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

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

Daniel Kirchner

Supervisor:
PD Dr. Christoph Benzmüller

Berlin, May 15, 2017

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[4]) 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's 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[4] employs a modal relational type theory as logical foundation for which a representation in functional type theory is known to be challenging[2].

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 ([4, 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 (TODO: reference).

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

Contents

1. Introduction

Calculemus!

Leibniz

1.1. Universal Logical Reasoning

TODO 1.1. *Add references throughout the section.*

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

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

Nowadays with the rise of powerful computer systems such a universal logical framework could have repercussions throughout the sciences (TODO: change this?) 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* (TODO: reference https://arxiv.org/abs/1703.09620).

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 (TODO: reference), for which higher-order modal logic was embedded in the computerized logic framework Isabelle/HOL.

1.2. Shallow Semantical Embeddings in HOL

TODO 1.2. *Think about terminology: background logic, target logic, embedded logic.*

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

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

While this approach will work for most target logics, it has several drawbacks. There are likely principles that are shared between the target logic and the background logic, such as alpha-conversion for lambda expressions or the equivalence of terms with renamed variables in general. In a deep embedding these principles 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 syntactical translation process 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 now 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 preverses a larger degree of automation. Furthermore axioms in the embedded logic can often be equivalently stated in the background logic, which makes the construction of models for the system easier and again increases the degree of automation

that can be retained.

The shallow semantical embedding of modal logic was the basis for the analysis of Gödel's onthological argument and has shown great potential as a universal tool for logical embeddings while retaining the existing infrastructure for automation as for example present in a framework like Isabelle/HOL. (TODO: more application examples)

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*([2]) 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 logical framework as 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 lambda 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 lambda expressions in the embedded logic, it could still be reproduced using an appropriate construction in the background logic.

The solution presented here circumvents this problem by identifying lambda expressions as one element of the target language that behaves differently than their counterparts in the background logic and consequently by representing lambda expressions of the target logic using a new *defined* kind of lambda expressions. This forces lambda 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 section 3.2.

1.4. Our Contribution

TODO 1.3. *Embedding of second order fragment of PLM. Complex semantics, term-based syntax, scope of the embedding, technical challenges. Kirchner's paradox.*

1.5. Overview of the following Chapters

TODO 1.4. *Improve and adjust.*

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 discusses the philosophical implications of the embedding and its relation to the target theory of PLM. Furthermore it describes the meta-logical results achieved using the embedding and states interesting open questions for future research.
- The last chapter consists of a technical discussion about some issues encountered during the construction of the embedding due to limitations of the logical framework of Isabelle/HOL and the solutions that were employed.

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

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*[3]

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."[5]

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 (TODO: reference, maybe again just [5]). 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 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 an object that *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 the concepts 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 [5] the principal objectives of the theory are summerized as follows:

- To describe the logic underlying (scientific) thought and reasoning by extending classical propositional, predicate, and modal logic.
- To describe the laws governing universal entities such as properties, relations, and propositions (i.e., states of affairs).
- To identify *theoretical* mathematical objects and relations as well as the *natural* mathematical objects such as natural numbers and natural sets.
- To analyze the distinction between fact and fiction and systematize the various relationships between stories, characters, and other fictional objects.
- To systematize our modal thoughts about possible (actual, necessary) objects, states of affairs, situations and worlds.
- To 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 to reason 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 serve as the basis for spreading the utilization of the theory itself as a foundational theory for various scientific fields by enabling convenient interactive and automated reasoning in a verified framework.

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

2.2. Basic Principles

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

The following are the two most important principles of the theory:

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

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

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

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

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

Other interesting examples include possible worlds, Platonic Forms or even basic logical objects like truth values. Here it is important to note that the theory allows it to

formulate a purely *syntactic* definition of objects like possible worlds and truth values and from these syntactic definitions it can be *derived* that there are two truth values or that the application of the modal box operator to a proposition is equivalent to the proposition being true in all possible worlds (where *being true in a possible world* is again defined syntactically).

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

2.3. The Language of PLM

The target of the embedding is the second order fragment of object theory as described in chapter 7 of Edward Zalta's upcoming *Principia Logico Metaphysica* (PLM) [4]. 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[4].

A compact description of the language can be given in Backus-Naur Form (BNF)[4, p. 170]. For this 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

The syntax of the target theory can now be described as BNF grammar[4, ibid.]:

	δ	::=	$a_1, a_2, \ldots$
	ν	::=	$x_1, x_2, \ldots$
$(n \geq 0)$	Σ^n	::=	$P^n_1, P^n_2, \ldots$
$(n \geq 0)$	Ω^n	::=	$F^n_1, F^n_2, \ldots$
	α	::=	$\nu \mid \Omega^n (n \geq 0)$
	κ	::=	$\delta \mid \nu \mid \iota\nu\phi$
$(n \geq 1)$	Π^n	::=	$\Sigma^n \mid \Omega^n \mid [\lambda \nu_1 \ldots \nu_n \ \phi^*]$
	Π^0	::=	$\Sigma^0 \mid \Omega^0 \mid [\lambda \ \phi^*] \mid \phi^*$
	ϕ^*	::=	$\Pi^n \kappa_1 \ldots \kappa_n (n \geq 1) \mid \Pi^0 \mid (\neg\phi^*) \mid (\phi^* \rightarrow \phi^*) \mid \forall\alpha\phi^* \mid$ $(\ \phi^*) \mid (\mathrm{A}\phi^*)$
	ϕ	::=	$\kappa_1\Pi^1 \mid \phi^* \mid (\neg\phi) \mid (\phi \rightarrow \phi) \mid \forall\alpha\phi \mid (\ \phi) \mid (\mathrm{A}\phi)$
	τ	::=	$\kappa \mid \Pi^n (n \geq 0)$

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

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 $\iota x(xQ)$ (for a formal definition of subformula refer to definition (8) in [4]). Thereby $F\iota x(xQ)$ is a propositional formula and $[\lambda y\ F\iota x(xQ)]$ a well-formed lambda expression. On the other hand xF is not a propositional formula and therefore $[\lambda x\ xF]$ not a well-formed lambda expression. This fact will become relevant in the discussion in TODO: reference, that describes a paradox in the presented formulation of the theory. (TODO: mention solvability already here?)

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

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

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

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

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

The identity for n-place relations with $n \geq 2$ is defined in terms of the identity of one-place relations, see (16)[4] for the full details.

The identity for ordinary objects follows Leibniz's 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 considered in more detail in section 2.5 that describes the *hyperintensionality* of relations in the theory.

2.4. The Axioms

Based on the language above an axiom system is defined that constructs a S5 modal logic with an actuality operator, axioms for definite descriptions that go along with Russell's analysis of descriptions, the substitution for 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 [4, chap. 8]. At this point the axioms of encoding are the most relevant, namely:

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

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

An interesting property of the Theory of Abstract Objects results from the definition of identity of one-place relations. Recall that two properties are defined to be identical if and only if they are *encoded* by the same (abstract) objects. Further note that 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*. Now 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 or G. It rather stipulates what needs to be proven, if $F = G$ is to be established, namely that they are necessarily encoded by the same objects. Therefore if two properties *should* be equal in some context an axiom has to be added to the theory that allows it to prove that both properties are encoded by the same abstract objects.

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

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

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

As outlined in the next chapter the hyperintensionality of the theory is a particular challenge for the representation of the theory in a shallow embedding (TODO: reference).

Furthermore it is important to note that the theory is only hyperintensional in exactly the sense described above. Propositonal reasoning is still governed by classical extensionality (TODO: be more precise).

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, hence there 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 always 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 the question is, does the Theory of Abstract Objects as constructed above have a model? An answer was provided by Peter Aczel who proposed the model structure illustrated in figure 2.1 (in fact to our knowledge Dana Scott proposed a first model for the theory before Peter Aczel that we believe is a special case of an Aczel model).

In an 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).

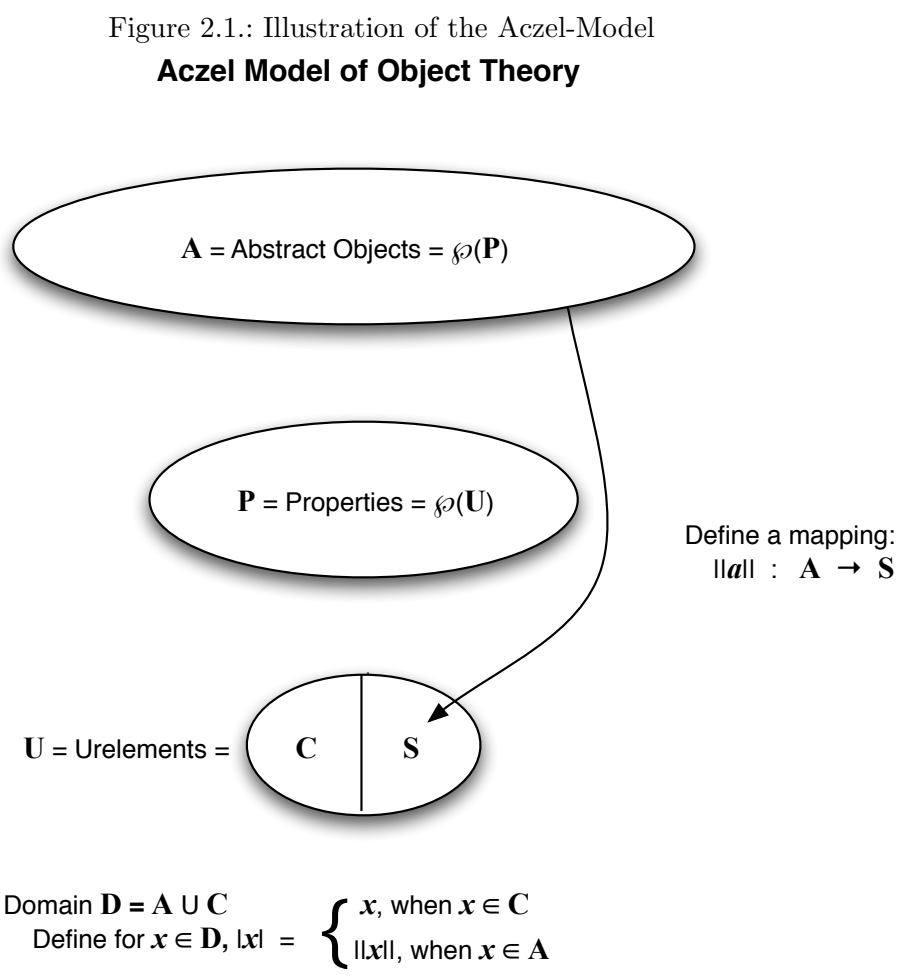
Ordinary urelements can serve as the denotations of the ordinary objects. Every abstract object on the other hand has a special urelement as its proxy. Which properties an abstract object exemplifies now solely depends on its proxy. Now note that the map from abstract objects to special urelements is not injective, so 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 are exemplification-indistinguishable, though. Interestingly the existence of abstract objects that are exemplification-indistinguishable is a theorem the Theory of Abstract Objects, see (197)[4].

Note that although the Aczel-model illustrated in figure 2.1 is non-modal, the extension to a modal version is straightforward by introducing primitive possible worlds as in the Kripke semantics of modal logic.

Also note that the Aczel-model in figure 2.1 is *extensional*. Since properties are represented as the power set of urelements, two properties are in fact equal if they are exemplified by the same objects. This has no bearing on the soundness of the Aczel-model as a model for the Theory of Abstract Objects, but it has the consequence, that statements like $[\lambda\ p \vee \neg p] = [\lambda\ q \vee \neg q]$ are true in the model, although they are not derivable from the axioms of object theory as explained in the last 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.

Figure 2.1.: Illustration of the Aczel-Model



3. The Embedding

TODO

Maybe skip the epigraph here.

3.1. Background

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

TODO: what more to say about Isabelle/HOL?

TODO: overview section? Reorder the sections?

TODO: explain color distinction between embedded and meta-logic somewhere.

3.2. Challenges

TODO: russell style paradox; hyperintensionality revisited; relations vs. functions revisited; general complexity, etc.

