



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

Representation and Partial Automation of the Principia-Logico Metaphysica in Isabelle/HOL

Daniel Kirchner

Matrikelnummer: 4387161

Supervisor:

Priv.-Doz. Dr.-Ing. Christoph Benzmüller

Berlin, May 29, 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. 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	7
1.1. Universal Logical Reasoning	7
1.2. Shallow Semantical Embeddings in HOL	8
1.3. Relational Type Theory vs. Functional Type Theory	9
1.4. Overview of the following Chapters	10
2. The Theory of Abstract Objects	11
2.1. Motivation	11
2.2. Basic Principles	13
2.3. The Language of PLM	14
2.4. The Axioms	16
2.5. Hyperintensionality of Relations	16
2.6. The Aczel-Model	17
3. The Embedding	19
3.1. The Framework Isabelle/HOL	19
3.2. A Russell-style Paradox	19
3.3. Basic Concepts	20
3.4. The Representation Layer	23
3.5. Semantical Abstraction	32
3.6. General All-Quantifier	36
3.7. Derived Language Elements	37
3.8. The Proving Method <code>meta_solver</code>	38
3.9. General Identity Relation	40
3.10. The Axiom System of PLM	42
3.11. The Deductive System PLM	48
3.12. Artificial Theorems	53
3.13. Sanity Tests	55
4. Technical Limitations of Isabelle/HOL	56
4.1. Limitations of Type Classes and Locales	56
4.2. Case Distinctions by Type	57
4.3. Structural Induction and Proof-Theoretic Reasoning	58
5. Discussion and Results	59
5.1. Differences between the Embedding and PLM	59
5.2. A Paradox in PLM	62

5.3. A Meta-Conjecture about Possible Worlds	64
5.4. Functional Object Theory	65
5.5. Relations vs. Functions	66
5.6. Conclusion	67
A. Isabelle Theory	69
A.1. Embedding	69
A.2. Semantics	74
A.3. General Quantification	80
A.4. Basic Definitions	81
A.5. MetaSolver	82
A.6. General Identity	92
A.7. The Axioms of Principia Metaphysica	94
A.8. Definitions	101
A.9. The Deductive System PLM	103
A.10. Possible Worlds	197
A.11. Artificial Theorems	202
A.12. Sanity Tests	204
A.13. Paradox	208
Bibliography	215

1. Introduction

Calculus!

Leibniz

1.1. Universal Logical Reasoning¹

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: *Calculus!*

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.

¹This 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 them.

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 ontological 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/HOL².

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 logic³.

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.

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

³ Early attempts of an embedding by Christoph Benz Müller (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

¹The 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 summarized 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.

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 (see [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.

Using these principles it is already possible 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. The theory allows it to formulate purely *syntactic* definitions of objects like possible worlds and truth values and from these 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

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

	δ	::=	a_1, a_2, \dots
	ν	::=	x_1, x_2, \dots
$(n \geq 0)$	Σ^n	::=	P_1^n, P_2^n, \dots
$(n \geq 0)$	Ω^n	::=	F_1^n, F_2^n, \dots
	α	::=	$\nu \mid \Omega^n \ (n \geq 0)$
	κ	::=	$\delta \mid \nu \mid \iota \nu \phi$
$(n \geq 1)$	Π^n	::=	$\Sigma^n \mid \Omega^n \mid [\lambda \nu_1 \dots \nu_n \phi^*]$
	Π^0	::=	$\Sigma^0 \mid \Omega^0 \mid [\lambda \phi^*] \mid \phi^*$
	ϕ^*	::=	$\Pi^n \kappa_1 \dots \kappa_n \ (n \geq 1) \mid \Pi^0 \mid (\neg \phi^*) \mid (\phi^* \rightarrow \phi^*) \mid \forall \alpha \phi^* \mid$ $(\phi^*) \mid (A \phi^*)$
	ϕ	::=	$\kappa_1 \Pi^1 \mid \phi^* \mid (\neg \phi) \mid (\phi \rightarrow \phi) \mid \forall \alpha \phi \mid (\phi) \mid (A \phi)$
	τ	::=	$\kappa \mid \Pi^n \ (n \geq 0)$

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 that xQ is *not* a subformula of $\iota x(xQ)$ ². 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 discussion in section 5.2, that describes a paradox in the formulation of the theory in the draft of PLM at the time of writing³.

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.

The language does not contain a primitive identity, but *defines* an identity for each type of term 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 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.

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

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

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.

The fact that the following relation terms do *not* necessarily denote the same relations illustrates the extent of this *hyperintensionality*:

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

Of course the theory can be extended in such a way that these properties are equal - without additional axioms, however, their equality is not derivable.

Although the relations of object theory are hyperintensional 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 Aczel⁴ 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 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].

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

⁴In 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

Aczel Model of Object Theory



Define, for assignment to variables g , In this model, the following are true:

$g \models Fx \text{ iff } g(x) \in g(F)$ $g \models xF \text{ iff } g(F) \in g(x)$	$\exists x (A!x \ \& \ \forall F (xF \equiv \varphi))$ $\exists F \forall x (Fx \equiv \varphi), \ \varphi \text{ has no encoding subformulas}$
--	--

exemplified by the same objects. Consequently 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 soundness and completeness that can be utilized in the analysis of semantical embeddings[3].

For the Theory of Abstract Objects the situation is different. 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 that serves as a provably consistent basis for the theory. Then several abstraction layers are implemented on top of the model structure in order to enable reasoning that is independent of the particular representation. 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 can still be defined as set membership of a relation in an abstract object. Exemplification is defined as function application of a relation to the urelement corresponding to an individual, a state and a possible world. 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 summary, 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 relations in the theory, 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 soundness and 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 away from the details of the representation by implementing an approximation of the formal semantics of PLM². 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 generalization), 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.

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

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

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 described 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 v is defined as **datatype** $v = \omega v \ \omega \mid \sigma v \ \sigma$. This type represents urelements and an object of this type can be either an ordinary or a special urelement (with the respective type constructors ωv and σv).
- 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 $v \Rightarrow j \Rightarrow i \Rightarrow bool$ 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 $v \Rightarrow v \Rightarrow j \Rightarrow i \Rightarrow bool$ and represents two-place relations.
- Type Π_3 is defined as the set of all functions of type $v \Rightarrow v \Rightarrow v \Rightarrow j \Rightarrow i \Rightarrow bool$ 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 evalo makeo ..` 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 *evalo* maps an object of abstract type o to its representative of type $j \Rightarrow i \Rightarrow bool$, whereas the morphism *makeo* 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 types³.

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 $\textcolor{blue}{_P}$ is introduced (see A.1.3):

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

The expression $\textcolor{blue}{x}^P$ (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:

$$\textcolor{blue}{\lambda}x . \varphi x = (\textit{if } \exists!x. (\varphi x) \textit{ dj dw then Some (THE } x. (\varphi x) \textit{ dj dw) else None})$$

³The 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 $\exists!x. \varphi x \text{ dj } dw$ holds, i.e. *there exists a unique x , such that φx holds for the actual state and the actual world*, the term $\iota x. \varphi x$ evaluates to *Some* (*THE* $x. \varphi x \text{ dj } dw$). 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:

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

the maps an object of type $'a \text{ option}$ that is of the form *Some* x to x and is undefined for *None*. For an object of type κ the expression $\text{proper } x$ is true, if the term is logically proper, and if this is the case, the expression $\text{rep } x$ 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 $\nu\nu$ of type $\nu \Rightarrow \nu$ can be defined as follows:

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

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

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

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

Remark. *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):

- $\langle p \rangle = p$
- $\langle F, x \rangle = (\lambda s w. \text{proper } x \wedge F (\nu\nu (\text{rep } x)) s w)$
- $\langle F, x, y \rangle = (\lambda s w. \text{proper } x \wedge \text{proper } y \wedge F (\nu\nu (\text{rep } x)) (\nu\nu (\text{rep } y)) s w)$
- $\langle F, x, y, z \rangle =$
 $(\lambda s w. \text{proper } x \wedge$
 $\text{proper } y \wedge \text{proper } z \wedge F (\nu\nu (\text{rep } x)) (\nu\nu (\text{rep } y)) (\nu\nu (\text{rep } z)) s w)$

3.4.5. Encoding

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

$$\langle x, F \rangle = (\lambda s w. \text{proper } x \wedge (\text{case rep } x \text{ of } \omega\nu \omega \Rightarrow \text{False} \mid \alpha\nu \alpha \Rightarrow F \in \alpha))$$

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

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 state⁴.

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

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

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):

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

⁴Early 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.

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

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 $s \neq dj$:

- $s = dj \implies \neg p s w = (\neg p s w)$
- $s \neq dj \implies \neg p s w = I-NOT s (p s) w$
- $s = dj \implies p \rightarrow q s w = (p s w \longrightarrow q s w)$
- $s \neq dj \implies p \rightarrow q s w = I-IMPL s (p s) (q s) w$

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):

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

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 λ^2 and λ^3 as shown above.

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

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

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

$$\lambda x. \varphi x u s w = (\exists x. \nu\nu x = u \wedge \varphi x s w)$$

If restricted to ordinary objects, the definition can be simplified, since $\nu\nu$ is bijective on the set of ordinary objects:

$$\lambda x. \varphi x (\omega\nu u) s w = (\varphi (\omega\nu u)) s w$$

However in general $\nu\nu$ 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 urelements corresponding to the individuals (see A.1.9):

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

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

$$IsProperInX \varphi = (\forall w x. \lambda x. \varphi (x^P) (\nu\nu x) dj w = \varphi (x^P) dj w)$$

Remark. *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).*

Maps that contain encoding formulas in their arguments are in general not proper and thereby the paradox mentioned in section 3.2 is prevented.

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 separately in section 5.1.1.

3.4.8. Validity

Semantic validity is defined as follows (see A.1.10):

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

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 . 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 $[p \text{ in } v]$ 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):

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

- $\exists w. \forall x. \text{ConcreteInWorld } x \ w \longrightarrow (\forall v. \text{ConcreteInWorld } x \ v)$

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

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

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	\circ
$\neg\varphi$	not φ	$\neg\varphi$	\circ
$\varphi \rightarrow \psi$	φ implies ψ	$\varphi \rightarrow \psi$	\circ
$\Box\varphi$	necessarily φ	$\Box\varphi$	\circ
$\mathcal{A}\varphi$	actually φ	$\mathcal{A}\varphi$	\circ
Πv	v (an individual term) exemplifies Π	$\langle \Pi, v \rangle$	\circ
Πx	x (an individual variable) exemplifies Π	$\langle \Pi, x^P \rangle$	\circ
$\Pi v_1 v_2$	v_1 and v_2 exemplify Π	$\langle \Pi, v_1, v_2 \rangle$	\circ
Πxy	x and y exemplify Π	$\langle \Pi, x^P, y^P \rangle$	\circ
$\Pi v_1 v_2 v_3$	v_1, v_2 and v_3 exemplify Π	$\langle \Pi, v_1, v_2, v_3 \rangle$	\circ
Πxyz	x, y and z exemplify Π	$\langle \Pi, x^P, y^P, z^P \rangle$	\circ
$v\Pi$	v encodes Π	$\{v, \Pi\}$	\circ
$\iota x \varphi$	the x , such that φ	$\iota x. \varphi \ x$	κ
$\forall x(\varphi)$	for all individuals x it holds that φ	$\forall_\nu x. \varphi \ x$	\circ
$\forall p(\varphi)$	for all propositions p it holds that φ	$\forall_0 p. \varphi \ p$	\circ
$\forall F(\varphi)$	for all relations F it holds that φ	$\forall_1 F. \varphi \ F$	\circ
		$\forall_2 F. \varphi \ F$	
		$\forall_3 F. \varphi \ F$	
$[\lambda \ p]$	being such that p	$\lambda^0 \ p$	Π_0
$[\lambda x \ \varphi]$	being x such that φ	$\lambda x. \varphi \ x$	Π_1
$[\lambda xy \ \varphi]$	being x and y such that φ	$\lambda^2 \ (\lambda x \ y. \varphi \ x \ y)$	Π_2
$[\lambda xyz \ \varphi]$	being x, y and z such that φ	$\lambda^3 \ (\lambda x \ y \ z. \varphi \ x \ y \ z)$	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 $\textcolor{blue}{_}^P$.
- 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:

$$[\textcolor{blue}{\varphi} \text{ in } v]$$

Here $\textcolor{blue}{\varphi}$ 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 $\textcolor{blue}{\varphi}$ and all possible worlds v (unless $\textcolor{blue}{\varphi}$ 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 , \circ 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 domain 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). 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, \dots, 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 \circ as synonym of $j \Rightarrow i \Rightarrow \text{bool}$, it was introduced as a distinct type for which the set of all functions of type $j \Rightarrow i \Rightarrow \text{bool}$ merely serves as the underlying set of representatives. An object of type \circ cannot directly be substituted for a variable of type $j \Rightarrow i \Rightarrow \text{bool}$. To do so it first has to be mapped to its representative of type $j \Rightarrow i \Rightarrow \text{bool}$ 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 values 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):

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

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).

Exemplification extensions as maps to sets of *individuals* are independent of the underlying model and in particular do not require the concept of *urelements* as they are present in an Aczel-model. Their use in the definition of truth conditions for exemplification formulas below 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):

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

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 Conditions of Formulas

Based on the 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):

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

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):

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

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

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

- $[\neg\psi \text{ in } w] = (\neg [\psi \text{ in } w])$
- $[\psi \rightarrow \chi \text{ in } w] = (\neg [\psi \text{ in } w] \vee [\chi \text{ in } w])$
- $[\Box\psi \text{ in } w] = (\forall v. [\psi \text{ in } v])$
- $[\mathcal{A}\psi \text{ in } w] = [\psi \text{ in } dw]$
- $[\forall_{\nu} x. \psi x \text{ in } w] = (\forall x. [\psi x \text{ in } w])$
- $[\forall_0 x. \psi x \text{ in } w] = (\forall x. [\psi x \text{ in } w])$
- $[\forall_1 x. \psi x \text{ in } w] = (\forall x. [\psi x \text{ in } w])$
- $[\forall_2 x. \psi x \text{ in } w] = (\forall x. [\psi x \text{ in } w])$
- $[\forall_3 x. \psi x \text{ in } w] = (\forall x. [\psi x \text{ in } w])$

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 $\mathcal{A}\psi$ 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:

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

If there exists a unique x , such that ψx 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 terms have denotations, 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):

- $d_1 \lambda x. (\Pi, x^P) = d_1 \Pi$
- $IsProperInX \varphi \implies$
 $Some\ r = d_1 \lambda x. \varphi (x^P) \wedge Some\ o_1 = d_\kappa x \longrightarrow (o_1 \in ex1\ r\ w) = [\varphi\ x\ in\ w]$
- $Some\ r = d_0 \lambda^0 \varphi \longrightarrow ex0\ r\ w = [\varphi\ in\ w]$

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.

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

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

$$d_1 F = d_1 G \implies F = G$$

6. The denotation function for individual terms is injective for denoting terms:

$$\text{Some } o_1 = d_\kappa x \wedge \text{Some } o_1 = d_\kappa y \implies x = y$$

Especially statements 5 and 6 are only derivable due to the specific construction of the representation layer: since the semantic domains were defined as the representation sets of the respective abstract types and denotations were defined canonically, objects that have the same denotation are identical as objects of the 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. Using the provided introduction rules Isabelle's unifier can derive the propriety of such maps automatically.

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 that the type class has to be *instantiated* for the desired type. This is done by first providing definitions for the constants of the type class specific to the respective type. Then each assumption made by the type class has to be proven given the particular type and the provided definitions. After that any statement that was proven for the type class can be applied to the instantiated 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: $[\forall x. \psi \ x \text{ in } w] = (\forall x. [\psi \ x \text{ in } w])$.

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 can completely 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 a minimal set of primitive elements. This section introduces further derived language elements that are defined directly in the embedded logic.

Notably identity is not part of the primitive language, but introduced as a *defined* concept.

3.7.1. Connectives

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

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

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

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

3.7.2. Identity

The definitions for identity are stated separately for each type of term (see A.4.3):

- $x =_E y = (\lambda^2 (\lambda x y . (\lambda O! . x^P) \& (\lambda O! . y^P) \& \square(\forall F . (\lambda F . x^P) \equiv (\lambda F . y^P))), x, y)$
- $F =_1 G = \square(\forall x . \{x^P, F\} \equiv \{x^P, G\})$
- $F =_2 G = \forall x . (\lambda y . (\lambda F . x^P, y^P)) =_1 (\lambda y . (\lambda G . x^P, y^P)) \& (\lambda y . (\lambda F . y^P, x^P)) =_1 (\lambda y . (\lambda G . y^P, x^P))$
- $F =_3 G = \forall x y . (\lambda z . (\lambda F . z^P, x^P, y^P)) =_1 (\lambda z . (\lambda G . z^P, x^P, y^P)) \& (\lambda z . (\lambda F . x^P, z^P, y^P)) =_1 (\lambda z . (\lambda G . x^P, z^P, y^P)) \& (\lambda z . (\lambda F . y^P, x^P, z^P)) =_1 (\lambda z . (\lambda G . y^P, x^P, z^P))$
- $p =_0 q = (\lambda x . p) =_1 (\lambda x . q)$

Similarly to the general all-quantifier it makes sense to introduce a general identity relation for all types of terms (κ , o resp. $\Pi_0, \Pi_1, \Pi_2, \Pi_3$). However, whereas all-quantification is characterized by a semantic criterion that can be generalized in a type class, identity is defined independently for each type. Therefore a general identity symbol will only be introduced in section 3.9, since it will then be possible to formulate and prove a reasonable property shared by the identity of all types of terms.

3.8. The Proving Method `meta_solver`

3.8.1. General Concept

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

The idea behind this structure is that reasoning in subsequent layers should - as far as possible - only rely on the previous layer. However, the restriction of proofs to a specific subset of the facts that are valid in the global context can be cumbersome for automated reasoning. While it is possible to restrict automated reasoning tools to only consider specific sets of facts, 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 utilizes the Isabelle package *Eisbach*. 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 the construction of more complex proving methods that involve pattern matching. This functionality is 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 a relation in the meta-logic as required by the semantics.

For example an implication formula can be deconstructed as follows:

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

Whereas the basic proving methods available in Isabelle cannot immediately prove $[\varphi \rightarrow \varphi \text{ in } v]$ without any facts about the definitions of validity and implication, they can prove $[\varphi \text{ in } v] \longrightarrow [\varphi \text{ in } v]$ 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 complex formulas of the embedded logic to meta-logical statements involving simpler formulas.

The formulation of appropriate introduction, elimination and substitution rules for the logical connectives and quantifiers is straightforward. 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):

$$[(\lambda F, x) \text{ in } v] = (\exists r \ o_1. \text{Some } r = d_1 \ F \wedge \text{Some } o_1 = d_\kappa \ x \wedge o_1 \in \text{ex1 } r \ v)$$

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.

Additionally the *meta-solver* is equipped with rules for being abstract and ordinary and for the defined identity.

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 derived as rules for the *meta-solver*:

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

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

The fact that they are derivable 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. A derivation that is solely based on the semantics on the other

hand, would require a complex induction proof. For this reason it is considered a reasonable compromise to include these statements as admissible rules for the *meta-solver*. However, future work may attempt to enforce the separation of layers more strictly and consequently abstain from these rules.

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 completely by the supplied rules. Moreover the constructed custom proving method serves as a proof of concept and may 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 to support 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. The *meta-solver* is only supplied with rules that are *reversible*. Thereby it is justified to use it to simplify a statement before employing a tool like **nitpick** in order to look for models or 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 is derivable in the formal system of PLM, since the result still depends on the specific structure of the representation layer. However, based on the concept of the *meta-solver* another proving method 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 proofs 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 term, some other property has to be found that is shared by the respective definitions and can reasonably be used as the condition of a type class.

A natural choice for such a condition is 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 *identifiable* (see A.6.1):

$$[\alpha = \beta \text{ in } v] \wedge [\varphi \alpha \text{ in } v] \implies [\varphi \beta \text{ in } v]$$

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 (see A.7):

$$[[\varphi]] = (\forall v. [\varphi \text{ in } v])$$

Axioms are unconditionally true in all possible worlds. The only exceptions are *necessitation-averse*, resp. *modally-fragile* axioms⁵. Such axioms are stated using the following syntax:

$$[\varphi] = [\varphi \text{ in } dw]$$

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. 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 (except for modally-fragile axioms) the necessitation of the schema is also an axiom.

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):

- $[[\varphi]] \implies [\varphi \text{ in } v]$
- $(\bigwedge x. [[\varphi x]]) \implies [[\forall x. \varphi x]]$
- $[[\varphi]] \implies [[\mathcal{A}\varphi]]$
- $[[\varphi]] \implies [[\Box\varphi]]$

For modally-fragile axioms only the following rules are introduced:

- $[\varphi] \implies [\varphi \text{ in } dw]$
- $(\bigwedge x. [\varphi x]) \implies [\forall x. \varphi x]$

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 $[\forall x. \Box\varphi x \rightarrow \varphi x \text{ in } v]$ by not referencing it directly as *qml-2*, but by applying the defined attributes to it: *qml-2[axiom-universal, axiom-instance]*

⁵Currently PLM uses only one such axiom, see 3.10.6.

3.10.2. Derivation of the Axioms

To simplify the derivation of the axioms a proving method *axiom-meta-solver* is introduced, that unfolds the dedicated syntax, then applies the meta-solver and if possible resolves the proof objective automatically.

Most of the axioms can be derived by the *axiom-meta-solver* directly. Some axioms, however, require more verbose proofs or their representation in the functional setting of Isabelle/HOL requires special attention. 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):

$$\bullet \llbracket \varphi \rightarrow (\psi \rightarrow \varphi) \rrbracket \quad (21.1)$$

$$\bullet \llbracket \varphi \rightarrow (\psi \rightarrow \chi) \rightarrow (\varphi \rightarrow \psi \rightarrow (\varphi \rightarrow \chi)) \rrbracket \quad (21.2)$$

$$\bullet \llbracket \neg\varphi \rightarrow \neg\psi \rightarrow (\neg\varphi \rightarrow \psi \rightarrow \varphi) \rrbracket \quad (21.3)$$

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):

$$\bullet \llbracket \alpha = \beta \rightarrow (\varphi \alpha \rightarrow \varphi \beta) \rrbracket \quad (25)$$

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):

$$\bullet \llbracket (\forall \alpha. \varphi \alpha) \rightarrow \varphi \alpha \rrbracket \quad (29.1)$$

$$\bullet \llbracket (\forall \alpha. \varphi (\alpha^P)) \rightarrow ((\exists \beta. \beta^P = \alpha) \rightarrow \varphi \alpha) \rrbracket \quad (29.1)$$

$$\bullet \llbracket (\forall \alpha. \varphi \alpha \rightarrow \psi \alpha) \rightarrow ((\forall \alpha. \varphi \alpha) \rightarrow (\forall \alpha. \psi \alpha)) \rrbracket \quad (29.3)$$

$$\bullet \llbracket \varphi \rightarrow (\forall \alpha. \varphi) \rrbracket \quad (29.4)$$

$$\bullet \text{SimpleExOrEnc } \psi \implies \llbracket \psi (\iota x. \varphi x) \rightarrow (\exists \alpha. \alpha^P = (\iota x. \varphi x)) \rrbracket \quad (29.5)$$

$$\bullet \text{SimpleExOrEnc } \psi \implies \llbracket \psi \tau \rightarrow (\exists \alpha. \alpha^P = \tau) \rrbracket \quad (29.5)$$

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

$$\bullet \llbracket (\forall \alpha. \varphi \alpha) \rightarrow ((\exists \beta. \beta = \tau) \rightarrow \varphi \tau) \rrbracket \quad (29.1)$$

$$\bullet \llbracket \exists \beta. \beta = \tau \rrbracket \quad (29.2)$$

$$\bullet \llbracket (\forall \alpha. \varphi \alpha \rightarrow \psi \alpha) \rightarrow ((\forall \alpha. \varphi \alpha) \rightarrow (\forall \alpha. \psi \alpha)) \rrbracket \quad (29.3)$$

$$\bullet \llbracket \varphi \rightarrow (\forall \alpha. \varphi) \rrbracket \quad (29.4)$$

$$\bullet \text{SimpleExOrEnc } \psi \implies \llbracket \psi (\iota x. \varphi x) \rightarrow (\exists \alpha. \alpha^P = (\iota x. \varphi x)) \rrbracket \quad (29.5)$$

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 - since a quantifier is used in the formulation, the problematic case of definite descriptions 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 reformulation is considered a reasonable compromise, since due to the type restrictions of the embedding the reformulated version of the axioms is *derivable* from the original version.

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

- *SimpleExOrEnc* ($\lambda x. \llbracket F, x \rrbracket$)
- *SimpleExOrEnc* ($\lambda x. \llbracket F, x, - \rrbracket$)
- *SimpleExOrEnc* ($\lambda x. \llbracket F, -, x \rrbracket$)
- *SimpleExOrEnc* ($\lambda x. \llbracket F, x, -, - \rrbracket$)
- *SimpleExOrEnc* ($\lambda x. \llbracket F, -, x, - \rrbracket$)
- *SimpleExOrEnc* ($\lambda x. \llbracket F, -, -, x \rrbracket$)
- *SimpleExOrEnc* ($\lambda x. \llbracket x, F \rrbracket$)

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):

$$\bullet [\mathcal{A}\varphi \equiv \varphi] \quad (30)$$

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:

$$\bullet \llbracket \mathcal{A}\neg\varphi \equiv \neg\mathcal{A}\varphi \rrbracket \quad (31.1)$$

$$\bullet \llbracket \mathcal{A}(\varphi \rightarrow \psi) \equiv (\mathcal{A}\varphi \rightarrow \mathcal{A}\psi) \rrbracket \quad (31.2)$$

$$\bullet \llbracket \mathcal{A}(\forall \alpha. \varphi \alpha) \equiv (\forall \alpha. \mathcal{A}\varphi \alpha) \rrbracket \quad (31.3)$$

$$\bullet \llbracket \mathcal{A}\varphi \equiv \mathcal{A}\mathcal{A}\varphi \rrbracket \quad (31.4)$$

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):

$$\bullet \llbracket \Box(\varphi \rightarrow \psi) \rightarrow (\Box\varphi \rightarrow \Box\psi) \rrbracket \quad (32.1)$$

$$\bullet \llbracket \Box\varphi \rightarrow \varphi \rrbracket \quad (32.2)$$

$$\bullet \llbracket \Diamond\varphi \rightarrow \Box\Diamond\varphi \rrbracket \quad (32.3)$$

$$\bullet \llbracket \Diamond(\exists x. (\mathcal{E}!, x^P)) \ \& \ \Diamond\neg(\mathcal{E}!, x^P) \ \& \ \Diamond\neg(\exists x. (\mathcal{E}!, x^P) \ \& \ \Diamond\neg(\mathcal{E}!, x^P)) \rrbracket \quad (32.4)$$

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 $\mathcal{E}!$. 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):

$$\bullet \llbracket \mathcal{A}\varphi \rightarrow \Box\mathcal{A}\varphi \rrbracket \quad (33.1)$$

$$\bullet \llbracket \Box\varphi \equiv \mathcal{A}(\Box\varphi) \rrbracket \quad (33.2)$$

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):

$$\bullet \llbracket x^P = (\iota x. \varphi x) \equiv (\forall z. \mathcal{A}\varphi z \equiv z = x) \rrbracket \quad (34)$$

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):

$$\bullet \lambda x. \varphi x = \lambda y. \varphi y \quad (36.1)$$

$$\bullet \text{IsProperInX } \varphi \implies [[\langle \lambda x. \varphi (x^P), x^P \rangle \equiv \varphi (x^P)]] \quad (36.2)$$

$$\bullet \text{IsProperInXY } \varphi \implies [[\langle \lambda^2 (\lambda x y. \varphi (x^P) (y^P)), x^P, y^P \rangle \equiv \varphi (x^P) (y^P)]] \quad (36.2)$$

$$\bullet \text{IsProperInXYZ } \varphi \implies [[\langle \lambda^3 (\lambda x y z. \varphi (x^P) (y^P) (z^P)), x^P, y^P, z^P \rangle \equiv \varphi (x^P) (y^P) (z^P)]] \quad (36.2)$$

$$\bullet [[\lambda^0 \varphi = \varphi]] \quad (36.3)$$

$$\bullet [[(\lambda x. \langle F, x^P \rangle) = F]] \quad (36.3)$$

$$\bullet [[\lambda^2 (\lambda x y. \langle F, x^P, y^P \rangle) = F]] \quad (36.3)$$

$$\bullet [[\lambda^3 (\lambda x y z. \langle F, x^P, y^P, z^P \rangle) = F]] \quad (36.3)$$

$$\bullet (\lambda x. [\mathcal{A}(\varphi x \equiv \psi x) \text{ in } v]) \implies [[\lambda^0 (\chi (\iota x. \varphi x)) = \lambda^0 (\chi (\iota x. \psi x))]] \quad (36.4)$$

$$\bullet (\lambda x. [\mathcal{A}(\varphi x \equiv \psi x) \text{ in } v]) \implies [[(\lambda x. \chi (\iota x. \varphi x) x) = (\lambda x. \chi (\iota x. \psi x) x)]] \quad (36.4)$$

$$\bullet (\lambda x. [\mathcal{A}(\varphi x \equiv \psi x) \text{ in } v]) \implies [[\lambda^2 (\chi (\iota x. \varphi x)) = \lambda^2 (\chi (\iota x. \psi x))]] \quad (36.4)$$

$$\bullet (\lambda x. [\mathcal{A}(\varphi x \equiv \psi x) \text{ in } v]) \implies [[\lambda^3 (\chi (\iota x. \varphi x)) = \lambda^3 (\chi (\iota x. \psi x))]] \quad (36.4)$$

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

As explained in section 3.4.7 β -conversion has to be restricted to *proper maps*. In PLM this restriction is implicit due to the fact that λ -expressions are only well-formed if their matrix is a propositional formula.

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 PLM:

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

$\chi^{*'}$ 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 \circ . Thereby substituting $\iota x \psi$ for occurrences of $\iota x \varphi$ can be expressed by comparing the function application of χ to $\iota x. \varphi x$ with the function application of χ to $\iota x. \psi x$.

Since in this representation φ and ψ are functions as well (from type ν to type \circ) 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):

$$\bullet \llbracket \{x, F\} \rightarrow \Box \{x, F\} \rrbracket \quad (37)$$

$$\bullet \llbracket \{O!, x\} \rightarrow \neg(\exists F. \{x, F\}) \rrbracket \quad (38)$$

$$\bullet \llbracket \exists x. \{A!, x^P\} \ \& \ (\forall F. \{x^P, F\} \equiv \varphi \ F) \rrbracket \quad (39)$$

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.

Furthermore in the functional setting φ has to be represented as 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 occurrence of x in φ . In the functional setting this is not necessary. Since x is bound by the existential quantifier and not explicitly given to φ as an argument, the condition φ imposes on F cannot depend on x by construction.

3.10.12. Summery

Although some of the axioms have to be adjusted to be representable in the functional environment, it is possible to arrive at a formulation that faithfully represents the original axiom system of PLM.

Furthermore a large part of the axioms can be derived independently of the technicalities of the representation layer with proofs that only depend 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 faithfully represents 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 or the semantics.

3.11. The Deductive System PLM

The derivation of the deductive system PLM ([12, Chap. 9]) from the 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.

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 theorems, that are derivable from the complete axiom system including the modally-fragile axiom, and the set of theorems, that have *modally-strict* proofs.

A proof is modally-strict, if it does not depend on any modally-fragile axiom.

In the embedding modally-strict theorems are stated to be true for an *arbitrary* semantic possible world: $[\varphi \text{ in } v]$

Here the variable v implicitly ranges over all semantic possible worlds 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.

Modally-fragile theorems on the other hand are stated to be true only for the designated actual world: $[\varphi \text{ in } dw]$

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

This representation of modally-strict and modally-fragile theorems 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):

$$\bullet [\varphi \text{ in } v] \wedge [\varphi \rightarrow \psi \text{ in } v] \Longrightarrow [\psi \text{ in } v] \quad (41)$$

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):

$$\bullet (\wedge \alpha. [\varphi \alpha \text{ in } v]) \Longrightarrow [\forall \alpha. \varphi \alpha \text{ in } v] \quad (49)$$

$$\bullet [\wedge w. [\varphi \text{ in } w] \Longrightarrow [\psi \text{ in } w]; [\Box \varphi \text{ in } v]] \Longrightarrow [\Box \psi \text{ in } v] \quad (51)$$

Although in PLM these rules can be derived by structural induction on the length of a derivation, 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):

$$\bullet ([\varphi \text{ in } v] \implies [\psi \text{ in } v]) \implies [\varphi \rightarrow \psi \text{ in } v] \quad (54)$$

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

These rules are the *only* exceptions to the concept that the deductive system of PLM is derived solely from the axiom system without relying on the previous layers of the embedding.

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, whenever an appropriate rule is derived as part of PLM directly or becomes trivially derivable from the proven theorems, it is added to the *PLM-solver*. Additionally the *PLM-solver* can instantiate any theorem of the deductive system PLM as well as any axiom, if doing so resolves the current proving goal.

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. In practice it can automatically prove a variety of simple statements and aid in more complex proofs throughout the derivation of the deductive system.

3.11.4. Additional Type Classes

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:

$$\bullet [\alpha = \alpha \text{ in } v] \quad (71.1)$$

$$\bullet [\alpha = \beta \rightarrow \beta = \alpha \text{ in } v] \quad (71.2)$$

$$\bullet [\alpha = \beta \ \& \ \beta = \gamma \rightarrow \alpha = \gamma \text{ in } v] \quad (71.3)$$

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 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 $\varphi \vdash \varphi'$]

A naive representation of the rule would be the following:

$$(\bigwedge v. [\psi \equiv \chi \text{ in } v]) \implies [\varphi \psi \text{ in } v] = [\varphi \chi \text{ in } v]$$

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

Formulas in the embedding have type \mathbf{o} which is internally represented by functions of the type $j \Rightarrow i \Rightarrow \text{bool}$. Therefore the formulation above could be instantiated with a function φ that has 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 $\bigwedge v. [\psi \equiv \chi \text{ in } v]$ on the other hand only requires ψ and χ to be (necessarily) equivalent 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 would not follow from the naive formulation above, even if it were derivable. Since x is *bound* by the existential quantifier, in the functional representation φ has to have a different type. In the example φ has to be $\lambda\psi. \exists x. \psi \ x$ which is of type $(\nu \Rightarrow \mathbf{o}) \Rightarrow \mathbf{o}$. ψ and χ have to be functions as well: $\psi = (\lambda x. (A!, x))$ and $\chi = (\lambda x. \neg\Diamond(E!, x))$. Consequently the equivalence condition for this case has to be reformulated to $\bigwedge x \ v. [\psi \ x \equiv \chi \ x \text{ in } v]$ ⁶.

Solution

The embedding employs a solution that is complex, but can successfully address the described issues.

The following definition is introduced (see A.9.10):

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

Given a condition *cond* a function φ is considered *Substable*, if and only if for all ψ and χ that satisfy *cond* it follows in each possible world v that $[\varphi \psi \equiv \varphi \chi \text{ in } v]$ ⁷.

⁶This 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.

⁷ ψ and χ can have an arbitrary type. φ is a function from this type to formulas.

Now several introduction rules for this property are derived. The idea is to capture the notion of *subformula* in PLM. A few examples are:

- *Substable cond* $(\lambda\varphi. \Theta)$
- *Substable cond* $\psi \implies \text{Substable cond } (\lambda\varphi. \neg\psi \varphi)$
- *Substable cond* $\psi \wedge \text{Substable cond } \chi \implies \text{Substable cond } (\lambda\varphi. \psi \varphi \rightarrow \chi \varphi)$

These rules can be derived using theorems of PLM.

As illustrated 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:

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

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 $[\varphi \psi \text{ in } v]$ is also true for $[\varphi \chi \text{ in } v]$.

As a base case this type class is *instantiated* for the type of formulas `o` with the following definition of *Substable-Cond*:

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

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*:

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

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. 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 $[\neg\neg\Diamond([E!,x]) \text{ in } v]$. Now it is possible to apply *PLM-subst-method* as follows:

$$\mathbf{apply} \ (PLM\text{-}subst\text{-}method \ ([A!,x]) \ (\neg(\Diamond([E!,x])))$$

The method automatically analyzes the current proving goal, uses pattern matching to find an appropriate choice for a function φ , applies the substitution rule and resolves the substitutability claim about φ .

Consequently it can resolve the current proof objective by producing two new proving goals: $\forall v. [\langle A!, x \rangle \equiv \neg \Diamond \langle E!, x \rangle \text{ in } v]$ and $[\neg \langle A!, x \rangle \text{ in } v]$, as expected. The complexity of the construction above is hidden away entirely.

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

apply (*PLM-subst-method* $\lambda x . \langle A!, x^P \rangle \lambda x . (\neg(\Diamond \langle E!, x^P \rangle))$)

This will result in the new proving goals: $\forall x v. [\langle A!, x^P \rangle \equiv \neg \Diamond \langle E!, x^P \rangle \text{ in } v]$ and $[\exists x. \langle A!, x^P \rangle \text{ in } v]$, 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 example⁸:

```

lemma  $[\Box(\varphi \rightarrow \Box\varphi) \rightarrow ((\neg\Box\varphi) \equiv (\Box(\neg\varphi))) \text{ in } v]$ 
proof (rule CP)
  assume  $[\Box(\varphi \rightarrow \Box\varphi) \text{ in } v]$ 
  hence  $[(\neg\Box(\neg\varphi)) \equiv \Box\varphi \text{ in } v]$ 
    by (metis sc-eq-box-box-1 diamond-def vdash-properties-10)
  thus  $[(\neg\Box\varphi) \equiv (\Box(\neg\varphi))] \text{ in } v$ 
    by (meson CP  $\equiv I \equiv E \neg I \neg E$ )
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 $[\Box(\varphi \rightarrow \Box\varphi) \text{ in } v] \implies [\neg\Box\varphi \equiv \Box\neg\varphi \text{ in } v]$, so $[\neg\Box\varphi \equiv \Box\neg\varphi \text{ in } v]$ has to be shown under the assumption $[\Box(\varphi \rightarrow \Box\varphi) \text{ in } v]$. Therefore the first step is to assume $[\Box(\varphi \rightarrow \Box\varphi) \text{ in } v]$.

⁸Since the whole proof is stated as raw Isabelle code, unfortunately no color-coding can be applied.

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.

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 one-line proof:

lemma $[\Box(\varphi \rightarrow \Box\varphi) \rightarrow ((\neg\Box\varphi) \equiv (\Box(\neg\varphi)))]$ *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.

3.11.7. Summery

A full representation of the deductive system PLM, as described in [12, Chap. 9], could be derived 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.

3.12. Artificial Theorems

The layered approach of the embedding provides the means to derive theorems independently of the representation layer and model structure. It is still 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):

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

In this case traditional β -conversion still holds, since the λ -expression does not contain encoding expressions involving its bound variable⁹. On the other hand the following is *not* a theorem in the embedding (the tool **nitpick** can find a counter-model):

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

⁹Consequently the matrix is a *proper map*.

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

- $[(\lambda x. \llbracket x^P, F \rrbracket, z^P) \text{ in } v] = (\exists y. \nu\nu y = \nu\nu z \wedge [\llbracket y^P, F \rrbracket \text{ in } v])$
- $[(\lambda x. \varphi(x^P), z^P) \text{ in } v] = (\exists y. \nu\nu y = \nu\nu z \wedge [\varphi(y^P) \text{ in } v])$

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

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

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, which is due to the following theorem¹⁰:

$$IsProperInX \ \varphi \implies [(\exists y. (\forall F. \llbracket F, x^P \rrbracket \equiv \llbracket F, y^P \rrbracket) \ \& \ \varphi(y^P)) \equiv \varphi(x^P) \text{ in } v]$$

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.:

$$[(\lambda x. \llbracket x^P, F \rrbracket \equiv \llbracket x^P, F \rrbracket, y^P) \text{ in } v]$$

This topic is discussed in more detail in section 5.1.1.

3.12.2. Consequences of the Aczel-model

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

$$[\forall F. \llbracket F, a^P \rrbracket \equiv \llbracket F, b^P \rrbracket \text{ in } v] \implies \lambda x. \llbracket R, x^P, a^P \rrbracket = \lambda x. \llbracket R, x^P, b^P \rrbracket$$

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 states. 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 follows.

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.

¹⁰Note that for propositional formulas an equivalent statement is derivable in PLM as well.

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):

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

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

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

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

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

4. Technical Limitations of Isabelle/HOL

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 $([\varphi \text{ in } v] \Rightarrow [\psi \text{ in } v]) \Rightarrow [\varphi \rightarrow \psi \text{ in } v]$, 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, there remain some 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 and will only be possible after PLM has recovered from the discovered paradox (see 5.2). Such an analysis will be 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 analyze 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 distinction 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* semantic 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 is that every λ -expression that is syntactically well-formed in PLM should 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 suggests that the theory of PLM can be consistently extended to include a larger set of proper, β -convertible λ -expressions. Since the set of relations of PLM already has to be adjusted to prevent the discovered paradox, such an extension presents a viable option.

Once PLM has recovered from the paradox, future research can consider available options to align the set of relations present in the embedding with the resulting set of relations of the new version of PLM.

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 individuals and individual terms have different *types*.

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 using the decoration $_P$, 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. Therefore 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. An alternative approach that could result in a more elegant representation would be to implement concepts of free logic based on the research in [4] for both individuals and relations.

5.1.3. Modally-strict Proofs and the Converse of RN

As described in section 3.11.1 modally-strict theorems in the embedding are stated in the form $[\varphi \text{ in } v]$ for arbitrary v . However, the set of modally-strict theorems in PLM corresponds to only a subset of the theorems that are semantically true in arbitrary (semantic) possible worlds.

Modally-strict theorems in PLM are defined using a proof-theoretic concept: modally-strict proofs are not allowed to use modally-fragile axioms. They are solely derived from axioms whose necessitations are axioms as well (see 3.10.1).

The metarule RN states in essence that if there is a modally-strict proof for φ , then $\Box\varphi$ is derivable as a theorem. PLM proves this fact by induction on the length of the derivation. However, remark (185)[12] gives an example of a case in which the converse is false: if $\Box\varphi$ is derivable as a theorem, this does not imply that there is a modally-strict proof for φ .

However, in the embedding the following is derivable from the semantics of the box operator:

$$[\Box\varphi \text{ in } dw] \implies \forall v. [\varphi \text{ in } v]$$

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

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

Unfortunately there is no semantic criterion that allows to decide whether a statement is a necessary theorem or a modally-strict theorem. Therefore the embedding has to express modally-strict theorems as necessary theorems, for which the converse of RN is in fact true.

This still does not compromise the claim that any statement that is derived in A.9 is also derivable in PLM: the basis for this claim is that no proofs in this layer may rely on the meta-logic, but only the fundamental meta-rules of PLM are allowed to derive theorems from the axioms. Since the converse of RN is neither a fundamental meta-rule of PLM,

nor derivable without using the semantics, it is not stated as an admissible rule for these proofs. Thereby it is guaranteed that no statement of the form $[\varphi \text{ in } v]$ is derived that is not a modally-strict theorem of PLM.

Unfortunately this has the consequence that the proving method *PLM-solver* cannot be equipped with a reversible elimination rule for the box operator, which reduces its power as a proving method. However, preserving the claim that theorems derived in the embedding are also theorems of PLM even when restricting to modally-strict theorems was given preference over an increased level of automation.

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 its 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 in PLM, which are the only formulas PLM allows as the matrix of λ -expressions (see 5.1.1).

While verifying the conjecture, 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 $[\lambda y \text{ F}\iota x(y[\lambda z \text{ R}xz])]$ 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.

Further analysis showed that a modification of the embedding that would allow β -conversion for such expressions, would have to involve a restriction of the Aczel-model (in particular of the map from abstract objects to urelements).

In order to understand how the Aczel-model could be adequately restricted, the consequences of allowing β -conversion in the mentioned cases *by assumption* were studied in the embedding. This led to the first proof of inconsistency (see A.13.4):

$$(\bigwedge G \varphi. \text{IsProperInX } (\lambda x. \llbracket G, \iota y. \varphi \ y \ x \rrbracket)) \implies \text{False}$$

Under the assumption that $\lambda x. \langle G, \iota y. \varphi y x \rangle$ 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 inconsistency can be derived without relying on the meta-logic, it was immediately possible to translate the proof back to the language of PLM. The resulting formulation 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 described in section 3.2 can be reconstructed. This refined version of the paradox is used in the inconsistency proof in A.13.3 and is outlined in the language of PLM in the next section. The general situation leading to the paradox is repeated 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 excluding non-propositional formulas in λ -expressions would be sufficient to prevent such inconsistencies. This was shown to be incorrect, though.

The problem is the *description backdoor*. The term $[\lambda y F \iota x \psi]$ is well-formed, even if ψ is *not* propositional. This is due to the definition of *subformula*: ψ is *not* a subformula of $F \iota x \psi$, so ψ *may* contain encoding subformulas itself and $F \iota x \psi$ is still a propositional formula.

This was deemed to be no problem and for cases like $[\lambda y F \iota x (xG)]$ as they are mentioned and used 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 expression, for example $[\lambda y F \iota x (xG \ \& \ yG)]$ is still a propositional formula.

Therefore the following construction is possible:

$$[\lambda y [\lambda z \forall p(p \rightarrow p)] \iota x (x = y \ \& \ \psi)] \quad (1)$$

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.

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

$$[\lambda y [\lambda z \forall p(p \rightarrow p)] \iota x (x = y \ \& \ \psi)] x \equiv \psi^x_y \quad (2)$$

Remark. Using a modally-strict proof only the following is derivable:

$$[\lambda y [\lambda z \forall p(p \rightarrow p)] \iota x(x = y \ \& \ \psi)]x \equiv \mathcal{A}\psi^x_y$$

For the construction of the paradox, the modally-fragile statement is sufficient. However, it is possible to construct similar paradoxical cases without appealing to any modally-fragile axioms or theorems as well.

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:

$$[\lambda y [\lambda z \forall p(p \rightarrow p)] \iota x(x = y \ \& \ (\exists F yF \ \& \ \neg Fy))] \quad (3)$$

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 use 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, will 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 analyzed 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 analyze the relationship between syntactic and semantic possible worlds and arrive at the following theorems (see A.10):

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

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):

$$\bullet \Diamond p \equiv \exists w(w \models p) \quad (433.1)$$

$$\bullet \Box p \equiv \forall w(w \models p) \quad (433.2)$$

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.

As a result a theory was constructed in Isabelle/HOL that - although its faithfulness 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 faithfulness 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 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

¹This formulation originates in the resulting e-mail correspondence between Bruno Woltzenlogel Paleo and Christoph Benzml ller.

could be reproduced in the target logic, strongly indicates the merits and success of this approach.

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 its 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 that relational type theory is more fundamental than functional type theory (see [8]). One of their main arguments is that the Theory of Abstract Objects is not representable in functional type theory. The success of the presented embedding, however, suggests that the topic has to be examined more closely.

Their result is 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 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 of translating relational type theory to functional type theory in fact fails for the Theory of Abstract Object. A simple version of functional type theory, that only involves two primitive types (for individuals and propositions), is insufficient for a representation of the theory.

Consequently 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 propositions.
- 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 \circ be the type of propositions. The general construct is now the following:

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

Not all functions of type $\iota \Rightarrow \dots \Rightarrow \iota \Rightarrow o$ are supposed to denote relations. However, in the proposed construction a concept used in the embedding of free logic can help². The function Λ_n can map functions of type $\iota \Rightarrow \dots \Rightarrow \iota \Rightarrow o$ that do not correspond to propositional formulas to objects of type R_n that represent invalid (resp. non-existing) relations. For invalid relations the functions used to represent encoding and exemplification can be defined to map to an object of type o that represents invalid propositions.

Oppenheimer and Zalta argue 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 formulas³. Although this is true for the case of a simple functional type theory, it does not apply to the constructed theory: since only objects of type R_n may fail to denote, non-propositional reasoning is unaffected.

Furthermore the embedding has shown that an intensional interpretation of the constructed theory can preserve the hyperintensionality of relations in λ -expressions.

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). Although this concept has merits as well, future versions of the embedding may instead 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 summary 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 intensional and 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 relations⁴ 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 and its success clearly shows the merits of the approach.

²See the embedding of free logic constructed in [4].

³See [8, pp. 30-31].

⁴Note, however, that the embedding can represent relations as functions acting on urelements following the Aczel-model.

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.

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` $v = \omega v \ \omega \mid \sigma v \ \sigma$ — urelements

A.1.2. Derived Types

`typedef` $o = UNIV :: (j \Rightarrow i \Rightarrow bool)$ *set*

`morphisms` *eval* *make* o .. — truth values

`type-synonym` $\Pi_0 = o$ — zero place relations

`typedef` $\Pi_1 = UNIV :: (v \Rightarrow j \Rightarrow i \Rightarrow bool)$ *set*

`morphisms` *eval* Π_1 *make* Π_1 .. — one place relations

`typedef` $\Pi_2 = UNIV :: (v \Rightarrow v \Rightarrow j \Rightarrow i \Rightarrow bool)$ *set*

`morphisms` *eval* Π_2 *make* Π_2 .. — two place relations

`typedef` $\Pi_3 = UNIV :: (v \Rightarrow v \Rightarrow v \Rightarrow j \Rightarrow i \Rightarrow bool)$ *set*

`morphisms` *eval* Π_3 *make* Π_3 .. — three place relations

`type-synonym` $\alpha = \Pi_1$ *set* — abstract objects

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

`typedef` $\kappa = UNIV :: (\nu \text{ option})$ *set*

`morphisms` *eval* κ *make* κ .. — individual terms

`setup-lifting` *type-definition-o*

`setup-lifting` *type-definition- κ*

`setup-lifting` *type-definition- Π_1*

`setup-lifting` *type-definition- Π_2*

`setup-lifting` *type-definition- Π_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 $\nu\kappa :: \nu \Rightarrow \kappa \text{ } (-^P \text{ } [90] \text{ } 90)$ **is** *Some* .
lift-definition *proper* $:: \kappa \Rightarrow \text{bool}$ **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**
 $\lambda \varphi . \text{if } (\exists! x . (\varphi x) \text{ } dj \text{ } dw)$
 $\text{then } \text{Some } (THE x . (\varphi x) \text{ } dj \text{ } dw)$
 $\text{else } \text{None} .$

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 $\alpha\sigma\text{-surj} :: \text{surj } \alpha\sigma$
definition $\nu\nu :: \nu \Rightarrow \nu$ **where** $\nu\nu \equiv \text{case-}\nu \text{ } \omega\nu \text{ } (\sigma\nu \circ \alpha\sigma)$

A.1.5. Exemplification of n-place relations.

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

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

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 $I\text{-}NOT :: j \Rightarrow (i \Rightarrow \text{bool}) \Rightarrow i \Rightarrow \text{bool}$
consts $I\text{-}IMPL :: j \Rightarrow (i \Rightarrow \text{bool}) \Rightarrow (i \Rightarrow \text{bool}) \Rightarrow (i \Rightarrow \text{bool})$

lift-definition $\text{not} :: o \Rightarrow o \ (\neg - [54] \ 70)$ **is**
 $\lambda p \ s \ w . s = dj \wedge \neg p \ dj \ w \vee s \neq dj \wedge (I\text{-}NOT \ s \ (p \ s) \ w) .$

lift-definition $\text{impl} :: o \Rightarrow o \Rightarrow o \ (\text{infixl} \rightarrow 51)$ **is**
 $\lambda p \ q \ s \ w . s = dj \wedge (p \ dj \ w \longrightarrow q \ dj \ w) \vee s \neq dj \wedge (I\text{-}IMPL \ s \ (p \ s) \ (q \ s) \ w) .$

lift-definition $\text{forall}_\nu :: (\nu \Rightarrow o) \Rightarrow o \ (\text{binder} \ \forall_\nu \ [8] \ 9)$ **is**
 $\lambda \varphi \ s \ w . \forall x :: \nu . (\varphi \ x) \ s \ w .$

lift-definition $\text{forall}_0 :: (\Pi_0 \Rightarrow o) \Rightarrow o \ (\text{binder} \ \forall_0 \ [8] \ 9)$ **is**
 $\lambda \varphi \ s \ w . \forall x :: \Pi_0 . (\varphi \ x) \ s \ w .$

lift-definition $\text{forall}_1 :: (\Pi_1 \Rightarrow o) \Rightarrow o \ (\text{binder} \ \forall_1 \ [8] \ 9)$ **is**
 $\lambda \varphi \ s \ w . \forall x :: \Pi_1 . (\varphi \ x) \ s \ w .$

lift-definition $\text{forall}_2 :: (\Pi_2 \Rightarrow o) \Rightarrow o \ (\text{binder} \ \forall_2 \ [8] \ 9)$ **is**
 $\lambda \varphi \ s \ w . \forall x :: \Pi_2 . (\varphi \ x) \ s \ w .$

lift-definition $\text{forall}_3 :: (\Pi_3 \Rightarrow o) \Rightarrow o \ (\text{binder} \ \forall_3 \ [8] \ 9)$ **is**
 $\lambda \varphi \ s \ w . \forall x :: \Pi_3 . (\varphi \ x) \ s \ w .$

lift-definition $\text{forall}_o :: (o \Rightarrow o) \Rightarrow o \ (\text{binder} \ \forall_o \ [8] \ 9)$ **is**
 $\lambda \varphi \ s \ w . \forall x :: o . (\varphi \ x) \ s \ w .$

lift-definition $\text{box} :: o \Rightarrow o \ (\Box - [62] \ 63)$ **is**
 $\lambda p \ s \ w . \forall v . p \ s \ v .$

lift-definition $\text{actual} :: o \Rightarrow o \ (\mathcal{A} - [64] \ 65)$ **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 $\text{lambdabinder0} :: o \Rightarrow \Pi_0 \ (\lambda^0)$ **is** $\text{id} .$

lift-definition $\text{lambdabinder1} :: (\nu \Rightarrow o) \Rightarrow \Pi_1 \ (\text{binder} \ \lambda \ [8] \ 9)$ **is**
 $\lambda \varphi \ u \ s \ w . \exists x . \nu v \ x = u \wedge \varphi \ x \ s \ w .$

lift-definition $\text{lambdabinder2} :: (\nu \Rightarrow \nu \Rightarrow o) \Rightarrow \Pi_2 \ (\lambda^2)$ **is**
 $\lambda \varphi \ u \ v \ s \ w . \exists x \ y . \nu v \ x = u \wedge \nu v \ y = v \wedge \varphi \ x \ y \ s \ w .$

lift-definition $\text{lambdabinder3} :: (\nu \Rightarrow \nu \Rightarrow \nu \Rightarrow o) \Rightarrow \Pi_3 \ (\lambda^3)$ **is**
 $\lambda \varphi \ u \ v \ r \ s \ w . \exists x \ y \ z . \nu v \ x = u \wedge \nu v \ y = v \wedge \nu v \ z = r \wedge \varphi \ x \ y \ z \ s \ w .$

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 $\text{IsProperInX} :: (\kappa \Rightarrow o) \Rightarrow \text{bool}$ **is**

$\lambda \varphi . \forall x \ v . (\exists a . \nu v \ a = \nu v \ x \wedge (\varphi \ (a^P) \ dj \ v)) = (\varphi \ (x^P) \ dj \ v) .$

lift-definition $\text{IsProperInXY} :: (\kappa \Rightarrow \kappa \Rightarrow o) \Rightarrow \text{bool}$ **is**

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

A.1.10. Validity

lift-definition *valid-in* :: $i \Rightarrow o \Rightarrow bool$ (**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**
 $PossiblyContingentObjectExists \equiv \exists x v . ConcreteInWorld x v$
 $\quad \wedge (\exists w . \neg ConcreteInWorld x w)$

abbreviation (*input*) *PossiblyNoContingentObjectExists* **where**
 $PossiblyNoContingentObjectExists \equiv \exists w . \forall x . ConcreteInWorld x w$
 $\quad \longrightarrow (\forall v . ConcreteInWorld x v)$

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:: $\Pi_1 (E!)$* **is**
 $\lambda u s w . case u of \omega v x \Rightarrow ConcreteInWorld x w \mid - \Rightarrow False .$

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] forallν-def[meta-defs]
  forall0-def[meta-defs] forall1-def[meta-defs]
  forall2-def[meta-defs] forall3-def[meta-defs] forallo-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-ax*.

named-theorems *meta-ax*

