction, \; deduction]$
$ord\text{-}eq\text{-}Eequiv\text{-}1[deduction] \; eq\text{-}E\text{-}simple\text{-}2[deduction] \; id\text{-}eq\text{-}2[deduction]$
$ord\text{-}eq\text{-}Eequiv\text{-}2[deduction] \; identity\text{-}\nu\text{-}def$ **by** *metis*
}
ultimately show $[(x^P = y^P) \equiv (x^P =_E y^P) \; in \; v]$
using $intro\text{-}elim\text{-}4\text{-}a \; CP$ **by** *blast*
qed

lemma $ord\text{-}eq\text{-}E[PLM]$:
$[((\!|O!,x^P|\!) \;\&\; (\!|O!,y^P|\!)) \rightarrow ((\forall \; F \; . \; (\!|F,x^P|\!) \equiv (\!|F,y^P|\!)) \rightarrow x^P =_E y^P) \; in \; v]$
proof $(rule \; CP; \; rule \; CP)$
assume $ord\text{-}xy$: $[(\!|O!,x^P|\!) \;\&\; (\!|O!,y^P|\!) \; in \; v]$
assume $[\forall \; F \; . \; (\!|F,x^P|\!) \equiv (\!|F,y^P|\!) \; in \; v]$
hence $[(\!|\boldsymbol{\lambda} \; z \; . \; z^P =_E x^P, \; x^P|\!) \equiv (\!|\boldsymbol{\lambda} \; z \; . \; z^P =_E x^P, \; y^P|\!) \; in \; v]$
by $(rule \; \forall \; E)$
moreover have $[(\!|\boldsymbol{\lambda} \; z \; . \; z^P =_E x^P, \; x^P|\!) \; in \; v]$
apply $(rule \; beta\text{-}C\text{-}meta\text{-}1[equiv\text{-}rl])$
unfolding $identity_E\text{-}infix\text{-}def$
apply *show-proper*
using $ord\text{-}eq\text{-}Eequiv\text{-}1[deduction] \; ord\text{-}xy[conj1]$
unfolding $identity_E\text{-}infix\text{-}def$ **by** *simp*
ultimately have $[(\!|\boldsymbol{\lambda} \; z \; . \; z^P =_E x^P, \; y^P|\!) \; in \; v]$
using $\equiv E$ **by** *blast*
hence $[y^P =_E x^P \; in \; v]$
unfolding $identity_E\text{-}infix\text{-}def$
apply $(safe \; intro!$:
$beta\text{-}C\text{-}meta\text{-}1[$**where** $\varphi = \lambda \; z \; . \; (\!|basic\text{-}identity_E,z,x^P|\!), \; equiv\text{-}lr])$
by *show-proper*
thus $[x^P =_E y^P \; in \; v]$
by $(rule \; ord\text{-}eq\text{-}Eequiv\text{-}2[deduction])$
qed

lemma $ord\text{-}eq\text{-}E2[PLM]$:
$[((\!|O!,x^P|\!) \;\&\; (\!|O!,y^P|\!)) \rightarrow$
$((x^P \neq y^P) \equiv (\boldsymbol{\lambda} z \; . \; z^P =_E x^P) \neq (\boldsymbol{\lambda} z \; . \; z^P =_E y^P)) \; in \; v]$
proof $(rule \; CP; \; rule \equiv I; \; rule \; CP)$
assume $ord\text{-}xy$: $[(\!|O!,x^P|\!) \;\&\; (\!|O!,y^P|\!) \; in \; v]$
assume $[x^P \neq y^P \; in \; v]$

hence $[\neg(x^P =_E y^P)\ in\ v]$
using *eq-E-simple-2 modus-tollens-1* **by** *fast*
moreover {
assume $[(\boldsymbol{\lambda} z\ .\ z^P =_E x^P) = (\boldsymbol{\lambda} z\ .\ z^P =_E y^P)\ in\ v]$
moreover have $[(\!|\boldsymbol{\lambda} z\ .\ z^P =_E x^P,\ x^P|\!)\ in\ v]$
apply (*rule beta-C-meta-1*[*equiv-rl*])
unfolding *identity*$_E$*-infix-def*
apply *show-proper*
using *ord-eq-Eequiv-1*[*deduction*] *ord-xy*[*conj1*]
unfolding *identity*$_E$*-infix-def* **by** *presburger*
ultimately have $[(\!|\boldsymbol{\lambda} z\ .\ z^P =_E y^P,\ x^P|\!)\ in\ v]$
using *l-identity*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
hence $[x^P =_E y^P\ in\ v]$
unfolding *identity*$_E$*-infix-def*
apply (*safe intro*!:
beta-C-meta-1[**where** $\varphi = \lambda\ z\ .\ (\!|basic\text{-}identity_E,z,y^P|\!)$, *equiv-lr*])
by *show-proper*
}
ultimately show $[(\boldsymbol{\lambda} z\ .\ z^P =_E x^P) \neq (\boldsymbol{\lambda} z\ .\ z^P =_E y^P)\ in\ v]$
using *modus-tollens-1 CP* **by** *blast*
next
assume *ord-xy*: $[(\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)\ in\ v]$
assume $[(\boldsymbol{\lambda} z\ .\ z^P =_E x^P) \neq (\boldsymbol{\lambda} z\ .\ z^P =_E y^P)\ in\ v]$
moreover {
assume $[x^P = y^P\ in\ v]$
hence $[(\boldsymbol{\lambda} z\ .\ z^P =_E x^P) = (\boldsymbol{\lambda} z\ .\ z^P =_E y^P)\ in\ v]$
using *id-eq-1 l-identity*[*axiom-instance*, *deduction*, *deduction*]
by *fast*
}
ultimately show $[x^P \neq y^P\ in\ v]$
using *modus-tollens-1 CP* **by** *blast*
qed

lemma *ab-obey-1*[*PLM*]:
$[((\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)) \rightarrow ((\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv \{\!|y^P,\ F|\!\}) \rightarrow x^P = y^P)\ in\ v]$
proof(*rule CP*; *rule CP*)
assume *abs-xy*: $[(\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ in\ v]$
assume *enc-equiv*: $[\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv \{\!|y^P,\ F|\!\}\ in\ v]$
{
fix P
have $[\{\!|x^P,\ P|\!\} \equiv \{\!|y^P,\ P|\!\}\ in\ v]$
using *enc-equiv* **by** (*rule* $\forall E$)
hence $[\Box(\{\!|x^P,\ P|\!\} \equiv \{\!|y^P,\ P|\!\})\ in\ v]$
using *en-eq-2 intro-elim-6-e intro-elim-6-f*
en-eq-5[*equiv-rl*] **by** *meson*
}
hence $[\Box(\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv \{\!|y^P,\ F|\!\})\ in\ v]$
using *BF*[*deduction*] $\forall I$ **by** *fast*
thus $[x^P = y^P\ in\ v]$
unfolding *identity-defs*
using $\vee I(2)$ *abs-xy* $\&I$ **by** *presburger*
qed

lemma *ab-obey-2*[*PLM*]:
$[((\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)) \rightarrow ((\exists\ F\ .\ \{\!|x^P,\ F|\!\}\ \&\ \neg\{\!|y^P,\ F|\!\}) \rightarrow x^P \neq y^P)\ in\ v]$
proof(*rule CP*; *rule CP*)
assume *abs-xy*: $[(\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ in\ v]$
assume $[\exists\ F\ .\ \{\!|x^P,\ F|\!\}\ \&\ \neg\{\!|y^P,\ F|\!\}\ in\ v]$

then obtain P **where** *P-prop*:
$[\{\!|x^P,\ P|\!\}\ \&\ \neg\{\!|y^P,\ P|\!\}\ in\ v]$
by (*rule* $\exists\,E$)
{
assume $[x^P = y^P\ \ in\ v]$
hence $[\{\!|x^P,\ P|\!\} \equiv \{\!|y^P,\ P|\!\}\ in\ v]$
using *l-identity*[*axiom-instance*, *deduction*, *deduction*]
oth-class-taut-4-a **by** *fast*
hence $[\{\!|y^P,\ P|\!\}\ in\ v]$
using *P-prop*[*conj1*] **by** (*rule* $\equiv E$)
}
thus $[x^P \neq y^P\ \ in\ v]$
using *P-prop*[*conj2*] *modus-tollens-1 CP* **by** *blast*
qed

lemma *ordnecfail*[*PLM*]:
$[(\!|O!,x^P|\!) \rightarrow \Box(\neg(\exists\ F\ .\ \{\!|x^P,\ F|\!\}))\ in\ v]$
proof (*rule CP*)
assume $[(\!|O!,x^P|\!)\ in\ v]$
hence $[\Box(\!|O!,x^P|\!)\ in\ v]$
using *oa-facts-1*[*deduction*] **by** *simp*
moreover hence $[\Box((\!|O!,x^P|\!) \rightarrow (\neg(\exists\ F\ .\ \{\!|x^P,\ F|\!\})))\ in\ v]$
using *nocoder*[*axiom-necessitation*, *axiom-instance*] **by** *simp*
ultimately show $[\Box(\neg(\exists\ F\ .\ \{\!|x^P,\ F|\!\}))\ in\ v]$
using *qml-1*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
qed

lemma *o-objects-exist-1*[*PLM*]:
$[\Diamond(\exists\ x\ .\ (\!|E!,x^P|\!))\ in\ v]$
proof –
have $[\Diamond(\exists\ x\ .\ (\!|E!,x^P|\!)\ \&\ \Diamond(\neg(\!|E!,x^P|\!)))\ in\ v]$
using *qml-4*[*axiom-instance*, *conj1*] **.**
hence $[\Diamond((\exists\ x\ .\ (\!|E!,x^P|\!))\ \&\ (\exists\ x\ .\ \Diamond(\neg(\!|E!,x^P|\!))))\ in\ v]$
using *sign-S5-thm-3*[*deduction*] **by** *fast*
hence $[\Diamond(\exists\ x\ .\ (\!|E!,x^P|\!))\ \&\ \Diamond(\exists\ x\ .\ \Diamond(\neg(\!|E!,x^P|\!)))\ in\ v]$
using *KBasic2-8*[*deduction*] **by** *blast*
thus *?thesis* **using** **&***E* **by** *blast*
qed

lemma *o-objects-exist-2*[*PLM*]:
$[\Box(\exists\ x\ .\ (\!|O!,x^P|\!))\ in\ v]$
apply (*rule RN*) **unfolding** *Ordinary-def*
apply (*PLM-subst-method* $\lambda\ x\ .\ \Diamond(\!|E!,x^P|\!)\ \lambda\ x\ .\ (\!|\boldsymbol{\lambda}y.\ \Diamond(\!|E!,y^P|\!),\ x^P|\!)$)
apply (*safe intro*!: *beta-C-meta-1*[*equiv-sym*])
apply *show-proper*
using *o-objects-exist-1 BF*$\Diamond$[*deduction*] **by** *blast*

lemma *o-objects-exist-3*[*PLM*]:
$[\Box(\neg(\forall\ x\ .\ (\!|A!,x^P|\!)))\ in\ v]$
apply (*PLM-subst-method* $(\exists\,x.\ \neg(\!|A!,x^P|\!))\ \neg(\forall\,x.\ (\!|A!,x^P|\!))$)
using *cqt-further-2*[*equiv-sym*] **apply** *fast*
apply (*PLM-subst-method* $\lambda\ x\ .\ (\!|O!,x^P|\!)\ \lambda\ x\ .\ \neg(\!|A!,x^P|\!)$)
using *oa-contingent-2 o-objects-exist-2* **by** *auto*

lemma *a-objects-exist-1*[*PLM*]:
$[\Box(\exists\ x\ .\ (\!|A!,x^P|\!))\ in\ v]$
proof –
{

```
      fix v
      have [∃ x . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ (F = F)) in v]
        using A-objects[axiom-instance] by simp
      hence [∃ x . ⦇A!,x^P⦈ in v]
        using cqt-further-5[deduction,conj1] by fast
    }
    thus ?thesis by (rule RN)
  qed

lemma a-objects-exist-2[PLM]:
  [□(¬(∀ x . ⦇O!,x^P⦈)) in v]
  apply (PLM-subst-method (∃ x. ¬⦇O!,x^P⦈) ¬(∀ x. ⦇O!,x^P⦈))
   using cqt-further-2[equiv-sym] apply fast
  apply (PLM-subst-method λ x . ⦇A!,x^P⦈ λ x . ¬⦇O!,x^P⦈)
   using oa-contingent-3 a-objects-exist-1 by auto

lemma a-objects-exist-3[PLM]:
  [□(¬(∀ x . ⦇E!,x^P⦈)) in v]
  proof –
    {
      fix v
      have [∃ x . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ (F = F)) in v]
        using A-objects[axiom-instance] by simp
      hence [∃ x . ⦇A!,x^P⦈ in v]
        using cqt-further-5[deduction,conj1] by fast
      then obtain a where
        [⦇A!,a^P⦈ in v]
        by (rule ∃ E)
      hence [¬(◇⦇E!,a^P⦈) in v]
        unfolding Abstract-def
        apply (safe intro!: beta-C-meta-1[equiv-lr])
        by show-proper
      hence [(¬⦇E!,a^P⦈) in v]
        using KBasic2-4[equiv-rl] qml-2[axiom-instance,deduction]
        by simp
      hence [¬(∀ x . ⦇E!,x^P⦈) in v]
        using ∃ I cqt-further-2[equiv-rl]
        by fast
    }
    thus ?thesis
      by (rule RN)
  qed

lemma encoders-are-abstract[PLM]:
  [(∃ F . ⦃x^P, F⦄) → ⦇A!,x^P⦈ in v]
  using nocoder[axiom-instance] contraposition-2
        oa-contingent-2[THEN oth-class-taut-5-d[equiv-lr], equiv-lr]
        useful-tautologies-1[deduction]
        vdash-properties-10 CP by metis

lemma A-objects-unique[PLM]:
  [∃ ! x . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F) in v]
  proof –
    have [∃ x . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F) in v]
      using A-objects[axiom-instance] by simp
    then obtain a where a-prop:
      [⦇A!,a^P⦈ & (∀ F . ⦃a^P, F⦄ ≡ φ F) in v] by (rule ∃ E)
    moreover have [∀ y . ⦇A!,y^P⦈ & (∀ F . ⦃y^P, F⦄ ≡ φ F) → (y = a) in v]
```

proof (*rule* ∀ *I*; *rule CP*)
fix *b*
assume *b-prop*: $[(\!|A!,b^P|\!) \,\&\, (\forall\ F\ .\ \{\!|b^P,\ F|\!\} \equiv \varphi\ F)\ in\ v]$
{
fix *P*
have $[\{\!|b^P,P|\!\} \equiv \{\!|a^P,\ P|\!\}\ in\ v]$
using *a-prop*[*conj2*] *b-prop*[*conj2*] ≡*I* ≡*E*(*1*) ≡*E*(*2*)
CP vdash-properties-10 ∀ *E* **by** *metis*
}
hence $[\forall\ F\ .\ \{\!|b^P,F|\!\} \equiv \{\!|a^P,\ F|\!\}\ in\ v]$
using ∀ *I* **by** *fast*
thus $[b = a\ in\ v]$
unfolding *identity-ν-def*
using *ab-obey-1*[*deduction*, *deduction*]
a-prop[*conj1*] *b-prop*[*conj1*] **&***I* **by** *blast*
qed
ultimately show *?thesis*
unfolding *exists-unique-def*
using **&***I* ∃ *I* **by** *fast*
qed

lemma *obj-oth-1*[*PLM*]:
$[\exists\,!\ x\ .\ (\!|A!,x^P|\!) \,\&\, (\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv (\!|F,\ y^P|\!))\ in\ v]$
using *A-objects-unique* **.**

lemma *obj-oth-2*[*PLM*]:
$[\exists\,!\ x\ .\ (\!|A!,x^P|\!) \,\&\, (\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv ((\!|F,\ y^P|\!) \,\&\, (\!|F,\ z^P|\!)))\ in\ v]$
using *A-objects-unique* **.**

lemma *obj-oth-3*[*PLM*]:
$[\exists\,!\ x\ .\ (\!|A!,x^P|\!) \,\&\, (\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv ((\!|F,\ y^P|\!) \lor (\!|F,\ z^P|\!)))\ in\ v]$
using *A-objects-unique* **.**

lemma *obj-oth-4*[*PLM*]:
$[\exists\,!\ x\ .\ (\!|A!,x^P|\!) \,\&\, (\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv (\Box(\!|F,\ y^P|\!)))\ in\ v]$
using *A-objects-unique* **.**

lemma *obj-oth-5*[*PLM*]:
$[\exists\,!\ x\ .\ (\!|A!,x^P|\!) \,\&\, (\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv (F = G))\ in\ v]$
using *A-objects-unique* **.**

lemma *obj-oth-6*[*PLM*]:
$[\exists\,!\ x\ .\ (\!|A!,x^P|\!) \,\&\, (\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv \Box(\forall\ y\ .\ (\!|G,\ y^P|\!) \rightarrow (\!|F,\ y^P|\!)))\ in\ v]$
using *A-objects-unique* **.**