3.2.1. 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 the naive representation of the lambda expressions of the theory using the unrestricted, beta-convertible lambda expressions of functional logic results in the following paradox (see [2, p. 24-25]):

Assume $[\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]$ were a valid lambda 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 beta-reduction this implies that there exists a property, that b encodes, but does not exemplify. Since 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 beta-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 lambda expressions, so in particular $[\lambda x\ \exists F\ (xF\ \&\ \neg Fx)]$ is not part of the language. TODO: reference paradox section? TODO: reference the lambda expression section?

3.3. Basic Concepts

The introduction mentioned that shallow semantical embeddings were used to successfully represent different varieties of modal logic by implementing them using Kripke semantics. The advantage here is that Kripke semantics is well understood and there are extensive results about its completeness (TODO: reference).

For the Theory of Abstract Object the situation is different. Although it is believed that Aczel-models are sound, section 2.6 already established that in the traditional Aczel-model theorems are true, that are not derivable from the axioms of object theory.

Although in the following a variant of an Aczel-model is constructed that preserves the hyperintensionality of the theory at least to some degree, it is still known that there are true theorems in this model that are not derivable from the axioms of object theory. TODO: reference later section.

Given this lack of a model with a well-understood degree of completeness, the embedding uses a different approach. The embedding is divided into several *layers* as follows:

- The first layer represents the primitives of PLM using a hyper-intensional 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 [4, Chap. 9] is derived solely using the axiom system of the third layer and the 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 the third layer is independent of the first two layers.

The reasoning 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 layer three is *derived*) their consistency is thereby guaranteed as well.

The second layer tries to abstract from the details of the representation by implementing an approximation of the preliminary formal semantics of PLM[1]. 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 should be universal. The only exceptions are the primitive meta-rules of PLM: modus ponens, RN (necessitation) and GEN (universal generalisation). These rules do not follow from the axiom system itself, but are derived from the semantics in the second layer. Still as the corresponding semantical rules will again have to be derivable for *any* model, this does not have an impact on the universality of the subsequent reasoning.

There remains one issue, though. Since the logic of PLM is formulated in relational type theory, whereas Isabelle/HOL employs functional reasoning some formulations have to be adjusted to be representable and there may still be some reservations about the accuracy of the representation of the axiom system. This issue is addressed in section (TODO: reference).

The next sections will now describe the technical details of the embedding, whereas the discussion about the theoretical concept including the underlying model will follow in the next chapter. (TODO: reverse this?)

3.4. The Representation Layer

The first layer of the embedding (see A.1) implements a version of the Aczel-model 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* that are used for different interpretations of relations and connectives to achieve a hyper-intensional logic (see below TODO: reference).
- Type *bool* represents meta-logical truth values (*True* or *False*) and is inherited from Isabelle/HOL.
- Type ω represents ordinary urelements.

[1] Our thanks to Edward Zalta for supplying us with a preliminary version.

- Type σ represents special urelements.

Two constants are introduced:

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

Based on the primitive types above the following types are defined (see A.1.2):

- Type o is defined as the set of all functions of type $j \Rightarrow i \Rightarrow bool$ and represents truth values in the embedded logic.
- Type υ is defined as **datatype** $\upsilon = \omega\upsilon\ \omega \mid \sigma\upsilon\ \sigma$. This type represents urelements and an object of this type can be either an ordinary or a special urelement (with the respective type constructors $\omega\upsilon$ and $\sigma\upsilon$).
- Type Π_0 is defined as a synonym for type o and represents zero-place relations.
- Type Π_1 is defined as the set of all functions of type $\upsilon \Rightarrow j \Rightarrow i \Rightarrow bool$ and represents one-place relations (for an urelement a one-place relation evaluates to a truth value in the embedded logic; for an urelement, a state and a possible world it evaluates to a meta-logical truth value).
- Type Π_2 is defined as the set of all functions of type $\upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow bool$ and represents two-place relations.
- Type Π_3 is defined as the set of all functions of type $\upsilon \Rightarrow \upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow bool$ and represents three-place relations.
- Type α is defined as a synonym of the type of sets of one-place relations Π_1 *set*, i.e. every set of one-place relations constitutes an object of type α. This type represents abstract objects.
- Type ν is defined as **datatype** $\nu = \omega\nu\ \omega \mid \alpha\nu\ \alpha$. This type represents individuals and can be either an ordinary urelement ω or an abstract object α (with the respective type constructors $\omega\nu$ and $\alpha\nu$).
- Type κ is defined as the set of all objects of type ν *option* and represents individual terms. The type *'a option* is part of Isabelle/HOL and consists of a type constructor *Some x* for an object x of type *'a* (in 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 3.1. *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**. *In the remainder of this document these morphisms are omitted and definitions are stated directly for the representation types.*

3.4.2. Individual Terms and Definite Descriptions

There are two basic types of individual terms in PLM: definite descriptions and individual variables. For any logically proper definite description there is an individual variable that denotes the same object. A definite description is logically proper if its matrix is true for a unique object.

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

$x^P = Some\ x$

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

It is now possible to define definite descriptions as follows:

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

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

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

$proper\ x = (None \neq x)$

$rep\ x = the\ x$

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

3.4.3. Mapping from Individuals to Urelements

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

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

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

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

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

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

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

3.4.4. Exemplification of n-place relations

Exemplification of n-place relations can now be defined. Exemplification of zero-place relations is simply defined as the identity, whereas exemplification of n-place relations for $n \geq 1$ is defined to be true, if all individual terms are logically proper and the function application of the relation to the urelements corresponding to the individuals yields true for a given possible world and state (see A.1.5):

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

3.4.5. Encoding

Encoding can now be defined as follows (see A.1.6):

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

That is for a given state s and a given possible world w it holds that an individual term x encodes F, if x is logically proper, the representative individual variable of x is of the form $\alpha\nu\ \alpha$ for some abstract object α and F is contained in α (remember that abstract objects are defined to be sets of one-place relations). Also note that encoding is a represented as a function of possible worlds and states to ensure type-correctness, but its evaluation does not depend on either. On the other hand whether F is contained in α actually does depend on the behavior of F in *all* states.

3.4.6. Connectives and Quantifiers

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

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

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

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

The basic connectives and quantifiers are now defined as follows (see A.1.7):

- $\neg p = (\lambda s\ w.\ s = dj \wedge \neg\ p\ dj\ w \vee s \neq dj \wedge I\text{-}NOT\ p\ s\ w)$
- $p \rightarrow q = (\lambda s\ w.\ s = dj \wedge (p\ dj\ w \longrightarrow q\ dj\ w) \vee s \neq dj \wedge I\text{-}IMPL\ p\ q\ s\ w)$
- $\forall_\nu\ x\ .\ \varphi\ x = (\lambda s\ w.\ \forall x.\ (\varphi\ x)\ s\ w)$
- $\forall_0\ p\ .\ \varphi\ p = (\lambda s\ w.\ \forall p.\ (\varphi\ p)\ s\ w)$
- $\forall_1\ F\ .\ \varphi\ F = (\lambda s\ w.\ \forall F.\ (\varphi\ F)\ s\ w)$
- $\forall_2\ F\ .\ \varphi\ F = (\lambda s\ w.\ \forall F.\ (\varphi\ F)\ s\ w)$
- $\forall_3\ F\ .\ \varphi\ F = (\lambda s\ w.\ \forall F.\ (\varphi\ F)\ s\ w)$
- $\Box p = (\lambda s\ w.\ \forall v.\ p\ s\ v)$
- $\mathcal{A}p = (\lambda s\ w.\ p\ s\ dw)$

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

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

Remark 3.2. *It may be that non-classical behavior in states $s \neq dj$ for negation and implication is found not to be sufficient for achieving the desired level of hyperintensionality. 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.*

3.4.7. Lambda Expressions

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

the definition of the lambda expressions of the embedded logic is non trivial. The embedding defines them as follows (see A.1.8):

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

Remark 3.3. *For technical reasons Isabelle only allows lambda expressions for one-place relations to use a nice binder notation. For two- and three-place relations the following notation is used instead:* $\boldsymbol{\lambda}^2\ (\lambda x\ y.\ \varphi\ x\ y)$, $\boldsymbol{\lambda}^3\ (\lambda x\ y\ z.\ \varphi\ x\ y\ z)$.

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

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

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

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

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

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

However in general $\nu\upsilon$ can map several abstract objects to the same special urelement, so an analog statement for abstract objects does not hold for arbitrary φ. As described in section 3.2.1 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 is defined to be proper if its truth evaluation for the actual state only depends on the urelement corresponding to the individual (see A.1.9):

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

Now by the definition of proper maps the evaluation of lambda expressions behaves as expected:

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

Remark 3.4. *Note that the above equation does not quantify over all states, but is only true for the actual state dj. This is sufficient given that truth evaluation only depends on the actual state (see TODO: reference) and goes along with the desired semantics of lambda expressions (see TODO: reference).*

The concept behind this is that maps that contain encoding formulas in its argument are in general not proper and thereby the paradox mentioned in section 3.2.1 is avoided. In fact proper maps are the most general kind of functions that may appear in a lambda-expression, such that beta-conversion holds. In what way proper maps correspond to the formulas that PLM allows as the matrix of a lambda expression is a complex question and discussed seperately in section TODO: reference. A detailed discussion about the denotations of *improper* lambda-expression and how the mentioned paradox is avoided precisely is presented in section TODO: reference (same section as before?).

3.4.8. Validity

A formula is considered semantically valid for a possible world v if it evaluates to *True* for the actual state *dj* and the given possible world v. Semantic validity is defined as follows (see A.1.10):

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

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

Remark 3.5. *The Isabelle Theory in the appendix defines the syntax $v \models p$ in the representation layer, as this is the syntax used in the preliminary formal semantics of PLM. The syntax $[p \ in \ v]$ that is easier to use in Isabelle due to bracketing the expression is only introduced after the semantics is derived in A.2.3. For simplicity only the latter syntax is used in this document.*

3.4.9. Concreteness

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

- Ordinary objects are possibly concrete. In the meta-logic this means that for every ordinary object there exists at least one possible world, in which the object is concrete.

- Abstract objects are never concrete.

An additional requirement is enforced by axiom (32.4)[4]. 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):

- $\forall\, x.\ \exists\, v.\ ConcreteInWorld\ x\ v$
- $\exists\, x\ v.\ ConcreteInWorld\ x\ v \wedge (\exists\, w.\ \neg\ ConcreteInWorld\ x\ w)$
- $\exists\, w.\ \forall\, x.\ ConcreteInWorld\ x\ w \longrightarrow (\forall\, v.\ ConcreteInWorld\ x\ v)$

Concreteness can now be defined as a one-place relation (see A.1.11):

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

The equivalence of the axioms stated in the meta-logic and the notion of concreteness in Principia can now be verified (see A.12.4):

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

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, that can be easily understood and that 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.

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

Several subtleties have to be considered:

- n-place relations are only represented up for $n \leq 3$. As the resulting language is already expressive enough to represent the most interesting parts of the theory and analog implementations for $n > 3$ would be trivial to implement, this is considered to be sufficient.
- There is a distinction between individual terms and variables. This circumstance was already mentioned in section 3.4.2: an individual term in PLM can either be an individual variable (or constant) or a definite description. Statements in PLM that use individual variables are represented using the decoration $\_^P$.
- Whereas conceptually in PLM if a general term φ as it occurs in definite descriptions, quantifications and lambda expressions above contains a *free* variable is used within the scope of a variable binding operator, the 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* and using the native concept of binding operators in Isabelle to convert this function to a term with bound variables. This concept is discussed in more detail in section TODO: reference.
- The representation layer of the embedding defines a separate quantifier for every type of variable in PLM. This is done to assure that only quantifications 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 quantifications ranging over individuals

(of type ν) are part of the language of PLM. Once the semantics is introduced in section TODO:reference, a *type class* is introduced that is defined by the semantics of all 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 TODO: reference.

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

$$[\varphi \; in \; v]$$

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

TODO: talk about scopes?

TODO: constants vs. variables.

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 the formulas of the embedded logic in a similar way as the (at the time of writing unpublished) preliminary semantics of object theory.

TODO: repeat more statements from the appendix throughout the section?