```

declare makeκ-inverse[meta-ax] evalκ-inverse[meta-ax]
  makeo-inverse[meta-ax] evalo-inverse[meta-ax]
  makeΠ1-inverse[meta-ax] evalΠ1-inverse[meta-ax]
  makeΠ2-inverse[meta-ax] evalΠ2-inverse[meta-ax]
  makeΠ3-inverse[meta-ax] evalΠ3-inverse[meta-ax]
lemma νν-ων-is-ων[meta-ax]: νν (ων x) = ων x by (simp add: νν-def)
lemma rep-proper-id[meta-ax]: rep (xP) = x
  by (simp add: meta-ax νκ-def rep-def)
lemma νκ-proper[meta-ax]: proper (xP)
  by (simp add: meta-ax νκ-def proper-def)
lemma no-αω[meta-ax]: ¬(νν (αν x) = ων y) by (simp add: νν-def)
lemma no-σω[meta-ax]: ¬(σν x = ων y) by blast
lemma νν-surj[meta-ax]: surj νν
  using ασ-surj unfolding νν-def surj-def
  by (metis ν.simps(5) ν.simps(6) v.exhaust comp-apply)
lemma lambdaΠ1-aux[meta-ax]:
  makeΠ1 (λu s w. ∃x. νν x = u ∧ evalΠ1 F (νν x) s w) = F
proof -
  have ∧ u s w φ . (∃ x . νν x = u ∧ φ (νν x) (s::j) (w::i)) ⟷ φ u s w
  using νν-surj unfolding surj-def by metis
  thus ?thesis apply transfer by simp
qed
lemma lambdaΠ2-aux[meta-ax]:
  makeΠ2 (λu v s w. ∃x . νν x = u ∧ (∃ y . νν y = v ∧ evalΠ2 F (νν x) (νν y) s w)) = F
proof -
  have ∧ u v (s::j) (w::i) φ .
    (∃ x . νν x = u ∧ (∃ y . νν y = v ∧ φ (νν x) (νν y) s w))
    ⟷ φ u v s w
  using νν-surj unfolding surj-def by metis
  thus ?thesis apply transfer by simp
qed
lemma lambdaΠ3-aux[meta-ax]:
  makeΠ3 (λu v r s w. ∃x. νν x = u ∧ (∃y. νν y = v ∧
    (∃z. νν z = r ∧ evalΠ3 F (νν x) (νν y) (νν z) s w))) = F
proof -
  have ∧ u v r (s::j) (w::i) φ . ∃x. νν x = u ∧ (∃y. νν y = v

```

$\wedge (\exists z. \nu\nu z = r \wedge \varphi (\nu\nu x) (\nu\nu y) (\nu\nu z) s w)) = \varphi u v r s w$
using *$\nu\nu$ -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 $R_0 = j \Rightarrow i \Rightarrow \text{bool}$
type-synonym $R_1 = v \Rightarrow R_0$
type-synonym $R_2 = v \Rightarrow v \Rightarrow R_0$
type-synonym $R_3 = v \Rightarrow v \Rightarrow v \Rightarrow R_0$
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 $ex0 :: R_0 \Rightarrow W \Rightarrow \text{bool}$
 where $ex0 \equiv \lambda F . F \ dj$
definition $ex1 :: R_1 \Rightarrow W \Rightarrow (R_\kappa \text{ set})$
 where $ex1 \equiv \lambda F w . \{ x . F (\nu\nu x) \ dj w \}$
definition $ex2 :: R_2 \Rightarrow W \Rightarrow ((R_\kappa \times R_\kappa) \text{ set})$
 where $ex2 \equiv \lambda F w . \{ (x,y) . F (\nu\nu x) (\nu\nu y) \ dj w \}$
definition $ex3 :: R_3 \Rightarrow W \Rightarrow ((R_\kappa \times R_\kappa \times R_\kappa) \text{ set})$
 where $ex3 \equiv \lambda F w . \{ (x,y,z) . F (\nu\nu x) (\nu\nu y) (\nu\nu z) \ dj w \}$

A.2.1.5. Encoding Extensions

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

A.2.1.6. Collection of Semantical Definitions

named-theorems *semantics-defs*
declare $d_0\text{-def}[semantics-defs]$ $d_1\text{-def}[semantics-defs]$
 $d_2\text{-def}[semantics-defs]$ $d_3\text{-def}[semantics-defs]$
 $ex0\text{-def}[semantics-defs]$ $ex1\text{-def}[semantics-defs]$
 $ex2\text{-def}[semantics-defs]$ $ex3\text{-def}[semantics-defs]$
 $en\text{-def}[semantics-defs]$ $d_\kappa\text{-def}[semantics-defs]$
 $w_0\text{-def}[semantics-defs]$

A.2.1.7. Truth Conditions of Exemplification Formulas

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

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

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

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

A.2.1.8. Truth Conditions of Encoding Formulas

lemma $T2[semantics]$:
 $(w \models \langle x, F \rangle) = (\exists r \ o_1 . \text{Some } r = d_1 F \wedge \text{Some } o_1 = d_\kappa x \wedge o_1 \in en \ r)$
unfolding *semantics-defs*
apply (*simp add: meta-defs meta-aux rep-def proper-def split: v.split*)
by (*metis v.exhaust v.inject(2) v.simps(4) v.k.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 $T7[semantics]$: $(w \models \mathcal{A}\psi) = (dw \models \psi)$
by (*simp add: meta-defs meta-aux*)

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

A.2.1.10. Denotations of Descriptions

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

A.2.1.11. Denotations of Lambda Expressions

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

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

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

lemma $D5-1[semantics]$:
assumes *IsProperInX* φ
shows $\bigwedge w o_1 r. Some\ r = d_1 (\lambda x. (\varphi (x^P))) \wedge Some\ o_1 = d_\kappa x$
 $\longrightarrow (o_1 \in ex1\ r\ w) = (w \models \varphi\ x)$
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 $\bigwedge w o_1 o_2 r. Some\ r = d_2 (\lambda^2 (\lambda x y. \varphi (x^P) (y^P)))$
 $\wedge Some\ o_1 = d_\kappa x \wedge Some\ o_2 = d_\kappa y$
 $\longrightarrow ((o_1, o_2) \in ex2\ r\ w) = (w \models \varphi\ x\ y)$
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 $\bigwedge w \ o_1 \ o_2 \ o_3 \ r . \text{Some } r = d_3 (\lambda^3 (\lambda x \ y \ z . \varphi (x^P) (y^P) (z^P)))$
 $\wedge \text{Some } o_1 = d_\kappa x \wedge \text{Some } o_2 = d_\kappa y \wedge \text{Some } o_3 = d_\kappa z$
 $\longrightarrow ((o_1, o_2, o_3) \in \text{ex3 } r \ w) = (w \models \varphi \ x \ y \ z)$
 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]*: $(\bigwedge w \ r . \text{Some } r = d_0 (\lambda^0 \varphi) \longrightarrow \text{ex0 } r \ w = (w \models \varphi))$
 by (*auto simp: meta-defs meta-aux semantics-defs*)

A.2.1.12. Auxiliary Lemmata

lemma *propex0*: $\exists r . \text{Some } r = d_0 F$
unfolding *d0-def* **by** *simp*
 lemma *propex1*: $\exists r . \text{Some } r = d_1 F$
unfolding *d1-def* **by** *simp*
 lemma *propex2*: $\exists r . \text{Some } r = d_2 F$
unfolding *d2-def* **by** *simp*
 lemma *propex3*: $\exists r . \text{Some } r = d_3 F$
unfolding *d3-def* **by** *simp*
 lemma *d κ -proper*: $d_\kappa (u^P) = \text{Some } u$
unfolding *d κ -def* **by** (*simp add: $\nu\kappa$ -def meta-aux*)
 lemma *ConcretenessSemantics1*:
 $\text{Some } r = d_1 E! \implies (\exists w . \omega \nu \ x \in \text{ex1 } r \ w)$
unfolding *semantics-defs* **apply** *transfer*
 by (*simp add: OrdinaryObjectsPossiblyConcreteAxiom $\nu\nu$ - $\omega\nu$ -is- $\omega\nu$*)
 lemma *ConcretenessSemantics2*:
 $\text{Some } r = d_1 E! \implies (x \in \text{ex1 } r \ w \longrightarrow (\exists y . x = \omega \nu \ y))$
unfolding *semantics-defs* **apply** *transfer* **apply** *simp*
 by (*metis ν .exhaust ν .exhaust ν .simps(6) no- $\alpha\omega$*)
 lemma *d0-inject*: $\bigwedge x \ y . d_0 x = d_0 y \implies x = y$
unfolding *d0-def* **by** (*simp add: eval0-inject*)
 lemma *d1-inject*: $\bigwedge x \ y . d_1 x = d_1 y \implies x = y$
unfolding *d1-def* **by** (*simp add: eval Π_1 -inject*)
 lemma *d2-inject*: $\bigwedge x \ y . d_2 x = d_2 y \implies x = y$
unfolding *d2-def* **by** (*simp add: eval Π_2 -inject*)
 lemma *d3-inject*: $\bigwedge x \ y . d_3 x = d_3 y \implies x = y$
unfolding *d3-def* **by** (*simp add: eval Π_3 -inject*)
 lemma *d κ -inject*: $\bigwedge x \ y \ o_1 . \text{Some } o_1 = d_\kappa x \wedge \text{Some } o_1 = d_\kappa y \implies x = y$
proof –
 fix $x :: \kappa$ and $y :: \kappa$ and $o_1 :: \nu$
 assume $\text{Some } o_1 = d_\kappa x \wedge \text{Some } o_1 = d_\kappa y$
 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*]:
 $\text{IsProperInX } (\lambda x . \chi$
 (* one place *) $(\lambda F . \langle F, x \rangle)$
 (* two place *) $(\lambda F . \langle F, x, x \rangle) (\lambda F \ a . \langle F, x, a \rangle) (\lambda F \ a . \langle F, a, x \rangle)$
 (* three place three x *) $(\lambda F . \langle F, x, x, x \rangle)$

(* 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 *)

```

(* one place *) (λ F . (⟦F,z⟧))
(* two place *) (λ F . (⟦F,z,z⟧)) (λ F a . (⟦F,z,a⟧)) (λ F a . (⟦F,a,z⟧))
(* three place three z *) (λ F . (⟦F,z,z,z⟧))
(* three place two z *) (λ F a . (⟦F,z,z,a⟧)) (λ F a . (⟦F,z,a,z⟧))
                      (λ F a . (⟦F,a,z,z⟧))
(* three place one z *) (λ F a b . (⟦F,z,a,b⟧)) (λ F a b . (⟦F,a,z,b⟧))
                      (λ F a b . (⟦F,a,b,z⟧))

(* 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⟧))

(* x and z *)
(* two place *) (λ F . (⟦F,x,z⟧)) (λ F . (⟦F,z,x⟧))
(* three place (x,z) *) (λ F a . (⟦F,x,z,a⟧)) (λ F a . (⟦F,x,a,z⟧))
                      (λ F a . (⟦F,a,x,z⟧))
(* three place (z,x) *) (λ F a . (⟦F,z,x,a⟧)) (λ F a . (⟦F,z,a,x⟧))
                      (λ F a . (⟦F,a,z,x⟧))
(* three place (x,x,z) *) (λ F . (⟦F,x,x,z⟧)) (λ F . (⟦F,x,z,x⟧))
                      (λ F . (⟦F,z,x,x⟧))
(* three place (x,z,z) *) (λ F . (⟦F,x,z,z⟧)) (λ F . (⟦F,z,x,z⟧))
                      (λ F . (⟦F,z,z,x⟧))
(* three place (x,x,x) *) (λ F . (⟦F,x,x,x⟧))
(* three place (z,z,z) *) (λ F . (⟦F,z,z,z⟧))

(* y and z *)
(* two place *) (λ F . (⟦F,y,z⟧)) (λ F . (⟦F,z,y⟧))
(* three place (y,z) *) (λ F a . (⟦F,y,z,a⟧)) (λ F a . (⟦F,y,a,z⟧))
                      (λ F a . (⟦F,a,y,z⟧))
(* three place (z,y) *) (λ F a . (⟦F,z,y,a⟧)) (λ F a . (⟦F,z,a,y⟧))
                      (λ F a . (⟦F,a,z,y⟧))
(* three place (y,y,z) *) (λ F . (⟦F,y,y,z⟧)) (λ F . (⟦F,y,z,y⟧))
                      (λ F . (⟦F,z,y,y⟧))
(* three place (y,z,z) *) (λ F . (⟦F,y,z,z⟧)) (λ F . (⟦F,z,y,z⟧))
                      (λ F . (⟦F,z,z,y⟧))
(* three place (y,y,y) *) (λ F . (⟦F,y,y,y⟧))
(* three place (z,z,z) *) (λ F . (⟦F,z,z,z⟧))

(* x y z *)
(* three place (x,...) *) (λ F . (⟦F,x,y,z⟧)) (λ F . (⟦F,x,z,y⟧))
(* three place (y,...) *) (λ F . (⟦F,y,x,z⟧)) (λ F . (⟦F,y,z,x⟧))
(* three place (z,...) *) (λ F . (⟦F,z,x,y⟧)) (λ F . (⟦F,z,y,x⟧))

```

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* :: $\text{o} \Rightarrow i \Rightarrow \text{bool}$ ($[- \text{ in } -] [1]$) **where**
validity-in $\equiv \lambda \varphi v . v \models \varphi$
definition *actual-validity* :: $\text{o} \Rightarrow \text{bool}$ ($[-] [1]$) **where**
actual-validity $\equiv \lambda \varphi . dw \models \varphi$
definition *necessary-validity* :: $\text{o} \Rightarrow \text{bool}$ ($\Box[-] [1]$) **where**
necessary-validity $\equiv \lambda \varphi . \forall v . (v \models \varphi)$

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  $\Rightarrow$  o)  $\Rightarrow$  o (binder  $\forall$  [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  $\nu$  :: quantifiable
begin
  definition forall- $\nu$  :: ( $\nu \Rightarrow$  o)  $\Rightarrow$  o where forall- $\nu \equiv$  forall_ $\nu$ 
  instance proof
    fix w :: i and  $\psi$  ::  $\nu \Rightarrow$  o
    show ( $w \models \forall x . \psi x$ ) = ( $\forall x . (w \models \psi x)$ )
      unfolding forall- $\nu$ -def using Semantics.T8- $\nu$  .
    qed
  end
```

```
instantiation o :: quantifiable
begin
  definition forall-o :: (o  $\Rightarrow$  o)  $\Rightarrow$  o where forall-o  $\equiv$  forall_o
  instance proof
    fix w :: i and  $\psi$  :: o  $\Rightarrow$  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  $\Pi_1$  :: quantifiable
begin
  definition forall- $\Pi_1$  :: ( $\Pi_1 \Rightarrow$  o)  $\Rightarrow$  o where forall- $\Pi_1 \equiv$  forall_ $\Pi_1$ 
  instance proof
    fix w :: i and  $\psi$  ::  $\Pi_1 \Rightarrow$  o
    show ( $w \models \forall x . \psi x$ ) = ( $\forall x . (w \models \psi x)$ )
```



```

    unfolding forall- $\Pi_1$ -def using Semantics.T8-1 .
qed
end

instantiation  $\Pi_2 :: \text{quantifiable}$ 
begin
  definition forall- $\Pi_2 :: (\Pi_2 \Rightarrow o) \Rightarrow o$  where forall- $\Pi_2 \equiv \text{forall}_2$ 
  instance proof
    fix w :: i and  $\psi :: \Pi_2 \Rightarrow o$ 
    show  $(w \models \forall x. \psi x) = (\forall x. (w \models \psi x))$ 
      unfolding forall- $\Pi_2$ -def using Semantics.T8-2 .
    qed
  end

instantiation  $\Pi_3 :: \text{quantifiable}$ 
begin
  definition forall- $\Pi_3 :: (\Pi_3 \Rightarrow o) \Rightarrow o$  where forall- $\Pi_3 \equiv \text{forall}_3$ 
  instance proof
    fix w :: i and  $\psi :: \Pi_3 \Rightarrow o$ 
    show  $(w \models \forall x. \psi x) = (\forall x. (w \models \psi x))$ 
      unfolding forall- $\Pi_3$ -def using Semantics.T8-3 .
    qed
  end
end

```

A.4. Basic Definitions

A.4.1. Derived Connectives

```

definition conj ::  $o \Rightarrow o \Rightarrow o$  (infixl & 53) where
  conj  $\equiv \lambda x y . \neg(x \rightarrow \neg y)$ 
definition disj ::  $o \Rightarrow o \Rightarrow o$  (infixl  $\vee$  52) where
  disj  $\equiv \lambda x y . \neg x \rightarrow y$ 
definition equiv ::  $o \Rightarrow o \Rightarrow o$  (infixl  $\equiv$  51) where
  equiv  $\equiv \lambda x y . (x \rightarrow y) \ \& \ (y \rightarrow x)$ 
definition diamond ::  $o \Rightarrow o$  ( $\Diamond$ - [62] 63) where
  diamond  $\equiv \lambda \varphi . \neg \Box \neg \varphi$ 
definition (in quantifiable) exists ::  $(\text{'a} \Rightarrow o) \Rightarrow o$  (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 ::  $\Pi_1 (O!)$  where Ordinary  $\equiv \lambda x . \Diamond(O!, x^P)$ 
definition Abstract ::  $\Pi_1 (A!)$  where Abstract  $\equiv \lambda x . \neg \Diamond(O!, x^P)$ 

```

A.4.3. Identity Definitions

```

definition basic-identityE ::  $\Pi_2$  where
  basic-identityE  $\equiv \lambda^2 (\lambda x y . (O!, x^P) \ \& \ (O!, y^P))$ 
    &  $\Box(\forall F . (F, x^P) \equiv (F, y^P))$ 

```

definition *basic-identity_E-infix* :: $\kappa \Rightarrow \kappa \Rightarrow o$ (**infixl** =_E 63) **where**

$x =_E y \equiv \langle \text{basic-identity}_E, x, y \rangle$

definition *basic-identity_κ* (**infixl** =_κ 63) **where**

$\text{basic-identity}_\kappa \equiv \lambda x y . (x =_E y) \vee \langle A!, x \rangle \ \& \ \langle A!, y \rangle$
 $\ \& \ \square(\forall F. \langle x, F \rangle \equiv \langle y, F \rangle)$

definition *basic-identity₁* (**infixl** =₁ 63) **where**

$\text{basic-identity}_1 \equiv \lambda F G . \square(\forall x. \langle x^P, F \rangle \equiv \langle x^P, G \rangle)$

definition *basic-identity₂* :: $\Pi_2 \Rightarrow \Pi_2 \Rightarrow o$ (**infixl** =₂ 63) **where**

$\text{basic-identity}_2 \equiv \lambda F G . \forall x. ((\lambda y. \langle F, x^P, y^P \rangle) =_1 (\lambda y. \langle G, x^P, y^P \rangle))$
 $\ \& \ ((\lambda y. \langle F, y^P, x^P \rangle) =_1 (\lambda y. \langle G, y^P, x^P \rangle))$

definition *basic-identity₃* :: $\Pi_3 \Rightarrow \Pi_3 \Rightarrow o$ (**infixl** =₃ 63) **where**

$\text{basic-identity}_3 \equiv \lambda F G . \forall x y. (\lambda z. \langle F, z^P, x^P, y^P \rangle) =_1 (\lambda z. \langle G, z^P, x^P, y^P \rangle)$
 $\ \& \ (\lambda z. \langle F, x^P, z^P, y^P \rangle) =_1 (\lambda z. \langle G, x^P, z^P, y^P \rangle)$
 $\ \& \ (\lambda z. \langle F, x^P, y^P, z^P \rangle) =_1 (\lambda z. \langle G, x^P, y^P, z^P \rangle)$

definition *basic-identity₀* :: $o \Rightarrow o \Rightarrow o$ (**infixl** =₀ 63) **where**

$\text{basic-identity}_0 \equiv \lambda F G . (\lambda y. F) =_1 (\lambda y. G)$

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. Furthermore 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*]: $([\varphi \text{ in } v] \Longrightarrow [\psi \text{ in } v]) \Longrightarrow ([\varphi \rightarrow \psi \text{ in } v])$

by (*simp add: Semantics.T5*)

lemma *ImplE*[*meta-elim*]: $([\varphi \rightarrow \psi \text{ in } v]) \Longrightarrow ([\varphi \text{ in } v] \longrightarrow [\psi \text{ in } v])$

by (*simp add: Semantics.T5*)

lemma *ImplS*[*meta-subst*]: $([\varphi \rightarrow \psi \text{ in } v]) = ([\varphi \text{ in } v] \longrightarrow [\psi \text{ in } v])$

by (*simp add: Semantics.T5*)

A.5.2. Rules for Negation

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

A.5.3. Rules for Conjunction

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

A.5.4. Rules for Equivalence

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

A.5.5. Rules for Disjunction

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

A.5.6. Rules for Necessity

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

A.5.7. Rules for Possibility

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

A.5.8. Rules for Quantification

lemma *AllI*[*meta-intro*]: $(\bigwedge x. [\varphi \ x \ in \ v]) \implies [\forall \ x. \varphi \ x \ in \ v]$
by (*auto simp*: *T8*)
lemma *AllE*[*meta-elim*]: $[\forall \ x. \varphi \ x \ in \ v] \implies (\bigwedge x. [\varphi \ x \ in \ v])$
by (*auto simp*: *T8*)
lemma *AllS*[*meta-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*: $([\varphi \ y \ in \ v]) \implies [\exists \ x. \varphi \ x \ in \ v]$
by (*auto simp*: *exists-def Semantics.T8 Semantics.T4*)
lemma *ExI*[*meta-intro*]: $(\exists \ y. [\varphi \ y \ in \ v]) \implies [\exists \ x. \varphi \ x \ in \ v]$
by (*auto simp*: *exists-def Semantics.T8 Semantics.T4*)
lemma *ExE*[*meta-elim*]: $[\exists \ x. \varphi \ x \ in \ v] \implies (\exists \ y. [\varphi \ y \ in \ v])$
by (*auto simp*: *exists-def Semantics.T8 Semantics.T4*)
lemma *ExS*[*meta-subst*]: $[\exists \ x. \varphi \ x \ in \ v] = (\exists \ y. [\varphi \ y \ in \ v])$
by (*auto simp*: *exists-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-intro*]: $[\varphi \ in \ dw] \implies [\mathcal{A}\varphi \ in \ v]$
by (*auto simp*: *Semantics.T7*)
lemma *ActualE*[*meta-elim*]: $[\mathcal{A}\varphi \ in \ v] \implies [\varphi \ in \ dw]$
by (*auto simp*: *Semantics.T7*)
lemma *ActualS*[*meta-subst*]: $[\mathcal{A}\varphi \ in \ v] = [\varphi \ in \ dw]$
by (*auto simp*: *Semantics.T7*)

A.5.10. Rules for Encoding

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

A.5.11. Rules for Exemplification

A.5.11.1. Zero-place Relations

lemma *ExeOI*[*meta-intro*]:
assumes $\exists \ r. \text{Some } r = d_0 \ p \wedge ex0 \ r \ v$
shows $[\llbracket p \rrbracket \ in \ v]$
using *assms* **by** (*auto simp*: *Semantics.T3*)
lemma *ExeOE*[*meta-elim*]:
assumes $[\llbracket p \rrbracket \ in \ v]$
shows $\exists \ r. \text{Some } r = d_0 \ p \wedge ex0 \ r \ v$
using *assms* **by** (*auto simp*: *Semantics.T3*)
lemma *ExeOS*[*meta-subst*]:

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

A.5.11.2. One-Place Relations

lemma *Exe1I[meta-intro]*:
assumes $\exists r o_1 . \text{Some } r = d_1 F \wedge \text{Some } o_1 = d_\kappa x \wedge o_1 \in \text{ex1 } r v$
shows $[(\downarrow F, x)] \text{ in } v$
using *assms* **by** (*auto simp: Semantics.T1-1*)

lemma *Exe1E[meta-elim]*:
assumes $[(\downarrow F, x)] \text{ in } v$
shows $\exists r o_1 . \text{Some } r = d_1 F \wedge \text{Some } o_1 = d_\kappa x \wedge o_1 \in \text{ex1 } r v$
using *assms* **by** (*auto simp: Semantics.T1-1*)

lemma *Exe1S[meta-subst]*:
 $[(\downarrow F, x)] \text{ in } v = (\exists r o_1 . \text{Some } r = d_1 F \wedge \text{Some } o_1 = d_\kappa x \wedge o_1 \in \text{ex1 } r v)$
by (*auto simp: Semantics.T1-1*)

A.5.11.3. Two-Place Relations

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

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

lemma *Exe2S[meta-subst]*:
 $[(\downarrow F, x, y)] \text{ in } v = (\exists r o_1 o_2 . \text{Some } r = d_2 F \wedge \text{Some } o_1 = d_\kappa x$
 $\wedge \text{Some } o_2 = d_\kappa y \wedge (o_1, o_2) \in \text{ex2 } r v)$
by (*auto simp: Semantics.T1-2*)

A.5.11.4. Three-Place Relations

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

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

lemma *Exe3S[meta-subst]*:
 $[(\downarrow F, x, y, z)] \text{ in } v = (\exists r o_1 o_2 o_3 . \text{Some } r = d_3 F \wedge \text{Some } o_1 = d_\kappa x$
 $\wedge \text{Some } o_2 = d_\kappa y \wedge \text{Some } o_3 = d_\kappa z$
 $\wedge (o_1, o_2, o_3) \in \text{ex3 } r v)$
by (*auto simp: Semantics.T1-3*)

A.5.12. Rules for Being Ordinary

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

```

shows  $[\langle O!, x \rangle \text{ in } v]$ 
proof –
  have  $IsProperInX (\lambda x. \Diamond \langle E!, x \rangle)$ 
    by show-proper
  moreover have  $[\langle E!, x \rangle \text{ in } v]$ 
    apply meta-solver
    using ConcretenessSemantics1 propex1 assms by fast
  ultimately show  $[\langle O!, x \rangle \text{ in } v]$ 
    unfolding Ordinary-def
    using D5-1 propex1 assms ConcretenessSemantics1 Exe1S
    by blast
qed
lemma OrdE[meta-elim]:
  assumes  $[\langle O!, x \rangle \text{ in } v]$ 
  shows  $\exists o_1 y. \text{Some } o_1 = d_\kappa x \wedge o_1 = \omega \nu y$ 
  proof –
    have  $\exists r o_1. \text{Some } r = d_1 O! \wedge \text{Some } o_1 = d_\kappa x \wedge o_1 \in ex1 r v$ 
      using assms Exe1E by simp
    moreover have  $IsProperInX (\lambda x. \Diamond \langle E!, x \rangle)$ 
      by show-proper
    ultimately have  $[\langle E!, x \rangle \text{ in } v]$ 
      using D5-1 unfolding Ordinary-def by fast
    thus ?thesis
      apply – apply meta-solver
      using ConcretenessSemantics2 by blast
  qed
lemma OrdS[meta-cong]:
   $[\langle O!, x \rangle \text{ in } v] = (\exists o_1 y. \text{Some } o_1 = d_\kappa x \wedge o_1 = \omega \nu y)$ 
  using OrdI OrdE by blast

```

A.5.13. Rules for Being Abstract

```

lemma AbsI[meta-intro]:
  assumes  $\exists o_1 y. \text{Some } o_1 = d_\kappa x \wedge o_1 = \alpha \nu y$ 
  shows  $[\langle A!, x \rangle \text{ in } v]$ 
  proof –
    have  $IsProperInX (\lambda x. \neg \Diamond \langle E!, x \rangle)$ 
      by show-proper
    moreover have  $[\neg \Diamond \langle E!, x \rangle \text{ in } v]$ 
      apply meta-solver
      using ConcretenessSemantics2 propex1 assms
      by (metis  $\nu.distinct(1)$  option.sel)
    ultimately show  $[\langle A!, x \rangle \text{ in } v]$ 
      unfolding Abstract-def
      using D5-1 propex1 assms ConcretenessSemantics1 Exe1S
      by blast
  qed
lemma AbsE[meta-elim]:
  assumes  $[\langle A!, x \rangle \text{ in } v]$ 
  shows  $\exists o_1 y. \text{Some } o_1 = d_\kappa x \wedge o_1 = \alpha \nu y$ 
  proof –
    have 1:  $IsProperInX (\lambda x. \neg \Diamond \langle E!, x \rangle)$ 
      by show-proper
    have  $\exists r o_1. \text{Some } r = d_1 A! \wedge \text{Some } o_1 = d_\kappa x \wedge o_1 \in ex1 r v$ 
      using assms Exe1E by simp
    moreover hence  $[\neg \Diamond \langle E!, x \rangle \text{ in } v]$ 
      using D5-1[OF 1]
      unfolding Abstract-def by fast
  qed

```

ultimately show *?thesis*
 apply – apply *meta-solver*
 using *ConcretenessSemantics1 properx₁*
 by (*metis v.exhaust*)
 qed
 lemma *AbsS[meta-cong]*:
 $[(\lambda A!,x) \text{ in } v] = (\exists o_1 y. \text{Some } o_1 = d_\kappa x \wedge o_1 = \alpha\nu y)$
 using *AbsI AbsE* by *blast*

A.5.14. Rules for Definite Descriptions

lemma *TheEqI*:
 assumes $\bigwedge x. [\varphi x \text{ in } dw] = [\psi x \text{ in } dw]$
 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 w₀-def* by *simp*
 {
 assume $\exists o_1. \text{Some } o_1 = d_\kappa (\iota x. \varphi x)$
 hence *?thesis* using 1 *d_κ-inject* by *force*
 }
 moreover {
 assume $\neg(\exists o_1. \text{Some } o_1 = d_\kappa (\iota x. \varphi x))$
 hence *?thesis* using 1 *D3*
 by (*metis d_κ.rep-eq evalκ-inverse*)
 }
 ultimately show *?thesis* by *blast*
 qed

A.5.15. Rules for Identity

A.5.15.1. Ordinary Objects

lemma *EqEI[meta-intro]*:
 assumes $\exists o_1 o_2. \text{Some } (\omega\nu o_1) = d_\kappa x \wedge \text{Some } (\omega\nu o_2) = d_\kappa y \wedge o_1 = o_2$
 shows $[x =_E y \text{ in } v]$
 proof –
 obtain $o_1 o_2$ where 1:
 $\text{Some } (\omega\nu o_1) = d_\kappa x \wedge \text{Some } (\omega\nu o_2) = d_\kappa y \wedge o_1 = o_2$
 using *assms* by *auto*
 obtain r where 2:
 $\text{Some } r = d_2 \text{ basic-identity}_E$
 using *properx₂* by *auto*
 have $[(\lambda O!,x) \ \& \ (\lambda O!,y) \ \& \ \Box(\forall F. (F,x) \equiv (F,y)) \text{ in } v]$
 proof –
 have $[(\lambda O!,x) \text{ in } v] \wedge [(\lambda O!,y) \text{ in } v]$
 using *OrdI 1* by *blast*
 moreover have $[\Box(\forall F. (F,x) \equiv (F,y)) \text{ in } v]$
 apply *meta-solver* using 1 by *force*
 ultimately show *?thesis* using *ConjI* by *simp*
 qed
 moreover have *IsProperInXY* $(\lambda x y. (\lambda O!,x) \ \& \ (\lambda O!,y) \ \& \ \Box(\forall F. (F,x) \equiv (F,y)))$
 by *show-proper*
 ultimately have $(\omega\nu o_1, \omega\nu o_2) \in \text{ex2 } r v$
 using *D5-2 1 2*
 unfolding *basic-identity_E-def* by *fast*
 thus $[x =_E y \text{ in } v]$

```

    using Exe2I 1 2
    unfolding basic-identityE-infix-def basic-identityE-def
    by blast
qed
lemma EqEE[meta-elim]:
  assumes  $[x =_E y \text{ in } v]$ 
  shows  $\exists o_1 o_2. \text{Some } (\omega\nu o_1) = d_\kappa x \wedge \text{Some } (\omega\nu o_2) = d_\kappa y \wedge o_1 = o_2$ 
proof -
  have IsProperInXY  $(\lambda x y. \langle O!,x \rangle \ \& \ \langle O!,y \rangle \ \& \ \Box(\forall F. \langle F,x \rangle \equiv \langle F,y \rangle))$ 
    by show-proper
  hence 1:  $[\langle O!,x \rangle \ \& \ \langle O!,y \rangle \ \& \ \Box(\forall F. \langle F,x \rangle \equiv \langle F,y \rangle)] \text{ in } v$ 
    using assms unfolding basic-identityE-def basic-identityE-infix-def
    using D4-2 T1-2 D5-2 by meson
  hence 2:  $\exists o_1 o_2. \text{Some } (\omega\nu o_1) = d_\kappa x$ 
     $\wedge \text{Some } (\omega\nu o_2) = d_\kappa y$ 
    apply (subst (asm) ConjS)
    apply (subst (asm) ConjS)
    using OrdE by auto
  then obtain  $o_1 o_2$  where 3:
     $\text{Some } (\omega\nu o_1) = d_\kappa x \wedge \text{Some } (\omega\nu o_2) = d_\kappa y$ 
    by auto
  have  $\exists r. \text{Some } r = d_1 (\lambda z. \text{makeo } (\lambda w s. d_\kappa (z^P) = \text{Some } (\omega\nu o_1)))$ 
    using proper1 by auto
  then obtain  $r$  where 4:
     $\text{Some } r = d_1 (\lambda z. \text{makeo } (\lambda w s. d_\kappa (z^P) = \text{Some } (\omega\nu o_1)))$ 
    by auto
  hence 5:  $r = (\lambda u s w. \exists x. \nu\nu x = u \wedge \text{Some } x = \text{Some } (\omega\nu o_1))$ 
    unfolding lambdabinder1-def d1-def dκ-proper
    apply transfer
    by simp
  have  $[\Box(\forall F. \langle F,x \rangle \equiv \langle F,y \rangle)] \text{ in } v$ 
    using 1 using ConjE by blast
  hence 6:  $\forall v F. [\langle F,x \rangle \text{ in } v] \longleftrightarrow [\langle F,y \rangle \text{ in } v]$ 
    using BoxE EquivE AllE by fast
  hence  $\forall v. ((\omega\nu o_1) \in \text{ex1 } r v) = ((\omega\nu o_2) \in \text{ex1 } r v)$ 
    using 2 4 unfolding valid-in-def
    by (metis 3 6 d1.rep-eq dκ-inject dκ-proper ex1-def evalo-inverse exe1.rep-eq
      mem-Collect-eq option.sel rep-proper-id νκ-proper valid-in.abs-eq)
  moreover have  $(\omega\nu o_1) \in \text{ex1 } r v$ 
    unfolding 5 ex1-def by simp
  ultimately have  $(\omega\nu o_2) \in \text{ex1 } r v$ 
    by auto
  hence  $o_1 = o_2$  unfolding 5 ex1-def by (auto simp: meta-aux)
  thus ?thesis
    using 3 by auto
qed
lemma EqES[meta-subst]:
   $[x =_E y \text{ in } v] = (\exists o_1 o_2. \text{Some } (\omega\nu o_1) = d_\kappa x \wedge \text{Some } (\omega\nu o_2) = d_\kappa y$ 
     $\wedge o_1 = o_2)$ 
  using EqEI EqEE by blast

```

A.5.15.2. Individuals

```

lemma EqκI[meta-intro]:
  assumes  $\exists o_1 o_2. \text{Some } o_1 = d_\kappa x \wedge \text{Some } o_2 = d_\kappa y \wedge o_1 = o_2$ 
  shows  $[x =_\kappa y \text{ in } v]$ 
proof -
  have  $x = y$  using assms dκ-inject by meson

```


moreover have $[x =_{\kappa} x \text{ in } v]$
 unfolding *basic-identity $_{\kappa}$ -def*
 apply *meta-solver*
 by (*metis* (*no-types*, *lifting*) *assms AbsI Exe1E ν .exhaust*)
 ultimately show *?thesis* by *auto*
 qed
 lemma *Eq $_{\kappa}$ -prop*:
 assumes $[x =_{\kappa} y \text{ in } v]$
 shows $[\varphi \ x \text{ in } v] = [\varphi \ y \text{ in } v]$
 proof –
 have $[x =_E y \vee (\downarrow A!, x) \ \& \ (\downarrow A!, y) \ \& \ \Box(\forall F. \llbracket x, F \rrbracket \equiv \llbracket y, F \rrbracket) \text{ in } v]$
 using *assms unfolding basic-identity $_{\kappa}$ -def* by *simp*
 moreover {
 assume $[x =_E y \text{ in } v]$
 hence $(\exists o_1 o_2. \text{Some } o_1 = d_{\kappa} x \wedge \text{Some } o_2 = d_{\kappa} y \wedge o_1 = o_2)$
 using *Eq $_E$* by *fast*
 }
 moreover {
 assume 1: $[(\downarrow A!, x) \ \& \ (\downarrow A!, y) \ \& \ \Box(\forall F. \llbracket x, F \rrbracket \equiv \llbracket y, F \rrbracket) \text{ in } v]$
 hence 2: $(\exists o_1 o_2 X Y. \text{Some } o_1 = d_{\kappa} x \wedge \text{Some } o_2 = d_{\kappa} y \wedge o_1 = \alpha \nu X \wedge o_2 = \alpha \nu Y)$
 using *AbsE ConjE* by *meson*
 moreover then obtain $o_1 o_2 X Y$ where 3:
 $\text{Some } o_1 = d_{\kappa} x \wedge \text{Some } o_2 = d_{\kappa} y \wedge o_1 = \alpha \nu X \wedge o_2 = \alpha \nu Y$
 by *auto*
 moreover have 4: $[\Box(\forall F. \llbracket x, F \rrbracket \equiv \llbracket y, F \rrbracket) \text{ in } v]$
 using 1 *ConjE* by *blast*
 hence 6: $\forall v F. [\llbracket x, F \rrbracket \text{ in } v] \longleftrightarrow [\llbracket y, F \rrbracket \text{ in } v]$
 using *BoxE Alle EquivE* by *fast*
 hence 7: $\forall v r. (\exists o_1. \text{Some } o_1 = d_{\kappa} x \wedge o_1 \in \text{en } r) = (\exists o_1. \text{Some } o_1 = d_{\kappa} y \wedge o_1 \in \text{en } r)$
 apply – apply *meta-solver*
 using *propex $_1$ d $_1$ -inject* apply *simp*
 apply *transfer* by *simp*
 hence 8: $\forall r. (o_1 \in \text{en } r) = (o_2 \in \text{en } r)$
 using 3 *d $_{\kappa}$ -inject d $_{\kappa}$ -proper* apply *simp*
 by (*metis option.inject*)
 hence $\forall r. (o_1 \in r) = (o_2 \in r)$
 unfolding *en-def* using 3
 by (*metis Collect-cong Collect-mem-eq ν .simps(6) mem-Collect-eq make Π_1 -cases*)
 hence $(o_1 \in \{x \mid o_1 = x\}) = (o_2 \in \{x \mid o_1 = x\})$
 by *metis*
 hence $o_1 = o_2$ by *simp*
 hence $(\exists o_1 o_2. \text{Some } o_1 = d_{\kappa} x \wedge \text{Some } o_2 = d_{\kappa} y \wedge o_1 = o_2)$
 using 3 by *auto*
 }
 ultimately have $x = y$
 using *DisjS* using *Semantics.d $_{\kappa}$ -inject* by *auto*
 thus $(v \models (\varphi \ x)) = (v \models (\varphi \ y))$ by *simp*
 qed
 lemma *Eq $_{\kappa}$ E[meta-elim]*:
 assumes $[x =_{\kappa} y \text{ in } v]$
 shows $\exists o_1 o_2. \text{Some } o_1 = d_{\kappa} x \wedge \text{Some } o_2 = d_{\kappa} y \wedge o_1 = o_2$
 proof –
 have $\forall \varphi. (v \models \varphi \ x) = (v \models \varphi \ y)$
 using *assms Eq $_{\kappa}$ -prop* by *blast*
 moreover obtain φ where φ -prop:

```

 $\varphi = (\lambda \alpha . \text{makeo } (\lambda w s . (\exists o_1 o_2 . \text{Some } o_1 = d_\kappa x$ 
 $\wedge \text{Some } o_2 = d_\kappa \alpha \wedge o_1 = o_2)))$ 
  by auto
ultimately have  $(v \models \varphi x) = (v \models \varphi y)$  by metis
moreover have  $(v \models \varphi x)$ 
  using assms unfolding  $\varphi$ -prop basic-identity $_\kappa$ -def
  by (metis (mono-tags, lifting) AbsS ConjE DisjS
      EqES valid-in.abs-eq)
ultimately have  $(v \models \varphi y)$  by auto
thus ?thesis
  unfolding  $\varphi$ -prop
  by (simp add: valid-in-def meta-aux)
qed
lemma EqKS[meta-subst]:
 $[x =_\kappa y \text{ in } v] = (\exists o_1 o_2 . \text{Some } o_1 = d_\kappa x \wedge \text{Some } o_2 = d_\kappa y \wedge o_1 = o_2)$ 
  using EqKI EqKE by blast

```

A.5.15.3. One-Place Relations

```

lemma Eq1I[meta-intro]:  $F = G \implies [F =_1 G \text{ in } v]$ 
  unfolding basic-identity $_1$ -def
  apply (rule BoxI, rule AllI, rule EquivI)
  by simp
lemma Eq1E[meta-elim]:  $[F =_1 G \text{ in } v] \implies F = G$ 
  unfolding basic-identity $_1$ -def
  apply (drule BoxE, drule-tac  $x=(\alpha v \{ F \})$  in AllE, drule EquivE)
  apply (simp add: Semantics.T2)
  unfolding en-def d $_\kappa$ -def d $_1$ -def
  using  $\nu\kappa$ -proper rep-proper-id
  by (simp add: rep-def proper-def meta-aux  $\nu\kappa$ .rep-eq)
lemma Eq1S[meta-subst]:  $[F =_1 G \text{ in } v] = (F = G)$ 
  using Eq1I Eq1E by auto
lemma Eq1-prop:  $[F =_1 G \text{ in } v] \implies [\varphi F \text{ in } v] = [\varphi G \text{ in } v]$ 
  using Eq1E by blast

```

A.5.15.4. Two-Place Relations

```

lemma Eq2I[meta-intro]:  $F = G \implies [F =_2 G \text{ in } v]$ 
  unfolding basic-identity $_2$ -def
  apply (rule AllI, rule ConjI, (subst Eq1S)+)
  by simp
lemma Eq2E[meta-elim]:  $[F =_2 G \text{ in } v] \implies F = G$ 
proof -
  assume  $[F =_2 G \text{ in } v]$ 
  hence 1:  $[\forall x. (\lambda y. \langle F, x^P, y^P \rangle) =_1 (\lambda y. \langle G, x^P, y^P \rangle) \text{ in } v]$ 
  unfolding basic-identity $_2$ -def
  apply - apply meta-solver by auto
{
  fix  $u v s w$ 
  obtain  $x$  where  $x$ -def:  $\nu v x = v$  by (metis  $\nu v$ -surj surj-def)
  obtain  $a$  where  $a$ -def:
     $a = (\lambda u s w. \exists xa. \nu v xa = u \wedge \text{eval}\Pi_2 F (\nu v x) (\nu v xa) s w)$ 
    by auto
  obtain  $b$  where  $b$ -def:
     $b = (\lambda u s w. \exists xa. \nu v xa = u \wedge \text{eval}\Pi_2 G (\nu v x) (\nu v xa) s w)$ 
    by auto
  have  $a = b$  unfolding  $a$ -def  $b$ -def
    using 1 apply - apply meta-solver

```

```

    by (auto simp: meta-defs meta-aux makeΠ1-inject)
  hence a u s w = b u s w by auto
  hence (evalΠ2 F (νν x) u s w) = (evalΠ2 G (νν x) u s w)
    unfolding a-def b-def
    by (metis (no-types, hide-lams) νν-surj surj-def)
  hence (evalΠ2 F v u s w) = (evalΠ2 G v u s w)
    unfolding x-def by auto
}
hence (evalΠ2 F) = (evalΠ2 G) by blast
thus F = G by (simp add: evalΠ2-inject)
qed
lemma Eq2S[meta-subst]: [F =2 G in v] = (F = G)
  using Eq2I Eq2E by auto
lemma Eq2-prop: [F =2 G in v]  $\implies$  [φ F in v] = [φ G in v]
  using Eq2E by blast

```

A.5.15.5. Three-Place Relations

```

lemma Eq3I[meta-intro]: F = G  $\implies$  [F =3 G in v]
  apply (simp add: meta-defs meta-aux conn-defs forall-ν-def basic-identity3-def)
  using MetaSolver.Eq1I valid-in.rep-eq by auto
lemma Eq3E[meta-elim]: [F =3 G in v]  $\implies$  F = G
proof -
  assume [F =3 G in v]
  hence 1: [∀ x y. (λz. (⟦F, xP, yP, zP⟧)) =1 (λz. (⟦G, xP, yP, zP⟧)) in v]
    unfolding basic-identity3-def
    apply - apply meta-solver by auto
  {
    fix u v r s w
    obtain x where x-def: νν x = v by (metis νν-surj surj-def)
    obtain y where y-def: νν y = r by (metis νν-surj surj-def)
    obtain a where a-def:
      a = (λu s w. ∃ xa. νν xa = u ∧ evalΠ3 F (νν x) (νν y) (νν xa) s w)
      by auto
    obtain b where b-def:
      b = (λu s w. ∃ xa. νν xa = u ∧ evalΠ3 G (νν x) (νν y) (νν 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 makeΠ1-inject)
    hence a u s w = b u s w by auto
    hence (evalΠ3 F (νν x) (νν y) u s w) = (evalΠ3 G (νν x) (νν y) u s w)
      unfolding a-def b-def
      by (metis (no-types, hide-lams) νν-surj surj-def)
    hence (evalΠ3 F v r u s w) = (evalΠ3 G v r u s w)
      unfolding x-def y-def by auto
  }
  hence (evalΠ3 F) = (evalΠ3 G) by blast
  thus F = G by (simp add: evalΠ3-inject)
qed
lemma Eq3S[meta-subst]: [F =3 G in v] = (F = G)
  using Eq3I Eq3E by auto
lemma Eq3-prop: [F =3 G in v]  $\implies$  [φ F in v] = [φ G in v]
  using Eq3E by blast

```

A.5.15.6. Propositions

```

lemma Eq0I[meta-intro]:  $x = y \implies [x =_0 y \text{ in } v]$ 
  unfolding basic-identity0-def by (simp add: Eq1S)
lemma Eq0E[meta-elim]:  $[F =_0 G \text{ in } v] \implies F = G$ 
  proof -
    assume  $[F =_0 G \text{ in } v]$ 
    hence  $[(\lambda y. F) =_1 (\lambda y. G) \text{ in } v]$ 
      unfolding basic-identity0-def by simp
    hence  $(\lambda y. F) = (\lambda y. G)$ 
      using Eq1S by simp
    hence  $(\lambda u s w. (\exists x. \nu v x = u) \wedge \text{evalo } F s w)$ 
      =  $(\lambda u s w. (\exists x. \nu v x = u) \wedge \text{evalo } G s w)$ 
      apply (simp add: meta-defs meta-aux)
      by (metis (no-types, lifting) UNIV-I make $\Pi_1$ -inverse)
    hence  $\bigwedge s w. (\text{evalo } F s w) = (\text{evalo } G s w)$ 
      by metis
    hence  $(\text{evalo } F) = (\text{evalo } G)$  by blast
    thus  $F = G$ 
      by (metis evalo-inverse)
  qed
lemma Eq0S[meta-subst]:  $[F =_0 G \text{ in } v] = (F = G)$ 
  using Eq0I Eq0E by auto
lemma Eq0-prop:  $[F =_0 G \text{ in } v] \implies [\varphi F \text{ in } v] = [\varphi G \text{ in } v]$ 
  using Eq0E 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  $\Rightarrow$  'a  $\Rightarrow$  o (infixl = 63)
assumes l-identity:
   $w \models x = y \implies w \models \varphi x \implies w \models \varphi y$ 
begin
  abbreviation notequal (infixl  $\neq$  63) where
    notequal  $\equiv \lambda x y . \neg(x = y)$ 
end

class quantifiable-and-identifiable = quantifiable + identifiable
begin
  definition exists-unique::('a  $\Rightarrow$  o)  $\Rightarrow$  o (binder  $\exists!$  [8] 9) where
    exists-unique  $\equiv \lambda \varphi . \exists \alpha . \varphi \alpha \ \& \ (\forall \beta . \varphi \beta \rightarrow \beta = \alpha)$ 

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

```

A.6.2. Instantiations

instantiation $\kappa :: \text{identifiable}$

begin

definition $\text{identity-}\kappa$ **where** $\text{identity-}\kappa \equiv \text{basic-identity}_\kappa$

instance proof

fix $x\ y :: \kappa$ **and** $w\ \varphi$

show $[x = y \text{ in } w] \implies [\varphi\ x \text{ in } w] \implies [\varphi\ y \text{ in } w]$

unfolding $\text{identity-}\kappa\text{-def}$

using $\text{MetaSolver.Eq}\kappa\text{-prop} \dots$

qed

end

instantiation $\nu :: \text{identifiable}$

begin

definition $\text{identity-}\nu$ **where** $\text{identity-}\nu \equiv \lambda\ x\ y . x^P = y^P$

instance proof

fix $\alpha :: \nu$ **and** $\beta :: \nu$ **and** $v\ \varphi$

assume $v \models \alpha = \beta$

hence $v \models \alpha^P = \beta^P$

unfolding $\text{identity-}\nu\text{-def}$ **by** auto

hence $\bigwedge \varphi. (v \models \varphi\ (\alpha^P)) \implies (v \models \varphi\ (\beta^P))$

using $l\text{-identity}$ **by** auto

hence $(v \models \varphi\ (\text{rep}\ (\alpha^P))) \implies (v \models \varphi\ (\text{rep}\ (\beta^P)))$

by meson

thus $(v \models \varphi\ \alpha) \implies (v \models \varphi\ \beta)$

by $(\text{simp only: rep-proper-id})$

qed

end

instantiation $\Pi_1 :: \text{identifiable}$

begin

definition $\text{identity-}\Pi_1$ **where** $\text{identity-}\Pi_1 \equiv \text{basic-identity}_1$

instance proof

fix $F\ G :: \Pi_1$ **and** $w\ \varphi$

show $(w \models F = G) \implies (w \models \varphi\ F) \implies (w \models \varphi\ G)$

unfolding $\text{identity-}\Pi_1\text{-def}$ **using** $\text{MetaSolver.Eq}_1\text{-prop} \dots$

qed

end

instantiation $\Pi_2 :: \text{identifiable}$

begin

definition $\text{identity-}\Pi_2$ **where** $\text{identity-}\Pi_2 \equiv \text{basic-identity}_2$

instance proof

fix $F\ G :: \Pi_2$ **and** $w\ \varphi$

show $(w \models F = G) \implies (w \models \varphi\ F) \implies (w \models \varphi\ G)$

unfolding $\text{identity-}\Pi_2\text{-def}$ **using** $\text{MetaSolver.Eq}_2\text{-prop} \dots$

qed

end

instantiation $\Pi_3 :: \text{identifiable}$

begin

definition $\text{identity-}\Pi_3$ **where** $\text{identity-}\Pi_3 \equiv \text{basic-identity}_3$

instance proof

fix $F\ G :: \Pi_3$ **and** $w\ \varphi$

show $(w \models F = G) \implies (w \models \varphi\ F) \implies (w \models \varphi\ G)$

unfolding $\text{identity-}\Pi_3\text{-def}$ **using** $\text{MetaSolver.Eq}_3\text{-prop} \dots$

qed

```

end

instantiation o :: identifiable
begin
  definition identity-o where identity-o  $\equiv$  basic-identity0
  instance proof
    fix F G :: o and w  $\varphi$ 
    show (w  $\models F = G$ )  $\implies$  (w  $\models \varphi F$ )  $\implies$  (w  $\models \varphi G$ )
      unfolding identity-o-def using MetaSolver.Eq0-prop ..
    qed
end

instance  $\nu$  :: quantifiable-and-identifiable ..
instance  $\Pi_1$  :: quantifiable-and-identifiable ..
instance  $\Pi_2$  :: quantifiable-and-identifiable ..
instance  $\Pi_3$  :: 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 identityE-def[identity-defs]:
  basic-identityE  $\equiv \lambda^2 (\lambda x y. \langle O!, x^P \rangle \ \& \ \langle O!, y^P \rangle \ \& \ \Box (\forall F. \langle F, x^P \rangle \equiv \langle F, y^P \rangle))$ 
  unfolding basic-identityE-def forall- $\Pi_1$ -def by simp
lemma identityE-infix-def[identity-defs]:
   $x =_E y \equiv \langle \text{basic-identity}_E, x, y \rangle$  using basic-identityE-infix-def .
lemma identity $\kappa$ -def[identity-defs]:
   $op = \equiv \lambda x y. x =_E y \vee \langle A!, x \rangle \ \& \ \langle A!, y \rangle \ \& \ \Box (\forall F. \langle x, F \rangle \equiv \langle y, F \rangle)$ 
  unfolding identity- $\kappa$ -def basic-identity $\kappa$ -def forall- $\Pi_1$ -def by simp
lemma identity $\nu$ -def[identity-defs]:
   $op = \equiv \lambda x y. (x^P) =_E (y^P) \vee \langle A!, x^P \rangle \ \& \ \langle A!, y^P \rangle \ \& \ \Box (\forall F. \langle x^P, F \rangle \equiv \langle y^P, F \rangle)$ 
  unfolding identity- $\nu$ -def identity $\kappa$ -def by simp
lemma identity1-def[identity-defs]:
   $op = \equiv \lambda F G. \Box (\forall x. \langle x^P, F \rangle \equiv \langle x^P, G \rangle)$ 
  unfolding identity- $\Pi_1$ -def basic-identity1-def forall- $\nu$ -def by simp
lemma identity2-def[identity-defs]:
   $op = \equiv \lambda F G. \forall x. (\lambda y. \langle F, x^P, y^P \rangle) = (\lambda y. \langle G, x^P, y^P \rangle)$ 
     $\ \& \ (\lambda y. \langle F, y^P, x^P \rangle) = (\lambda y. \langle G, y^P, x^P \rangle)$ 
  unfolding identity- $\Pi_2$ -def identity- $\Pi_1$ -def basic-identity2-def forall- $\nu$ -def by simp
lemma identity3-def[identity-defs]:
   $op = \equiv \lambda F G. \forall x y. (\lambda z. \langle F, z^P, x^P, y^P \rangle) = (\lambda z. \langle G, z^P, x^P, y^P \rangle)$ 
     $\ \& \ (\lambda z. \langle F, x^P, z^P, y^P \rangle) = (\lambda z. \langle G, x^P, z^P, y^P \rangle)$ 
     $\ \& \ (\lambda z. \langle F, x^P, y^P, z^P \rangle) = (\lambda z. \langle G, x^P, y^P, z^P \rangle)$ 
  unfolding identity- $\Pi_3$ -def identity- $\Pi_1$ -def basic-identity3-def forall- $\nu$ -def by simp
lemma identityo-def[identity-defs]:  $op = \equiv \lambda F G. (\lambda y. F) = (\lambda y. G)$ 
  unfolding identity-o-def identity- $\Pi_1$ -def basic-identity0-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 $[[\cdot]]$ 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  $\Rightarrow$  bool ( $[[\cdot]]$ ) where axiom  $\equiv \lambda \varphi . \forall v . [\varphi \text{ in } v]$ 

```

```

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]:  $[[\varphi]] \Rightarrow [\varphi \text{ in } v]$ 
  unfolding axiom-def by simp
lemma closures-universal[axiom]:  $(\bigwedge x. [[\varphi x]]) \Rightarrow [[\forall x. \varphi x]]$ 
  by axiom-meta-solver
lemma closures-actualization[axiom]:  $[[\varphi]] \Rightarrow [[\mathcal{A} \varphi]]$ 
  by axiom-meta-solver
lemma closures-necessitation[axiom]:  $[[\varphi]] \Rightarrow [[\Box \varphi]]$ 
  by axiom-meta-solver
lemma necessitation-averse-axiom-instance[axiom]:  $[\varphi] \Rightarrow [\varphi \text{ in } dw]$ 
  by axiom-meta-solver
lemma necessitation-averse-closures-universal[axiom]:  $(\bigwedge x. [\varphi x]) \Rightarrow [\forall x. \varphi x]$ 
  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]:
  [[ $\varphi \rightarrow (\psi \rightarrow \varphi)$ ]]
  by axiom-meta-solver
lemma pl-2[axiom]:
  [[ $(\varphi \rightarrow (\psi \rightarrow \chi)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \chi))$ ]]
  by axiom-meta-solver
lemma pl-3[axiom]:
  [[ $(\neg\varphi \rightarrow \neg\psi) \rightarrow ((\neg\varphi \rightarrow \psi) \rightarrow \varphi)$ ]]
  by axiom-meta-solver

```

A.7.3. Axioms of Identity

```

lemma l-identity[axiom]:
  [[ $\alpha = \beta \rightarrow (\varphi \alpha \rightarrow \varphi \beta)$ ]]
  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]:
  [[ $(\forall \alpha. \varphi \alpha) \rightarrow \varphi \alpha$ ]]
  by axiom-meta-solver
lemma cqt-1- $\kappa$ [axiom]:
  [[ $(\forall \alpha. \varphi(\alpha^P)) \rightarrow ((\exists \beta. (\beta^P) = \alpha) \rightarrow \varphi \alpha)$ ]]
  proof -
  {
    fix v
    assume 1: [[ $(\forall \alpha. \varphi(\alpha^P))$  in v]]
    assume [[ $(\exists \beta. (\beta^P) = \alpha)$  in v]]
    then obtain  $\beta$  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  $\varphi=\varphi$ , axiom-instance]
      ImplS 2 by simp
  }
  thus [[ $(\forall \alpha. \varphi(\alpha^P)) \rightarrow ((\exists \beta. (\beta^P) = \alpha) \rightarrow \varphi \alpha)$ ]]
    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  $(\lambda x . \langle F, x \rangle)$ 
    | SimpleExOrEnc  $(\lambda x . \langle F, x, y \rangle)$ 
    | SimpleExOrEnc  $(\lambda x . \langle F, y, x \rangle)$ 
    | SimpleExOrEnc  $(\lambda x . \langle F, x, y, z \rangle)$ 
    | SimpleExOrEnc  $(\lambda x . \langle F, y, x, z \rangle)$ 
    | SimpleExOrEnc  $(\lambda x . \langle F, y, z, x \rangle)$ 
    | SimpleExOrEnc  $(\lambda x . \langle x, F \rangle)$ 

lemma cqt-5[axiom]:
  assumes SimpleExOrEnc  $\psi$ 
  shows [[ $(\psi (\iota x . \varphi x)) \rightarrow (\exists \alpha. (\alpha^P) = (\iota x . \varphi x))$ ]]
  proof -
    have  $\forall w . ((\psi (\iota x . \varphi x)) \text{ in } w) \longrightarrow (\exists o_1 . \text{Some } o_1 = d_\kappa (\iota x . \varphi x))$ 
      using assms apply induct by (meta-solver;metis)+
    thus ?thesis
      apply - unfolding identity- $\kappa$ -def
      apply axiom-meta-solver
      using  $d_\kappa$ -proper by auto
  qed

lemma cqt-5-mod[axiom]:
  assumes SimpleExOrEnc  $\psi$ 
  shows [[ $\psi \tau \rightarrow (\exists \alpha . (\alpha^P) = \tau)$ ]]
  proof -
    have  $\forall w . ((\psi \tau) \text{ in } w) \longrightarrow (\exists o_1 . \text{Some } o_1 = d_\kappa \tau)$ 
      using assms apply induct by (meta-solver;metis)+
    thus ?thesis
      apply - unfolding identity- $\kappa$ -def
      apply axiom-meta-solver
      using  $d_\kappa$ -proper by auto
  qed

```

A.7.5. Axioms of Actuality

Remark. The necessitation aversive 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]: [ $(\mathcal{A}\varphi) \equiv \varphi$ ]
  by axiom-meta-solver
lemma [[ $(\mathcal{A}\varphi) \equiv \varphi$ ]]
  nitpick[user-axioms, expect = genuine, card = 1, card i = 2]
  oops — Counter-model by nitpick

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

```

```

[[ $\mathcal{A}(\varphi \rightarrow \psi) \equiv (\mathcal{A}\varphi \rightarrow \mathcal{A}\psi)$ ]]
by axiom-meta-solver
lemma logic-actual-nec-3[axiom]:
[[ $\mathcal{A}(\forall \alpha. \varphi \alpha) \equiv (\forall \alpha. \mathcal{A}(\varphi \alpha))$ ]]
by axiom-meta-solver
lemma logic-actual-nec-4[axiom]:
[[ $\mathcal{A}\varphi \equiv \mathcal{A}\mathcal{A}\varphi$ ]]
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]:
[[ $\Diamond(\exists x. (\Box E!, x^P) \ \& \ \Diamond\neg(\Box E!, x^P)) \ \& \ \Diamond\neg(\exists x. (\Box E!, x^P) \ \& \ \Diamond\neg(\Box E!, x^P))$ ]]]
unfolding axiom-def
using PossiblyContingentObjectExistsAxiom
PossiblyNoContingentObjectExistsAxiom
apply (simp add: meta-defs meta-aux conn-defs forall- $\nu$ -def
split:  $\nu$ .split  $\nu$ .split)
by (metis  $\nu\nu$ - $\omega\nu$ -is- $\omega\nu$   $\nu$ .distinct(1)  $\nu$ .inject(1))

```

A.7.7. Axioms of Necessity and Actuality

```

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

```

A.7.8. Axioms of Descriptions