lemma *A-Exists-1*[*PLM*]:
$[\boldsymbol{\mathcal{A}}(\exists\,!\ x :: ('a :: id\text{-}act)\ .\ \varphi\ x) \equiv (\exists\,!\ x\ .\ \boldsymbol{\mathcal{A}}(\varphi\ x))\ in\ v]$
unfolding *exists-unique-def*
proof (*rule* ≡*I*; *rule CP*)
assume $[\boldsymbol{\mathcal{A}}(\exists\,\alpha.\ \varphi\ \alpha \,\&\, (\forall\,\beta.\ \varphi\ \beta \rightarrow \beta = \alpha))\ in\ v]$
hence $[\exists\,\alpha.\ \boldsymbol{\mathcal{A}}(\varphi\ \alpha \,\&\, (\forall\,\beta.\ \varphi\ \beta \rightarrow \beta = \alpha))\ in\ v]$
using *Act-Basic-11*[*equiv-lr*] **by** *blast*
then obtain *α* **where**
$[\boldsymbol{\mathcal{A}}(\varphi\ \alpha \,\&\, (\forall\,\beta.\ \varphi\ \beta \rightarrow \beta = \alpha))\ in\ v]$
by (*rule* ∃ *E*)
hence *1*: $[\boldsymbol{\mathcal{A}}(\varphi\ \alpha) \,\&\, \boldsymbol{\mathcal{A}}(\forall\,\beta.\ \varphi\ \beta \rightarrow \beta = \alpha)\ in\ v]$
using *Act-Basic-2*[*equiv-lr*] **by** *blast*
find-theorems $\boldsymbol{\mathcal{A}}(?p = ?q)$

have *2*: $[\forall\beta.\ \boldsymbol{\mathcal{A}}(\varphi\ \beta \to \beta = \alpha)\ \mathit{in}\ v]$
using *1*[*conj2*] *logic-actual-nec-3*[*axiom-instance*, *equiv-lr*] **by** *blast*
{
fix β
have $[\boldsymbol{\mathcal{A}}(\varphi\ \beta \to \beta = \alpha)\ \mathit{in}\ v]$
using *2* **by** (*rule* $\forall E$)
hence $[\boldsymbol{\mathcal{A}}(\varphi\ \beta) \to (\beta = \alpha)\ \mathit{in}\ v]$
using *logic-actual-nec-2*[*axiom-instance*, *equiv-lr*, *deduction*]
id-act-3[*equiv-rl*] *CP* **by** *blast*
}
hence $[\forall\ \beta\ .\ \boldsymbol{\mathcal{A}}(\varphi\ \beta) \to (\beta = \alpha)\ \mathit{in}\ v]$
by (*rule* $\forall I$)
thus $[\exists\,\alpha.\ \boldsymbol{\mathcal{A}}\varphi\ \alpha\ \&\ (\forall\beta.\ \boldsymbol{\mathcal{A}}\varphi\ \beta \to \beta = \alpha)\ \mathit{in}\ v]$
using *1*[*conj1*] **&***I* $\exists I$ **by** *fast*
next
assume $[\exists\,\alpha.\ \boldsymbol{\mathcal{A}}\varphi\ \alpha\ \&\ (\forall\beta.\ \boldsymbol{\mathcal{A}}\varphi\ \beta \to \beta = \alpha)\ \mathit{in}\ v]$
then obtain α **where** *1*:
$[\boldsymbol{\mathcal{A}}\varphi\ \alpha\ \&\ (\forall\beta.\ \boldsymbol{\mathcal{A}}\varphi\ \beta \to \beta = \alpha)\ \mathit{in}\ v]$
by (*rule* $\exists E$)
{
fix β
have $[\boldsymbol{\mathcal{A}}(\varphi\ \beta) \to \beta = \alpha\ \mathit{in}\ v]$
using *1*[*conj2*] **by** (*rule* $\forall E$)
hence $[\boldsymbol{\mathcal{A}}(\varphi\ \beta \to \beta = \alpha)\ \mathit{in}\ v]$
using *logic-actual-nec-2*[*axiom-instance*, *equiv-rl*] *id-act-3*[*equiv-lr*]
vdash-properties-10 *CP* **by** *blast*
}
hence $[\forall\ \beta\ .\ \boldsymbol{\mathcal{A}}(\varphi\ \beta \to \beta = \alpha)\ \mathit{in}\ v]$
by (*rule* $\forall I$)
hence $[\boldsymbol{\mathcal{A}}(\forall\ \beta\ .\ \varphi\ \beta \to \beta = \alpha)\ \mathit{in}\ v]$
using *logic-actual-nec-3*[*axiom-instance*, *equiv-rl*] **by** *fast*
hence $[\boldsymbol{\mathcal{A}}(\varphi\ \alpha\ \&\ (\forall\ \beta\ .\ \varphi\ \beta \to \beta = \alpha))\ \mathit{in}\ v]$
using *1*[*conj1*] *Act-Basic-2*[*equiv-rl*] **&***I* **by** *blast*
hence $[\exists\,\alpha.\ \boldsymbol{\mathcal{A}}(\varphi\ \alpha\ \&\ (\forall\beta.\ \varphi\ \beta \to \beta = \alpha))\ \mathit{in}\ v]$
using $\exists I$ **by** *fast*
thus $[\boldsymbol{\mathcal{A}}(\exists\,\alpha.\ \varphi\ \alpha\ \&\ (\forall\beta.\ \varphi\ \beta \to \beta = \alpha))\ \mathit{in}\ v]$
using *Act-Basic-11*[*equiv-rl*] **by** *fast*
qed

lemma *A-Exists-2*[*PLM*]:
$[(\exists\ y\ .\ y^P = (\iota x\ .\ \varphi\ x)) \equiv \boldsymbol{\mathcal{A}}(\exists\,!x\ .\ \varphi\ x)\ \mathit{in}\ v]$
using *actual-desc-1* *A-Exists-1*[*equiv-sym*]
intro-elim-6-e **by** *blast*

lemma *A-descriptions*[*PLM*]:
$[\exists\ y\ .\ y^P = (\iota x\ .\ (\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F))\ \mathit{in}\ v]$
using *A-objects-unique*[*THEN RN*, *THEN nec-imp-act*[*deduction*]]
A-Exists-2[*equiv-rl*] **by** *auto*

lemma *thm-can-terms2*[*PLM*]:
$[(y^P = (\iota x\ .\ (\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F)))$
$\to ((\!|A!,y^P|\!)\ \&\ (\forall\ F\ .\ \{\!|y^P,F|\!\} \equiv \varphi\ F))\ \mathit{in}\ dw]$
using *y-in-2* **by** *auto*

lemma *can-ab2*[*PLM*]:
$[(y^P = (\iota x\ .\ (\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F))) \to (\!|A!,y^P|\!)\ \mathit{in}\ v]$
proof (*rule CP*)
assume $[y^P = (\iota x\ .\ (\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F))\ \mathit{in}\ v]$

hence $[\boldsymbol{\mathcal{A}}(\llparenthesis A!,y^P\rrparenthesis)$ & $\boldsymbol{\mathcal{A}}(\forall\ F\ .\ \{\!|y^P,F|\!\} \equiv \varphi\ F)\ in\ v]$
using *nec-hintikka-scheme*[*equiv-lr*, *conj1*]
Act-Basic-2[*equiv-lr*] **by** *blast*
thus $[(\llparenthesis A!,y^P\rrparenthesis\ in\ v]$
using *oa-facts-8*[*equiv-rl*] &*E* **by** *blast*
qed

lemma *desc-encode*[*PLM*]:
$[\{\!|\boldsymbol{\iota}x\ .\ \llparenthesis A!,x^P\rrparenthesis$ & $(\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F),\ G|\!\} \equiv \varphi\ G\ in\ dw]$
proof −
obtain a **where**
$[a^P = (\boldsymbol{\iota}x\ .\ \llparenthesis A!,x^P\rrparenthesis$ & $(\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F))\ in\ dw]$
using *A-descriptions* **by** (*rule* $\exists$ *E*)
moreover hence $[\{\!|a^P,\ G|\!\} \equiv \varphi\ G\ in\ dw]$
using *hintikka*[*equiv-lr*, *conj1*] &*E* $\forall$ *E* **by** *fast*
ultimately show *?thesis*
using *l-identity*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
qed

lemma *desc-nec-encode*[*PLM*]:
$[\{\!|\boldsymbol{\iota}x\ .\ \llparenthesis A!,x^P\rrparenthesis$ & $(\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F),\ G|\!\} \equiv \boldsymbol{\mathcal{A}}(\varphi\ G)\ in\ v]$
proof −
obtain a **where**
$[a^P = (\boldsymbol{\iota}x\ .\ \llparenthesis A!,x^P\rrparenthesis$ & $(\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F))\ in\ v]$
using *A-descriptions* **by** (*rule* $\exists$ *E*)
moreover {
hence $[\boldsymbol{\mathcal{A}}((\llparenthesis A!,a^P\rrparenthesis$ & $(\forall\ F\ .\ \{\!|a^P,F|\!\} \equiv \varphi\ F))\ in\ v]$
using *nec-hintikka-scheme*[*equiv-lr*, *conj1*] **by** *fast*
hence $[\boldsymbol{\mathcal{A}}(\forall\ F\ .\ \{\!|a^P,F|\!\} \equiv \varphi\ F)\ in\ v]$
using *Act-Basic-2*[*equiv-lr*,*conj2*] **by** *blast*
hence $[\forall\ F\ .\ \boldsymbol{\mathcal{A}}(\ \{\!|a^P,F|\!\} \equiv \varphi\ F)\ in\ v]$
using *logic-actual-nec-3*[*axiom-instance*, *equiv-lr*] **by** *blast*
hence $[\boldsymbol{\mathcal{A}}(\{\!|a^P,\ G|\!\} \equiv \varphi\ G)\ in\ v]$
using $\forall$ *E* **by** *fast*
hence $[\boldsymbol{\mathcal{A}}\{\!|a^P,\ G|\!\} \equiv \boldsymbol{\mathcal{A}}(\varphi\ G)\ in\ v]$
using *Act-Basic-5*[*equiv-lr*] **by** *fast*
hence $[\{\!|a^P,\ G|\!\} \equiv \boldsymbol{\mathcal{A}}(\varphi\ G)\ in\ v]$
using *en-eq-10*[*equiv-sym*] *intro-elim-6-e* **by** *blast*
}
ultimately show *?thesis*
using *l-identity*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
qed

notepad
begin
fix v
let $?x = \boldsymbol{\iota}x\ .\ \llparenthesis A!,x^P\rrparenthesis$ & $(\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv (\exists\ q\ .\ q$ & $F = (\boldsymbol{\lambda}\ y\ .\ q)))$
have $[\Box(\exists\ p\ .\ ContingentlyTrue\ p)\ in\ v]$
using *cont-tf-thm-3 RN* **by** *auto*
hence $[\boldsymbol{\mathcal{A}}(\exists\ p\ .\ ContingentlyTrue\ p)\ in\ v]$
using *nec-imp-act*[*deduction*] **by** *simp*
hence $[\exists\ p\ .\ \boldsymbol{\mathcal{A}}(ContingentlyTrue\ p)\ in\ v]$
using *Act-Basic-11*[*equiv-lr*] **by** *auto*
then obtain p_1 **where**
$[\boldsymbol{\mathcal{A}}(ContingentlyTrue\ p_1)\ in\ v]$
by (*rule* $\exists$ *E*)
hence $[\boldsymbol{\mathcal{A}}p_1\ in\ v]$
unfolding *ContingentlyTrue-def*

```
      using Act-Basic-2[equiv-lr] &E by fast
    hence [𝒜p₁ & 𝒜((λ y . p₁) = (λ y . p₁)) in v]
      using &I id-eq-1[THEN RN, THEN nec-imp-act[deduction]] by fast
    hence [𝒜(p₁ & (λ y . p₁) = (λ y . p₁)) in v]
      using Act-Basic-2[equiv-rl] by fast
    hence [∃ q . 𝒜( q & (λ y . p₁) = (λ y . q)) in v]
      using ∃I by fast
    hence [𝒜(∃ q . q & (λ y . p₁) = (λ y . q)) in v]
      using Act-Basic-11[equiv-rl] by fast
    moreover have [⦃?x, λ y . p₁⦄ ≡ 𝒜(∃ q . q & (λ y . p₁) = (λ y . q)) in v]
      using desc-nec-encode by fast
    ultimately have [⦃?x, λ y . p₁⦄ in v]
      using ≡E by blast
end

lemma Box-desc-encode-1[PLM]:
  [□(φ G) → ⦃(ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄ in v]
  proof (rule CP)
    assume [□(φ G) in v]
    hence [𝒜(φ G) in v]
      using nec-imp-act[deduction] by auto
    thus [⦃ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P,F⦄ ≡ φ F), G⦄ in v]
      using desc-nec-encode[equiv-rl] by simp
  qed

lemma Box-desc-encode-2[PLM]:
  [□(φ G) → □(⦃(ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄ ≡ φ G) in v]
  proof (rule CP)
    assume a: [□(φ G) in v]
    hence [□(⦃(ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄ → φ G) in v]
      using KBasic-1[deduction] by simp
    moreover {
      have [⦃(ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄ in v]
        using a Box-desc-encode-1[deduction] by auto
      hence [□⦃(ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄ in v]
        using encoding[axiom-instance,deduction] by blast
      hence [□(φ G → ⦃(ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄) in v]
        using KBasic-1[deduction] by simp
    }
    ultimately show [□(⦃(ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄
                    ≡ φ G) in v]
      using &I KBasic-4[equiv-rl] by blast
  qed

lemma box-phi-a-1[PLM]:
  assumes [□(∀ F . φ F → □(φ F)) in v]
  shows [(⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)) → □(⦇A!,x^P⦈
         & (∀ F . ⦃x^P, F⦄ ≡ φ F)) in v]
  proof (rule CP)
    assume a: [(⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)) in v]
    have [□⦇A!,x^P⦈ in v]
      using oa-facts-2[deduction] a[conj1] by auto
    moreover have [□(∀ F . ⦃x^P, F⦄ ≡ φ F) in v]
      proof (rule BF[deduction]; rule ∀I)
        fix F
        have ϑ: [□(φ F → □(φ F)) in v]
          using assms[THEN CBF[deduction]] by (rule ∀E)
        moreover have [□(⦃x^P, F⦄ → □⦃x^P, F⦄) in v]
```

```
        using encoding[axiom-necessitation, axiom-instance] by simp
      moreover have [□⦃x^P, F⦄ ≡ □(φ F) in v]
        proof (rule ≡I; rule CP)
          assume [□⦃x^P, F⦄ in v]
          hence [⦃x^P, F⦄ in v]
            using qml-2[axiom-instance, deduction] by blast
          hence [φ F in v]
            using a[conj2] ∀E[where 'a=Π₁] ≡E by blast
          thus [□(φ F) in v]
            using ϑ[THEN qml-2[axiom-instance, deduction], deduction] by simp
        next
          assume [□(φ F) in v]
          hence [φ F in v]
            using qml-2[axiom-instance, deduction] by blast
          hence [⦃x^P, F⦄ in v]
            using a[conj2] ∀E[where 'a=Π₁] ≡E by blast
          thus [□⦃x^P, F⦄ in v]
            using encoding[axiom-instance, deduction] by simp
        qed
      ultimately show [□(⦃x^P,F⦄ ≡ φ F) in v]
        using sc-eq-box-box-3[deduction, deduction] &I by blast
    qed
   ultimately show [□(⦇A!,x^P⦈ & (∀F. ⦃x^P,F⦄ ≡ φ F)) in v]
    using &I KBasic-3[equiv-rl] by blast
 qed

lemma box-phi-a-2[PLM]:
 assumes [□(∀ F . φ F → □(φ F)) in v]
 shows [y^P = (ιx . ⦇A!,x^P⦈ & (∀ F. ⦃x^P, F⦄ ≡ φ F))
       → (⦇A!,y^P⦈ & (∀ F . ⦃y^P, F⦄ ≡ φ F)) in v]
 proof −
   let ?ψ = λ x . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)
   have [∀ x . ?ψ x → □(?ψ x) in v]
     using box-phi-a-1[OF assms] ∀I by fast
   hence [(∃! x . ?ψ x) → (∀ y . y^P = (ιx . ?ψ x) → ?ψ y) in v]
     using unique-box-desc[deduction] by fast
   hence [(∀ y . y^P = (ιx . ?ψ x) → ?ψ y) in v]
     using A-objects-unique modus-ponens by blast
   thus ?thesis by (rule ∀E)
 qed

lemma box-phi-a-3[PLM]:
 assumes [□(∀ F . φ F → □(φ F)) in v]
 shows [⦃ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F), G⦄ ≡ φ G in v]
 proof −
   obtain a where
     [a^P = (ιx . ⦇A!,x^P⦈ & (∀ F . ⦃x^P, F⦄ ≡ φ F)) in v]
     using A-descriptions by (rule ∃E)
   moreover {
     hence [(∀ F . ⦃a^P, F⦄ ≡ φ F) in v]
       using box-phi-a-2[OF assms, deduction, conj2] by blast
     hence [⦃a^P, G⦄ ≡ φ G in v] by (rule ∀E)
   }
   ultimately show ?thesis
     using l-identity[axiom-instance, deduction, deduction] by fast
 qed

lemma null-uni-uniq-1[PLM]:
```