3.5.1. Domains and Denotation Functions

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

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

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

In the embedding all relation terms always denote, therefore the denotation functions $d_0, \ldots, d_3$ 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 3.6. *The definitions for semantical domains and denotations seem trivial. However, it must be considered that conceptually the abstract types of the representation layer now have the role of primitive types. Although for simplicity the last section regarded the type* o *as synonym of* $j \Rightarrow i \Rightarrow$ *bool, it was introduced as a distinct type for which the set of all functions of type* $j \Rightarrow i \Rightarrow$ *bool merely serves as the underlying set of representatives. An object of type* o *cannot directly be substituted for a variable of type* $j \Rightarrow i \Rightarrow$ *bool. To do so it first has to be mapped to its representative of type* $j \Rightarrow i \Rightarrow$ *bool by the use of the morphism eval*o *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 trivial, 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

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

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

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

The fact that all exemplification extensions only depend on the evaluation of the respective relation in the actual state goes along with the concepts described in section TODO: reference.

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

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

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

The encoding extension of a relation is defined as the set of all abstract objects that contain the relation. This is again in accordance with the Aczel-model. Since encoding is modally rigid the encoding extension does not need to be relativized for possible worlds.

3.5.3. Truth Conditons of Formulas

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

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

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

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

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

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

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

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

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

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. For the actuality operator $\mathcal{A}\psi$ is defined to be semantically true, if and only if ψ is true in the designated actual world.

Once more it is important to note that all introduced truth conditions do *not* depend on the actual state following the ideas of section TODO: reference.

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:

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

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

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

3.5.5. Denotation of Lambda Expressions

The most complex part of the semantical abstraction is the definition of a denotation for lambda expressions. The preliminary 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 lambda expressions are coupled to syntactical conditions. This fact is represented using the notion of *proper maps* as a restriction for the matrix of a lambda expression that was introduced in section 3.4.7. The definitions are implemented as follows (see A.2.1.11):

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

The first condition for *elementary* lambda expressions is straightforward. The general case in the second condition is more complex: Given that φ is a proper map then the

relation denoted by the lambda 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 beta-conversion restricted to denoting individuals: the truth condition of the lambda 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 again straightforward.

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 derived. Future work may try to refrain from this second kind of statements.

The statements are comprised of the following (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:
   $Some\ r = d_1\ E! \Longrightarrow \forall\, x.\ \exists\, w.\ \omega\nu\ x \in ex1\ r\ w$
4. An object that is contained in the extension of the concreteness property in any world is an ordinary object:
   $Some\ r = d_1\ E! \Longrightarrow \forall\, x.\ x \in ex1\ r\ w \longrightarrow (\exists\, y.\ x = \omega\nu\ y)$
5. The denotation functions for relation terms are injective, e.g.
   $d_1\ F = d_1\ G \Longrightarrow F = G$
6. The denotation functions for individual terms is injective for denoting terms:
   $Some\ o_1 = d_\kappa\ x \wedge Some\ o_1 = d_\kappa\ y \Longrightarrow x = y$

Especially the statements 5 and 6 are only derivable due to the specific construction of the representation layer: since the semantical domains were defined as the representation sets of the respective type of term and denotations were defined canonically, objects that have the same denotation are identical as objects of their abstract type.

3.5.7. Proper Maps

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

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

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

3.6. General All-Quantifier

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

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

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

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

Since it was already shown in the last section that all variable types satisfy this property, the type class can immediately be instantiated for the types ν, Π_0, Π_1, Π_2 and Π_3 (see **??**). 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 be used and the type specific versions of the quantifiers are no longer needed. 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.

3.7. Derived Language Elements

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

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

3.7.1. Connectives

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

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

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

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

3.7.2. Identity

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

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

TODO: reformatting/line breaks in the list somehow.

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

3.8. Proving Method *meta-solver*

3.8.1. Concept

Since the section 3.5 constructed a first abstraction layer on top of the first layer that focuses on a single kind of model and is connected with specific technicalities, it makes sense to revisit the general concept of the layered structure of the embedding.

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

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

Remark 3.7. *The Eisbach package even allows to construct more complex proving methods that involve pattern matching. This is utilized in the construction of a substitution method in section TODO: reference details*

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

For example an implication formula can be deconstructed as follows:

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

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

3.8.2. Implementation

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

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

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

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

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

Notably the representation layer has the property that the defined identities are equivalent to the identity in the meta-logic. Formally the following statements are true and defined as rules for the *meta-solver* (see A.5.15):

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

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

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

Remark 3.8. *Instead of introducing a custom proving method using the Eisbach package, a similar effect could be achieved by instead supplying the derived introduction, elimination and substitution rules directly to one of the existing proving methods like auto or clarsimp. In practice, however, we found that the custom meta-solver produces more reliable results, especially in the case that a proving objective cannot be solved by the supplied rules completely. Moreover the construction of a custom proving method may serve as a proof of concept and inspire the development of further more complex proving methods that go beyond a simple resolution prover in the future.*

3.8.3. Applicability

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

The main application of the method in the embedding is its use in the derivation of the axiom system as described in section TODO: reference. Furthermore the *meta-solver* can aid in examining the meta-logical properties of the embedding. Care has been taken that the meta-solver only employs *reversable* transformations, thereby it is for example

justified to use it to simplify a statement before employing a tool like **nitpick** in order to look for counter-models for a statement.

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

3.9. General Identity Relation

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

Since the identity of PLM is not directly defined using semantic truth conditions, but instead by the means of a complex formulas in the embedded logic that is specific to each type of terms, the question is whether they share a property that can reasonably be used as the condition of a type class.

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

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

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

Since now general quantification and general identity are available, it seems reasonable to define the unique existence quantifier that involves both quantification and identity. To that end a derived type class that is introduced that is the combination of the *quantifiable* and the *identifiable* classes (since both general quantification and identity have to be available).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 allows:

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

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 used as the basis of the embedding 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. To that end axioms are not stated relative to a specific possible world, but using the following definition (see A.7):

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

Axioms are unconditionally true in all possible worlds. The only exceptions are *necessitation-averse*, resp. *modally fragile* axioms (currently there is only one such axiom). Such axioms are stated using the following definition:

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

Remark 3.9. *Additionally a proving method axiom-meta-solver is introduced, that unfolds above definition, then applies the meta-solver and if possible resolves the proof objective automatically.*

3.10.1. Axioms as Schemata

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

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

For modally fragile axioms only the following rules are introduced:

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

Remark 3.10. *To simplify the instantiation of the axioms in subsequent proofs, a set of* attributes *is defined that can be used to transform the statement of the axioms using the rules defined above.*

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

3.10.2. Derivation of the Axioms

Most of the axioms can be derived by the *axiom-meta-solver* directly. Some axioms, however, require more verbose proofs or special attention has to be taken regarding their representation in the functional setting of Isabelle/HOL. Therefore in the following the complete axiom system is listed and discussed in detail where necessary. Additionally they are associated with the numbering in the current draft of PLM [4].

3.10.2.1. Axioms for Negations and Conditionals

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

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

- $[\![\neg\varphi \to \neg\psi \to (\neg\varphi \to \psi \to \varphi)]\!]$ (21.3)

3.10.2.2. Axioms of Identity

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

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

TODO: discussion

3.10.2.3. Axioms of Quantification

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

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

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

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

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

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

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

What's 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 for the axioms, especially considering the most recent developments described in TODO: reference. At the time of writing the fact that due to the type restrictions of the embedding, the reformulated version of the axioms is *derivable* from the original version must suffice.

Remark 3.11. *A formulation of the axioms as in PLM could be possible using the concept of domain restricted quantification as employed in embedding free logics (TODO: reference) to define a restricted quantification over the type κ. This would require some non-trivial restructuring of the embedding, though.*

What's left is to clarify the precondition for (29.5). The predicate *SimpleExOrEnc* is defined as an inductive predicate with the following introduction rules:

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

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

3.10.2.4. 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):

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

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

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

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

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

3.10.2.5. Axioms of Necessity

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

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

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

3.10.2.6. Axioms of Necessity and Actuality

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

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

3.10.2.7. Axioms of Descriptions

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

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

Given the technicalities of descriptions already discussed in section 3.10.2.3 it comes at no surprise that the proof for this statement is rather verbose.

3.10.2.8. Axioms of Complex Relation Terms

The axioms of complex relation terms deal with the properties of lambda expressions.

Since the *meta-solver* was not equipped with explicit rules for lambda expressions, the statements rely on their semantic properties as described in section 3.5 directly.

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

- $\boldsymbol{\lambda} x.\ \varphi\ x = \boldsymbol{\lambda} y.\ \varphi\ y$ (36.1)
- $\mathit{IsProperInX}\ \varphi \Longrightarrow [\![\ (\!|\boldsymbol{\lambda} x.\ \varphi\ (x^P),x^P|\!) \equiv \varphi\ (x^P)\ ]\!]$ (36.2)
- $\mathit{IsProperInXY}\ \varphi \Longrightarrow [\![\ (\!|\boldsymbol{\lambda}^2\ (\lambda x\ y.\ \varphi\ (x^P)\ (y^P)),x^P,y^P|\!) \equiv \varphi\ (x^P)\ (y^P)\ ]\!]$ (36.2)
- $\mathit{IsProperInXYZ}\ \varphi \Longrightarrow$
  $[\![\ (\!|\boldsymbol{\lambda}^3\ (\lambda x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),x^P,y^P,z^P|\!) \equiv \varphi\ (x^P)\ (y^P)\ (z^P)\ ]\!]$

(36.2)

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

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

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

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

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

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

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

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

3.10.2.9. Axioms of Encoding

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

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

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

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

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

3.10.2.10. Summery

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

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

To verify that the axiom system is a good representation of the reference, as a next step the deductive system PLM as described in [4, Chap. 9] is derived solely based on the formulation of the axioms without falling back to the meta-logic.

3.11. The Deductive System PLM

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

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

A. Isabelle Theory

A.1. Embedding

A.1.1. Primitives

typedecl i — possible worlds
typedecl j — states

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

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

A.1.2. Derived Types

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

type-synonym Π_0 = o — zero place relations
typedef Π_1 = $UNIV::(\upsilon \Rightarrow j \Rightarrow i \Rightarrow bool)\ set$
  morphisms *eval*Π_1 *make*Π_1 **..** — one place relations
typedef Π_2 = $UNIV::(\upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow bool)\ set$
  morphisms *eval*Π_2 *make*Π_2 **..** — two place relations
typedef Π_3 = $UNIV::(\upsilon \Rightarrow \upsilon \Rightarrow \upsilon \Rightarrow j \Rightarrow i \Rightarrow bool)\ set$
  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 κ = $UNIV::(\nu\ option)\ set$
  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 A.1. *Individual terms can be definite descriptions which may not denote. Therefore the type for individual terms κ is defined as ν option. Individuals are represented by Some x for an individual x of type ν, whereas non-denoting individual terms are represented by None. Note that relation terms on the other hand always denote, so there is no need for a similar distinction between relation terms and relations.*

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

Remark A.2. *Individual terms can be explicitly marked to only range over logically proper objects (e.g. x^P). Their logical propriety and (in case they are logically proper) the represented individual can be extracted from the internal representation as ν option.*

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

Remark A.3. *Definite descriptions map conditions on individuals to individual terms. If no unique object satisfying the condition exists (and therefore the definite description is not logically proper), the individual term is set to None.*

A.1.4. Mapping from objects to urelements

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

A.1.5. Exemplification of n-place relations.

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

Remark A.4. *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* :: $\kappa \Rightarrow \Pi_1 \Rightarrow o$ ($\{\!|$-,-$|\!\}$) **is**
$\lambda\ x\ F\ w\ s$. $(proper\ x) \wedge$ *case-*ν $(\lambda\ \omega$. *False*$)$ $(\lambda\ \alpha\ .\ F \in \alpha)$ $(rep\ x)$ **.**

Remark A.5. *An encoding formula can again only be true if the individual term is logically proper. Furthermore ordinary objects never encode, whereas abstract objects encode a property if and only if the property is contained in it as per the Aczel Model.*

A.1.7. Connectives and Quantifiers

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

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

Remark A.6. *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 A.7. *Lambda expressions have to convert maps from individuals to propositions to relations that are represented by maps from urelements to truth values.*

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

A.1.9. Proper Maps from Individual Terms to Propositions

Remark A.8. *The embedding introduces the notion of* proper *maps from individual terms to propositions.*
Such a map is proper if and only for all proper individual terms its truth evaluation in the actual state only depends on the urelement corresponding to the individual the term denotes.
Proper maps are exactly those maps that - when used in a lambda-expression - unconditionally allow beta-reduction.