```

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

```

```

  ((w0 ⊨ φ X) ∧ (∀ y. (w0 ⊨ φ y) → y = X))
  by auto
moreover have o1 = (THE x. (w0 ⊨ φ x))
  using 2 3 by auto
ultimately have 5: X = o1
  by (metis (mono-tags) theI)
have ∀ z . [Aφ z in v] = [(zP) = (xP) in v]
proof
  fix z
  have [Aφ z in v] ⇒ [(zP) = (xP) in v]
    unfolding identity-κ-def apply meta-solver
    using 4 5 2 dκ-proper w0-def by auto
  moreover have [(zP) = (xP) in v] ⇒ [Aφ z in v]
    unfolding identity-κ-def apply meta-solver
    using 2 4 5
    by (simp add: dκ-proper w0-def)
  ultimately show [Aφ z in v] = [(zP) = (xP) in v]
    by auto
qed
thus [∀ z. Aφ z ≡ (z) = (x) in v]
  unfolding identity-ν-def
  by (simp add: AllEquivS)
next
  fix v
  assume [∀ z. Aφ z ≡ (z) = (x) in v]
  hence ∧z. (dw ⊨ φ z) = (∃ o1 o2. Some o1 = dκ (zP)
    ∧ Some o2 = dκ (xP) ∧ o1 = o2)
    apply - unfolding identity-ν-def identity-κ-def by meta-solver
  hence ∀ z . (dw ⊨ φ z) = (z = x)
    by (simp add: dκ-proper)
  moreover hence x = (THE z . (dw ⊨ φ z)) by simp
  ultimately have xP = (ιx. φ x)
    using D3 dκ-inject dκ-proper w0-def by presburger
  thus [xP = (ιx. φ x) in v]
    using EqκS unfolding identity-κ-def by (metis dκ-proper)
qed

```

A.7.9. Axioms for Complex Relation Terms

lemma lambda-predicates-1[axiom]:

(λ x . φ x) = (λ y . φ y) ..

lemma lambda-predicates-2-1[axiom]:

assumes IsProperInX φ

shows [[(λ x . φ (x^P), x^P) ≡ φ (x^P)]]

apply axiom-meta-solver

using D5-1[OF assms] d_κ-proper propex₁

by metis

lemma lambda-predicates-2-2[axiom]:

assumes IsProperInXY φ

shows [[(λ² (λ x y . φ (x^P) (y^P)), x^P, y^P) ≡ φ (x^P) (y^P)]]

apply axiom-meta-solver

using D5-2[OF assms] d_κ-proper propex₂

by metis

lemma lambda-predicates-2-3[axiom]:

assumes IsProperInXYZ φ

shows $[[\langle (\lambda^3 (\lambda x y z . \varphi (x^P) (y^P) (z^P))), x^P, y^P, z^P \rangle \equiv \varphi (x^P) (y^P) (z^P)]]$
proof –
 have $[[\langle (\lambda^3 (\lambda x y z . \varphi (x^P) (y^P) (z^P))), x^P, y^P, z^P \rangle \rightarrow \varphi (x^P) (y^P) (z^P)]]$
 apply *axiom-meta-solver* **using** *D5-3[OF assms]* **by** *auto*
 moreover have
 $[[\varphi (x^P) (y^P) (z^P) \rightarrow \langle (\lambda^3 (\lambda x y z . \varphi (x^P) (y^P) (z^P))), x^P, y^P, z^P \rangle]]$
 apply *axiom-meta-solver*
using *D5-3[OF assms]* *d_κ-proper propex₃*
by (*metis (no-types, lifting)*)
 ultimately show *?thesis* **unfolding** *axiom-def equiv-def ConjS* **by** *blast*
qed

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

lemma *lambda-predicates-3-1[axiom]*:
 $[[\langle \lambda x . \langle F, x^P \rangle \rangle = F]]$
unfolding *axiom-def*
apply (*rule allI*)
unfolding *identity-Π₁-def* **apply** (*rule Eq₁I*)
using *D4-1 d₁-inject* **by** *simp*

lemma *lambda-predicates-3-2[axiom]*:
 $[[\langle \lambda^2 (\lambda x y . \langle F, x^P, y^P \rangle) \rangle = F]]$
unfolding *axiom-def*
apply (*rule allI*)
unfolding *identity-Π₂-def* **apply** (*rule Eq₂I*)
using *D4-2 d₂-inject* **by** *simp*

lemma *lambda-predicates-3-3[axiom]*:
 $[[\langle \lambda^3 (\lambda x y z . \langle F, x^P, y^P, z^P \rangle) \rangle = F]]$
unfolding *axiom-def*
apply (*rule allI*)
unfolding *identity-Π₃-def* **apply** (*rule Eq₃I*)
using *D4-3 d₃-inject* **by** *simp*

lemma *lambda-predicates-4-0[axiom]*:
assumes $\bigwedge x. [\langle \mathcal{A}(\varphi x \equiv \psi x) \rangle \text{ in } v]$
shows $[[\langle \lambda^0 (\chi (\iota x. \varphi x)) \rangle = \lambda^0 (\chi (\iota x. \psi x))]]$
unfolding *axiom-def identity-o-def* **apply** – **apply** (*rule allI; rule Eq₀I*)
using *TheEqI[OF assms[THEN ActualE, THEN EquivE]]* **by** *auto*

lemma *lambda-predicates-4-1[axiom]*:
assumes $\bigwedge x. [\langle \mathcal{A}(\varphi x \equiv \psi x) \rangle \text{ in } v]$
shows $[[\langle (\lambda x . \chi (\iota x. \varphi x) x) \rangle = \langle \lambda x . \chi (\iota x. \psi x) x \rangle]]$
unfolding *axiom-def identity-Π₁-def* **apply** – **apply** (*rule allI; rule Eq₁I*)
using *TheEqI[OF assms[THEN ActualE, THEN EquivE]]* **by** *auto*

lemma *lambda-predicates-4-2[axiom]*:
assumes $\bigwedge x. [\langle \mathcal{A}(\varphi x \equiv \psi x) \rangle \text{ in } v]$
shows $[[\langle (\lambda^2 (\lambda x y . \chi (\iota x. \varphi x) x y)) \rangle = \langle \lambda^2 (\lambda x y . \chi (\iota x. \psi x) x y) \rangle]]$
unfolding *axiom-def identity-Π₂-def* **apply** – **apply** (*rule allI; rule Eq₂I*)
using *TheEqI[OF assms[THEN ActualE, THEN EquivE]]* **by** *auto*

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

```

assumes  $\bigwedge x. [\mathcal{A}(\varphi x \equiv \psi x) \text{ in } v]$ 
shows  $[(\lambda^3 (\lambda x y z . \chi (\iota x. \varphi x) x y z)) = (\lambda^3 (\lambda x y z . \chi (\iota x. \psi x) x y z))]$ 
unfolding axiom-def identity- $\Pi_3$ -def apply – apply (rule allI; rule Eq3I)
using TheEqI[OF assms[THEN ActualE, THEN EquivE]] by auto

```

A.7.10. Axioms of Encoding

```

lemma encoding[axiom]:
   $[[\langle x, F \rangle \rightarrow \Box \langle x, F \rangle]]$ 
  by axiom-meta-solver
lemma nocoder[axiom]:
   $[[\langle \langle O!, x \rangle \rightarrow \neg(\exists F . \langle x, F \rangle)]]$ 
  unfolding axiom-def
  apply (rule allI, rule ImplI, subst (asm) OrdS)
  apply meta-solver unfolding en-def
  by (metis  $\nu$ .simps(5) mem-Collect-eq option.sel)
lemma A-objects[axiom]:
   $[[\exists x. \langle A!, x^P \rangle \ \& \ (\forall F . (\langle x^P, F \rangle \equiv \varphi F))]]$ 
  unfolding axiom-def
  proof (rule allI, rule ExIRule)
    fix v
    let  $?x = \alpha \nu \{ F . [\varphi F \text{ in } v] \}$ 
    have  $[\langle A!, ?x^P \rangle \text{ in } v]$  by (simp add: AbsS  $d_\kappa$ -proper)
    moreover have  $[(\forall F. \langle ?x^P, F \rangle \equiv \varphi F) \text{ in } v]$ 
      apply meta-solver unfolding en-def
      using  $d_1$ .rep-eq  $d_\kappa$ -def  $d_\kappa$ -proper eval $\Pi_1$ -inverse by auto
    ultimately show  $[\langle A!, ?x^P \rangle \ \& \ (\forall F. \langle ?x^P, F \rangle \equiv \varphi F) \text{ in } v]$ 
      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 propnot0  $\equiv$  propnot ::  $\Pi_0 \Rightarrow \Pi_0$ 
      propnot1  $\equiv$  propnot ::  $\Pi_1 \Rightarrow \Pi_1$ 
      propnot2  $\equiv$  propnot ::  $\Pi_2 \Rightarrow \Pi_2$ 
      propnot3  $\equiv$  propnot ::  $\Pi_3 \Rightarrow \Pi_3$ 
begin
  definition propnot0 ::  $\Pi_0 \Rightarrow \Pi_0$  where
    propnot0  $\equiv \lambda p . \lambda^0 (\neg p)$ 
  definition propnot1 where
    propnot1  $\equiv \lambda F . \lambda x . \neg \langle F, x^P \rangle$ 
  definition propnot2 where
    propnot2  $\equiv \lambda F . \lambda^2 (\lambda x y . \neg \langle F, x^P, y^P \rangle)$ 
  definition propnot3 where
    propnot3  $\equiv \lambda F . \lambda^3 (\lambda x y z . \neg \langle F, x^P, y^P, z^P \rangle)$ 
end

```

named-theorems propnot-defs

```

declare propnot0-def[propnot-defs] propnot1-def[propnot-defs]
          propnot2-def[propnot-defs] propnot3-def[propnot-defs]

```

A.8.2. Noncontingent and Contingent Relations

```

consts Necessary :: 'a ⇒ o
overloading Necessary0 ≡ Necessary :: Π0 ⇒ o
              Necessary1 ≡ Necessary :: Π1 ⇒ o
              Necessary2 ≡ Necessary :: Π2 ⇒ o
              Necessary3 ≡ Necessary :: Π3 ⇒ o

begin
  definition Necessary0 where
    Necessary0 ≡ λ p . □p
  definition Necessary1 :: Π1 ⇒ o where
    Necessary1 ≡ λ F . □(∀ x . (F, xP))
  definition Necessary2 where
    Necessary2 ≡ λ F . □(∀ x y . (F, xP, yP))
  definition Necessary3 where
    Necessary3 ≡ λ F . □(∀ x y z . (F, xP, yP, zP))
end

named-theorems Necessary-defs
declare Necessary0-def[Necessary-defs] Necessary1-def[Necessary-defs]
          Necessary2-def[Necessary-defs] Necessary3-def[Necessary-defs]

consts Impossible :: 'a ⇒ o
overloading Impossible0 ≡ Impossible :: Π0 ⇒ o
              Impossible1 ≡ Impossible :: Π1 ⇒ o
              Impossible2 ≡ Impossible :: Π2 ⇒ o
              Impossible3 ≡ Impossible :: Π3 ⇒ o

begin
  definition Impossible0 where
    Impossible0 ≡ λ p . □¬p
  definition Impossible1 where
    Impossible1 ≡ λ F . □(∀ x . ¬(F, xP))
  definition Impossible2 where
    Impossible2 ≡ λ F . □(∀ x y . ¬(F, xP, yP))
  definition Impossible3 where
    Impossible3 ≡ λ F . □(∀ x y z . ¬(F, xP, yP, zP))
end

named-theorems Impossible-defs
declare Impossible0-def[Impossible-defs] Impossible1-def[Impossible-defs]
          Impossible2-def[Impossible-defs] Impossible3-def[Impossible-defs]

definition NonContingent where
  NonContingent ≡ λ F . (Necessary F) ∨ (Impossible F)
definition Contingent where
  Contingent ≡ λ F . ¬(Necessary F ∨ Impossible F)

definition ContingentlyTrue :: o ⇒ o where
  ContingentlyTrue ≡ λ p . p & ◇¬p
definition ContingentlyFalse :: o ⇒ o where
  ContingentlyFalse ≡ λ p . ¬p & ◇p

definition WeaklyContingent where
  WeaklyContingent ≡ λ F . Contingent F & (∀ x . ◇(F, xP) → □(F, xP))

```

A.8.3. Null and Universal Objects

definition $Null :: \kappa \Rightarrow o$ **where**

$Null \equiv \lambda x . \langle A!, x \rangle \ \& \ \neg(\exists F . \langle x, F \rangle)$

definition $Universal :: \kappa \Rightarrow o$ **where**

$Universal \equiv \lambda x . \langle A!, x \rangle \ \& \ (\forall F . \langle x, F \rangle)$

definition $NullObject :: \kappa (a_0)$ **where**

$NullObject \equiv (\iota x . Null (x^P))$

definition $UniversalObject :: \kappa (a_V)$ **where**

$UniversalObject \equiv (\iota x . Universal (x^P))$

A.8.4. Propositional Properties

definition $Propositional$ **where**

$Propositional F \equiv \exists p . F = (\lambda x . p)$

A.8.5. Indiscriminate Properties

definition $Indiscriminate :: \Pi_1 \Rightarrow o$ **where**

$Indiscriminate \equiv \lambda F . \Box((\exists x . \langle F, x^P \rangle) \rightarrow (\forall x . \langle F, x^P \rangle))$

A.8.6. Miscellaneous

definition $not_identical_E :: \kappa \Rightarrow \kappa \Rightarrow o$ (**infixl** \neq_E 63)

where $not_identical_E \equiv \lambda x y . \langle (\lambda^2 (\lambda x y . x^P =_E y^P))^{-}, x, y \rangle$

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

$= ((assumption \mid (match \text{ axiom in } A: [[\varphi]] \text{ for } \varphi \Rightarrow \langle fact A[axiom_instance] \rangle)$
 $\mid fact \ PLM \mid rule \ PLM_intro \mid subst \ PLM_subst \mid subst \ (asm) \ PLM_subst$
 $\mid fastforce \mid safe \mid drule \ PLM_dest \mid erule \ PLM_elim); (PLM_solver)?)$

A.9.2. Modus Ponens

lemma $modus_ponens[PLM]:$

$[[\varphi \text{ in } v]; [\varphi \rightarrow \psi \text{ in } v]] \Longrightarrow [\psi \text{ in } v]$

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]:
   $\llbracket [\varphi \text{ in } v]; [\varphi \rightarrow \psi \text{ in } v] \rrbracket \Longrightarrow [\psi \text{ in } v]$ 
  using modus-ponens .
lemma vdash-properties-9[PLM]:
   $[\varphi \text{ in } v] \Longrightarrow [\psi \rightarrow \varphi \text{ in } v]$ 
  using modus-ponens pl-1[axiom-instance] by blast
lemma vdash-properties-10[PLM]:
   $[\varphi \rightarrow \psi \text{ in } v] \Longrightarrow ([\varphi \text{ in } v] \Longrightarrow [\psi \text{ in } v])$ 
  using vdash-properties-6 .

attribute-setup deduction =  $\langle\langle$ 
  Scan.succeed (Thm.rule-attribute [])
  (fn - => fn thm => thm RS @{thm vdash-properties-10})
 $\rangle\rangle$ 

```

A.9.5. GEN and RN

```

lemma rule-gen[PLM]:
   $\llbracket \bigwedge \alpha . [\varphi \alpha \text{ in } v] \rrbracket \Longrightarrow [\forall \alpha . \varphi \alpha \text{ in } v]$ 
  by (simp add: Semantics.T8)

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

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

```

A.9.6. Negations and Conditionals

```

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

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

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

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

```


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

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

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

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

using *reductio-aa-1 vdash-properties-10* **by** *blast*
lemma *reductio-aa-4[PLM]*:

$$\llbracket [\varphi \rightarrow \neg\psi \text{ in } v]; [\varphi \rightarrow \psi \text{ in } v] \rrbracket \Longrightarrow [\neg\varphi \text{ in } v]$$
using *reductio-aa-2 vdash-properties-10* **by** *blast*
lemma *raa-cor-1[PLM]*:

$$\llbracket [\varphi \text{ in } v]; [\neg\psi \text{ in } v] \rrbracket \Longrightarrow [\neg\varphi \text{ in } v] \Longrightarrow ([\varphi \text{ in } v] \Longrightarrow [\psi \text{ in } v])$$
using *reductio-aa-1 vdash-properties-9* **by** *blast*
lemma *raa-cor-2[PLM]*:

$$\llbracket [\neg\varphi \text{ in } v]; [\neg\psi \text{ in } v] \rrbracket \Longrightarrow [\varphi \text{ in } v] \Longrightarrow ([\neg\varphi \text{ in } v] \Longrightarrow [\psi \text{ in } v])$$
using *reductio-aa-1 vdash-properties-9* **by** *blast*
lemma *raa-cor-3[PLM]*:

$$\llbracket [\varphi \text{ in } v]; [\neg\psi \rightarrow \neg\varphi \text{ in } v] \rrbracket \Longrightarrow ([\varphi \text{ in } v] \Longrightarrow [\psi \text{ in } v])$$
using *raa-cor-1 vdash-properties-10* **by** *blast*
lemma *raa-cor-4[PLM]*:

$$\llbracket [\neg\varphi \text{ in } v]; [\neg\psi \rightarrow \varphi \text{ in } v] \rrbracket \Longrightarrow ([\neg\varphi \text{ in } v] \Longrightarrow [\psi \text{ in } v])$$
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]*:

$$\llbracket [\varphi \text{ in } v]; [\psi \text{ in } v] \rrbracket \Longrightarrow [\varphi \ \& \ \psi \text{ in } v]$$
unfolding *conj-def* **using** *ded-thm-cor-4 if-p-then-p modus-tollens-2* **by** *blast*
lemmas $\&I = \text{intro-elim-1}$
lemma *intro-elim-2-a[PLM]*:

$$[\varphi \ \& \ \psi \text{ in } v] \Longrightarrow [\varphi \text{ in } v]$$
unfolding *conj-def* **using** *CP reductio-aa-1* **by** *blast*
lemma *intro-elim-2-b[PLM]*:

$$[\varphi \ \& \ \psi \text{ in } v] \Longrightarrow [\psi \text{ in } v]$$
unfolding *conj-def* **using** *pl-1 CP reductio-aa-1 axiom-instance* **by** *blast*
lemmas $\&E = \text{intro-elim-2-a intro-elim-2-b}$
lemma *intro-elim-3-a[PLM]*:

$$[\varphi \text{ in } v] \Longrightarrow [\varphi \vee \psi \text{ in } v]$$
unfolding *disj-def* **using** *ded-thm-cor-4 useful-tautologies-3* **by** *blast*
lemma *intro-elim-3-b[PLM]*:

$$[\psi \text{ in } v] \Longrightarrow [\varphi \vee \psi \text{ in } v]$$
by (*simp only: disj-def vdash-properties-9*)
lemmas $\vee I = \text{intro-elim-3-a intro-elim-3-b}$
lemma *intro-elim-4-a[PLM]*:

$$\llbracket [\varphi \vee \psi \text{ in } v]; [\varphi \rightarrow \chi \text{ in } v]; [\psi \rightarrow \chi \text{ in } v] \rrbracket \Longrightarrow [\chi \text{ in } v]$$
unfolding *disj-def* **by** (*meson reductio-aa-2 vdash-properties-10*)
lemma *intro-elim-4-b[PLM]*:

$$\llbracket [\varphi \vee \psi \text{ in } v]; [\neg\varphi \text{ in } v] \rrbracket \Longrightarrow [\psi \text{ in } v]$$
unfolding *disj-def* **using** *vdash-properties-10* **by** *blast*
lemma *intro-elim-4-c[PLM]*:

$$\llbracket [\varphi \vee \psi \text{ in } v]; [\neg\psi \text{ in } v] \rrbracket \Longrightarrow [\varphi \text{ in } v]$$
unfolding *disj-def* **using** *raa-cor-2 vdash-properties-10* **by** *blast*
lemma *intro-elim-4-d[PLM]*:

$$\llbracket [\varphi \vee \psi \text{ in } v]; [\varphi \rightarrow \chi \text{ in } v]; [\psi \rightarrow \Theta \text{ in } v] \rrbracket \Longrightarrow [\chi \vee \Theta \text{ in } v]$$
unfolding *disj-def* **using** *contraposition-1 ded-thm-cor-3* **by** *blast*
lemma *intro-elim-4-e[PLM]*:

$$\llbracket [\varphi \vee \psi \text{ in } v]; [\varphi \equiv \chi \text{ in } v]; [\psi \equiv \Theta \text{ in } v] \rrbracket \Longrightarrow [\chi \vee \Theta \text{ in } v]$$
unfolding *equiv-def* **using** *&E(1) intro-elim-4-d* **by** *blast*
lemmas $\vee E = \text{intro-elim-4-a intro-elim-4-b intro-elim-4-c intro-elim-4-d}$
lemma *intro-elim-5[PLM]*:

$$\llbracket [\varphi \rightarrow \psi \text{ in } v]; [\psi \rightarrow \varphi \text{ in } v] \rrbracket \Longrightarrow [\varphi \equiv \psi \text{ in } v]$$

```

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

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

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

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

```

```

using NotI NotE by blast

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

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

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

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

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

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

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

```

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

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

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

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

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

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

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

```

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

attribute-setup equiv-lr = ⟨⟨
  Scan.succeed (Thm.rule-attribute []
    (fn - => fn thm => thm RS @{thm  $\equiv E(1)$ }))
  ⟩⟩

attribute-setup equiv-rl = ⟨⟨
  Scan.succeed (Thm.rule-attribute []
    (fn - => fn thm => thm RS @{thm  $\equiv E(2)$ }))
  ⟩⟩

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]:
[[ $((F::\Pi_1) = G) \rightarrow (G = F)$ ] 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]:
[[ $((F::\Pi_1) = G) \& (G = H) \rightarrow (F = H)$ ] in  $v$ ]
by (metis l-identity[axiom-instance] ded-thm-cor-4 CP &E)

```

```

lemma id-eq-prop-prop-4-a[PLM]:
  [(F::Π2) = F in v]
  unfolding identity-defs by PLM-solver
lemma id-eq-prop-prop-4-b[PLM]:
  [(F::Π3) = F in v]
  unfolding identity-defs by PLM-solver
lemma id-eq-prop-prop-5-a[PLM]:
  [((F::Π2) = G) → (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::Π3) = G) → (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::Π2) = G) & (G = H)) → (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::Π3) = G) & (G = H)) → (F = H) in v]
  by (metis l-identity[axiom-instance] ded-thm-cor-4 CP &E)
lemma id-eq-prop-prop-7[PLM]:
  [(p::Π0) = p in v]
  unfolding identity-defs by PLM-solver
lemma id-eq-prop-prop-7-b[PLM]:
  [(p::o) = p in v]
  unfolding identity-defs by PLM-solver
lemma id-eq-prop-prop-8[PLM]:
  [((p::Π0) = q) → (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::o) = q) → (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::Π0) = q) & (q = r)) → (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::o) = q) & (q = r)) → (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) ≡ ((O!,x) & (O!,y) & □(∀ F . (F,x) ≡ (F,y))) in v]
  proof (rule ≡I; rule CP)
    assume 1: [x =E y in v]
    have [∀ x y . ((xP) =E (yP)) ≡ ((O!,xP) & (O!,yP)
      & □(∀ F . (F,xP) ≡ (F,yP))) in v]
      unfolding identityE-infix-def identityE-def
      apply (rule lambda-predicates-2-2[axiom-universal, axiom-universal, axiom-instance])
      by show-proper
    moreover have [∃ α . (αP) = x in v]
      apply (rule cqt-5-mod[where ψ=λ x . x =E y, axiom-instance, deduction])
      unfolding identityE-infix-def
      apply (rule SimpleExOrEnc.intros)
      using 1 unfolding identityE-infix-def by auto
    moreover have [∃ β . (βP) = y in v]
      apply (rule cqt-5-mod[where ψ=λ y . x =E y, axiom-instance, deduction])
      unfolding identityE-infix-def
      apply (rule SimpleExOrEnc.intros) using 1
      unfolding identityE-infix-def 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 [(⟦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 . ((xP) =E (yP)) ≡ (⟦O!,xP⟧ & ⟦O!,yP⟧
    & □(∀ F . ⟦F,xP⟧ ≡ ⟦F,yP⟧)) in v]
    unfolding identityE-def identityE-infix-def
    apply (rule lambda-predicates-2-2[axiom-universal, axiom-universal, axiom-instance])
    by show-proper
  moreover have [∃ α . (αP) = 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 . (βP) = 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]: [(xP) = (xP) in v]
proof –
  have [(⟦O!, xP⟧) ∨ (¬⟦O!, xP⟧) in v]
    using PLM.oth-class-taut-2 by simp
  hence [(⟦O!, xP⟧) in v] ∨ [(¬⟦O!, xP⟧) in v]
    using CP ∨ E(1) by blast
  moreover {
    assume [(⟦O!, xP⟧) in v]
    hence [(λx. ⟦O!,xP⟧),xP] in v]
      apply (rule lambda-predicates-2-1[axiom-instance, equiv-rl, rotated])
      by show-proper
    hence [(λx. ⟦O!,xP⟧),xP] & [(λx. ⟦O!,xP⟧),xP]
      & □(∀ F . ⟦F,xP⟧ ≡ ⟦F,xP⟧) in v]
      apply – by PLM-solver
    hence [(xP) =E (xP) in v]
      using eq-E-simple-1[equiv-rl] unfolding Ordinary-def by fast
  }
  moreover {
    assume [(¬⟦O!, xP⟧) in v]
    hence [(λx. ¬⟦O!,xP⟧),xP] in v]
      apply (rule lambda-predicates-2-1[axiom-instance, equiv-rl, rotated])
      by show-proper
    hence [(λx. ¬⟦O!,xP⟧),xP] & [(λx. ¬⟦O!,xP⟧),xP]
      & □(∀ F . ⟦xP,F⟧ ≡ ⟦xP,F⟧) in v]
      apply – by PLM-solver
  }

```

```

    }
    ultimately show ?thesis unfolding identity-defs Ordinary-def Abstract-def
      using  $\forall I$  by blast
  qed
lemma id-eq-obj-2[PLM]:
   $[(x^P) = (y^P)) \rightarrow ((y^P) = (x^P))$  in  $v$ 
  by (meson l-identity[axiom-instance] id-eq-obj-1 CP ded-thm-cor-3)
lemma id-eq-obj-3[PLM]:
   $[(x^P) = (y^P)) \ \& \ ((y^P) = (z^P)) \rightarrow ((x^P) = (z^P))$  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:  $[(x :: 'a) = y] \rightarrow (y = x)$  in  $v$ 
  assumes id-eq-3:  $[(x :: 'a) = y] \ \& \ (y = z) \rightarrow (x = z)$  in  $v$ 

```

```

instantiation  $\nu :: id-eq$ 
begin
  instance proof
    fix  $x :: \nu$  and  $v$ 
    show  $[x = x]$  in  $v$ 
      using PLM.id-eq-obj-1
      by (simp add: identity- $\nu$ -def)
  next
    fix  $x y :: \nu$  and  $v$ 
    show  $[x = y \rightarrow y = x]$  in  $v$ 
      using PLM.id-eq-obj-2
      by (simp add: identity- $\nu$ -def)
  next
    fix  $x y z :: \nu$  and  $v$ 
    show  $[(x = y) \ \& \ (y = z)] \rightarrow x = z$  in  $v$ 
      using PLM.id-eq-obj-3
      by (simp add: identity- $\nu$ -def)
  qed
end

```

```

instantiation  $\circ :: id-eq$ 
begin
  instance proof
    fix  $x :: \circ$  and  $v$ 
    show  $[x = x]$  in  $v$ 
      using PLM.id-eq-prop-prop-7 .
  next
    fix  $x y :: \circ$  and  $v$ 
    show  $[x = y \rightarrow y = x]$  in  $v$ 
      using PLM.id-eq-prop-prop-8 .
  next
    fix  $x y z :: \circ$  and  $v$ 
    show  $[(x = y) \ \& \ (y = z)] \rightarrow x = z$  in  $v$ 
      using PLM.id-eq-prop-prop-9 .
  qed
end

```

```

instantiation  $\Pi_1 :: id-eq$ 
begin

```

```

instance proof
  fix x ::  $\Pi_1$  and v
  show  $[x = x \text{ in } v]$ 
    using PLM.id-eq-prop-prop-1 .
next
  fix x y ::  $\Pi_1$  and v
  show  $[x = y \rightarrow y = x \text{ in } v]$ 
    using PLM.id-eq-prop-prop-2 .
next
  fix x y z ::  $\Pi_1$  and v
  show  $[(x = y) \ \& \ (y = z) \rightarrow x = z \text{ in } v]$ 
    using PLM.id-eq-prop-prop-3 .
qed
end

```

```

instantiation  $\Pi_2 :: \text{id-eq}$ 
begin
  instance proof
    fix x ::  $\Pi_2$  and v
    show  $[x = x \text{ in } v]$ 
      using PLM.id-eq-prop-prop-4-a .
  next
    fix x y ::  $\Pi_2$  and v
    show  $[x = y \rightarrow y = x \text{ in } v]$ 
      using PLM.id-eq-prop-prop-5-a .
  next
    fix x y z ::  $\Pi_2$  and v
    show  $[(x = y) \ \& \ (y = z) \rightarrow x = z \text{ in } v]$ 
      using PLM.id-eq-prop-prop-6-a .
  qed
end

```

```

instantiation  $\Pi_3 :: \text{id-eq}$ 
begin
  instance proof
    fix x ::  $\Pi_3$  and v
    show  $[x = x \text{ in } v]$ 
      using PLM.id-eq-prop-prop-4-b .
  next
    fix x y ::  $\Pi_3$  and v
    show  $[x = y \rightarrow y = x \text{ in } v]$ 
      using PLM.id-eq-prop-prop-5-b .
  next
    fix x y z ::  $\Pi_3$  and v
    show  $[(x = y) \ \& \ (y = z) \rightarrow x = z \text{ in } v]$ 
      using PLM.id-eq-prop-prop-6-b .
  qed
end

```

```

context PLM
begin
  lemma id-eq-1[PLM]:
     $[(x :: 'a :: \text{id-eq}) = x \text{ in } v]$ 
    using id-eq-1 .
  lemma id-eq-2[PLM]:
     $[(x :: 'a :: \text{id-eq}) = y \rightarrow (y = x) \text{ in } v]$ 
    using id-eq-2 .
  lemma id-eq-3[PLM]:

```

$[(x :: 'a :: id\text{-}eq) = y) \ \& \ (y = z) \rightarrow (x = z) \text{ in } v]$
using *id-eq-3* .

attribute-setup *eq-sym* = $\langle\langle$
Scan.succeed (*Thm.rule-attribute* []
 (*fn* - => *fn thm* => *thm RS* @{*thm id-eq-2*[*deduction*]})
 $\rangle\rangle$

lemma *all-self-eq-1*[*PLM*]:
 $[\Box(\forall \alpha :: 'a :: id\text{-}eq . \alpha = \alpha) \text{ in } v]$
by *PLM-solver*

lemma *all-self-eq-2*[*PLM*]:
 $[\forall \alpha :: 'a :: id\text{-}eq . \Box(\alpha = \alpha) \text{ in } v]$
by *PLM-solver*

lemma *t-id-t-proper-1*[*PLM*]:
 $[\tau = \tau' \rightarrow (\exists \beta . (\beta^P) = \tau) \text{ in } v]$
proof (*rule CP*)
assume $[\tau = \tau' \text{ in } v]$
moreover {
assume $[\tau =_E \tau' \text{ in } v]$
hence $[\exists \beta . (\beta^P) = \tau \text{ in } v]$
apply -
apply (*rule cqt-5-mod*[**where** $\psi = \lambda \tau . \tau =_E \tau'$, *axiom-instance*, *deduction*])
subgoal unfolding *identity-defs* **by** (*rule SimpleExOrEnc.intros*)
by *simp*
 }
moreover {
assume $[(\llbracket A! , \tau \rrbracket \ \& \ \llbracket A! , \tau' \rrbracket) \ \& \ \Box(\forall F . \llbracket \tau , F \rrbracket \equiv \llbracket \tau' , F \rrbracket) \text{ in } v]$
hence $[\exists \beta . (\beta^P) = \tau \text{ in } v]$
apply -
apply (*rule cqt-5-mod*[**where** $\psi = \lambda \tau . \llbracket A! , \tau \rrbracket$, *axiom-instance*, *deduction*])
subgoal unfolding *identity-defs* **by** (*rule SimpleExOrEnc.intros*)
by *PLM-solver*
 }
ultimately show $[\exists \beta . (\beta^P) = \tau \text{ in } v]$ **unfolding** *identity_κ-def*
using *intro-elim-4-b* *reductio-aa-1* **by** *blast*
qed

lemma *t-id-t-proper-2*[*PLM*]: $[\tau = \tau' \rightarrow (\exists \beta . (\beta^P) = \tau') \text{ in } v]$
proof (*rule CP*)
assume $[\tau = \tau' \text{ in } v]$
moreover {
assume $[\tau =_E \tau' \text{ in } v]$
hence $[\exists \beta . (\beta^P) = \tau' \text{ in } v]$
apply -
apply (*rule cqt-5-mod*[**where** $\psi = \lambda \tau' . \tau =_E \tau'$, *axiom-instance*, *deduction*])
subgoal unfolding *identity-defs* **by** (*rule SimpleExOrEnc.intros*)
by *simp*
 }
moreover {
assume $[(\llbracket A! , \tau \rrbracket \ \& \ \llbracket A! , \tau' \rrbracket) \ \& \ \Box(\forall F . \llbracket \tau , F \rrbracket \equiv \llbracket \tau' , F \rrbracket) \text{ in } v]$
hence $[\exists \beta . (\beta^P) = \tau' \text{ in } v]$
apply -
apply (*rule cqt-5-mod*[**where** $\psi = \lambda \tau . \llbracket A! , \tau \rrbracket$, *axiom-instance*, *deduction*])
subgoal unfolding *identity-defs* **by** (*rule SimpleExOrEnc.intros*)
by *PLM-solver*
 }

```

}
ultimately show  $[\exists \beta . (\beta^P) = \tau' \text{ in } v]$  unfolding identity $_{\kappa}$ -def
  using intro-elim-4-b reductio-aa-1 by blast
qed

lemma id-nec[PLM]:  $[(\alpha::'a::id-eq) = (\beta)) \equiv \Box((\alpha) = (\beta)) \text{ in } v]$ 
  apply (rule  $\equiv I$ )
  using l-identity[where  $\varphi = (\lambda \beta . \Box((\alpha) = (\beta)))$ , axiom-instance]
    id-eq-1 RN ded-thm-cor-4 unfolding identity- $\nu$ -def
  apply blast
  using qml-2[axiom-instance] by blast

lemma id-nec-desc[PLM]:
   $[(\lambda x . \varphi x) = (\lambda x . \psi x)) \equiv \Box((\lambda x . \varphi x) = (\lambda x . \psi x)) \text{ in } v]$ 
proof (cases  $[(\exists \alpha . (\alpha^P) = (\lambda x . \varphi x)) \text{ in } v] \wedge [(\exists \beta . (\beta^P) = (\lambda x . \psi x)) \text{ in } v]$ )
  assume  $[(\exists \alpha . (\alpha^P) = (\lambda x . \varphi x)) \text{ in } v] \wedge [(\exists \beta . (\beta^P) = (\lambda x . \psi x)) \text{ in } v]$ 
  then obtain  $\alpha$  and  $\beta$  where
     $[(\alpha^P) = (\lambda x . \varphi x) \text{ in } v] \wedge [(\beta^P) = (\lambda x . \psi x) \text{ in } v]$ 
    apply – unfolding conn-defs by PLM-solver
  moreover {
    moreover have  $[(\alpha) = (\beta) \equiv \Box((\alpha) = (\beta)) \text{ in } v]$  by PLM-solver
    ultimately have  $[(\lambda x . \varphi x) = (\beta^P) \equiv \Box((\lambda x . \varphi x) = (\beta^P)) \text{ in } v]$ 
      using l-identity[where  $\varphi = \lambda \alpha . (\alpha) = (\beta^P) \equiv \Box((\alpha) = (\beta^P))$ , axiom-instance]
        modus-ponens unfolding identity- $\nu$ -def by metis
    }
  ultimately show ?thesis
    using l-identity[where  $\varphi = \lambda \alpha . (\lambda x . \varphi x) = (\alpha)$ 
       $\equiv \Box((\lambda x . \varphi x) = (\alpha))$ , axiom-instance]
      modus-ponens by metis
next
  assume  $\neg[(\exists \alpha . (\alpha^P) = (\lambda x . \varphi x)) \text{ in } v] \wedge [(\exists \beta . (\beta^P) = (\lambda x . \psi x)) \text{ in } v]$ 
  hence  $\neg[(\lambda A! . (\lambda x . \varphi x)) \text{ in } v] \wedge \neg[(\lambda x . \varphi x) =_E (\lambda x . \psi x) \text{ in } v]$ 
     $\vee \neg[(\lambda A! . (\lambda x . \psi x)) \text{ in } v] \wedge \neg[(\lambda x . \varphi x) =_E (\lambda x . \psi x) \text{ in } v]$ 
  unfolding identity $_E$ -infix-def
  using cqt-5[axiom-instance] PLM.contraposition-1 SimpleExOrEnc.intros
    vdash-properties-10 by meson
  hence  $\neg[(\lambda x . \varphi x) = (\lambda x . \psi x) \text{ in } v]$ 
    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]:
   $[\forall \alpha . \varphi \alpha \text{ in } v] \implies [\varphi \beta \text{ in } v]$ 
  by (meson cqt-1[axiom-instance, deduction])
lemmas  $\forall E = \text{rule-ui}$ 

lemma rule-ui-2[PLM, PLM-elim, PLM-dest]:
   $[[\forall \alpha . \varphi (\alpha^P) \text{ in } v]; [\exists \alpha . (\alpha)^P = \beta \text{ in } v]] \implies [\varphi \beta \text{ in } v]$ 
  using cqt-1- $\kappa$ [axiom-instance, deduction, deduction] by blast

lemma cqt-orig-1[PLM]:
   $[(\forall \alpha . \varphi \alpha) \rightarrow \varphi \beta \text{ in } v]$ 
  by PLM-solver
lemma cqt-orig-2[PLM]:
   $[(\forall \alpha . \varphi \rightarrow \psi \alpha) \rightarrow (\varphi \rightarrow (\forall \alpha . \psi \alpha)) \text{ in } v]$ 

```

by *PLM-solver*

lemma *universal*[*PLM*]:
 $(\bigwedge \alpha . [\varphi \ \alpha \text{ in } v]) \implies [\forall \ \alpha . \varphi \ \alpha \text{ in } v]$
 using *rule-gen* .

lemmas $\forall I = \text{universal}$

lemma *cqt-basic-1*[*PLM*]:
 $[(\forall \alpha . (\forall \beta . \varphi \ \alpha \ \beta)) \equiv (\forall \beta . (\forall \alpha . \varphi \ \alpha \ \beta)) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-2*[*PLM*]:
 $[(\forall \alpha . \varphi \ \alpha \equiv \psi \ \alpha) \equiv ((\forall \alpha . \varphi \ \alpha \rightarrow \psi \ \alpha) \ \& \ (\forall \alpha . \psi \ \alpha \rightarrow \varphi \ \alpha)) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-3*[*PLM*]:
 $[(\forall \alpha . \varphi \ \alpha \equiv \psi \ \alpha) \rightarrow ((\forall \alpha . \varphi \ \alpha) \equiv (\forall \alpha . \psi \ \alpha)) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-4*[*PLM*]:
 $[(\forall \alpha . \varphi \ \alpha \ \& \ \psi \ \alpha) \equiv ((\forall \alpha . \varphi \ \alpha) \ \& \ (\forall \alpha . \psi \ \alpha)) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-6*[*PLM*]:
 $[(\forall \alpha . (\forall \alpha . \varphi \ \alpha)) \equiv (\forall \alpha . \varphi \ \alpha) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-7*[*PLM*]:
 $[(\varphi \rightarrow (\forall \alpha . \psi \ \alpha)) \equiv (\forall \alpha . (\varphi \rightarrow \psi \ \alpha)) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-8*[*PLM*]:
 $[((\forall \alpha . \varphi \ \alpha) \vee (\forall \alpha . \psi \ \alpha)) \rightarrow (\forall \alpha . (\varphi \ \alpha \vee \psi \ \alpha)) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-9*[*PLM*]:
 $[((\forall \alpha . \varphi \ \alpha \rightarrow \psi \ \alpha) \ \& \ (\forall \alpha . \psi \ \alpha \rightarrow \chi \ \alpha)) \rightarrow (\forall \alpha . \varphi \ \alpha \rightarrow \chi \ \alpha) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-10*[*PLM*]:
 $[((\forall \alpha . \varphi \ \alpha \equiv \psi \ \alpha) \ \& \ (\forall \alpha . \psi \ \alpha \equiv \chi \ \alpha)) \rightarrow (\forall \alpha . \varphi \ \alpha \equiv \chi \ \alpha) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-11*[*PLM*]:
 $[(\forall \alpha . \varphi \ \alpha \equiv \psi \ \alpha) \equiv (\forall \alpha . \psi \ \alpha \equiv \varphi \ \alpha) \text{ in } v]$
 by *PLM-solver*

lemma *cqt-basic-12*[*PLM*]:
 $[(\forall \alpha . \varphi \ \alpha) \equiv (\forall \beta . \varphi \ \beta) \text{ in } v]$
 by *PLM-solver*

lemma *existential*[*PLM*,*PLM-intro*]:
 $[\varphi \ \alpha \text{ in } v] \implies [\exists \ \alpha . \varphi \ \alpha \text{ in } v]$
 unfolding *exists-def* by *PLM-solver*

lemmas $\exists I = \text{existential}$

lemma *instantiation*-[*PLM*,*PLM-elim*,*PLM-dest*]:
 $[[\exists \alpha . \varphi \ \alpha \text{ in } v]; (\bigwedge \alpha . [\varphi \ \alpha \text{ in } v] \implies [\psi \text{ in } v])] \implies [\psi \text{ in } v]$
 unfolding *exists-def* by *PLM-solver*

lemma *Instantiate*:
 assumes $[\exists \ x . \varphi \ x \text{ in } v]$
 obtains x where $[\varphi \ x \text{ in } v]$
 apply (*insert assms*) unfolding *exists-def* by *PLM-solver*

lemmas $\exists E = \text{Instantiate}$

lemma *cqt-further-1*[*PLM*]:
 $[(\forall \alpha . \varphi \ \alpha) \rightarrow (\exists \alpha . \varphi \ \alpha) \text{ in } v]$
 by *PLM-solver*

```

lemma cqt-further-2[PLM]:
   $[(\neg(\forall \alpha. \varphi \alpha)) \equiv (\exists \alpha. \neg \varphi \alpha) \text{ in } v]$ 
  unfolding exists-def by PLM-solver
lemma cqt-further-3[PLM]:
   $[(\forall \alpha. \varphi \alpha) \equiv \neg(\exists \alpha. \neg \varphi \alpha) \text{ in } v]$ 
  unfolding exists-def by PLM-solver
lemma cqt-further-4[PLM]:
   $[(\neg(\exists \alpha. \varphi \alpha)) \equiv (\forall \alpha. \neg \varphi \alpha) \text{ in } v]$ 
  unfolding exists-def by PLM-solver
lemma cqt-further-5[PLM]:
   $[(\exists \alpha. \varphi \alpha \ \& \ \psi \alpha) \rightarrow ((\exists \alpha. \varphi \alpha) \ \& \ (\exists \alpha. \psi \alpha)) \text{ in } v]$ 
  unfolding exists-def by PLM-solver
lemma cqt-further-6[PLM]:
   $[(\exists \alpha. \varphi \alpha \vee \psi \alpha) \equiv ((\exists \alpha. \varphi \alpha) \vee (\exists \alpha. \psi \alpha)) \text{ in } v]$ 
  unfolding exists-def by PLM-solver
lemma cqt-further-10[PLM]:
   $[(\varphi(\alpha::'a::id\text{-eq}) \ \& \ (\forall \beta. \varphi \beta \rightarrow \beta = \alpha)) \equiv (\forall \beta. \varphi \beta \equiv \beta = \alpha) \text{ in } v]$ 
  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]:
   $[(\forall \alpha. \varphi \alpha \ \& \ (\forall \alpha. \psi \alpha)) \rightarrow (\forall \alpha. \varphi \alpha \equiv \psi \alpha) \text{ in } v]$ 
  by PLM-solver
lemma cqt-further-12[PLM]:
   $[(\neg(\exists \alpha. \varphi \alpha) \ \& \ (\neg(\exists \alpha. \psi \alpha))) \rightarrow (\forall \alpha. \varphi \alpha \equiv \psi \alpha) \text{ in } v]$ 
  unfolding exists-def by PLM-solver
lemma cqt-further-13[PLM]:
   $[(\exists \alpha. \varphi \alpha \ \& \ (\neg(\exists \alpha. \psi \alpha))) \rightarrow (\neg(\forall \alpha. \varphi \alpha \equiv \psi \alpha)) \text{ in } v]$ 
  unfolding exists-def by PLM-solver
lemma cqt-further-14[PLM]:
   $[(\exists \alpha. \exists \beta. \varphi \alpha \beta) \equiv (\exists \beta. \exists \alpha. \varphi \alpha \beta) \text{ in } v]$ 
  unfolding exists-def by PLM-solver

lemma nec-exist-unique[PLM]:
   $[(\forall x. \varphi x \rightarrow \Box(\varphi x)) \rightarrow ((\exists !x. \varphi x) \rightarrow (\exists !x. \Box(\varphi x))) \text{ in } v]$ 
  proof (rule CP)
    assume a:  $[\forall x. \varphi x \rightarrow \Box \varphi x \text{ in } v]$ 
    show  $[(\exists !x. \varphi x) \rightarrow (\exists !x. \Box \varphi x) \text{ in } v]$ 
    proof (rule CP)
      assume  $[(\exists !x. \varphi x) \text{ in } v]$ 
      hence  $[\exists \alpha. \varphi \alpha \ \& \ (\forall \beta. \varphi \beta \rightarrow \beta = \alpha) \text{ in } v]$ 
        by (simp only: exists-unique-def)
      then obtain  $\alpha$  where 1:
         $[\varphi \alpha \ \& \ (\forall \beta. \varphi \beta \rightarrow \beta = \alpha) \text{ in } v]$ 
        by (rule  $\exists E$ )
      {
        fix  $\beta$ 
        have  $[\Box \varphi \beta \rightarrow \beta = \alpha \text{ in } v]$ 
          using 1 & E(2) qml-2[axiom-instance]
          ded-thm-cor-3  $\forall E$  by fastforce
      }
      hence  $[\forall \beta. \Box \varphi \beta \rightarrow \beta = \alpha \text{ in } v]$  by (rule  $\forall I$ )
      moreover have  $[\Box(\varphi \alpha) \text{ in } v]$ 
        using 1 & E(1) a vdash-properties-10 cqt-orig-1[deduction]
        by fast
      ultimately have  $[\exists \alpha. \Box(\varphi \alpha) \ \& \ (\forall \beta. \Box \varphi \beta \rightarrow \beta = \alpha) \text{ in } v]$ 
        using &I  $\exists I$  by fast

```

```

    thus  $[(\exists !x. \Box \varphi x) \text{ in } v]$ 
    unfolding exists-unique-def by assumption
qed
qed

```

A.9.9. Actuality and Descriptions

```

lemma nec-imp-act[PLM]:  $[\Box \varphi \rightarrow \mathcal{A}\varphi \text{ in } v]$ 
  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]:
   $[\mathcal{A}(\mathcal{A}\varphi \rightarrow \varphi) \text{ in } v]$ 
  using equiv-def logic-actual-nec-2[axiom-instance]
    logic-actual-nec-4[axiom-instance] & E(2)  $\equiv$  E(2)
  by metis
lemma act-conj-act-2[PLM]:
   $[\mathcal{A}(\varphi \rightarrow \mathcal{A}\varphi) \text{ in } v]$ 
  using logic-actual-nec-2[axiom-instance] qml-act-1[axiom-instance]
    ded-thm-cor-3  $\equiv$  E(2) nec-imp-act
  by blast
lemma act-conj-act-3[PLM]:
   $[(\mathcal{A}\varphi \ \& \ \mathcal{A}\psi) \rightarrow \mathcal{A}(\varphi \ \& \ \psi) \text{ in } v]$ 
  unfolding conn-defs
  by (metis logic-actual-nec-2[axiom-instance]
    logic-actual-nec-1[axiom-instance]
     $\equiv$  E(2) CP  $\equiv$  E(4) reductio-aa-2
    vdash-properties-10)
lemma act-conj-act-4[PLM]:
   $[\mathcal{A}(\mathcal{A}\varphi \equiv \varphi) \text{ in } v]$ 
  unfolding equiv-def
  by (PLM-solver PLM-intro: act-conj-act-3[where  $\varphi = \mathcal{A}\varphi \rightarrow \varphi$ 
    and  $\psi = \varphi \rightarrow \mathcal{A}\varphi$ , deduction])
lemma closure-act-1a[PLM]:
   $[\mathcal{A}\mathcal{A}(\mathcal{A}\varphi \equiv \varphi) \text{ in } v]$ 
  using logic-actual-nec-4[axiom-instance]
    act-conj-act-4  $\equiv$  E(1)
  by blast
lemma closure-act-1b[PLM]:
   $[\mathcal{A}\mathcal{A}\mathcal{A}(\mathcal{A}\varphi \equiv \varphi) \text{ in } v]$ 
  using logic-actual-nec-4[axiom-instance]
    act-conj-act-4  $\equiv$  E(1)
  by blast
lemma closure-act-1c[PLM]:
   $[\mathcal{A}\mathcal{A}\mathcal{A}\mathcal{A}(\mathcal{A}\varphi \equiv \varphi) \text{ in } v]$ 
  using logic-actual-nec-4[axiom-instance]
    act-conj-act-4  $\equiv$  E(1)
  by blast
lemma closure-act-2[PLM]:
   $[\forall \alpha. \mathcal{A}(\mathcal{A}(\varphi \ \alpha) \equiv \varphi \ \alpha) \text{ in } v]$ 
  by PLM-solver
lemma closure-act-3[PLM]:
   $[\mathcal{A}(\forall \alpha. \mathcal{A}(\varphi \ \alpha) \equiv \varphi \ \alpha) \text{ in } v]$ 
  by (PLM-solver PLM-intro: logic-actual-nec-3[axiom-instance, equiv-rl])
lemma closure-act-4[PLM]:

```


$[\mathcal{A}(\forall \alpha_1 \alpha_2. \mathcal{A}(\varphi \alpha_1 \alpha_2) \equiv \varphi \alpha_1 \alpha_2) \text{ in } v]$
 by (*PLM-solver PLM-intro: logic-actual-nec-3[axiom-instance, equiv-rl]*)
lemma *closure-act-4-b[PLM]*:
 $[\mathcal{A}(\forall \alpha_1 \alpha_2 \alpha_3. \mathcal{A}(\varphi \alpha_1 \alpha_2 \alpha_3) \equiv \varphi \alpha_1 \alpha_2 \alpha_3) \text{ in } v]$
 by (*PLM-solver PLM-intro: logic-actual-nec-3[axiom-instance, equiv-rl]*)
lemma *closure-act-4-c[PLM]*:
 $[\mathcal{A}(\forall \alpha_1 \alpha_2 \alpha_3 \alpha_4. \mathcal{A}(\varphi \alpha_1 \alpha_2 \alpha_3 \alpha_4) \equiv \varphi \alpha_1 \alpha_2 \alpha_3 \alpha_4) \text{ in } v]$
 by (*PLM-solver PLM-intro: logic-actual-nec-3[axiom-instance, equiv-rl]*)

lemma *RA[PLM,PLM-intro]*:
 $([\varphi \text{ in } dw]) \implies [\mathcal{A}\varphi \text{ in } dw]$
 using *logic-actual[necessitation-averse-axiom-instance, equiv-rl]* .

lemma *RA-2[PLM,PLM-intro]*:
 $([\psi \text{ in } dw] \implies [\varphi \text{ in } dw]) \implies ([\mathcal{A}\psi \text{ in } dw] \implies [\mathcal{A}\varphi \text{ in } dw])$
 using *RA logic-actual[necessitation-averse-axiom-instance] intro-elim-6-a* by *blast*

context
begin
private lemma *ActualE[PLM,PLM-elim,PLM-dest]*:
 $[\mathcal{A}\varphi \text{ in } dw] \implies [\varphi \text{ in } dw]$
 using *logic-actual[necessitation-averse-axiom-instance, equiv-lr]* .

private lemma *NotActualD[PLM-dest]*:
 $\neg[\mathcal{A}\varphi \text{ in } dw] \implies \neg[\varphi \text{ in } dw]$
 using *RA* by *metis*

private lemma *ActualImplI[PLM-intro]*:
 $[\mathcal{A}\varphi \rightarrow \mathcal{A}\psi \text{ in } v] \implies [\mathcal{A}(\varphi \rightarrow \psi) \text{ in } v]$
 using *logic-actual-nec-2[axiom-instance, equiv-rl]* .
private lemma *ActualImplE[PLM-dest, PLM-elim]*:
 $[\mathcal{A}(\varphi \rightarrow \psi) \text{ in } v] \implies [\mathcal{A}\varphi \rightarrow \mathcal{A}\psi \text{ in } v]$
 using *logic-actual-nec-2[axiom-instance, equiv-lr]* .
private lemma *NotActualImplD[PLM-dest]*:
 $\neg[\mathcal{A}(\varphi \rightarrow \psi) \text{ in } v] \implies \neg[\mathcal{A}\varphi \rightarrow \mathcal{A}\psi \text{ in } v]$
 using *ActualImplI* by *blast*

private lemma *ActualNotI[PLM-intro]*:
 $[\neg\mathcal{A}\varphi \text{ in } v] \implies [\mathcal{A}\neg\varphi \text{ in } v]$
 using *logic-actual-nec-1[axiom-instance, equiv-rl]* .
lemma *ActualNotE[PLM-elim,PLM-dest]*:
 $[\mathcal{A}\neg\varphi \text{ in } v] \implies [\neg\mathcal{A}\varphi \text{ in } v]$
 using *logic-actual-nec-1[axiom-instance, equiv-lr]* .
lemma *NotActualNotD[PLM-dest]*:
 $\neg[\mathcal{A}\neg\varphi \text{ in } v] \implies \neg[\neg\mathcal{A}\varphi \text{ in } v]$
 using *ActualNotI* by *blast*

private lemma *ActualConjI[PLM-intro]*:
 $[\mathcal{A}\varphi \ \& \ \mathcal{A}\psi \text{ in } v] \implies [\mathcal{A}(\varphi \ \& \ \psi) \text{ in } v]$
 unfolding *equiv-def*
 by (*PLM-solver PLM-intro: act-conj-act-3[deduction]*)
private lemma *ActualConjE[PLM-elim,PLM-dest]*:
 $[\mathcal{A}(\varphi \ \& \ \psi) \text{ in } v] \implies [\mathcal{A}\varphi \ \& \ \mathcal{A}\psi \text{ in } v]$
 unfolding *conj-def* by *PLM-solver*

private lemma *ActualEquivI[PLM-intro]*:
 $[\mathcal{A}\varphi \equiv \mathcal{A}\psi \text{ in } v] \implies [\mathcal{A}(\varphi \equiv \psi) \text{ in } v]$
 unfolding *equiv-def*

by (*PLM-solver PLM-intro: act-conj-act-3[deduction]*)
private lemma *ActualEquivE*[*PLM-elim, PLM-dest*]:
 $[\mathcal{A}(\varphi \equiv \psi) \text{ in } v] \implies [\mathcal{A}\varphi \equiv \mathcal{A}\psi \text{ in } v]$
 unfolding *equiv-def* by *PLM-solver*

private lemma *ActualBoxI*[*PLM-intro*]:
 $[\Box\varphi \text{ in } v] \implies [\mathcal{A}(\Box\varphi) \text{ in } v]$
 using *qml-act-2[axiom-instance, equiv-lr]* .
private lemma *ActualBoxE*[*PLM-elim, PLM-dest*]:
 $[\mathcal{A}(\Box\varphi) \text{ in } v] \implies [\Box\varphi \text{ in } v]$
 using *qml-act-2[axiom-instance, equiv-rl]* .
private lemma *NotActualBoxD*[*PLM-dest*]:
 $\neg[\mathcal{A}(\Box\varphi) \text{ in } v] \implies \neg[\Box\varphi \text{ in } v]$
 using *ActualBoxI* by *blast*

private lemma *ActualDisjI*[*PLM-intro*]:
 $[\mathcal{A}\varphi \vee \mathcal{A}\psi \text{ in } v] \implies [\mathcal{A}(\varphi \vee \psi) \text{ in } v]$
 unfolding *disj-def* by *PLM-solver*
private lemma *ActualDisjE*[*PLM-elim, PLM-dest*]:
 $[\mathcal{A}(\varphi \vee \psi) \text{ in } v] \implies [\mathcal{A}\varphi \vee \mathcal{A}\psi \text{ in } v]$
 unfolding *disj-def* by *PLM-solver*
private lemma *NotActualDisjD*[*PLM-dest*]:
 $\neg[\mathcal{A}(\varphi \vee \psi) \text{ in } v] \implies \neg[\mathcal{A}\varphi \vee \mathcal{A}\psi \text{ in } v]$
 using *ActualDisjI* by *blast*

private lemma *ActualForallI*[*PLM-intro*]:
 $[\forall x . \mathcal{A}(\varphi x) \text{ in } v] \implies [\mathcal{A}(\forall x . \varphi x) \text{ in } v]$
 using *logic-actual-nec-3[axiom-instance, equiv-rl]* .
lemma *ActualForallE*[*PLM-elim, PLM-dest*]:
 $[\mathcal{A}(\forall x . \varphi x) \text{ in } v] \implies [\forall x . \mathcal{A}(\varphi x) \text{ in } v]$
 using *logic-actual-nec-3[axiom-instance, equiv-lr]* .
lemma *NotActualForallD*[*PLM-dest*]:
 $\neg[\mathcal{A}(\forall x . \varphi x) \text{ in } v] \implies \neg[\forall x . \mathcal{A}(\varphi x) \text{ in } v]$
 using *ActualForallI* by *blast*

lemma *ActualActualI*[*PLM-intro*]:
 $[\mathcal{A}\varphi \text{ in } v] \implies [\mathcal{A}\mathcal{A}\varphi \text{ in } v]$
 using *logic-actual-nec-4[axiom-instance, equiv-lr]* .
lemma *ActualActualE*[*PLM-elim, PLM-dest*]:
 $[\mathcal{A}\mathcal{A}\varphi \text{ in } v] \implies [\mathcal{A}\varphi \text{ in } v]$
 using *logic-actual-nec-4[axiom-instance, equiv-rl]* .
lemma *NotActualActualD*[*PLM-dest*]:
 $\neg[\mathcal{A}\mathcal{A}\varphi \text{ in } v] \implies \neg[\mathcal{A}\varphi \text{ in } v]$
 using *ActualActualI* by *blast*

end

lemma *ANeg-1*[*PLM*]:
 $[\neg\mathcal{A}\varphi \equiv \neg\varphi \text{ in } dw]$
 by *PLM-solver*
lemma *ANeg-2*[*PLM*]:
 $[\neg\mathcal{A}\neg\varphi \equiv \varphi \text{ in } dw]$
 by *PLM-solver*
lemma *Act-Basic-1*[*PLM*]:
 $[\mathcal{A}\varphi \vee \mathcal{A}\neg\varphi \text{ in } v]$
 by *PLM-solver*
lemma *Act-Basic-2*[*PLM*]:
 $[\mathcal{A}(\varphi \& \psi) \equiv (\mathcal{A}\varphi \& \mathcal{A}\psi) \text{ in } v]$
 by *PLM-solver*

lemma *Act-Basic-3*[PLM]:
 $[\mathcal{A}(\varphi \equiv \psi) \equiv ((\mathcal{A}(\varphi \rightarrow \psi)) \ \& \ (\mathcal{A}(\psi \rightarrow \varphi))) \text{ in } v]$
by *PLM-solver*

lemma *Act-Basic-4*[PLM]:
 $[(\mathcal{A}(\varphi \rightarrow \psi) \ \& \ \mathcal{A}(\psi \rightarrow \varphi)) \equiv (\mathcal{A}\varphi \equiv \mathcal{A}\psi) \text{ in } v]$
by *PLM-solver*

lemma *Act-Basic-5*[PLM]:
 $[\mathcal{A}(\varphi \equiv \psi) \equiv (\mathcal{A}\varphi \equiv \mathcal{A}\psi) \text{ in } v]$
by *PLM-solver*

lemma *Act-Basic-6*[PLM]:
 $[\Diamond\varphi \equiv \mathcal{A}(\Diamond\varphi) \text{ in } v]$
unfolding *diamond-def* **by** *PLM-solver*

lemma *Act-Basic-7*[PLM]:
 $[\mathcal{A}\varphi \equiv \Box\mathcal{A}\varphi \text{ in } v]$
by (*simp add: qml-2[axiom-instance] qml-act-1[axiom-instance] $\equiv I$*)

lemma *Act-Basic-8*[PLM]:
 $[\mathcal{A}(\Box\varphi) \rightarrow \Box\mathcal{A}\varphi \text{ in } v]$
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]:
 $[\Box\varphi \rightarrow \Box\mathcal{A}\varphi \text{ in } v]$
using *qml-act-1[axiom-instance] ded-thm-cor-3 nec-imp-act* **by** *blast*

lemma *Act-Basic-10*[PLM]:
 $[\mathcal{A}(\varphi \vee \psi) \equiv \mathcal{A}\varphi \vee \mathcal{A}\psi \text{ in } v]$
by *PLM-solver*

lemma *Act-Basic-11*[PLM]:
 $[\mathcal{A}(\exists \alpha. \varphi \ \alpha) \equiv (\exists \alpha. \mathcal{A}(\varphi \ \alpha)) \text{ in } v]$
proof –
have $[\mathcal{A}(\forall \alpha. \neg \varphi \ \alpha) \equiv (\forall \alpha. \mathcal{A}\neg \varphi \ \alpha) \text{ in } v]$
using *logic-actual-nec-3[axiom-instance]* **by** *blast*
hence $[\neg \mathcal{A}(\forall \alpha. \neg \varphi \ \alpha) \equiv \neg(\forall \alpha. \mathcal{A}\neg \varphi \ \alpha) \text{ in } v]$
using *oth-class-taut-5-d[equiv-lr]* **by** *blast*
moreover have $[\mathcal{A}\neg(\forall \alpha. \neg \varphi \ \alpha) \equiv \neg \mathcal{A}(\forall \alpha. \neg \varphi \ \alpha) \text{ in } v]$
using *logic-actual-nec-1[axiom-instance]* **by** *blast*
ultimately have $[\mathcal{A}\neg(\forall \alpha. \neg \varphi \ \alpha) \equiv \neg(\forall \alpha. \mathcal{A}\neg \varphi \ \alpha) \text{ in } v]$
using $\equiv E(5)$ **by** *auto*
moreover {
have $[\forall \alpha. \mathcal{A}\neg \varphi \ \alpha \equiv \neg \mathcal{A}\varphi \ \alpha \text{ in } v]$
using *logic-actual-nec-1[axiom-universal, axiom-instance]* **by** *blast*
hence $[(\forall \alpha. \mathcal{A}\neg \varphi \ \alpha) \equiv (\forall \alpha. \neg \mathcal{A}\varphi \ \alpha) \text{ in } v]$
using *cqt-basic-3[deduction]* **by** *fast*
hence $[(\neg(\forall \alpha. \mathcal{A}\neg \varphi \ \alpha)) \equiv \neg(\forall \alpha. \neg \mathcal{A}\varphi \ \alpha) \text{ in } v]$
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]:
 $[(\forall z. \mathcal{A}\varphi z \equiv z = x) \equiv (\forall z. \varphi z \equiv z = x) \text{ in } dw]$
by *PLM-solver*

lemma *fund-cont-desc*[PLM]:
 $[(x^P = (\iota x. \varphi x)) \equiv (\forall z. \varphi z \equiv (z = x)) \text{ in } dw]$
using *descriptions[axiom-instance] act-quant-uniq $\equiv E(5)$* **by** *fast*

lemma *hintikka*[PLM]:
 $[(x^P = (\iota x. \varphi x)) \equiv (\varphi x \ \& \ (\forall z. \varphi z \rightarrow z = x)) \text{ in } dw]$

proof –

have $[(\forall z . \varphi z \equiv z = x) \equiv (\varphi x \ \& \ (\forall z . \varphi z \rightarrow z = x)) \text{ in } dw]$
unfolding *identity- ν -def* **apply** *PLM-solver* **using** *id-eq-obj-1* **apply** *simp*
using *l-identity* **[where** $\varphi = \lambda x . \varphi x$, *axiom-instance*,
deduction, *deduction*]
using *id-eq-obj-2* **[deduction]** **unfolding** *identity- ν -def* **by** *fastforce*
thus *?thesis* **using** $\equiv E(5)$ *fund-cont-desc* **by** *blast*
qed

lemma *russell-axiom-a* [*PLM*]:

$[(\langle F, \iota x. \varphi x \rangle) \equiv (\exists x . \varphi x \ \& \ (\forall z . \varphi z \rightarrow z = x) \ \& \ (\langle F, x^P \rangle)) \text{ in } dw]$
(is $[?lhs \equiv ?rhs \text{ in } dw]$ **)**

proof –

{
assume $1: [?lhs \text{ in } dw]$
hence $\exists \alpha. \alpha^P = (\iota x. \varphi x) \text{ in } dw]$
using *cqt-5* **[axiom-instance**, *deduction*]
SimpleExOrEnc.intros
by *blast*
then obtain α **where** $2:$
 $[\alpha^P = (\iota x. \varphi x) \text{ in } dw]$
using $\exists E$ **by** *auto*
hence $3: [\varphi \alpha \ \& \ (\forall z . \varphi z \rightarrow z = \alpha) \text{ in } dw]$
using *hintikka* **[equiv-lr]** **by** *simp*
from 2 **have** $[(\iota x. \varphi x) = (\alpha^P) \text{ in } dw]$
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 $[(\langle F, \alpha^P \rangle) \text{ in } dw]$
using *l-identity* **[where** $\beta = \alpha^P$ **and** $\alpha = \iota x. \varphi x$ **and** $\varphi = \lambda x . (\langle F, x \rangle)$,
axiom-instance, *deduction*, *deduction*] **by** *auto*
with 3 **have** $[\varphi \alpha \ \& \ (\forall z . \varphi z \rightarrow z = \alpha) \ \& \ (\langle F, \alpha^P \rangle) \text{ in } dw]$ **by** *(rule &I)*
hence $[?rhs \text{ in } dw]$ **using** $\exists I$ **[where** $\alpha = \alpha$ **by** *simp*
}
moreover {
assume $[?rhs \text{ in } dw]$
then obtain α **where** $4:$
 $[\varphi \alpha \ \& \ (\forall z . \varphi z \rightarrow z = \alpha) \ \& \ (\langle F, \alpha^P \rangle) \text{ in } dw]$
using $\exists E$ **by** *auto*
hence $[\alpha^P = (\iota x . \varphi x) \text{ in } dw] \wedge [(\langle F, \alpha^P \rangle) \text{ in } dw]$
using *hintikka* **[equiv-rl]** **&E** **by** *blast*
hence $[?lhs \text{ in } dw]$
using *l-identity* **[axiom-instance**, *deduction*, *deduction*]
by *blast*
}
ultimately show *?thesis* **by** *PLM-solver*
qed

lemma *russell-axiom-g* [*PLM*]:

$[(\langle \iota x. \varphi x, F \rangle) \equiv (\exists x . \varphi x \ \& \ (\forall z . \varphi z \rightarrow z = x) \ \& \ \langle x^P, F \rangle) \text{ in } dw]$
(is $[?lhs \equiv ?rhs \text{ in } dw]$ **)**

proof –

{
assume $1: [?lhs \text{ in } dw]$
hence $\exists \alpha. \alpha^P = (\iota x. \varphi x) \text{ in } dw]$
using *cqt-5* **[axiom-instance**, *deduction*]
SimpleExOrEnc.intros **by** *blast*
then obtain α **where** $2: [\alpha^P = (\iota x. \varphi x) \text{ in } dw]$ **by** *(rule $\exists E$)*
hence $3: [(\varphi \alpha \ \& \ (\forall z . \varphi z \rightarrow z = \alpha)) \text{ in } dw]$

```

    using hintikka[equiv-lr] by simp
  from 2 have  $[(\iota x. \varphi x) = \alpha^P \text{ in } dw]$ 
    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  $[\{\alpha^P, F\} \text{ in } dw]$ 
  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  $[(\varphi \alpha \ \& \ (\forall z. \varphi z \rightarrow z = \alpha)) \ \& \ \{\alpha^P, F\} \text{ in } dw]$ 
    using &I by auto
  hence  $[?rhs \text{ in } dw]$  using  $\exists I$ [where  $\alpha = \alpha$ ] by (simp add: identity-defs)
}
moreover {
  assume  $[?rhs \text{ in } dw]$ 
  then obtain  $\alpha$  where 4:
     $[\varphi \alpha \ \& \ (\forall z. \varphi z \rightarrow z = \alpha) \ \& \ \{\alpha^P, F\} \text{ in } dw]$ 
    using  $\exists E$  by auto
  hence  $[\alpha^P = (\iota x. \varphi x) \text{ in } dw] \wedge [\{\alpha^P, F\} \text{ in } dw]$ 
    using hintikka[equiv-rl] &E by blast
  hence  $[?lhs \text{ in } dw]$ 
    using l-identity[axiom-instance, deduction, deduction]
    by fast
}
ultimately show ?thesis by PLM-solver
qed

```

lemma russell-axiom[PLM]:

assumes SimpleExOrEnc ψ

shows $[\psi (\iota x. \varphi x) \equiv (\exists x. \varphi x \ \& \ (\forall z. \varphi z \rightarrow z = x) \ \& \ \psi (x^P)) \text{ in } dw]$
 (is $[?lhs \equiv ?rhs \text{ in } dw]$)

proof –

```

{
  assume 1:  $[?lhs \text{ in } dw]$ 
  hence  $[\exists \alpha. \alpha^P = (\iota x. \varphi x) \text{ in } dw]$ 
    using cqt-5[axiom-instance, deduction] assms by blast
  then obtain  $\alpha$  where 2:  $[\alpha^P = (\iota x. \varphi x) \text{ in } dw]$  by (rule  $\exists E$ )
  hence 3:  $[(\varphi \alpha \ \& \ (\forall z. \varphi z \rightarrow z = \alpha)) \text{ in } dw]$ 
    using hintikka[equiv-lr] by simp
  from 2 have  $[(\iota x. \varphi x) = (\alpha^P) \text{ in } dw]$ 
    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  $[\psi (\alpha^P) \text{ in } dw]$ 
    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  $[\varphi \alpha \ \& \ (\forall z. \varphi z \rightarrow z = \alpha) \ \& \ \psi (\alpha^P) \text{ in } dw]$ 
    using &I by auto
  hence  $[?rhs \text{ in } dw]$  using  $\exists I$ [where  $\alpha = \alpha$ ] by (simp add: identity-defs)
}
moreover {
  assume  $[?rhs \text{ in } dw]$ 
  then obtain  $\alpha$  where 4:
     $[\varphi \alpha \ \& \ (\forall z. \varphi z \rightarrow z = \alpha) \ \& \ \psi (\alpha^P) \text{ in } dw]$ 
    using  $\exists E$  by auto
  hence  $[\alpha^P = (\iota x. \varphi x) \text{ in } dw] \wedge [\psi (\alpha^P) \text{ in } dw]$ 
    using hintikka[equiv-rl] &E by blast
  hence  $[?lhs \text{ in } dw]$ 
    using l-identity[axiom-instance, deduction, deduction]

```

by fast
 }
 ultimately show ?thesis by PLM-solver
 qed

lemma *unique-exists*[PLM]:
 $[(\exists y . y^P = (\iota x . \varphi x)) \equiv (\exists !x . \varphi x) \text{ in } dw]$
proof((rule $\equiv I$, rule CP, rule-tac[2] CP))
 assume $[\exists y . y^P = (\iota x . \varphi x) \text{ in } dw]$
 then obtain α where
 $[\alpha^P = (\iota x . \varphi x) \text{ in } dw]$
 by (rule $\exists E$)
 hence $[\varphi \alpha \ \& \ (\forall \beta . \varphi \beta \rightarrow \beta = \alpha) \text{ in } dw]$
 using *hintikka*[equiv-lr] by auto
 thus $[\exists !x . \varphi x \text{ in } dw]$
 unfolding *exists-unique-def* using $\exists I$ by fast
next
 assume $[\exists !x . \varphi x \text{ in } dw]$
 then obtain α where
 $[\varphi \alpha \ \& \ (\forall \beta . \varphi \beta \rightarrow \beta = \alpha) \text{ in } dw]$
 unfolding *exists-unique-def* by (rule $\exists E$)
 hence $[\alpha^P = (\iota x . \varphi x) \text{ in } dw]$
 using *hintikka*[equiv-rl] by auto
 thus $[\exists y . y^P = (\iota x . \varphi x) \text{ in } dw]$
 using $\exists I$ by fast
 qed

lemma *y-in-1*[PLM]:
 $[x^P = (\iota x . \varphi) \rightarrow \varphi \text{ in } dw]$
 using *hintikka*[equiv-lr, conj1] by (rule CP)

lemma *y-in-2*[PLM]:
 $[z^P = (\iota x . \varphi x) \rightarrow \varphi z \text{ in } dw]$
 using *hintikka*[equiv-lr, conj1] by (rule CP)

lemma *y-in-3*[PLM]:
 $[(\exists y . y^P = (\iota x . \varphi (x^P))) \rightarrow \varphi (\iota x . \varphi (x^P)) \text{ in } dw]$
proof (rule CP)
 assume $[(\exists y . y^P = (\iota x . \varphi (x^P))) \text{ in } dw]$
 then obtain y where 1:
 $[y^P = (\iota x . \varphi (x^P)) \text{ in } dw]$
 by (rule $\exists E$)
 hence $[\varphi (y^P) \text{ in } dw]$
 using *y-in-2*[deduction] unfolding *identity- ν -def* by blast
 thus $[\varphi (\iota x . \varphi (x^P)) \text{ in } dw]$
 using *l-identity*[axiom-instance, deduction,
 deduction] 1 by fast
 qed

lemma *act-quant-nec*[PLM]:
 $[(\forall z . (\mathcal{A}\varphi z \equiv z = x)) \equiv (\forall z . \mathcal{A}\mathcal{A}\varphi z \equiv z = x) \text{ in } v]$
 by PLM-solver

lemma *equi-desc-descA-1*[PLM]:
 $[(x^P = (\iota x . \varphi x)) \equiv (x^P = (\iota x . \mathcal{A}\varphi x)) \text{ in } v]$
 using *descriptions*[axiom-instance] apply (rule $\equiv E(5)$)
 using *act-quant-nec* apply (rule $\equiv E(5)$)
 using *descriptions*[axiom-instance]

by (*meson* $\equiv E(6)$ *oth-class-taut-4-a*)

lemma *equi-desc-descA-2*[PLM]:

$[(\exists y . y^P = (\iota x . \varphi x)) \rightarrow ((\iota x . \varphi x) = (\iota x . \mathcal{A}\varphi x)) \text{ in } v]$

proof (*rule CP*)

assume $[\exists y . y^P = (\iota x . \varphi x) \text{ in } v]$

then obtain *y* **where**

$[y^P = (\iota x . \varphi x) \text{ in } v]$

by (*rule* $\exists E$)

moreover hence $[y^P = (\iota x . \mathcal{A}\varphi x) \text{ in } v]$

using *equi-desc-descA-1*[*equiv-lr*] **by** *auto*

ultimately show $[(\iota x . \varphi x) = (\iota x . \mathcal{A}\varphi x) \text{ in } v]$

using *l-identity*[*axiom-instance*, *deduction*, *deduction*]

by *fast*

qed

lemma *equi-desc-descA-3*[PLM]:

assumes *SimpleExOrEnc* ψ

shows $[\psi (\iota x . \varphi x) \rightarrow (\exists y . y^P = (\iota x . \mathcal{A}\varphi x)) \text{ in } v]$

proof (*rule CP*)

assume $[\psi (\iota x . \varphi x) \text{ in } v]$

hence $[\exists \alpha . \alpha^P = (\iota x . \varphi x) \text{ in } v]$

using *cqt-5*[*OF assms*, *axiom-instance*, *deduction*] **by** *auto*

then obtain α **where** $[\alpha^P = (\iota x . \varphi x) \text{ in } v]$ **by** (*rule* $\exists E$)

hence $[\alpha^P = (\iota x . \mathcal{A}\varphi x) \text{ in } v]$

using *equi-desc-descA-1*[*equiv-lr*] **by** *auto*

thus $[\exists y . y^P = (\iota x . \mathcal{A}\varphi x) \text{ in } v]$

using $\exists I$ **by** *fast*

qed

lemma *equi-desc-descA-4*[PLM]:

assumes *SimpleExOrEnc* ψ

shows $[\psi (\iota x . \varphi x) \rightarrow ((\iota x . \varphi x) = (\iota x . \mathcal{A}\varphi x)) \text{ in } v]$

proof (*rule CP*)

assume $[\psi (\iota x . \varphi x) \text{ in } v]$

hence $[\exists \alpha . \alpha^P = (\iota x . \varphi x) \text{ in } v]$

using *cqt-5*[*OF assms*, *axiom-instance*, *deduction*] **by** *auto*

then obtain α **where** $[\alpha^P = (\iota x . \varphi x) \text{ in } v]$ **by** (*rule* $\exists E$)

moreover hence $[\alpha^P = (\iota x . \mathcal{A}\varphi x) \text{ in } v]$

using *equi-desc-descA-1*[*equiv-lr*] **by** *auto*

ultimately show $[(\iota x . \varphi x) = (\iota x . \mathcal{A}\varphi x) \text{ in } v]$

using *l-identity*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*

qed

lemma *nec-hintikka-scheme*[PLM]:

$[(x^P = (\iota x . \varphi x)) \equiv (\mathcal{A}\varphi x \ \& \ (\forall z . \mathcal{A}\varphi z \rightarrow z = x)) \text{ in } v]$

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  $\wedge x. [\mathcal{A}(\varphi x \equiv \psi x) \text{ in } v]$ 
shows  $[(\forall x. ((x^P = (\iota x. \varphi x)) \equiv (x^P = (\iota x. \psi x)))) \text{ in } v]$ 
proof(rule  $\forall I$ )
  fix x
  {
    assume  $[x^P = (\iota x. \varphi x) \text{ in } v]$ 
    hence 1:  $[\mathcal{A}\varphi x \ \& \ (\forall z. \mathcal{A}\varphi z \rightarrow z = x) \text{ in } v]$ 
      using nec-hintikka-scheme[equiv-lr] by auto
    hence 2:  $[\mathcal{A}\varphi x \text{ in } v] \wedge [(\forall z. \mathcal{A}\varphi z \rightarrow z = x) \text{ in } v]$ 
      using &E by blast
    {
      fix z
      {
        assume  $[\mathcal{A}\psi z \text{ in } v]$ 
        hence  $[\mathcal{A}\varphi z \text{ in } v]$ 
          using assms[where  $x=z$ ] apply – by PLM-solver
        moreover have  $[\mathcal{A}\varphi z \rightarrow z = x \text{ in } v]$ 
          using 2 cqt-1[axiom-instance,deduction] by auto
        ultimately have  $[z = x \text{ in } v]$ 
          using vdash-properties-10 by auto
      }
      hence  $[\mathcal{A}\psi z \rightarrow z = x \text{ in } v]$  by (rule CP)
    }
    hence  $[(\forall z. \mathcal{A}\psi z \rightarrow z = x) \text{ in } v]$  by (rule  $\forall I$ )
    moreover have  $[\mathcal{A}\psi x \text{ in } v]$ 
      using 1[conj1] assms[where  $x=x$ ]
      apply – by PLM-solver
    ultimately have  $[\mathcal{A}\psi x \ \& \ (\forall z. \mathcal{A}\psi z \rightarrow z = x) \text{ in } v]$ 
      by PLM-solver
    hence  $[x^P = (\iota x. \psi x) \text{ in } v]$ 
      using nec-hintikka-scheme[where  $\varphi=\psi$ , equiv-rl] by auto
  }
  moreover {
    assume  $[x^P = (\iota x. \psi x) \text{ in } v]$ 
    hence 1:  $[\mathcal{A}\psi x \ \& \ (\forall z. \mathcal{A}\psi z \rightarrow z = x) \text{ in } v]$ 
      using nec-hintikka-scheme[equiv-lr] by auto
    hence 2:  $[\mathcal{A}\psi x \text{ in } v] \wedge [(\forall z. \mathcal{A}\psi z \rightarrow z = x) \text{ in } v]$ 
      using &E by blast
    {
      fix z
      {
        assume  $[\mathcal{A}\varphi z \text{ in } v]$ 
        hence  $[\mathcal{A}\psi z \text{ in } v]$ 
          using assms[where  $x=z$ ]
          apply – by PLM-solver
        moreover have  $[\mathcal{A}\psi z \rightarrow z = x \text{ in } v]$ 
          using 2 cqt-1[axiom-instance,deduction] by auto
        ultimately have  $[z = x \text{ in } v]$ 
          using vdash-properties-10 by auto
      }
      hence  $[\mathcal{A}\varphi z \rightarrow z = x \text{ in } v]$  by (rule CP)
    }
    hence  $[(\forall z. \mathcal{A}\varphi z \rightarrow z = x) \text{ in } v]$  by (rule  $\forall I$ )
    moreover have  $[\mathcal{A}\varphi x \text{ in } v]$ 
      using 1[conj1] assms[where  $x=x$ ]
      apply – by PLM-solver
    ultimately have  $[\mathcal{A}\varphi x \ \& \ (\forall z. \mathcal{A}\varphi z \rightarrow z = x) \text{ in } v]$ 
      by PLM-solver
  }

```



```

    hence  $[x^P = (\iota x. \varphi x) \text{ in } v]$ 
      using nec-hintikka-scheme[where  $\varphi = \varphi, \text{equiv-rl}$ ]
      by auto
  }
  ultimately show  $[x^P = (\iota x. \varphi x) \equiv (x^P) = (\iota x. \psi x) \text{ in } v]$ 
    using  $\equiv I$  CP by auto
qed

```

lemma *UniqueAux*:

```

assumes  $[(\mathcal{A}\varphi(\alpha::\nu) \ \& \ (\forall z. \mathcal{A}(\varphi z) \rightarrow z = \alpha)) \text{ in } v]$ 
shows  $[(\forall z. (\mathcal{A}(\varphi z) \equiv (z = \alpha))) \text{ in } v]$ 
proof -
  {
    fix  $z$ 
    {
      assume  $[\mathcal{A}(\varphi z) \text{ in } v]$ 
      hence  $[z = \alpha \text{ in } v]$ 
        using assms[conj2, THEN cqt-1[where  $\alpha = z$ ,
          axiom-instance, deduction],
          deduction] by auto
    }
    moreover {
      assume  $[z = \alpha \text{ in } v]$ 
      hence  $[\alpha = z \text{ in } v]$ 
        unfolding identity- $\nu$ -def
        using id-eq-obj-2[deduction] by fast
      hence  $[\mathcal{A}(\varphi z) \text{ in } v]$  using assms[conj1]
        using l-identity[axiom-instance, deduction,
          deduction] by fast
    }
    ultimately have  $[(\mathcal{A}(\varphi z) \equiv (z = \alpha)) \text{ in } v]$ 
      using  $\equiv I$  CP by auto
  }
  thus  $[(\forall z. (\mathcal{A}(\varphi z) \equiv (z = \alpha))) \text{ in } v]$ 
    by (rule  $\forall I$ )
qed

```

lemma *nec-russell-axiom*[*PLM*]:

```

assumes SimpleExOrEnc  $\psi$ 
shows  $[(\psi(\iota x. \varphi x)) \equiv (\exists x. (\mathcal{A}\varphi x \ \& \ (\forall z. \mathcal{A}(\varphi z) \rightarrow z = x)) \ \& \ \psi(x^P)) \text{ in } v]$ 
(is  $[?lhs \equiv ?rhs \text{ in } v]$ )
proof -
  {
    assume 1:  $[?lhs \text{ in } v]$ 
    hence  $[\exists \alpha. (\alpha^P) = (\iota x. \varphi x) \text{ in } v]$ 
      using cqt-5[axiom-instance, deduction] assms by blast
    then obtain  $\alpha$  where 2:  $[(\alpha^P) = (\iota x. \varphi x) \text{ in } v]$  by (rule  $\exists E$ )
    hence  $[(\forall z. (\mathcal{A}(\varphi z) \equiv (z = \alpha))) \text{ in } v]$ 
      using descriptions[axiom-instance, equiv-lr] by auto
    hence 3:  $[(\mathcal{A}\varphi \alpha) \ \& \ (\forall z. (\mathcal{A}(\varphi z) \rightarrow (z = \alpha))) \text{ in } v]$ 
      using cqt-1[where  $\alpha = \alpha$  and  $\varphi = \lambda z. (\mathcal{A}(\varphi z) \equiv (z = \alpha))$ ,
        axiom-instance, deduction, equiv-rl]
      using id-eq-obj-1[where  $x = \alpha$ ] unfolding identity- $\nu$ -def
      using hintikka[equiv-lr] cqt-basic-2[equiv-lr, conj1]
      & I by fast
    from 2 have  $[(\iota x. \varphi x) = (\alpha^P) \text{ in } v]$ 
      using l-identity[where  $\beta = (\iota x. \varphi x)$  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  $v$ ]
    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  $\alpha$  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  $\alpha$  where
      [ $((\alpha^P) = (\iota x. \varphi x))$  in  $v$ ]
      by (rule  $\exists E$ )
    hence [ $(\lambda A!.( \iota x. \varphi x))$  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  $\psi=\lambda x. (\lambda A!.( \iota x. \varphi x))$ , 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  $\equiv I$  CP by auto
qed

```

```

lemma actual-desc-2[PLM]:
   $[(x^P) = (\iota x. \varphi) \rightarrow \mathcal{A}\varphi \text{ in } v]$ 
  using nec-hintikka-scheme[equiv-lr, conj1]
  by (rule CP)

```

```

lemma actual-desc-3[PLM]:
   $[(z^P) = (\iota x. \varphi x) \rightarrow \mathcal{A}(\varphi z) \text{ in } v]$ 
  using nec-hintikka-scheme[equiv-lr, conj1]
  by (rule CP)

```

```

lemma actual-desc-4[PLM]:
   $[(\exists y. ((y^P) = (\iota x. \varphi (x^P)))) \rightarrow \mathcal{A}(\varphi (\iota x. \varphi (x^P))) \text{ in } v]$ 
  proof (rule CP)
    assume  $[(\exists y. (y^P) = (\iota x. \varphi (x^P))) \text{ in } v]$ 
    then obtain y where 1:
       $[y^P = (\iota x. \varphi (x^P)) \text{ in } v]$ 
      by (rule  $\exists E$ )
    hence  $[\mathcal{A}(\varphi (y^P)) \text{ in } v]$  using actual-desc-3[deduction] by fast
    thus  $[\mathcal{A}(\varphi (\iota x. \varphi (x^P))) \text{ in } v]$ 
      using l-identity[axiom-instance, deduction,
        deduction] 1 by fast
  qed

```

```

lemma unique-box-desc-1[PLM]:
   $[(\exists !x. \Box(\varphi x)) \rightarrow (\forall y. (y^P) = (\iota x. \varphi x) \rightarrow \varphi y) \text{ in } v]$ 
  proof (rule CP)
    assume  $[(\exists !x. \Box(\varphi x)) \text{ in } v]$ 
    then obtain  $\alpha$  where 1:
       $[\Box \varphi \alpha \ \& \ (\forall \beta. \Box(\varphi \beta) \rightarrow \beta = \alpha) \text{ in } v]$ 
      unfolding exists-unique-def by (rule  $\exists E$ )
    {
      fix y
      {
        assume  $[(y^P) = (\iota x. \varphi x) \text{ in } v]$ 
        hence  $[\mathcal{A}\varphi \alpha \rightarrow \alpha = y \text{ in } v]$ 
          using nec-hintikka-scheme[where  $x=y$  and  $\varphi=\varphi$ , equiv-lr, conj2,
            THEN cqt-1[where  $\alpha=\alpha$ , axiom-instance, deduction]] by simp
        hence  $[\alpha = y \text{ in } v]$ 
          using 1[conj1] nec-imp-act vdash-properties-10 by blast
        hence  $[\varphi y \text{ in } v]$ 
          using 1[conj1] qml-2[axiom-instance, deduction]
            l-identity[axiom-instance, deduction, deduction]
          by fast
      }
      hence  $[(y^P) = (\iota x. \varphi x) \rightarrow \varphi y \text{ in } v]$ 
        by (rule CP)
    }
    thus  $[\forall y. (y^P) = (\iota x. \varphi x) \rightarrow \varphi y \text{ in } v]$ 
      by (rule  $\forall I$ )
  qed

```

```

lemma unique-box-desc[PLM]:
   $[(\forall x. (\varphi x \rightarrow \Box(\varphi x))) \rightarrow ((\exists !x. \varphi x)$ 

```

$\rightarrow (\forall y . (y^P = (\iota x . \varphi x)) \rightarrow \varphi y)) \text{ in } v]$
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]:
 $(\bigwedge v. [\varphi \rightarrow \psi \text{ in } v]) \implies [\Box \varphi \rightarrow \Box \psi \text{ in } v]$
using RN qml-1[axiom-instance] vdash-properties-10 **by** blast

lemma RM-1-b[PLM]:
 $(\bigwedge v. [\chi \text{ in } v] \implies [\varphi \rightarrow \psi \text{ in } v]) \implies ([\Box \chi \text{ in } v] \implies [\Box \varphi \rightarrow \Box \psi \text{ in } v])$
using RN-2 qml-1[axiom-instance] vdash-properties-10 **by** blast

lemma RM-2[PLM]:
 $(\bigwedge v. [\varphi \rightarrow \psi \text{ in } v]) \implies [\Diamond \varphi \rightarrow \Diamond \psi \text{ in } v]$
unfolding diamond-def
using RM-1 contraposition-1 **by** auto

lemma RM-2-b[PLM]:
 $(\bigwedge v. [\chi \text{ in } v] \implies [\varphi \rightarrow \psi \text{ in } v]) \implies ([\Box \chi \text{ in } v] \implies [\Diamond \varphi \rightarrow \Diamond \psi \text{ in } v])$
unfolding diamond-def
using RM-1-b contraposition-1 **by** blast

lemma KBasic-1[PLM]:
 $[\Box \varphi \rightarrow \Box(\psi \rightarrow \varphi) \text{ in } v]$
by (simp only: pl-1[axiom-instance] RM-1)

lemma KBasic-2[PLM]:
 $[\Box(\neg \varphi) \rightarrow \Box(\varphi \rightarrow \psi) \text{ in } v]$
by (simp only: RM-1 useful-tautologies-3)

lemma KBasic-3[PLM]:
 $[\Box(\varphi \ \&\ \psi) \equiv \Box \varphi \ \&\ \Box \psi \text{ in } v]$
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]:
 $[\Box(\varphi \equiv \psi) \equiv (\Box(\varphi \rightarrow \psi) \ \&\ \Box(\psi \rightarrow \varphi)) \text{ in } v]$
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]:
 $[(\Box(\varphi \rightarrow \psi) \ \&\ \Box(\psi \rightarrow \varphi)) \rightarrow (\Box \varphi \equiv \Box \psi) \text{ in } v]$
by (metis qml-1[axiom-instance] CP $\&E \equiv I$ vdash-properties-10)

lemma KBasic-6[PLM]:
 $[\Box(\varphi \equiv \psi) \rightarrow (\Box \varphi \equiv \Box \psi) \text{ in } v]$
using KBasic-4 KBasic-5 **by** (metis equiv-def ded-thm-cor-3 $\&E(1)$)

lemma $[(\Box \varphi \equiv \Box \psi) \rightarrow \Box(\varphi \equiv \psi) \text{ in } v]$
nitpick[expect=genuine, user-axioms, card = 1, card i = 2]
oops — countermodel as desired

lemma KBasic-7[PLM]:

$[(\Box\varphi \ \& \ \Box\psi) \rightarrow \Box(\varphi \equiv \psi) \text{ in } v]$
proof (*rule CP*)
 assume $[\Box\varphi \ \& \ \Box\psi \text{ in } v]$
 hence $[\Box(\psi \rightarrow \varphi) \text{ in } v] \wedge [\Box(\varphi \rightarrow \psi) \text{ in } v]$
 using $\&E$ *KBasic-1 vdash-properties-10* **by** *blast*
 thus $[\Box(\varphi \equiv \psi) \text{ in } v]$
 using *KBasic-4 $\equiv E(2)$ intro-elim-1* **by** *blast*
qed

lemma *KBasic-8[PLM]*:
 $[\Box(\varphi \ \& \ \psi) \rightarrow \Box(\varphi \equiv \psi) \text{ in } v]$
 using *KBasic-7 KBasic-3*
by (*metis equiv-def PLM.ded-thm-cor-3 $\&E(1)$*)

lemma *KBasic-9[PLM]*:
 $[\Box((\neg\varphi) \ \& \ (\neg\psi)) \rightarrow \Box(\varphi \equiv \psi) \text{ in } v]$
proof (*rule CP*)
 assume $[\Box((\neg\varphi) \ \& \ (\neg\psi)) \text{ in } v]$
 hence $[\Box((\neg\varphi) \equiv (\neg\psi)) \text{ in } v]$
 using *KBasic-8 vdash-properties-10* **by** *blast*
 moreover have $\bigwedge v. [((\neg\varphi) \equiv (\neg\psi)) \rightarrow (\varphi \equiv \psi) \text{ in } v]$
 using *CP $\equiv E(2)$ oth-class-taut-5-d* **by** *blast*
 ultimately show $[\Box(\varphi \equiv \psi) \text{ in } v]$
 using *RM-1 PLM.vdash-properties-10* **by** *blast*
qed

lemma *rule-sub-lem-1-a[PLM]*:
 $[\Box(\psi \equiv \chi) \text{ in } v] \implies [(\neg\psi) \equiv (\neg\chi) \text{ in } v]$
 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]*:
 $[\Box(\psi \equiv \chi) \text{ in } v] \implies [(\psi \rightarrow \Theta) \equiv (\chi \rightarrow \Theta) \text{ in } v]$
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]*:
 $[\Box(\psi \equiv \chi) \text{ in } v] \implies [(\Theta \rightarrow \psi) \equiv (\Theta \rightarrow \chi) \text{ in } v]$
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]*:
 $(\bigwedge x. [\Box(\psi \ x \equiv \chi \ x) \text{ in } v]) \implies [(\forall \alpha. \psi \ \alpha) \equiv (\forall \alpha. \chi \ \alpha) \text{ in } v]$
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]*:
 $[\Box(\psi \equiv \chi) \text{ in } v] \implies [\mathcal{A}\psi \equiv \mathcal{A}\chi \text{ in } v]$
 using *Act-Basic-5 $\equiv E(1)$ nec-imp-act*
vdash-properties-10
by *blast*

lemma *rule-sub-lem-1-f[PLM]*:
 $[\Box(\psi \equiv \chi) \text{ in } v] \implies [\Box\psi \equiv \Box\chi \text{ in } v]$
 using *KBasic-6 $\equiv I \equiv E(1)$ vdash-properties-9*
by *blast*

named-theorems *Substable-intros*

definition *Substable* :: $('a \Rightarrow 'a \Rightarrow \text{bool}) \Rightarrow ('a \Rightarrow \text{o}) \Rightarrow \text{bool}$
 where *Substable* $\equiv (\lambda \text{ cond } \varphi . \forall \psi \ \chi \ v . (\text{cond } \psi \ \chi) \longrightarrow [\varphi \ \psi \equiv \varphi \ \chi \text{ in } v])$

```

lemma Substable-intro-const[Substable-intros]:
  Substable cond ( $\lambda \varphi . \Theta$ )
  unfolding Substable-def using oth-class-taut-4-a by blast

lemma Substable-intro-not[Substable-intros]:
  assumes Substable cond  $\psi$ 
  shows Substable cond ( $\lambda \varphi . \neg(\psi \varphi)$ )
  using assms unfolding Substable-def
  using rule-sub-lem-1-a RN-2  $\equiv E$  oth-class-taut-5-d by metis
lemma Substable-intro-impl[Substable-intros]:
  assumes Substable cond  $\psi$ 
  and Substable cond  $\chi$ 
  shows Substable cond ( $\lambda \varphi . \psi \varphi \rightarrow \chi \varphi$ )
  using assms unfolding Substable-def
  by (metis  $\equiv I$  CP intro-elim-6-a intro-elim-6-b)
lemma Substable-intro-box[Substable-intros]:
  assumes Substable cond  $\psi$ 
  shows Substable cond ( $\lambda \varphi . \Box(\psi \varphi)$ )
  using assms unfolding Substable-def
  using rule-sub-lem-1-f RN by meson
lemma Substable-intro-actual[Substable-intros]:
  assumes Substable cond  $\psi$ 
  shows Substable cond ( $\lambda \varphi . \mathcal{A}(\psi \varphi)$ )
  using assms unfolding Substable-def
  using rule-sub-lem-1-e RN by meson
lemma Substable-intro-all[Substable-intros]:
  assumes  $\forall x . \text{Substable cond } (\psi x)$ 
  shows Substable cond ( $\lambda \varphi . \forall x . \psi x \varphi$ )
  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 :: ' $a \Rightarrow a \Rightarrow \text{bool}$ '
  assumes rule-sub-nec:
     $\bigwedge \varphi \psi \chi \Theta v . \llbracket \text{PLM.Substable Substable-Cond } \varphi; \text{Substable-Cond } \psi \chi \rrbracket$ 
     $\Rightarrow \Theta [\varphi \psi \text{ in } v] \Rightarrow \Theta [\varphi \chi \text{ in } v]$ 

instantiation o :: Substable
begin
  definition Substable-Cond-o where  $[\text{PLM.Substable-Cond-defs}]$ :
    Substable-Cond-o  $\equiv \lambda \varphi \psi . \forall v . [\varphi \equiv \psi \text{ in } v]$ 
  instance proof
    interpret PLM .
    fix  $\varphi :: o \Rightarrow o$  and  $\psi \chi :: o$  and  $\Theta :: \text{bool} \Rightarrow \text{bool}$  and  $v :: i$ 
    assume Substable Substable-Cond  $\varphi$ 
    moreover assume Substable-Cond  $\psi \chi$ 
    ultimately have  $[\varphi \psi \equiv \varphi \chi \text{ in } v]$ 
    unfolding Substable-def by blast
    hence  $[\varphi \psi \text{ in } v] = [\varphi \chi \text{ in } v]$  using  $\equiv E$  by blast
    moreover assume  $\Theta [\varphi \psi \text{ in } v]$ 
    ultimately show  $\Theta [\varphi \chi \text{ in } v]$  by simp
  qed
end

instantiation fun :: (type, Substable) Substable

```

```

begin
  definition Substable-Cond-fun where [PLM.Substable-Cond-defs]:
    Substable-Cond-fun  $\equiv \lambda \varphi \psi . \forall x . \text{Substable-Cond } (\varphi x) (\psi x)$ 
  instance proof
    interpret PLM .
    fix  $\varphi :: ('a \Rightarrow 'b) \Rightarrow o$  and  $\psi \chi :: 'a \Rightarrow 'b$  and  $\Theta v$ 
    assume Substable Substable-Cond  $\varphi$ 
    moreover assume Substable-Cond  $\psi \chi$ 
    ultimately have  $[\varphi \psi \equiv \varphi \chi \text{ in } v]$ 
      unfolding Substable-def by blast
    hence  $[\varphi \psi \text{ in } v] = [\varphi \chi \text{ in } v]$  using  $\equiv E$  by blast
    moreover assume  $\Theta [\varphi \psi \text{ in } v]$ 
    ultimately show  $\Theta [\varphi \chi \text{ in } v]$  by simp
  qed
end

context PLM
begin

  lemma Substable-intro-equiv[Substable-intros]:
    assumes Substable cond  $\psi$ 
    and Substable cond  $\chi$ 
    shows Substable cond  $(\lambda \varphi . \psi \varphi \equiv \chi \varphi)$ 
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-conj[Substable-intros]:
    assumes Substable cond  $\psi$ 
    and Substable cond  $\chi$ 
    shows Substable cond  $(\lambda \varphi . \psi \varphi \ \& \ \chi \varphi)$ 
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-disj[Substable-intros]:
    assumes Substable cond  $\psi$ 
    and Substable cond  $\chi$ 
    shows Substable cond  $(\lambda \varphi . \psi \varphi \vee \chi \varphi)$ 
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-diamond[Substable-intros]:
    assumes Substable cond  $\psi$ 
    shows Substable cond  $(\lambda \varphi . \Diamond(\psi \varphi))$ 
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-exist[Substable-intros]:
    assumes  $\forall x . \text{Substable cond } (\psi x)$ 
    shows Substable cond  $(\lambda \varphi . \exists x . \psi x \varphi)$ 
    unfolding conn-defs by (simp add: assms Substable-intros)

  lemma Substable-intro-id-o[Substable-intros]:
    Substable Substable-Cond  $(\lambda \varphi . \varphi)$ 
    unfolding Substable-def Substable-Cond-o-def by blast
  lemma Substable-intro-id-fun[Substable-intros]:
    assumes Substable Substable-Cond  $\psi$ 
    shows Substable Substable-Cond  $(\lambda \varphi . \psi (\varphi x))$ 
    using assms unfolding Substable-def Substable-Cond-fun-def
    by blast

  method PLM-subst-method for  $\psi :: 'a :: \text{Substable}$  and  $\chi :: 'a :: \text{Substable} =$ 
    (match conclusion in  $\Theta [\varphi \chi \text{ in } v]$  for  $\Theta$  and  $\varphi$  and  $v \Rightarrow$ 
       $\langle (\text{rule rule-sub-nec}[\text{where } \Theta = \Theta \text{ and } \chi = \chi \text{ and } \psi = \psi \text{ and } \varphi = \varphi \text{ and } v = v],$ 
         $((\text{fast intro: Substable-intros, } ((\text{assumption})+)?) +; \text{fail}),$ 
        unfold Substable-Cond-defs))

```

```

method PLM-autosubst =
  (match premises in  $\bigwedge v . [\psi \equiv \chi \text{ in } v]$  for  $\psi$  and  $\chi \Rightarrow$ 
    (match conclusion in  $\Theta [\varphi \chi \text{ in } v]$  for  $\Theta \varphi$  and  $v \Rightarrow$ 
      (rule rule-sub-nec[where  $\Theta=\Theta$  and  $\chi=\chi$  and  $\psi=\psi$  and  $\varphi=\varphi$  and  $v=v$ ],
        ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
        unfold Substable-Cond-defs)> >)

```

```

method PLM-autosubst1 =
  (match premises in  $\bigwedge v x . [\psi x \equiv \chi x \text{ in } v]$ 
    for  $\psi::'a::\text{type} \Rightarrow o$  and  $\chi::'a \Rightarrow o \Rightarrow$ 
    (match conclusion in  $\Theta [\varphi \chi \text{ in } v]$  for  $\Theta \varphi$  and  $v \Rightarrow$ 
      (rule rule-sub-nec[where  $\Theta=\Theta$  and  $\chi=\chi$  and  $\psi=\psi$  and  $\varphi=\varphi$  and  $v=v$ ],
        ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
        unfold Substable-Cond-defs)> >)

```

```

method PLM-autosubst2 =
  (match premises in  $\bigwedge v x y . [\psi x y \equiv \chi x y \text{ in } v]$ 
    for  $\psi::'a::\text{type} \Rightarrow 'a \Rightarrow o$  and  $\chi::'a::\text{type} \Rightarrow 'a \Rightarrow o \Rightarrow$ 
    (match conclusion in  $\Theta [\varphi \chi \text{ in } v]$  for  $\Theta \varphi$  and  $v \Rightarrow$ 
      (rule rule-sub-nec[where  $\Theta=\Theta$  and  $\chi=\chi$  and  $\psi=\psi$  and  $\varphi=\varphi$  and  $v=v$ ],
        ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
        unfold Substable-Cond-defs)> >)

```

```

method PLM-subst-goal-method for  $\varphi::'a::\text{Substable} \Rightarrow o$  and  $\psi::'a =$ 
  (match conclusion in  $\Theta [\varphi \chi \text{ in } v]$  for  $\Theta$  and  $\chi$  and  $v \Rightarrow$ 
    (rule rule-sub-nec[where  $\Theta=\Theta$  and  $\chi=\chi$  and  $\psi=\psi$  and  $\varphi=\varphi$  and  $v=v$ ],
      ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
      unfold Substable-Cond-defs)>)

```

```

lemma rule-sub-nec[PLM]:
  assumes Substable Substable-Cond  $\varphi$ 
  shows  $(\bigwedge v. [(\psi \equiv \chi) \text{ in } v]) \Longrightarrow \Theta [\varphi \psi \text{ in } v] \Longrightarrow \Theta [\varphi \chi \text{ in } v]$ 
  proof –
    assume  $(\bigwedge v. [(\psi \equiv \chi) \text{ in } v])$ 
    hence  $[\varphi \psi \text{ in } v] = [\varphi \chi \text{ in } v]$ 
      using assms RN unfolding Substable-def Substable-Cond-defs
      using  $\equiv I CP \equiv E(1) \equiv E(2)$  by meson
    thus  $\Theta [\varphi \psi \text{ in } v] \Longrightarrow \Theta [\varphi \chi \text{ in } v]$  by auto
  qed

```

```

lemma rule-sub-nec1[PLM]:
  assumes Substable Substable-Cond  $\varphi$ 
  shows  $(\bigwedge v x . [(\psi x \equiv \chi x) \text{ in } v]) \Longrightarrow \Theta [\varphi \psi \text{ in } v] \Longrightarrow \Theta [\varphi \chi \text{ in } v]$ 
  proof –
    assume  $(\bigwedge v x . [(\psi x \equiv \chi x) \text{ in } v])$ 
    hence  $[\varphi \psi \text{ in } v] = [\varphi \chi \text{ in } v]$ 
      using assms RN unfolding Substable-def Substable-Cond-defs
      using  $\equiv I CP \equiv E(1) \equiv E(2)$  by metis
    thus  $\Theta [\varphi \psi \text{ in } v] \Longrightarrow \Theta [\varphi \chi \text{ in } v]$  by auto
  qed

```

```

lemma rule-sub-nec2[PLM]:
  assumes Substable Substable-Cond  $\varphi$ 
  shows  $(\bigwedge v x y . [\psi x y \equiv \chi x y \text{ in } v]) \Longrightarrow \Theta [\varphi \psi \text{ in } v] \Longrightarrow \Theta [\varphi \chi \text{ in } v]$ 
  proof –
    assume  $(\bigwedge v x y . [\psi x y \equiv \chi x y \text{ in } v])$ 

```


hence $[\varphi \psi \text{ in } v] = [\varphi \chi \text{ in } v]$
 using *assms RN unfolding Substable-def Substable-Cond-defs*
 using $\equiv I \text{ CP } \equiv E(1) \equiv E(2)$ by *metis*
 thus $\Theta [\varphi \psi \text{ in } v] \Longrightarrow \Theta [\varphi \chi \text{ in } v]$ by *auto*
 qed

lemma rule-sub-remark-1-autosubst:
 assumes $(\bigwedge v. [\![A!, x]\!] \equiv (\neg(\Diamond(\![E!, x]\!))) \text{ in } v]$
 and $[\neg(\![A!, x]\!) \text{ in } v]$
 shows $[\neg\neg\Diamond(\![E!, x]\!) \text{ in } v]$
 apply (insert *assms*) apply *PLM-autosubst* by *auto*

lemma rule-sub-remark-1:
 assumes $(\bigwedge v. [\![A!, x]\!] \equiv (\neg(\Diamond(\![E!, x]\!))) \text{ in } v]$
 and $[\neg(\![A!, x]\!) \text{ in } v]$
 shows $[\neg\neg\Diamond(\![E!, x]\!) \text{ in } v]$
 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 $(\bigwedge v. [\![R, x, y]\!] \equiv ((\![R, x, y]\!] \ \&\ (\![Q, a]\!] \vee (\neg(\![Q, a]\!)))) \text{ in } v]$
 and $[p \rightarrow (\![R, x, y]\!) \text{ in } v]$
 shows $[p \rightarrow ((\![R, x, y]\!] \ \&\ (\![Q, a]\!] \vee (\neg(\![Q, a]\!)))) \text{ in } v]$
 apply (insert *assms*) apply *PLM-autosubst* by *auto*

lemma rule-sub-remark-3-autosubst:
 assumes $(\bigwedge v \ x. [\![A!, x^P]\!] \equiv (\neg(\Diamond(\![E!, x^P]\!))) \text{ in } v]$
 and $[\exists \ x. \ (\![A!, x^P]\!) \text{ in } v]$
 shows $[\exists \ x. \ (\neg(\Diamond(\![E!, x^P]\!))) \text{ in } v]$
 apply (insert *assms*) apply *PLM-autosubst1* by *auto*

lemma rule-sub-remark-3:
 assumes $(\bigwedge v \ x. [\![A!, x^P]\!] \equiv (\neg(\Diamond(\![E!, x^P]\!))) \text{ in } v]$
 and $[\exists \ x. \ (\![A!, x^P]\!) \text{ in } v]$
 shows $[\exists \ x. \ (\neg(\Diamond(\![E!, x^P]\!))) \text{ in } v]$
 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 $\bigwedge v \ x. [(\neg(\neg(\![P, x^P]\!))) \equiv (\![P, x^P]\!) \text{ in } v]$
 and $[\mathcal{A}(\neg(\neg(\![P, x^P]\!))) \text{ in } v]$
 shows $[\mathcal{A}(\![P, x^P]\!) \text{ in } v]$
 apply (insert *assms*) apply *PLM-autosubst1* by *auto*

lemma rule-sub-remark-5:
 assumes $\bigwedge v. [(\varphi \rightarrow \psi) \equiv ((\neg\psi) \rightarrow (\neg\varphi)) \text{ in } v]$
 and $[\Box(\varphi \rightarrow \psi) \text{ in } v]$
 shows $[\Box((\neg\psi) \rightarrow (\neg\varphi)) \text{ in } v]$
 apply (insert *assms*) apply *PLM-autosubst* by *auto*

lemma rule-sub-remark-6:
 assumes $\bigwedge v. [\psi \equiv \chi \text{ in } v]$
 and $[\Box(\varphi \rightarrow \psi) \text{ in } v]$
 shows $[\Box(\varphi \rightarrow \chi) \text{ in } v]$
 apply (insert *assms*) apply *PLM-autosubst* by *auto*

lemma *rule-sub-remark-7*:

assumes $\bigwedge v. [\varphi \equiv (\neg(\neg\varphi)) \text{ in } v]$
and $[\Box(\varphi \rightarrow \varphi) \text{ in } v]$
shows $[\Box((\neg(\neg\varphi)) \rightarrow \varphi) \text{ in } v]$
apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-8*:

assumes $\bigwedge v. [\mathcal{A}\varphi \equiv \varphi \text{ in } v]$
and $[\Box(\mathcal{A}\varphi) \text{ in } v]$
shows $[\Box(\varphi) \text{ in } v]$
apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-9*:

assumes $\bigwedge v. [(\Box P, a) \equiv ((\Box P, a) \ \& \ ((\Box Q, b) \vee (\neg(\Box Q, b)))) \text{ in } v]$
and $[(\Box P, a) = (\Box P, a) \text{ in } v]$
shows $[(\Box P, a) = ((\Box P, a) \ \& \ ((\Box Q, b) \vee (\neg(\Box Q, b)))) \text{ in } v]$
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*]:

$[\Box\varphi \equiv \Box(\neg(\neg\varphi)) \text{ in } v]$
apply (*PLM-subst-method* $\varphi \ (\neg(\neg\varphi))$)
by *PLM-solver+*

lemma *KBasic2-2*[*PLM*]:

$[(\neg(\Box\varphi)) \equiv \Diamond(\neg\varphi) \text{ in } v]$
unfolding *diamond-def*
apply (*PLM-subst-method* $\varphi \ \neg(\neg\varphi)$)
by *PLM-solver+*

lemma *KBasic2-3*[*PLM*]:

$[\Box\varphi \equiv (\neg(\Diamond(\neg\varphi))) \text{ in } v]$
unfolding *diamond-def*
apply (*PLM-subst-method* $\varphi \ \neg(\neg\varphi)$)
apply *PLM-solver*
by (*simp add: oth-class-taut-4-b*)

lemmas *Df* $\Box = KBasic2-3$

lemma *KBasic2-4*[*PLM*]:

$[\Box(\neg(\varphi)) \equiv (\neg(\Diamond\varphi)) \text{ in } v]$
unfolding *diamond-def*
by (*simp add: oth-class-taut-4-b*)

lemma *KBasic2-5*[*PLM*]:

$[\Box(\varphi \rightarrow \psi) \rightarrow (\Diamond\varphi \rightarrow \Diamond\psi) \text{ in } v]$
by (*simp only: CP RM-2-b*)

lemmas *K* $\Diamond = KBasic2-5$

lemma *KBasic2-6*[*PLM*]:

$[\Diamond(\varphi \vee \psi) \equiv (\Diamond\varphi \vee \Diamond\psi) \text{ in } v]$
proof —
have $[\Box((\neg\varphi) \ \& \ (\neg\psi)) \equiv (\Box(\neg\varphi) \ \& \ \Box(\neg\psi)) \text{ in } v]$
using *KBasic-3* **by** *blast*
hence $[(\neg(\Diamond(\neg(\neg\varphi) \ \& \ (\neg\psi)))) \equiv (\Box(\neg\varphi) \ \& \ \Box(\neg\psi)) \text{ in } v]$
using *Df* \Box **by** (*rule* $\equiv E(6)$)

hence $[(\neg(\Diamond(\neg(\neg\varphi) \ \& \ (\neg\psi)))) \equiv ((\neg(\Diamond\varphi)) \ \& \ (\neg(\Diamond\psi))) \text{ in } v]$
apply – **apply** (*PLM-subst-method* $\Box(\neg\varphi) \ \neg(\Diamond\varphi)$)
apply (*simp add: KBasic2-4*)
apply (*PLM-subst-method* $\Box(\neg\psi) \ \neg(\Diamond\psi)$)
apply (*simp add: KBasic2-4*)
unfolding *diamond-def* **by** *assumption*
hence $[(\neg(\Diamond(\varphi \vee \psi))) \equiv ((\neg(\Diamond\varphi)) \ \& \ (\neg(\Diamond\psi))) \text{ in } v]$
apply – **apply** (*PLM-subst-method* $\neg((\neg\varphi) \ \& \ (\neg\psi)) \ \varphi \vee \psi$)
using *oth-class-taut-6-b[equiv-sym]* **by** *auto*
hence $[(\neg(\neg(\Diamond(\varphi \vee \psi)))) \equiv (\neg((\neg(\Diamond\varphi)) \ \& \ (\neg(\Diamond\psi)))) \text{ in } v]$
by (*rule oth-class-taut-5-d[equiv-lr]*)
hence $[\Diamond(\varphi \vee \psi) \equiv (\neg((\neg(\Diamond\varphi)) \ \& \ (\neg(\Diamond\psi)))) \text{ in } v]$
apply – **apply** (*PLM-subst-method* $\neg(\neg(\Diamond(\varphi \vee \psi))) \ \Diamond(\varphi \vee \psi)$)
using *oth-class-taut-4-b[equiv-sym]* **by** *auto*
thus *?thesis*
apply – **apply** (*PLM-subst-method* $\neg((\neg(\Diamond\varphi)) \ \& \ (\neg(\Diamond\psi))) \ (\Diamond\varphi) \vee (\Diamond\psi)$)
using *oth-class-taut-6-b[equiv-sym]* **by** *auto*
qed

lemma *KBasic2-7[PLM]*:

$[(\Box\varphi \vee \Box\psi) \rightarrow \Box(\varphi \vee \psi) \text{ in } v]$

proof –

have $\bigwedge v . [\varphi \rightarrow (\varphi \vee \psi) \text{ in } v]$

by (*metis contraposition-1 contraposition-2 useful-tautologies-3 disj-def*)

hence $[\Box\varphi \rightarrow \Box(\varphi \vee \psi) \text{ in } v]$ **using** *RM-1* **by** *auto*

moreover {

have $\bigwedge v . [\psi \rightarrow (\varphi \vee \psi) \text{ in } v]$

by (*simp only: pl-1[axiom-instance] disj-def*)

hence $[\Box\psi \rightarrow \Box(\varphi \vee \psi) \text{ in } v]$

using *RM-1* **by** *auto*

}

ultimately show *?thesis*

using *oth-class-taut-10-d vdash-properties-10* **by** *blast*

qed

lemma *KBasic2-8[PLM]*:

$[\Diamond(\varphi \ \& \ \psi) \rightarrow (\Diamond\varphi \ \& \ \Diamond\psi) \text{ in } v]$

by (*metis CP RM-2 &I oth-class-taut-9-a*

oth-class-taut-9-b vdash-properties-10)

lemma *KBasic2-9[PLM]*:

$[\Diamond(\varphi \rightarrow \psi) \equiv (\Box\varphi \rightarrow \Diamond\psi) \text{ in } v]$

apply (*PLM-subst-method* $(\neg(\Box\varphi)) \vee (\Diamond\psi) \ \Box\varphi \rightarrow \Diamond\psi$)

using *oth-class-taut-5-k[equiv-sym]* **apply** *simp*

apply (*PLM-subst-method* $(\neg\varphi) \vee \psi \ \varphi \rightarrow \psi$)

using *oth-class-taut-5-k[equiv-sym]* **apply** *simp*

apply (*PLM-subst-method* $\Diamond(\neg\varphi) \ \neg(\Box\varphi)$)

using *KBasic2-2[equiv-sym]* **apply** *simp*

using *KBasic2-6* .

lemma *KBasic2-10[PLM]*:

$[\Diamond(\Box\varphi) \equiv (\neg(\Box\Diamond(\neg\varphi))) \text{ in } v]$

unfolding *diamond-def* **apply** (*PLM-subst-method* $\varphi \ \neg\neg\varphi$)

using *oth-class-taut-4-b oth-class-taut-4-a* **by** *auto*

lemma *KBasic2-11[PLM]*:

$[\Diamond\Diamond\varphi \equiv (\neg(\Box\Box(\neg\varphi))) \text{ in } v]$

unfolding *diamond-def*