$[\exists!\ x\ .\ \mathit{Null}\ (x^P)\ \mathit{in}\ v]$
proof –
have $[\exists\ x\ .\ ⦇A!,x^P⦈\ \&\ (\forall\ F\ .\ \{\!|x^P,\ F|\!\}\equiv(F\neq F))\ \mathit{in}\ v]$
using *A-objects*[*axiom-instance*] **by** *simp*
then obtain a **where** *a-prop*:
$[⦇A!,a^P⦈\ \&\ (\forall\ F\ .\ \{\!|a^P,\ F|\!\}\equiv(F\neq F))\ \mathit{in}\ v]$
by (*rule* $\exists E$)
have *1*: $[⦇A!,a^P⦈\ \&\ (\neg(\exists\ F\ .\ \{\!|a^P,\ F|\!\}))\ \mathit{in}\ v]$
using *a-prop*[*conj1*] **apply** (*rule* **&***I*)
proof –
{
assume $[\exists\ F\ .\ \{\!|a^P,\ F|\!\}\ \mathit{in}\ v]$
then obtain P **where**
$[\{\!|a^P,\ P|\!\}\ \mathit{in}\ v]$ **by** (*rule* $\exists E$)
hence $[P\neq P\ \mathit{in}\ v]$
using *a-prop*[*conj2*, *THEN* $\forall E$, *equiv-lr*] **by** *simp*
hence $[\neg(\exists\ F\ .\ \{\!|a^P,\ F|\!\})\ \mathit{in}\ v]$
using *id-eq-1 reductio-aa-1* **by** *fast*
}
thus $[\neg(\exists\ F\ .\ \{\!|a^P,\ F|\!\})\ \mathit{in}\ v]$
using *reductio-aa-1* **by** *blast*
qed
moreover have $[\forall\ y\ .\ (⦇A!,y^P⦈\ \&\ (\neg(\exists\ F\ .\ \{\!|y^P,\ F|\!\})))\rightarrow y=a\ \mathit{in}\ v]$
proof (*rule* $\forall I$; *rule CP*)
fix y
assume *2*: $[⦇A!,y^P⦈\ \&\ (\neg(\exists\ F\ .\ \{\!|y^P,\ F|\!\}))\ \mathit{in}\ v]$
have $[\forall\ F\ .\ \{\!|y^P,\ F|\!\}\equiv\{\!|a^P,\ F|\!\}\ \mathit{in}\ v]$
using *cqt-further-12*[*deduction*] *1*[*conj2*] *2*[*conj2*] **&***I* **by** *blast*
thus $[y=a\ \mathit{in}\ v]$
using *ab-obey-1*[*deduction*, *deduction*]
&*I*[*OF 2*[*conj1*] *1*[*conj1*]] *identity-ν-def* **by** *presburger*
qed
ultimately show *?thesis*
using **&***I* $\exists I$
unfolding *Null-def exists-unique-def* **by** *fast*
qed

lemma *null-uni-uniq-2*[*PLM*]:
$[\exists!\ x\ .\ \mathit{Universal}\ (x^P)\ \mathit{in}\ v]$
proof –
have $[\exists\ x\ .\ ⦇A!,x^P⦈\ \&\ (\forall\ F\ .\ \{\!|x^P,\ F|\!\}\equiv(F=F))\ \mathit{in}\ v]$
using *A-objects*[*axiom-instance*] **by** *simp*
then obtain a **where** *a-prop*:
$[⦇A!,a^P⦈\ \&\ (\forall\ F\ .\ \{\!|a^P,\ F|\!\}\equiv(F=F))\ \mathit{in}\ v]$
by (*rule* $\exists E$)
have *1*: $[⦇A!,a^P⦈\ \&\ (\forall\ F\ .\ \{\!|a^P,\ F|\!\})\ \mathit{in}\ v]$
using *a-prop*[*conj1*] **apply** (*rule* **&***I*)
using $\forall I$ *a-prop*[*conj2*, *THEN* $\forall E$, *equiv-rl*] *id-eq-1* **by** *fast*
moreover have $[\forall\ y\ .\ (⦇A!,y^P⦈\ \&\ (\forall\ F\ .\ \{\!|y^P,\ F|\!\}))\rightarrow y=a\ \mathit{in}\ v]$
proof (*rule* $\forall I$; *rule CP*)
fix y
assume *2*: $[⦇A!,y^P⦈\ \&\ (\forall\ F\ .\ \{\!|y^P,\ F|\!\})\ \mathit{in}\ v]$
have $[\forall\ F\ .\ \{\!|y^P,\ F|\!\}\equiv\{\!|a^P,\ F|\!\}\ \mathit{in}\ v]$
using *cqt-further-11*[*deduction*] *1*[*conj2*] *2*[*conj2*] **&***I* **by** *blast*
thus $[y=a\ \mathit{in}\ v]$
using *ab-obey-1*[*deduction*, *deduction*]
&*I*[*OF 2*[*conj1*] *1*[*conj1*]] *identity-ν-def*
by *presburger*

```
      qed
    ultimately show ?thesis
      using &I ∃ I
      unfolding Universal-def exists-unique-def by fast
  qed

lemma null-uni-uniq-3[PLM]:
  [∃ y . y^P = (ιx . Null (x^P)) in v]
  using null-uni-uniq-1[THEN RN, THEN nec-imp-act[deduction]]
        A-Exists-2[equiv-rl] by auto

lemma null-uni-uniq-4[PLM]:
  [∃ y . y^P = (ιx . Universal (x^P)) in v]
  using null-uni-uniq-2[THEN RN, THEN nec-imp-act[deduction]]
        A-Exists-2[equiv-rl] by auto

lemma null-uni-facts-1[PLM]:
  [Null (x^P) → □(Null (x^P)) in v]
  proof (rule CP)
    assume [Null (x^P) in v]
    hence 1: [(|A!,x^P|) & (¬(∃ F . {|x^P,F|})) in v]
      unfolding Null-def .
    have [□(|A!,x^P|) in v]
      using 1[conj1] oa-facts-2[deduction] by simp
    moreover have [□(¬(∃ F . {|x^P,F|})) in v]
      proof –
        {
          assume [¬□(¬(∃ F . {|x^P,F|})) in v]
          hence [◇(∃ F . {|x^P,F|}) in v]
            unfolding diamond-def .
          hence [∃ F . ◇{|x^P,F|} in v]
            using BF◇[deduction] by blast
          then obtain P where [◇{|x^P,P|} in v]
            by (rule ∃ E)
          hence [{|x^P, P|} in v]
            using en-eq-3[equiv-lr] by simp
          hence [∃  F . {|x^P, F|} in v]
            using ∃ I by fast
        }
        thus ?thesis
          using 1[conj2] modus-tollens-1 CP
                useful-tautologies-1[deduction] by metis
      qed
    ultimately show [□Null (x^P) in v]
      unfolding Null-def
      using &I KBasic-3[equiv-rl] by blast
  qed

lemma null-uni-facts-2[PLM]:
  [Universal (x^P) → □(Universal (x^P)) in v]
  proof (rule CP)
    assume [Universal (x^P) in v]
    hence 1: [(|A!,x^P|) & (∀ F . {|x^P,F|}) in v]
      unfolding Universal-def .
    have [□(|A!,x^P|) in v]
      using 1[conj1] oa-facts-2[deduction] by simp
    moreover have [□(∀ F . {|x^P,F|}) in v]
      proof (rule BF[deduction]; rule ∀ I)
```

```
      fix F
      have [⦃x^P, F⦄ in v]
        using 1[conj2] by (rule ∀ E)
      thus [□⦃x^P, F⦄ in v]
        using encoding[axiom-instance, deduction] by auto
    qed
  ultimately show [□Universal (x^P) in v]
    unfolding Universal-def
    using &I KBasic-3[equiv-rl] by blast
qed

lemma null-uni-facts-3[PLM]:
  [Null (a_∅) in v]
proof −
  let ?ψ = λ x . Null x
  have [((∃! x . ?ψ (x^P)) → (∀ y . y^P = (ιx . ?ψ (x^P)) → ?ψ (y^P))) in v]
    using unique-box-desc[deduction] null-uni-facts-1[THEN ∀ I] by fast
  have 1: [(∀ y . y^P = (ιx . ?ψ (x^P)) → ?ψ (y^P)) in v]
    using unique-box-desc[deduction, deduction] null-uni-uniq-1
          null-uni-facts-1[THEN ∀ I] by fast
  have [∃ y . y^P = (a_∅) in v]
    unfolding NullObject-def using null-uni-uniq-3 .
  then obtain y where [y^P = (a_∅) in v]
    by (rule ∃ E)
  moreover hence [?ψ (y^P) in v]
    using 1[THEN ∀ E, deduction] unfolding NullObject-def by simp
  ultimately show [?ψ (a_∅) in v]
    using l-identity[axiom-instance, deduction, deduction] by blast
qed

lemma null-uni-facts-4[PLM]:
  [Universal (a_V) in v]
proof −
  let ?ψ = λ x . Universal x
  have [((∃! x . ?ψ (x^P)) → (∀ y . y^P = (ιx . ?ψ (x^P)) → ?ψ (y^P))) in v]
    using unique-box-desc[deduction] null-uni-facts-2[THEN ∀ I] by fast
  have 1: [(∀ y . y^P = (ιx . ?ψ (x^P)) → ?ψ (y^P)) in v]
    using unique-box-desc[deduction, deduction] null-uni-uniq-2
          null-uni-facts-2[THEN ∀ I] by fast
  have [∃ y . y^P = (a_V) in v]
    unfolding UniversalObject-def using null-uni-uniq-4 .
  then obtain y where [y^P = (a_V) in v]
    by (rule ∃ E)
  moreover hence [?ψ (y^P) in v]
    using 1[THEN ∀ E, deduction]
    unfolding UniversalObject-def by simp
  ultimately show [?ψ (a_V) in v]
    using l-identity[axiom-instance, deduction, deduction] by blast
qed

lemma aclassical-1[PLM]:
  [∀ R . ∃ x y . ⦇A!,x^P⦈ & ⦇A!,y^P⦈ & (x ≠ y)
    & (λ z . ⦇R,z^P,x^P⦈) = (λ z . ⦇R,z^P,y^P⦈) in v]
proof (rule ∀ I)
  fix R
  obtain a where ϑ:
    [⦇A!,a^P⦈ & (∀ F . ⦃a^P, F⦄ ≡ (∃ y . ⦇A!,y^P⦈
      & F = (λ z . ⦇R,z^P,y^P⦈) & ¬⦃y^P, F⦄)) in v]
```

using *A-objects*[*axiom-instance*] **by** (*rule* $\exists E$)
{
assume $[\neg\{\!\!\{a^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!))\}\!\!\}\ in\ v]$
hence $[\neg((\!|A!,a^P|\!)\ \&\ (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!)) = (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!))$
$\&\ \neg\{\!\!\{a^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!))\}\!\!\})\ in\ v]$
using ϑ[*conj2*, *THEN* $\forall E$, *THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-lr*]
cqt-further-4[*equiv-lr*] $\forall E$ **by** *fast*
hence $[(\!|A!,a^P|\!)\ \&\ (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!)) = (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!))$
$\rightarrow \{\!\!\{a^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!))\}\!\!\}\ in\ v]$
apply − **by** *PLM-solver*
hence $[\{\!\!\{a^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!))\}\!\!\}\ in\ v]$
using ϑ[*conj1*] *id-eq-1* **&***I vdash-properties-10* **by** *fast*
}
hence *1*: $[\{\!\!\{a^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!))\}\!\!\}\ in\ v]$
using *reductio-aa-1 CP if-p-then-p* **by** *blast*
then obtain b **where** ξ:
$[(\!|A!,b^P|\!)\ \&\ (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!)) = (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,b^P|\!))$
$\&\ \neg\{\!\!\{b^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!))\}\!\!\}\ in\ v]$
using ϑ[*conj2*, *THEN* $\forall E$, *equiv-lr*] $\exists E$ **by** *blast*
have $[a \neq b\ in\ v]$
proof −
{
assume $[a = b\ in\ v]$
hence $[\{\!\!\{b^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!))\}\!\!\}\ in\ v]$
using *1 l-identity*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
hence *?thesis*
using ξ[*conj2*] *reductio-aa-1* **by** *blast*
}
thus *?thesis* **using** *reductio-aa-1* **by** *blast*
qed
hence $[(\!|A!,a^P|\!)\ \&\ (\!|A!,b^P|\!)\ \&\ a \neq b$
$\&\ (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,a^P|\!)) = (\boldsymbol{\lambda}\ z\ .\ (\!|R,z^P,b^P|\!))\ in\ v]$
using ϑ[*conj1*] ξ[*conj1*, *conj1*] ξ[*conj1*, *conj2*] **&***I* **by** *presburger*
hence $[\exists\ y\ .\ (\!|A!,a^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ a \neq y$
$\&\ (\boldsymbol{\lambda}z.\ (\!|R,z^P,a^P|\!)) = (\boldsymbol{\lambda}z.\ (\!|R,z^P,y^P|\!))\ in\ v]$
using $\exists I$ **by** *fast*
thus $[\exists\ x\ y\ .\ (\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ x \neq y$
$\&\ (\boldsymbol{\lambda}z.\ (\!|R,z^P,x^P|\!)) = (\boldsymbol{\lambda}z.\ (\!|R,z^P,y^P|\!))\ in\ v]$
using $\exists I$ **by** *fast*
qed

lemma *aclassical-2*[*PLM*]:
$[\forall\ R\ .\ \exists\ x\ y\ .\ (\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ (x \neq y)$
$\&\ (\boldsymbol{\lambda}\ z\ .\ (\!|R,x^P,z^P|\!)) = (\boldsymbol{\lambda}\ z\ .\ (\!|R,y^P,z^P|\!))\ in\ v]$
proof (*rule* $\forall I$)
fix R
obtain a **where** ϑ:
$[(\!|A!,a^P|\!)\ \&\ (\forall\ F\ .\ \{\!\!\{a^P, F\}\!\!\} \equiv (\exists\ y\ .\ (\!|A!,y^P|\!)$
$\&\ F = (\boldsymbol{\lambda}\ z\ .\ (\!|R,y^P,z^P|\!))\ \&\ \neg\{\!\!\{y^P, F\}\!\!\}))\ in\ v]$
using *A-objects*[*axiom-instance*] **by** (*rule* $\exists E$)
{
assume $[\neg\{\!\!\{a^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,a^P,z^P|\!))\}\!\!\}\ in\ v]$
hence $[\neg((\!|A!,a^P|\!)\ \&\ (\boldsymbol{\lambda}\ z\ .\ (\!|R,a^P,z^P|\!)) = (\boldsymbol{\lambda}\ z\ .\ (\!|R,a^P,z^P|\!))$
$\&\ \neg\{\!\!\{a^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,a^P,z^P|\!))\}\!\!\})\ in\ v]$
using ϑ[*conj2*, *THEN* $\forall E$, *THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-lr*]
cqt-further-4[*equiv-lr*] $\forall E$ **by** *fast*
hence $[(\!|A!,a^P|\!)\ \&\ (\boldsymbol{\lambda}\ z\ .\ (\!|R,a^P,z^P|\!)) = (\boldsymbol{\lambda}\ z\ .\ (\!|R,a^P,z^P|\!))$
$\rightarrow \{\!\!\{a^P, (\boldsymbol{\lambda}\ z\ .\ (\!|R,a^P,z^P|\!))\}\!\!\}\ in\ v]$