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

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

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.9. *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*
$\qquad \wedge\ (\exists\ w\ .\ \neg$ *ConcreteInWorld x w*$)$
abbreviation (*input*) *PossiblyNoContingentObjectExists* **where**
PossiblyNoContingentObjectExists $\equiv \exists\ w\ .\ \forall\ x\ .$ *ConcreteInWorld x w*
$\qquad \longrightarrow\ (\forall\ v\ .$ *ConcreteInWorld x v*$)$
axiomatization where
OrdinaryObjectsPossiblyConcreteAxiom:
OrdinaryObjectsPossiblyConcrete
and *PossiblyContingentObjectExistsAxiom*:
PossiblyContingentObjectExists
and *PossiblyNoContingentObjectExistsAxiom*:
PossiblyNoContingentObjectExists

Remark A.10. *In order to define concreteness, care has to be taken that the defined notion of concreteness coincides with the meta-logical distinction between abstract objects and ordinary objects. Furthermore the axioms about concreteness have to be satisfied. This is achieved by introducing an uninterpreted constant ConcreteInWorld that determines whether an ordinary object is concrete in a given possible world. This constant is axiomatized, such that all ordinary objects are possibly concrete, contingent objects possibly exist and possibly no contingent objects exist.*

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

Remark A.11. *Concreteness of ordinary objects is now defined using this axiomatized uninterpreted constant. Abstract objects on the other hand are never concrete.*

A.1.12. Collection of Meta-Definitions

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

named-theorems *meta-defs*

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

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

A.1.13. Auxiliary Lemmata

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

named-theorems *meta-aux*

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

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

A.2. Semantics

A.2.1. Definition

locale *Semantics*
begin
named-theorems *semantics*

A.2.1.1. Semantical Domains

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

A.2.1.2. Denotation Functions

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

A.2.1.3. Actual World

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

A.2.1.4. Exemplification Extensions

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

A.2.1.5. Encoding Extensions

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

A.2.1.6. Collection of Semantical Definitions

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

A.2.1.7. Truth Conditions of Exemplification Formulas

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

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

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

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

A.2.1.8. Truth Conditions of Encoding Formulas

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

A.2.1.9. Truth Conditions of Complex Formulas

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

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

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

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

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

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

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

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

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

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

A.2.1.10. Denotations of Descriptions

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

A.2.1.11. Denotations of Lambda Expressions

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

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

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

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

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

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

shows $\bigwedge w\ o_1\ o_2\ o_3\ r$. *Some* $r = d_3\ (\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \varphi\ (x^P)\ (y^P)\ (z^P)))$
$\wedge$ *Some* $o_1 = d_\kappa\ x \wedge$ *Some* $o_2 = d_\kappa\ y \wedge$ *Some* $o_3 = d_\kappa\ z$
$\longrightarrow ((o_1,o_2,o_3) \in ex3\ r\ w) = (w \models \varphi\ x\ y\ z)$
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*]: $(\bigwedge w\ r$. *Some* $r = d_0\ (\boldsymbol{\lambda}^0\ \varphi) \longrightarrow ex0\ r\ w = (w \models \varphi))$
by (*auto simp*: *meta-defs meta-aux semantics-defs*)

A.2.1.12. Auxiliary Lemmata

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

A.2.2. Introduction Rules for Proper Maps

Remark A.12. *Introduction rules for proper maps are derived. In particular every map whose argument only occurs in exemplification expressions is proper.*

named-theorems *IsProper-intros*

lemma *IsProperInX-intro*[*IsProper-intros*]:
IsProperInX $(\lambda\ x\ .\ \chi$
$(*$ *one place* $*)\ (\lambda\ F\ .\ (\!|F,x|\!))$
$(*$ *two place* $*)\ (\lambda\ F\ .\ (\!|F,x,x|\!))\ (\lambda\ F\ a\ .\ (\!|F,x,a|\!))\ (\lambda\ F\ a\ .\ (\!|F,a,x|\!))$
$(*$ *three place three x* $*)\ (\lambda\ F\ .\ (\!|F,x,x,x|\!))$

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

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

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

```
    (* 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* :: o⇒*i*⇒*bool* ([- *in* -] [*1*]) **where**
validity-in ≡ λ φ *v* . *v* ⊨ φ
definition *actual-validity* :: o⇒*bool* ([-] [*1*]) **where**
actual-validity ≡ λ φ . *dw* ⊨ φ
definition *necessary-validity* :: o⇒*bool* (□[-] [*1*]) **where**
necessary-validity ≡ λ φ . ∀ *v* . (*v* ⊨ φ)

A.3. General Quantification

Remark A.13. *In order to define general quantifiers that can act on individuals as well as relations a type class is introduced which assumes the semantics of the all quantifier. This type class is then instantiated for individuals and relations.*

A.3.1. Type Class

Type class for quantifiable types:

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

Semantics for the general all quantifier:

lemma (**in** *Semantics*) *T8*: **shows** (*w* ⊨ ∀ *x* . ψ *x*) = (∀ *x* . (*w* ⊨ ψ *x*))
using *quantifiable-T8* **.**

A.3.2. Instantiations

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

instantiation o :: *quantifiable*
begin
definition *forall*-o :: (o⇒o)⇒o **where** *forall*-o ≡ $forall_{o}$
instance proof
fix *w* :: *i* **and** ψ :: o⇒o
show (*w* ⊨ ∀*x*. ψ *x*) = (∀*x*. (*w* ⊨ ψ *x*))
unfolding *forall-o-def* **using** *Semantics.T8*-o **.**
qed
end

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

```
    unfolding forall-Π1-def using Semantics.T8-1 .
  qed
end

instantiation Π2 :: quantifiable
begin
  definition forall-Π2 :: (Π2⇒o)⇒o where forall-Π2 ≡ forall2
  instance proof
    fix w :: i and ψ :: Π2⇒o
    show (w ⊨ ∀ x. ψ x) = (∀ x. (w ⊨ ψ x))
      unfolding forall-Π2-def using Semantics.T8-2 .
  qed
end

instantiation Π3 :: quantifiable
begin
  definition forall-Π3 :: (Π3⇒o)⇒o where forall-Π3 ≡ forall3
  instance proof
    fix w :: i and ψ :: Π3⇒o
    show (w ⊨ ∀ x. ψ x) = (∀ x. (w ⊨ ψ x))
      unfolding forall-Π3-def using Semantics.T8-3 .
  qed
end
```

A.4. Basic Definitions

A.4.1. Derived Connectives

```
definition conj::o⇒o⇒o (infixl & 53) where
  conj ≡ λ x y . ¬(x → ¬y)
definition disj::o⇒o⇒o (infixl ∨ 52) where
  disj ≡ λ x y . ¬x → y
definition equiv::o⇒o⇒o (infixl ≡ 51) where
  equiv ≡ λ x y . (x → y) & (y → x)
definition diamond::o⇒o (◇- [62] 63) where
  diamond ≡ λ φ . ¬□¬φ
definition (in quantifiable) exists :: ('a⇒o)⇒o (binder ∃ [8] 9) where
    exists ≡ λ φ . ¬(∀ x . ¬φ 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 :: Π1 (O!) where Ordinary ≡ λx. ◇(|E!,x^P|)
definition Abstract :: Π1 (A!) where Abstract ≡ λx. ¬◇(|E!,x^P|)
```

A.4.3. Identity Definitions