```

apply (PLM-subst-method  $\Box(\neg\varphi) \neg(\neg(\Box(\neg\varphi)))$ )
using oth-class-taut-4-b oth-class-taut-4-a by auto

lemma KBasic2-12[PLM]:  $[\Box(\varphi \vee \psi) \rightarrow (\Box\varphi \vee \Diamond\psi) \text{ in } v]$ 
proof –
  have  $[\Box(\psi \vee \varphi) \rightarrow (\Box(\neg\psi) \rightarrow \Box\varphi) \text{ in } v]$ 
    using CP RM-1-b  $\vee E(2)$  by blast
  hence  $[\Box(\psi \vee \varphi) \rightarrow (\Diamond\psi \vee \Box\varphi) \text{ in } v]$ 
    unfolding diamond-def disj-def
    by (meson CP  $\neg\neg E$  vdash-properties-6)
  thus ?thesis apply –
    apply (PLM-subst-method  $(\Diamond\psi \vee \Box\varphi) (\Box\varphi \vee \Diamond\psi)$ )
    apply (simp add: PLM.oth-class-taut-3-e)
    apply (PLM-subst-method  $(\psi \vee \varphi) (\varphi \vee \psi)$ )
    apply (simp add: PLM.oth-class-taut-3-e)
    by assumption
qed

lemma TBasic[PLM]:
   $[\varphi \rightarrow \Diamond\varphi \text{ in } v]$ 
  unfolding diamond-def
  apply (subst contraposition-1)
  apply (PLM-subst-method  $\Box\neg\varphi \neg\neg\Box\neg\varphi$ )
  apply (simp add: PLM.oth-class-taut-4-b)
  using qml-2[where  $\varphi=\neg\varphi$ , axiom-instance]
  by simp
lemmas T $\Diamond$  = TBasic

lemma S5Basic-1[PLM]:
   $[\Diamond\Box\varphi \rightarrow \Box\varphi \text{ in } v]$ 
proof (rule CP)
  assume  $[\Diamond\Box\varphi \text{ in } v]$ 
  hence  $[\neg\Box\Diamond\neg\varphi \text{ in } v]$ 
    using KBasic2-10[equiv-lr] by simp
  moreover have  $[\Diamond(\neg\varphi) \rightarrow \Box\Diamond(\neg\varphi) \text{ in } v]$ 
    by (simp add: qml-3[axiom-instance])
  ultimately have  $[\neg\Diamond\neg\varphi \text{ in } v]$ 
    by (simp add: PLM.modus-tollens-1)
  thus  $[\Box\varphi \text{ in } v]$ 
    unfolding diamond-def apply –
    apply (PLM-subst-method  $\neg\neg\varphi \varphi$ )
    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 S5 $\Diamond$  = S5Basic-1

lemma S5Basic-2[PLM]:
   $[\Box\varphi \equiv \Diamond\Box\varphi \text{ in } v]$ 
using S5 $\Diamond$  T $\Diamond$   $\equiv I$  by blast

lemma S5Basic-3[PLM]:
   $[\Diamond\varphi \equiv \Box\Diamond\varphi \text{ in } v]$ 
using qml-3[axiom-instance] qml-2[axiom-instance]  $\equiv I$  by blast

lemma S5Basic-4[PLM]:
   $[\varphi \rightarrow \Box\Diamond\varphi \text{ in } v]$ 
using T $\Diamond$ [deduction, THEN S5Basic-3[equiv-lr]]

```

by (rule CP)

lemma *S5Basic-5*[PLM]:
 $[\Diamond\Box\varphi \rightarrow \varphi \text{ in } v]$
using *S5Basic-2*[equiv-rl, THEN qml-2[axiom-instance, deduction]]
by (rule CP)

lemmas $B\Diamond = \text{S5Basic-5}$

lemma *S5Basic-6*[PLM]:
 $[\Box\varphi \rightarrow \Box\Box\varphi \text{ in } v]$
using *S5Basic-4*[deduction] *RM-1*[OF *S5Basic-1*, deduction] CP **by** auto

lemmas $4\Box = \text{S5Basic-6}$

lemma *S5Basic-7*[PLM]:
 $[\Box\varphi \equiv \Box\Box\varphi \text{ in } v]$
using $4\Box$ qml-2[axiom-instance] **by** (rule $\equiv I$)

lemma *S5Basic-8*[PLM]:
 $[\Diamond\Diamond\varphi \rightarrow \Diamond\varphi \text{ in } v]$
using *S5Basic-6*[**where** $\varphi = \neg\varphi$, THEN contraposition-1[THEN iffD1], deduction]
KBasic2-11[equiv-lr] CP **unfolding** diamond-def **by** auto

lemmas $4\Diamond = \text{S5Basic-8}$

lemma *S5Basic-9*[PLM]:
 $[\Diamond\Diamond\varphi \equiv \Diamond\varphi \text{ in } v]$
using $4\Diamond$ T \Diamond **by** (rule $\equiv I$)

lemma *S5Basic-10*[PLM]:
 $[\Box(\varphi \vee \Box\psi) \equiv (\Box\varphi \vee \Box\psi) \text{ in } v]$
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]:
 $[\Box(\varphi \vee \Diamond\psi) \equiv (\Box\varphi \vee \Diamond\psi) \text{ in } v]$
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]:
 $[\Diamond(\varphi \ \& \ \Diamond\psi) \equiv (\Diamond\varphi \ \& \ \Diamond\psi) \text{ in } v]$
proof –
have $[\Box((\neg\varphi) \vee \Box(\neg\psi)) \equiv (\Box(\neg\varphi) \vee \Box(\neg\psi)) \text{ in } v]$
using *S5Basic-10* **by** auto
hence 1: $[\neg(\Box((\neg\varphi) \vee \Box(\neg\psi))) \equiv \neg(\Box(\neg\varphi) \vee \Box(\neg\psi)) \text{ in } v]$
using oth-class-taut-5-d[equiv-lr] **by** auto
have 2: $[\Diamond(\neg(\neg(\neg\varphi) \vee (\neg(\Diamond\psi)))) \equiv (\neg(\neg(\Diamond\varphi)) \vee (\neg(\Diamond\psi))) \text{ in } v]$
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  $\neg \Box((\neg \varphi) \vee \Box(\neg \psi))$ ) ( $\Diamond(\neg((\neg \varphi) \vee (\Box(\neg \psi))))$ )
    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  $\neg((\neg \varphi) \vee (\neg \Diamond \psi))$ )  $\varphi \ \& \ \Diamond \psi$ 
    using oth-class-taut-6-a[equiv-sym] apply simp
    apply (PLM-subst-method  $\neg((\neg(\Diamond \varphi)) \vee (\neg \Diamond \psi))$ )  $\Diamond \varphi \ \& \ \Diamond \psi$ 
    using oth-class-taut-6-a[equiv-sym] apply simp
    using 2 by assumption
qed

```

lemma S5Basic-13[PLM]:

```

 $[\Diamond(\varphi \ \& \ (\Box \psi)) \equiv (\Diamond \varphi \ \& \ (\Box \psi)) \text{ in } v]$ 
  apply (PLM-subst-method  $\Diamond \Box \psi \ \Box \psi$ )
  using S5Basic-2[equiv-sym] apply simp
  using S5Basic-12 by simp

```

lemma S5Basic-14[PLM]:

```

 $[\Box(\varphi \rightarrow (\Box \psi)) \equiv \Box(\Diamond \varphi \rightarrow \psi) \text{ in } v]$ 
  proof (rule  $\equiv I$ ; rule CP)
    assume  $[\Box(\varphi \rightarrow \Box \psi) \text{ in } v]$ 
    moreover {
      have  $\bigwedge v. [\Box(\varphi \rightarrow \Box \psi) \rightarrow (\Diamond \varphi \rightarrow \psi) \text{ in } v]$ 
      proof (rule CP)
        fix v
        assume  $[\Box(\varphi \rightarrow \Box \psi) \text{ in } v]$ 
        hence  $[\Diamond \varphi \rightarrow \Diamond \Box \psi \text{ in } v]$ 
        using K $\Diamond$ [deduction] by auto
        thus  $[\Diamond \varphi \rightarrow \psi \text{ in } v]$ 
        using B $\Diamond$  ded-thm-cor-3 by blast
      qed
      hence  $[\Box(\Box(\varphi \rightarrow \Box \psi) \rightarrow (\Diamond \varphi \rightarrow \psi)) \text{ in } v]$ 
      by (rule RN)
      hence  $[\Box(\Box(\varphi \rightarrow \Box \psi)) \rightarrow \Box((\Diamond \varphi \rightarrow \psi)) \text{ in } v]$ 
      using qml-1[axiom-instance, deduction] by auto
    }
    ultimately show  $[\Box(\Diamond \varphi \rightarrow \psi) \text{ in } v]$ 
    using S5Basic-6 CP vdash-properties-10 by meson
  next
    assume  $[\Box(\Diamond \varphi \rightarrow \psi) \text{ in } v]$ 
    moreover {
      fix v
      {
        assume  $[\Box(\Diamond \varphi \rightarrow \psi) \text{ in } v]$ 
        hence 1:  $[\Box \Diamond \varphi \rightarrow \Box \psi \text{ in } v]$ 
        using qml-1[axiom-instance, deduction] by auto
        assume  $[\varphi \text{ in } v]$ 
        hence  $[\Box \Diamond \varphi \text{ in } v]$ 
        using S5Basic-4[deduction] by auto
        hence  $[\Box \psi \text{ in } v]$ 
        using 1[deduction] by auto
      }
      hence  $[\Box(\Diamond \varphi \rightarrow \psi) \text{ in } v] \implies [\varphi \rightarrow \Box \psi \text{ in } v]$ 
      using CP by auto
    }
    ultimately show  $[\Box(\varphi \rightarrow \Box \psi) \text{ in } v]$ 

```

using *S5Basic-6 RN-2 vdash-properties-10* by *blast*
qed

lemma *sc-eg-box-box-1* [PLM]:

$[\Box(\varphi \rightarrow \Box\varphi) \rightarrow (\Diamond\varphi \equiv \Box\varphi) \text{ in } v]$
proof (*rule CP*)
 assume 1: $[\Box(\varphi \rightarrow \Box\varphi) \text{ in } v]$
 hence $[\Box(\Diamond\varphi \rightarrow \varphi) \text{ in } v]$
 using *S5Basic-14* [equiv-lr] by *auto*
 hence $[\Diamond\varphi \rightarrow \varphi \text{ in } v]$
 using *qml-2* [axiom-instance, deduction] by *auto*
 moreover from 1 have $[\varphi \rightarrow \Box\varphi \text{ in } v]$
 using *qml-2* [axiom-instance, deduction] by *auto*
 ultimately have $[\Diamond\varphi \rightarrow \Box\varphi \text{ in } v]$
 using *ded-thm-cor-3* by *auto*
 moreover have $[\Box\varphi \rightarrow \Diamond\varphi \text{ in } v]$
 using *qml-2* [axiom-instance] *T* \Diamond
 by (*rule ded-thm-cor-3*)
 ultimately show $[\Diamond\varphi \equiv \Box\varphi \text{ in } v]$
 by (*rule $\equiv I$*)
 qed

lemma *sc-eg-box-box-2* [PLM]:

$[\Box(\varphi \rightarrow \Box\varphi) \rightarrow ((\neg\Box\varphi) \equiv (\Box(\neg\varphi))) \text{ in } v]$
proof (*rule CP*)
 assume $[\Box(\varphi \rightarrow \Box\varphi) \text{ in } v]$
 hence $[(\neg\Box(\neg\varphi)) \equiv \Box\varphi \text{ in } v]$
 using *sc-eg-box-box-1* [deduction] unfolding *diamond-def* by *auto*
 thus $[(\neg\Box\varphi) \equiv (\Box(\neg\varphi)) \text{ in } v]$
 by (*meson CP $\equiv I \equiv E(3)$*)
 $\equiv E(4) \neg\neg I \neg\neg E$
 qed

lemma *sc-eg-box-box-3* [PLM]:

$[(\Box(\varphi \rightarrow \Box\varphi) \ \& \ \Box(\psi \rightarrow \Box\psi)) \rightarrow ((\Box\varphi \equiv \Box\psi) \rightarrow \Box(\varphi \equiv \psi)) \text{ in } v]$
proof (*rule CP*)
 assume 1: $[(\Box(\varphi \rightarrow \Box\varphi) \ \& \ \Box(\psi \rightarrow \Box\psi)) \text{ in } v]$
 {
 assume $[\Box\varphi \equiv \Box\psi \text{ in } v]$
 hence $[(\Box\varphi \ \& \ \Box\psi) \vee ((\neg(\Box\varphi)) \ \& \ (\neg(\Box\psi))) \text{ in } v]$
 using *oth-class-taut-5-i* [equiv-lr] by *auto*
 moreover {
 assume $[\Box\varphi \ \& \ \Box\psi \text{ in } v]$
 hence $[\Box(\varphi \equiv \psi) \text{ in } v]$
 using *KBasic-7* [deduction] by *auto*
 }
 moreover {
 assume $[(\neg(\Box\varphi)) \ \& \ (\neg(\Box\psi)) \text{ in } v]$
 hence $[\Box(\neg\varphi) \ \& \ \Box(\neg\psi) \text{ in } v]$
 using 1 & *E* & *I* *sc-eg-box-box-2* [deduction, equiv-lr]
 by *metis*
 hence $[\Box((\neg\varphi) \ \& \ (\neg\psi)) \text{ in } v]$
 using *KBasic-3* [equiv-rl] by *auto*
 hence $[\Box(\varphi \equiv \psi) \text{ in } v]$
 using *KBasic-9* [deduction] by *auto*
 }
 ultimately have $[\Box(\varphi \equiv \psi) \text{ in } v]$
 using *CP $\vee E(1)$* by *blast*

```

}
thus  $\Box\varphi \equiv \Box\psi \rightarrow \Box(\varphi \equiv \psi)$  in  $v$ 
  using CP by auto
qed

```

```

lemma derived-S5-rules-1-a[PLM]:
  assumes  $\bigwedge v. [\chi \text{ in } v] \Rightarrow [\Diamond\varphi \rightarrow \psi \text{ in } v]$ 
  shows  $[\Box\chi \text{ in } v] \Rightarrow [\varphi \rightarrow \Box\psi \text{ in } v]$ 
  proof -
    have  $[\Box\chi \text{ in } v] \Rightarrow [\Box\Diamond\varphi \rightarrow \Box\psi \text{ in } v]$ 
      using assms RM-1-b by metis
    thus  $[\Box\chi \text{ in } v] \Rightarrow [\varphi \rightarrow \Box\psi \text{ in } v]$ 
      using S5Basic-4 vdash-properties-10 CP by metis
  qed

```

```

lemma derived-S5-rules-1-b[PLM]:
  assumes  $\bigwedge v. [\Diamond\varphi \rightarrow \psi \text{ in } v]$ 
  shows  $[\varphi \rightarrow \Box\psi \text{ in } v]$ 
  using derived-S5-rules-1-a all-self-eq-1 assms by blast

```

```

lemma derived-S5-rules-2-a[PLM]:
  assumes  $\bigwedge v. [\chi \text{ in } v] \Rightarrow [\varphi \rightarrow \Box\psi \text{ in } v]$ 
  shows  $[\Box\chi \text{ in } v] \Rightarrow [\Diamond\varphi \rightarrow \psi \text{ in } v]$ 
  proof -
    have  $[\Box\chi \text{ in } v] \Rightarrow [\Diamond\varphi \rightarrow \Diamond\Box\psi \text{ in } v]$ 
      using RM-2-b assms by metis
    thus  $[\Box\chi \text{ in } v] \Rightarrow [\Diamond\varphi \rightarrow \psi \text{ in } v]$ 
      using BDiamond vdash-properties-10 CP by metis
  qed

```

```

lemma derived-S5-rules-2-b[PLM]:
  assumes  $\bigwedge v. [\varphi \rightarrow \Box\psi \text{ in } v]$ 
  shows  $[\Diamond\varphi \rightarrow \psi \text{ in } v]$ 
  using assms derived-S5-rules-2-a all-self-eq-1 by blast

```

```

lemma BFs-1[PLM]:  $[(\forall \alpha. \Box(\varphi \alpha)) \rightarrow \Box(\forall \alpha. \varphi \alpha)]$  in  $v$ 
  proof (rule derived-S5-rules-1-b)
    fix  $v$ 
    {
      fix  $\alpha$ 
      have  $\bigwedge v. [(\forall \alpha. \Box(\varphi \alpha)) \rightarrow \Box(\varphi \alpha)]$  in  $v$ 
        using cqt-orig-1 by metis
      hence  $[\Diamond(\forall \alpha. \Box(\varphi \alpha)) \rightarrow \Diamond\Box(\varphi \alpha)]$  in  $v$ 
        using RM-2 by metis
      moreover have  $[\Diamond\Box(\varphi \alpha) \rightarrow (\varphi \alpha)]$  in  $v$ 
        using BDiamond by auto
      ultimately have  $[\Diamond(\forall \alpha. \Box(\varphi \alpha)) \rightarrow (\varphi \alpha)]$  in  $v$ 
        using ded-thm-cor-3 by auto
    }
    hence  $[\forall \alpha. \Diamond(\forall \alpha. \Box(\varphi \alpha)) \rightarrow (\varphi \alpha)]$  in  $v$ 
      using  $\forall I$  by metis
    thus  $[\Diamond(\forall \alpha. \Box(\varphi \alpha)) \rightarrow (\forall \alpha. \varphi \alpha)]$  in  $v$ 
      using cqt-orig-2[deduction] by auto
  qed

```

```

lemmas BF = BFs-1

```

```

lemma BFs-2[PLM]:
   $[\Box(\forall \alpha. \varphi \alpha) \rightarrow (\forall \alpha. \Box(\varphi \alpha))]$  in  $v$ 

```



```

proof -
{
  fix  $\alpha$ 
  {
    fix  $v$ 
    have  $[(\forall \alpha. \varphi \alpha) \rightarrow \varphi \alpha \text{ in } v]$  using cqt-orig-1 by metis
  }
  hence  $[\Box(\forall \alpha. \varphi \alpha) \rightarrow \Box(\varphi \alpha) \text{ in } v]$  using RM-1 by auto
}
hence  $[\forall \alpha. \Box(\forall \alpha. \varphi \alpha) \rightarrow \Box(\varphi \alpha) \text{ in } v]$  using  $\forall I$  by metis
thus ?thesis using cqt-orig-2[deduction] by metis
qed
lemmas CBF = BFs-2

```

```

lemma BFs-3[PLM]:
 $[\Diamond(\exists \alpha. \varphi \alpha) \rightarrow (\exists \alpha. \Diamond(\varphi \alpha)) \text{ in } v]$ 
proof -
  have  $[(\forall \alpha. \Box(\neg(\varphi \alpha))) \rightarrow \Box(\forall \alpha. \neg(\varphi \alpha)) \text{ in } v]$ 
    using BF by metis
  hence 1:  $[(\neg(\Box(\forall \alpha. \neg(\varphi \alpha)))) \rightarrow (\neg(\forall \alpha. \Box(\neg(\varphi \alpha)))) \text{ in } v]$ 
    using contraposition-1 by simp
  have 2:  $[\Diamond(\neg(\forall \alpha. \neg(\varphi \alpha))) \rightarrow (\neg(\forall \alpha. \Box(\neg(\varphi \alpha)))) \text{ in } v]$ 
    apply (PLM-subst-method  $\neg\Box(\forall \alpha. \neg(\varphi \alpha)) \Diamond(\neg(\forall \alpha. \neg(\varphi \alpha)))$ )
    using KBasic2-2 1 by simp+
  have  $[\Diamond(\neg(\forall \alpha. \neg(\varphi \alpha))) \rightarrow (\exists \alpha. \neg(\Box(\neg(\varphi \alpha)))) \text{ in } v]$ 
    apply (PLM-subst-method  $\neg(\forall \alpha. \Box(\neg(\varphi \alpha))) \exists \alpha. \neg(\Box(\neg(\varphi \alpha)))$ )
    using cqt-further-2 apply metis
    using 2 by metis
  thus ?thesis
    unfolding exists-def diamond-def by auto
qed
lemmas BF $\Diamond$  = BFs-3

```

```

lemma BFs-4[PLM]:
 $[(\exists \alpha. \Diamond(\varphi \alpha)) \rightarrow \Diamond(\exists \alpha. \varphi \alpha) \text{ in } v]$ 
proof -
  have 1:  $[\Box(\forall \alpha. \neg(\varphi \alpha)) \rightarrow (\forall \alpha. \Box(\neg(\varphi \alpha))) \text{ in } v]$ 
    using CBF by auto
  have 2:  $[(\exists \alpha. (\neg(\Box(\neg(\varphi \alpha))))) \rightarrow (\neg(\Box(\forall \alpha. \neg(\varphi \alpha)))) \text{ in } v]$ 
    apply (PLM-subst-method  $\neg(\forall \alpha. \Box(\neg(\varphi \alpha))) (\exists \alpha. (\neg(\Box(\neg(\varphi \alpha)))))$ )
    using cqt-further-2 apply blast
    using 1 using contraposition-1 by metis
  have  $[(\exists \alpha. (\neg(\Box(\neg(\varphi \alpha))))) \rightarrow \Diamond(\neg(\forall \alpha. \neg(\varphi \alpha))) \text{ in } v]$ 
    apply (PLM-subst-method  $\neg(\Box(\forall \alpha. \neg(\varphi \alpha))) \Diamond(\neg(\forall \alpha. \neg(\varphi \alpha)))$ )
    using KBasic2-2 apply blast
    using 2 by assumption
  thus ?thesis
    unfolding diamond-def exists-def by auto
qed
lemmas CBF $\Diamond$  = BFs-4

```

```

lemma sign-S5-thm-1[PLM]:
 $[(\exists \alpha. \Box(\varphi \alpha)) \rightarrow \Box(\exists \alpha. \varphi \alpha) \text{ in } v]$ 
proof (rule CP)
  assume  $[\exists \alpha. \Box(\varphi \alpha) \text{ in } v]$ 
  then obtain  $\tau$  where  $[\Box(\varphi \tau) \text{ in } v]$ 
    by (rule  $\exists E$ )
  moreover {

```

```

    fix v
    assume [ $\varphi \tau$  in  $v$ ]
    hence [ $\exists \alpha . \varphi \alpha$  in  $v$ ]
      by (rule  $\exists I$ )
  }
  ultimately show [ $\Box(\exists \alpha . \varphi \alpha)$  in  $v$ ]
    using RN-2 by blast
qed
lemmas Buridan = sign-S5-thm-1

lemma sign-S5-thm-2[PLM]:
  [ $\Diamond(\forall \alpha . \varphi \alpha) \rightarrow (\forall \alpha . \Diamond(\varphi \alpha))$  in  $v$ ]
proof -
  {
    fix  $\alpha$ 
    {
      fix  $v$ 
      have [ $(\forall \alpha . \varphi \alpha) \rightarrow \varphi \alpha$  in  $v$ ]
        using cqt-orig-1 by metis
      }
      hence [ $\Diamond(\forall \alpha . \varphi \alpha) \rightarrow \Diamond(\varphi \alpha)$  in  $v$ ]
        using RM-2 by metis
    }
    hence [ $\forall \alpha . \Diamond(\forall \alpha . \varphi \alpha) \rightarrow \Diamond(\varphi \alpha)$  in  $v$ ]
      using  $\forall I$  by metis
    thus ?thesis
      using cqt-orig-2[deduction] by metis
  }
qed
lemmas Buridan $\Diamond$  = sign-S5-thm-2

lemma sign-S5-thm-3[PLM]:
  [ $\Diamond(\exists \alpha . \varphi \alpha \ \& \ \psi \alpha) \rightarrow \Diamond((\exists \alpha . \varphi \alpha) \ \& \ (\exists \alpha . \psi \alpha))$  in  $v$ ]
  by (simp only: RM-2 cqt-further-5)

lemma sign-S5-thm-4[PLM]:
  [ $((\Box(\forall \alpha . \varphi \alpha \rightarrow \psi \alpha)) \ \& \ (\Box(\forall \alpha . \psi \alpha \rightarrow \chi \alpha))) \rightarrow \Box(\forall \alpha . \varphi \alpha \rightarrow \chi \alpha)$  in  $v$ ]
proof (rule CP)
  assume [ $\Box(\forall \alpha . \varphi \alpha \rightarrow \psi \alpha) \ \& \ \Box(\forall \alpha . \psi \alpha \rightarrow \chi \alpha)$  in  $v$ ]
  hence [ $\Box((\forall \alpha . \varphi \alpha \rightarrow \psi \alpha) \ \& \ (\forall \alpha . \psi \alpha \rightarrow \chi \alpha))$  in  $v$ ]
    using KBasic-3[equiv-rl] by blast
  moreover {
    fix  $v$ 
    assume [ $((\forall \alpha . \varphi \alpha \rightarrow \psi \alpha) \ \& \ (\forall \alpha . \psi \alpha \rightarrow \chi \alpha))$  in  $v$ ]
    hence [ $(\forall \alpha . \varphi \alpha \rightarrow \chi \alpha)$  in  $v$ ]
      using cqt-basic-9[deduction] by blast
  }
  ultimately show [ $\Box(\forall \alpha . \varphi \alpha \rightarrow \chi \alpha)$  in  $v$ ]
    using RN-2 by blast
qed

lemma sign-S5-thm-5[PLM]:
  [ $((\Box(\forall \alpha . \varphi \alpha \equiv \psi \alpha)) \ \& \ (\Box(\forall \alpha . \psi \alpha \equiv \chi \alpha))) \rightarrow (\Box(\forall \alpha . \varphi \alpha \equiv \chi \alpha))$  in  $v$ ]
proof (rule CP)
  assume [ $\Box(\forall \alpha . \varphi \alpha \equiv \psi \alpha) \ \& \ \Box(\forall \alpha . \psi \alpha \equiv \chi \alpha)$  in  $v$ ]
  hence [ $\Box((\forall \alpha . \varphi \alpha \equiv \psi \alpha) \ \& \ (\forall \alpha . \psi \alpha \equiv \chi \alpha))$  in  $v$ ]
    using KBasic-3[equiv-rl] by blast
  moreover {
    fix  $v$ 

```

```

    assume  $[(\forall \alpha. \varphi \alpha \equiv \psi \alpha) \ \& \ (\forall \alpha. \psi \alpha \equiv \chi \alpha)] \text{ in } v]$ 
    hence  $[(\forall \alpha. \varphi \alpha \equiv \chi \alpha) \text{ in } v]$ 
    using cqt-basic-10[deduction] by blast
  }
  ultimately show  $[\Box(\forall \alpha. \varphi \alpha \equiv \chi \alpha) \text{ in } v]$ 
  using RN-2 by blast
qed

```

```

lemma id-nec2-1[PLM]:
   $[\Diamond((\alpha::'a::id-eq) = \beta) \equiv (\alpha = \beta) \text{ in } v]$ 
  apply (rule  $\equiv I$ ; rule CP)
  using id-nec[equiv-lr] derived-S5-rules-2-b CP modus-ponens apply blast
  using T $\Diamond$ [deduction] by auto

```

```

lemma id-nec2-2-Aux:
   $[(\Diamond \varphi) \equiv \psi \text{ in } v] \implies [(\neg \psi) \equiv \Box(\neg \varphi) \text{ in } v]$ 
  proof -
    assume  $[(\Diamond \varphi) \equiv \psi \text{ in } v]$ 
    moreover have  $\bigwedge \varphi \psi. [(\neg \varphi) \equiv \psi \text{ in } v] \implies [(\neg \psi) \equiv \varphi \text{ in } v]$ 
    by PLM-solver
    ultimately show ?thesis
    unfolding diamond-def by blast
  qed

```

```

lemma id-nec2-2[PLM]:
   $[(\Diamond((\alpha::'a::id-eq) \neq \beta) \equiv \Box(\alpha \neq \beta) \text{ in } v]$ 
  using id-nec2-1[THEN id-nec2-2-Aux] by auto

```

```

lemma id-nec2-3[PLM]:
   $[(\Diamond((\alpha::'a::id-eq) \neq \beta)) \equiv (\alpha \neq \beta) \text{ in } v]$ 
  using T $\Diamond \equiv I$  id-nec2-2[equiv-lr]
    CP derived-S5-rules-2-b by metis

```

```

lemma exists-desc-box-1[PLM]:
   $[(\exists y. (y^P) = (\iota x. \varphi x)) \rightarrow (\exists y. \Box((y^P) = (\iota x. \varphi x))) \text{ in } v]$ 
  proof (rule CP)
    assume  $[\exists y. (y^P) = (\iota x. \varphi x) \text{ in } v]$ 
    then obtain y where  $[(y^P) = (\iota x. \varphi x) \text{ in } v]$ 
    by (rule  $\exists E$ )
    hence  $[\Box(y^P = (\iota x. \varphi x)) \text{ in } v]$ 
    using l-identity[axiom-instance, deduction, deduction]
      cqt-1[axiom-instance] all-self-eq-2[where 'a= $\nu$ ]
      modus-ponens unfolding identity- $\nu$ -def by fast
    thus  $[\exists y. \Box((y^P) = (\iota x. \varphi x)) \text{ in } v]$ 
    by (rule  $\exists I$ )
  qed

```

```

lemma exists-desc-box-2[PLM]:
   $[(\exists y. (y^P) = (\iota x. \varphi x)) \rightarrow \Box(\exists y. ((y^P) = (\iota x. \varphi x))) \text{ in } v]$ 
  using exists-desc-box-1 Buridan ded-thm-cor-3 by fast

```

```

lemma en-eq-1[PLM]:
   $[\Diamond \llbracket x, F \rrbracket \equiv \Box \llbracket x, F \rrbracket \text{ in } v]$ 
  using encoding[axiom-instance] RN
    sc-eq-box-box-1 modus-ponens by blast

```

```

lemma en-eq-2[PLM]:
   $[\llbracket x, F \rrbracket \equiv \Box \llbracket x, F \rrbracket \text{ in } v]$ 
  using encoding[axiom-instance] qml-2[axiom-instance] by (rule  $\equiv I$ )

```

lemma *en-eq-3*[PLM]:
 $[\Diamond \{x, F\} \equiv \{x, F\} \text{ in } v]$
using *encoding*[*axiom-instance*] *derived-S5-rules-2-b* $\equiv I$ *T* \Diamond **by** *auto*

lemma *en-eq-4*[PLM]:
 $[(\{x, F\} \equiv \{y, G\}) \equiv (\Box \{x, F\} \equiv \Box \{y, G\}) \text{ in } v]$
by (*metis CP en-eq-2* $\equiv I \equiv E(1) \equiv E(2)$)

lemma *en-eq-5*[PLM]:
 $[\Box (\{x, F\} \equiv \{y, G\}) \equiv (\Box \{x, F\} \equiv \Box \{y, G\}) \text{ in } v]$
using $\equiv I$ *KBasic-6 encoding*[*axiom-necessitation*, *axiom-instance*]
sc-eq-box-box-3[*deduction*] **&I** **by** *simp*

lemma *en-eq-6*[PLM]:
 $[(\{x, F\} \equiv \{y, G\}) \equiv \Box (\{x, F\} \equiv \{y, G\}) \text{ in } v]$
using *en-eq-4 en-eq-5 oth-class-taut-4-a* $\equiv E(6)$ **by** *meson*

lemma *en-eq-7*[PLM]:
 $[(\neg \{x, F\}) \equiv \Box (\neg \{x, F\}) \text{ in } v]$
using *en-eq-3*[*THEN id-nec2-2-Aux*] **by** *blast*

lemma *en-eq-8*[PLM]:
 $[\Diamond (\neg \{x, F\}) \equiv (\neg \{x, F\}) \text{ in } v]$
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]:
 $[\Diamond (\neg \{x, F\}) \equiv \Box (\neg \{x, F\}) \text{ in } v]$
using *en-eq-8 en-eq-7* $\equiv E(5)$ **by** *blast*

lemma *en-eq-10*[PLM]:
 $[\mathcal{A} \{x, F\} \equiv \{x, F\} \text{ in } v]$
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 $\bigwedge x. [\varphi (x^P) \equiv \psi (x^P) \text{ in } v]$
shows $[(\lambda y. \varphi (y^P), x^P) \equiv (\lambda y. \psi (y^P), x^P) \text{ in } v]$
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 $\bigwedge x y. [\varphi (x^P) (y^P) \equiv \psi (x^P) (y^P) \text{ in } v]$
shows $[(\lambda^2 (\lambda x y. \varphi (x^P) (y^P)), x^P, y^P) \equiv (\lambda^2 (\lambda x y. \psi (x^P) (y^P)), x^P, y^P) \text{ in } v]$
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 $\bigwedge x y z. [\varphi (x^P) (y^P) (z^P) \equiv \psi (x^P) (y^P) (z^P) \text{ in } v]$
shows $[(\lambda^3 (\lambda x y z. \varphi (x^P) (y^P) (z^P)), x^P, y^P, z^P)]$
 $\equiv (\lambda^3 (\lambda x y z. \psi (x^P) (y^P) (z^P)), x^P, y^P, z^P) \text{ in } v]$
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 $[(\Box(\forall x. \varphi (x^P) \equiv \psi (x^P))) \rightarrow$
 $(\Box(\forall x. (\lambda y. \varphi (y^P), x^P) \equiv (\lambda y. \psi (y^P), x^P))) \text{ in } v]$
apply (*rule qml-1*[*axiom-instance*, *deduction*])
apply (*rule RN*)
proof (*rule CP*, *rule* $\forall I$)
fix $v x$
assume $[\forall x. \varphi (x^P) \equiv \psi (x^P) \text{ in } v]$
hence $\bigwedge x. [\varphi (x^P) \equiv \psi (x^P) \text{ in } v]$
by *PLM-solver*
thus $[(\lambda y. \varphi (y^P), x^P) \equiv (\lambda y. \psi (y^P), x^P) \text{ in } v]$
using *assms beta-equiv-eq-1-1* **by** *auto*
qed

lemma *beta-equiv-eq-2-2*[*PLM*]:

assumes *IsProperInXY* φ
and *IsProperInXY* ψ
shows $[(\Box(\forall x y. \varphi (x^P) (y^P) \equiv \psi (x^P) (y^P))) \rightarrow$
 $(\Box(\forall x y. (\lambda^2 (\lambda x y. \varphi (x^P) (y^P)), x^P, y^P) \equiv$
 $(\lambda^2 (\lambda x y. \psi (x^P) (y^P)), x^P, y^P))) \text{ in } v]$
apply (*rule qml-1*[*axiom-instance*, *deduction*])
apply (*rule RN*)
proof (*rule CP*, *rule* $\forall I$, *rule* $\forall I$)
fix $v x y$
assume $[\forall x y. \varphi (x^P) (y^P) \equiv \psi (x^P) (y^P) \text{ in } v]$
hence $(\bigwedge x y. [\varphi (x^P) (y^P) \equiv \psi (x^P) (y^P) \text{ in } v])$
by (*meson* $\forall E$)
thus $[(\lambda^2 (\lambda x y. \varphi (x^P) (y^P)), x^P, y^P) \equiv$
 $(\lambda^2 (\lambda x y. \psi (x^P) (y^P)), x^P, y^P) \text{ in } v]$
using *assms beta-equiv-eq-1-2* **by** *auto*
qed

lemma *beta-equiv-eq-2-3*[*PLM*]:

assumes *IsProperInXYZ* φ
and *IsProperInXYZ* ψ
shows $[(\Box(\forall x y z. \varphi (x^P) (y^P) (z^P) \equiv \psi (x^P) (y^P) (z^P))) \rightarrow$
 $(\Box(\forall x y z. (\lambda^3 (\lambda x y z. \varphi (x^P) (y^P) (z^P)), x^P, y^P, z^P) \equiv$
 $(\lambda^3 (\lambda x y z. \psi (x^P) (y^P) (z^P)), x^P, y^P, z^P))) \text{ in } v]$
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 $[\forall x y z. \varphi (x^P) (y^P) (z^P) \equiv \psi (x^P) (y^P) (z^P) \text{ in } v]$
hence $(\bigwedge x y z. [\varphi (x^P) (y^P) (z^P) \equiv \psi (x^P) (y^P) (z^P) \text{ in } v])$
by (*meson* $\forall E$)
thus $[(\lambda^3 (\lambda x y z. \varphi (x^P) (y^P) (z^P)), x^P, y^P, z^P) \equiv$
 $(\lambda^3 (\lambda x y z. \psi (x^P) (y^P) (z^P)), x^P, y^P, z^P) \text{ in } v]$

using *assms beta-equiv-eq-1-3* **by** *auto*
qed

lemma *beta-C-meta-1*[*PLM*]:

assumes *IsProperInX* φ
shows $[(\lambda y. \varphi (y^P), x^P) \equiv \varphi (x^P) \text{ in } v]$
using *lambda-predicates-2-1*[*OF assms, axiom-instance*] **by** *auto*

lemma *beta-C-meta-2*[*PLM*]:

assumes *IsProperInXY* φ
shows $[(\lambda^2 (\lambda x y. \varphi (x^P) (y^P)), x^P, y^P) \equiv \varphi (x^P) (y^P) \text{ in } v]$
using *lambda-predicates-2-2*[*OF assms, axiom-instance*] **by** *auto*

lemma *beta-C-meta-3*[*PLM*]:

assumes *IsProperInXYZ* φ
shows $[(\lambda^3 (\lambda x y z. \varphi (x^P) (y^P) (z^P)), x^P, y^P, z^P) \equiv \varphi (x^P) (y^P) (z^P) \text{ in } v]$
using *lambda-predicates-2-3*[*OF assms, axiom-instance*] **by** *auto*

lemma *relations-1*[*PLM*]:

assumes *IsProperInX* φ
shows $[\exists F. \Box (\forall x. (F, x^P) \equiv \varphi (x^P)) \text{ in } v]$
using *assms* **apply** – **by** *PLM-solver*

lemma *relations-2*[*PLM*]:

assumes *IsProperInXY* φ
shows $[\exists F. \Box (\forall x y. (F, x^P, y^P) \equiv \varphi (x^P) (y^P)) \text{ in } v]$
using *assms* **apply** – **by** *PLM-solver*

lemma *relations-3*[*PLM*]:

assumes *IsProperInXYZ* φ
shows $[\exists F. \Box (\forall x y z. (F, x^P, y^P, z^P) \equiv \varphi (x^P) (y^P) (z^P)) \text{ in } v]$
using *assms* **apply** – **by** *PLM-solver*

lemma *prop-equiv*[*PLM*]:

shows $[(\forall x. (\{x^P, F\} \equiv \{x^P, G\})) \rightarrow F = G \text{ in } v]$
proof (*rule CP*)
assume *1*: $[\forall x. \{x^P, F\} \equiv \{x^P, G\} \text{ in } v]$
{
fix *x*
have $[\{x^P, F\} \equiv \{x^P, G\} \text{ in } v]$
using *1* **by** (*rule* $\forall E$)
hence $[\Box (\{x^P, F\} \equiv \{x^P, G\}) \text{ in } v]$
using *PLM.en-eq-6* $\equiv E(1)$ **by** *blast*
}
hence $[\forall x. \Box (\{x^P, F\} \equiv \{x^P, G\}) \text{ in } v]$
by (*rule* $\forall I$)
thus $[F = G \text{ in } v]$
unfolding *identity-defs*
by (*rule* *BF*[*deduction*])
qed

lemma *propositions-lemma-1*[*PLM*]:

$[\lambda^0 \varphi = \varphi \text{ in } v]$
using *lambda-predicates-3-0*[*axiom-instance*] .

lemma *propositions-lemma-2*[*PLM*]:

$[\lambda^0 \varphi \equiv \varphi \text{ in } v]$
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 $\bigwedge x. [\mathcal{A}(\varphi x \equiv \psi x) \text{ in } v]$
shows $[(\chi :: \kappa \Rightarrow o) (\iota x. \varphi x) = \chi (\iota x. \psi x) \text{ in } v]$
proof –
 have $[\lambda^0 (\chi (\iota x. \varphi x)) = \lambda^0 (\chi (\iota x. \psi x)) \text{ in } v]$
 using *assms lambda-predicates-4-0*[*axiom-instance*]
 by *blast*
 hence $[(\chi (\iota x. \varphi x)) = \lambda^0 (\chi (\iota x. \psi x)) \text{ in } v]$
 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*]:
 $[\exists p. \Box(p \equiv p') \text{ in } v]$
by *PLM-solver*

lemma *pos-not-equiv-then-not-eq*[*PLM*]:
 $[\Diamond(\neg(\forall x. (\Downarrow F, x^P) \equiv (\Downarrow G, x^P))) \rightarrow F \neq G \text{ in } v]$
unfolding *diamond-def*
proof (*subst contraposition-1*[*symmetric*], *rule CP*)
 assume $[F = G \text{ in } v]$
 thus $[\Box(\neg(\neg(\forall x. (\Downarrow F, x^P) \equiv (\Downarrow G, x^P)))) \text{ in } v]$
 apply (*rule l-identity*[*axiom-instance*, *deduction*, *deduction*])
 by *PLM-solver*
qed

lemma *thm-relation-negation-1-1*[*PLM*]:
 $[(\Downarrow F^-, x^P) \equiv \neg(\Downarrow F, x^P) \text{ in } v]$
unfolding *propnot-defs*
apply (*rule lambda-predicates-2-1*[*axiom-instance*])
by *show-proper*

lemma *thm-relation-negation-1-2*[*PLM*]:
 $[(\Downarrow F^-, x^P, y^P) \equiv \neg(\Downarrow F, x^P, y^P) \text{ in } v]$
unfolding *propnot-defs*
apply (*rule lambda-predicates-2-2*[*axiom-instance*])
by *show-proper*

lemma *thm-relation-negation-1-3*[*PLM*]:
 $[(\Downarrow F^-, x^P, y^P, z^P) \equiv \neg(\Downarrow F, x^P, y^P, z^P) \text{ in } v]$
unfolding *propnot-defs*
apply (*rule lambda-predicates-2-3*[*axiom-instance*])
by *show-proper*

lemma *thm-relation-negation-2-1*[*PLM*]:
 $[(\neg(\Downarrow F^-, x^P)) \equiv (\Downarrow F, x^P) \text{ in } v]$
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*]:
 $[(\neg(\Downarrow F^-, x^P, y^P)) \equiv (\Downarrow F, x^P, y^P) \text{ in } v]$

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*]:
 $[(\neg(\neg F^-, x^P, y^P, z^P)) \equiv (F, x^P, y^P, z^P)]$ 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*]:
 $[(\neg((p::o)^-)) \equiv p]$ 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*]:
 $[(p::o) \neq (p^-)]$ 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*]:
 $[((p::o)^-) = \neg p]$ in *v*
unfolding *propnot-defs* **using** *propositions-lemma-1* **by** *simp*


```

lemma thm-relation-negation-8[PLM]:
  [(p::o)  $\neq$   $\neg p$  in v]
  unfolding propnot-defs
  using id-eq-prop-prop-8-b[deduction]
    l-identity[where  $\varphi = \lambda G . G \equiv \neg(p)$ , 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)  $\rightarrow$  (( $\neg p$ ) = ( $\neg q$ )) in v]
  using l-identity[where  $\alpha = p$  and  $\beta = q$  and  $\varphi = \lambda x . (\neg p) = (\neg x)$ ,
    axiom-instance, deduction]
    id-eq-prop-prop-7-b using CP modus-ponens by blast

lemma thm-relation-negation-10[PLM]:
  [((p::o) = q)  $\rightarrow$  (( $p^-$ ) = ( $q^-$ )) in v]
  using l-identity[where  $\alpha = p$  and  $\beta = q$  and  $\varphi = \lambda x . (p^-) = (x^-)$ ,
    axiom-instance, deduction]
    id-eq-prop-prop-7-b using CP modus-ponens by blast

lemma thm-cont-prop-1[PLM]:
  [NonContingent ( $F::\Pi_1$ )  $\equiv$  NonContingent ( $F^-$ ) in v]
  proof (rule  $\equiv I$ ; rule CP)
    assume [NonContingent  $F$  in v]
    hence [ $\Box(\forall x. \langle F, x^P \rangle) \vee \Box(\forall x. \neg \langle F, x^P \rangle)$  in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs .
    hence [ $\Box(\forall x. \neg \langle F^-, x^P \rangle) \vee \Box(\forall x. \neg \langle F, x^P \rangle)$  in v]
      apply -
      apply (PLM-subst-method  $\lambda x . \langle F, x^P \rangle \lambda x . \neg \langle F^-, x^P \rangle$ )
      using thm-relation-negation-2-1[equiv-sym] by auto
    hence [ $\Box(\forall x. \neg \langle F^-, x^P \rangle) \vee \Box(\forall x. \langle F^-, x^P \rangle)$  in v]
      apply -
      apply (PLM-subst-goal-method
         $\lambda \varphi . \Box(\forall x. \neg \langle F^-, x^P \rangle) \vee \Box(\forall x. \varphi x) \lambda x . \neg \langle F, x^P \rangle$ )
      using thm-relation-negation-1-1[equiv-sym] by auto
    hence [ $\Box(\forall x. \langle F^-, x^P \rangle) \vee \Box(\forall x. \neg \langle F^-, x^P \rangle)$  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 [ $\Box(\forall x. \neg \langle F^-, x^P \rangle) \vee \Box(\forall x. \langle F^-, x^P \rangle)$  in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs
      by (rule oth-class-taut-3-e[equiv-lr])
    hence [ $\Box(\forall x. \langle F, x^P \rangle) \vee \Box(\forall x. \langle F^-, x^P \rangle)$  in v]
      apply -
      apply (PLM-subst-method  $\lambda x . \neg \langle F^-, x^P \rangle \lambda x . \langle F, x^P \rangle$ )
      using thm-relation-negation-2-1 by auto
    hence [ $\Box(\forall x. \langle F, x^P \rangle) \vee \Box(\forall x. \neg \langle F, x^P \rangle)$  in v]
      apply -
      apply (PLM-subst-method  $\lambda x . \langle F^-, x^P \rangle \lambda x . \neg \langle F, x^P \rangle$ )
      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 $F \equiv \Diamond(\exists x . \langle F, x^P \rangle) \ \& \ \Diamond(\exists x . \neg \langle F, x^P \rangle)$ in v]

proof (rule $\equiv I$; rule CP)

assume [Contingent F in v]

hence [$\neg(\Box(\forall x . \langle F, x^P \rangle) \vee \Box(\forall x . \neg \langle F, x^P \rangle))$ in v]

unfolding Contingent-def Necessary-defs Impossible-defs .

hence [$(\neg\Box(\forall x . \langle F, x^P \rangle)) \ \& \ (\neg\Box(\forall x . \neg \langle F, x^P \rangle))$ in v]

by (rule oth-class-taut-6-d[equiv-lr])

hence [$(\Diamond\neg(\forall x . \neg \langle F, x^P \rangle)) \ \& \ (\Diamond\neg(\forall x . \langle F, x^P \rangle))$ in v]

using KBasic2-2[equiv-lr] &I &E by meson

thus [$(\Diamond(\exists x . \langle F, x^P \rangle)) \ \& \ (\Diamond(\exists x . \neg \langle F, x^P \rangle))$ in v]

unfolding exists-def apply -

apply (PLM-subst-method $\lambda x . \langle F, x^P \rangle \ \lambda x . \neg \neg \langle F, x^P \rangle$)

using oth-class-taut-4-b by auto

next

assume [$(\Diamond(\exists x . \langle F, x^P \rangle)) \ \& \ (\Diamond(\exists x . \neg \langle F, x^P \rangle))$ in v]

hence [$(\Diamond\neg(\forall x . \neg \langle F, x^P \rangle)) \ \& \ (\Diamond\neg(\forall x . \langle F, x^P \rangle))$ in v]

unfolding exists-def apply -

apply (PLM-subst-goal-method

$\lambda \varphi . (\Diamond\neg(\forall x . \neg \langle F, x^P \rangle)) \ \& \ (\Diamond\neg(\forall x . \varphi x)) \ \lambda x . \neg \neg \langle F, x^P \rangle$)

using oth-class-taut-4-b[equiv-sym] by auto

hence [$(\neg\Box(\forall x . \langle F, x^P \rangle)) \ \& \ (\neg\Box(\forall x . \neg \langle F, x^P \rangle))$ in v]

using KBasic2-2[equiv-rl] &I &E by meson

hence [$\neg(\Box(\forall x . \langle F, x^P \rangle) \vee \Box(\forall x . \neg \langle F, x^P \rangle))$ 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 $(F::\Pi_1) \equiv$ 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]:

[$\Diamond(\exists x . \langle F, x^P \rangle) \ \& \ (\Diamond(\neg \langle F, x^P \rangle)) \equiv \Diamond(\exists x . ((\neg \langle F, x^P \rangle) \ \& \ \Diamond \langle F, x^P \rangle))$ in v]

proof -

have [$\Diamond(\exists x . \langle F, x^P \rangle) \ \& \ (\Diamond(\neg \langle F, x^P \rangle))$ in v]

= [$(\exists x . \Diamond(\langle F, x^P \rangle) \ \& \ \Diamond(\neg \langle F, x^P \rangle))$ in v]

using BFD[deduction] CBF[deduction] by fast

also have ... = [$\exists x . (\Diamond \langle F, x^P \rangle) \ \& \ \Diamond(\neg \langle F, x^P \rangle)$ in v]

apply (PLM-subst-method

$\lambda x . \Diamond(\langle F, x^P \rangle) \ \& \ \Diamond(\neg \langle F, x^P \rangle)$

$\lambda x . \Diamond \langle F, x^P \rangle \ \& \ \Diamond(\neg \langle F, x^P \rangle)$)

using S5Basic-12 by auto

also have ... = [$\exists x . \Diamond(\neg \langle F, x^P \rangle) \ \& \ \Diamond \langle F, x^P \rangle$ in v]

apply (PLM-subst-method

$\lambda x . \Diamond \langle F, x^P \rangle \ \& \ \Diamond(\neg \langle F, x^P \rangle)$

$\lambda x . \Diamond(\neg \langle F, x^P \rangle) \ \& \ \Diamond \langle F, x^P \rangle$)

using oth-class-taut-3-b by auto

also have ... = [$\exists x . \Diamond((\neg \langle F, x^P \rangle) \ \& \ \Diamond \langle F, x^P \rangle)$ in v]

apply (PLM-subst-method

$\lambda x . \Diamond(\neg \langle F, x^P \rangle) \ \& \ \Diamond \langle F, x^P \rangle$

$\lambda x . \Diamond((\neg \langle F, x^P \rangle) \ \& \ \Diamond \langle F, x^P \rangle)$)

using S5Basic-12[equiv-sym] by auto

also have ... = [$\Diamond(\exists x . ((\neg \langle F, x^P \rangle) \ \& \ \Diamond \langle F, x^P \rangle))$ in v]

using CBF[deduction] BFD[deduction] by fast

finally show *?thesis* using $\equiv I$ CP by blast
qed

lemma *lem-cont-e-2*[PLM]:

$[\Diamond(\exists x . \langle F, x^P \rangle \ \& \ \Diamond(\neg \langle F, x^P \rangle)) \equiv \Diamond(\exists x . \langle F^-, x^P \rangle \ \& \ \Diamond(\neg \langle F^-, x^P \rangle)) \text{ in } v]$
apply (*PLM-subst-method* $\lambda x . \langle F, x^P \rangle \ \lambda x . \neg \langle F^-, x^P \rangle$)
using *thm-relation-negation-2-1*[*equiv-sym*] **apply** *simp*
apply (*PLM-subst-method* $\lambda x . \neg \langle F, x^P \rangle \ \lambda x . \langle F^-, x^P \rangle$)
using *thm-relation-negation-1-1*[*equiv-sym*] **apply** *simp*
using *lem-cont-e* **by** *simp*

lemma *thm-cont-e-1*[PLM]:

$[\Diamond(\exists x . ((\neg \langle E!, x^P \rangle) \ \& \ (\Diamond \langle E!, x^P \rangle))) \text{ in } v]$
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]:

$[\exists (F::\Pi_1) G . (F \neq G \ \& \ \text{Contingent } F \ \& \ \text{Contingent } G) \text{ in } v]$
apply (*rule-tac* $\alpha=E!$ **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** $L \equiv (\lambda x . \langle E!, x^P \rangle \rightarrow \langle E!, x^P \rangle)$

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* $\forall I(1)$)

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]*:
 $[\exists (F::\Pi_1) G . F \neq G \ \& \ NonContingent F \ \& \ NonContingent G \text{ in } v]$
apply (*rule-tac* $\alpha=L$ **in** $\exists I$, *rule-tac* $\alpha=L^-$ **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]*:
 $[NonContingent (F::\Pi_1) \rightarrow \neg(\exists G . (Contingent G \ \& \ G = F)) \text{ in } v]$
proof (*rule* *CP*)
assume $[NonContingent F \text{ in } v]$
hence $[\neg(Contingent F) \text{ in } v]$
unfolding *NonContingent-def* *Contingent-def*
apply – **by** *PLM-solver*
moreover {
assume $[\exists G . Contingent G \ \& \ G = F \text{ in } v]$
then obtain *P* **where** $[Contingent P \ \& \ P = F \text{ in } v]$
by (*rule* $\exists E$)
hence $[Contingent F \text{ in } v]$
using $\&E$ *l-identity*[*axiom-instance*, *deduction*, *deduction*]
by *blast*
}
ultimately show $[\neg(\exists G . Contingent G \ \& \ G = F) \text{ in } v]$
using *modus-tollens-1* *CP* **by** *blast*
qed

lemma *four-distinct-2[PLM]*:
 $[Contingent (F::\Pi_1) \rightarrow \neg(\exists G . (NonContingent G \ \& \ G = F)) \text{ in } v]$
proof (*rule* *CP*)
assume $[Contingent F \text{ in } v]$
hence $[\neg(NonContingent F) \text{ in } v]$
unfolding *NonContingent-def* *Contingent-def*
apply – **by** *PLM-solver*
moreover {
assume $[\exists G . NonContingent G \ \& \ G = F \text{ in } v]$
then obtain *P* **where** $[NonContingent P \ \& \ P = F \text{ in } v]$
by (*rule* $\exists E$)
hence $[NonContingent F \text{ in } v]$
using $\&E$ *l-identity*[*axiom-instance*, *deduction*, *deduction*]
by *blast*
}
ultimately show $[\neg(\exists G . NonContingent G \ \& \ G = F) \text{ in } v]$
using *modus-tollens-1* *CP* **by** *blast*
qed

lemma *four-distinct-3[PLM]*:
 $[L \neq (L^-) \ \& \ L \neq E! \ \& \ L \neq (E!^-) \ \& \ (L^-) \neq E!$
 $\ \& \ (L^-) \neq (E!^-) \ \& \ E! \neq (E!^-) \text{ in } v]$
proof (*rule* $\&I$)
show $[L \neq (L^-) \text{ in } v]$
by (*rule* *thm-relation-negation-5-1*)
next
{
assume $[L = E! \text{ in } v]$

```

    hence [NonContingent  $L \ \& \ L = E!$  in  $v$ ]
      using thm-noncont-e-e-3 &I by auto
    hence [ $\exists \ G \ . \ NonContingent \ G \ \& \ G = E!$  in  $v$ ]
      using thm-noncont-e-e-3 &I  $\exists I$  by fast
  }
  thus [ $L \neq E!$  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 [ $\exists \ G \ . \ NonContingent \ G \ \& \ G = (E!^-)$  in  $v$ ]
    using thm-noncont-e-e-3 &I  $\exists I$  by fast
}
thus [ $L \neq (E!^-)$  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 [ $\exists \ G \ . \ NonContingent \ G \ \& \ G = E!$  in  $v$ ]
    using thm-noncont-e-e-3 &I  $\exists I$  by fast
}
thus [ $(L^-) \neq E!$  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 [ $\exists \ G \ . \ NonContingent \ G \ \& \ G = (E!^-)$  in  $v$ ]
    using thm-noncont-e-e-3 &I  $\exists I$  by fast
}
thus [ $(L^-) \neq (E!^-)$  in  $v$ ]
  using four-distinct-2[deduction, OF thm-cont-e-3]
    modus-tollens-1 CP
  by blast
next
  show [ $E! \neq (E!^-)$  in  $v$ ]
    by (rule thm-relation-negation-5-1)
qed
end

```

```

lemma thm-cont-propos-1[PLM]:
  [NonContingent  $(p::o) \equiv NonContingent \ (p^-)$  in  $v$ ]
proof (rule  $\equiv I$ ; rule CP)
  assume [NonContingent  $p$  in  $v$ ]
  hence [ $\Box p \vee \Box \neg p$  in  $v$ ]
    unfolding NonContingent-def Necessary-defs Impossible-defs .
  hence [ $\Box(\neg(p^-)) \vee \Box(\neg p)$  in  $v$ ]

```

```

    apply -
    apply (PLM-subst-method  $p \neg(p^-)$ )
    using thm-relation-negation-4[equiv-sym] by auto
  hence  $[\Box(\neg(p^-)) \vee \Box(p^-) \text{ in } v]$ 
    apply -
    apply (PLM-subst-goal-method  $\lambda\varphi . \Box(\neg(p^-)) \vee \Box(\varphi) \neg p$ )
    using thm-relation-negation-3[equiv-sym] by auto
  hence  $[\Box(p^-) \vee \Box(\neg(p^-)) \text{ in } v]$ 
    by (rule oth-class-taut-3-e[equiv-lr])
  thus  $[NonContingent(p^-) \text{ in } v]$ 
    unfolding NonContingent-def Necessary-defs Impossible-defs .
next
  assume  $[NonContingent(p^-) \text{ in } v]$ 
  hence  $[\Box(\neg(p^-)) \vee \Box(p^-) \text{ in } v]$ 
    unfolding NonContingent-def Necessary-defs Impossible-defs
    by (rule oth-class-taut-3-e[equiv-lr])
  hence  $[\Box(p) \vee \Box(p^-) \text{ in } v]$ 
    apply -
    apply (PLM-subst-goal-method  $\lambda\varphi . \Box\varphi \vee \Box(p^-) \neg(p^-)$ )
    using thm-relation-negation-4 by auto
  hence  $[\Box(p) \vee \Box(\neg p) \text{ in } v]$ 
    apply -
    apply (PLM-subst-method  $p^- \neg p$ )
    using thm-relation-negation-3 by auto
  thus  $[NonContingent p \text{ in } v]$ 
    unfolding NonContingent-def Necessary-defs Impossible-defs .
qed

```

lemma *thm-cont-propos-2[PLM]:*
 $[Contingent p \equiv \Diamond p \ \& \ \Diamond(\neg p) \text{ in } v]$
proof (rule $\equiv I$; rule *CP*)
 assume $[Contingent p \text{ in } v]$
 hence $[\neg(\Box p \vee \Box(\neg p)) \text{ in } v]$
 unfolding Contingent-def Necessary-defs Impossible-defs .
 hence $[(\neg\Box p) \ \& \ (\neg\Box(\neg p)) \text{ in } v]$
 by (rule oth-class-taut-6-d[equiv-lr])
 hence $[(\Diamond\neg(\neg p)) \ \& \ (\Diamond\neg p) \text{ in } v]$
 using KBasic2-2[equiv-lr] &I &E by meson
 thus $[(\Diamond p) \ \& \ (\Diamond(\neg p)) \text{ in } v]$
 apply - apply PLM-solver
 apply (PLM-subst-method $\neg\neg p$)
 using oth-class-taut-4-b[equiv-sym] by auto
next
 assume $[(\Diamond p) \ \& \ (\Diamond\neg(p)) \text{ in } v]$
 hence $[(\Diamond\neg(\neg p)) \ \& \ (\Diamond\neg(p)) \text{ in } v]$
 apply - apply PLM-solver
 apply (PLM-subst-method $p \neg\neg p$)
 using oth-class-taut-4-b by auto
 hence $[(\neg\Box p) \ \& \ (\neg\Box(\neg p)) \text{ in } v]$
 using KBasic2-2[equiv-rl] &I &E by meson
 hence $[\neg(\Box(p) \vee \Box(\neg p)) \text{ in } v]$
 by (rule oth-class-taut-6-d[equiv-rl])
 thus $[Contingent p \text{ in } v]$
 unfolding Contingent-def Necessary-defs Impossible-defs .
qed

lemma *thm-cont-propos-3[PLM]:*
 $[Contingent(p::o) \equiv Contingent(p^-) \text{ in } v]$

using *thm-cont-propos-1*
unfolding *NonContingent-def Contingent-def*
by (*rule oth-class-taut-5-d[equiv-lr]*)

context

begin

private definition p_0 **where**
 $p_0 \equiv \forall x. (E!, x^P) \rightarrow (E!, x^P)$

lemma *thm-noncont-propos-1*[*PLM*]:

[*Necessary* p_0 in v]

unfolding *Necessary-defs* p_0 -*def*

apply (*rule RN*, *rule* $\forall I$)

using *if-p-then-p* .

lemma *thm-noncont-propos-2*[*PLM*]:

[*Impossible* (p_0^-) in v]

unfolding *Impossible-defs*

apply (*PLM-subst-method* $\neg p_0$ p_0^-)

using *thm-relation-negation-3[equiv-sym]* **apply** *simp*

apply (*PLM-subst-method* p_0 $\neg \neg p_0$)

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* (p_0) in v]

unfolding *NonContingent-def* **using** *thm-noncont-propos-1*

by (*rule* $\forall I(1)$)

lemma *thm-noncont-propos-4*[*PLM*]:

[*NonContingent* (p_0^-) in v]

unfolding *NonContingent-def* **using** *thm-noncont-propos-2*

by (*rule* $\forall I(2)$)

lemma *thm-noncont-propos-5*[*PLM*]:

[$\exists (p::o) q . p \neq q \ \& \ NonContingent \ p \ \& \ NonContingent \ q$ in v]

apply (*rule-tac* $\alpha=p_0$ **in** $\exists I$, *rule-tac* $\alpha=p_0^-$ **in** $\exists I$)

using $\exists I$ *thm-relation-negation-6* *thm-noncont-propos-3*

thm-noncont-propos-4 **&I** **by** *simp*

private definition q_0 **where**

$q_0 \equiv \exists x . (E!, x^P) \ \& \ \Diamond(\neg(E!, x^P))$

lemma *basic-prop-1*[*PLM*]:

[$\exists p . \Diamond p \ \& \ \Diamond(\neg p)$ in v]

apply (*rule-tac* $\alpha=q_0$ **in** $\exists I$) **unfolding** q_0 -*def*

using *qml-4[axiom-instance]* **by** *simp*

lemma *basic-prop-2*[*PLM*]:

[*Contingent* q_0 in v]

unfolding *Contingent-def Necessary-defs Impossible-defs*

apply (*rule oth-class-taut-6-d[equiv-rl]*)

apply (*PLM-subst-goal-method* $\lambda \varphi . (\neg \Box(\varphi))$ **&** $\neg \Box \neg q_0 \ \neg \neg q_0$)

using *oth-class-taut-4-b[equiv-sym]* **apply** *simp*

using *qml-4[axiom-instance, conj-sym]*

unfolding q_0 -*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]:
  [ $\exists (p::o) q . p \neq q \ \& \text{Contingent } p \ \& \text{Contingent } q$  in  $v$ ]
  apply (rule-tac  $\alpha=q_0$  in  $\exists I$ , rule-tac  $\alpha=q_0^-$  in  $\exists I$ )
  using thm-relation-negation-6 basic-prop-2 basic-prop-3  $\&I$  by simp

lemma four-distinct-props-1[PLM]:
  [NonContingent ( $p::\Pi_0$ )  $\rightarrow (\neg(\exists q . \text{Contingent } q \ \& \ q = p))$  in  $v$ ]
  proof (rule CP)
    assume [NonContingent  $p$  in  $v$ ]
    hence [ $\neg(\text{Contingent } p)$  in  $v$ ]
      unfolding NonContingent-def Contingent-def
      apply – by PLM-solver
    moreover {
      assume [ $\exists q . \text{Contingent } q \ \& \ q = p$  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 [ $\neg(\exists q . \text{Contingent } q \ \& \ q = p)$  in  $v$ ]
      using modus-tollens-1 CP by blast
  qed

lemma four-distinct-props-2[PLM]:
  [Contingent ( $p::o$ )  $\rightarrow \neg(\exists q . (\text{NonContingent } q \ \& \ q = p))$  in  $v$ ]
  proof (rule CP)
    assume [Contingent  $p$  in  $v$ ]
    hence [ $\neg(\text{NonContingent } p)$  in  $v$ ]
      unfolding NonContingent-def Contingent-def
      apply – by PLM-solver
    moreover {
      assume [ $\exists q . \text{NonContingent } q \ \& \ q = p$  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 [ $\neg(\exists q . \text{NonContingent } q \ \& \ q = p)$  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 [ $\exists q . \text{NonContingent } q \ \& \ q = q_0$  in  $v$ ]
    }