apply – **by** *PLM-solver*
hence $[\{\!| a^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P,z^P \rrparenthesis) |\!\}\ in\ v]$
using ϑ[*conj1*] *id-eq-1* **&***I* *vdash-properties-10* **by** *fast*
}
hence *1*: $[\{\!| a^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P,z^P \rrparenthesis) |\!\}\ in\ v]$
using *reductio-aa-1 CP if-p-then-p* **by** *blast*
then obtain b **where** ξ:
$[\llparenthesis A!,b^P \rrparenthesis\ \&\ (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P,z^P \rrparenthesis) = (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,b^P,z^P \rrparenthesis)$
$\&\ \neg\{\!| b^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P,z^P \rrparenthesis) |\!\}\ in\ v]$
using ϑ[*conj2*, *THEN* $\forall E$, *equiv-lr*] $\exists E$ **by** *blast*
have $[a \neq b\ in\ v]$
proof –
{
assume $[a = b\ in\ v]$
hence $[\{\!| b^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P,z^P \rrparenthesis) |\!\}\ in\ v]$
using *1 l-identity*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
hence *?thesis* **using** ξ[*conj2*] *reductio-aa-1* **by** *blast*
}
thus *?thesis* **using** ξ[*conj2*] *reductio-aa-1* **by** *blast*
qed
hence $[\llparenthesis A!,a^P \rrparenthesis\ \&\ \llparenthesis A!,b^P \rrparenthesis\ \&\ a \neq b$
$\&\ (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P,z^P \rrparenthesis) = (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,b^P,z^P \rrparenthesis)\ in\ v]$
using ϑ[*conj1*] ξ[*conj1*, *conj1*] ξ[*conj1*, *conj2*] **&***I* **by** *presburger*
hence $[\exists\ y\ .\ \llparenthesis A!,a^P \rrparenthesis\ \&\ \llparenthesis A!,y^P \rrparenthesis\ \&\ a \neq y$
$\&\ (\boldsymbol{\lambda} z.\ \llparenthesis R,a^P,z^P \rrparenthesis) = (\boldsymbol{\lambda} z.\ \llparenthesis R,y^P,z^P \rrparenthesis)\ in\ v]$
using $\exists I$ **by** *fast*
thus $[\exists\ x\ y\ .\ \llparenthesis A!,x^P \rrparenthesis\ \&\ \llparenthesis A!,y^P \rrparenthesis\ \&\ x \neq y$
$\&\ (\boldsymbol{\lambda} z.\ \llparenthesis R,x^P,z^P \rrparenthesis) = (\boldsymbol{\lambda} z.\ \llparenthesis R,y^P,z^P \rrparenthesis)\ in\ v]$
using $\exists I$ **by** *fast*
qed

lemma *aclassical-3*[*PLM*]:
$[\forall\ F\ .\ \exists\ x\ y\ .\ \llparenthesis A!,x^P \rrparenthesis\ \&\ \llparenthesis A!,y^P \rrparenthesis\ \&\ (x \neq y)$
$\&\ ((\boldsymbol{\lambda}^0\ \llparenthesis F,x^P \rrparenthesis) = (\boldsymbol{\lambda}^0\ \llparenthesis F,y^P \rrparenthesis))\ in\ v]$
proof (*rule* $\forall I$)
fix R
obtain a **where** ϑ:
$[\llparenthesis A!,a^P \rrparenthesis\ \&\ (\forall\ F\ .\ \{\!| a^P, F |\!\} \equiv (\exists\ y\ .\ \llparenthesis A!,y^P \rrparenthesis$
$\&\ F = (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,y^P \rrparenthesis)\ \&\ \neg\{\!| y^P, F |\!\}))\ in\ v]$
using *A-objects*[*axiom-instance*] **by** (*rule* $\exists E$)
{
assume $[\neg\{\!| a^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis) |\!\}\ in\ v]$
hence $[\neg(\llparenthesis A!,a^P \rrparenthesis\ \&\ (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis) = (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis)$
$\&\ \neg\{\!| a^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis) |\!\})\ in\ v]$
using ϑ[*conj2*, *THEN* $\forall E$, *THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-lr*]
cqt-further-4[*equiv-lr*] $\forall E$ **by** *fast*
hence $[\llparenthesis A!,a^P \rrparenthesis\ \&\ (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis) = (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis)$
$\rightarrow \{\!| a^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis) |\!\}\ in\ v]$
apply – **by** *PLM-solver*
hence $[\{\!| a^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis) |\!\}\ in\ v]$
using ϑ[*conj1*] *id-eq-1* **&***I* *vdash-properties-10* **by** *fast*
}
hence *1*: $[\{\!| a^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis) |\!\}\ in\ v]$
using *reductio-aa-1 CP if-p-then-p* **by** *blast*
then obtain b **where** ξ:
$[\llparenthesis A!,b^P \rrparenthesis\ \&\ (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis) = (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,b^P \rrparenthesis)$
$\&\ \neg\{\!| b^P, (\boldsymbol{\lambda}\ z\ .\ \llparenthesis R,a^P \rrparenthesis) |\!\}\ in\ v]$
using ϑ[*conj2*, *THEN* $\forall E$, *equiv-lr*] $\exists E$ **by** *blast*

```
    have [a ≠ b in v]
      proof –
        {
          assume [a = b in v]
          hence [⦃b^P, (λ z . ⦇R,a^P⦈)⦄ in v]
            using 1 l-identity[axiom-instance, deduction, deduction] by fast
          hence ?thesis
            using ξ[conj2] reductio-aa-1 by blast
        }
        thus ?thesis using reductio-aa-1 by blast
      qed
    moreover {
      have [⦇R,a^P⦈ = ⦇R,b^P⦈ in v]
        unfolding identity_o-def
        using ξ[conj1, conj2] by auto
      hence [(λ^0 ⦇R,a^P⦈) = (λ^0 ⦇R,b^P⦈) in v]
        using lambda-p-q-p-eq-q[equiv-rl] by simp
    }
    ultimately have [⦇A!,a^P⦈ & ⦇A!,b^P⦈ & a ≠ b
              & ((λ^0 ⦇R,a^P⦈) =(λ^0 ⦇R,b^P⦈)) in v]
      using ϑ[conj1] ξ[conj1, conj1] ξ[conj1, conj2] &I
      by presburger
    hence [∃ y . ⦇A!,a^P⦈ & ⦇A!,y^P⦈ & a ≠ y
            & (λ^0 ⦇R,a^P⦈) = (λ^0 ⦇R,y^P⦈) in v]
      using ∃ I by fast
    thus [∃ x y . ⦇A!,x^P⦈ & ⦇A!,y^P⦈ & x ≠ y
          & (λ^0 ⦇R,x^P⦈) = (λ^0 ⦇R,y^P⦈) in v]
      using ∃ I by fast
  qed

lemma aclassical2[PLM]:
  [∃ x y . ⦇A!,x^P⦈ & ⦇A!,y^P⦈ & x ≠ y & (∀ F . ⦇F,x^P⦈ ≡ ⦇F,y^P⦈) in v]
  proof –
    let ?R_1 = λ^2 (λ x y . ∀ F . ⦇F,x^P⦈ ≡ ⦇F,y^P⦈)
    have [∃ x y . ⦇A!,x^P⦈ & ⦇A!,y^P⦈ & x ≠ y
          & (λz. ⦇?R_1,z^P,x^P⦈) = (λz. ⦇?R_1,z^P,y^P⦈) in v]
      using aclassical-1 by (rule ∀ E)
    then obtain a where
      [∃ y . ⦇A!,a^P⦈ & ⦇A!,y^P⦈ & a ≠ y
        & (λz. ⦇?R_1,z^P,a^P⦈) = (λz. ⦇?R_1,z^P,y^P⦈) in v]
      by (rule ∃ E)
    then obtain b where ab-prop:
      [⦇A!,a^P⦈ & ⦇A!,b^P⦈ & a ≠ b
        & (λz. ⦇?R_1,z^P,a^P⦈) = (λz. ⦇?R_1,z^P,b^P⦈) in v]
      by (rule ∃ E)
    have [⦇?R_1, a^P, a^P⦈ in v]
      apply (rule beta-C-meta-2[equiv-rl])
       apply show-proper
      using oth-class-taut-4-a[THEN ∀ I] by fast
    hence [⦇λ z . ⦇?R_1, z^P, a^P⦈, a^P⦈ in v]
      apply – apply (rule beta-C-meta-1[equiv-rl])
       apply show-proper
      by auto
    hence [⦇λ z . ⦇?R_1, z^P, b^P⦈, a^P⦈ in v]
      using ab-prop[conj2] l-identity[axiom-instance, deduction, deduction]
      by fast
    hence [⦇?R_1, a^P, b^P⦈ in v]
      apply (safe intro!: beta-C-meta-1[where φ=
```

λz . $(\!|\boldsymbol{\lambda}^2$ $(\lambda x\ y.\ \forall F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!)),z,b^P|\!)$, *equiv-lr*])
by *show-proper*
moreover have *IsProperInXY* $(\lambda x\ y.\ \forall F.\ (\!|F,x|\!) \equiv (\!|F,y|\!))$
by *show-proper*
ultimately have $[\forall F.\ (\!|F,a^P|\!) \equiv (\!|F,b^P|\!)$ *in v*]
using *beta-C-meta-2*[*equiv-lr*] **by** *blast*
hence $[(\!|A!,a^P|\!)$ **&** $(\!|A!,b^P|\!)$ **&** $a \neq b$ **&** $(\forall F.\ (\!|F,a^P|\!) \equiv (\!|F,b^P|\!))$ *in v*]
using *ab-prop*[*conj1*] **&***I* **by** *presburger*
hence $[\exists\ y\ .\ (\!|A!,a^P|\!)$ **&** $(\!|A!,y^P|\!)$ **&** $a \neq y$ **&** $(\forall F.\ (\!|F,a^P|\!) \equiv (\!|F,y^P|\!))$ *in v*]
using $\exists I$ **by** *fast*
thus *?thesis* **using** $\exists I$ **by** *fast*
qed

A.9.13. Propositional Properties

lemma *prop-prop2-1*:
$[\forall\ p\ .\ \exists\ F\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)$ *in v*]
proof (*rule* $\forall I$)
fix p
have $[(\boldsymbol{\lambda}\ x\ .\ p) = (\boldsymbol{\lambda}\ x\ .\ p)$ *in v*]
using *id-eq-prop-prop-1* **by** *auto*
thus $[\exists\ F\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)$ *in v*]
by *PLM-solver*
qed

lemma *prop-prop2-2*:
$[F = (\boldsymbol{\lambda}\ x\ .\ p) \rightarrow \Box(\forall\ x\ .\ (\!|F,x^P|\!) \equiv p)$ *in v*]
proof (*rule CP*)
assume *1*: $[F = (\boldsymbol{\lambda}\ x\ .\ p)$ *in v*]
{
fix v
{
fix x
have $[(\!|(\boldsymbol{\lambda}\ x\ .\ p),\ x^P|\!) \equiv p$ *in v*]
apply (*rule beta-C-meta-1*)
by *show-proper*
}
hence $[\forall\ x\ .\ (\!|(\boldsymbol{\lambda}\ x\ .\ p),\ x^P|\!) \equiv p$ *in v*]
by (*rule* $\forall I$)
}
hence $[\Box(\forall\ x\ .\ (\!|(\boldsymbol{\lambda}\ x\ .\ p),\ x^P|\!) \equiv p)$ *in v*]
by (*rule RN*)
thus $[\Box(\forall x.\ (\!|F,x^P|\!) \equiv p)$ *in v*]
using *l-identity*[*axiom-instance*,*deduction*,*deduction*,
OF 1[*THEN id-eq-prop-prop-2*[*deduction*]]] **by** *fast*
qed

lemma *prop-prop2-3*:
[*Propositional F* $\rightarrow \Box$(*Propositional F*) *in v*]
proof (*rule CP*)
assume [*Propositional F in v*]
hence $[\exists\ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)$ *in v*]
unfolding *Propositional-def* **.**
then obtain q **where** $[F = (\boldsymbol{\lambda}\ x\ .\ q)$ *in v*]
by (*rule* $\exists E$)
hence $[\Box(F = (\boldsymbol{\lambda}\ x\ .\ q))$ *in v*]
using *id-nec*[*equiv-lr*] **by** *auto*
hence $[\exists\ p\ .\ \Box(F = (\boldsymbol{\lambda}\ x\ .\ p))$ *in v*]

```
      using ∃I by fast
    thus [□(Propositional F) in v]
      unfolding Propositional-def
      using sign-S5-thm-1[deduction] by fast
  qed


lemma prop-indis:
  [Indiscriminate F → (¬(∃ x y . ⦇F,x^P⦈ & (¬⦇F,y^P⦈))) in v]
  proof (rule CP)
    assume [Indiscriminate F in v]
    hence 1: [□((∃x. ⦇F,x^P⦈) → (∀x. ⦇F,x^P⦈)) in v]
      unfolding Indiscriminate-def .
    {
      assume [∃ x y . ⦇F,x^P⦈ & ¬⦇F,y^P⦈ in v]
      then obtain x where [∃ y . ⦇F,x^P⦈ & ¬⦇F,y^P⦈ in v]
        by (rule ∃E)
      then obtain y where 2: [⦇F,x^P⦈ & ¬⦇F,y^P⦈ in v]
        by (rule ∃E)
      hence [∃ x . ⦇F, x^P⦈ in v]
        using &E(1) ∃I by fast
      hence [∀ x . ⦇F,x^P⦈ in v]
        using 1[THEN qml-2[axiom-instance, deduction], deduction] by fast
      hence [⦇F,y^P⦈ in v]
        using cqt-orig-1[deduction] by fast
      hence [⦇F,y^P⦈ & (¬⦇F,y^P⦈) in v]
        using 2 &I &E by fast
      hence [¬(∃ x y . ⦇F,x^P⦈ & ¬⦇F,y^P⦈) in v]
        using pl-1[axiom-instance, deduction, THEN modus-tollens-1]
              oth-class-taut-1-a by blast
    }
    thus [¬(∃ x y . ⦇F,x^P⦈ & ¬⦇F,y^P⦈) in v]
      using reductio-aa-2 if-p-then-p deduction-theorem by blast
  qed


lemma prop-in-thm:
  [Propositional F → Indiscriminate F in v]
  proof (rule CP)
    assume [Propositional F in v]
    hence [□(Propositional F) in v]
      using prop-prop2-3[deduction] by auto
    moreover {
      fix w
      assume [∃ p . (F = (λ y . p)) in w]
      then obtain q where q-prop: [F = (λ y . q) in w]
        by (rule ∃E)
      {
        assume [∃ x . ⦇F,x^P⦈ in w]
        then obtain a where [⦇F,a^P⦈ in w]
          by (rule ∃E)
        hence [⦇λ y . q, a^P⦈ in w]
          using q-prop l-identity[axiom-instance,deduction,deduction] by fast
        hence q: [q in w]
          apply (safe intro!: beta-C-meta-1[where φ=λy. q, equiv-lr])
           apply show-proper
          by simp
        {
```

```
          fix x
          have [(|λ y . q, x^P|) in w]
            apply (safe intro!: q beta-C-meta-1[equiv-rl])
            by show-proper
          hence [(|F,x^P|) in w]
            using q-prop[eq-sym] l-identity[axiom-instance, deduction, deduction]
            by fast
        }
        hence [∀ x . (|F,x^P|) in w]
          by (rule ∀I)
      }
      hence [(∃ x . (|F,x^P|)) → (∀ x . (|F, x^P|)) in w]
        by (rule CP)
    }
    ultimately show [Indiscriminate F in v]
      unfolding Propositional-def Indiscriminate-def
      using RM-1[deduction] deduction-theorem by blast
  qed

lemma prop-in-f-1:
  [Necessary F → Indiscriminate F in v]
  unfolding Necessary-defs Indiscriminate-def
  using pl-1[axiom-instance, THEN RM-1] by simp

lemma prop-in-f-2:
  [Impossible F → Indiscriminate F in v]
  proof –
    {
      fix w
      have [(¬(∃ x . (|F,x^P|))) → ((∃ x . (|F,x^P|)) → (∀ x . (|F,x^P|))) in w]
        using useful-tautologies-3 by auto
      hence [(∀ x . ¬(|F,x^P|)) → ((∃ x . (|F,x^P|)) → (∀ x . (|F,x^P|))) in w]
        apply – apply (PLM-subst-method ¬(∃ x. (|F,x^P|)) (∀ x. ¬(|F,x^P|)))
        using cqt-further-4 unfolding exists-def by fast+
    }
    thus ?thesis
      unfolding Impossible-defs Indiscriminate-def using RM-1 CP by blast
  qed

lemma prop-in-f-3-a:
  [¬(Indiscriminate (E!)) in v]
  proof (rule reductio-aa-2)
    show [□¬(∀x. (|E!,x^P|)) in v]
      using a-objects-exist-3 .
  next
    assume [Indiscriminate E! in v]
    thus [¬□¬(∀ x . (|E!,x^P|)) in v]
      unfolding Indiscriminate-def
      using o-objects-exist-1 KBasic2-5[deduction,deduction]
      unfolding diamond-def by blast
  qed

lemma prop-in-f-3-b:
  [¬(Indiscriminate (E!⁻)) in v]
  proof (rule reductio-aa-2)
    assume [Indiscriminate (E!⁻) in v]
    moreover have [□(∃ x . (|E!⁻, x^P|)) in v]
      apply (PLM-subst-method λ x . ¬(|E!, x^P|) λ x . (|E!⁻, x^P|))
```

```
      using thm-relation-negation-1-1[equiv-sym] apply simp
     unfolding exists-def
     apply (PLM-subst-method λ x . (|E!, x^P|) λ x . ¬¬(|E!, x^P|))
      using oth-class-taut-4-b apply simp
     using a-objects-exist-3 by auto
    ultimately have [□(∀ x. (|E!⁻,x^P|)) in v]
     unfolding Indiscriminate-def
     using qml-1[axiom-instance, deduction, deduction] by blast
    thus [□(∀ x. ¬(|E!,x^P|)) in v]
     apply −
     apply (PLM-subst-method λ x . (|E!⁻, x^P|) λ x . ¬(|E!, x^P|))
     using thm-relation-negation-1-1 by auto
  next
   show [¬□(∀ x . ¬(|E!, x^P|)) in v]
     using o-objects-exist-1
     unfolding diamond-def exists-def
     apply −
     apply (PLM-subst-method ¬¬(∀ x. ¬(|E!,x^P|)) ∀ x. ¬(|E!,x^P|))
     using oth-class-taut-4-b[equiv-sym] by auto
  qed

lemma prop-in-f-3-c:
  [¬(Indiscriminate (O!)) in v]
  proof (rule reductio-aa-2)
   show [¬(∀ x . (|O!,x^P|)) in v]
     using a-objects-exist-2[THEN qml-2[axiom-instance, deduction]]
        by blast
  next
   assume [Indiscriminate O! in v]
   thus [(∀ x . (|O!,x^P|)) in v]
     unfolding Indiscriminate-def
     using o-objects-exist-2 qml-1[axiom-instance, deduction, deduction]
        qml-2[axiom-instance, deduction] by blast
  qed

lemma prop-in-f-3-d:
  [¬(Indiscriminate (A!)) in v]
  proof (rule reductio-aa-2)
   show [¬(∀ x . (|A!,x^P|)) in v]
     using o-objects-exist-3[THEN qml-2[axiom-instance, deduction]]
        by blast
  next
   assume [Indiscriminate A! in v]
   thus [(∀ x . (|A!,x^P|)) in v]
     unfolding Indiscriminate-def
     using a-objects-exist-1 qml-1[axiom-instance, deduction, deduction]
        qml-2[axiom-instance, deduction] by blast
  qed

lemma prop-in-f-4-a:
  [¬(Propositional E!) in v]
  using prop-in-thm[deduction] prop-in-f-3-a modus-tollens-1 CP
  by meson

lemma prop-in-f-4-b:
  [¬(Propositional (E!⁻)) in v]
  using prop-in-thm[deduction] prop-in-f-3-b modus-tollens-1 CP
  by meson
```