```
definition basic-identity_E::Π2 where
  basic-identity_E ≡ λ² (λ x y . (|O!,x^P|) & (|O!,y^P|)
                       & □(∀ F. (|F,x^P|) ≡ (|F,y^P|)))
```

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

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

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

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

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

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

A.5. MetaSolver

Remark A.14. *meta-solver is a resolution prover that translates expressions in the embedded logic to expressions in the meta-logic, resp. semantic expressions as far as possible. The rules for connectives, quantifiers, exemplification and encoding are easy to prove. Futhermore rules for the defined identities are derived using more verbose proofs. By design the defined identities in the embedded logic coincide with the meta-logical equality.*

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

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

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

A.5.1. Rules for Implication

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

A.5.2. Rules for Negation

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

A.5.3. Rules for Conjunction

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

A.5.4. Rules for Equivalence

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

A.5.5. Rules for Disjunction

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

A.5.6. Rules for Necessity

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

A.5.7. Rules for Possibility

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

A.5.8. Rules for Quantification

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

A.5.10. Rules for Encoding

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

A.5.11. Rules for Exemplification

A.5.11.1. Zero-place Relations

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

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

A.5.11.2. One-Place Relations

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

A.5.11.3. Two-Place Relations

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

A.5.11.4. Three-Place Relations

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

A.5.12. Rules for Being Ordinary

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

shows $[(\!|O!,x|\!)\ in\ v]$
proof −
have *IsProperInX* $(\lambda x.\ \Diamond(\!|E!,x|\!))$
by *show-proper*
moreover have $[\Diamond(\!|E!,x|\!)\ in\ v]$
apply *meta-solver*
using *ConcretenessSemantics1 propex*$_1$ *assms* **by** *fast*
ultimately show $[(\!|O!,x|\!)\ in\ v]$
unfolding *Ordinary-def*
using *D5-1 propex*$_1$ *assms ConcretenessSemantics1 Exe1S*
by *blast*
qed
lemma *OrdE*[*meta-elim*]:
assumes $[(\!|O!,x|\!)\ in\ v]$
shows $\exists\ o_1\ y.\ Some\ o_1 = d_\kappa\ x \wedge o_1 = \omega\nu\ y$
proof −
have $\exists r\ o_1.\ Some\ r = d_1\ O! \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in ex1\ r\ v$
using *assms Exe1E* **by** *simp*
moreover have *IsProperInX* $(\lambda x.\ \Diamond(\!|E!,x|\!))$
by *show-proper*
ultimately have $[\Diamond(\!|E!,x|\!)\ in\ v]$
using *D5-1* **unfolding** *Ordinary-def* **by** *fast*
thus *?thesis*
apply − **apply** *meta-solver*
using *ConcretenessSemantics2* **by** *blast*
qed
lemma *OrdS*[*meta-cong*]:
$[(\!|O!,x|\!)\ in\ v] = (\exists\ o_1\ y.\ Some\ o_1 = d_\kappa\ x \wedge o_1 = \omega\nu\ y)$
using *OrdI OrdE* **by** *blast*

A.5.13. Rules for Being Abstract

lemma *AbsI*[*meta-intro*]:
assumes $\exists\ o_1\ y.\ Some\ o_1 = d_\kappa\ x \wedge o_1 = \alpha\nu\ y$
shows $[(\!|A!,x|\!)\ in\ v]$
proof −
have *IsProperInX* $(\lambda x.\ \neg\Diamond(\!|E!,x|\!))$
by *show-proper*
moreover have $[\neg\Diamond(\!|E!,x|\!)\ in\ v]$
apply *meta-solver*
using *ConcretenessSemantics2 propex*$_1$ *assms*
by (*metis* ν.*distinct*(*1*) *option.sel*)
ultimately show $[(\!|A!,x|\!)\ in\ v]$
unfolding *Abstract-def*
using *D5-1 propex*$_1$ *assms ConcretenessSemantics1 Exe1S*
by *blast*
qed
lemma *AbsE*[*meta-elim*]:
assumes $[(\!|A!,x|\!)\ in\ v]$
shows $\exists\ o_1\ y.\ Some\ o_1 = d_\kappa\ x \wedge o_1 = \alpha\nu\ y$
proof −
have *1*: *IsProperInX* $(\lambda x.\ \neg\Diamond(\!|E!,x|\!))$
by *show-proper*
have $\exists r\ o_1.\ Some\ r = d_1\ A! \wedge Some\ o_1 = d_\kappa\ x \wedge o_1 \in ex1\ r\ v$
using *assms Exe1E* **by** *simp*
moreover hence $[\neg\Diamond(\!|E!,x|\!)\ in\ v]$
using *D5-1*[*OF 1*]
unfolding *Abstract-def* **by** *fast*

ultimately show *?thesis*
apply − **apply** *meta-solver*
using *ConcretenessSemantics1 propex*$_1$
by (*metis* ν*.exhaust*)
qed
lemma *AbsS*[*meta-cong*]:
$[(\!|A!,x|\!)\ in\ v] = (\exists\ o_1\ y.\ Some\ o_1 = d_\kappa\ x \wedge o_1 = \alpha\nu\ y)$
using *AbsI AbsE* **by** *blast*

A.5.14. Rules for Definite Descriptions

lemma *TheEqI*:
assumes $\bigwedge x.\ [\varphi\ x\ in\ dw] = [\psi\ x\ in\ dw]$
shows $(\boldsymbol{\iota} x.\ \varphi\ x) = (\boldsymbol{\iota} x.\ \psi\ x)$
proof −
have *1*: $d_\kappa\ (\boldsymbol{\iota} x.\ \varphi\ x) = d_\kappa\ (\boldsymbol{\iota} x.\ \psi\ x)$
using *assms D3* **unfolding** w_0*-def* **by** *simp*
{
assume $\exists\ o_1\ .\ Some\ o_1 = d_\kappa\ (\boldsymbol{\iota} x.\ \varphi\ x)$
hence *?thesis* **using** *1* d_κ*-inject* **by** *force*
}
moreover {
assume $\neg(\exists\ o_1\ .\ Some\ o_1 = d_\kappa\ (\boldsymbol{\iota} x.\ \varphi\ x))$
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 $Eq_E I$[*meta-intro*]:
assumes $\exists\ o_1\ X\ o_2.\ Some\ o_1 = d_\kappa\ x \wedge Some\ o_2 = d_\kappa\ y \wedge o_1 = o_2 \wedge o_1 = \omega\nu\ X$
shows $[x =_E y\ in\ v]$
proof −
obtain $o_1\ X\ o_2$ **where** *1*:
$Some\ o_1 = d_\kappa\ x \wedge Some\ o_2 = d_\kappa\ y \wedge o_1 = o_2 \wedge o_1 = \omega\nu\ X$
using *assms* **by** *auto*
obtain r **where** *2*:
$Some\ r = d_2\ basic\text{-}identity_E$
using $propex_2$ **by** *auto*
have $[(\!|O!,x|\!)\ \&\ (\!|O!,y|\!)\ \&\ \Box(\forall F.\ (\!|F,x|\!) \equiv (\!|F,y|\!))\ in\ v]$
proof −
have $[(\!|O!,x|\!)\ in\ v] \wedge [(\!|O!,y|\!)\ in\ v]$
using *OrdI 1* **by** *blast*
moreover have $[\Box(\forall F.\ (\!|F,x|\!) \equiv (\!|F,y|\!))\ in\ v]$
apply *meta-solver* **using** *1* **by** *force*
ultimately show *?thesis* **using** *ConjI* **by** *simp*
qed
moreover have $IsProperInXY\ (\lambda\ x\ y\ .\ (\!|O!,x|\!)\ \&\ (\!|O!,y|\!)\ \&\ \Box(\forall F.\ (\!|F,x|\!) \equiv (\!|F,y|\!)))$
by *show-proper*
ultimately have $(o_1,\ o_2) \in ex2\ r\ v$
using *D5-2 1 2*
unfolding $basic\text{-}identity_E$*-def* **by** *fast*
thus $[x =_E y\ in\ v]$

using *Exe2I 1 2*
unfolding *basic-identity$_E$-infix-def basic-identity$_E$-def*
by *blast*
qed
lemma *Eq$_E$E[meta-elim]*:
assumes $[x =_E y\ in\ v]$
shows $\exists\ o_1\ X\ o_2.\ Some\ o_1 = d_\kappa\ x \wedge Some\ o_2 = d_\kappa\ y \wedge o_1 = o_2 \wedge o_1 = \omega\nu\ X$
proof −
have *IsProperInXY* $(\lambda\ x\ y\ .\ (\!|O!,x|\!)\ \&\ (\!|O!,y|\!)\ \&\ \Box(\forall\ F.\ (\!|F,x|\!) \equiv (\!|F,y|\!)))$
by *show-proper*
hence *1*: $[(\!|O!,x|\!)\ \&\ (\!|O!,y|\!)\ \&\ \Box(\forall\ F.\ (\!|F,x|\!) \equiv (\!|F,y|\!))\ in\ v]$
using *assms* **unfolding** *basic-identity$_E$-def basic-identity$_E$-infix-def*
using *D4-2 T1-2 D5-2* **by** *meson*
hence *2*: $\exists\ o_1\ o_2\ X\ Y\ .\ Some\ o_1 = d_\kappa\ x \wedge o_1 = \omega\nu\ X$
$\wedge\ Some\ o_2 = d_\kappa\ y \wedge o_2 = \omega\nu\ Y$
apply $(subst\ (asm)\ ConjS)$
apply $(subst\ (asm)\ ConjS)$
using *OrdE* **by** *auto*
then obtain $o_1\ o_2\ X\ Y$ **where** *3*:
$Some\ o_1 = d_\kappa\ x \wedge o_1 = \omega\nu\ X \wedge Some\ o_2 = d_\kappa\ y \wedge o_2 = \omega\nu\ Y$
by *auto*
have $\exists\ r\ .\ Some\ r = d_1\ (\boldsymbol{\lambda}\ z\ .\ makeo\ (\lambda\ w\ s\ .\ d_\kappa\ (z^P) = Some\ o_1))$
using *propex$_1$* **by** *auto*
then obtain r **where** *4*:
$Some\ r = d_1\ (\boldsymbol{\lambda}\ z\ .\ makeo\ (\lambda\ w\ s\ .\ d_\kappa\ (z^P) = Some\ o_1))$
by *auto*
hence *5*: $r = (\lambda u\ s\ w.\ \exists\ x\ .\ \nu\upsilon\ x = u \wedge Some\ x = Some\ o_1)$
unfolding *lambdabinder1-def d$_1$-def d$_\kappa$-proper*
apply *transfer*
by *simp*
have $[\Box(\forall\ F.\ (\!|F,x|\!) \equiv (\!|F,y|\!))\ in\ v]$
using *1* **using** *ConjE* **by** *blast*
hence *6*: $\forall\ v\ F\ .\ [(\!|F,x|\!)\ in\ v] \longleftrightarrow [(\!|F,y|\!)\ in\ v]$
using *BoxE EquivE AllE* **by** *fast*
hence *7*: $\forall\ v\ .\ (o_1 \in ex1\ r\ v) = (o_2 \in ex1\ r\ v)$
using *2 4* **unfolding** *valid-in-def*
by (*metis 3 6 d$_1$.rep-eq d$_\kappa$-inject d$_\kappa$-proper ex1-def evalo-inverse exe1.rep-eq mem-Collect-eq option.sel rep-proper-id νκ-proper valid-in.abs-eq*)
have $o_1 \in ex1\ r\ v$
using *5 3* **unfolding** *ex1-def* **by** (*simp add*: *meta-aux*)
hence $o_2 \in ex1\ r\ v$
using *7* **by** *auto*
hence $o_1 = o_2$
unfolding *ex1-def 5* **using** *3* **by** (*auto simp*: *meta-aux*)
thus *?thesis*
using *3* **by** *auto*
qed
lemma *Eq$_E$S[meta-subst]*:
$[x =_E y\ in\ v] = (\exists\ o_1\ X\ o_2.\ Some\ o_1 = d_\kappa\ x \wedge Some\ o_2 = d_\kappa\ y$
$\wedge\ o_1 = o_2 \wedge o_1 = \omega\nu\ X)$
using *Eq$_E$I Eq$_E$E* **by** *blast*

A.5.15.2. Individuals

lemma *Eqκ I[meta-intro]*:
assumes $\exists\ o_1\ o_2.\ Some\ o_1 = d_\kappa\ x \wedge Some\ o_2 = d_\kappa\ y \wedge o_1 = o_2$
shows $[x =_\kappa y\ in\ v]$
proof −

have $x = y$ **using** *assms* d_κ*-inject* **by** *meson*
moreover have $[x =_\kappa x \ in \ v]$
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*κ*-prop*:
assumes $[x =_\kappa y \ in \ v]$
shows $[\varphi \ x \ in \ v] = [\varphi \ y \ in \ v]$
proof −
have $[x =_E y \vee (\!|A!,x|\!) \ \& \ (\!|A!,y|\!) \ \& \ \Box(\forall \ F. \ \{\!|x,F|\!\} \equiv \{\!|y,F|\!\}) \ in \ v]$
using *assms* **unfolding** *basic-identity*$_\kappa$*-def* **by** *simp*
moreover {
assume $[x =_E y \ in \ v]$
hence $(\exists \ o_1 \ o_2. \ Some \ o_1 = d_\kappa \ x \wedge Some \ o_2 = d_\kappa \ y \wedge o_1 = o_2)$
using *Eq*$_E$*E* **by** *fast*
}
moreover {
assume *1*: $[(\!|A!,x|\!) \ \& \ (\!|A!,y|\!) \ \& \ \Box(\forall \ F. \ \{\!|x,F|\!\} \equiv \{\!|y,F|\!\}) \ in \ v]$
hence *2*: $(\exists \ o_1 \ o_2 \ X \ Y. \ Some \ o_1 = d_\kappa \ x \wedge Some \ o_2 = d_\kappa \ y$
$\wedge \ o_1 = \alpha\nu \ X \wedge o_2 = \alpha\nu \ Y)$
using *AbsE ConjE* **by** *meson*
moreover then obtain $o_1 \ o_2 \ X \ Y$ **where** *3*:
$Some \ o_1 = d_\kappa \ x \wedge Some \ o_2 = d_\kappa \ y \wedge o_1 = \alpha\nu \ X \wedge o_2 = \alpha\nu \ Y$
by *auto*
moreover have *4*: $[\Box(\forall \ F. \ \{\!|x,F|\!\} \equiv \{\!|y,F|\!\}) \ in \ v]$
using *1 ConjE* **by** *blast*
hence *6*: $\forall \ v \ F \ . \ [\{\!|x,F|\!\} \ in \ v] \longleftrightarrow [\{\!|y,F|\!\} \ in \ v]$
using *BoxE AllE EquivE* **by** *fast*
hence *7*: $\forall v \ r. \ (\exists \ o_1. \ Some \ o_1 = d_\kappa \ x \wedge o_1 \in en \ r)$
$= (\exists \ o_1. \ Some \ o_1 = d_\kappa \ y \wedge o_1 \in en \ r)$
apply − **apply** *meta-solver*
using *propex*$_1$ *d*$_1$*-inject* **apply** *simp*
apply *transfer* **by** *simp*
hence *8*: $\forall \ r. \ (o_1 \in en \ r) = (o_2 \in en \ r)$
using *3* d_κ*-inject* d_κ*-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 makeΠ_1*-cases*)
hence $(o_1 \in \{ \ x \ . \ o_1 = x \ \}) = (o_2 \in \{ \ x \ . \ o_1 = x \ \})$
by *metis*
hence $o_1 = o_2$ **by** *simp*
hence $(\exists \ o_1 \ o_2. \ Some \ o_1 = d_\kappa \ x \wedge Some \ o_2 = d_\kappa \ y \wedge o_1 = o_2)$
using *3* **by** *auto*
}
ultimately have $x = y$
using *DisjS* **using** *Semantics.*d_κ*-inject* **by** *auto*
thus $(v \models (\varphi \ x)) = (v \models (\varphi \ y))$ **by** *simp*
qed
lemma *Eq*κ*E*[*meta-elim*]:
assumes $[x =_\kappa y \ in \ v]$
shows $\exists \ o_1 \ o_2. \ Some \ o_1 = d_\kappa \ x \wedge Some \ o_2 = d_\kappa \ y \wedge o_1 = o_2$
proof −
have $\forall \ \varphi \ . \ (v \models \varphi \ x) = (v \models \varphi \ y)$
using *assms Eq*κ*-prop* **by** *blast*

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

A.5.15.3. One-Place Relations

lemma Eq_1I[*meta-intro*]: $F = G \Longrightarrow [F =_1 G\ in\ v]$
unfolding *basic-identity*$_1$*-def*
apply (*rule BoxI*, *rule AllI*, *rule EquivI*)
by *simp*
lemma Eq_1E[*meta-elim*]: $[F =_1 G\ in\ v] \Longrightarrow F = G$
unfolding *basic-identity*$_1$*-def*
apply (*drule BoxE*, *drule-tac* $x{=}(\alpha\nu\ \{\ F\ \})$ **in** *AllE*, *drule EquivE*)
apply (*simp add*: *Semantics.T2*)
unfolding *en-def* d_κ*-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 Eq_1S[*meta-subst*]: $[F =_1 G\ in\ v] = (F = G)$
using Eq_1I Eq_1E **by** *auto*
lemma Eq_1*-prop*: $[F =_1 G\ in\ v] \Longrightarrow [\varphi\ F\ in\ v] = [\varphi\ G\ in\ v]$
using Eq_1E **by** *blast*

A.5.15.4. Two-Place Relations

lemma Eq_2I[*meta-intro*]: $F = G \Longrightarrow [F =_2 G\ in\ v]$
unfolding *basic-identity*$_2$*-def*
apply (*rule AllI*, *rule ConjI*, (*subst* Eq_1S)+)
by *simp*
lemma Eq_2E[*meta-elim*]: $[F =_2 G\ in\ v] \Longrightarrow F = G$
proof −
assume $[F =_2 G\ in\ v]$
hence *1*: $[\forall\ x.\ (\boldsymbol{\lambda}y.\ (\!|F,x^P,y^P|\!)) =_1 (\boldsymbol{\lambda}y.\ (\!|G,x^P,y^P|\!))\ in\ v]$
unfolding *basic-identity*$_2$*-def*
apply − **apply** *meta-solver* **by** *auto*
{
fix *u v s w*
obtain *x* **where** *x-def*: $\nu\upsilon\ x = v$ **by** (*metis* $\nu\upsilon$*-surj surj-def*)
obtain *a* **where** *a-def*:
$a = (\lambda u\ s\ w.\ \exists xa.\ \nu\upsilon\ xa = u \wedge eval\Pi_2\ F\ (\nu\upsilon\ x)\ (\nu\upsilon\ xa)\ s\ w)$
by *auto*
obtain *b* **where** *b-def*:
$b = (\lambda u\ s\ w.\ \exists xa.\ \nu\upsilon\ xa = u \wedge eval\Pi_2\ G\ (\nu\upsilon\ x)\ (\nu\upsilon\ xa)\ s\ w)$
by *auto*
have $a = b$ **unfolding** *a-def b-def*