```



```

    using &I thm-noncont-propos-3  $\exists I$ [where  $\alpha=p_0$ ]
    by simp
  }
  thus  $[p_0 \neq q_0 \text{ in } v]$ 
    using four-distinct-props-2[deduction, OF basic-prop-2]
      modus-tollens-1 CP
    by blast
next
{
  assume  $[p_0 = (q_0^-) \text{ in } v]$ 
  hence  $[\exists q . \text{NonContingent } q \ \& \ q = (q_0^-) \text{ in } v]$ 
    using thm-noncont-propos-3 &I  $\exists I$ [where  $\alpha=p_0$ ] by simp
}
  thus  $[p_0 \neq (q_0^-) \text{ in } v]$ 
    using four-distinct-props-2[deduction, OF basic-prop-3]
      modus-tollens-1 CP
    by blast
next
{
  assume  $[(p_0^-) = q_0 \text{ in } v]$ 
  hence  $[\exists q . \text{NonContingent } q \ \& \ q = q_0 \text{ in } v]$ 
    using thm-noncont-propos-4 &I  $\exists I$ [where  $\alpha=p_0^-$ ] by auto
}
  thus  $[(p_0^-) \neq q_0 \text{ in } v]$ 
    using four-distinct-props-2[deduction, OF basic-prop-2]
      modus-tollens-1 CP
    by blast
next
{
  assume  $[(p_0^-) = (q_0^-) \text{ in } v]$ 
  hence  $[\exists q . \text{NonContingent } q \ \& \ q = (q_0^-) \text{ in } v]$ 
    using thm-noncont-propos-4 &I  $\exists I$ [where  $\alpha=p_0^-$ ] by auto
}
  thus  $[(p_0^-) \neq (q_0^-) \text{ in } v]$ 
    using four-distinct-props-2[deduction, OF basic-prop-3]
      modus-tollens-1 CP
    by blast
next
  show  $[q_0 \neq (q_0^-) \text{ in } v]$ 
    by (rule thm-relation-negation-6)
qed

```

lemma *cont-true-cont-1*[PLM]:

```

[ContingentlyTrue  $p \rightarrow$  Contingent  $p$  in  $v$ ]
apply (rule CP, rule thm-cont-propos-2[equiv-rl])
unfolding ContingentlyTrue-def
apply (rule &I, drule &E(1))
using  $T\Diamond$ [deduction] apply simp
by (rule &E(2))

```

lemma *cont-true-cont-2*[PLM]:

```

[ContingentlyFalse  $p \rightarrow$  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  $T\Diamond$ [deduction] by simp

```

```

lemma cont-true-cont-3[PLM]:
  [ContingentlyTrue  $p \equiv \text{ContingentlyFalse } (p^-)$  in  $v$ ]
  unfolding ContingentlyTrue-def ContingentlyFalse-def
  apply (PLM-subst-method  $\neg p \ p^-$ )
    using thm-relation-negation-3[equiv-sym] apply simp
  apply (PLM-subst-method  $p \ \neg\neg p$ )
  by PLM-solver +

lemma cont-true-cont-4[PLM]:
  [ContingentlyFalse  $p \equiv \text{ContingentlyTrue } (p^-)$  in  $v$ ]
  unfolding ContingentlyTrue-def ContingentlyFalse-def
  apply (PLM-subst-method  $\neg p \ p^-$ )
    using thm-relation-negation-3[equiv-sym] apply simp
  apply (PLM-subst-method  $p \ \neg\neg p$ )
  by PLM-solver +

lemma cont-tf-thm-1[PLM]:
  [ContingentlyTrue  $q_0 \vee \text{ContingentlyFalse } q_0$  in  $v$ ]
  proof –
    have [ $q_0 \vee \neg q_0$  in  $v$ ]
      by PLM-solver
    moreover {
      assume [ $q_0$  in  $v$ ]
      hence [ $q_0 \ \& \ \Diamond\neg q_0$  in  $v$ ]
        unfolding q0-def
        using qml-4[axiom-instance,conj2] &I
        by auto
    }
    moreover {
      assume [ $\neg q_0$  in  $v$ ]
      hence [ $(\neg q_0) \ \& \ \Diamond q_0$  in  $v$ ]
        unfolding q0-def
        using qml-4[axiom-instance,conj1] &I
        by auto
    }
    ultimately show ?thesis
      unfolding ContingentlyTrue-def ContingentlyFalse-def
      using  $\vee E(4)$  CP by auto
  qed

lemma cont-tf-thm-2[PLM]:
  [ContingentlyFalse  $q_0 \vee \text{ContingentlyFalse } (q_0^-)$  in  $v$ ]
  using cont-tf-thm-1 cont-true-cont-3[where  $p=q_0$ ]
    cont-true-cont-4[where  $p=q_0$ ]
  apply – by PLM-solver

lemma cont-tf-thm-3[PLM]:
  [ $\exists \ p . \text{ContingentlyTrue } p$  in  $v$ ]
  proof (rule  $\vee E(1)$ ; (rule CP)?)
    show [ContingentlyTrue  $q_0 \vee \text{ContingentlyFalse } q_0$  in  $v$ ]
      using cont-tf-thm-1 .
  next
    assume [ContingentlyTrue  $q_0$  in  $v$ ]
    thus ?thesis
      using  $\exists I$  by metis
  next
    assume [ContingentlyFalse  $q_0$  in  $v$ ]

```

hence $[ContingentlyTrue (q_0^-) \text{ in } v]$
 using *cont-true-cont-4*[*equiv-lr*] **by** *simp*
 thus *?thesis*
 using $\exists I$ **by** *metis*
qed

lemma *cont-tf-thm-4*[*PLM*]:
 $[\exists p . ContingentlyFalse p \text{ in } v]$
proof (*rule* $\vee E(1)$; (*rule* *CP*)?)
 show $[ContingentlyTrue q_0 \vee ContingentlyFalse q_0 \text{ in } v]$
 using *cont-tf-thm-1* .
next
 assume $[ContingentlyTrue q_0 \text{ in } v]$
 hence $[ContingentlyFalse (q_0^-) \text{ in } v]$
 using *cont-true-cont-3*[*equiv-lr*] **by** *simp*
 thus *?thesis*
 using $\exists I$ **by** *metis*
next
 assume $[ContingentlyFalse q_0 \text{ in } v]$
 thus *?thesis*
 using $\exists I$ **by** *metis*
qed

lemma *cont-tf-thm-5*[*PLM*]:
 $[ContingentlyTrue p \ \& \ Necessary \ q \rightarrow p \neq q \text{ in } v]$
proof (*rule* *CP*)
 assume $[ContingentlyTrue p \ \& \ Necessary \ q \text{ in } v]$
 hence 1: $[\Diamond(\neg p) \ \& \ \Box q \text{ in } v]$
 unfolding *ContingentlyTrue-def* *Necessary-defs*
 using $\&E$ $\&I$ **by** *blast*
 hence $[\neg\Box p \text{ in } v]$
 apply – **apply** (*drule* $\&E(1)$)
 unfolding *diamond-def*
 apply (*PLM-subst-method* $\neg\neg p \ p$)
 using *oth-class-taut-4-b*[*equiv-sym*] **by** *auto*
moreover {
 assume $[p = q \text{ in } v]$
 hence $[\Box p \text{ in } v]$
 using *l-identity*[**where** $\alpha=q$ **and** $\beta=p$ **and** $\varphi=\lambda x . \Box x$,
axiom-instance, *deduction*, *deduction*]
 1[*conj2*] *id-eq-prop-prop-8-b*[*deduction*]
by *blast*
 }
 ultimately show $[p \neq q \text{ in } v]$
 using *modus-tollens-1* *CP* **by** *blast*
qed

lemma *cont-tf-thm-6*[*PLM*]:
 $[(ContingentlyFalse p \ \& \ Impossible \ q) \rightarrow p \neq q \text{ in } v]$
proof (*rule* *CP*)
 assume $[ContingentlyFalse p \ \& \ Impossible \ q \text{ in } v]$
 hence 1: $[\Diamond p \ \& \ \Box(\neg q) \text{ in } v]$
 unfolding *ContingentlyFalse-def* *Impossible-defs*
 using $\&E$ $\&I$ **by** *blast*
 hence $[\neg\Diamond q \text{ in } v]$
 unfolding *diamond-def* **apply** – **by** *PLM-solver*
moreover {
 assume $[p = q \text{ in } v]$

```

    hence  $[\Diamond q \text{ in } v]$ 
      using l-identity[axiom-instance, deduction, deduction] 1[conj1]
      id-eq-prop-prop-8-b[deduction]
    by blast
  }
  ultimately show  $[p \neq q \text{ in } v]$ 
    using modus-tollens-1 CP by blast
qed
end

```

lemma *oa-contingent-1*[*PLM*]:

```

 $[O! \neq A! \text{ in } v]$ 
proof -
{
  assume  $[O! = A! \text{ in } v]$ 
  hence  $[(\lambda x. \Diamond(E!, x^P)) = (\lambda x. \neg \Diamond(E!, x^P)) \text{ in } v]$ 
    unfolding Ordinary-def Abstract-def .
  moreover have  $[(\Diamond(\lambda x. \Diamond(E!, x^P)), x^P) \equiv \Diamond(E!, x^P) \text{ in } v]$ 
    apply (rule beta-C-meta-1)
    by show-proper
  ultimately have  $[(\Diamond(\lambda x. \neg \Diamond(E!, x^P)), x^P) \equiv \Diamond(E!, x^P) \text{ in } v]$ 
    using l-identity[axiom-instance, deduction, deduction] by fast
  moreover have  $[(\Diamond(\lambda x. \neg \Diamond(E!, x^P)), x^P) \equiv \neg \Diamond(E!, x^P) \text{ in } v]$ 
    apply (rule beta-C-meta-1)
    by show-proper
  ultimately have  $[\Diamond(E!, x^P) \equiv \neg \Diamond(E!, x^P) \text{ in } v]$ 
    apply - by PLM-solver
}
thus ?thesis
  using oth-class-taut-1-b modus-tollens-1 CP
  by blast
qed

```

lemma *oa-contingent-2*[*PLM*]:

```

 $[(\Diamond O!, x^P) \equiv \neg(\Diamond A!, x^P) \text{ in } v]$ 
proof -
  have  $[(\Diamond(\lambda x. \neg \Diamond(E!, x^P)), x^P) \equiv \neg \Diamond(E!, x^P) \text{ in } v]$ 
    apply (rule beta-C-meta-1)
    by show-proper
  hence  $[(\neg(\lambda x. \neg \Diamond(E!, x^P)), x^P) \equiv \Diamond(E!, x^P) \text{ in } v]$ 
    using oth-class-taut-5-d[equiv-lr] oth-class-taut-4-b[equiv-sym]
     $\equiv E(5)$  by blast
  moreover have  $[(\Diamond(\lambda x. \Diamond(E!, x^P)), x^P) \equiv \Diamond(E!, x^P) \text{ in } v]$ 
    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*]:

```

 $[(\Diamond A!, x^P) \equiv \neg(\Diamond O!, x^P) \text{ in } v]$ 
using oa-contingent-2
apply - by PLM-solver

```

lemma *oa-contingent-4*[*PLM*]:

```

 $[Contingent O! \text{ in } v]$ 
apply (rule thm-cont-prop-2[equiv-rl], rule &I)

```

```

subgoal
  unfolding Ordinary-def
  apply (PLM-subst-method  $\lambda x . \Diamond(\langle E!, x^P \rangle) \lambda x . (\lambda x. \Diamond(\langle E!, x^P \rangle), x^P)$ )
  apply (safe intro!: beta-C-meta-1[equiv-sym])
  apply show-proper
  using BF $\Diamond$ [deduction, OF thm-cont-prop-2[equiv-lr, OF thm-cont-e-2, conj1]]
  by (rule T $\Diamond$ [deduction])
subgoal
  apply (PLM-subst-method  $\lambda x . \langle A!, x^P \rangle \lambda x . \neg(\langle O!, x^P \rangle)$ )
  using oa-contingent-3 apply simp
  using cqt-further-5[deduction, conj1, OF A-objects[axiom-instance]]
  by (rule T $\Diamond$ [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 $\Diamond$ [deduction])
  subgoal
    unfolding Abstract-def
    apply (PLM-subst-method  $\lambda x . \neg\Diamond(\langle E!, x^P \rangle) \lambda x . (\lambda x. \neg\Diamond(\langle E!, x^P \rangle), x^P)$ )
    apply (safe intro!: beta-C-meta-1[equiv-sym])
    apply show-proper
    apply (PLM-subst-method  $\lambda x . \Diamond(\langle E!, x^P \rangle) \lambda x . \neg\neg\Diamond(\langle E!, x^P \rangle)$ )
    using oth-class-taut-4-b apply simp
    using BF $\Diamond$ [deduction, OF thm-cont-prop-2[equiv-lr, OF thm-cont-e-2, conj1]]
    by (rule T $\Diamond$ [deduction])
  done

lemma oa-contingent-6[PLM]:
  [(O!-)  $\neq$  (A!-) in v]
  proof -
    {
      assume [(O!-) = (A!-) in v]
      hence [( $\lambda x. \neg(\langle O!, x^P \rangle) = (\lambda x. \neg(\langle A!, x^P \rangle))$ ) in v]
        unfolding propnot-defs .
      moreover have [( $\langle (\lambda x. \neg(\langle O!, x^P \rangle), x^P \rangle \equiv \neg(\langle O!, x^P \rangle)$ ) in v]
        apply (rule beta-C-meta-1)
        by show-proper
      ultimately have [( $\langle \lambda x. \neg(\langle A!, x^P \rangle), x^P \rangle \equiv \neg(\langle O!, x^P \rangle)$ ) in v]
        using l-identity[axiom-instance, deduction, deduction]
        by fast
      hence [( $\neg(\langle A!, x^P \rangle) \equiv \neg(\langle O!, x^P \rangle)$ ) in v]
        apply -
        apply (PLM-subst-method ( $\langle \lambda x. \neg(\langle A!, x^P \rangle), x^P \rangle (\neg(\langle A!, x^P \rangle))$ )
          apply (safe intro!: beta-C-meta-1)
          by show-proper
        hence [( $\langle O!, x^P \rangle \equiv \neg(\langle O!, x^P \rangle)$ ) 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]:

```

$[(\downarrow O!^-, x^P) \equiv \neg(\downarrow A!^-, x^P)] \text{ in } v]$

proof –

have $[(\neg(\downarrow \lambda x. \neg(\downarrow A!, x^P), x^P)) \equiv (\downarrow A!, x^P)] \text{ in } v]$
 apply (*PLM-subst-method* $(\neg(\downarrow A!, x^P)) (\downarrow \lambda x. \neg(\downarrow A!, x^P), x^P)$)
 apply (*safe intro!*: *beta-C-meta-1* [*equiv-sym*])
 apply *show-proper*
 using *oth-class-taut-4-b* [*equiv-sym*] **by** *auto*
moreover have $[(\downarrow \lambda x. \neg(\downarrow O!, x^P), x^P) \equiv \neg(\downarrow O!, x^P)] \text{ in } v]$
 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*]:

$[Contingent (O!^-) \text{ in } v]$

using *oa-contingent-4* *thm-cont-prop-3* [*equiv-lr*] **by** *auto*

lemma *oa-contingent-9* [*PLM*]:

$[Contingent (A!^-) \text{ in } v]$

using *oa-contingent-5* *thm-cont-prop-3* [*equiv-lr*] **by** *auto*

lemma *oa-facts-1* [*PLM*]:

$[(\downarrow O!, x^P) \rightarrow \Box(\downarrow O!, x^P)] \text{ in } v]$

proof (*rule CP*)

assume $[(\downarrow O!, x^P) \text{ in } v]$

hence $[\Diamond(\downarrow E!, x^P) \text{ in } v]$

unfolding *Ordinary-def* **apply** –
 apply (*rule beta-C-meta-1* [*equiv-lr*])
 by *show-proper*

hence $[\Box\Diamond(\downarrow E!, x^P) \text{ in } v]$

using *qml-3* [*axiom-instance*, *deduction*] **by** *auto*

thus $[\Box(\downarrow O!, x^P) \text{ in } v]$

unfolding *Ordinary-def*
 apply –

apply (*PLM-subst-method* $\Diamond(\downarrow E!, x^P) (\downarrow \lambda x. \Diamond(\downarrow E!, x^P), x^P)$)
 apply (*safe intro!*: *beta-C-meta-1* [*equiv-sym*])

by *show-proper*

qed

lemma *oa-facts-2* [*PLM*]:

$[(\downarrow A!, x^P) \rightarrow \Box(\downarrow A!, x^P)] \text{ in } v]$

proof (*rule CP*)

assume $[(\downarrow A!, x^P) \text{ in } v]$

hence $[\neg\Diamond(\downarrow E!, x^P) \text{ in } v]$

unfolding *Abstract-def* **apply** –
 apply (*rule beta-C-meta-1* [*equiv-lr*])
 by *show-proper*

hence $[\Box\Box\neg(\downarrow E!, x^P) \text{ in } v]$

using *KBasic2-4* [*equiv-rl*] $\Box\Box$ [*deduction*] **by** *auto*

hence $[\Box\neg\Diamond(\downarrow E!, x^P) \text{ in } v]$

apply –

apply (*PLM-subst-method* $\Box\neg(\downarrow E!, x^P) \neg\Diamond(\downarrow E!, x^P)$)
 using *KBasic2-4* **by** *auto*

thus $[\Box(\downarrow A!, x^P) \text{ in } v]$

unfolding *Abstract-def*

apply –
apply (*PLM-subst-method* $\neg\Diamond(\langle E!, x^P \rangle) (\lambda x. \neg\Diamond(\langle E!, x^P \rangle, x^P))$)
apply (*safe intro!*: *beta-C-meta-1*[*equiv-sym*])
by *show-proper*
qed

lemma *oa-facts-3*[*PLM*]:
 $[\Diamond(\langle O!, x^P \rangle) \rightarrow (\langle O!, x^P \rangle) \text{ in } v]$
using *oa-facts-1* **by** (*rule derived-S5-rules-2-b*)

lemma *oa-facts-4*[*PLM*]:
 $[\Diamond(\langle A!, x^P \rangle) \rightarrow (\langle A!, x^P \rangle) \text{ in } v]$
using *oa-facts-2* **by** (*rule derived-S5-rules-2-b*)

lemma *oa-facts-5*[*PLM*]:
 $[\Diamond(\langle O!, x^P \rangle) \equiv \Box(\langle O!, x^P \rangle) \text{ in } v]$
using *oa-facts-1*[*deduction*, *OF oa-facts-3*[*deduction*]]
 $T\Diamond[\text{deduction}, \text{OF qml-2}[\text{axiom-instance}, \text{deduction}]]$
 $\equiv I \text{ CP}$ **by** *blast*

lemma *oa-facts-6*[*PLM*]:
 $[\Diamond(\langle A!, x^P \rangle) \equiv \Box(\langle A!, x^P \rangle) \text{ in } v]$
using *oa-facts-2*[*deduction*, *OF oa-facts-4*[*deduction*]]
 $T\Diamond[\text{deduction}, \text{OF qml-2}[\text{axiom-instance}, \text{deduction}]]$
 $\equiv I \text{ CP}$ **by** *blast*

lemma *oa-facts-7*[*PLM*]:
 $[(\langle O!, x^P \rangle) \equiv \mathcal{A}(\langle O!, x^P \rangle) \text{ in } v]$
apply (*rule* $\equiv I$; *rule CP*)
apply (*rule nec-imp-act*[*deduction*, *OF oa-facts-1*[*deduction*]]; *assumption*)
proof –
assume $[\mathcal{A}(\langle O!, x^P \rangle) \text{ in } v]$
hence $[\mathcal{A}(\Diamond(\langle E!, x^P \rangle)) \text{ in } v]$
unfolding *Ordinary-def* **apply** –
apply (*PLM-subst-method* $(\lambda x. \Diamond(\langle E!, x^P \rangle, x^P) \Diamond(\langle E!, x^P \rangle))$)
apply (*safe intro!*: *beta-C-meta-1*)
by *show-proper*
hence $[\Diamond(\langle E!, x^P \rangle) \text{ in } v]$
using *Act-Basic-6*[*equiv-rl*] **by** *auto*
thus $[(\langle O!, x^P \rangle) \text{ in } v]$
unfolding *Ordinary-def* **apply** –
apply (*PLM-subst-method* $\Diamond(\langle E!, x^P \rangle) (\lambda x. \Diamond(\langle E!, x^P \rangle, x^P))$)
apply (*safe intro!*: *beta-C-meta-1*[*equiv-sym*])
by *show-proper*
qed

lemma *oa-facts-8*[*PLM*]:
 $[(\langle A!, x^P \rangle) \equiv \mathcal{A}(\langle A!, x^P \rangle) \text{ in } v]$
apply (*rule* $\equiv I$; *rule CP*)
apply (*rule nec-imp-act*[*deduction*, *OF oa-facts-2*[*deduction*]]; *assumption*)
proof –
assume $[\mathcal{A}(\langle A!, x^P \rangle) \text{ in } v]$
hence $[\mathcal{A}(\neg\Diamond(\langle E!, x^P \rangle)) \text{ in } v]$
unfolding *Abstract-def* **apply** –
apply (*PLM-subst-method* $(\lambda x. \neg\Diamond(\langle E!, x^P \rangle, x^P) \neg\Diamond(\langle E!, x^P \rangle))$)
apply (*safe intro!*: *beta-C-meta-1*)
by *show-proper*
hence $[\mathcal{A}(\Box(\neg\Diamond(\langle E!, x^P \rangle)) \text{ in } v]$

```

  apply -
  apply (PLM-subst-method ( $\neg \Diamond \langle E!, x^P \rangle$ ) ( $\Box \neg \langle E!, x^P \rangle$ ))
  using KBasic2-4[equiv-sym] by auto
hence [ $\neg \Diamond \langle E!, x^P \rangle$  in  $v$ ]
  using qml-act-2[axiom-instance, equiv-rl] KBasic2-4[equiv-lr] by auto
thus [ $\langle A!, x^P \rangle$  in  $v$ ]
  unfolding Abstract-def apply -
  apply (PLM-subst-method  $\neg \Diamond \langle E!, x^P \rangle$  ( $\langle \lambda x. \neg \Diamond \langle E!, x^P \rangle, x^P \rangle$ ))
  apply (safe intro!: beta-C-meta-1[equiv-sym])
  by show-proper
qed

```

lemma cont-nec-fact1-1[PLM]:

```

[WeaklyContingent  $F \equiv$  WeaklyContingent ( $F^-$ ) in  $v$ ]
proof (rule  $\equiv I$ ; rule CP)
  assume [WeaklyContingent  $F$  in  $v$ ]
  hence wc-def: [Contingent  $F \ \& \ (\forall x. (\Diamond \langle F, x^P \rangle \rightarrow \Box \langle F, x^P \rangle))$  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 [ $\Diamond \langle F^-, x^P \rangle$  in  $v$ ]
      hence [ $\neg \Box \langle F, x^P \rangle$  in  $v$ ]
        unfolding diamond-def apply -
        apply (PLM-subst-method  $\neg \langle F^-, x^P \rangle$  ( $\langle F, x^P \rangle$ ))
        using thm-relation-negation-2-1 by auto
      moreover {
        assume [ $\neg \Box \langle F^-, x^P \rangle$  in  $v$ ]
        hence [ $\neg \Box \langle \lambda x. \neg \langle F, x^P \rangle, x^P \rangle$  in  $v$ ]
          unfolding propnot-defs .
        hence [ $\Diamond \langle F, x^P \rangle$  in  $v$ ]
          unfolding diamond-def
          apply - apply (PLM-subst-method ( $\langle \lambda x. \neg \langle F, x^P \rangle, x^P \rangle$ )  $\neg \langle F, x^P \rangle$ )
          apply (safe intro!: beta-C-meta-1)
          by show-proper
        hence [ $\Box \langle F, x^P \rangle$  in  $v$ ]
          using wc-def[conj2] cqt-1[axiom-instance, deduction]
          modus-ponens by fast
      }
      ultimately have [ $\Box \langle F^-, x^P \rangle$  in  $v$ ]
        using  $\neg \neg E$  modus-tollens-1 CP by blast
    }
    hence [ $\forall x. \Diamond \langle F^-, x^P \rangle \rightarrow \Box \langle F^-, x^P \rangle$  in  $v$ ]
      using  $\forall I$  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 ( $F^-$ )  $\& \ (\forall x. (\Diamond \langle F^-, x^P \rangle \rightarrow \Box \langle F^-, x^P \rangle))$  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 [ $\Diamond(\Box(F, x^P))$  in  $v$ ]
hence [ $\neg\Box(\Box(F^-, x^P))$  in  $v$ ]
  unfolding diamond-def apply  $-$ 
  apply (PLM-subst-method  $\neg(\Box(F, x^P))$   $(\Box(F^-, x^P))$ )
  using thm-relation-negation-1-1[equiv-sym] by auto
moreover {
  assume [ $\neg\Box(\Box(F, x^P))$  in  $v$ ]
  hence [ $\Diamond(\Box(F^-, x^P))$  in  $v$ ]
    unfolding diamond-def
    apply  $-$  apply (PLM-subst-method  $(\Box(F, x^P))$   $\neg(\Box(F^-, x^P))$ )
    using thm-relation-negation-2-1[equiv-sym] by auto
  hence [ $\Box(\Box(F^-, x^P))$  in  $v$ ]
    using wc-def[conj2] cqt-1[axiom-instance, deduction]
    modus-ponens by fast
}
ultimately have [ $\Box(\Box(F, x^P))$  in  $v$ ]
  using  $\neg\neg E$  modus-tollens-1 CP by blast
}
hence [ $\forall x . \Diamond(\Box(F, x^P)) \rightarrow \Box(\Box(F, x^P))$  in  $v$ ]
  using  $\forall I$  CP by fast
}
ultimately show [WeaklyContingent ( $F$ ) in  $v$ ]
  unfolding WeaklyContingent-def by (rule &I)
qed

```

lemma *cont-nec-fact1-2*[*PLM*]:
 $[(\text{WeaklyContingent } F \ \& \ \neg(\text{WeaklyContingent } G)) \rightarrow (F \neq G)]$ in v
using *l-identity*[*axiom-instance*, *deduction*, *deduction*] $\&E$ $\&I$
modus-tollens-1 CP **by** *metis*

lemma *cont-nec-fact2-1*[*PLM*]:
 $[\text{WeaklyContingent } (O!)]$ in v
unfolding *WeaklyContingent-def*
apply (*rule &I*)
using *oa-contingent-4* **apply** *simp*
using *oa-facts-5* **unfolding** *equiv-def*
using $\&E(1)$ $\forall I$ **by** *fast*

lemma *cont-nec-fact2-2*[*PLM*]:
 $[\text{WeaklyContingent } (A!)]$ in v
unfolding *WeaklyContingent-def*
apply (*rule &I*)
using *oa-contingent-5* **apply** *simp*
using *oa-facts-6* **unfolding** *equiv-def*
using $\&E(1)$ $\forall I$ **by** *fast*

lemma *cont-nec-fact2-3*[*PLM*]:
 $[\neg(\text{WeaklyContingent } (E!))]$ in v
proof (*rule modus-tollens-1*, *rule CP*)
assume [*WeaklyContingent* $E!$ in v]
thus [$\forall x . \Diamond(\Box(E!, x^P)) \rightarrow \Box(\Box(E!, x^P))$ in v]
unfolding *WeaklyContingent-def* **using** $\&E(2)$ **by** *fast*
next
 {
assume $1: [\forall x . \Diamond(\Box(E!, x^P)) \rightarrow \Box(\Box(E!, x^P))]$ in v
have [$\exists x . \Diamond(\Box(E!, x^P)) \ \& \ \Diamond(\neg(\Box(E!, x^P)))$ in v]
using *qml-4*[*axiom-instance*, *conj1*, *THEN BFs-3*[*deduction*]] .
then obtain x **where** [$\Diamond(\Box(E!, x^P)) \ \& \ \Diamond(\neg(\Box(E!, x^P)))$ in v]
 }

by (rule $\exists E$)
 hence $[\Diamond(E!, x^P) \ \& \ \Diamond(\neg(E!, x^P)) \text{ in } v]$
 using *KBasic2-8[deduction]* *S5Basic-8[deduction]*
 &I &E by blast
 hence $[\Box(E!, x^P) \ \& \ (\neg\Box(E!, x^P)) \text{ in } v]$
 using *1[THEN $\forall E$, deduction]* &E &I
KBasic2-2[equiv-rl] by blast
 hence $[\neg(\forall x . \Diamond(E!, x^P) \rightarrow \Box(E!, x^P)) \text{ in } v]$
 using *oth-class-taut-1-a modus-tollens-1 CP* by blast
 }
 thus $[\neg(\forall x . \Diamond(E!, x^P) \rightarrow \Box(E!, x^P)) \text{ in } v]$
 using *reductio-aa-2 if-p-then-p CP* by meson
 qed

lemma *cont-nec-fact2-4[PLM]*:
 $[\neg(\text{WeaklyContingent } (PLM.L)) \text{ in } v]$
proof –
 {
 assume $[\text{WeaklyContingent } PLM.L \text{ in } v]$
 hence $[\text{Contingent } PLM.L \text{ in } v]$
 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]*:
 $[O! \neq E! \ \& \ O! \neq (E!^-) \ \& \ O! \neq PLM.L \ \& \ O! \neq (PLM.L^-) \text{ in } v]$
proof ((rule &I)+)
 show $[O! \neq E! \text{ in } v]$
 using *cont-nec-fact2-1 cont-nec-fact2-3*
cont-nec-fact1-2[deduction] &I by simp
 next
 have $[\neg(\text{WeaklyContingent } (E!^-)) \text{ in } v]$
 using *cont-nec-fact1-1[THEN oth-class-taut-5-d[equiv-lr], equiv-lr]*
cont-nec-fact2-3 by auto
 thus $[O! \neq (E!^-) \text{ in } v]$
 using *cont-nec-fact2-1 cont-nec-fact1-2[deduction]* &I by simp
 next
 show $[O! \neq PLM.L \text{ in } v]$
 using *cont-nec-fact2-1 cont-nec-fact2-4*
cont-nec-fact1-2[deduction] &I by simp
 next
 have $[\neg(\text{WeaklyContingent } (PLM.L^-)) \text{ in } v]$
 using *cont-nec-fact1-1[THEN oth-class-taut-5-d[equiv-lr], equiv-lr]*
cont-nec-fact2-4 by auto
 thus $[O! \neq (PLM.L^-) \text{ in } v]$
 using *cont-nec-fact2-1 cont-nec-fact1-2[deduction]* &I by simp
 qed

lemma *cont-nec-fact2-6[PLM]*:
 $[A! \neq E! \ \& \ A! \neq (E!^-) \ \& \ A! \neq PLM.L \ \& \ A! \neq (PLM.L^-) \text{ in } v]$
proof ((rule &I)+)
 show $[A! \neq E! \text{ in } v]$
 using *cont-nec-fact2-2 cont-nec-fact2-3*
cont-nec-fact1-2[deduction] &I by simp

```

next
  have  $\neg(\text{WeaklyContingent } (E!^-)) \text{ in } v$ 
    using cont-nec-fact1-1 [THEN oth-class-taut-5-d[equiv-lr], equiv-lr]
      cont-nec-fact2-3 by auto
  thus  $[A! \neq (E!^-) \text{ in } v]$ 
    using cont-nec-fact2-2 cont-nec-fact1-2[deduction] &I by simp
next
  show  $[A! \neq \text{PLM}.L \text{ in } v]$ 
    using cont-nec-fact2-2 cont-nec-fact2-4
      cont-nec-fact1-2[deduction] &I by simp
next
  have  $\neg(\text{WeaklyContingent } (\text{PLM}.L^-)) \text{ in } v$ 
    using cont-nec-fact1-1 [THEN oth-class-taut-5-d[equiv-lr],
      equiv-lr] cont-nec-fact2-4 by auto
  thus  $[A! \neq (\text{PLM}.L^-) \text{ in } v]$ 
    using cont-nec-fact2-2 cont-nec-fact1-2[deduction] &I by simp
qed

lemma id-nec3-1[PLM]:
   $[(x^P) =_E (y^P)) \equiv (\Box((x^P) =_E (y^P))) \text{ in } v]$ 
proof (rule  $\equiv I$ ; rule CP)
  assume  $[(x^P) =_E (y^P) \text{ in } v]$ 
  hence  $[\Box(O!, x^P) \text{ in } v] \wedge [\Box(O!, y^P) \text{ in } v] \wedge [\Box(\forall F. \Box(F, x^P) \equiv \Box(F, y^P)) \text{ in } v]$ 
    using eq-E-simple-1[equiv-lr] using &E by blast
  hence  $[\Box(\Box(O!, x^P) \text{ in } v) \wedge \Box(\Box(O!, y^P) \text{ in } v)$ 
     $\wedge [\Box(\Box(\forall F. \Box(F, x^P) \equiv \Box(F, y^P)) \text{ in } v)]$ 
    using oa-facts-1[deduction] S5Basic-6[deduction] by blast
  hence  $[\Box(\Box(O!, x^P) \wedge \Box(O!, y^P) \wedge \Box(\forall F. \Box(F, x^P) \equiv \Box(F, y^P))) \text{ in } v]$ 
    using &I KBasic-3[equiv-rl] by presburger
  thus  $[\Box((x^P) =_E (y^P)) \text{ in } v]$ 
    apply –
    apply (PLM-subst-method
       $(\Box(O!, x^P) \wedge \Box(O!, y^P) \wedge \Box(\forall F. \Box(F, x^P) \equiv \Box(F, y^P)))$ 
       $(x^P) =_E (y^P))$ 
    using eq-E-simple-1[equiv-sym] by auto
next
  assume  $[\Box((x^P) =_E (y^P)) \text{ in } v]$ 
  thus  $[(x^P) =_E (y^P) \text{ in } v]$ 
    using qml-2[axiom-instance, deduction] by simp
qed

lemma id-nec3-2[PLM]:
   $[\Diamond((x^P) =_E (y^P)) \equiv ((x^P) =_E (y^P)) \text{ in } v]$ 
proof (rule  $\equiv I$ ; rule CP)
  assume  $[\Diamond((x^P) =_E (y^P)) \text{ in } v]$ 
  thus  $[(x^P) =_E (y^P) \text{ in } v]$ 
    using derived-S5-rules-2-b[deduction] id-nec3-1[equiv-lr]
      CP modus-ponens by blast
next
  assume  $[(x^P) =_E (y^P) \text{ in } v]$ 
  thus  $[\Diamond((x^P) =_E (y^P)) \text{ in } v]$ 
    by (rule TBasic[deduction])
qed

lemma thm-neg-eqE[PLM]:
   $[(x^P) \neq_E (y^P)) \equiv (\neg((x^P) =_E (y^P))) \text{ in } v]$ 
proof –
  have  $[(x^P) \neq_E (y^P) \text{ in } v] = [\Box(\lambda^2 (\lambda x y. (x^P) =_E (y^P)))^-, x^P, y^P] \text{ in } v]$ 

```

unfolding *not-identical_E-def* **by** *simp*
also have ... = $\lceil \neg(\lambda^2 (\lambda x y . (x^P) =_E (y^P))), x^P, y^P \rceil$ *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 ... = $\lceil \neg((x^P) =_E (y^P)) \rceil$ *in v*
apply (*PLM-subst-method*
 $\lceil \lambda^2 (\lambda x y . (x^P) =_E (y^P))), x^P, y^P \rceil$
 $(x^P) =_E (y^P)$
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*]:
 $\lceil ((x^P) \neq_E (y^P)) \equiv \Box((x^P) \neq_E (y^P)) \rceil$ *in v*
proof –
have $\lceil \neg((x^P) =_E (y^P)) \equiv \Box(\neg((x^P) =_E (y^P))) \rceil$ *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* $\lceil \neg((x^P) =_E (y^P)) \rceil$) $(x^P) \neq_E (y^P)$
using *thm-neg-eqE*[*equiv-sym*] **by** *auto*
qed

lemma *id-nec4-2*[*PLM*]:
 $\lceil \Diamond((x^P) \neq_E (y^P)) \equiv ((x^P) \neq_E (y^P)) \rceil$ *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*]:
 $\lceil ((x^P) =_E (y^P)) \equiv (\mathcal{A}((x^P) =_E (y^P))) \rceil$ *in v*
proof (*rule* $\equiv I$; *rule* *CP*)
assume $\lceil (x^P) =_E (y^P) \rceil$ *in v*
hence $\lceil \Box((x^P) =_E (y^P)) \rceil$ *in v*
using *id-nec3-1*[*equiv-lr*] **by** *auto*
thus $\lceil \mathcal{A}((x^P) =_E (y^P)) \rceil$ *in v*
using *nec-imp-act*[*deduction*] **by** *fast*
next
assume $\lceil \mathcal{A}((x^P) =_E (y^P)) \rceil$ *in v*
hence $\lceil \mathcal{A}(\lceil O!, x^P \rceil \ \& \ \lceil O!, y^P \rceil \ \& \ \Box(\forall F . \lceil F, x^P \rceil \equiv \lceil F, y^P \rceil)) \rceil$ *in v*
apply –
apply (*PLM-subst-method*
 $(x^P) =_E (y^P)$
 $(\lceil O!, x^P \rceil \ \& \ \lceil O!, y^P \rceil \ \& \ \Box(\forall F . \lceil F, x^P \rceil \equiv \lceil F, y^P \rceil))$
using *eq-E-simple-1* **by** *auto*
hence $\lceil \mathcal{A}(\lceil O!, x^P \rceil \ \& \ \mathcal{A}(\lceil O!, y^P \rceil \ \& \ \mathcal{A}(\Box(\forall F . \lceil F, x^P \rceil \equiv \lceil F, y^P \rceil))) \rceil$ *in v*
using *Act-Basic-2*[*equiv-lr*] $\& I$ $\& E$ **by** *meson*
thus $\lceil (x^P) =_E (y^P) \rceil$ *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*]:
 $\lceil ((x^P) \neq_E (y^P)) \equiv (\mathcal{A}((x^P) \neq_E (y^P))) \rceil$ *in v*
apply (*PLM-subst-method* $\lceil \neg((x^P) =_E (y^P)) \rceil$) $((x^P) \neq_E (y^P))$

```

    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:  $[\mathcal{A}(\alpha = \beta) \text{ in } v] \implies [(\alpha = \beta) \text{ in } v]$ 

instantiation  $\nu :: id-act$ 
begin
  instance proof
    interpret PLM .
    fix  $x::\nu$  and  $y::\nu$  and  $v::i$ 
    assume  $[\mathcal{A}(x = y) \text{ in } v]$ 
    hence  $[\mathcal{A}(((x^P) =_E (y^P))) \vee ((\downarrow A!, x^P) \& (\downarrow A!, y^P))$ 
       $\& \Box(\forall F . \llbracket x^P, F \rrbracket \equiv \llbracket y^P, F \rrbracket)) \text{ in } v]$ 
      unfolding identity-defs by auto
    hence  $[\mathcal{A}(((x^P) =_E (y^P))) \vee \mathcal{A}((\downarrow A!, x^P) \& (\downarrow A!, y^P))$ 
       $\& \Box(\forall F . \llbracket x^P, F \rrbracket \equiv \llbracket y^P, F \rrbracket)) \text{ in } v]$ 
      using Act-Basic-10[equiv-lr] by auto
    moreover {
      assume  $[\mathcal{A}(((x^P) =_E (y^P))) \text{ in } v]$ 
      hence  $[(x^P) = (y^P) \text{ in } v]$ 
      using id-act-1[equiv-rl] eq-E-simple-2[deduction] by auto
    }
    moreover {
      assume  $[\mathcal{A}((\downarrow A!, x^P) \& (\downarrow A!, y^P) \& \Box(\forall F . \llbracket x^P, F \rrbracket \equiv \llbracket y^P, F \rrbracket)) \text{ in } v]$ 
      hence  $[\mathcal{A}(\downarrow A!, x^P) \& \mathcal{A}(\downarrow A!, y^P) \& \mathcal{A}(\Box(\forall F . \llbracket x^P, F \rrbracket \equiv \llbracket y^P, F \rrbracket)) \text{ in } v]$ 
      using Act-Basic-2[equiv-lr] &I &E by meson
      hence  $[(\downarrow A!, x^P) \& (\downarrow A!, y^P) \& (\Box(\forall F . \llbracket x^P, F \rrbracket \equiv \llbracket y^P, F \rrbracket)) \text{ in } v]$ 
      using oa-facts-8[equiv-rl] qml-act-2[axiom-instance,equiv-rl]
      &I &E by meson
      hence  $[(x^P) = (y^P) \text{ in } v]$ 
      unfolding identity-defs using  $\vee I$  by auto
    }
    ultimately have  $[(x^P) = (y^P) \text{ in } v]$ 
      using intro-elim-4-a CP by meson
    thus  $[x = y \text{ in } v]$ 
      unfolding identity-defs by auto
  qed
end

instantiation  $\Pi_1 :: id-act$ 
begin
  instance proof
    interpret PLM .
    fix  $F::\Pi_1$  and  $G::\Pi_1$  and  $v::i$ 
    show  $[\mathcal{A}(F = G) \text{ in } v] \implies [(F = G) \text{ in } v]$ 
      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 [A(p = q) in v] ==> [p = q in v]
  unfolding identityo-def using id-act-prop by blast
qed
end

instantiation Π2 :: id-act
begin
instance proof
interpret PLM .
fix F::Π2 and G::Π2 and v::i
assume a: [A(F = G) in v]
{
  fix x
  have [A((λy. (↓F,xP,yP)) = (λy. (↓G,xP,yP))
    & (λy. (↓F,yP,xP)) = (λy. (↓G,yP,xP)) in v]
    using a logic-actual-nec-3[axiom-instance, equiv-lr] cqt-basic-4[equiv-lr] ∀ E
    unfolding identity2-def by fast
  hence [(λy. (↓F,xP,yP)) = (λy. (↓G,xP,yP))
    & ((λy. (↓F,yP,xP)) = (λy. (↓G,yP,xP)) 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 ∀ I)
qed
end

```

```

instantiation Π3 :: id-act
begin
instance proof
interpret PLM .
fix F::Π3 and G::Π3 and v::i
assume a: [A(F = G) in v]
let ?p = λ x y . (λz. (↓F,zP,xP,yP)) = (λz. (↓G,zP,xP,yP))
  & (λz. (↓F,xP,zP,yP)) = (λz. (↓G,xP,zP,yP))
  & (λz. (↓F,xP,yP,zP)) = (λz. (↓G,xP,yP,zP))

{
  fix x
  {
    fix y
    have [A(?p x y) in v]
      using a logic-actual-nec-3[axiom-instance, equiv-lr]
      cqt-basic-4[equiv-lr] ∀ E[where 'a=ν]
      unfolding identity3-def by blast
    hence [?p x y in v]
      using &I &E id-act-prop Act-Basic-2[equiv-lr] by metis
  }
  hence [∀ y . ?p x y in v]
    by (rule ∀ I)
}
thus [F = G in v]
  unfolding identity3-def by (rule ∀ I)
qed
end

```

```

context PLM
begin
lemma id-act-3[PLM]:
  [((α::('a::id-act)) = β) ≡ A(α = β) 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*]:
 $[(\alpha::('a::id-act)) \neq \beta) \equiv \mathcal{A}(\alpha \neq \beta) \text{ in } v]$
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*]:
 $[(y^P) = (\iota x . x = y) \text{ in } v]$
using *descriptions*[*axiom-instance*, *equiv-rl*]
id-act-3[*equiv-sym*] $\forall I$ **by** *fast*

lemma *eta-conversion-lemma-1*[*PLM*]:
 $[(\lambda x . \langle F, x^P \rangle) = F \text{ in } v]$
using *lambda-predicates-3-1*[*axiom-instance*] .

lemma *eta-conversion-lemma-0*[*PLM*]:
 $[(\lambda^0 p) = p \text{ in } v]$
using *lambda-predicates-3-0*[*axiom-instance*] .

lemma *eta-conversion-lemma-2*[*PLM*]:
 $[(\lambda^2 (\lambda x y . \langle F, x^P, y^P \rangle)) = F \text{ in } v]$
using *lambda-predicates-3-2*[*axiom-instance*] .

lemma *eta-conversion-lemma-3*[*PLM*]:
 $[(\lambda^3 (\lambda x y z . \langle F, x^P, y^P, z^P \rangle)) = F \text{ in } v]$
using *lambda-predicates-3-3*[*axiom-instance*] .

lemma *lambda-p-q-p-eq-q*[*PLM*]:
 $[(\lambda^0 p) = (\lambda^0 q) \equiv (p = q) \text{ in } v]$
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*]:
 $[\forall x . \langle O!, x^P \rangle \vee \langle A!, x^P \rangle \text{ in } v]$
proof (*rule* $\forall I$)
fix *x*
have $[\Diamond \langle E!, x^P \rangle \vee \neg \Diamond \langle E!, x^P \rangle \text{ in } v]$
by *PLM-solver*
moreover have $[\Diamond \langle E!, x^P \rangle \equiv \langle \lambda y . \Diamond \langle E!, y^P \rangle, x^P \rangle \text{ in } v]$
apply (*rule* *beta-C-meta-1*[*equiv-sym*])
by *show-proper*
moreover have $[(\neg \Diamond \langle E!, x^P \rangle) \equiv \langle \lambda y . \neg \Diamond \langle E!, y^P \rangle, x^P \rangle \text{ in } v]$
apply (*rule* *beta-C-meta-1*[*equiv-sym*])
by *show-proper*
ultimately show $[\langle O!, x^P \rangle \vee \langle A!, x^P \rangle \text{ in } v]$
unfolding *Ordinary-def Abstract-def* **by** *PLM-solver*
qed

lemma *partition-2*[*PLM*]:
 $[\neg (\exists x . \langle O!, x^P \rangle \ \& \ \langle A!, x^P \rangle) \text{ in } v]$
proof —

```

{
  assume  $[\exists x . \langle O!, x^P \rangle \ \& \ \langle A!, x^P \rangle \text{ in } v]$ 
  then obtain  $b$  where  $[\langle O!, b^P \rangle \ \& \ \langle A!, b^P \rangle \text{ in } v]$ 
    by (rule  $\exists E$ )
  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*]:
 $[\langle O!, x \rangle \rightarrow (x =_E x) \text{ in } v]$
proof (rule *CP*)
 assume $[\langle O!, x \rangle \text{ in } v]$
 moreover have $[\Box(\forall F . \langle F, x \rangle \equiv \langle F, x \rangle) \text{ in } v]$
 by *PLM-solver*
 ultimately show $[(x) =_E (x) \text{ in } v]$
 using $\&I$ *eq-E-simple-1*[*equiv-rl*] by *blast*
 qed

lemma *ord-eq-Eequiv-2*[*PLM*]:
 $[(x =_E y) \rightarrow (y =_E x) \text{ in } v]$
proof (rule *CP*)
 assume $[x =_E y \text{ in } v]$
 hence 1: $[\langle O!, x \rangle \ \& \ \langle O!, y \rangle \ \& \ \Box(\forall F . \langle F, x \rangle \equiv \langle F, y \rangle) \text{ in } v]$
 using *eq-E-simple-1*[*equiv-lr*] by *simp*
 have $[\Box(\forall F . \langle F, y \rangle \equiv \langle F, x \rangle) \text{ in } v]$
 apply (*PLM-subst-method*
 $\lambda F . \langle F, x \rangle \equiv \langle F, y \rangle$
 $\lambda F . \langle F, y \rangle \equiv \langle F, x \rangle$)
 using *oth-class-taut-3-g 1*[*conj2*] by *auto*
 thus $[y =_E x \text{ in } v]$
 using *eq-E-simple-1*[*equiv-rl*] 1[*conj1*]
 $\&E$ $\&I$ by *meson*
 qed

lemma *ord-eq-Eequiv-3*[*PLM*]:
 $[(x =_E y) \ \& \ (y =_E z) \rightarrow (x =_E z) \text{ in } v]$
proof (rule *CP*)
 assume $a: [(x =_E y) \ \& \ (y =_E z) \text{ in } v]$
 have $[\Box((\forall F . \langle F, x \rangle \equiv \langle F, y \rangle) \ \& \ (\forall F . \langle F, y \rangle \equiv \langle F, z \rangle)) \text{ in } v]$
 using *KBasic-3*[*equiv-rl*] $a[conj1, \text{ THEN } eq-E-simple-1[equiv-lr, conj2]]$
 $a[conj2, \text{ THEN } eq-E-simple-1[equiv-lr, conj2]] \ \&I$ by *blast*
 moreover {
 {
 fix w
 have $[(\forall F . \langle F, x \rangle \equiv \langle F, y \rangle) \ \& \ (\forall F . \langle F, y \rangle \equiv \langle F, z \rangle)]$
 $\rightarrow (\forall F . \langle F, x \rangle \equiv \langle F, z \rangle) \text{ in } w$
 by *PLM-solver*
 }
 hence $[\Box((\forall F . \langle F, x \rangle \equiv \langle F, y \rangle) \ \& \ (\forall F . \langle F, y \rangle \equiv \langle F, z \rangle))$
 $\rightarrow (\forall F . \langle F, x \rangle \equiv \langle F, z \rangle) \text{ in } v]$
 by (rule *RN*)
 }
 ultimately have $[\Box(\forall F . \langle F, x \rangle \equiv \langle F, z \rangle) \text{ in } v]$
 using *qml-1*[*axiom-instance, deduction, deduction*] by *blast*

thus $[x =_E z \text{ in } v]$
using $a[\text{conj1}, \text{ THEN eq-E-simple-1}[\text{equiv-lr}, \text{conj1}, \text{conj1}]]$
using $a[\text{conj2}, \text{ THEN eq-E-simple-1}[\text{equiv-lr}, \text{conj1}, \text{conj2}]]$
 $\text{eq-E-simple-1}[\text{equiv-rl}] \ \&I$
by *presburger*
qed

lemma *ord-eq-E-eq[PLM]*:
 $[(\langle O!, x^P \rangle \vee \langle O!, y^P \rangle) \rightarrow ((x^P = y^P) \equiv (x^P =_E y^P)) \text{ in } v]$
proof (*rule CP*)
assume $[\langle O!, x^P \rangle \vee \langle O!, y^P \rangle \text{ in } v]$
moreover {
assume $[\langle O!, x^P \rangle \text{ in } v]$
hence $[(x^P = y^P) \equiv (x^P =_E y^P) \text{ in } v]$
using $\equiv I \text{ CP } l\text{-identity}[\text{axiom-instance}, \text{deduction}, \text{deduction}]$
 $\text{ord-eq-Eequiv-1}[\text{deduction}] \text{ eq-E-simple-2}[\text{deduction}] \text{ by } \text{metis}$
}
moreover {
assume $[\langle O!, y^P \rangle \text{ in } v]$
hence $[(x^P = y^P) \equiv (x^P =_E y^P) \text{ in } v]$
using $\equiv I \text{ CP } l\text{-identity}[\text{axiom-instance}, \text{deduction}, \text{deduction}]$
 $\text{ord-eq-Eequiv-1}[\text{deduction}] \text{ eq-E-simple-2}[\text{deduction}] \text{ id-eq-2}[\text{deduction}]$
 $\text{ord-eq-Eequiv-2}[\text{deduction}] \text{ identity-}\nu\text{-def} \text{ by } \text{metis}$
}
ultimately show $[(x^P = y^P) \equiv (x^P =_E y^P) \text{ in } v]$
using *intro-elim-4-a CP* **by** *blast*
qed

lemma *ord-eq-E[PLM]*:
 $[(\langle O!, x^P \rangle \ \& \ \langle O!, y^P \rangle) \rightarrow ((\forall F . \langle F, x^P \rangle \equiv \langle F, y^P \rangle) \rightarrow x^P =_E y^P) \text{ in } v]$
proof (*rule CP; rule CP*)
assume *ord-xy*: $[\langle O!, x^P \rangle \ \& \ \langle O!, y^P \rangle \text{ in } v]$
assume $[\forall F . \langle F, x^P \rangle \equiv \langle F, y^P \rangle \text{ in } v]$
hence $[\langle \lambda z . z^P =_E x^P, x^P \rangle \equiv \langle \lambda z . z^P =_E x^P, y^P \rangle \text{ in } v]$
by (*rule $\forall E$*)
moreover have $[\langle \lambda z . z^P =_E x^P, x^P \rangle \text{ in } v]$
apply (*rule beta-C-meta-1[equiv-rl]*)
unfolding *identity_E-infix-def*
apply *show-proper*
using *ord-eq-Eequiv-1*[\text{deduction}] *ord-xy*[\text{conj1}]
unfolding *identity_E-infix-def* **by** *simp*
ultimately have $[\langle \lambda z . z^P =_E x^P, y^P \rangle \text{ in } v]$
using $\equiv E$ **by** *blast*
hence $[y^P =_E x^P \text{ in } v]$
unfolding *identity_E-infix-def*
apply (*safe intro!*:
 $\text{beta-C-meta-1}[\text{where } \varphi = \lambda z . \langle \text{basic-identity}_{E,z,x^P} \rangle, \text{equiv-lr}])$
by *show-proper*
thus $[x^P =_E y^P \text{ in } v]$
by (*rule ord-eq-Eequiv-2*[\text{deduction}])
qed

lemma *ord-eq-E2[PLM]*:
 $[(\langle O!, x^P \rangle \ \& \ \langle O!, y^P \rangle) \rightarrow ((x^P \neq y^P) \equiv (\lambda z . z^P =_E x^P) \neq (\lambda z . z^P =_E y^P)) \text{ in } v]$
proof (*rule CP; rule $\equiv I$; rule CP*)
assume *ord-xy*: $[\langle O!, x^P \rangle \ \& \ \langle O!, y^P \rangle \text{ in } v]$
assume $[x^P \neq y^P \text{ in } v]$

hence $[\neg(x^P =_E y^P) \text{ in } v]$
 using *eq-E-simple-2 modus-tollens-1* by *fast*
 moreover {
 assume $[(\lambda z . z^P =_E x^P) = (\lambda z . z^P =_E y^P) \text{ in } v]$
 moreover have $[(\lambda z . z^P =_E x^P, x^P) \text{ in } v]$
 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 $[(\lambda z . z^P =_E y^P, x^P) \text{ in } v]$
 using *l-identity[axiom-instance, deduction, deduction]* by *fast*
 hence $[x^P =_E y^P \text{ in } v]$
 unfolding *identity_E-infix-def*
 apply (*safe intro!*:
 beta-C-meta-1[where $\varphi = \lambda z . (\lambda \text{basic-identity}_{E,z,y^P})$, equiv-lr])
 by *show-proper*
 }
 ultimately show $[(\lambda z . z^P =_E x^P) \neq (\lambda z . z^P =_E y^P) \text{ in } v]$
 using *modus-tollens-1 CP* by *blast*
 next
 assume *ord-xy*: $[(\lambda O!, x^P) \ \& \ (\lambda O!, y^P) \text{ in } v]$
 assume $[(\lambda z . z^P =_E x^P) \neq (\lambda z . z^P =_E y^P) \text{ in } v]$
 moreover {
 assume $[x^P = y^P \text{ in } v]$
 hence $[(\lambda z . z^P =_E x^P) = (\lambda z . z^P =_E y^P) \text{ in } v]$
 using *id-eq-1 l-identity[axiom-instance, deduction, deduction]*
 by *fast*
 }
 ultimately show $[x^P \neq y^P \text{ in } v]$
 using *modus-tollens-1 CP* by *blast*
 qed

lemma *ab-obey-1[PLM]*:
 $[(\lambda A!, x^P) \ \& \ (\lambda A!, y^P)] \rightarrow ((\forall F . \{x^P, F\} \equiv \{y^P, F\}) \rightarrow x^P = y^P) \text{ in } v]$
 proof(*rule CP; rule CP*)
 assume *abs-xy*: $[(\lambda A!, x^P) \ \& \ (\lambda A!, y^P) \text{ in } v]$
 assume *enc-equiv*: $[\forall F . \{x^P, F\} \equiv \{y^P, F\} \text{ in } v]$
 {
 fix *P*
 have $[\{x^P, P\} \equiv \{y^P, P\} \text{ in } v]$
 using *enc-equiv* by (*rule $\forall E$*)
 hence $[\Box(\{x^P, P\} \equiv \{y^P, P\}) \text{ in } v]$
 using *en-eq-2 intro-elim-6-e intro-elim-6-f*
 en-eq-5[equiv-rl] by *meson*
 }
 hence $[\Box(\forall F . \{x^P, F\} \equiv \{y^P, F\}) \text{ in } v]$
 using *BF[deduction] $\forall I$* by *fast*
 thus $[x^P = y^P \text{ in } v]$
 unfolding *identity-defs*
 using $\forall I(2)$ *abs-xy* & *I* by *presburger*
 qed

lemma *ab-obey-2[PLM]*:
 $[(\lambda A!, x^P) \ \& \ (\lambda A!, y^P)] \rightarrow ((\exists F . \{x^P, F\} \ \& \ \neg\{y^P, F\}) \rightarrow x^P \neq y^P) \text{ in } v]$
 proof(*rule CP; rule CP*)
 assume *abs-xy*: $[(\lambda A!, x^P) \ \& \ (\lambda A!, y^P) \text{ in } v]$
 assume $[\exists F . \{x^P, F\} \ \& \ \neg\{y^P, F\} \text{ in } v]$

then obtain P where P -prop:
 $\llbracket x^P, P \rrbracket \ \& \ \neg \llbracket y^P, P \rrbracket \text{ in } v$
by (rule $\exists E$)
{
 assume $[x^P = y^P \text{ in } v]$
 hence $\llbracket x^P, P \rrbracket \equiv \llbracket y^P, P \rrbracket \text{ in } v$
 using *l-identity*[*axiom-instance*, *deduction*, *deduction*]
 oth-class-taut-4-a **by** *fast*
 hence $\llbracket y^P, P \rrbracket \text{ in } v$
 using P -prop[*conj1*] **by** (rule $\equiv E$)
}
thus $[x^P \neq y^P \text{ in } v]$
 using P -prop[*conj2*] *modus-tollens-1 CP* **by** *blast*
qed

lemma *ordnecfail*[PLM]:
 $\llbracket O!, x^P \rrbracket \rightarrow \Box(\neg(\exists F . \llbracket x^P, F \rrbracket)) \text{ in } v$
proof (rule *CP*)
 assume $\llbracket O!, x^P \rrbracket \text{ in } v$
 hence $\Box \llbracket O!, x^P \rrbracket \text{ in } v$
 using *oa-facts-1*[*deduction*] **by** *simp*
 moreover hence $\Box(\llbracket O!, x^P \rrbracket \rightarrow (\neg(\exists F . \llbracket x^P, F \rrbracket))) \text{ in } v$
 using *nocoder*[*axiom-necessitation*, *axiom-instance*] **by** *simp*
 ultimately show $\Box(\neg(\exists F . \llbracket x^P, F \rrbracket)) \text{ in } v$
 using *qml-1*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
qed

lemma *o-objects-exist-1*[PLM]:
 $\Diamond(\exists x . \llbracket E!, x^P \rrbracket) \text{ in } v$
proof –
 have $\Diamond(\exists x . \llbracket E!, x^P \rrbracket \ \& \ \Diamond(\neg \llbracket E!, x^P \rrbracket)) \text{ in } v$
 using *qml-4*[*axiom-instance*, *conj1*] .
 hence $\Diamond((\exists x . \llbracket E!, x^P \rrbracket) \ \& \ (\exists x . \Diamond(\neg \llbracket E!, x^P \rrbracket))) \text{ in } v$
 using *sign-S5-thm-3*[*deduction*] **by** *fast*
 hence $\Diamond(\exists x . \llbracket E!, x^P \rrbracket) \ \& \ \Diamond(\exists x . \Diamond(\neg \llbracket E!, x^P \rrbracket)) \text{ in } v$
 using *KBasic2-8*[*deduction*] **by** *blast*
 thus *?thesis* **using** $\&E$ **by** *blast*
qed

lemma *o-objects-exist-2*[PLM]:
 $\Box(\exists x . \llbracket O!, x^P \rrbracket) \text{ in } v$
apply (rule *RN*) **unfolding** *Ordinary-def*
apply (*PLM-subst-method* $\lambda x . \Diamond \llbracket E!, x^P \rrbracket \lambda x . \llbracket \lambda y . \Diamond \llbracket E!, y^P \rrbracket, x^P \rrbracket$)
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]:
 $\Box(\neg(\forall x . \llbracket A!, x^P \rrbracket)) \text{ in } v$
apply (*PLM-subst-method* $(\exists x . \neg \llbracket A!, x^P \rrbracket) \neg(\forall x . \llbracket A!, x^P \rrbracket)$)
 using *cqt-further-2*[*equiv-sym*] **apply** *fast*
apply (*PLM-subst-method* $\lambda x . \llbracket O!, x^P \rrbracket \lambda x . \neg \llbracket A!, x^P \rrbracket$)
using *oa-contingent-2 o-objects-exist-2* **by** *auto*

lemma *a-objects-exist-1*[PLM]:
 $\Box(\exists x . \llbracket A!, x^P \rrbracket) \text{ in } v$
proof –
{