lemma *prop-in-f-4-c*:
  $[\neg(Propositional\ (O!))\ in\ v]$
  using *prop-in-thm*[*deduction*] *prop-in-f-3-c modus-tollens-1 CP*
  by *meson*

lemma *prop-in-f-4-d*:
  $[\neg(Propositional\ (A!))\ in\ v]$
  using *prop-in-thm*[*deduction*] *prop-in-f-3-d modus-tollens-1 CP*
  by *meson*

lemma *prop-prop-nec-1*:
  $[\Diamond(\exists\ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)) \to (\exists\ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
  proof (*rule CP*)
    assume $[\Diamond(\exists\ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
    hence $[\exists\ p\ .\ \Diamond(F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
      using *BF*$\Diamond$[*deduction*] **by** *auto*
    then obtain p **where** $[\Diamond(F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
      by (*rule* $\exists\,E$)
    hence $[\Diamond\Box(\forall\, x.\ \{\!|x^P,F|\!\} \equiv \{\!|x^P,\boldsymbol{\lambda}x.\ p|\!\})\ in\ v]$
      unfolding *identity-defs* **.**
    hence $[\Box(\forall\, x.\ \{\!|x^P,F|\!\} \equiv \{\!|x^P,\boldsymbol{\lambda}x.\ p|\!\})\ in\ v]$
      using *5*$\Diamond$[*deduction*] **by** *auto*
    hence $[(F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
      unfolding *identity-defs* **.**
    thus $[\exists\ p\ .\ (F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
      by *PLM-solver*
  qed

lemma *prop-prop-nec-2*:
  $[(\forall\ p\ .\ F \neq (\boldsymbol{\lambda}\ x\ .\ p)) \to \Box(\forall\ p\ .\ F \neq (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
  apply (*PLM-subst-method*
        $\neg(\exists\ \ p\ .\ (F = (\boldsymbol{\lambda}\ x\ .\ p)))$
        $(\forall\ p\ .\ \neg(F = (\boldsymbol{\lambda}\ x\ .\ p))))$
   using *cqt-further-4* **apply** *blast*
  apply (*PLM-subst-method*
        $\neg\Diamond(\exists\, p.\ F = (\boldsymbol{\lambda}x.\ p))$
        $\Box\neg(\exists\, p.\ F = (\boldsymbol{\lambda}x.\ p)))$
   using *KBasic2-4*[*equiv-sym*] *prop-prop-nec-1*
        contraposition-1 **by** *auto*

lemma *prop-prop-nec-3*:
  $[(\exists\ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)) \to \Box(\exists\ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
  using *prop-prop-nec-1 derived-S5-rules-1-b* **by** *simp*

lemma *prop-prop-nec-4*:
  $[\Diamond(\forall\ p\ .\ F \neq (\boldsymbol{\lambda}\ x\ .\ p)) \to (\forall\ p\ .\ F \neq (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
  using *prop-prop-nec-2 derived-S5-rules-2-b* **by** *simp*

lemma *enc-prop-nec-1*:
  $[\Diamond(\forall\ F\ .\ \{\!|x^P,\ F|\!\} \to (\exists\ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)))$
   $\to (\forall\ F\ .\ \{\!|x^P,\ F|\!\} \to (\exists\ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)))\ in\ v]$
  proof (*rule CP*)
    assume $[\Diamond(\forall\, F.\ \{\!|x^P,F|\!\} \to (\exists\, p.\ F = (\boldsymbol{\lambda}x.\ p)))\ in\ v]$
    hence *1*: $[(\forall\, F.\ \Diamond(\{\!|x^P,F|\!\} \to (\exists\, p.\ F = (\boldsymbol{\lambda}x.\ p))))\ in\ v]$
      using *Buridan*$\Diamond$[*deduction*] **by** *auto*
    {
      fix Q

```
      assume [⦃x^P,Q⦄ in v]
      hence [□⦃x^P,Q⦄ in v]
        using encoding[axiom-instance, deduction] by auto
      moreover have [◇(⦃x^P,Q⦄ → (∃ p. Q = (λx. p))) in v]
        using cqt-1[axiom-instance, deduction] 1 by fast
      ultimately have [◇(∃ p. Q = (λx. p)) in v]
        using KBasic2-9[equiv-lr,deduction] by auto
      hence [(∃ p. Q = (λx. p)) in v]
        using prop-prop-nec-1[deduction] by auto
    }
    thus [(∀ F . ⦃x^P, F⦄ → (∃ p . F = (λ x . p))) in v]
      apply − by PLM-solver
  qed

  lemma enc-prop-nec-2:
    [(∀ F . ⦃x^P, F⦄ → (∃ p . F = (λ x . p))) → □(∀ F . ⦃x^P, F⦄
      → (∃ p . F = (λ x . p))) in v]
    using derived-S5-rules-1-b enc-prop-nec-1 by blast
end
end
```

A.10. Possible Worlds

```
locale PossibleWorlds = PLM
begin
```

A.10.1. Definitions

```
  definition Situation where
    Situation x ≡ ⦇A!,x⦈ & (∀ F. ⦃x,F⦄ → Propositional F)

  definition EncodeProposition (infixl Σ 70) where
    xΣp ≡ ⦇A!,x⦈ & ⦃x, λ x . p⦄
  definition TrueInSituation (infixl ⊨ 10) where
    x ⊨ p ≡ Situation x & xΣp
  definition PossibleWorld where
    PossibleWorld x ≡ Situation x & ◇(∀ p . xΣp ≡ p)
```

A.10.2. Auxiliary Lemmata

```
  lemma possit-sit-1:
    [Situation (x^P) ≡ □(Situation (x^P)) in v]
    proof (rule ≡I; rule CP)
      assume [Situation (x^P) in v]
      hence 1: [(⦇A!,x^P⦈ & (∀ F. ⦃x^P,F⦄ → Propositional F) in v]
        unfolding Situation-def by auto
      have [□⦇A!,x^P⦈ in v]
        using 1[conj1, THEN oa-facts-2[deduction]] .
      moreover have [□(∀ F. ⦃x^P,F⦄ → Propositional F) in v]
         using 1[conj2] unfolding Propositional-def
         by (rule enc-prop-nec-2[deduction])
      ultimately show [□Situation (x^P) in v]
        unfolding Situation-def
        apply cut-tac apply (rule KBasic-3[equiv-rl])
        by (rule intro-elim-1)
    next
```

```
    assume [□Situation (x^P) in v]
    thus [Situation (x^P) in v]
      using qml-2[axiom-instance, deduction] by auto
  qed

lemma possworld-nec:
  [PossibleWorld (x^P) ≡ □(PossibleWorld (x^P)) in v]
  apply (rule ≡I; rule CP)
   subgoal unfolding PossibleWorld-def
   apply (rule KBasic-3[equiv-rl])
   apply (rule intro-elim-1)
    using possit-sit-1[equiv-lr] &E(1) apply blast
   using qml-3[axiom-instance, deduction] &E(2) by blast
  using qml-2[axiom-instance,deduction] by auto

lemma TrueInWorldNecc:
  [((x^P) ⊨ p) ≡ □((x^P) ⊨ p) in v]
  proof (rule ≡I; rule CP)
    assume [x^P ⊨ p in v]
    hence [Situation (x^P) & (⦇A!,x^P⦈ & ⦃x^P,λx. p⦄) in v]
      unfolding TrueInSituation-def EncodeProposition-def .
    hence [(□Situation (x^P) & □⦇A!,x^P⦈) & □⦃x^P, λx. p⦄ in v]
      using &I &E possit-sit-1[equiv-lr] oa-facts-2[deduction]
            encoding[axiom-instance,deduction] by metis
    thus [□((x^P) ⊨ p) in v]
      unfolding TrueInSituation-def EncodeProposition-def
      using KBasic-3[equiv-rl] &I &E by metis
  next
    assume [□(x^P ⊨ p) in v]
    thus [x^P ⊨ p in v]
      using qml-2[axiom-instance,deduction] by auto
  qed


lemma PossWorldAux:
  [((⦇A!,x^P⦈ & (∀ F . (⦃x^P,F⦄ ≡ (∃ p . p & (F = (λ x . p))))))
    → (PossibleWorld (x^P)) in v]
  proof (rule CP)
    assume DefX: [(⦇A!,x^P⦈ & (∀ F . (⦃x^P,F⦄ ≡
          (∃ p . p & (F = (λ x . p))))) in v]

    have [Situation (x^P) in v]
    proof -
      have [⦇A!,x^P⦈ in v]
        using DefX[conj1] .
      moreover have [(∀ F. ⦃x^P,F⦄ → Propositional F) in v]
        proof (rule ∀I; rule CP)
          fix F
          assume [⦃x^P,F⦄ in v]
          moreover have [⦃x^P,F⦄ ≡ (∃ p . p & (F = (λ x . p))) in v]
            using DefX[conj2] cqt-1[axiom-instance, deduction] by auto
          ultimately have [(∃ p . p & (F = (λ x . p))) in v]
            using ≡E(1) by blast
          then obtain p where [p & (F = (λ x . p)) in v]
            by (rule ∃E)
          hence [(F = (λ x . p)) in v]
            by (rule &E(2))
          hence [(∃ p . (F = (λ x . p))) in v]
```

```
          by PLM-solver
        thus [Propositional F in v]
          unfolding Propositional-def .
      qed
    ultimately show [Situation (x^P) in v]
      unfolding Situation-def by (rule &I)
  qed
  moreover have [◇(∀ p. x^P Σ p ≡ p) in v]
    unfolding EncodeProposition-def
    proof (rule TBasic[deduction]; rule ∀I)
      fix q
      have EncodeLambda:
        [⦃x^P, λx. q⦄ ≡ (∃ p . p & ((λx. q) = (λ x . p))) in v]
        using DefX[conj2] by (rule cqt-1[axiom-instance, deduction])
      moreover {
          assume [q in v]
          moreover have [(λx. q) = (λ x . q) in v]
           using id-eq-prop-prop-1 by auto
          ultimately have [q & ((λx. q) = (λ x . q)) in v]
            by (rule &I)
          hence [∃ p . p & ((λx. q) = (λ x . p)) in v]
            by PLM-solver
          moreover have [⦇A!,x^P⦈ in v]
            using DefX[conj1] .
          ultimately have [⦇A!,x^P⦈ & ⦃x^P, λx. q⦄ in v]
            using EncodeLambda[equiv-rl] &I by auto
      }
      moreover {
        assume [⦇A!,x^P⦈ & ⦃x^P, λx. q⦄ in v]
        hence [⦃x^P, λx. q⦄ in v]
          using &E(2) by auto
        hence [∃ p . p & ((λx. q) = (λ x . p)) in v]
          using EncodeLambda[equiv-lr] by auto
        then obtain p where p-and-lambda-q-is-lambda-p:
          [p & ((λx. q) = (λ x . p)) in v]
          by (rule ∃E)
        have [⦇(λ x . p), x^P⦈ ≡ p in v]
          apply (rule beta-C-meta-1)
          by show-proper
        hence [⦇(λ x . p), x^P⦈ in v]
          using p-and-lambda-q-is-lambda-p[conj1] ≡E(2) by auto
        hence [⦇(λ x . q), x^P⦈ in v]
          using p-and-lambda-q-is-lambda-p[conj2, THEN id-eq-prop-prop-2[deduction]]
            l-identity[axiom-instance, deduction, deduction] by fast
        moreover have [⦇(λ x . q), x^P⦈ ≡ q in v]
          apply (rule beta-C-meta-1) by show-proper
        ultimately have [q in v]
          using ≡E(1) by blast
      }
      ultimately show [⦇A!,x^P⦈ & ⦃x^P,λx. q⦄ ≡ q in v]
        using &I ≡I CP by auto
    qed

  ultimately show [PossibleWorld (x^P) in v]
    unfolding PossibleWorld-def by (rule &I)
qed
```

A.10.3. For every syntactic Possible World there is a semantic Possible World

```
theorem SemanticPossibleWorldForSyntacticPossibleWorlds:
  ∀ x . [PossibleWorld (x^P) in w] ⟶
   (∃ v . ∀ p . [(x^P ⊨ p) in w] ⟷ [p in v])
 proof
   fix x
   {
     assume PossWorldX: [PossibleWorld (x^P) in w]
     hence SituationX: [Situation (x^P) in w]
       unfolding PossibleWorld-def apply cut-tac by PLM-solver
     have PossWorldExpanded:
       [⦇A!,x^P⦈ & (∀ F. ⦃x^P,F⦄ → (∃ p. F = (λx. p)))
         & ◇(∀ p. ⦇A!,x^P⦈ & ⦃x^P,λx. p⦄ ≡ p) in w]
        using PossWorldX
        unfolding PossibleWorld-def Situation-def
                  Propositional-def EncodeProposition-def .
     have AbstractX: [⦇A!,x^P⦈ in w]
       using PossWorldExpanded[conj1,conj1] .

     have [◇(∀ p. ⦃x^P,λx. p⦄ ≡ p) in w]
       apply (PLM-subst-method
              λp. ⦇A!,x^P⦈ & ⦃x^P,λx. p⦄
              λ p . ⦃x^P,λx. p⦄)
        subgoal using PossWorldExpanded[conj1,conj1,THEN oa-facts-2[deduction]]
                using Semantics.T6 apply cut-tac by PLM-solver
       using PossWorldExpanded[conj2] .

     hence ∃ v. ∀ p. ([⦃x^P,λx. p⦄ in v])
                 = [p in v]
      unfolding diamond-def equiv-def conj-def
      apply (simp add: Semantics.T4 Semantics.T6 Semantics.T5
                       Semantics.T8)
      by auto

     then obtain v where PropsTrueInSemWorld:
       ∀ p. ([⦃x^P,λx. p⦄ in v]) = [p in v]
       by auto
     {
       fix p
       {
         assume [((x^P) ⊨ p) in w]
         hence [((x^P) ⊨ p) in v]
           using TrueInWorldNecc[equiv-lr] Semantics.T6 by simp
         hence [Situation (x^P) & (⦇A!,x^P⦈ & ⦃x^P,λx. p⦄) in v]
           unfolding TrueInSituation-def EncodeProposition-def .
         hence [⦃x^P,λx. p⦄ in v]
           using &E(2) by blast
         hence [p in v]
           using PropsTrueInSemWorld by blast
       }
       moreover {
         assume [p in v]
         hence [⦃x^P,λx. p⦄ in v]
           using PropsTrueInSemWorld by blast
         hence [(x^P) ⊨ p in v]
           apply cut-tac unfolding TrueInSituation-def EncodeProposition-def
           apply (rule &I) using SituationX[THEN possit-sit-1[equiv-lr]]
```

```
          subgoal using Semantics.T6 by auto
          apply (rule &I)
          subgoal using AbstractX[THEN oa-facts-2[deduction]]
            using Semantics.T6 by auto
          by assumption
        hence [□((x^P) ⊨ p) in v]
          using TrueInWorldNecc[equiv-lr] by simp
        hence [(x^P) ⊨ p in w]
          using Semantics.T6 by simp
      }
      ultimately have [p in v] ⟷ [(x^P) ⊨ p in w]
        by auto
    }
    hence (∃ v . ∀ p . [p in v] ⟷ [(x^P) ⊨ p in w])
      by blast
  }
  thus [PossibleWorld (x^P) in w] ⟶
        (∃ v. ∀ p . [(x^P) ⊨ p in w] ⟷ [p in v])
    by blast
qed
```