```
      using 1 apply − apply meta-solver
      by (auto simp: meta-defs meta-aux makeΠ1-inject)
    hence a u s w = b u s w by auto
    hence (evalΠ2 F (νυ x) u s w) = (evalΠ2 G (νυ x) u s w)
      unfolding a-def b-def
      by (metis (no-types, hide-lams) νυ-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] ⟹ [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 ⟹ [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] ⟹ F = G
proof −

  assume [F =3 G in v]
  hence 1: [∀ x y. (λz. (|F,x^P,y^P,z^P|)) =1 (λz. (|G,x^P,y^P,z^P|)) in v]
    unfolding basic-identity3-def
    apply − apply meta-solver by auto
  {
    fix u v r s w
    obtain x where x-def: νυ x = v by (metis νυ-surj surj-def)
    obtain y where y-def: νυ y = r by (metis νυ-surj surj-def)
    obtain a where a-def:
      a = (λu s w. ∃xa. νυ xa = u ∧ evalΠ3 F (νυ x) (νυ y) (νυ xa) s w)
      by auto
    obtain b where b-def:
      b = (λu s w. ∃xa. νυ xa = u ∧ evalΠ3 G (νυ x) (νυ y) (νυ 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 (νυ x) (νυ y) u s w) = (evalΠ3 G (νυ x) (νυ y) u s w)
      unfolding a-def b-def
      by (metis (no-types, hide-lams) νυ-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] ⟹ [φ F in v] = [φ G in v]
  using Eq3E by blast
```

A.5.15.6. Propositions

lemma Eq_0I[*meta-intro*]: $x = y \Longrightarrow [x =_0 y \ in \ v]$
unfolding $basic\text{-}identity_0\text{-}def$ **by** (*simp add*: Eq_1S)
lemma Eq_0E[*meta-elim*]: $[F =_0 G \ in \ v] \Longrightarrow F = G$
proof −
assume $[F =_0 G \ in \ v]$
hence $[(\boldsymbol{\lambda} y.\ F) =_1 (\boldsymbol{\lambda} y.\ G) \ in \ v]$
unfolding $basic\text{-}identity_0\text{-}def$ **by** *simp*
hence $(\boldsymbol{\lambda} y.\ F) = (\boldsymbol{\lambda} y.\ G)$
using Eq_1S **by** *simp*
hence $(\lambda u\ s\ w.\ (\exists x.\ \nu\upsilon\ x = u) \wedge evalo\ F\ s\ w)$
$= (\lambda u\ s\ w.\ (\exists x.\ \nu\upsilon\ x = u) \wedge evalo\ G\ s\ w)$
apply (*simp add*: *meta-defs meta-aux*)
by (*metis* (*no-types*, *lifting*) *UNIV-I* $make\Pi_1\text{-}inverse$)
hence $\bigwedge s\ w.(evalo\ F\ s\ w) = (evalo\ G\ s\ w)$
by *metis*
hence $(evalo\ F) = (evalo\ G)$ **by** *blast*
thus $F = G$
by (*metis evalo-inverse*)
qed
lemma Eq_0S[*meta-subst*]: $[F =_0 G \ in \ v] = (F = G)$
using Eq_0I Eq_0E **by** *auto*
lemma $Eq_0\text{-}prop$: $[F =_0 G \ in \ v] \Longrightarrow [\varphi \ F \ in \ v] = [\varphi \ G \ in \ v]$
using Eq_0E **by** *blast*

end

A.6. General Identity

Remark A.15. *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*:
$w \models x = y \Longrightarrow w \models \varphi\ x \Longrightarrow w \models \varphi\ y$
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 κ :: *identifiable*
begin
  definition *identity-*κ **where** *identity-*$\kappa \equiv$ *basic-identity*$_\kappa$
  instance proof
    fix $x\ y :: \kappa$ **and** $w\ \varphi$
    show $[x = y\ in\ w] \Longrightarrow [\varphi\ x\ in\ w] \Longrightarrow [\varphi\ y\ in\ w]$
      unfolding *identity-*κ*-def*
      using *MetaSolver.Eq*κ*-prop* **..**
  qed
end

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

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

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

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

end

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

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

A.6.3. New Identity Definitions

Remark A.16. *The basic definitions of identity used the type specific quantifiers and identities. We now introduce equivalent definitions that use the general identity and general quantifiers.*

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

A.7. The Axioms of Principia Metaphysica

Remark A.17. *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 A.18. *The special syntax* [[-]] *is introduced for axioms. This allows to formulate special rules resembling the concepts of closures in PM. To simplify the instantiation of axioms later, special attributes are introduced to automatically resolve the special axiom syntax. Necessitation averse axioms are stated with the syntax for actual validity* [-].

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

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

A.7.1. Closures

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

attribute-setup *axiom-instance* = ⟪
Scan.succeed (*Thm.rule-attribute* []
(*fn* - => *fn thm* => *thm RS* @{*thm axiom-instance*}))
⟫

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

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

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

attribute-setup *axiom-universal* = ⟪
Scan.succeed (*Thm.rule-attribute* []
(*fn* - => *fn thm* => *thm RS* @{*thm closures-universal*}))
⟫

A.7.2. Axioms for Negations and Conditionals

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

A.7.3. Axioms of Identity

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

A.7.4. Axioms of Quantification

Remark A.19. *The axioms of quantification differ slightly 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. 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 the datatype 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 on datatype κ holds for all denoting terms ($\forall\ \alpha.\ \varphi\ (\alpha^P)$) then the formula holds for an individual $\varphi\ \alpha$, 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. Therefore 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.*

TODO A.1. *Reformulate the above more precisely.*

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

```
    unfolding axiom-def using ImplI by blast
  qed
lemma cqt-3[axiom]:
  [[(∀ α. φ α → ψ α) → ((∀ α. φ α) → (∀ α. ψ α))]]
  by axiom-meta-solver
lemma cqt-4[axiom]:
  [[φ → (∀ α. φ)]]
  by axiom-meta-solver

inductive SimpleExOrEnc
  where SimpleExOrEnc (λ x . (|F,x|))
      | SimpleExOrEnc (λ x . (|F,x,y|))
      | SimpleExOrEnc (λ x . (|F,y,x|))
      | SimpleExOrEnc (λ x . (|F,x,y,z|))
      | SimpleExOrEnc (λ x . (|F,y,x,z|))
      | SimpleExOrEnc (λ x . (|F,y,z,x|))
      | SimpleExOrEnc (λ x . {|x,F|})

lemma cqt-5[axiom]:
  assumes SimpleExOrEnc ψ
  shows [[(ψ (ιx . φ x)) → (∃ α. (α^P) = (ιx . φ x))]]
  proof −
    have ∀ w . ([(ψ (ιx . φ x)) in w] ⟶ (∃ o₁ . Some o₁ = d_κ (ιx . φ x)))
      using assms apply induct by (meta-solver;metis)+
   thus ?thesis
    apply − unfolding identity-κ-def
    apply axiom-meta-solver
    using d_κ-proper by auto
  qed

lemma cqt-5-mod[axiom]:
  assumes SimpleExOrEnc ψ
  shows [[ψ τ → (∃ α . (α^P) = τ)]]
  proof −
    have ∀ w . ([(ψ τ) in w] ⟶ (∃ o₁ . Some o₁ = d_κ τ))
      using assms apply induct by (meta-solver;metis)+
    thus ?thesis
      apply − unfolding identity-κ-def
      apply axiom-meta-solver
      using d_κ-proper by auto
  qed
```

A.7.5. Axioms of Actuality

Remark A.20. *The necessitation averse axiom of actuality is stated to be actually true; for the statement as a proper axiom (for which necessitation would be allowed) nitpick can find a counter-model as desired.*

```
  lemma logic-actual[axiom]: [(𝒜φ) ≡ φ]
    by axiom-meta-solver
  lemma [[(𝒜φ) ≡ φ]]
    nitpick[user-axioms, expect = genuine, card = 1, card i = 2]
    oops — Counter-model by nitpick