```

fix v
have  $[\exists x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv (F = F)) \text{ in } v]$ 
  using A-objects[axiom-instance] by simp
hence  $[\exists x . \langle A!, x^P \rangle \text{ in } v]$ 
  using cqt-further-5[deduction, conj1] by fast
}
thus ?thesis by (rule RN)
qed

```

```

lemma a-objects-exist-2[PLM]:
 $[\Box(\neg(\forall x . \langle O!, x^P \rangle)) \text{ in } v]$ 
apply (PLM-subst-method  $(\exists x . \neg \langle O!, x^P \rangle) \neg(\forall x . \langle O!, x^P \rangle)$ )
  using cqt-further-2[equiv-sym] apply fast
apply (PLM-subst-method  $\lambda x . \langle A!, x^P \rangle \lambda x . \neg \langle O!, x^P \rangle$ )
  using oa-contingent-3 a-objects-exist-1 by auto

```

```

lemma a-objects-exist-3[PLM]:
 $[\Box(\neg(\forall x . \langle E!, x^P \rangle)) \text{ in } v]$ 
proof -
{
  fix v
  have  $[\exists x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv (F = F)) \text{ in } v]$ 
    using A-objects[axiom-instance] by simp
  hence  $[\exists x . \langle A!, x^P \rangle \text{ in } v]$ 
    using cqt-further-5[deduction, conj1] by fast
  then obtain a where
     $[\langle A!, a^P \rangle \text{ in } v]$ 
    by (rule  $\exists E$ )
  hence  $[\neg(\langle E!, a^P \rangle) \text{ in } v]$ 
    unfolding Abstract-def
    apply (safe intro!: beta-C-meta-1[equiv-lr])
    by show-proper
  hence  $[\neg \langle E!, a^P \rangle \text{ in } v]$ 
    using KBasic2-4[equiv-rl] qml-2[axiom-instance, deduction]
    by simp
  hence  $[\neg(\forall x . \langle E!, x^P \rangle) \text{ in } v]$ 
    using  $\exists I$  cqt-further-2[equiv-rl]
    by fast
}
thus ?thesis
  by (rule RN)
qed

```

```

lemma encoders-are-abstract[PLM]:
 $[(\exists F . \langle x^P, F \rangle) \rightarrow \langle A!, x^P \rangle \text{ in } v]$ 
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]:
 $[\exists! x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv \varphi F) \text{ in } v]$ 
proof -
  have  $[\exists x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv \varphi F) \text{ in } v]$ 
    using A-objects[axiom-instance] by simp
  then obtain a where a-prop:
     $[\langle A!, a^P \rangle \ \& \ (\forall F . \langle a^P, F \rangle \equiv \varphi F) \text{ in } v]$  by (rule  $\exists E$ )
  moreover have  $[\forall y . \langle A!, y^P \rangle \ \& \ (\forall F . \langle y^P, F \rangle \equiv \varphi F) \rightarrow (y = a) \text{ in } v]$ 

```

```

proof (rule  $\forall I$ ; rule CP)
  fix  $b$ 
  assume  $b\text{-prop}$ :  $[\langle A!, b^P \rangle \ \& \ (\forall F . \langle b^P, F \rangle \equiv \varphi F)]$  in  $v$ 
  {
    fix  $P$ 
    have  $[\langle b^P, P \rangle \equiv \langle a^P, P \rangle]$  in  $v$ 
    using  $a\text{-prop}[conj2]$   $b\text{-prop}[conj2] \equiv I \equiv E(1) \equiv E(2)$ 
       $CP \text{ vdash-properties-10 } \forall E$  by metis
  }
  hence  $[\forall F . \langle b^P, F \rangle \equiv \langle a^P, F \rangle]$  in  $v$ 
  using  $\forall I$  by fast
  thus  $[b = a]$  in  $v$ 
  unfolding identity- $\nu$ -def
  using  $ab\text{-obey-1}[deduction, deduction]$ 
     $a\text{-prop}[conj1]$   $b\text{-prop}[conj1]$   $\&I$  by blast
  qed
ultimately show ?thesis
  unfolding exists-unique-def
  using  $\&I \exists I$  by fast
qed

lemma obj-oth-1[PLM]:
   $[\exists! x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv \langle F, y^P \rangle)]$  in  $v$ 
  using A-objects-unique .

lemma obj-oth-2[PLM]:
   $[\exists! x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv (\langle F, y^P \rangle \ \& \ \langle F, z^P \rangle))]$  in  $v$ 
  using A-objects-unique .

lemma obj-oth-3[PLM]:
   $[\exists! x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv (\langle F, y^P \rangle \vee \langle F, z^P \rangle))]$  in  $v$ 
  using A-objects-unique .

lemma obj-oth-4[PLM]:
   $[\exists! x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv (\Box \langle F, y^P \rangle))]$  in  $v$ 
  using A-objects-unique .

lemma obj-oth-5[PLM]:
   $[\exists! x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv (F = G))]$  in  $v$ 
  using A-objects-unique .

lemma obj-oth-6[PLM]:
   $[\exists! x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv \Box(\forall y . \langle G, y^P \rangle \rightarrow \langle F, y^P \rangle))]$  in  $v$ 
  using A-objects-unique .

lemma A-Exists-1[PLM]:
   $[\mathcal{A}(\exists! x :: ('a :: id\text{-}act) . \varphi x) \equiv (\exists! x . \mathcal{A}(\varphi x))]$  in  $v$ 
  unfolding exists-unique-def
  proof (rule  $\equiv I$ ; rule CP)
    assume  $[\mathcal{A}(\exists \alpha . \varphi \alpha \ \& \ (\forall \beta . \varphi \beta \rightarrow \beta = \alpha))]$  in  $v$ 
    hence  $[\exists \alpha . \mathcal{A}(\varphi \alpha \ \& \ (\forall \beta . \varphi \beta \rightarrow \beta = \alpha))]$  in  $v$ 
    using Act-Basic-11[equiv-lr] by blast
    then obtain  $\alpha$  where
       $[\mathcal{A}(\varphi \alpha \ \& \ (\forall \beta . \varphi \beta \rightarrow \beta = \alpha))]$  in  $v$ 
    by (rule  $\exists E$ )
    hence  $1$ :  $[\mathcal{A}(\varphi \alpha) \ \& \ \mathcal{A}(\forall \beta . \varphi \beta \rightarrow \beta = \alpha)]$  in  $v$ 
    using Act-Basic-2[equiv-lr] by blast
    find-theorems  $\mathcal{A}(?p = ?q)$ 

```

```

have 2: [ $\forall \beta. \mathcal{A}(\varphi \beta \rightarrow \beta = \alpha)$  in  $v$ ]
  using 1[conj2] logic-actual-nec-3[axiom-instance, equiv-lr] by blast
{
  fix  $\beta$ 
  have [ $\mathcal{A}(\varphi \beta \rightarrow \beta = \alpha)$  in  $v$ ]
    using 2 by (rule  $\forall E$ )
  hence [ $\mathcal{A}(\varphi \beta) \rightarrow (\beta = \alpha)$  in  $v$ ]
    using logic-actual-nec-2[axiom-instance, equiv-lr, deduction]
      id-act-3[equiv-rl] CP by blast
}
hence [ $\forall \beta. \mathcal{A}(\varphi \beta) \rightarrow (\beta = \alpha)$  in  $v$ ]
  by (rule  $\forall I$ )
thus [ $\exists \alpha. \mathcal{A}\varphi \alpha \ \& \ (\forall \beta. \mathcal{A}\varphi \beta \rightarrow \beta = \alpha)$  in  $v$ ]
  using 1[conj1] &I  $\exists I$  by fast
next
assume [ $\exists \alpha. \mathcal{A}\varphi \alpha \ \& \ (\forall \beta. \mathcal{A}\varphi \beta \rightarrow \beta = \alpha)$  in  $v$ ]
then obtain  $\alpha$  where 1:
  [ $\mathcal{A}\varphi \alpha \ \& \ (\forall \beta. \mathcal{A}\varphi \beta \rightarrow \beta = \alpha)$  in  $v$ ]
  by (rule  $\exists E$ )
{
  fix  $\beta$ 
  have [ $\mathcal{A}(\varphi \beta) \rightarrow \beta = \alpha$  in  $v$ ]
    using 1[conj2] by (rule  $\forall E$ )
  hence [ $\mathcal{A}(\varphi \beta \rightarrow \beta = \alpha)$  in  $v$ ]
    using logic-actual-nec-2[axiom-instance, equiv-rl] id-act-3[equiv-lr]
      vdash-properties-10 CP by blast
}
hence [ $\forall \beta. \mathcal{A}(\varphi \beta \rightarrow \beta = \alpha)$  in  $v$ ]
  by (rule  $\forall I$ )
hence [ $\mathcal{A}(\forall \beta. \varphi \beta \rightarrow \beta = \alpha)$  in  $v$ ]
  using logic-actual-nec-3[axiom-instance, equiv-rl] by fast
hence [ $\mathcal{A}(\varphi \alpha \ \& \ (\forall \beta. \varphi \beta \rightarrow \beta = \alpha))$  in  $v$ ]
  using 1[conj1] Act-Basic-2[equiv-rl] &I by blast
hence [ $\exists \alpha. \mathcal{A}(\varphi \alpha \ \& \ (\forall \beta. \varphi \beta \rightarrow \beta = \alpha))$  in  $v$ ]
  using  $\exists I$  by fast
thus [ $\mathcal{A}(\exists \alpha. \varphi \alpha \ \& \ (\forall \beta. \varphi \beta \rightarrow \beta = \alpha))$  in  $v$ ]
  using Act-Basic-11[equiv-rl] by fast
qed

```

lemma *A-Exists-2*[PLM]:

```

[( $\exists y. y^P = (\iota x. \varphi x)$ )  $\equiv \mathcal{A}(\exists !x. \varphi x)$  in  $v$ ]
using actual-desc-1 A-Exists-1[equiv-sym]
  intro-elim-6-e by blast

```

lemma *A-descriptions*[PLM]:

```

[ $\exists y. y^P = (\iota x. \langle A!, x^P \rangle) \ \& \ (\forall F. \langle x^P, F \rangle \equiv \varphi F)$  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]:

```

[( $y^P = (\iota x. \langle A!, x^P \rangle) \ \& \ (\forall F. \langle x^P, F \rangle \equiv \varphi F)$ )
 $\rightarrow (\langle A!, y^P \rangle \ \& \ (\forall F. \langle y^P, F \rangle \equiv \varphi F))$  in  $dw$ ]
using y-in-2 by auto

```

lemma *can-ab2*[PLM]:

```

[( $y^P = (\iota x. \langle A!, x^P \rangle) \ \& \ (\forall F. \langle x^P, F \rangle \equiv \varphi F)$ )
 $\rightarrow \langle A!, y^P \rangle$  in  $v$ ]
proof (rule CP)
  assume [ $y^P = (\iota x. \langle A!, x^P \rangle) \ \& \ (\forall F. \langle x^P, F \rangle \equiv \varphi F)$  in  $v$ ]

```

hence $[\mathcal{A}(\downarrow A!, y^P) \ \& \ \mathcal{A}(\forall F . \downarrow y^P, F) \equiv \varphi F) \text{ in } v]$
 using *nec-hintikka-scheme*[*equiv-lr*, *conj1*]
Act-Basic-2[*equiv-lr*] **by** *blast*
 thus $[\downarrow A!, y^P) \text{ in } v]$
 using *oa-facts-8*[*equiv-rl*] $\&E$ **by** *blast*
qed

lemma *desc-encode*[*PLM*]:
 $[\downarrow \iota x . (\downarrow A!, x^P) \ \& \ (\forall F . \downarrow x^P, F) \equiv \varphi F), G] \equiv \varphi G \text{ in } dw]$
proof –
 obtain *a* **where**
 $[a^P = (\iota x . (\downarrow A!, x^P) \ \& \ (\forall F . \downarrow x^P, F) \equiv \varphi F)) \text{ in } dw]$
 using *A-descriptions* **by** (*rule* $\exists E$)
moreover hence $[\downarrow a^P, G] \equiv \varphi G \text{ in } dw]$
 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*]:
 $[\downarrow \iota x . (\downarrow A!, x^P) \ \& \ (\forall F . \downarrow x^P, F) \equiv \varphi F), G] \equiv \mathcal{A}(\varphi G) \text{ in } v]$
proof –
 obtain *a* **where**
 $[a^P = (\iota x . (\downarrow A!, x^P) \ \& \ (\forall F . \downarrow x^P, F) \equiv \varphi F)) \text{ in } v]$
 using *A-descriptions* **by** (*rule* $\exists E$)
moreover {
 hence $[\mathcal{A}(\downarrow A!, a^P) \ \& \ (\forall F . \downarrow a^P, F) \equiv \varphi F) \text{ in } v]$
 using *nec-hintikka-scheme*[*equiv-lr*, *conj1*] **by** *fast*
 hence $[\mathcal{A}(\forall F . \downarrow a^P, F) \equiv \varphi F) \text{ in } v]$
 using *Act-Basic-2*[*equiv-lr*, *conj2*] **by** *blast*
 hence $[\forall F . \mathcal{A}(\downarrow a^P, F) \equiv \varphi F) \text{ in } v]$
 using *logic-actual-nec-3*[*axiom-instance*, *equiv-lr*] **by** *blast*
 hence $[\mathcal{A}(\downarrow a^P, G) \equiv \varphi G) \text{ in } v]$
 using $\forall E$ **by** *fast*
 hence $[\mathcal{A}(\downarrow a^P, G) \equiv \mathcal{A}(\varphi G) \text{ in } v]$
 using *Act-Basic-5*[*equiv-lr*] **by** *fast*
 hence $[\downarrow a^P, G] \equiv \mathcal{A}(\varphi G) \text{ in } v]$
 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* = $\iota x . (\downarrow A!, x^P) \ \& \ (\forall F . \downarrow x^P, F) \equiv (\exists q . q \ \& \ F = (\lambda y . q))$
 have $[\Box(\exists p . \text{ContingentlyTrue } p) \text{ in } v]$
 using *cont-tf-thm-3* *RN* **by** *auto*
 hence $[\mathcal{A}(\exists p . \text{ContingentlyTrue } p) \text{ in } v]$
 using *nec-imp-act*[*deduction*] **by** *simp*
 hence $[\exists p . \mathcal{A}(\text{ContingentlyTrue } p) \text{ in } v]$
 using *Act-Basic-11*[*equiv-lr*] **by** *auto*
then obtain *p*₁ **where**
 $[\mathcal{A}(\text{ContingentlyTrue } p_1) \text{ in } v]$
by (*rule* $\exists E$)
 hence $[\mathcal{A}p_1 \text{ in } v]$
 unfolding *ContingentlyTrue-def*

using *Act-Basic-2*[*equiv-lr*] &E by fast
 hence $[\mathcal{A}p_1 \ \& \ \mathcal{A}((\lambda y . p_1) = (\lambda y . p_1)) \text{ in } v]$
 using &I *id-eq-1*[*THEN RN, THEN nec-imp-act*[*deduction*]] by fast
 hence $[\mathcal{A}(p_1 \ \& \ (\lambda y . p_1) = (\lambda y . p_1)) \text{ in } v]$
 using *Act-Basic-2*[*equiv-rl*] by fast
 hence $[\exists q . \mathcal{A}(q \ \& \ (\lambda y . p_1) = (\lambda y . q)) \text{ in } v]$
 using $\exists I$ by fast
 hence $[\mathcal{A}(\exists q . q \ \& \ (\lambda y . p_1) = (\lambda y . q)) \text{ in } v]$
 using *Act-Basic-11*[*equiv-rl*] by fast
 moreover have $[\llbracket ?x, \lambda y . p_1 \rrbracket \equiv \mathcal{A}(\exists q . q \ \& \ (\lambda y . p_1) = (\lambda y . q)) \text{ in } v]$
 using *desc-nec-encode* by fast
 ultimately have $[\llbracket ?x, \lambda y . p_1 \rrbracket \text{ in } v]$
 using $\equiv E$ by blast
 end

lemma *Box-desc-encode-1*[*PLM*]:

$[\Box(\varphi G \rightarrow \llbracket (\lambda x . \llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) , G \rrbracket \text{ in } v]$

proof (rule CP)

assume $[\Box(\varphi G) \text{ in } v]$

hence $[\mathcal{A}(\varphi G) \text{ in } v]$

using *nec-imp-act*[*deduction*] by auto

thus $[\llbracket (\lambda x . \llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F) , G \rrbracket \text{ in } v]$

using *desc-nec-encode*[*equiv-rl*] by simp

qed

lemma *Box-desc-encode-2*[*PLM*]:

$[\Box(\varphi G \rightarrow \Box(\llbracket (\lambda x . \llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) , G \rrbracket \equiv \varphi G) \text{ in } v]$

proof (rule CP)

assume $a: [\Box(\varphi G) \text{ in } v]$

hence $[\Box(\llbracket (\lambda x . \llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) , G \rrbracket \rightarrow \varphi G) \text{ in } v]$

using *KBasic-1*[*deduction*] by simp

moreover {

have $[\llbracket (\lambda x . \llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) , G \rrbracket \text{ in } v]$

using *a Box-desc-encode-1*[*deduction*] by auto

hence $[\Box(\llbracket (\lambda x . \llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) , G \rrbracket \text{ in } v]$

using *encoding*[*axiom-instance, deduction*] by blast

hence $[\Box(\varphi G \rightarrow \llbracket (\lambda x . \llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) , G \rrbracket \text{ in } v]$

using *KBasic-1*[*deduction*] by simp

}

ultimately show $[\Box(\llbracket (\lambda x . \llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) , G \rrbracket \equiv \varphi G) \text{ in } v]$

using &I *KBasic-4*[*equiv-rl*] by blast

qed

lemma *box-phi-a-1*[*PLM*]:

assumes $[\Box(\forall F . \varphi F \rightarrow \Box(\varphi F)) \text{ in } v]$

shows $[(\llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) \rightarrow \Box(\llbracket A!, x^P \rrbracket)$

$\ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) \text{ in } v]$

proof (rule CP)

assume $a: [(\llbracket A!, x^P \rrbracket) \ \& \ (\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F)) \text{ in } v]$

have $[\Box(\llbracket A!, x^P \rrbracket) \text{ in } v]$

using *oa-facts-2*[*deduction*] *a*[*conj1*] by auto

moreover have $[\Box(\forall F . \llbracket x^P, F \rrbracket \equiv \varphi F) \text{ in } v]$

proof (rule BF[*deduction*]; rule $\forall I$)

fix F

have $\vartheta: [\Box(\varphi F \rightarrow \Box(\varphi F)) \text{ in } v]$

using *assms*[*THEN CBF*[*deduction*]] by (rule $\forall E$)

moreover have $[\Box(\llbracket x^P, F \rrbracket \rightarrow \Box(\llbracket x^P, F \rrbracket)) \text{ in } v]$


```

    using encoding[axiom-necessitation, axiom-instance] by simp
  moreover have  $[\Box\{x^P, F\} \equiv \Box(\varphi F) \text{ in } v]$ 
  proof (rule  $\equiv I$ ; rule CP)
    assume  $[\Box\{x^P, F\} \text{ in } v]$ 
    hence  $[\{x^P, F\} \text{ in } v]$ 
      using qml-2[axiom-instance, deduction] by blast
    hence  $[\varphi F \text{ in } v]$ 
      using a[conj2]  $\forall E$ [where 'a= $\Pi_1$ ]  $\equiv E$  by blast
    thus  $[\Box(\varphi F) \text{ in } v]$ 
      using  $\vartheta$ [THEN qml-2[axiom-instance, deduction], deduction] by simp
  next
    assume  $[\Box(\varphi F) \text{ in } v]$ 
    hence  $[\varphi F \text{ in } v]$ 
      using qml-2[axiom-instance, deduction] by blast
    hence  $[\{x^P, F\} \text{ in } v]$ 
      using a[conj2]  $\forall E$ [where 'a= $\Pi_1$ ]  $\equiv E$  by blast
    thus  $[\Box\{x^P, F\} \text{ in } v]$ 
      using encoding[axiom-instance, deduction] by simp
  qed
  ultimately show  $[\Box(\{x^P, F\} \equiv \varphi F) \text{ in } v]$ 
    using sc-eq-box-box-3[deduction, deduction] &I by blast
  qed
  ultimately show  $[\Box((\Box A!, x^P) \ \& \ (\forall F. \{x^P, F\} \equiv \varphi F)) \text{ in } v]$ 
    using &I KBasic-3[equiv-rl] by blast
  qed

```

lemma box-phi-a-2[PLM]:

```

  assumes  $[\Box(\forall F. \varphi F \rightarrow \Box(\varphi F)) \text{ in } v]$ 
  shows  $[y^P = (\iota x. (\Box A!, x^P) \ \& \ (\forall F. \{x^P, F\} \equiv \varphi F))$ 
     $\rightarrow ((\Box A!, y^P) \ \& \ (\forall F. \{y^P, F\} \equiv \varphi F)) \text{ in } v]$ 
  proof -
    let  $?\psi = \lambda x. (\Box A!, x^P) \ \& \ (\forall F. \{x^P, F\} \equiv \varphi F)$ 
    have  $[\forall x. ?\psi x \rightarrow \Box(?\psi x) \text{ in } v]$ 
      using box-phi-a-1[OF assms]  $\forall I$  by fast
    hence  $[(\exists! x. ?\psi x) \rightarrow (\forall y. y^P = (\iota x. ?\psi x) \rightarrow ?\psi y) \text{ in } v]$ 
      using unique-box-desc[deduction] by fast
    hence  $[(\forall y. y^P = (\iota x. ?\psi x) \rightarrow ?\psi y) \text{ in } v]$ 
      using A-objects-unique modus-ponens by blast
    thus ?thesis by (rule  $\forall E$ )
  qed

```

lemma box-phi-a-3[PLM]:

```

  assumes  $[\Box(\forall F. \varphi F \rightarrow \Box(\varphi F)) \text{ in } v]$ 
  shows  $[(\iota x. (\Box A!, x^P) \ \& \ (\forall F. \{x^P, F\} \equiv \varphi F), G) \equiv \varphi G \text{ in } v]$ 
  proof -
    obtain a where
       $[a^P = (\iota x. (\Box A!, x^P) \ \& \ (\forall F. \{x^P, F\} \equiv \varphi F)) \text{ in } v]$ 
      using A-descriptions by (rule  $\exists E$ )
    moreover {
      hence  $[(\forall F. \{a^P, F\} \equiv \varphi F) \text{ in } v]$ 
        using box-phi-a-2[OF assms, deduction, conj2] by blast
      hence  $[\{a^P, G\} \equiv \varphi G \text{ in } v]$  by (rule  $\forall E$ )
    }
    ultimately show ?thesis
      using l-identity[axiom-instance, deduction, deduction] by fast
  qed

```

lemma null-uni-uniq-1[PLM]:

```

[ $\exists! x . \text{Null } (x^P) \text{ in } v$ ]
proof –
  have [ $\exists x . (\downarrow A!, x^P) \ \& \ (\forall F . \{x^P, F\} \equiv (F \neq F)) \text{ in } v$ ]
    using A-objects[axiom-instance] by simp
  then obtain a where a-prop:
    [ $(\downarrow A!, a^P) \ \& \ (\forall F . \{a^P, F\} \equiv (F \neq F)) \text{ in } v$ ]
    by (rule  $\exists E$ )
  have 1: [ $(\downarrow A!, a^P) \ \& \ (\neg(\exists F . \{a^P, F\})) \text{ in } v$ ]
    using a-prop[conj1] apply (rule  $\&I$ )
  proof –
    {
      assume [ $\exists F . \{a^P, F\} \text{ in } v$ ]
      then obtain P where
        [ $\{a^P, P\} \text{ in } v$ ] by (rule  $\exists E$ )
      hence [ $P \neq P \text{ in } v$ ]
        using a-prop[conj2, THEN  $\forall E$ , equiv-lr] by simp
      hence [ $\neg(\exists F . \{a^P, F\}) \text{ in } v$ ]
        using id-eq-1 reductio-aa-1 by fast
    }
  thus [ $\neg(\exists F . \{a^P, F\}) \text{ in } v$ ]
    using reductio-aa-1 by blast
qed
moreover have [ $\forall y . ((\downarrow A!, y^P) \ \& \ (\neg(\exists F . \{y^P, F\}))) \rightarrow y = a \text{ in } v$ ]
proof (rule  $\forall I$ ; rule CP)
  fix y
  assume 2: [ $(\downarrow A!, y^P) \ \& \ (\neg(\exists F . \{y^P, F\})) \text{ in } v$ ]
  have [ $\forall F . \{y^P, F\} \equiv \{a^P, F\} \text{ in } v$ ]
    using cqt-further-12[deduction] 1[conj2] 2[conj2]  $\&I$  by blast
  thus [ $y = a \text{ in } v$ ]
    using ab-obey-1[deduction, deduction]
     $\&I$ [OF 2[conj1] 1[conj1]] identity- $\nu$ -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]:
[ $\exists! x . \text{Universal } (x^P) \text{ in } v$ ]
proof –
  have [ $\exists x . (\downarrow A!, x^P) \ \& \ (\forall F . \{x^P, F\} \equiv (F = F)) \text{ in } v$ ]
    using A-objects[axiom-instance] by simp
  then obtain a where a-prop:
    [ $(\downarrow A!, a^P) \ \& \ (\forall F . \{a^P, F\} \equiv (F = F)) \text{ in } v$ ]
    by (rule  $\exists E$ )
  have 1: [ $(\downarrow A!, a^P) \ \& \ (\forall F . \{a^P, F\}) \text{ in } v$ ]
    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 [ $\forall y . ((\downarrow A!, y^P) \ \& \ (\forall F . \{y^P, F\})) \rightarrow y = a \text{ in } v$ ]
proof (rule  $\forall I$ ; rule CP)
  fix y
  assume 2: [ $(\downarrow A!, y^P) \ \& \ (\forall F . \{y^P, F\}) \text{ in } v$ ]
  have [ $\forall F . \{y^P, F\} \equiv \{a^P, F\} \text{ in } v$ ]
    using cqt-further-11[deduction] 1[conj2] 2[conj2]  $\&I$  by blast
  thus [ $y = a \text{ in } v$ ]
    using ab-obey-1[deduction, deduction]
     $\&I$ [OF 2[conj1] 1[conj1]] identity- $\nu$ -def
    by presburger

```

qed
ultimately show ?thesis
using &I $\exists I$
unfolding *Universal-def exists-unique-def* by fast
qed

lemma *null-uni-uniq-3*[PLM]:
 $[\exists y . y^P = (\iota x . \text{Null } (x^P)) \text{ in } v]$
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]:
 $[\exists y . y^P = (\iota x . \text{Universal } (x^P)) \text{ in } v]$
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]:
 $[\text{Null } (x^P) \rightarrow \Box(\text{Null } (x^P)) \text{ in } v]$
proof (rule CP)
assume $[\text{Null } (x^P) \text{ in } v]$
hence 1: $[(\Box A!, x^P) \ \& \ (\neg(\exists F . \Box x^P, F))] \text{ in } v]$
unfolding *Null-def* .
have $[\Box(\Box A!, x^P) \text{ in } v]$
using 1[*conj1*] *oa-facts-2*[deduction] by simp
moreover have $[\Box(\neg(\exists F . \Box x^P, F))] \text{ in } v]$
proof –
{
assume $[\neg\Box(\neg(\exists F . \Box x^P, F))] \text{ in } v]$
hence $[\Diamond(\exists F . \Box x^P, F) \text{ in } v]$
unfolding *diamond-def* .
hence $[\exists F . \Diamond\Box x^P, F \text{ in } v]$
using *BF* \Diamond [deduction] by blast
then obtain *P* where $[\Diamond\Box x^P, P \text{ in } v]$
by (rule $\exists E$)
hence $[\Box x^P, P \text{ in } v]$
using *en-eq-3*[*equiv-lr*] by simp
hence $[\exists F . \Box x^P, F \text{ in } v]$
using $\exists I$ by fast
}
thus ?thesis
using 1[*conj2*] *modus-tollens-1 CP*
useful-tautologies-1[deduction] by metis
qed
ultimately show $[\Box\text{Null } (x^P) \text{ in } v]$
unfolding *Null-def*
using &I *KBasic-3*[*equiv-rl*] by blast
qed

lemma *null-uni-facts-2*[PLM]:
 $[\text{Universal } (x^P) \rightarrow \Box(\text{Universal } (x^P)) \text{ in } v]$
proof (rule CP)
assume $[\text{Universal } (x^P) \text{ in } v]$
hence 1: $[(\Box A!, x^P) \ \& \ (\forall F . \Box x^P, F)] \text{ in } v]$
unfolding *Universal-def* .
have $[\Box(\Box A!, x^P) \text{ in } v]$
using 1[*conj1*] *oa-facts-2*[deduction] by simp
moreover have $[\Box(\forall F . \Box x^P, F) \text{ in } v]$
proof (rule *BF*[deduction]; rule $\forall I$)

```

fix F
have [ $\llbracket x^P, F \rrbracket$  in  $v$ ]
  using 1[conj2] by (rule  $\forall E$ )
thus [ $\Box \llbracket x^P, F \rrbracket$  in  $v$ ]
  using encoding[axiom-instance, deduction] by auto
qed
ultimately show [ $\Box Universal (x^P)$  in  $v$ ]
  unfolding Universal-def
  using &I KBasic-3[equiv-rl] by blast
qed

```

lemma null-uni-facts-3[PLM]:

```

[Null ( $\mathbf{a}_\emptyset$ ) in  $v$ ]
proof -
  let  $? \psi = \lambda x . Null x$ 
  have [ $((\exists ! x . ? \psi (x^P)) \rightarrow (\forall y . y^P = (\iota x . ? \psi (x^P)) \rightarrow ? \psi (y^P)))$  in  $v$ ]
    using unique-box-desc[deduction] null-uni-facts-1[THEN  $\forall I$ ] by fast
  have 1: [ $(\forall y . y^P = (\iota x . ? \psi (x^P)) \rightarrow ? \psi (y^P))$  in  $v$ ]
    using unique-box-desc[deduction, deduction] null-uni-uniq-1
      null-uni-facts-1[THEN  $\forall I$ ] by fast
  have [ $\exists y . y^P = (\mathbf{a}_\emptyset)$  in  $v$ ]
    unfolding NullObject-def using null-uni-uniq-3 .
  then obtain  $y$  where [ $y^P = (\mathbf{a}_\emptyset)$  in  $v$ ]
    by (rule  $\exists E$ )
  moreover hence [ $? \psi (y^P)$  in  $v$ ]
    using 1[THEN  $\forall E$ , deduction] unfolding NullObject-def by simp
  ultimately show [ $? \psi (\mathbf{a}_\emptyset)$  in  $v$ ]
    using l-identity[axiom-instance, deduction, deduction] by blast
qed

```

lemma null-uni-facts-4[PLM]:

```

[Universal ( $\mathbf{a}_V$ ) in  $v$ ]
proof -
  let  $? \psi = \lambda x . Universal x$ 
  have [ $((\exists ! x . ? \psi (x^P)) \rightarrow (\forall y . y^P = (\iota x . ? \psi (x^P)) \rightarrow ? \psi (y^P)))$  in  $v$ ]
    using unique-box-desc[deduction] null-uni-facts-2[THEN  $\forall I$ ] by fast
  have 1: [ $(\forall y . y^P = (\iota x . ? \psi (x^P)) \rightarrow ? \psi (y^P))$  in  $v$ ]
    using unique-box-desc[deduction, deduction] null-uni-uniq-2
      null-uni-facts-2[THEN  $\forall I$ ] by fast
  have [ $\exists y . y^P = (\mathbf{a}_V)$  in  $v$ ]
    unfolding UniversalObject-def using null-uni-uniq-4 .
  then obtain  $y$  where [ $y^P = (\mathbf{a}_V)$  in  $v$ ]
    by (rule  $\exists E$ )
  moreover hence [ $? \psi (y^P)$  in  $v$ ]
    using 1[THEN  $\forall E$ , deduction]
    unfolding UniversalObject-def by simp
  ultimately show [ $? \psi (\mathbf{a}_V)$  in  $v$ ]
    using l-identity[axiom-instance, deduction, deduction] by blast
qed

```

lemma aclassical-1[PLM]:

```

[ $\forall R . \exists x y . (\llbracket A!, x^P \rrbracket \ \& \ \llbracket A!, y^P \rrbracket) \ \& \ (x \neq y)$ 
  & ( $\lambda z . (\llbracket R, z^P, x^P \rrbracket) = (\lambda z . (\llbracket R, z^P, y^P \rrbracket))$  in  $v$ ]
proof (rule  $\forall I$ )
  fix R
  obtain  $a$  where  $\vartheta$ :
    [ $(\llbracket A!, a^P \rrbracket \ \& \ (\forall F . \llbracket a^P, F \rrbracket \equiv (\exists y . (\llbracket A!, y^P \rrbracket) \ \& \ F = (\lambda z . (\llbracket R, z^P, y^P \rrbracket) \ \& \ \neg \llbracket y^P, F \rrbracket)))$  in  $v$ ]

```

using *A-objects*[*axiom-instance*] by (rule $\exists E$)
 {
 assume $\neg \langle a^P, (\lambda z . \langle R, z^P, a^P \rangle) \rangle$ in *v*
 hence $\neg (\langle A!, a^P \rangle \ \& \ (\lambda z . \langle R, z^P, a^P \rangle) = (\lambda z . \langle R, z^P, a^P \rangle))$
 $\& \neg \langle a^P, (\lambda z . \langle R, z^P, a^P \rangle) \rangle$ in *v*
 using $\vartheta[\text{conj2}, \text{THEN } \forall E, \text{THEN oth-class-taut-5-d}[\text{equiv-lr}], \text{equiv-lr}]$
 $\text{cqt-further-4}[\text{equiv-lr}] \ \forall E$ by fast
 hence $\langle \langle A!, a^P \rangle \ \& \ (\lambda z . \langle R, z^P, a^P \rangle) = (\lambda z . \langle R, z^P, a^P \rangle) \rightarrow \langle a^P, (\lambda z . \langle R, z^P, a^P \rangle) \rangle$ in *v*
 apply – by *PLM-solver*
 hence $\langle \langle a^P, (\lambda z . \langle R, z^P, a^P \rangle) \rangle \rangle$ in *v*
 using $\vartheta[\text{conj1}] \text{ id-eq-1 } \& I \text{ vdash-properties-10}$ by fast
 }
 hence 1: $\langle \langle a^P, (\lambda z . \langle R, z^P, a^P \rangle) \rangle \rangle$ in *v*
 using *reductio-aa-1 CP if-p-then-p* by blast
 then obtain *b* where ξ :
 $\langle \langle A!, b^P \rangle \ \& \ (\lambda z . \langle R, z^P, a^P \rangle) = (\lambda z . \langle R, z^P, b^P \rangle) \ \& \neg \langle b^P, (\lambda z . \langle R, z^P, a^P \rangle) \rangle \rangle$ in *v*
 using $\vartheta[\text{conj2}, \text{THEN } \forall E, \text{equiv-lr}] \ \exists E$ by blast
 have $[a \neq b \text{ in } v]$
 proof –
 {
 assume $[a = b \text{ in } v]$
 hence $\langle \langle b^P, (\lambda z . \langle R, z^P, a^P \rangle) \rangle \rangle$ in *v*
 using 1 *l-identity*[*axiom-instance*, *deduction*, *deduction*] by fast
 hence ?thesis
 using $\xi[\text{conj2}] \text{ reductio-aa-1}$ by blast
 }
 thus ?thesis using *reductio-aa-1* by blast
 qed
 hence $\langle \langle A!, a^P \rangle \ \& \ \langle A!, b^P \rangle \ \& a \neq b \ \& (\lambda z . \langle R, z^P, a^P \rangle) = (\lambda z . \langle R, z^P, b^P \rangle) \rangle$ in *v*
 using $\vartheta[\text{conj1}] \ \xi[\text{conj1}, \text{conj1}] \ \xi[\text{conj1}, \text{conj2}] \ \& I$ by presburger
 hence $\langle \exists y . \langle A!, a^P \rangle \ \& \ \langle A!, y^P \rangle \ \& a \neq y \ \& (\lambda z . \langle R, z^P, a^P \rangle) = (\lambda z . \langle R, z^P, y^P \rangle) \rangle$ in *v*
 using $\exists I$ by fast
 thus $\langle \exists x y . \langle A!, x^P \rangle \ \& \ \langle A!, y^P \rangle \ \& x \neq y \ \& (\lambda z . \langle R, z^P, x^P \rangle) = (\lambda z . \langle R, z^P, y^P \rangle) \rangle$ in *v*
 using $\exists I$ by fast
 qed

lemma *aclassical-2*[*PLM*]:

$\langle \forall R . \exists x y . \langle A!, x^P \rangle \ \& \ \langle A!, y^P \rangle \ \& (x \neq y) \ \& (\lambda z . \langle R, x^P, z^P \rangle) = (\lambda z . \langle R, y^P, z^P \rangle) \rangle$ in *v*

proof (rule $\forall I$)

fix *R*

obtain *a* where ϑ :

$\langle \langle A!, a^P \rangle \ \& \ (\forall F . \langle a^P, F \rangle \equiv (\exists y . \langle A!, y^P \rangle \ \& F = (\lambda z . \langle R, y^P, z^P \rangle) \ \& \neg \langle y^P, F \rangle)) \rangle$ in *v*

using *A-objects*[*axiom-instance*] by (rule $\exists E$)

{

assume $\neg \langle a^P, (\lambda z . \langle R, a^P, z^P \rangle) \rangle$ in *v*

hence $\neg (\langle A!, a^P \rangle \ \& \ (\lambda z . \langle R, a^P, z^P \rangle) = (\lambda z . \langle R, a^P, z^P \rangle))$

$\& \neg \langle a^P, (\lambda z . \langle R, a^P, z^P \rangle) \rangle$ in *v*

using $\vartheta[\text{conj2}, \text{THEN } \forall E, \text{THEN oth-class-taut-5-d}[\text{equiv-lr}], \text{equiv-lr}]$

$\text{cqt-further-4}[\text{equiv-lr}] \ \forall E$ by fast

hence $\langle \langle A!, a^P \rangle \ \& \ (\lambda z . \langle R, a^P, z^P \rangle) = (\lambda z . \langle R, a^P, z^P \rangle) \rightarrow \langle a^P, (\lambda z . \langle R, a^P, z^P \rangle) \rangle$ in *v*

$\rightarrow \langle a^P, (\lambda z . \langle R, a^P, z^P \rangle) \rangle$ in *v*

apply – by *PLM-solver*
 hence $[\llbracket a^P, (\lambda z . \llbracket R, a^P, z^P \rrbracket) \rrbracket \text{ in } v]$
 using $\vartheta[\text{conj1}]$ *id-eq-1* & *I vdash-properties-10* by *fast*
 }
 hence 1: $[\llbracket a^P, (\lambda z . \llbracket R, a^P, z^P \rrbracket) \rrbracket \text{ in } v]$
 using *reductio-aa-1 CP if-p-then-p* by *blast*
 then obtain *b* where ξ :
 $[\llbracket A!, b^P \rrbracket \ \& \ (\lambda z . \llbracket R, a^P, z^P \rrbracket) = (\lambda z . \llbracket R, b^P, z^P \rrbracket)$
 $\ \& \ \neg \llbracket b^P, (\lambda z . \llbracket R, a^P, z^P \rrbracket) \rrbracket \text{ in } v]$
 using $\vartheta[\text{conj2}, \text{THEN } \forall E, \text{equiv-lr}] \exists E$ by *blast*
 have $[a \neq b \text{ in } v]$
 proof –
 {
 assume $[a = b \text{ in } v]$
 hence $[\llbracket b^P, (\lambda z . \llbracket R, a^P, z^P \rrbracket) \rrbracket \text{ in } v]$
 using 1 *l-identity* [*axiom-instance, deduction, deduction*] by *fast*
 hence *?thesis* using $\xi[\text{conj2}]$ *reductio-aa-1* by *blast*
 }
 thus *?thesis* using $\xi[\text{conj2}]$ *reductio-aa-1* by *blast*
 qed
 hence $[\llbracket A!, a^P \rrbracket \ \& \ \llbracket A!, b^P \rrbracket \ \& \ a \neq b$
 $\ \& \ (\lambda z . \llbracket R, a^P, z^P \rrbracket) = (\lambda z . \llbracket R, b^P, z^P \rrbracket) \text{ in } v]$
 using $\vartheta[\text{conj1}] \xi[\text{conj1}, \text{conj1}] \xi[\text{conj1}, \text{conj2}]$ & *I* by *presburger*
 hence $[\exists y . \llbracket A!, a^P \rrbracket \ \& \ \llbracket A!, y^P \rrbracket \ \& \ a \neq y$
 $\ \& \ (\lambda z . \llbracket R, a^P, z^P \rrbracket) = (\lambda z . \llbracket R, y^P, z^P \rrbracket) \text{ in } v]$
 using $\exists I$ by *fast*
 thus $[\exists x y . \llbracket A!, x^P \rrbracket \ \& \ \llbracket A!, y^P \rrbracket \ \& \ x \neq y$
 $\ \& \ (\lambda z . \llbracket R, x^P, z^P \rrbracket) = (\lambda z . \llbracket R, y^P, z^P \rrbracket) \text{ in } v]$
 using $\exists I$ by *fast*
 qed

lemma *aclassical-3[PLM]*:

$[\forall F . \exists x y . \llbracket A!, x^P \rrbracket \ \& \ \llbracket A!, y^P \rrbracket \ \& \ (x \neq y)$
 $\ \& \ ((\lambda^0 \llbracket F, x^P \rrbracket) = (\lambda^0 \llbracket F, y^P \rrbracket)) \text{ in } v]$
 proof (*rule* $\forall I$)
 fix *R*
 obtain *a* where ϑ :
 $[\llbracket A!, a^P \rrbracket \ \& \ (\forall F . \llbracket a^P, F \rrbracket \equiv (\exists y . \llbracket A!, y^P \rrbracket$
 $\ \& \ F = (\lambda z . \llbracket R, y^P \rrbracket) \ \& \ \neg \llbracket y^P, F \rrbracket)) \text{ in } v]$
 using *A-objects* [*axiom-instance*] by (*rule* $\exists E$)
 {
 assume $[\neg \llbracket a^P, (\lambda z . \llbracket R, a^P \rrbracket) \rrbracket \text{ in } v]$
 hence $[\neg (\llbracket A!, a^P \rrbracket \ \& \ (\lambda z . \llbracket R, a^P \rrbracket) = (\lambda z . \llbracket R, a^P \rrbracket))$
 $\ \& \ \neg \llbracket a^P, (\lambda z . \llbracket R, a^P \rrbracket) \rrbracket \text{ in } v]$
 using $\vartheta[\text{conj2}, \text{THEN } \forall E, \text{THEN } \text{oth-class-taut-5-d}[\text{equiv-lr}], \text{equiv-lr}]$
 $\text{cqt-further-4}[\text{equiv-lr}] \forall E$ by *fast*
 hence $[\llbracket A!, a^P \rrbracket \ \& \ (\lambda z . \llbracket R, a^P \rrbracket) = (\lambda z . \llbracket R, a^P \rrbracket)$
 $\rightarrow \llbracket a^P, (\lambda z . \llbracket R, a^P \rrbracket) \rrbracket \text{ in } v]$
 apply – by *PLM-solver*
 hence $[\llbracket a^P, (\lambda z . \llbracket R, a^P \rrbracket) \rrbracket \text{ in } v]$
 using $\vartheta[\text{conj1}]$ *id-eq-1* & *I vdash-properties-10* by *fast*
 }
 hence 1: $[\llbracket a^P, (\lambda z . \llbracket R, a^P \rrbracket) \rrbracket \text{ in } v]$
 using *reductio-aa-1 CP if-p-then-p* by *blast*
 then obtain *b* where ξ :
 $[\llbracket A!, b^P \rrbracket \ \& \ (\lambda z . \llbracket R, a^P \rrbracket) = (\lambda z . \llbracket R, b^P \rrbracket)$
 $\ \& \ \neg \llbracket b^P, (\lambda z . \llbracket R, a^P \rrbracket) \rrbracket \text{ in } v]$
 using $\vartheta[\text{conj2}, \text{THEN } \forall E, \text{equiv-lr}] \exists E$ by *blast*

```

have [a ≠ b in v]
proof -
{
  assume [a = b in v]
  hence [(λ z . (R, aP)) 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, aP) = (R, bP) in v]
    unfolding identityo-def
    using ξ[conj1, conj2] by auto
  hence [(λ0 (R, aP)) = (λ0 (R, bP)) in v]
    using lambda-p-q-p-eq-q[equiv-rl] by simp
}
ultimately have [(A!, aP) & (A!, bP) & a ≠ b
  & ((λ0 (R, aP)) = (λ0 (R, bP))) in v]
  using ∅[conj1] ξ[conj1, conj1] ξ[conj1, conj2] & I
  by presburger
hence [∃ y . (A!, aP) & (A!, yP) & a ≠ y
  & (λ0 (R, aP)) = (λ0 (R, yP)) in v]
  using ∃ I by fast
thus [∃ x y . (A!, xP) & (A!, yP) & x ≠ y
  & (λ0 (R, xP)) = (λ0 (R, yP)) in v]
  using ∃ I by fast
qed

```

lemma aclassical2[PLM]:

```

[∃ x y . (A!, xP) & (A!, yP) & x ≠ y & (∀ F . (F, xP) ≡ (F, yP)) in v]
proof -
  let ?R1 = λ2 (λ x y . ∀ F . (F, xP) ≡ (F, yP))
  have [∃ x y . (A!, xP) & (A!, yP) & x ≠ y
    & (λ z . (?R1, zP, xP)) = (λ z . (?R1, zP, yP)) in v]
    using aclassical-1 by (rule ∀ E)
  then obtain a where
    [∃ y . (A!, aP) & (A!, yP) & a ≠ y
    & (λ z . (?R1, zP, aP)) = (λ z . (?R1, zP, yP)) in v]
    by (rule ∃ E)
  then obtain b where ab-prop:
    [(A!, aP) & (A!, bP) & a ≠ b
    & (λ z . (?R1, zP, aP)) = (λ z . (?R1, zP, bP)) in v]
    by (rule ∃ E)
  have [(?R1, aP, aP) 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 . (?R1, zP, aP), aP) in v]
    apply - apply (rule beta-C-meta-1[equiv-rl])
    apply show-proper
    by auto
  hence [(λ z . (?R1, zP, bP), aP) in v]
    using ab-prop[conj2] l-identity[axiom-instance, deduction, deduction]
    by fast
  hence [(?R1, aP, bP) in v]
    apply (safe intro!: beta-C-meta-1[where φ=

```

$\lambda z . (\lambda^2 (\lambda x y . \forall F . \langle F, x^P \rangle \equiv \langle F, y^P \rangle), z, b^P), \text{equiv-lr}]$
 by *show-proper*
 moreover have *IsProperInXY* $(\lambda x y . \forall F . \langle F, x \rangle \equiv \langle F, y \rangle)$
 by *show-proper*
 ultimately have $[\forall F . \langle F, a^P \rangle \equiv \langle F, b^P \rangle \text{ in } v]$
 using *beta-C-meta-2[equiv-lr]* by *blast*
 hence $[\langle A!, a^P \rangle \ \& \ \langle A!, b^P \rangle \ \& \ a \neq b \ \& \ (\forall F . \langle F, a^P \rangle \equiv \langle F, b^P \rangle) \text{ in } v]$
 using *ab-prop[conj1]* & *I* by *presburger*
 hence $[\exists y . \langle A!, a^P \rangle \ \& \ \langle A!, y^P \rangle \ \& \ a \neq y \ \& \ (\forall F . \langle F, a^P \rangle \equiv \langle F, y^P \rangle) \text{ in } v]$
 using $\exists I$ by *fast*
 thus *?thesis* using $\exists I$ by *fast*
 qed

A.9.13. Propositional Properties

lemma *prop-prop2-1*:
 $[\forall p . \exists F . F = (\lambda x . p) \text{ in } v]$
proof (*rule* $\forall I$)
 fix *p*
 have $[(\lambda x . p) = (\lambda x . p) \text{ in } v]$
 using *id-eq-prop-prop-1* by *auto*
 thus $[\exists F . F = (\lambda x . p) \text{ in } v]$
 by *PLM-solver*
 qed

lemma *prop-prop2-2*:
 $[F = (\lambda x . p) \rightarrow \Box(\forall x . \langle F, x^P \rangle \equiv p) \text{ in } v]$
proof (*rule* *CP*)
 assume 1: $[F = (\lambda x . p) \text{ in } v]$
 {
 fix *v*
 {
 fix *x*
 have $[\langle (\lambda x . p), x^P \rangle \equiv p \text{ in } v]$
 apply (*rule* *beta-C-meta-1*)
 by *show-proper*
 }
 hence $[\forall x . \langle (\lambda x . p), x^P \rangle \equiv p \text{ in } v]$
 by (*rule* $\forall I$)
 }
 hence $[\Box(\forall x . \langle (\lambda x . p), x^P \rangle \equiv p) \text{ in } v]$
 by (*rule* *RN*)
 thus $[\Box(\forall x . \langle F, x^P \rangle \equiv p) \text{ in } v]$
 using *l-identity[axiom-instance,deduction,deduction,*
 $OF \ 1 \ [THEN \ id-eq-prop-prop-2[deduction]]]$ by *fast*
 qed

lemma *prop-prop2-3*:
 $[Propositional \ F \rightarrow \Box(Propositional \ F) \text{ in } v]$
proof (*rule* *CP*)
 assume $[Propositional \ F \text{ in } v]$
 hence $[\exists p . F = (\lambda x . p) \text{ in } v]$
 unfolding *Propositional-def* .
 then obtain *q* where $[F = (\lambda x . q) \text{ in } v]$
 by (*rule* $\exists E$)
 hence $[\Box(F = (\lambda x . q)) \text{ in } v]$
 using *id-nec[equiv-lr]* by *auto*
 hence $[\exists p . \Box(F = (\lambda x . p)) \text{ in } v]$

using $\exists I$ by fast
 thus $[\Box(\text{Propositional } F) \text{ in } v]$
 unfolding *Propositional-def*
 using *sign-S5-thm-1*[deduction] by fast
 qed

lemma *prop-indis*:

$[\text{Indiscriminate } F \rightarrow (\neg(\exists x y . \langle F, x^P \rangle \ \& \ (\neg\langle F, y^P \rangle))) \text{ in } v]$
 proof (rule CP)
 assume $[\text{Indiscriminate } F \text{ in } v]$
 hence 1: $[\Box((\exists x . \langle F, x^P \rangle) \rightarrow (\forall x . \langle F, x^P \rangle)) \text{ in } v]$
 unfolding *Indiscriminate-def* .
 {
 assume $[\exists x y . \langle F, x^P \rangle \ \& \ \neg\langle F, y^P \rangle \text{ in } v]$
 then obtain x where $[\exists y . \langle F, x^P \rangle \ \& \ \neg\langle F, y^P \rangle \text{ in } v]$
 by (rule $\exists E$)
 then obtain y where 2: $[\langle F, x^P \rangle \ \& \ \neg\langle F, y^P \rangle \text{ in } v]$
 by (rule $\exists E$)
 hence $[\exists x . \langle F, x^P \rangle \text{ in } v]$
 using $\&E(1) \ \exists I$ by fast
 hence $[\forall x . \langle F, x^P \rangle \text{ in } v]$
 using 1[*THEN qml-2*[*axiom-instance*, *deduction*], *deduction*] by fast
 hence $[\langle F, y^P \rangle \text{ in } v]$
 using *cqt-orig-1*[*deduction*] by fast
 hence $[\langle F, y^P \rangle \ \& \ (\neg\langle F, y^P \rangle) \text{ in } v]$
 using 2 $\&I \ \&E$ by fast
 hence $[\neg(\exists x y . \langle F, x^P \rangle \ \& \ \neg\langle F, y^P \rangle) \text{ in } v]$
 using *pl-1*[*axiom-instance*, *deduction*, *THEN modus-tollens-1*]
oth-class-taut-1-a by blast
 }
 thus $[\neg(\exists x y . \langle F, x^P \rangle \ \& \ \neg\langle F, y^P \rangle) \text{ in } v]$
 using *reductio-aa-2 if-p-then-p deduction-theorem* by blast
 qed

lemma *prop-in-thm*:

$[\text{Propositional } F \rightarrow \text{Indiscriminate } F \text{ in } v]$
 proof (rule CP)
 assume $[\text{Propositional } F \text{ in } v]$
 hence $[\Box(\text{Propositional } F) \text{ in } v]$
 using *prop-prop2-3*[deduction] by auto
 moreover {
 fix w
 assume $[\exists p . (F = (\lambda y . p)) \text{ in } w]$
 then obtain q where *q-prop*: $[F = (\lambda y . q) \text{ in } w]$
 by (rule $\exists E$)
 {
 assume $[\exists x . \langle F, x^P \rangle \text{ in } w]$
 then obtain a where $[\langle F, a^P \rangle \text{ in } w]$
 by (rule $\exists E$)
 hence $[(\lambda y . q, a^P) \text{ in } w]$
 using *q-prop l-identity*[*axiom-instance*, *deduction*, *deduction*] by fast
 hence q : $[q \text{ in } w]$
 apply (*safe intro!*: *beta-C-meta-1*[*where* $\varphi = \lambda y . q$, *equiv-lr*])
 apply *show-proper*
 by *simp*
 }
 }

```

    fix x
    have [(λ y . q, xP) in w]
      apply (safe intro!: q beta-C-meta-1[equiv-rl])
      by show-proper
    hence [(F, xP) in w]
      using q-prop[eq-sym] l-identity[axiom-instance, deduction, deduction]
      by fast
  }
  hence [∀ x . (F, xP) in w]
    by (rule ∀ I)
}
hence [(∃ x . (F, xP)) → (∀ x . (F, xP)) 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  $\lambda x . (\langle E!, x^P \rangle \lambda x . \neg \neg (\langle E!, x^P \rangle))$ )
    using oth-class-taut-4-b apply simp
    using a-objects-exist-3 by auto
  ultimately have  $[\Box (\forall x . \langle E!^-, x^P \rangle) \text{ in } v]$ 
    unfolding Indiscriminate-def
    using qml-1 [axiom-instance, deduction, deduction] by blast
  thus  $[\Box (\forall x . \neg (\langle E!, x^P \rangle) \text{ in } v)]$ 
    apply -
    apply (PLM-subst-method  $\lambda x . (\langle E!^-, x^P \rangle \lambda x . \neg (\langle E!, x^P \rangle))$ )
    using thm-relation-negation-1-1 by auto
next
  show  $[\neg \Box (\forall x . \neg (\langle E!, x^P \rangle) \text{ in } v)]$ 
    using o-objects-exist-1
    unfolding diamond-def exists-def
    apply -
    apply (PLM-subst-method  $\neg \neg (\forall x . \neg (\langle E!, x^P \rangle) \forall x . \neg (\langle E!, x^P \rangle))$ )
    using oth-class-taut-4-b [equiv-sym] by auto
qed

lemma prop-in-f-3-c:
   $[\neg (\text{Indiscriminate } (O!)) \text{ in } v]$ 
proof (rule reductio-aa-2)
  show  $[\neg (\forall x . \langle O!, x^P \rangle) \text{ in } v]$ 
    using a-objects-exist-2 [THEN qml-2 [axiom-instance, deduction]]
    by blast
next
  assume  $[\text{Indiscriminate } O! \text{ in } v]$ 
  thus  $[(\forall x . \langle O!, x^P \rangle) \text{ in } v]$ 
    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:
   $[\neg (\text{Indiscriminate } (A!)) \text{ in } v]$ 
proof (rule reductio-aa-2)
  show  $[\neg (\forall x . \langle A!, x^P \rangle) \text{ in } v]$ 
    using o-objects-exist-3 [THEN qml-2 [axiom-instance, deduction]]
    by blast
next
  assume  $[\text{Indiscriminate } A! \text{ in } v]$ 
  thus  $[(\forall x . \langle A!, x^P \rangle) \text{ in } v]$ 
    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:
   $[\neg (\text{Propositional } E!) \text{ in } v]$ 
using prop-in-thm [deduction] prop-in-f-3-a modus-tollens-1 CP
by meson

lemma prop-in-f-4-b:
   $[\neg (\text{Propositional } (E!^-)) \text{ in } v]$ 
using prop-in-thm [deduction] prop-in-f-3-b modus-tollens-1 CP
by meson