A.10.4. For every semantic Possible World there is a syntactic Possible World

```
theorem SyntacticPossibleWorldForSemanticPossibleWorlds:
 ∀ v . ∃ x . [PossibleWorld (x^P) in w] ∧
  (∀ p . [p in v] ⟷ [((x^P) ⊨ p) in w])
  proof
    fix v
    have [∃ x. ⦇A!,x^P⦈ & (∀ F . (⦃x^P,F⦄ ≡
          (∃ p . p & (F = (λ x . p))))) in v]
      using A-objects[axiom-instance] by fast
    then obtain x where DefX:
      [⦇A!,x^P⦈ & (∀ F . (⦃x^P,F⦄ ≡ (∃ p . p & (F = (λ x . p))))) in v]
      by (rule ∃E)
    hence PossWorldX: [PossibleWorld (x^P) in v]
      using PossWorldAux[deduction] by blast
    hence [PossibleWorld (x^P) in w]
      using possworld-nec[equiv-lr] Semantics.T6 by auto
    moreover have (∀ p . [p in v] ⟷ [(x^P) ⊨ p in w])
    proof
      fix q
      {
        assume [q in v]
        moreover have [(λ x . q) = (λ x . q) in v]
          using id-eq-prop-prop-1 by auto
        ultimately have [q & (λ x . q) = (λ x . q) in v]
          using &I by auto
        hence [(∃ p . p & ((λ x . q) = (λ x . p))) in v]
          by PLM-solver
        hence 4: [⦃x^P, (λ x . q)⦄ in v]
          using cqt-1[axiom-instance,deduction, OF DefX[conj2], equiv-rl]
          by blast
        have [(x^P ⊨ q) in v]
          unfolding TrueInSituation-def apply (rule &I)
           using PossWorldX unfolding PossibleWorld-def
           using &E(1) apply blast
          unfolding EncodeProposition-def apply (rule &I)
           using DefX[conj1] apply simp
```

```
          using 4 .
        hence [(x^P ⊨ q) in w]
          using TrueInWorldNecc[equiv-lr] Semantics.T6 by auto
      }
      moreover {
        assume [(x^P ⊨ q) in w]
        hence [(x^P ⊨ q) in v]
          using TrueInWorldNecc[equiv-lr] Semantics.T6
          by auto
        hence [⦃x^P, (λ x . q)⦄ in v]
          unfolding TrueInSituation-def EncodeProposition-def
          using &E(2) by blast
        hence [(∃ p . p & ((λ x . q) = (λ x . p))) in v]
          using cqt-1[axiom-instance,deduction, OF DefX[conj2], equiv-lr]
          by blast
        then obtain p where 4:
          [(p & ((λ x . q) = (λ x . p))) in v]
          by (rule ∃ E)
        have [⦇(λ x . p),x^P⦈ ≡ p in v]
          apply (rule beta-C-meta-1)
          by show-proper
        hence [⦇(λ x . q),x^P⦈ ≡ p in v]
            using l-identity[where β=(λ x . q) and α=(λ x . p),
                            axiom-instance, deduction, deduction]
            using 4[conj2,THEN id-eq-prop-prop-2[deduction]] by meson
        hence [⦇(λ x . q),x^P⦈ in v] using 4[conj1] ≡E(2) by blast
        moreover have [⦇(λ x . q),x^P⦈ ≡ q in v]
          apply (rule beta-C-meta-1)
          by show-proper
        ultimately have [q in v]
          using ≡E(1) by blast
      }
      ultimately show [q in v] ⟷ [(x^P) ⊨ q in w]
        by blast
    qed
    ultimately show ∃ x . [PossibleWorld (x^P) in w]
                      ∧ (∀ p . [p in v] ⟷ [(x^P) ⊨ p in w])
      by auto
  qed
end
```

A.11. Artificial Theorems

Remark. *Some examples of theorems that can be derived from the meta-logic, but which are not derivable from the deductive system PLM itself.*

```
locale ArtificialTheorems
begin

  lemma lambda-enc-1:
    [⦇λx . ⦃x^P, F⦄ ≡ ⦃x^P, F⦄, y^P⦈ in v]
    by (auto simp: meta-defs meta-aux conn-defs forall-Π₁-def)

  lemma lambda-enc-2:
    [⦇λ x . ⦃y^P, G⦄, x^P⦈ ≡ ⦃y^P, G⦄ in v]
    by (auto simp: meta-defs meta-aux conn-defs forall-Π₁-def)
```

Remark. *The following is* not *a theorem and nitpick can find a countermodel. This is expected and important because, if this were a theorem, the theory would become inconsistent.*

lemma *lambda-enc-3*:
$[((\!|\boldsymbol{\lambda}\ x\ .\ \{\!|x^P,\ F|\!\},\ x^P|\!)) \rightarrow \{\!|x^P,\ F|\!\})\ in\ v]$
apply (*simp add*: *meta-defs meta-aux conn-defs forall*-Π_1-*def*)
nitpick[*user-axioms*, *expect=genuine*]
oops — countermodel by nitpick

Remark. *Instead the following two statements hold.*

lemma *lambda-enc-4*:
$[(\!|(\boldsymbol{\lambda}\ x\ .\ \{\!|x^P,\ F|\!\}),\ x^P|\!)\ in\ v] = (\exists\ y\ .\ \nu\upsilon\ y = \nu\upsilon\ x \wedge [\{\!|y^P,\ F|\!\}\ in\ v])$
by (*simp add*: *meta-defs meta-aux*)

lemma *lambda-ex*:
$[(\!|(\boldsymbol{\lambda}\ x\ .\ \varphi\ (x^P)),\ x^P|\!)\ in\ v] = (\exists\ y\ .\ \nu\upsilon\ y = \nu\upsilon\ x \wedge [\varphi\ (y^P)\ in\ v])$
by (*simp add*: *meta-defs meta-aux*)

Remark. *These statements can be translated to statements in the embedded logic.*

lemma *lambda-ex-emb*:
$[(\!|(\boldsymbol{\lambda}\ x\ .\ \varphi\ (x^P)),\ x^P|\!) \equiv (\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ (y^P))\ in\ v]$
proof(*rule MetaSolver.EquivI*)
interpret *MetaSolver* **.**
{
assume $[(\!|(\boldsymbol{\lambda}\ x\ .\ \varphi\ (x^P)),\ x^P|\!)\ in\ v]$
then obtain y **where** $\nu\upsilon\ y = \nu\upsilon\ x \wedge [\varphi\ (y^P)\ in\ v]$
using *lambda-ex* **by** *blast*
moreover hence $[(\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ in\ v]$
apply − **apply** *meta-solver*
by (*simp add*: *Semantics.*d_κ*-proper Semantics.ex1-def*)
ultimately have $[\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ (y^P)\ in\ v]$
using *ExIRule ConjI* **by** *fast*
}
moreover {
assume $[\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ (y^P)\ in\ v]$
then obtain y **where** *y-def*: $[(\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ (y^P)\ in\ v]$
by (*rule ExERule*)
hence $\bigwedge\ F\ .\ [(\!|F,x^P|\!)\ in\ v] = [(\!|F,y^P|\!)\ in\ v]$
apply − **apply** (*drule ConjE*) **apply** (*drule conjunct1*)
apply (*drule AllE*) **apply** (*drule EquivE*) **by** *simp*
hence $[(\!|make\Pi_1\ (\lambda\ u\ s\ w\ .\ \nu\upsilon\ y = u),x^P|\!)\ in\ v]$
$= [(\!|make\Pi_1\ (\lambda\ u\ s\ w\ .\ \nu\upsilon\ y = u),y^P|\!)\ in\ v]$ **by** *auto*
hence $\nu\upsilon\ y = \nu\upsilon\ x$ **by** (*simp add*: *meta-defs meta-aux*)
moreover have $[\varphi\ (y^P)\ in\ v]$ **using** *y-def ConjE* **by** *blast*
ultimately have $[(\!|(\boldsymbol{\lambda}\ x\ .\ \varphi\ (x^P)),\ x^P|\!)\ in\ v]$
using *lambda-ex* **by** *blast*
}
ultimately show $[(\!|\boldsymbol{\lambda}x.\ \varphi\ (x^P),x^P|\!)\ in\ v]$
$= [\exists\,y.\ (\forall\,F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ (y^P)\ in\ v]$
by *auto*
qed

lemma *lambda-enc-emb*:
$[(\!|(\boldsymbol{\lambda}\ x\ .\ \{\!|x^P,\ F|\!\}),\ x^P|\!) \equiv (\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \{\!|y^P,\ F|\!\})\ in\ v]$
using *lambda-ex-emb* **by** *fast*

Remark. *In the case of proper maps, the generalized β-conversion reduces to classical β-conversion.*

lemma *proper-beta*:
  assumes *IsProperInX* φ
  shows $[(\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ (y^P)) \equiv \varphi\ (x^P)\ in\ v]$
proof (*rule MetaSolver.EquivI*; *rule*)
  interpret *MetaSolver* **.**
  assume $[\exists\, y.\ (\forall\, F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ (y^P)\ in\ v]$
  then obtain y **where** *y-def*: $[(\forall\, F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ (y^P)\ in\ v]$ **by** (*rule ExERule*)
  hence $[(\!|make\Pi_1\ (\lambda\ u\ s\ w\ .\ \nu\upsilon\ y = u),\ x^P|\!)\ in\ v] = [(\!|make\Pi_1\ (\lambda\ u\ s\ w\ .\ \nu\upsilon\ y = u),\ y^P|\!)\ in\ v]$
    using *EquivS AllE ConjE* **by** *blast*
  hence $\nu\upsilon\ y = \nu\upsilon\ x$ **by** (*simp add*: *meta-defs meta-aux*)
  thus $[\varphi\ (x^P)\ in\ v]$
    using *y-def*[*THEN ConjE*[*THEN conjunct2*]]
        assms IsProperInX.rep-eq valid-in.rep-eq
    by *blast*
next
  interpret *MetaSolver* **.**
  assume $[\varphi\ (x^P)\ in\ v]$
  moreover have $[\forall\, F.\ (\!|F,x^P|\!) \equiv (\!|F,x^P|\!)\ in\ v]$ **apply** *meta-solver* **by** *blast*
  ultimately show $[\exists\, y.\ (\forall\, F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ (y^P)\ in\ v]$
    by (*meson ConjI ExI*)
qed

Remark. *The following theorem is a consequence of the constructed Aczel-model, but not part of PLM. Separate research on possible modifications of the embedding suggest that this artificial theorem can be avoided by introducing a dependency on states for the mapping from abstract objects to special urelements.*

lemma *lambda-rel-extensional*:
  assumes $[\forall\, F\ .\ (\!|F,a^P|\!) \equiv (\!|F,b^P|\!)\ in\ v]$
  shows $(\boldsymbol{\lambda} x.\ (\!|R,x^P,a^P|\!)) = (\boldsymbol{\lambda} x\ .\ (\!|R,x^P,\ b^P|\!))$
proof −
  interpret *MetaSolver* **.**
  obtain F **where** *F-def*: $F = make\Pi_1\ (\lambda\ u\ s\ w\ .\ u = \nu\upsilon\ a)$ **by** *auto*
  have $[(\!|F,\ a^P|\!) \equiv (\!|F,\ b^P|\!)\ in\ v]$ **using** *assms* **by** (*rule AllE*)
  moreover have $[(\!|F,\ a^P|\!)\ in\ v]$
    unfolding *F-def* **by** (*simp add*: *meta-defs meta-aux*)
  ultimately have $[(\!|F,\ b^P|\!)\ in\ v]$ **using** *EquivE* **by** *auto*
  hence $\nu\upsilon\ a = \nu\upsilon\ b$ **using** *F-def* **by** (*simp add*: *meta-defs meta-aux*)
  thus *?thesis* **by** (*simp add*: *meta-defs meta-aux*)
qed

end

A.12. Sanity Tests

locale *SanityTests*
begin
  interpretation *MetaSolver***.**
  interpretation *Semantics***.**

A.12.1. Consistency

lemma *True*
  nitpick[*expect=genuine*, *user-axioms*, *satisfy*]
  by *auto*

A.12.2. Intensionality

lemma $[(\boldsymbol{\lambda} y.\ (q \vee \neg q)) = (\boldsymbol{\lambda} y.\ (p \vee \neg p))\ in\ v]$
  unfolding *identity*-Π_1-*def conn-defs*
  apply (*rule* $Eq_1 I$) **apply** (*simp add*: *meta-defs*)
  nitpick[*expect = genuine*, *user-axioms=true*, *card i = 2*,
    card j = 2, *card* ω *= 1*, *card* σ *= 1*,
    sat-solver = MiniSat-JNI, *verbose*, *show-all*]
  oops — Countermodel by Nitpick
lemma $[(\boldsymbol{\lambda} y.\ (p \vee q)) = (\boldsymbol{\lambda} y.\ (q \vee p))\ in\ v]$
  unfolding *identity*-Π_1-*def*
  apply (*rule* $Eq_1 I$) **apply** (*simp add*: *meta-defs*)
  nitpick[*expect = genuine*, *user-axioms=true*,
    sat-solver = MiniSat-JNI, *card i = 2*,
    card j = 2, *card* σ *= 1*, *card* ω *= 1*,
    card υ *= 2*, *verbose*, *show-all*]
  oops — Countermodel by Nitpick

A.12.3. Concreteness coindices with Object Domains

lemma *OrdCheck*:
  $[(\!|\boldsymbol{\lambda}\ x\ .\ \neg\Box(\neg(\!|E!,\ x^P|\!)),\ x|\!)\ in\ v] \longleftrightarrow$
   $(proper\ x) \wedge (case\ (rep\ x)\ of\ \omega\nu\ y \Rightarrow True\ |\ \text{-} \Rightarrow False)$
  using *OrdinaryObjectsPossiblyConcreteAxiom*
  apply (*simp add*: *meta-defs meta-aux split*: ν*.split* υ*.split*)
  using $\nu\upsilon$-$\omega\nu$-*is*-$\omega\upsilon$ **by** *fastforce*
lemma *AbsCheck*:
  $[(\!|\boldsymbol{\lambda}\ x\ .\ \Box(\neg(\!|E!,\ x^P|\!)),\ x|\!)\ in\ v] \longleftrightarrow$
   $(proper\ x) \wedge (case\ (rep\ x)\ of\ \alpha\nu\ y \Rightarrow True\ |\ \text{-} \Rightarrow False)$
  using *OrdinaryObjectsPossiblyConcreteAxiom*
  apply (*simp add*: *meta-defs meta-aux split*: ν*.split* υ*.split*)
  using *no*-$\alpha\omega$ **by** *blast*

A.12.4. Justification for Meta-Logical Axioms

Remark. *OrdinaryObjectsPossiblyConcreteAxiom is equivalent to "all ordinary objects are possibly concrete".*

lemma *OrdAxiomCheck*:
  OrdinaryObjectsPossiblyConcrete $\longleftrightarrow$
   $(\forall\ x.\ ([(\!|\boldsymbol{\lambda}\ x\ .\ \neg\Box(\neg(\!|E!,\ x^P|\!)),\ x^P|\!)\ in\ v]$
    $\longleftrightarrow (case\ x\ of\ \omega\nu\ y \Rightarrow True\ |\ \text{-} \Rightarrow False)))$
  unfolding *Concrete-def*
  apply (*simp add*: *meta-defs meta-aux split*: ν*.split* υ*.split*)
  using $\nu\upsilon$-$\omega\nu$-*is*-$\omega\upsilon$ **by** *fastforce*

Remark. *OrdinaryObjectsPossiblyConcreteAxiom is equivalent to "all abstract objects are necessarily not concrete".*

```
lemma AbsAxiomCheck:
  OrdinaryObjectsPossiblyConcrete ⟷
    (∀ x. ([(|λ x . □(¬(|E!, x^P|)), x^P|) in v]
     ⟷ (case x of αν y ⇒ True | - ⇒ False)))
  apply (simp add: meta-defs meta-aux split: ν.split υ.split)
  using νυ-ων-is-ωυ no-αω by fastforce
```

Remark. *PossiblyContingentObjectExistsAxiom is equivalent to the corresponding statement in the embedded logic.*

```
lemma PossiblyContingentObjectExistsCheck:
  PossiblyContingentObjectExists ⟷ [¬(□(∀ x. (|E!,x^P|) → □(|E!,x^P|))) in v]
   apply (simp add: meta-defs forall-ν-def meta-aux split: ν.split υ.split)
   by (metis ν.simps(5) νυ-def υ.simps(1) no-σω υ.exhaust)
```

Remark. *PossiblyNoContingentObjectExistsAxiom is equivalent to the corresponding statement in the embedded logic.*

```
lemma PossiblyNoContingentObjectExistsCheck:
  PossiblyNoContingentObjectExists ⟷ [¬(□(¬(∀ x. (|E!,x^P|) → □(|E!,x^P|)))) in v]
  apply (simp add: meta-defs forall-ν-def meta-aux split: ν.split υ.split)
  using νυ-ων-is-ωυ by blast
```