```

```

lemma prop-in-f-4-c:
  [¬(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:
  [¬(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:
  [◇(∃ p . F = (λ x . p)) → (∃ p . F = (λ x . p)) in v]
  proof (rule CP)
    assume [◇(∃ p . F = (λ x . p)) in v]
    hence [∃ p . ◇(F = (λ x . p)) in v]
      using BF◇[deduction] by auto
    then obtain p where [◇(F = (λ x . p)) in v]
      by (rule ∃ E)
    hence [◇□(∀ x. ⌊xP, F⌋ ≡ ⌊xP, λx. p⌋) in v]
      unfolding identity-defs .
    hence [□(∀ x. ⌊xP, F⌋ ≡ ⌊xP, λx. p⌋) in v]
      using 5◇[deduction] by auto
    hence [(F = (λ x . p)) in v]
      unfolding identity-defs .
    thus [∃ p . (F = (λ x . p)) in v]
      by PLM-solver
  qed

```

```

lemma prop-prop-nec-2:
  [(∀ p . F ≠ (λ x . p)) → □(∀ p . F ≠ (λ x . p)) in v]
  apply (PLM-subst-method
    ¬(∃ p . (F = (λ x . p)))
    (∀ p . ¬(F = (λ x . p))))
  using cqt-further-4 apply blast
  apply (PLM-subst-method
    ¬◇(∃ p. F = (λx. p))
    □¬(∃ p. F = (λx. p)))
  using KBasic2-4[equiv-sym] prop-prop-nec-1
    contraposition-1 by auto

```

```

lemma prop-prop-nec-3:
  [(∃ p . F = (λ x . p)) → □(∃ p . F = (λ x . p)) in v]
  using prop-prop-nec-1 derived-S5-rules-1-b by simp

```

```

lemma prop-prop-nec-4:
  [◇(∀ p . F ≠ (λ x . p)) → (∀ p . F ≠ (λ x . p)) in v]
  using prop-prop-nec-2 derived-S5-rules-2-b by simp

```

```

lemma enc-prop-nec-1:
  [◇(∀ F . ⌊xP, F⌋ → (∃ p . F = (λ x . p)))
  → (∀ F . ⌊xP, F⌋ → (∃ p . F = (λ x . p))) in v]
  proof (rule CP)
    assume [◇(∀ F . ⌊xP, F⌋ → (∃ p . F = (λx. p))) in v]
    hence 1: [(∀ F . ◇(⌊xP, F⌋ → (∃ p . F = (λx. p)))) in v]
      using Buridan◇[deduction] by auto
    {
      fix Q
    }
  qed

```

```

    assume [ $\{x^P, Q\}$  in  $v$ ]
    hence [ $\Box\{x^P, Q\}$  in  $v$ ]
      using encoding[axiom-instance, deduction] by auto
    moreover have [ $\Diamond(\{x^P, Q\} \rightarrow (\exists p. Q = (\lambda x. p)))$  in  $v$ ]
      using cqt-1[axiom-instance, deduction] 1 by fast
    ultimately have [ $\Diamond(\exists p. Q = (\lambda x. p))$  in  $v$ ]
      using KBasic2-9[equiv-lr, deduction] by auto
    hence [ $(\exists p. Q = (\lambda x. p))$  in  $v$ ]
      using prop-prop-nec-1[deduction] by auto
  }
  thus [ $(\forall F. \{x^P, F\} \rightarrow (\exists p. F = (\lambda x. p)))$  in  $v$ ]
    apply – by PLM-solver
qed

```

```

lemma enc-prop-nec-2:
  [ $(\forall F. \{x^P, F\} \rightarrow (\exists p. F = (\lambda x. p))) \rightarrow \Box(\forall F. \{x^P, F\} \rightarrow (\exists p. F = (\lambda x. p)))$  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  $x \equiv (\Box A!, x) \ \& \ (\forall F. \{x, F\} \rightarrow \text{Propositional } F)$ 

definition EncodeProposition (infixl  $\Sigma$  70) where
   $x\Sigma p \equiv (\Box A!, x) \ \& \ \{x, \lambda x. p\}$ 

definition TrueInSituation (infixl  $\models$  10) where
   $x \models p \equiv \text{Situation } x \ \& \ x\Sigma p$ 

definition PossibleWorld where
  PossibleWorld  $x \equiv \text{Situation } x \ \& \ \Diamond(\forall p. x\Sigma p \equiv p)$ 

```

A.10.2. Auxiliary Lemmata

```

lemma possit-sit-1:
  [Situation  $(x^P) \equiv \Box(\text{Situation } (x^P))$  in  $v$ ]
proof (rule  $\equiv I$ ; rule CP)
  assume [Situation  $(x^P)$  in  $v$ ]
  hence 1: [ $(\Box A!, x^P) \ \& \ (\forall F. \{x^P, F\} \rightarrow \text{Propositional } F)$  in  $v$ ]
    unfolding Situation-def by auto
  have [ $\Box(\Box A!, x^P)$  in  $v$ ]
    using 1[conj1, THEN oa-facts-2[deduction]] .
  moreover have [ $\Box(\forall F. \{x^P, F\} \rightarrow \text{Propositional } F)$  in  $v$ ]
    using 1[conj2] unfolding Propositional-def
    by (rule enc-prop-nec-2[deduction])
  ultimately show [ $\Box \text{Situation } (x^P)$  in  $v$ ]
    unfolding Situation-def
    apply cut-tac apply (rule KBasic-3[equiv-rl])
    by (rule intro-elim-1)
next

```

```

  assume  $\Box \text{Situation } (x^P) \text{ in } v$ 
  thus  $\text{Situation } (x^P) \text{ in } v$ 
    using qml-2[axiom-instance, deduction] by auto
qed

```

```

lemma possworld-nec:
   $[\text{PossibleWorld } (x^P) \equiv \Box(\text{PossibleWorld } (x^P)) \text{ in } v]$ 
  apply (rule  $\equiv I$ ; 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 TrueInWorldNec:
   $[(x^P) \models p \equiv \Box((x^P) \models p) \text{ in } v]$ 
  proof (rule  $\equiv I$ ; rule CP)
    assume  $[x^P \models p \text{ in } v]$ 
    hence  $[\text{Situation } (x^P) \ \& \ (\!|A!, x^P|) \ \& \ \{x^P, \lambda x. p\}) \text{ in } v]$ 
      unfolding TrueInSituation-def EncodeProposition-def .
    hence  $[\Box \text{Situation } (x^P) \ \& \ \Box(\!|A!, x^P|) \ \& \ \Box\{x^P, \lambda x. p\} \text{ in } v]$ 
      using & I & E possit-sit-1[equiv-lr] oa-facts-2[deduction]
        encoding[axiom-instance, deduction] by metis
    thus  $[\Box((x^P) \models p) \text{ in } v]$ 
      unfolding TrueInSituation-def EncodeProposition-def
      using KBasic-3[equiv-rl] & I & E by metis
  next
    assume  $[\Box(x^P \models p) \text{ in } v]$ 
    thus  $[x^P \models p \text{ in } v]$ 
      using qml-2[axiom-instance, deduction] by auto
  qed

```

```

lemma PossWorldAux:
   $[(\!|A!, x^P|) \ \& \ (\forall F. (\{x^P, F\} \equiv (\exists p. p \ \& \ (F = (\lambda x. p))))) \rightarrow (\text{PossibleWorld } (x^P)) \text{ in } v]$ 
  proof (rule CP)
    assume DefX:  $[(\!|A!, x^P|) \ \& \ (\forall F. (\{x^P, F\} \equiv (\exists p. p \ \& \ (F = (\lambda x. p))))) \text{ in } v]$ 

    have  $[\text{Situation } (x^P) \text{ in } v]$ 
    proof -
      have  $[(\!|A!, x^P|) \text{ in } v]$ 
      using DefX[conj1] .
      moreover have  $[(\forall F. \{x^P, F\} \rightarrow \text{Propositional } F) \text{ in } v]$ 
      proof (rule  $\forall I$ ; rule CP)
        fix F
        assume  $[\{x^P, F\} \text{ in } v]$ 
        moreover have  $[\{x^P, F\} \equiv (\exists p. p \ \& \ (F = (\lambda x. p))) \text{ in } v]$ 
          using DefX[conj2] cqt-1[axiom-instance, deduction] by auto
        ultimately have  $[(\exists p. p \ \& \ (F = (\lambda x. p))) \text{ in } v]$ 
          using  $\equiv E(1)$  by blast
        then obtain p where  $[p \ \& \ (F = (\lambda x. p)) \text{ in } v]$ 
          by (rule  $\exists E$ )
        hence  $[(F = (\lambda x. p)) \text{ in } v]$ 
          by (rule & E(2))
        hence  $[(\exists p. (F = (\lambda x. p))) \text{ in } v]$ 

```

```

    by PLM-solver
  thus [Propositional F in v]
    unfolding Propositional-def .
qed
ultimately show [Situation (xP) in v]
  unfolding Situation-def by (rule &I)
qed
moreover have [ $\Diamond(\forall p. x^P \Sigma p \equiv p)$  in v]
  unfolding EncodeProposition-def
  proof (rule TBasic[deduction]; rule  $\forall I$ )
    fix q
    have EncodeLambda:
      [ $\llbracket x^P, \lambda x. q \rrbracket \equiv (\exists p. p \ \& \ ((\lambda x. q) = (\lambda x. p)))$  in v]
      using DefX[conj2] by (rule cqt-1[axiom-instance, deduction])
    moreover {
      assume [q in v]
      moreover have [ $(\lambda x. q) = (\lambda x. q)$  in v]
        using id-eq-prop-prop-1 by auto
      ultimately have [ $q \ \& \ ((\lambda x. q) = (\lambda x. q))$  in v]
        by (rule &I)
      hence [ $\exists p. p \ \& \ ((\lambda x. q) = (\lambda x. p))$  in v]
        by PLM-solver
      moreover have [ $\llbracket A!, x^P \rrbracket$  in v]
        using DefX[conj1] .
      ultimately have [ $\llbracket A!, x^P \rrbracket \ \& \ \llbracket x^P, \lambda x. q \rrbracket$  in v]
        using EncodeLambda[equiv-rl] &I by auto
    }
    moreover {
      assume [ $\llbracket A!, x^P \rrbracket \ \& \ \llbracket x^P, \lambda x. q \rrbracket$  in v]
      hence [ $\llbracket x^P, \lambda x. q \rrbracket$  in v]
        using &E(2) by auto
      hence [ $\exists p. p \ \& \ ((\lambda x. q) = (\lambda x. p))$  in v]
        using EncodeLambda[equiv-lr] by auto
      then obtain p where p-and-lambda-q-is-lambda-p:
        [ $p \ \& \ ((\lambda x. q) = (\lambda x. p))$  in v]
        by (rule  $\exists E$ )
      have [ $\llbracket (\lambda x. p), x^P \rrbracket \equiv p$  in v]
        apply (rule beta-C-meta-1)
        by show-proper
      hence [ $\llbracket (\lambda x. p), x^P \rrbracket$  in v]
        using p-and-lambda-q-is-lambda-p[conj1]  $\equiv E(2)$  by auto
      hence [ $\llbracket (\lambda x. q), x^P \rrbracket$  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 [ $\llbracket (\lambda x. q), x^P \rrbracket \equiv q$  in v]
        apply (rule beta-C-meta-1) by show-proper
      ultimately have [q in v]
        using  $\equiv E(1)$  by blast
    }
  }
  ultimately show [ $\llbracket A!, x^P \rrbracket \ \& \ \llbracket x^P, \lambda x. q \rrbracket \equiv q$  in v]
    using &I  $\equiv I$  CP by auto
qed

ultimately show [PossibleWorld (xP) 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:*

```

 $\forall x . [PossibleWorld (x^P) \text{ in } w] \longrightarrow$ 
 $(\exists v . \forall p . [(x^P \models p) \text{ in } w] \longleftrightarrow [p \text{ in } v])$ 
proof
  fix  $x$ 
  {
    assume PossWorldX:  $[PossibleWorld (x^P) \text{ in } w]$ 
    hence SituationX:  $[Situation (x^P) \text{ in } w]$ 
    unfolding PossibleWorld-def apply cut-tac by PLM-solver
    have PossWorldExpanded:
       $[(\langle A!, x^P \rangle) \ \& \ (\forall F. \langle x^P, F \rangle \rightarrow (\exists p. F = (\lambda x. p)))]$ 
       $\ \& \ \Diamond(\forall p. (\langle A!, x^P \rangle) \ \& \ \langle x^P, \lambda x. p \rangle \equiv p) \text{ in } w]$ 
    using PossWorldX
    unfolding PossibleWorld-def Situation-def
      Propositional-def EncodeProposition-def .
    have AbstractX:  $[(\langle A!, x^P \rangle) \text{ in } w]$ 
    using PossWorldExpanded[conj1, conj1] .

    have  $[\Diamond(\forall p. \langle x^P, \lambda x. p \rangle \equiv p) \text{ in } w]$ 
    apply (PLM-subst-method
       $\lambda p. (\langle A!, x^P \rangle) \ \& \ \langle x^P, \lambda x. p \rangle$ 
       $\lambda p . \langle x^P, \lambda x. p \rangle$ )
    subgoal using PossWorldExpanded[conj1, conj1, THEN oa-facts-2[deduction]]
      using Semantics.T6 apply cut-tac by PLM-solver
    using PossWorldExpanded[conj2] .

    hence  $\exists v. \forall p. ([\langle x^P, \lambda x. p \rangle \text{ in } v])$ 
       $= [p \text{ in } v]$ 
    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:
       $\forall p. ([\langle x^P, \lambda x. p \rangle \text{ in } v]) = [p \text{ in } v]$ 
    by auto
    {
      fix  $p$ 
      {
        assume  $[(x^P \models p) \text{ in } w]$ 
        hence  $[(x^P \models p) \text{ in } v]$ 
        using TrueInWorldNecc[equiv-lr] Semantics.T6 by simp
        hence  $[Situation (x^P) \ \& \ (\langle A!, x^P \rangle) \ \& \ \langle x^P, \lambda x. p \rangle \text{ in } v]$ 
        unfolding TrueInSituation-def EncodeProposition-def .
        hence  $[\langle x^P, \lambda x. p \rangle \text{ in } v]$ 
        using  $\&E(2)$  by blast
        hence  $[p \text{ in } v]$ 
        using PropsTrueInSemWorld by blast
      }
    }
    moreover {
      assume  $[p \text{ in } v]$ 
      hence  $[\langle x^P, \lambda x. p \rangle \text{ in } v]$ 
      using PropsTrueInSemWorld by blast
      hence  $[(x^P \models p) \text{ in } v]$ 
      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  $\Box((x^P) \models p)$  in  $v$ 
      using TrueInWorldNec[equiv-lr] by simp
    hence  $[(x^P) \models p]$  in  $w$ 
      using Semantics.T6 by simp
  }
  ultimately have  $[p \text{ in } v] \longleftrightarrow [(x^P) \models p \text{ in } w]$ 
    by auto
  }
  hence  $(\exists v . \forall p . [p \text{ in } v] \longleftrightarrow [(x^P) \models p \text{ in } w])$ 
    by blast
}
thus  $[PossibleWorld (x^P) \text{ in } w] \longrightarrow$ 
   $(\exists v . \forall p . [(x^P) \models p \text{ in } w] \longleftrightarrow [p \text{ in } v])$ 
  by blast
qed

```

A.10.4. For every semantic Possible World there is a syntactic Possible World

theorem *SyntacticPossibleWorldForSemanticPossibleWorlds:*

```

 $\forall v . \exists x . [PossibleWorld (x^P) \text{ in } w] \wedge$ 
 $(\forall p . [p \text{ in } v] \longleftrightarrow [(x^P) \models p \text{ in } w])$ 
proof
  fix v
  have  $[\exists x . (\lambda A!, x^P) \ \& \ (\forall F . (\lambda x^P, F \models$ 
     $(\exists p . p \ \& \ (F = (\lambda x . p)))) \text{ in } v]$ 
    using A-objects[axiom-instance] by fast
  then obtain x where DefX:
     $[(\lambda A!, x^P) \ \& \ (\forall F . (\lambda x^P, F \models (\exists p . p \ \& \ (F = (\lambda x . p)))) \text{ in } v]$ 
    by (rule  $\exists E$ )
  hence PossWorldX:  $[PossibleWorld (x^P) \text{ in } v]$ 
    using PossWorldAux[deduction] by blast
  hence  $[PossibleWorld (x^P) \text{ in } w]$ 
    using possworld-nec[equiv-lr] Semantics.T6 by auto
  moreover have  $(\forall p . [p \text{ in } v] \longleftrightarrow [(x^P) \models p \text{ in } w])$ 
proof
  fix q
  {
    assume  $[q \text{ in } v]$ 
    moreover have  $[(\lambda x . q) = (\lambda x . q) \text{ in } v]$ 
      using id-eq-prop-prop-1 by auto
    ultimately have  $[q \ \& \ (\lambda x . q) = (\lambda x . q) \text{ in } v]$ 
      using &I by auto
    hence  $[(\exists p . p \ \& \ ((\lambda x . q) = (\lambda x . p))) \text{ in } v]$ 
      by PLM-solver
    hence  $\lambda: [\lambda x^P, (\lambda x . q)] \text{ in } v$ 
      using cqt-1[axiom-instance, deduction, OF DefX[conj2], equiv-rl]
      by blast
    have  $[(x^P) \models q] \text{ in } v$ 
      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  $[(x^P \models q) \text{ in } w]$ 
    using TrueInWorldNecc[equiv-lr] Semantics.T6 by auto
  }
  moreover {
    assume  $[(x^P \models q) \text{ in } w]$ 
    hence  $[(x^P \models q) \text{ in } v]$ 
    using TrueInWorldNecc[equiv-lr] Semantics.T6
    by auto
    hence  $[\llbracket x^P, (\lambda x . q) \rrbracket \text{ in } v]$ 
    unfolding TrueInSituation-def EncodeProposition-def
    using  $\&E(2)$  by blast
    hence  $[(\exists p . p \ \& \ ((\lambda x . q) = (\lambda x . p))) \text{ in } v]$ 
    using cqt-1[axiom-instance, deduction, OF DefX[conj2], equiv-lr]
    by blast
    then obtain  $p$  where 4:
       $[(p \ \& \ ((\lambda x . q) = (\lambda x . p))) \text{ in } v]$ 
    by (rule  $\exists E$ )
    have  $[\llbracket (\lambda x . p), x^P \rrbracket \equiv p \text{ in } v]$ 
    apply (rule beta-C-meta-1)
    by show-proper
    hence  $[\llbracket (\lambda x . q), x^P \rrbracket \equiv p \text{ in } v]$ 
    using l-identity[where  $\beta = (\lambda x . q)$  and  $\alpha = (\lambda x . p)$ ,
      axiom-instance, deduction, deduction]
    using 4[conj2, THEN id-eq-prop-prop-2[deduction]] by meson
    hence  $[\llbracket (\lambda x . q), x^P \rrbracket \text{ in } v]$  using 4[conj1]  $\equiv E(2)$  by blast
    moreover have  $[\llbracket (\lambda x . q), x^P \rrbracket \equiv q \text{ in } v]$ 
    apply (rule beta-C-meta-1)
    by show-proper
    ultimately have  $[q \text{ in } v]$ 
    using  $\equiv E(1)$  by blast
  }
  ultimately show  $[q \text{ in } v] \longleftrightarrow [(x^P \models q) \text{ in } w]$ 
  by blast
qed
ultimately show  $\exists x . [PossibleWorld(x^P) \text{ in } w]$ 
   $\wedge (\forall p . [p \text{ in } v] \longleftrightarrow [(x^P \models p) \text{ in } w])$ 
  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*:
 $[\llbracket \lambda x . \llbracket x^P, F \rrbracket \equiv \llbracket x^P, F \rrbracket, y^P \rrbracket \text{ in } v]$
 by (*auto simp: meta-defs meta-aux conn-defs forall- Π_1 -def*)

lemma *lambda-enc-2*:
 $[\llbracket \lambda x . \llbracket y^P, G \rrbracket, x^P \rrbracket \equiv \llbracket y^P, G \rrbracket \text{ in } v]$
 by (*auto simp: meta-defs meta-aux conn-defs forall- Π_1 -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:
  [⟦(λ x . ⟨xP, F⟩), xP⟧ → ⟨xP, F⟩⟧ in v]
  apply (simp add: meta-defs meta-aux conn-defs forall-Π1-def)
  nitpick[user-axioms, expect=genuine]
  oops — countermodel by nitpick
```

Remark. *Instead the following two statements hold.*

```
lemma lambda-enc-4:
  [⟦(λ x . ⟨xP, F⟩), xP⟧ in v] = (∃ y . νv y = νv x ∧ [⟦yP, F⟧ in v])
  by (simp add: meta-defs meta-aux)
```

```
lemma lambda-ex:
  [⟦(λ x . φ (xP)), xP⟧ in v] = (∃ y . νv y = νv x ∧ [φ (yP) 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:
  [⟦(λ x . φ (xP)), xP⟧ ≡ (∃ y . (∀ F . ⟨F, xP⟩ ≡ ⟨F, yP⟩) & φ (yP)) in v]
  proof(rule MetaSolver.EquivI)
    interpret MetaSolver .
    {
      assume [⟦(λ x . φ (xP)), xP⟧ in v]
      then obtain y where νv y = νv x ∧ [φ (yP) in v]
        using lambda-ex by blast
      moreover hence [(∀ F . ⟨F, xP⟩ ≡ ⟨F, yP⟩) in v]
        apply — apply meta-solver
        by (simp add: Semantics.dκ-proper Semantics.ex1-def)
      ultimately have [∃ y . (∀ F . ⟨F, xP⟩ ≡ ⟨F, yP⟩) & φ (yP) in v]
        using ExIRule ConjI by fast
    }
    moreover {
      assume [∃ y . (∀ F . ⟨F, xP⟩ ≡ ⟨F, yP⟩) & φ (yP) in v]
      then obtain y where y-def: [(∀ F . ⟨F, xP⟩ ≡ ⟨F, yP⟩) & φ (yP) in v]
        by (rule ExERule)
      hence ∧ F . [⟦F, xP⟧ in v] = [⟦F, yP⟧ in v]
        apply — apply (drule ConjE) apply (drule conjunct1)
        apply (drule AllE) apply (drule EquivE) by simp
      hence [⟦makeΠ1 (λ u s w . νv y = u), xP⟧ in v]
        = [⟦makeΠ1 (λ u s w . νv y = u), yP⟧ in v] by auto
      hence νv y = νv x by (simp add: meta-defs meta-aux)
      moreover have [φ (yP) in v] using y-def ConjE by blast
      ultimately have [⟦(λ x . φ (xP)), xP⟧ in v]
        using lambda-ex by blast
    }
  }
  ultimately show [⟦λx. φ (xP), xP⟧ in v]
    = [∃ y. (∀ F. ⟨F, xP⟩ ≡ ⟨F, yP⟩) & φ (yP) in v]
    by auto
qed
```

```
lemma lambda-enc-emb:
  [⟦(λ x . ⟨xP, F⟩), xP⟧ ≡ (∃ y . (∀ F . ⟨F, xP⟩ ≡ ⟨F, yP⟩) & ⟨yP, 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  $\varphi$ 
  shows  $[(\exists y. (\forall F. \langle F, x^P \rangle \equiv \langle F, y^P \rangle) \ \& \ \varphi(y^P)) \equiv \varphi(x^P) \text{ in } v]$ 
proof (rule MetaSolver.EquivI; rule)
  interpret MetaSolver .
  assume  $[\exists y. (\forall F. \langle F, x^P \rangle \equiv \langle F, y^P \rangle) \ \& \ \varphi(y^P) \text{ in } v]$ 
  then obtain y where y-def:  $[(\forall F. \langle F, x^P \rangle \equiv \langle F, y^P \rangle) \ \& \ \varphi(y^P) \text{ in } v]$  by (rule ExERule)
  hence  $[\langle \text{make}\Pi_1 (\lambda u s w. \nu\nu y = u), x^P \rangle \text{ in } v] = [\langle \text{make}\Pi_1 (\lambda u s w. \nu\nu y = u), y^P \rangle \text{ in } v]$ 
    using EquivS Alle ConjE by blast
  hence  $\nu\nu y = \nu\nu x$  by (simp add: meta-defs meta-aux)
  thus  $[\varphi(x^P) \text{ in } v]$ 
    using y-def[THEN ConjE[THEN conjunct2]]
      assms IsProperInX.rep-eq valid-in.rep-eq
    by blast
next
  interpret MetaSolver .
  assume  $[\varphi(x^P) \text{ in } v]$ 
  moreover have  $[\forall F. \langle F, x^P \rangle \equiv \langle F, x^P \rangle \text{ in } v]$  apply meta-solver by blast
  ultimately show  $[\exists y. (\forall F. \langle F, x^P \rangle \equiv \langle F, y^P \rangle) \ \& \ \varphi(y^P) \text{ in } v]$ 
    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  $[\forall F. \langle F, a^P \rangle \equiv \langle F, b^P \rangle \text{ in } v]$ 
  shows  $(\lambda x. \langle R, x^P, a^P \rangle) = (\lambda x. \langle R, x^P, b^P \rangle)$ 
proof -
  interpret MetaSolver .
  obtain F where F-def:  $F = \text{make}\Pi_1 (\lambda u s w. u = \nu\nu a)$  by auto
  have  $[\langle F, a^P \rangle \equiv \langle F, b^P \rangle \text{ in } v]$  using assms by (rule Alle)
  moreover have  $[\langle F, a^P \rangle \text{ in } v]$ 
    unfolding F-def by (simp add: meta-defs meta-aux)
  ultimately have  $[\langle F, b^P \rangle \text{ in } v]$  using EquivE by auto
  hence  $\nu\nu a = \nu\nu b$  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 [(\lambda y. (q \vee \neg q)) = (\lambda y. (p \vee \neg p)) in v]
  unfolding identity-\Pi_1-def conn-defs
  apply (rule Eq_1I) apply (simp add: meta-defs)
  nitpick[expect = genuine, user-axioms=true, card i = 2,
    card j = 2, card \omega = 1, card \sigma = 1,
    sat-solver = MiniSat-JNI, verbose, show-all]
  oops — Countermodel by Nitpick
lemma [(\lambda y. (p \vee q)) = (\lambda y. (q \vee p)) in v]
  unfolding identity-\Pi_1-def
  apply (rule Eq_1I) apply (simp add: meta-defs)
  nitpick[expect = genuine, user-axioms=true,
    sat-solver = MiniSat-JNI, card i = 2,
    card j = 2, card \sigma = 1, card \omega = 1,
    card v = 2, verbose, show-all]
  oops — Countermodel by Nitpick
```

A.12.3. Concreteness coindices with Object Domains

```
lemma OrdCheck:
  [(\lambda x . \neg \Box(\neg(E!, x^P)), x) in v] \longleftrightarrow
  (proper x) \wedge (case (rep x) of \omega\nu y \Rightarrow True | - \Rightarrow False)
  using OrdinaryObjectsPossiblyConcreteAxiom
  apply (simp add: meta-defs meta-aux split: \nu.split v.split)
  using \nu\nu-\omega\nu-is-\omega\nu by fastforce
lemma AbsCheck:
  [(\lambda x . \Box(\neg(E!, x^P)), x) in v] \longleftrightarrow
  (proper x) \wedge (case (rep x) of \alpha\nu y \Rightarrow True | - \Rightarrow False)
  using OrdinaryObjectsPossiblyConcreteAxiom
  apply (simp add: meta-defs meta-aux split: \nu.split v.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. ((\lambda x . \neg \Box(\neg(E!, x^P)), x^P) in v)
    \longleftrightarrow (case x of \omega\nu y \Rightarrow True | - \Rightarrow False)))
  unfolding Concrete-def
  apply (simp add: meta-defs meta-aux split: \nu.split v.split)
  using \nu\nu-\omega\nu-is-\omega\nu by fastforce
```

Remark. *OrdinaryObjectsPossiblyConcreteAxiom* is equivalent to "all abstract objects are necessarily not concrete".

lemma *AbsAxiomCheck*:
 $\text{OrdinaryObjectsPossiblyConcrete} \longleftrightarrow$
 $(\forall x. ([\lambda x. \Box(\neg(E!, x^P)), x^P] \text{ in } v)$
 $\longleftrightarrow (\text{case } x \text{ of } \alpha\nu y \Rightarrow \text{True} \mid - \Rightarrow \text{False})))$
apply (*simp add: meta-defs meta-aux split: $\nu.\text{split}$ $v.\text{split}$*)
using $\nu\nu\text{-}\omega\nu\text{-is-}\omega\nu$ $\text{no-}\alpha\omega$ **by** *fastforce*

Remark. *PossiblyContingentObjectExistsAxiom* is equivalent to the corresponding statement in the embedded logic.

lemma *PossiblyContingentObjectExistsCheck*:
 $\text{PossiblyContingentObjectExists} \longleftrightarrow [\neg(\Box(\forall x. (E!, x^P) \rightarrow \Box(E!, x^P))) \text{ in } v]$
apply (*simp add: meta-defs forall- ν -def meta-aux split: $\nu.\text{split}$ $v.\text{split}$*)
by (*metis $\nu.\text{simsps}(5)$ $\nu\nu\text{-def}$ $v.\text{simsps}(1)$ $\text{no-}\sigma\omega$ $v.\text{exhaust}$*)

Remark. *PossiblyNoContingentObjectExistsAxiom* is equivalent to the corresponding statement in the embedded logic.

lemma *PossiblyNoContingentObjectExistsCheck*:
 $\text{PossiblyNoContingentObjectExists} \longleftrightarrow [\neg(\Box(\neg(\forall x. (E!, x^P) \rightarrow \Box(E!, x^P)))) \text{ in } v]$
apply (*simp add: meta-defs forall- ν -def meta-aux split: $\nu.\text{split}$ $v.\text{split}$*)
using $\nu\nu\text{-}\omega\nu\text{-is-}\omega\nu$ **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*:
 $[\forall x. \Box((F, x^P) \equiv (G, x^P)) \text{ in } v] \longleftrightarrow$
 $((\lambda x. (\text{eval}\Pi_1 F) x dj) = (\lambda x. (\text{eval}\Pi_1 G) x dj))$
proof
assume $1: [\forall x. \Box((F, x^P) \equiv (G, x^P)) \text{ in } v]$
 $\{$
fix v
fix y
obtain x **where** $y\text{-def}: y = \nu\nu x$
by (*meson $\nu\nu\text{-surj}$ surj-def*)
have $(\exists r o_1. \text{Some } r = d_1 F \wedge \text{Some } o_1 = d_\kappa(x^P) \wedge o_1 \in \text{ex1 } r v) =$
 $(\exists r o_1. \text{Some } r = d_1 G \wedge \text{Some } o_1 = d_\kappa(x^P) \wedge o_1 \in \text{ex1 } r v)$
using 1 **apply** $-$ **by** *meta-solver*
moreover obtain r **where** $r\text{-def}: \text{Some } r = d_1 F$
unfolding $d_1\text{-def}$ **by** *auto*
moreover obtain s **where** $s\text{-def}: \text{Some } s = d_1 G$
unfolding $d_1\text{-def}$ **by** *auto*
moreover have $\text{Some } x = d_\kappa(x^P)$
using $d_\kappa\text{-proper}$ **by** *simp*
ultimately have $(x \in \text{ex1 } r v) = (x \in \text{ex1 } s v)$
by (*metis option.inject*)
hence $(\text{eval}\Pi_1 F) y dj v = (\text{eval}\Pi_1 G) y dj v$
using $r\text{-def}$ $s\text{-def}$ $y\text{-def}$ **by** (*simp add: $d_1.\text{rep-eq}$ ex1-def*)
 $\}$
thus $(\lambda x. \text{eval}\Pi_1 F x dj) = (\lambda x. \text{eval}\Pi_1 G x dj)$
by *auto*
next
assume $1: (\lambda x. \text{eval}\Pi_1 F x dj) = (\lambda x. \text{eval}\Pi_1 G x dj)$

```

{
  fix y v
  obtain x where x-def: x = νv y
    by simp
  hence evalΠ1 F x dj = evalΠ1 G x dj
    using 1 by metis
  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
  ultimately have (y ∈ ex1 r v) = (y ∈ ex1 s v)
    by (simp add: d1.rep-eq ex1-def νv-surj x-def)
  hence [(F, yP) ≡ (G, yP) in v]
    apply – apply meta-solver
    using r-def s-def by (metis Semantics.dκ-proper option.inject)
}
thus [∀ x. □((F, xP) ≡ (G, xP) 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 [∀ x . □((F, xP) ≡ (G, xP) in v)]
  shows [F = G in v] ⟷
    (∀ s . s ≠ dj ⟶ (∀ x . (evalΠ1 F) x s = (evalΠ1 G) x s))
proof
  interpret MetaSolver .
  assume [F = G in v]
  hence F = G
    apply – unfolding identity-Π1-def by meta-solver
  thus ∀ s. s ≠ dj ⟶ (∀ x. evalΠ1 F x s = evalΠ1 G x s)
    by auto
next
  interpret MetaSolver .
  assume ∀ s. s ≠ dj ⟶ (∀ x. evalΠ1 F x s = evalΠ1 G x s)
  moreover have ((λ x . (evalΠ1 F) x dj) = (λ x . (evalΠ1 G) x dj))
    using assms mat-eq-is-eq-dj by auto
  ultimately have ∀ s x. evalΠ1 F x s = evalΠ1 G x s
    by metis
  hence evalΠ1 F = evalΠ1 G
    by blast
  hence F = G
    by (metis evalΠ1-inverse)
  thus [F = G in v]
    unfolding identity-Π1-def using Eq1I 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 ∀ F x s . (evalΠ1 F) x s = (evalΠ1 F) x dj
  shows [∀ x . □((F, xP) ≡ (G, xP) in v)] ⟷ [F = G in v]
  by (metis (no-types) MetaSolver.Eq1S assms identity-Π1-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 (λx. (R, xP, a)) = (λx . (R, xP, b))
proof -
  have a = b
    using MetaSolver.EqκS Semantics.dκ-inject assms
      identity-κ-def by auto
  thus ?thesis by simp
qed

```

```

lemma lambda-interpret-2:
  assumes [a = (λy. (G, yP)) in v]
  shows (λx. (R, xP, a)) = (λx . (R, xP, λy. (G, yP)))
proof -
  have a = (λy. (G, yP))
    using MetaSolver.EqκS Semantics.dκ-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 $\lambda x. (G, \lambda y. \varphi y x)$ for arbitrary φ are *proper maps*, for which β -conversion holds, the theory becomes inconsistent.

A.13.1. Auxiliary Lemmata

```

lemma exe-impl-exists:
  [(λx . ∀ p . p → p), λy . φ y x] ≡ (∃!y . Aφ y x) in v]
proof (rule ≡I; rule CP)
  fix φ :: ν ⇒ ν ⇒ o and x :: ν and v :: i
  assume [(λx . ∀ p . p → p), λy . φ y x] in v]
  hence [∃ y. Aφ y x & (∀ z. Aφ z x → z = y)
    & [(λx . ∀ p . p → p), yP] in v]
    using nec-russell-axiom[equiv-lr] SimpleExOrEnc.intros by auto
  then obtain y where
    [Aφ y x & (∀ z. Aφ z x → z = y)
      & [(λx . ∀ p . p → p), yP] in v]
    by (rule Instantiate)
  hence [Aφ y x & (∀ z. Aφ z x → z = y) in v]
    using &E by blast
  hence [∃ y . Aφ y x & (∀ z. Aφ z x → z = y) in v]
    by (rule existential)
  thus [∃!y. Aφ y x in v]
    unfolding exists-unique-def by simp
next
  fix φ :: ν ⇒ ν ⇒ o and x :: ν and v :: i
  assume [∃!y. Aφ y x in v]
  hence [∃ y. Aφ y x & (∀ z. Aφ z x → z = y) in v]

```


unfolding *exists-unique-def* by *simp*
 then obtain y where
 $[\mathcal{A}_\varphi y x \ \& \ (\forall z. \mathcal{A}_\varphi z x \rightarrow z = y) \text{ in } v]$
 by (rule *Instantiate*)
 moreover have $[(\lambda x. \forall p. p \rightarrow p), y^P] \text{ in } v]$
 apply (rule *beta-C-meta-1*[*equiv-rl*])
 apply *show-proper*
 by *PLM-solver*
 ultimately have $[\mathcal{A}_\varphi y x \ \& \ (\forall z. \mathcal{A}_\varphi z x \rightarrow z = y)$
 $\ \& \ [(\lambda x. \forall p. p \rightarrow p), y^P] \text{ in } v]$
 using *&I* by *blast*
 hence $[\exists y. \mathcal{A}_\varphi y x \ \& \ (\forall z. \mathcal{A}_\varphi z x \rightarrow z = y)$
 $\ \& \ [(\lambda x. \forall p. p \rightarrow p), y^P] \text{ in } v]$
 by (rule *existential*)
 thus $[(\lambda x. \forall p. p \rightarrow p), \iota y. \varphi y x] \text{ in } v]$
 using *nec-russell-axiom*[*equiv-rl*]
SimpleExOrEnc.intros by *auto*
 qed

lemma *exists-unique-actual-equiv*:
 $[(\exists !y. \mathcal{A}(y = x \ \& \ \psi(x^P))) \equiv \mathcal{A}\psi(x^P) \text{ in } v]$
proof (rule *$\equiv I$* ; rule *CP*)
 fix $x v$
 let $? \varphi = \lambda y x. y = x \ \& \ \psi(x^P)$
 assume $[\exists !y. \mathcal{A}? \varphi y x \text{ in } v]$
 hence $[\exists \alpha. \mathcal{A}? \varphi \alpha x \ \& \ (\forall \beta. \mathcal{A}? \varphi \beta x \rightarrow \beta = \alpha) \text{ in } v]$
 unfolding *exists-unique-def* by *simp*
 then obtain α where
 $[\mathcal{A}? \varphi \alpha x \ \& \ (\forall \beta. \mathcal{A}? \varphi \beta x \rightarrow \beta = \alpha) \text{ in } v]$
 by (rule *Instantiate*)
 hence $[\mathcal{A}(\alpha = x \ \& \ \psi(x^P)) \text{ in } v]$
 using *&E* by *blast*
 thus $[\mathcal{A}(\psi(x^P)) \text{ in } v]$
 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: $[\mathcal{A}\psi(x^P) \text{ in } v]$
 have $[x = x \text{ in } v]$
 using *id-eq-1*[*where 'a=v*] by *simp*
 hence $[\mathcal{A}(x = x) \text{ in } v]$
 using *id-act-3*[*equiv-lr*] by *fast*
 hence $[\mathcal{A}(x = x \ \& \ \psi(x^P)) \text{ in } v]$
 using 1 *Act-Basic-2*[*equiv-rl*] *&I* by *blast*
 hence $[\mathcal{A}? \varphi x x \text{ in } v]$
 by *simp*
 moreover have $[\forall \beta. \mathcal{A}? \varphi \beta x \rightarrow \beta = x \text{ in } v]$
proof (rule *$\forall I$* ; rule *CP*)
 fix β
 assume $[\mathcal{A}? \varphi \beta x \text{ in } v]$
 hence $[\mathcal{A}(\beta = x) \text{ in } v]$
 using *Act-Basic-2*[*equiv-lr*] *&E* by *fast*
 thus $[\beta = x \text{ in } v]$ using *id-act-3*[*equiv-rl*] by *fast*
 qed
 ultimately have $[\mathcal{A}? \varphi x x \ \& \ (\forall \beta. \mathcal{A}? \varphi \beta x \rightarrow \beta = x) \text{ in } v]$
 using *&I* by *fast*
 hence $[\exists \alpha. \mathcal{A}? \varphi \alpha x \ \& \ (\forall \beta. \mathcal{A}? \varphi \beta x \rightarrow \beta = \alpha) \text{ in } v]$
 by (rule *existential*)

thus $[\exists !y. \mathcal{A}^? \varphi y x \text{ in } v]$
 unfolding *exists-unique-def* by *simp*
 qed

A.13.2. Fake β -Conversion using Description Backdoor

definition *backdoor* where

$backdoor \equiv \lambda \psi . \lambda x . \langle (\lambda x . \forall p . p \rightarrow p), \iota y . y = x \ \& \ \psi (x^P) \rangle$

lemma *fake-beta*:

assumes $\bigwedge G \varphi. IsProperInX (\lambda x . \langle G, \iota y . \varphi y x \rangle)$
shows $\langle backdoor (\lambda x . \psi x), x^P \rangle \equiv \mathcal{A} \psi (x^P) \text{ in } v$

proof (rule $\equiv I$; rule *CP*)

assume $\langle backdoor \psi, x^P \rangle \text{ in } v$

hence $\langle \lambda x . \forall p . p \rightarrow p, \iota y . y = x \ \& \ \psi (x^P) \rangle \text{ in } v$

using *beta-C-meta-1*[*equiv-lr*, *OF assms*]

unfolding *backdoor-def identity- ν -def* by *fast*

hence $[\exists !y. \mathcal{A} (y = x \ \& \ \psi (x^P)) \text{ in } v]$

using *exe-impl-exists*[*equiv-lr*] by *fast*

thus $[\mathcal{A} \psi (x^P) \text{ in } v]$

using *exists-unique-actual-equiv*[*equiv-lr*] by *blast*

next

assume $[\mathcal{A} \psi (x^P) \text{ in } v]$

hence $[\exists !y. \mathcal{A} (y = x \ \& \ \psi (x^P)) \text{ in } v]$

using *exists-unique-actual-equiv*[*equiv-rl*] by *blast*

hence $\langle \lambda x . \forall p . p \rightarrow p, \iota y . y = x \ \& \ \psi (x^P) \rangle \text{ in } v$

using *exe-impl-exists*[*equiv-rl*] by *fast*

thus $\langle backdoor \psi, x^P \rangle \text{ in } v$

using *beta-C-meta-1*[*equiv-rl*, *OF assms*]

unfolding *backdoor-def* **unfolding** *identity- ν -def* by *fast*

qed

lemma *fake-beta-act*:

assumes $\bigwedge G \varphi. IsProperInX (\lambda x . \langle G, \iota y . \varphi y x \rangle)$

shows $\langle backdoor (\lambda x . \psi x), x^P \rangle \equiv \psi (x^P) \text{ in } dw$

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 $\bigwedge G \varphi. IsProperInX (\lambda x . \langle G, \iota y . \varphi y x \rangle)$

shows *False*

proof –

obtain *K* where *K-def*:

$K = backdoor (\lambda x . \exists F . \langle x, F \rangle \ \& \ \neg \langle F, x \rangle)$ by *auto*

have $[\exists x . \langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv (F = K)) \text{ in } dw]$

using *A-objects*[*axiom-instance*] by *fast*

then obtain *x* where *x-prop*:

$[\langle A!, x^P \rangle \ \& \ (\forall F . \langle x^P, F \rangle \equiv (F = K)) \text{ in } dw]$

by (rule *Instantiate*)

{

assume $[\langle K, x^P \rangle \text{ in } dw]$

hence $[\exists F . \langle x^P, F \rangle \ \& \ \neg \langle F, x^P \rangle \text{ in } dw]$

unfolding *K-def* **using** *fake-beta-act*[*OF assms*, *equiv-lr*]

by *blast*

then obtain *F* where *F-def*:

```

    [ $\{x^P, F\} \& \neg(F, x^P)$ ] in dw] by (rule Instantiate)
  hence  $[F = K$  in dw]
    using  $x\text{-prop}[conj2, THEN \forall E[\text{where } \beta=F], equiv\text{-}lr]$ 
    &E unfolding  $K\text{-def}$  by blast
  hence  $[\neg(F, x^P)]$  in dw]
    using  $l\text{-identity}[axiom\text{-}instance, deduction, deduction]$ 
     $F\text{-def}[conj2]$  by fast
}
hence 1:  $[\neg(F, x^P)]$  in dw]
  using  $reductio\text{-}aa\text{-}1$  by blast
hence  $[\neg(\exists F . \{x^P, F\} \& \neg(F, x^P))]$  in dw]
  using  $fake\text{-}beta\text{-}act[OF\ assms,$ 
     $THEN oth\text{-}class\text{-}taut\text{-}5\text{-}d[equiv\text{-}lr],$ 
     $equiv\text{-}lr]$ 
  unfolding  $K\text{-def}$  by blast
hence  $[\forall F . \{x^P, F\} \rightarrow (F, x^P)]$  in dw]
  apply – unfolding  $exists\text{-}def$  by PLM-solver
moreover have  $[\{x^P, K\}]$  in dw]
  using  $x\text{-prop}[conj2, THEN \forall E[\text{where } \beta=K], equiv\text{-}rl]$ 
   $id\text{-}eq\text{-}1$  by blast
ultimately have  $[(K, x^P)]$  in dw]
  using  $\forall E\ vdash\text{-}properties\text{-}10$  by blast
hence  $\bigwedge \varphi. [\varphi]$  in dw]
  using  $raa\text{-}cor\text{-}2\ 1$  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  $\bigwedge G \varphi. IsProperInX (\lambda x . (G, \iota y . \varphi y x))$ 
shows  $Fx\text{-}equiv\text{-}xH: [\forall H . \exists F . \Box(\forall x. (F, x^P) \equiv \{x^P, H\})]$  in v]
proof (rule  $\forall I$ )
  fix H
  let ?G =  $(\lambda x . \forall p . p \rightarrow p)$ 
  obtain  $\varphi$  where  $\varphi\text{-}def: \varphi = (\lambda y x . (y^P) = x \& \{x, H\})$  by auto
  have  $[\exists F. \Box(\forall x. (F, x^P) \equiv (\varphi G, \iota y . \varphi y (x^P)))]$  in v]
    using  $relations\text{-}1[OF\ assms]$  by simp
  hence 1:  $[\exists F. \Box(\forall x. (F, x^P) \equiv (\exists! y . \mathcal{A}\varphi y (x^P)))]$  in v]
    apply – apply (PLM-subst-method
       $\lambda x . (\varphi G, \iota y . \varphi y (x^P)) \lambda x . (\exists! y. \mathcal{A}\varphi y (x^P))$ )
    using  $exe\text{-}impl\text{-}exists$  by auto
  then obtain F where  $F\text{-}def: [\Box(\forall x. (F, x^P) \equiv (\exists! y . \mathcal{A}\varphi y (x^P)))]$  in v]
    by (rule Instantiate)
  moreover have 2:  $\bigwedge v x . [(\exists! y . \mathcal{A}\varphi y (x^P)) \equiv \{x^P, H\}]$  in v]
proof (rule  $\equiv I$ ; rule CP)
  fix x v
  assume  $[\exists! y. \mathcal{A}\varphi y (x^P)]$  in v]
  hence  $[\exists \alpha. \mathcal{A}\varphi \alpha (x^P) \& (\forall \beta. \mathcal{A}\varphi \beta (x^P) \rightarrow \beta = \alpha)]$  in v]
    unfolding  $exists\text{-}unique\text{-}def$  by simp
  then obtain  $\alpha$  where  $[\mathcal{A}\varphi \alpha (x^P) \& (\forall \beta. \mathcal{A}\varphi \beta (x^P) \rightarrow \beta = \alpha)]$  in v]
    by (rule Instantiate)
  hence  $[\mathcal{A}(\alpha^P = x^P \& \{x^P, H\})]$  in v]
    unfolding  $\varphi\text{-}def$  using &E by blast
  hence  $[\mathcal{A}(\{x^P, H\})]$  in v]

```

```

    using Act-Basic-2[equiv-lr] &E by blast
  thus [ $\llbracket x^P, H \rrbracket$  in  $v$ ]
    using en-eq-10[equiv-lr] by simp
next
fix  $x v$ 
assume [ $\llbracket x^P, H \rrbracket$  in  $v$ ]
hence 1: [ $\mathcal{A}(\llbracket x^P, H \rrbracket)$  in  $v$ ]
  using en-eq-10[equiv-rl] by blast
have [ $x = x$  in  $v$ ]
  using id-eq-1[where 'a= $\nu$ ] by simp
hence [ $\mathcal{A}(x = x)$  in  $v$ ]
  using id-act-3[equiv-lr] by fast
hence [ $\mathcal{A}(x^P = x^P \ \& \ \llbracket x^P, H \rrbracket)$  in  $v$ ]
  unfolding identity- $\nu$ -def using 1 Act-Basic-2[equiv-rl] &I by blast
hence [ $\mathcal{A}\varphi x (x^P)$  in  $v$ ]
  unfolding  $\varphi$ -def by simp
moreover have [ $\forall \beta. \mathcal{A}\varphi \beta (x^P) \rightarrow \beta = x$  in  $v$ ]
proof (rule  $\forall I$ ; rule CP)
  fix  $\beta$ 
  assume [ $\mathcal{A}\varphi \beta (x^P)$  in  $v$ ]
  hence [ $\mathcal{A}(\beta = x)$  in  $v$ ]
    unfolding  $\varphi$ -def identity- $\nu$ -def
    using Act-Basic-2[equiv-lr] &E by fast
  thus [ $\beta = x$  in  $v$ ] using id-act-3[equiv-rl] by fast
qed
ultimately have [ $\mathcal{A}\varphi x (x^P) \ \& \ (\forall \beta. \mathcal{A}\varphi \beta (x^P) \rightarrow \beta = x)$  in  $v$ ]
  using &I by fast
hence [ $\exists \alpha. \mathcal{A}\varphi \alpha (x^P) \ \& \ (\forall \beta. \mathcal{A}\varphi \beta (x^P) \rightarrow \beta = \alpha)$  in  $v$ ]
  by (rule existential)
thus [ $\exists !y. \mathcal{A}\varphi y (x^P)$  in  $v$ ]
  unfolding exists-unique-def by simp
qed
have [ $\Box(\forall x. \llbracket F, x^P \rrbracket \equiv \llbracket x^P, H \rrbracket)$  in  $v$ ]
  apply (PLM-subst-goal-method
     $\lambda \varphi. \Box(\forall x. \llbracket F, x^P \rrbracket \equiv \varphi x)$ 
     $\lambda x. (\exists !y. \mathcal{A}\varphi y (x^P))$ )
  using 2 F-def by auto
thus [ $\exists F. \Box(\forall x. \llbracket F, x^P \rrbracket \equiv \llbracket x^P, H \rrbracket)$  in  $v$ ]
  by (rule existential)
qed

```

lemma

```

assumes is-propositional: ( $\bigwedge G \varphi. \text{IsProperInX } (\lambda x. \llbracket G, \iota y. \varphi y x \rrbracket)$ )
  and Abs-x: [ $\llbracket A!, x^P \rrbracket$  in  $v$ ]
  and Abs-y: [ $\llbracket A!, y^P \rrbracket$  in  $v$ ]
  and noteq: [ $x \neq y$  in  $v$ ]
shows diffprop: [ $\exists F. \neg(\llbracket F, x^P \rrbracket \equiv \llbracket F, y^P \rrbracket)$  in  $v$ ]
proof -
  have [ $\exists F. \neg(\llbracket x^P, F \rrbracket \equiv \llbracket y^P, F \rrbracket)$  in  $v$ ]
    using noteq unfolding exists-def
  proof (rule reductio-aa-2)
    assume 1: [ $\forall F. \neg(\llbracket x^P, F \rrbracket \equiv \llbracket y^P, F \rrbracket)$  in  $v$ ]
    {
      fix  $F$ 
      have [ $\llbracket x^P, F \rrbracket \equiv \llbracket y^P, F \rrbracket$  in  $v$ ]
        using 1[THEN  $\forall E$ ] useful-tautologies-1[deduction] by blast
    }
  }

```

hence $\forall F. \llbracket x^P, F \rrbracket \equiv \llbracket y^P, F \rrbracket$ in v by (rule $\forall I$)
 thus $[x = y$ in $v]$
 unfolding *identity- ν -def*
 using *ab-obey-1*[*deduction*, *deduction*]
 Abs-x Abs-y &I by *blast*
 qed
 then obtain H where $H\text{-def}$: $[\neg(\llbracket x^P, H \rrbracket \equiv \llbracket y^P, H \rrbracket)]$ in v
 by (rule *Instantiate*)
 hence 2: $[(\llbracket x^P, H \rrbracket \ \& \ \neg\llbracket y^P, H \rrbracket) \vee (\neg\llbracket x^P, H \rrbracket \ \& \ \llbracket y^P, H \rrbracket)]$ in v
 apply – by *PLM-solver*
 have $[\exists F. \Box(\forall x. \llbracket F, x^P \rrbracket \equiv \llbracket x^P, H \rrbracket)]$ in v
 using *Fx-equiv-xH*[*OF is-propositional*, *THEN $\forall E$*] by *simp*
 then obtain F where $[\Box(\forall x. \llbracket F, x^P \rrbracket \equiv \llbracket x^P, H \rrbracket)]$ in v
 by (rule *Instantiate*)
 hence $F\text{-prop}$: $[\forall x. \llbracket F, x^P \rrbracket \equiv \llbracket x^P, H \rrbracket]$ in v
 using *qml-2*[*axiom-instance*, *deduction*] by *blast*
 hence a : $[\llbracket F, x^P \rrbracket \equiv \llbracket x^P, H \rrbracket]$ in v
 using $\forall E$ by *blast*
 have b : $[\llbracket F, y^P \rrbracket \equiv \llbracket y^P, H \rrbracket]$ in v
 using $F\text{-prop}$ $\forall E$ by *blast*
 {
 assume 1: $[\llbracket x^P, H \rrbracket \ \& \ \neg\llbracket y^P, H \rrbracket]$ in v
 hence $[\llbracket F, x^P \rrbracket]$ in v
 using *a*[*equiv-rl*] & E by *blast*
 moreover have $[\neg\llbracket F, y^P \rrbracket]$ in v
 using *b*[*THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-rl*] 1[*conj2*] by *auto*
 ultimately have $[\llbracket F, x^P \rrbracket \ \& \ (\neg\llbracket F, y^P \rrbracket)]$ in v
 by (rule *&I*)
 hence $[(\llbracket F, x^P \rrbracket \ \& \ \neg\llbracket F, y^P \rrbracket) \vee (\neg\llbracket F, x^P \rrbracket \ \& \ \llbracket F, y^P \rrbracket)]$ in v
 using $\vee I$ by *blast*
 hence $[\neg(\llbracket F, x^P \rrbracket \equiv \llbracket F, y^P \rrbracket)]$ in v
 using *oth-class-taut-5-j*[*equiv-rl*] by *blast*
 }
 moreover {
 assume 1: $[\neg\llbracket x^P, H \rrbracket \ \& \ \llbracket y^P, H \rrbracket]$ in v
 hence $[\llbracket F, y^P \rrbracket]$ in v
 using *b*[*equiv-rl*] & E by *blast*
 moreover have $[\neg\llbracket F, x^P \rrbracket]$ in v
 using *a*[*THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-rl*] 1[*conj1*] by *auto*
 ultimately have $[\neg\llbracket F, x^P \rrbracket \ \& \ \llbracket F, y^P \rrbracket]$ in v
 using *&I* by *blast*
 hence $[(\llbracket F, x^P \rrbracket \ \& \ \neg\llbracket F, y^P \rrbracket) \vee (\neg\llbracket F, x^P \rrbracket \ \& \ \llbracket F, y^P \rrbracket)]$ in v
 using $\vee I$ by *blast*
 hence $[\neg(\llbracket F, x^P \rrbracket \equiv \llbracket F, y^P \rrbracket)]$ in v
 using *oth-class-taut-5-j*[*equiv-rl*] by *blast*
 }
 ultimately have $[\neg(\llbracket F, x^P \rrbracket \equiv \llbracket F, y^P \rrbracket)]$ in v
 using 2 *intro-elim-4-b* *reductio-aa-1* by *blast*
 thus $[\exists F. \neg(\llbracket F, x^P \rrbracket \equiv \llbracket F, y^P \rrbracket)]$ in v
 by (rule *existential*)
 qed

lemma *original-paradox*:

assumes *is-propositional*: $(\bigwedge G \ \varphi. \text{IsProperInX} \ (\lambda x. \llbracket G, \iota y. \varphi \ y \ x \rrbracket))$
 shows *False*

proof –

fix v

have $[\exists x \ y. \llbracket A!, x^P \rrbracket \ \& \ \llbracket A!, y^P \rrbracket \ \& \ x \neq y \ \& \ (\forall F. \llbracket F, x^P \rrbracket \equiv \llbracket F, y^P \rrbracket)]$ in v

```

    using aclassical2 by auto
  then obtain x where
    [ $\exists y. \langle A!, x^P \rangle \ \& \ \langle A!, y^P \rangle \ \& \ x \neq y \ \& \ (\forall F. \langle F, x^P \rangle \equiv \langle F, y^P \rangle)$  in v]
    by (rule Instantiate)
  then obtain y where xy-def:
    [ $\langle A!, x^P \rangle \ \& \ \langle A!, y^P \rangle \ \& \ x \neq y \ \& \ (\forall F. \langle F, x^P \rangle \equiv \langle F, y^P \rangle)$  in v]
    by (rule Instantiate)
  have [ $\exists F. \neg(\langle F, x^P \rangle \equiv \langle F, y^P \rangle)$  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 [ $\neg(\langle F, x^P \rangle \equiv \langle F, y^P \rangle)$  in v]
    by (rule Instantiate)
  moreover have [ $\langle F, x^P \rangle \equiv \langle F, y^P \rangle$  in v]
    using xy-def[conj2] by (rule  $\forall E$ )
  ultimately have  $\bigwedge \varphi. [\varphi$  in v]
    using PLM.raa-cor-2 by blast
  thus False
    using Semantics.T4 by auto
qed
end

```

Bibliography

- [1] C. Benz Müller. Universal reasoning, rational argumentation and human-machine interaction. *CoRR*, abs/1703.09620, 2017.
- [2] C. Benz Müller 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. Benz Müller and L. Paulson. Quantified multimodal logics in simple type theory. *Logica Universalis (Special Issue on Multimodal Logics)*, 7(1):7–20, 2013.
- [4] C. Benz Müller and D. S. Scott. Axiomatizing category theory in free logic. *CoRR*, abs/1609.01493, 2016.
- [5] C. Benz Müller 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: April 01, 2017].
- [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, 29. Mai 2017

Daniel Kirchner