A.12.5. Relations in the Meta-Logic

Remark. *Material equality in the embedded logic corresponds to equality in the actual state in the meta-logic.*

```
lemma mat-eq-is-eq-dj:
  [∀ x . □((|F,x^P|) ≡ (|G,x^P|)) in v] ⟷
   ((λ x . (evalΠ1 F) x dj) = (λ x . (evalΠ1 G) x dj))
proof
  assume 1: [∀ x. □((|F,x^P|) ≡ (|G,x^P|)) in v]
  {
    fix v
    fix y
    obtain x where y-def: y = νυ x
      by (meson νυ-surj surj-def)
    have (∃ r o1. Some r = d1 F ∧ Some o1 = dκ (x^P) ∧ o1 ∈ ex1 r v) =
          (∃ r o1. Some r = d1 G ∧ Some o1 = dκ (x^P) ∧ o1 ∈ ex1 r v)
        using 1 apply − by meta-solver
    moreover obtain r where r-def: Some r = d1 F
      unfolding d1-def by auto
    moreover obtain s where s-def: Some s = d1 G
      unfolding d1-def by auto
    moreover have Some x = dκ (x^P)
      using dκ-proper by simp
    ultimately have (x ∈ ex1 r v) = (x ∈ ex1 s v)
      by (metis option.inject)
    hence (evalΠ1 F) y dj v = (evalΠ1 G) y dj v
      using r-def s-def y-def by (simp add: d1.rep-eq ex1-def)
  }
  thus (λx. evalΠ1 F x dj) = (λx. evalΠ1 G x dj)
    by auto
next
  assume 1: (λx. evalΠ1 F x dj) = (λx. evalΠ1 G x dj)
```

{
fix $y\ v$
obtain x **where** *x-def*: $x = \nu\upsilon\ y$
by *simp*
hence $eval\Pi_1\ F\ x\ dj = eval\Pi_1\ G\ x\ dj$
using *1* **by** *metis*
moreover obtain r **where** *r-def*: $Some\ r = d_1\ F$
unfolding d_1*-def* **by** *auto*
moreover obtain s **where** *s-def*: $Some\ s = d_1\ G$
unfolding d_1*-def* **by** *auto*
ultimately have $(y \in ex1\ r\ v) = (y \in ex1\ s\ v)$
by (*simp add*: d_1*.rep-eq ex1-def* $\nu\upsilon$*-surj x-def*)
hence $[(\!|F,y^P|\!) \equiv (\!|G,y^P|\!)\ in\ v]$
apply − **apply** *meta-solver*
using *r-def s-def* **by** (*metis Semantics*.d_κ*-proper option.inject*)
}
thus $[\forall\, x.\ \Box((\!|F,x^P|\!) \equiv (\!|G,x^P|\!))\ in\ v]$
using *T6 T8* **by** *fast*
qed

Remark. *Materially equivalent relations are equal in the embedded logic if and only if they also coincide in all other states.*

lemma *mat-eq-is-eq-if-eq-forall-j*:
assumes $[\forall\ x\ .\ \Box((\!|F,x^P|\!) \equiv (\!|G,x^P|\!))\ in\ v]$
shows $[F = G\ in\ v] \longleftrightarrow$
$(\forall\ s\ .\ s \neq dj \longrightarrow (\forall\ x\ .\ (eval\Pi_1\ F)\ x\ s = (eval\Pi_1\ G)\ x\ s))$
proof
interpret *MetaSolver* **.**
assume $[F = G\ in\ v]$
hence $F = G$
apply − **unfolding** *identity*-Π_1*-def* **by** *meta-solver*
thus $\forall\, s.\ s \neq dj \longrightarrow (\forall\, x.\ eval\Pi_1\ F\ x\ s = eval\Pi_1\ G\ x\ s)$
by *auto*
next
interpret *MetaSolver* **.**
assume $\forall\, s.\ s \neq dj \longrightarrow (\forall\, x.\ eval\Pi_1\ F\ x\ s = eval\Pi_1\ G\ x\ s)$
moreover have $((\lambda\ x\ .\ (eval\Pi_1\ F)\ x\ dj) = (\lambda\ x\ .\ (eval\Pi_1\ G)\ x\ dj))$
using *assms mat-eq-is-eq-dj* **by** *auto*
ultimately have $\forall\, s\ x.\ eval\Pi_1\ F\ x\ s = eval\Pi_1\ G\ x\ s$
by *metis*
hence $eval\Pi_1\ F = eval\Pi_1\ G$
by *blast*
hence $F = G$
by (*metis eval*Π_1*-inverse*)
thus $[F = G\ in\ v]$
unfolding *identity*-Π_1*-def* **using** Eq_1I **by** *auto*
qed

Remark. *Under the assumption that all properties behave in all states like in the actual state the defined equality degenerates to material equality.*

lemma assumes $\forall\ F\ x\ s\ .\ (eval\Pi_1\ F)\ x\ s = (eval\Pi_1\ F)\ x\ dj$
shows $[\forall\ x\ .\ \Box((\!|F,x^P|\!) \equiv (\!|G,x^P|\!))\ in\ v] \longleftrightarrow [F = G\ in\ v]$
by (*metis* (*no-types*) *MetaSolver*.Eq_1S *assms identity*-Π_1*-def*
mat-eq-is-eq-dj mat-eq-is-eq-if-eq-forall-j)

A.12.6. Lambda Expressions in the Meta-Logic

lemma *lambda-interpret-1*:
assumes $[a = b \ in \ v]$
shows $(\boldsymbol{\lambda} x.\ (|R,x^P,a|)) = (\boldsymbol{\lambda} x\ .\ (|R,x^P,\ b|))$
proof −
  have $a = b$
    using *MetaSolver.EqκS Semantics.*d_κ*-inject assms*
        identity-κ-def **by** *auto*
  thus *?thesis* **by** *simp*
qed

lemma *lambda-interpret-2*:
assumes $[a = (\boldsymbol{\iota} y.\ (|G,y^P|))\ in\ v]$
shows $(\boldsymbol{\lambda} x.\ (|R,x^P,a|)) = (\boldsymbol{\lambda} x\ .\ (|R,x^P,\ \boldsymbol{\iota} y.\ (|G,y^P|)|))$
proof −
  have $a = (\boldsymbol{\iota} y.\ (|G,y^P|))$
    using *MetaSolver.EqκS Semantics.*d_κ*-inject assms*
        identity-κ-def **by** *auto*
  thus *?thesis* **by** *simp*
qed

end

theory *TAO-99-Paradox*
imports *TAO-9-PLM TAO-98-ArtificialTheorems*
begin

A.13. Paradox

Under the additional assumption that expressions of the form $\lambda x.\ (|G,\boldsymbol{\iota} y.\ \varphi\ y\ x|)$ for arbitrary φ are *proper maps*, for which β-conversion holds, the theory becomes inconsistent.

A.13.1. Auxiliary Lemmata

lemma *exe-impl-exists*:
  $[(|(\boldsymbol{\lambda} x\ .\ \forall\ p\ .\ p \rightarrow p),\ \boldsymbol{\iota} y\ .\ \varphi\ y\ x|) \equiv (\exists\,!y\ .\ \boldsymbol{\mathcal{A}}\varphi\ y\ x)\ in\ v]$
  proof (*rule* $\equiv I$; *rule CP*)
    fix $\varphi :: \nu{\Rightarrow}\nu{\Rightarrow}o$ **and** $x :: \nu$ **and** $v :: i$
    assume $[(|(\boldsymbol{\lambda} x\ .\ \forall\ p\ .\ p \rightarrow p),\boldsymbol{\iota} y\ .\ \varphi\ y\ x|)\ in\ v]$
    hence $[\exists\, y.\ \boldsymbol{\mathcal{A}}\varphi\ y\ x\ \&\ (\forall\, z.\ \boldsymbol{\mathcal{A}}\varphi\ z\ x \rightarrow z = y)$
        $\&\ (|(\boldsymbol{\lambda} x\ .\ \forall\ p\ .\ p \rightarrow p),\ y^P|)\ in\ v]$
      using *nec-russell-axiom*[*equiv-lr*] *SimpleExOrEnc.intros* **by** *auto*
    then obtain y **where**
      $[\boldsymbol{\mathcal{A}}\varphi\ y\ x\ \&\ (\forall\, z.\ \boldsymbol{\mathcal{A}}\varphi\ z\ x \rightarrow z = y)$
       $\&\ (|(\boldsymbol{\lambda} x\ .\ \forall\ p\ .\ p \rightarrow p),\ y^P|)\ in\ v]$
      by (*rule Instantiate*)
    hence $[\boldsymbol{\mathcal{A}}\varphi\ y\ x\ \&\ (\forall\, z.\ \boldsymbol{\mathcal{A}}\varphi\ z\ x \rightarrow z = y)\ in\ v]$
      using $\&E$ **by** *blast*
    hence $[\exists\, y\ .\ \boldsymbol{\mathcal{A}}\varphi\ y\ x\ \&\ (\forall\, z.\ \boldsymbol{\mathcal{A}}\varphi\ z\ x \rightarrow z = y)\ in\ v]$
      by (*rule existential*)
    thus $[\exists\,!y.\ \boldsymbol{\mathcal{A}}\varphi\ y\ x\ in\ v]$
      unfolding *exists-unique-def* **by** *simp*
  next
    fix $\varphi :: \nu{\Rightarrow}\nu{\Rightarrow}o$ **and** $x :: \nu$ **and** $v :: i$
    assume $[\exists\,!y.\ \boldsymbol{\mathcal{A}}\varphi\ y\ x\ in\ v]$
    hence $[\exists\, y.\ \boldsymbol{\mathcal{A}}\varphi\ y\ x\ \&\ (\forall\, z.\ \boldsymbol{\mathcal{A}}\varphi\ z\ x \rightarrow z = y)\ in\ v]$

unfolding *exists-unique-def* **by** *simp*
then obtain y **where**
$[\boldsymbol{\mathcal{A}}\varphi\ y\ x\ \&\ (\forall z.\ \boldsymbol{\mathcal{A}}\varphi\ z\ x \to z = y)\ \mathit{in}\ v]$
by (*rule Instantiate*)
moreover have $[(\!|(\boldsymbol{\lambda}x\ .\ \forall\ \ p\ .\ p \to p), y^P|\!)\ \mathit{in}\ v]$
apply (*rule beta-C-meta-1*[*equiv-rl*])
apply *show-proper*
by *PLM-solver*
ultimately have $[\boldsymbol{\mathcal{A}}\varphi\ y\ x\ \&\ (\forall z.\ \boldsymbol{\mathcal{A}}\varphi\ z\ x \to z = y)$
$\&\ (\!|(\boldsymbol{\lambda}x\ .\ \forall\ \ p\ .\ p \to p), y^P|\!)\ \mathit{in}\ v]$
using &*I* **by** *blast*
hence $[\exists\ \ y\ .\ \boldsymbol{\mathcal{A}}\varphi\ y\ x\ \&\ (\forall z.\ \boldsymbol{\mathcal{A}}\varphi\ z\ x \to z = y)$
$\&\ (\!|(\boldsymbol{\lambda}x\ .\ \forall\ \ p\ .\ p \to p), y^P|\!)\ \mathit{in}\ v]$
by (*rule existential*)
thus $[(\!|(\boldsymbol{\lambda}x\ .\ \forall\ \ p\ .\ p \to p),\ \boldsymbol{\iota}y.\ \varphi\ y\ x|\!)\ \mathit{in}\ v]$
using *nec-russell-axiom*[*equiv-rl*]
SimpleExOrEnc.intros **by** *auto*
qed

lemma *exists-unique-actual-equiv*:
$[(\exists !y\ .\ \boldsymbol{\mathcal{A}}(y = x\ \&\ \psi\ (x^P))) \equiv \boldsymbol{\mathcal{A}}\psi\ (x^P)\ \mathit{in}\ v]$
proof (*rule* $\equiv$*I*; *rule CP*)
fix $x\ v$
let $?\varphi = \lambda\ y\ x.\ y = x\ \&\ \psi\ (x^P)$
assume $[\exists !y.\ \boldsymbol{\mathcal{A}}?\varphi\ y\ x\ \mathit{in}\ v]$
hence $[\exists \alpha.\ \boldsymbol{\mathcal{A}}?\varphi\ \alpha\ x\ \&\ (\forall \beta.\ \boldsymbol{\mathcal{A}}?\varphi\ \beta\ x \to \beta = \alpha)\ \mathit{in}\ v]$
unfolding *exists-unique-def* **by** *simp*
then obtain α **where**
$[\boldsymbol{\mathcal{A}}?\varphi\ \alpha\ x\ \&\ (\forall \beta.\ \boldsymbol{\mathcal{A}}?\varphi\ \beta\ x \to \beta = \alpha)\ \mathit{in}\ v]$
by (*rule Instantiate*)
hence $[\boldsymbol{\mathcal{A}}(\alpha = x\ \&\ \psi\ (x^P))\ \mathit{in}\ v]$
using &*E* **by** *blast*
thus $[\boldsymbol{\mathcal{A}}(\psi\ (x^P))\ \mathit{in}\ v]$
using *Act-Basic-2*[*equiv-lr*] &*E* **by** *blast*
next
fix $x\ v$
let $?\varphi = \lambda\ y\ x.\ y = x\ \&\ \psi\ (x^P)$
assume *1*: $[\boldsymbol{\mathcal{A}}\psi\ (x^P)\ \mathit{in}\ v]$
have $[x = x\ \mathit{in}\ v]$
using *id-eq-1*[**where** *'a=ν*] **by** *simp*
hence $[\boldsymbol{\mathcal{A}}(x = x)\ \mathit{in}\ v]$
using *id-act-3*[*equiv-lr*] **by** *fast*
hence $[\boldsymbol{\mathcal{A}}(x = x\ \&\ \psi\ (x^P))\ \mathit{in}\ v]$
using *1 Act-Basic-2*[*equiv-rl*] &*I* **by** *blast*
hence $[\boldsymbol{\mathcal{A}}?\varphi\ x\ x\ \mathit{in}\ v]$
by *simp*
moreover have $[\forall \beta.\ \boldsymbol{\mathcal{A}}?\varphi\ \beta\ x \to \beta = x\ \mathit{in}\ v]$
proof (*rule* $\forall$*I*; *rule CP*)
fix β
assume $[\boldsymbol{\mathcal{A}}?\varphi\ \beta\ x\ \mathit{in}\ v]$
hence $[\boldsymbol{\mathcal{A}}(\beta = x)\ \mathit{in}\ v]$
using *Act-Basic-2*[*equiv-lr*] &*E* **by** *fast*
thus $[\beta = x\ \mathit{in}\ v]$ **using** *id-act-3*[*equiv-rl*] **by** *fast*
qed
ultimately have $[\boldsymbol{\mathcal{A}}?\varphi\ x\ x\ \&\ (\forall \beta.\ \boldsymbol{\mathcal{A}}?\varphi\ \beta\ x \to \beta = x)\ \mathit{in}\ v]$
using &*I* **by** *fast*
hence $[\exists \alpha.\ \boldsymbol{\mathcal{A}}?\varphi\ \alpha\ x\ \&\ (\forall \beta.\ \boldsymbol{\mathcal{A}}?\varphi\ \beta\ x \to \beta = \alpha)\ \mathit{in}\ v]$
by (*rule existential*)

thus $[\exists\,!y.\ \boldsymbol{\mathcal{A}}?\varphi\ y\ x\ in\ v]$
unfolding *exists-unique-def* **by** *simp*
qed

A.13.2. Fake β-Conversion using Description Backdoor

definition *backdoor* **where**
$backdoor \equiv \lambda\ \psi\ .\ \boldsymbol{\lambda} x\ .\ (\!|(\boldsymbol{\lambda} x\ .\ \forall\ \ p\ .\ p \to p),\ \boldsymbol{\iota} y\ .\ y = x\ \&\ \psi\ (x^P)|\!)$

lemma *fake-beta*:
assumes $\bigwedge\ G\ \varphi.\ IsProperInX\ (\lambda x\ .\ (\!|G,\boldsymbol{\iota} y\ .\ \varphi\ y\ x|\!))$
shows $[(\!|backdoor\ (\lambda\ x\ .\ \psi\ x),\ x^P|\!) \equiv \boldsymbol{\mathcal{A}}\psi\ (x^P)\ in\ v]$
proof ($rule\ \equiv I$; *rule CP*)
assume $[(\!|backdoor\ \psi,x^P|\!)\ in\ v]$
hence $[(\!|\boldsymbol{\lambda} x.\ \forall\,p.\ p \to p,\boldsymbol{\iota} y.\ y = x\ \&\ \psi\ (x^P)|\!)\ in\ v]$
using *beta-C-meta-1*[*equiv-lr*, *OF assms*]
unfolding *backdoor-def identity-ν-def* **by** *fast*
hence $[\exists\,!y.\ \boldsymbol{\mathcal{A}}\ (y = x\ \&\ \psi\ (x^P))\ in\ v]$
using *exe-impl-exists*[*equiv-lr*] **by** *fast*
thus $[\boldsymbol{\mathcal{A}}\psi\ (x^P)\ in\ v]$
using *exists-unique-actual-equiv*[*equiv-lr*] **by** *blast*
next
assume $[\boldsymbol{\mathcal{A}}\psi\ (x^P)\ in\ v]$
hence $[\exists\,!y.\ \boldsymbol{\mathcal{A}}\ (y = x\ \&\ \psi\ (x^P))\ in\ v]$
using *exists-unique-actual-equiv*[*equiv-rl*] **by** *blast*
hence $[(\!|\boldsymbol{\lambda} x.\ \forall\,p.\ p \to p,\boldsymbol{\iota} y.\ y = x\ \&\ \psi\ (x^P)|\!)\ in\ v]$
using *exe-impl-exists*[*equiv-rl*] **by** *fast*
thus $[(\!|backdoor\ \psi,x^P|\!)\ in\ v]$
using *beta-C-meta-1*[*equiv-rl*, *OF assms*]
unfolding *backdoor-def* **unfolding** *identity-ν-def* **by** *fast*
qed

lemma *fake-beta-act*:
assumes $\bigwedge\ G\ \varphi.\ IsProperInX\ (\lambda x\ .\ (\!|G,\boldsymbol{\iota} y\ .\ \varphi\ y\ x|\!))$
shows $[(\!|backdoor\ (\lambda\ x\ .\ \psi\ x),\ x^P|\!) \equiv \psi\ (x^P)\ in\ dw]$
using *fake-beta*[*OF assms*]
logic-actual[*necessitation-averse-axiom-instance*]
intro-elim-6-e **by** *blast*

A.13.3. Resulting Paradox

lemma *paradox*:
assumes $\bigwedge\ G\ \varphi.\ IsProperInX\ (\lambda x\ .\ (\!|G,\boldsymbol{\iota} y\ .\ \varphi\ y\ x|\!))$
shows *False*
proof −
obtain K **where** *K-def*:
$K = backdoor\ (\lambda\ x\ .\ \exists\ \ F\ .\ \{\!|x,F|\!\}\ \&\ \neg(\!|F,x|\!))$ **by** *auto*
have $[\exists\,x.\ (\!|A!,x^P|\!)\ \&\ (\forall\,F.\ \{\!|x^P,F|\!\} \equiv (F = K))\ in\ dw]$
using *A-objects*[*axiom-instance*] **by** *fast*
then obtain x **where** *x-prop*:
$[(\!|A!,x^P|\!)\ \&\ (\forall\,F.\ \{\!|x^P,F|\!\} \equiv (F = K))\ in\ dw]$
by (*rule Instantiate*)
{
assume $[(\!|K,x^P|\!)\ in\ dw]$
hence $[\exists\ \ F\ .\ \{\!|x^P,F|\!\}\ \&\ \neg(\!|F,x^P|\!)\ in\ dw]$
unfolding *K-def* **using** *fake-beta-act*[*OF assms*, *equiv-lr*]
by *blast*
then obtain F **where** *F-def*:

```
      [⦃x^P,F⦄ & ¬⦇F,x^P⦈ in dw] by (rule Instantiate)
    hence [F = K in dw]
      using x-prop[conj2, THEN ∀E[where β=F], equiv-lr]
        &E unfolding K-def by blast
    hence [¬⦇K,x^P⦈ in dw]
      using l-identity[axiom-instance,deduction,deduction]
            F-def[conj2] by fast
  }
  hence 1: [¬⦇K,x^P⦈ in dw]
    using reductio-aa-1 by blast
  hence [¬(∃ F . ⦃x^P,F⦄ & ¬⦇F,x^P⦈) in dw]
    using fake-beta-act[OF assms,
          THEN oth-class-taut-5-d[equiv-lr],
          equiv-lr]
    unfolding K-def by blast
  hence [∀ F . ⦃x^P,F⦄ → ⦇F,x^P⦈ in dw]
    apply − unfolding exists-def by PLM-solver
  moreover have [⦃x^P,K⦄ in dw]
    using x-prop[conj2, THEN ∀E[where β=K], equiv-rl]
          id-eq-1 by blast
  ultimately have [⦇K,x^P⦈ in dw]
    using ∀E vdash-properties-10 by blast
  hence ⋀φ. [φ in dw]
    using raa-cor-2 1 by blast
  thus False using Semantics.T4 by auto
qed
```

A.13.4. Original Version of the Paradox

Originally the paradox was discovered using the following construction based on the comprehension theorem for relations without the explicit construction of the description backdoor and the resulting fake-β-conversion.

```
lemma assumes ⋀ G φ. IsProperInX (λx . ⦇G,ιy . φ y x⦈)
shows Fx-equiv-xH: [∀ H . ∃ F . □(∀x. ⦇F,x^P⦈ ≡ ⦃x^P,H⦄) in v]
proof (rule ∀I)
  fix H
  let ?G = (𝛌x . ∀ p . p → p)
  obtain φ where φ-def: φ = (λ y x . (y^P) = x & ⦃x,H⦄) by auto
  have [∃F. □(∀x. ⦇F,x^P⦈ ≡ ⦇?G,ιy . φ y (x^P)⦈) in v]
    using relations-1[OF assms] by simp
  hence 1: [∃F. □(∀x. ⦇F,x^P⦈ ≡ (∃!y . 𝓐φ y (x^P))) in v]
    apply − apply (PLM-subst-method
        λ x . ⦇?G,ιy . φ y (x^P)⦈ λ x . (∃!y. 𝓐φ y (x^P)))
    using exe-impl-exists by auto
  then obtain F where F-def: [□(∀x. ⦇F,x^P⦈ ≡ (∃!y . 𝓐φ y (x^P))) in v]
    by (rule Instantiate)
  moreover have 2: ⋀ v x . [(∃!y . 𝓐φ y (x^P)) ≡ ⦃x^P,H⦄ in v]
  proof (rule ≡I; rule CP)
    fix x v
    assume [∃!y. 𝓐φ y (x^P) in v]
    hence [∃α. 𝓐φ α (x^P) & (∀β. 𝓐φ β (x^P) → β = α) in v]
      unfolding exists-unique-def by simp
    then obtain α where [𝓐φ α (x^P) & (∀β. 𝓐φ β (x^P) → β = α) in v]
      by (rule Instantiate)
    hence [𝓐(α^P = x^P & ⦃x^P,H⦄) in v]
      unfolding φ-def using &E by blast
    hence [𝓐(⦃x^P,H⦄) in v]
```

```
      using Act-Basic-2[equiv-lr] &E by blast
    thus [⦃x^P,H⦄ in v]
      using en-eq-10[equiv-lr] by simp
  next
    fix x v
    assume [⦃x^P,H⦄ in v]
    hence 1: [𝓐(⦃x^P,H⦄) in v]
      using en-eq-10[equiv-rl] by blast
    have [x = x in v]
      using id-eq-1[where 'a=ν] by simp
    hence [𝓐(x = x) in v]
      using id-act-3[equiv-lr] by fast
    hence [𝓐(x^P = x^P & ⦃x^P,H⦄) in v]
      unfolding identity-ν-def using 1 Act-Basic-2[equiv-rl] &I by blast
    hence [𝓐φ x (x^P) in v]
      unfolding φ-def by simp
    moreover have [∀ β. 𝓐φ β (x^P) → β = x in v]
    proof (rule ∀ I; rule CP)
      fix β
      assume [𝓐φ β (x^P) in v]
      hence [𝓐(β = x) in v]
        unfolding φ-def identity-ν-def
        using Act-Basic-2[equiv-lr] &E by fast
      thus [β = x in v] using id-act-3[equiv-rl] by fast
    qed
    ultimately have [𝓐φ x (x^P) & (∀ β. 𝓐φ β (x^P) → β = x) in v]
      using &I by fast
    hence [∃ α. 𝓐φ α (x^P) & (∀ β. 𝓐φ β (x^P) → β = α) in v]
      by (rule existential)
    thus [∃ !y. 𝓐φ y (x^P) in v]
      unfolding exists-unique-def by simp
  qed
  have [□(∀ x. ⦇F,x^P⦈ ≡ ⦃x^P,H⦄) in v]
    apply (PLM-subst-goal-method
        λφ . □(∀ x. ⦇F,x^P⦈ ≡ φ x)
        λ x . (∃ !y . 𝓐φ y (x^P)))
    using 2 F-def by auto
  thus [∃ F . □(∀ x. ⦇F,x^P⦈ ≡ ⦃x^P,H⦄) in v]
    by (rule existential)
qed


lemma
  assumes is-propositional: (⋀G φ. IsProperInX (λx. ⦇G,ιy. φ y x⦈))
      and Abs-x: [⦇A!,x^P⦈ in v]
      and Abs-y: [⦇A!,y^P⦈ in v]
      and noteq: [x ≠ y in v]
shows diffprop: [∃ F . ¬(⦇F,x^P⦈ ≡ ⦇F,y^P⦈) in v]
proof –
  have [∃ F . ¬(⦃x^P, F⦄ ≡ ⦃y^P, F⦄) in v]
    using noteq unfolding exists-def
  proof (rule reductio-aa-2)
    assume 1: [∀ F. ¬¬(⦃x^P,F⦄ ≡ ⦃y^P,F⦄) in v]
    {
      fix F
      have [(⦃x^P,F⦄ ≡ ⦃y^P,F⦄) in v]
        using 1[THEN ∀ E] useful-tautologies-1[deduction] by blast
    }
```

hence $[\forall F.\ \{\!|x^P,F|\!\} \equiv \{\!|y^P,F|\!\}\ \mathit{in}\ v]$ **by** (*rule* $\forall I$)
thus $[x = y\ \mathit{in}\ v]$
unfolding *identity-ν-def*
using *ab-obey-1*[*deduction*, *deduction*]
Abs-x Abs-y **&***I* **by** *blast*
qed
then obtain H **where** *H-def*: $[\neg(\{\!|x^P,\ H|\!\} \equiv \{\!|y^P,\ H|\!\})\ \mathit{in}\ v]$
by (*rule Instantiate*)
hence *2*: $[(\{\!|x^P,\ H|\!\}$ **&** $\neg\{\!|y^P,\ H|\!\}) \lor (\neg\{\!|x^P,\ H|\!\}$ **&** $\{\!|y^P,\ H|\!\})\ \mathit{in}\ v]$
apply − **by** *PLM-solver*
have $[\exists F.\ \Box(\forall x.\ (\!|F,x^P|\!) \equiv \{\!|x^P,H|\!\})\ \mathit{in}\ v]$
using *Fx-equiv-xH*[*OF is-propositional*, *THEN* $\forall E$] **by** *simp*
then obtain F **where** $[\Box(\forall x.\ (\!|F,x^P|\!) \equiv \{\!|x^P,H|\!\})\ \mathit{in}\ v]$
by (*rule Instantiate*)
hence *F-prop*: $[\forall x.\ (\!|F,x^P|\!) \equiv \{\!|x^P,H|\!\}\ \mathit{in}\ v]$
using *qml-2*[*axiom-instance*, *deduction*] **by** *blast*
hence *a*: $[(\!|F,x^P|\!) \equiv \{\!|x^P,H|\!\}\ \mathit{in}\ v]$
using $\forall E$ **by** *blast*
have *b*: $[(\!|F,y^P|\!) \equiv \{\!|y^P,H|\!\}\ \mathit{in}\ v]$
using *F-prop* $\forall E$ **by** *blast*
{
assume *1*: $[\{\!|x^P,\ H|\!\}$ **&** $\neg\{\!|y^P,\ H|\!\}\ \mathit{in}\ v]$
hence $[(\!|F,x^P|\!)\ \mathit{in}\ v]$
using *a*[*equiv-rl*] **&***E* **by** *blast*
moreover have $[\neg(\!|F,y^P|\!)\ \mathit{in}\ v]$
using *b*[*THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-rl*] *1*[*conj2*] **by** *auto*
ultimately have $[(\!|F,x^P|\!)$ **&** $(\neg(\!|F,y^P|\!))\ \mathit{in}\ v]$
by (*rule* **&***I*)
hence $[((\!|F,x^P|\!)$ **&** $\neg(\!|F,y^P|\!)) \lor (\neg(\!|F,x^P|\!)$ **&** $(\!|F,y^P|\!))\ \mathit{in}\ v]$
using $\lor I$ **by** *blast*
hence $[\neg((\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \mathit{in}\ v]$
using *oth-class-taut-5-j*[*equiv-rl*] **by** *blast*
}
moreover {
assume *1*: $[\neg\{\!|x^P,\ H|\!\}$ **&** $\{\!|y^P,\ H|\!\}\ \mathit{in}\ v]$
hence $[(\!|F,y^P|\!)\ \mathit{in}\ v]$
using *b*[*equiv-rl*] **&***E* **by** *blast*
moreover have $[\neg(\!|F,x^P|\!)\ \mathit{in}\ v]$
using *a*[*THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-rl*] *1*[*conj1*] **by** *auto*
ultimately have $[\neg(\!|F,x^P|\!)$ **&** $(\!|F,y^P|\!)\ \mathit{in}\ v]$
using **&***I* **by** *blast*
hence $[((\!|F,x^P|\!)$ **&** $\neg(\!|F,y^P|\!)) \lor (\neg(\!|F,x^P|\!)$ **&** $(\!|F,y^P|\!))\ \mathit{in}\ v]$
using $\lor I$ **by** *blast*
hence $[\neg((\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \mathit{in}\ v]$
using *oth-class-taut-5-j*[*equiv-rl*] **by** *blast*
}
ultimately have $[\neg((\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \mathit{in}\ v]$
using *2 intro-elim-4-b reductio-aa-1* **by** *blast*
thus $[\exists\ F\ .\ \neg((\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \mathit{in}\ v]$
by (*rule existential*)
qed

lemma *original-paradox*:
assumes *is-propositional*: $(\bigwedge G\ \varphi.\ \mathit{IsProperInX}\ (\lambda x.\ (\!|G,\iota y.\ \varphi\ y\ x|\!)))$
shows *False*
proof −
fix v
have $[\exists x\ y.\ (\!|A!,x^P|\!)$ **&** $(\!|A!,y^P|\!)$ **&** $x \neq y$ **&** $(\forall F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \mathit{in}\ v]$

using *aclassical2* **by** *auto*
then obtain x **where**
$[\exists\ y.\ (\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ x \neq y\ \&\ (\forall\, F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \mathit{in}\ v]$
by (*rule Instantiate*)
then obtain y **where** *xy-def*:
$[(\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ x \neq y\ \&\ (\forall\, F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \mathit{in}\ v]$
by (*rule Instantiate*)
have $[\exists\ F\ .\ \neg((\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \mathit{in}\ v]$
using *diffprop*[*OF assms*, *OF xy-def*[*conj1*,*conj1*,*conj1*],
OF xy-def[*conj1*,*conj1*,*conj2*],
OF xy-def[*conj1*,*conj2*]]
by *auto*
then obtain F **where** $[\neg((\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \mathit{in}\ v]$
by (*rule Instantiate*)
moreover have $[(\!|F,x^P|\!) \equiv (\!|F,y^P|\!)\ \mathit{in}\ v]$
using *xy-def*[*conj2*] **by** (*rule* $\forall$ *E*)
ultimately have $\bigwedge\varphi.[\varphi\ \mathit{in}\ v]$
using *PLM.raa-cor-2* **by** *blast*
thus *False*
using *Semantics.T4* **by** *auto*
qed

end

Bibliography

[1] C. Benzmüller. Universal reasoning, rational argumentation and human-machine interaction. *CoRR*, abs/1703.09620, 2017.

[2] C. Benzmüller and D. Miller. Automation of higher-order logic. In D. M. Gabbay, J. H. Siekmann, and J. Woods, editors, *Handbook of the History of Logic, Volume 9 — Computational Logic*, pages 215–254. North Holland, Elsevier, 2014.

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

[4] C. Benzmüller and D. S. Scott. Axiomatizing category theory in free logic. *CoRR*, abs/1609.01493, 2016.

[5] 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.

[6] I. M. L. D'Ottaviano and H. de Araújo Feitosa. On gödel's modal interpretation of the intuitionistic logic. In *Universal Logic: An Anthology*, pages 71–88. Springer Basel, 2012.

[7] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

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

[9] G. Rosen. Abstract objects. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2017 edition, 2017.

[10] E. Zalta. *Abstract Objects: An Introduction to Axiomatic Metaphysics*. Synthese Library. Springer, 1983.

[11] E. Zalta. *Intensional Logic and the Metaphysics of Intentionality*. A Bradford book. MIT Press, 1988.

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

[13] E. N. Zalta. The theory of abstract objects. http://mally.stanford.edu/theory.html. Accessed: April 04, 2017.

Selbstständigkeitserklärung

Name:	Kirchner
Vorname:	Daniel
geb.am:	22.05.1989
Matr.Nr.:	4387161

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle Ausführungen, die wörtlich oder inhaltlich aus fremden Quellen übernommen sind, habe ich als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch bei keiner anderen Universität als Prüfungsleistung eingereicht und ist auch noch nicht veröffentlicht.

Berlin, 27. Mai 2017 Daniel Kirchner