  lemma logic-actual-nec-1[axiom]:
    [[𝒜¬φ ≡ ¬𝒜φ]]
    by axiom-meta-solver
```

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

A.7.7. Axioms of Necessity and Actuality

lemma *qml-act-1*[*axiom*]:
[[$\mathcal{A}\varphi \to \Box\mathcal{A}\varphi$]]
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*:
$\exists\, o_1\ o_2.$ *Some* $o_1 = d_\kappa\ (x^P) \wedge$ *Some* $o_2 = d_\kappa\ (\iota x.\ \varphi\ x) \wedge o_1 = o_2$
apply − **unfolding** *identity-κ-def* **by** *meta-solver*
then obtain $o_1\ o_2$ **where** *2*:
Some $o_1 = d_\kappa\ (x^P) \wedge$ *Some* $o_2 = d_\kappa\ (\iota x.\ \varphi\ x) \wedge o_1 = o_2$
by *auto*
hence *3*:
$(\exists\ x\ .((w_0 \models \varphi\ x) \wedge (\forall\, y.\ (w_0 \models \varphi\ y) \longrightarrow y = x)))$
$\wedge\ d_\kappa\ (\iota x.\ \varphi\ x) =$ *Some* (*THE x*. $(w_0 \models \varphi\ x)$)
using *D3* **by** (*metis option.distinct*(*1*))

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

A.7.9. Axioms for Complex Relation Terms

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

lemma *lambda-predicates-2-1*[*axiom*]:
assumes *IsProperInX* φ
shows $[[(\!|\boldsymbol{\lambda}\ x\ .\ \varphi\ (x^P),\ x^P|\!) \equiv \varphi\ (x^P)]]$
apply *axiom-meta-solver*
using *D5-1*[*OF assms*] d_κ*-proper propex*$_1$
by *metis*

lemma *lambda-predicates-2-2*[*axiom*]:
assumes *IsProperInXY* φ
shows $[[(\!|(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ \varphi\ (x^P)\ (y^P))),\ x^P,\ y^P|\!) \equiv \varphi\ (x^P)\ (y^P)]]$
apply *axiom-meta-solver*
using *D5-2*[*OF assms*] d_κ*-proper propex*$_2$
by *metis*

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

assumes *IsProperInXYZ* φ
shows $[[(\!|(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \varphi\ (x^P)\ (y^P)\ (z^P))),x^P,y^P,z^P|\!)\equiv\varphi\ (x^P)\ (y^P)\ (z^P)]]$
proof −
have $[[(\!|(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \varphi\ (x^P)\ (y^P)\ (z^P))),x^P,y^P,z^P|\!)\rightarrow\varphi\ (x^P)\ (y^P)\ (z^P)]]$
apply *axiom-meta-solver* **using** *D5-3*[*OF assms*] **by** *auto*
moreover have
$[[\varphi\ (x^P)\ (y^P)\ (z^P)\rightarrow(\!|(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \varphi\ (x^P)\ (y^P)\ (z^P))),x^P,y^P,z^P|\!)]]$
apply *axiom-meta-solver*
using *D5-3*[*OF assms*] d_κ*-proper propex*$_3$
by (*metis* (*no-types*, *lifting*))
ultimately show *?thesis* **unfolding** *axiom-def equiv-def ConjS* **by** *blast*
qed

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

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

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

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

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

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

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

lemma *lambda-predicates-4-3*[*axiom*]:
assumes $\bigwedge x.[(\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x))\ in\ v]$
shows $[[(\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \chi\ (\boldsymbol{\iota}x.\ \varphi\ x)\ x\ y\ z)) = (\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z\ .\ \chi\ (\boldsymbol{\iota}x.\ \psi\ x)\ x\ y\ z))]]$
unfolding *axiom-def identity-*Π_3*-def* **apply** − **apply** (*rule allI*; *rule* Eq_3I)
using *TheEqI*[*OF assms*[*THEN ActualE*, *THEN EquivE*]] **by** *auto*

A.7.10. Axioms of Encoding

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

end

A.8. Definitions

Various definitions needed throughout PLM.

A.8.1. Property Negations

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

named-theorems *propnot-defs*
declare $propnot_0$*-def*[*propnot-defs*] $propnot_1$*-def*[*propnot-defs*]
$propnot_2$*-def*[*propnot-defs*] $propnot_3$*-def*[*propnot-defs*]

A.8.2. Noncontingent and Contingent Relations

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

named-theorems *Necessary-defs*
declare $Necessary_0$*-def*[*Necessary-defs*] $Necessary_1$*-def*[*Necessary-defs*]
$Necessary_2$*-def*[*Necessary-defs*] $Necessary_3$*-def*[*Necessary-defs*]

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

named-theorems *Impossible-defs*
declare $Impossible_0$*-def*[*Impossible-defs*] $Impossible_1$*-def*[*Impossible-defs*]
$Impossible_2$*-def*[*Impossible-defs*] $Impossible_3$*-def*[*Impossible-defs*]

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

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

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

A.8.3. Null and Universal Objects

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

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

A.8.4. Propositional Properties

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

A.8.5. Indiscriminate Properties

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

A.8.6. Miscellaneous

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

A.9. The Deductive System PLM

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

locale *PLM* = *Axioms*
begin

A.9.1. Automatic Solver

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

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

A.9.2. Modus Ponens

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

A.9.3. Axioms

interpretation *Axioms* .
declare *axiom*[*PLM*]

A.9.4. (Modally Strict) Proofs and Derivations

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

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

A.9.5. GEN and RN

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

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

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

A.9.6. Negations and Conditionals

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

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

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

lemma *useful-tautologies-1*[*PLM*]:
$[\neg\neg\varphi \rightarrow \varphi\ in\ v]$
by (*meson pl-1 pl-3 ded-thm-cor-3 ded-thm-cor-4 axiom-instance*)
lemma *useful-tautologies-2*[*PLM*]:
$[\varphi \rightarrow \neg\neg\varphi\ in\ v]$
by (*meson pl-1 pl-3 ded-thm-cor-3 useful-tautologies-1*
vdash-properties-10 axiom-instance)
lemma *useful-tautologies-3*[*PLM*]:

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

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

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

lemma *reductio-aa-1*[*PLM*]:
$[\![[\neg\varphi\ in\ v] \Longrightarrow [\neg\psi\ in\ v];\ [\neg\varphi\ in\ v] \Longrightarrow [\psi\ in\ v]]\!] \Longrightarrow [\varphi\ in\ v]$
using *CP modus-tollens-2 useful-tautologies-1*
vdash-properties-10 **by** *blast*
lemma *reductio-aa-2*[*PLM*]:
$[\![[\varphi\ in\ v] \Longrightarrow [\neg\psi\ in\ v];\ [\varphi\ in\ v] \Longrightarrow [\psi\ in\ v]]\!] \Longrightarrow [\neg\varphi\ in\ v]$
by (*meson contraposition-1 reductio-aa-1*)
lemma *reductio-aa-3*[*PLM*]:
$[\![[\neg\varphi \to \neg\psi\ in\ v];\ [\neg\varphi \to \psi\ in\ v]]\!] \Longrightarrow [\varphi\ in\ v]$
using *reductio-aa-1 vdash-properties-10* **by** *blast*
lemma *reductio-aa-4*[*PLM*]:
$[\![[\varphi \to \neg\psi\ in\ v];\ [\varphi \to \psi\ in\ v]]\!] \Longrightarrow [\neg\varphi\ in\ v]$
using *reductio-aa-2 vdash-properties-10* **by** *blast*

lemma *raa-cor-1*[*PLM*]:

$[\![[\varphi\ in\ v];\ [\neg\psi\ in\ v] \Longrightarrow [\neg\varphi\ in\ v]]\!] \Longrightarrow ([\varphi\ in\ v] \Longrightarrow [\psi\ in\ v])$
using *reductio-aa-1 vdash-properties-9* **by** *blast*

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

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

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

Remark A.21. *The classical introduction and elimination rules are proven earlier than in PM. The statements proven so far are sufficient for the proofs and using these rules Isabelle can prove the tautologies automatically.*

lemma *intro-elim-1*[*PLM*]:
$[\![[\varphi\ in\ v];\ [\psi\ in\ v]]\!] \Longrightarrow [\varphi\ \&\ \psi\ in\ v]$
unfolding *conj-def* **using** *ded-thm-cor-4 if-p-then-p modus-tollens-2* **by** *blast*

lemmas $\&I$ = *intro-elim-1*

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

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

lemmas $\&E$ = *intro-elim-2-a intro-elim-2-b*

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

lemma *intro-elim-3-b*[*PLM*]:
$[\psi\ in\ v] \Longrightarrow [\varphi \vee \psi\ in\ v]$
by (*simp only*: *disj-def vdash-properties-9*)

lemmas $\vee I$ = *intro-elim-3-a intro-elim-3-b*

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

lemma *intro-elim-4-b*[*PLM*]:
$[\![[\varphi \vee \psi\ in\ v];\ [\neg\varphi\ in\ v]]\!] \Longrightarrow [\psi\ in\ v]$
unfolding *disj-def* **using** *vdash-properties-10* **by** *blast*

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

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

lemma *intro-elim-4-e*[*PLM*]:
$[\![[\varphi \vee \psi\ in\ v];\ [\varphi \equiv \chi\ in\ v];\ [\psi \equiv \Theta\ in\ v]]\!] \Longrightarrow [\chi \vee \Theta\ in\ v]$
unfolding *equiv-def* **using** $\&E(1)$ *intro-elim-4-d* **by** *blast*

lemmas $\vee E$ = *intro-elim-4-a intro-elim-4-b intro-elim-4-c intro-elim-4-d*

lemma *intro-elim-5*[*PLM*]:
$[\![[\varphi \rightarrow \psi\ in\ v];\ [\psi \rightarrow \varphi\ in\ v]]\!] \Longrightarrow [\varphi \equiv \psi\ in\ v]$
by (*simp only*: *equiv-def* $\&I$)

lemmas $\equiv I$ = *intro-elim-5*

lemma *intro-elim-6-a*[*PLM*]:
$[\![[\varphi \equiv \psi\ in\ v];\ [\varphi\ in\ v]]\!] \Longrightarrow [\psi\ in\ v]$
unfolding *equiv-def* **using** $\&E(1)$ *vdash-properties-10* **by** *blast*

lemma *intro-elim-6-b*[*PLM*]:

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

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

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

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

private lemma *ConjI*[*PLM-intro*]:
$[\![[\varphi \ in \ v]; [\psi \ in \ v]]\!] \Longrightarrow [\varphi \ \& \ \psi \ in \ v]$
using &I **by** *blast*
private lemma *ConjE*[*PLM-elim*,*PLM-dest*]:

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

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

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

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

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

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

lemma *oth-class-taut-1-a*[*PLM*]:
$[\neg(\varphi \ \& \ \neg\varphi) \ in \ v]$
by *PLM-solver*
lemma *oth-class-taut-1-b*[*PLM*]:
$[\neg(\varphi \equiv \neg\varphi) \ in \ v]$
by *PLM-solver*
lemma *oth-class-taut-2*[*PLM*]:

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

by *PLM-solver*
lemma *oth-class-taut-5-j*[*PLM*]:
[(¬(φ ≡ ψ)) ≡ ((φ & ¬ψ) ∨ (¬φ & ψ)) *in v*]
by *PLM-solver*
lemma *oth-class-taut-5-k*[*PLM*]:
[(φ → ψ) ≡ (¬φ ∨ ψ) *in v*]
by *PLM-solver*

lemma *oth-class-taut-6-a*[*PLM*]:
[(φ & ψ) ≡ ¬(¬φ ∨ ¬ψ) *in v*]
by *PLM-solver*
lemma *oth-class-taut-6-b*[*PLM*]:
[(φ ∨ ψ) ≡ ¬(¬φ & ¬ψ) *in v*]
by *PLM-solver*
lemma *oth-class-taut-6-c*[*PLM*]:
[¬(φ & ψ) ≡ (¬φ ∨ ¬ψ) *in v*]
by *PLM-solver*
lemma *oth-class-taut-6-d*[*PLM*]:
[¬(φ ∨ ψ) ≡ (¬φ & ¬ψ) *in v*]
by *PLM-solver*

lemma *oth-class-taut-7-a*[*PLM*]:
[(φ & (ψ ∨ χ)) ≡ ((φ & ψ) ∨ (φ & χ)) *in v*]
by *PLM-solver*
lemma *oth-class-taut-7-b*[*PLM*]:
[(φ ∨ (ψ & χ)) ≡ ((φ ∨ ψ) & (φ ∨ χ)) *in v*]
by *PLM-solver*

lemma *oth-class-taut-8-a*[*PLM*]:
[((φ & ψ) → χ) → (φ → (ψ → χ)) *in v*]
by *PLM-solver*
lemma *oth-class-taut-8-b*[*PLM*]:
[(φ → (ψ → χ)) → ((φ & ψ) → χ) *in v*]
by *PLM-solver*

lemma *oth-class-taut-9-a*[*PLM*]:
[(φ & ψ) → φ *in v*]
by *PLM-solver*
lemma *oth-class-taut-9-b*[*PLM*]:
[(φ & ψ) → ψ *in v*]
by *PLM-solver*

lemma *oth-class-taut-10-a*[*PLM*]:
[φ → (ψ → (φ & ψ)) *in v*]
by *PLM-solver*
lemma *oth-class-taut-10-b*[*PLM*]:
[(φ → (ψ → χ)) ≡ (ψ → (φ → χ)) *in v*]
by *PLM-solver*
lemma *oth-class-taut-10-c*[*PLM*]:
[(φ → ψ) → ((φ → χ) → (φ → (ψ & χ))) *in v*]
by *PLM-solver*
lemma *oth-class-taut-10-d*[*PLM*]:
[(φ → χ) → ((ψ → χ) → ((φ ∨ ψ) → χ)) *in v*]
by *PLM-solver*
lemma *oth-class-taut-10-e*[*PLM*]:
[(φ → ψ) → ((χ → Θ) → ((φ & χ) → (ψ & Θ))) *in v*]
by *PLM-solver*
lemma *oth-class-taut-10-f*[*PLM*]:

```
  [((φ & ψ) ≡ (φ & χ)) ≡ (φ → (ψ ≡ χ)) in v]
  by PLM-solver
lemma oth-class-taut-10-g[PLM]:
  [((φ & ψ) ≡ (χ & ψ)) ≡ (ψ → (φ ≡ χ)) in v]
  by PLM-solver

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

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

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

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

attribute-setup conj2 = ⟪
  Scan.succeed (Thm.rule-attribute []
    (fn - => fn thm => thm RS @{thm &E(2)}))
⟫

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

A.9.7. Identity

Remark A.22. *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::Π₁) = F in v]
  unfolding identity-defs by PLM-solver
lemma id-eq-prop-prop-2[PLM]:
  [((F::Π₁) = G) → (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::Π₁) = G) & (G = H)) → (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*::Π$_2$) = *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*::Π$_3$) = *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*::Π$_2$) = *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*::Π$_3$) = *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*::Π$_0$) = *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*::Π$_0$) = *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*::Π$_0$) = *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* . ((*x*^{P}) =$_E$ (*y*^{P})) ≡ (⌊*O*!,*x*^{P}⌋ & ⌊*O*!,*y*^{P}⌋
& □(∀ *F* . (⌊*F*,*x*^{P}⌋ ≡ ⌊*F*,*y*^{P}⌋))) *in v*]
unfolding *identity*$_E$*-infix-def identity*$_E$*-def*
apply (*rule lambda-predicates-2-2*[*axiom-universal*, *axiom-universal*, *axiom-instance*])
by *show-proper*
moreover have [∃ α . (α^P) = *x in v*]
apply (*rule cqt-5-mod*[**where** ψ=λ *x* . *x* =$_E$ *y*, *axiom-instance*,*deduction*])
unfolding *identity*$_E$*-infix-def*
apply (*rule SimpleExOrEnc.intros*)
using *1* **unfolding** *identity*$_E$*-infix-def* **by** *auto*
moreover have [∃ β . (β^P) = *y in v*]
apply (*rule cqt-5-mod*[**where** ψ=λ *y* . *x* =$_E$ *y*,*axiom-instance*,*deduction*])
unfolding *identity*$_E$*-infix-def*
apply (*rule SimpleExOrEnc.intros*) **using** *1*
unfolding *identity*$_E$*-infix-def* **by** *auto*
ultimately have [(*x* =$_E$ *y*) ≡ (⌊*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* . ((*x*^{P}) =$_E$ (*y*^{P})) ≡ (⌊*O*!,*x*^{P}⌋ & ⌊*O*!,*y*^{P}⌋

```
        & □(∀ F . (|F,x^P|) ≡ (|F,y^P|))) in v]
    unfolding identity_E-def identity_E-infix-def
    apply (rule lambda-predicates-2-2[axiom-universal, axiom-universal, axiom-instance])
    by show-proper
  moreover have [∃ α . (α^P) = x in v]
    apply (rule cqt-5-mod[where ψ=λ x . (|O!,x|),axiom-instance,deduction])
    apply (rule SimpleExOrEnc.intros)
    using 1[conj1,conj1] by auto
  moreover have [∃ β . (β^P) = y in v]
    apply (rule cqt-5-mod[where ψ=λ y . (|O!,y|),axiom-instance,deduction])
     apply (rule SimpleExOrEnc.intros)
    using 1[conj1,conj2] by auto
  ultimately have [(x =_E y) ≡ ((|O!,x|) & (|O!,y|)
                & □(∀ F . (|F,x|) ≡ (|F,y|))) in v]
  using cqt-1-κ[axiom-instance,deduction, deduction] by meson
  thus [(x =_E y) in v] using 1 ≡E(2) by blast
 qed
lemma eq-E-simple-2[PLM]:
 [(x =_E y) → (x = y) in v]
 unfolding identity-defs by PLM-solver
lemma eq-E-simple-3[PLM]:
 [(x = y) ≡ (((|O!,x|) & (|O!,y|) & □(∀ F . (|F,x|) ≡ (|F,y|)))
          ∨ ((|A!,x|) & (|A!,y|) & □(∀ F. {|x,F|} ≡ {|y,F|}))) in v]
 using eq-E-simple-1
 apply − unfolding identity-defs
 by PLM-solver

lemma id-eq-obj-1[PLM]: [(x^P) = (x^P) in v]
 proof −
  have [(◇(|E!, x^P|)) ∨ (¬◇(|E!, x^P|)) in v]
    using PLM.oth-class-taut-2 by simp
  hence [(◇(|E!, x^P|)) in v] ∨ [(¬◇(|E!, x^P|)) in v]
    using CP ∨E(1) by blast
  moreover {
    assume [(◇(|E!, x^P|)) in v]
    hence [(|λx. ◇(|E!,x^P|),x^P|) in v]
      apply (rule lambda-predicates-2-1[axiom-instance, equiv-rl, rotated])
      by show-proper
    hence [(|λx. ◇(|E!,x^P|),x^P|) & (|λx. ◇(|E!,x^P|),x^P|)
          & □(∀ F. (|F,x^P|) ≡ (|F,x^P|)) in v]
      apply − by PLM-solver
    hence [(x^P) =_E (x^P) in v]
      using eq-E-simple-1[equiv-rl] unfolding Ordinary-def by fast
  }
  moreover {
    assume [(¬◇(|E!, x^P|)) in v]
    hence [(|λx. ¬◇(|E!,x^P|),x^P|) in v]
      apply (rule lambda-predicates-2-1[axiom-instance, equiv-rl, rotated])
      by show-proper
    hence [(|λx. ¬◇(|E!,x^P|),x^P|) & (|λx. ¬◇(|E!,x^P|),x^P|)
          & □(∀ F. {|x^P,F|} ≡ {|x^P,F|}) in v]
      apply − by PLM-solver
  }
  ultimately show ?thesis unfolding identity-defs Ordinary-def Abstract-def
    using ∨I by blast
 qed
lemma id-eq-obj-2[PLM]:
 [((x^P) = (y^P)) → ((y^P) = (x^P)) in v]
```

by (*meson l-identity*[*axiom-instance*] *id-eq-obj-1 CP ded-thm-cor-3*)
lemma *id-eq-obj-3*[*PLM*]:
[$((x^P) = (y^P))$ **&** $((y^P) = (z^P)) \rightarrow ((x^P) = (z^P))$ *in* v]
by (*metis l-identity*[*axiom-instance*] *ded-thm-cor-4 CP* **&***E*)
end

Remark A.23. *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 ν :: *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-*ν*-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-*ν*-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-*ν*-def*)
qed
end

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

instantiation Π_1 :: *id-eq*
begin
instance proof
fix $x :: \Pi_1$ **and** v
show [$x = x$ *in* v]
using *PLM.id-eq-prop-prop-1* **.**
next
fix $x\ y :: \Pi_1$ **and** v

```
    show [x = y → y = x in v]
      using PLM.id-eq-prop-prop-2 .
  next
    fix x y z :: Π1 and v
    show [((x = y) & (y = z)) → x = z in v]
      using PLM.id-eq-prop-prop-3 .
  qed
end

instantiation Π2 :: id-eq
begin
  instance proof
    fix x :: Π2 and v
    show [x = x in v]
      using PLM.id-eq-prop-prop-4-a .
  next
    fix x y :: Π2 and v
    show [x = y → y = x in v]
      using PLM.id-eq-prop-prop-5-a .
  next
    fix x y z :: Π2 and v
    show [((x = y) & (y = z)) → x = z in v]
      using PLM.id-eq-prop-prop-6-a .
  qed
end

instantiation Π3 :: id-eq
begin
  instance proof
    fix x :: Π3 and v
    show [x = x in v]
      using PLM.id-eq-prop-prop-4-b .
  next
    fix x y :: Π3 and v
    show [x = y → y = x in v]
      using PLM.id-eq-prop-prop-5-b .
  next
    fix x y z :: Π3 and v
    show [((x = y) & (y = z)) → x = z in v]
      using PLM.id-eq-prop-prop-6-b .
  qed
end

context PLM
begin
  lemma id-eq-1[PLM]:
    [(x::'a::id-eq) = x in v]
    using id-eq-1 .
  lemma id-eq-2[PLM]:
    [((x::'a::id-eq) = y) → (y = x) in v]
    using id-eq-2 .
  lemma id-eq-3[PLM]:
    [((x::'a::id-eq) = y) & (y = z) → (x = z) in v]
    using id-eq-3 .

  attribute-setup eq-sym = ⟪
    Scan.succeed (Thm.rule-attribute []
      (fn - => fn thm => thm RS @{thm id-eq-2[deduction]}))
```

⟫

lemma *all-self-eq-1*[*PLM*]:
[□(∀ α :: 'a::*id-eq* . α = α) *in v*]
by *PLM-solver*
lemma *all-self-eq-2*[*PLM*]:
[∀ α :: 'a::*id-eq* . □(α = α) *in v*]
by *PLM-solver*

lemma *t-id-t-proper-1*[*PLM*]:
[τ = τ' → (∃ β . (β^P) = τ) *in v*]
proof (*rule CP*)
assume [τ = τ' *in v*]
moreover {
assume [τ =$_E$ τ' *in v*]
hence [∃ β . (β^P) = τ *in v*]
apply −
apply (*rule cqt-5-mod*[**where** ψ=λ τ . τ =$_E$ τ', *axiom-instance*, *deduction*])
subgoal unfolding *identity-defs* **by** (*rule SimpleExOrEnc.intros*)
by *simp*
}
moreover {
assume [(|A!,τ|) & (|A!,τ'|) & □(∀ F. {|τ,F|} ≡ {|τ',F|}) *in v*]
hence [∃ β . (β^P) = τ *in v*]
apply −
apply (*rule cqt-5-mod*[**where** ψ=λ τ . (|A!,τ|), *axiom-instance*, *deduction*])
subgoal unfolding *identity-defs* **by** (*rule SimpleExOrEnc.intros*)
by *PLM-solver*
}
ultimately show [∃ β . (β^P) = τ *in v*] **unfolding** *identity*$_\kappa$*-def*
using *intro-elim-4-b reductio-aa-1* **by** *blast*
qed

lemma *t-id-t-proper-2*[*PLM*]: [τ = τ' → (∃ β . (β^P) = τ') *in v*]
proof (*rule CP*)
assume [τ = τ' *in v*]
moreover {
assume [τ =$_E$ τ' *in v*]
hence [∃ β . (β^P) = τ' *in v*]
apply −
apply (*rule cqt-5-mod*[**where** ψ=λ τ' . τ =$_E$ τ', *axiom-instance*, *deduction*])
subgoal unfolding *identity-defs* **by** (*rule SimpleExOrEnc.intros*)
by *simp*
}
moreover {
assume [(|A!,τ|) & (|A!,τ'|) & □(∀ F. {|τ,F|} ≡ {|τ',F|}) *in v*]
hence [∃ β . (β^P) = τ' *in v*]
apply −
apply (*rule cqt-5-mod*[**where** ψ=λ τ . (|A!,τ|), *axiom-instance*, *deduction*])
subgoal unfolding *identity-defs* **by** (*rule SimpleExOrEnc.intros*)
by *PLM-solver*
}
ultimately show [∃ β . (β^P) = τ' *in v*] **unfolding** *identity*$_\kappa$*-def*
using *intro-elim-4-b reductio-aa-1* **by** *blast*
qed

lemma *id-nec*[*PLM*]: [((α::'a::*id-eq*) = (β)) ≡ □((α) = (β)) *in v*]

apply (*rule* ≡*I*)
using *l-identity*[**where** $\varphi = (\lambda\ \beta\ .\ \Box((\alpha) = (\beta)))$, *axiom-instance*]
id-eq-1 RN ded-thm-cor-4 **unfolding** *identity-ν-def*
apply *blast*
using *qml-2*[*axiom-instance*] **by** *blast*

lemma *id-nec-desc*[*PLM*]:
$[((\iota x.\ \varphi\ x) = (\iota x.\ \psi\ x)) \equiv \Box((\iota x.\ \varphi\ x) = (\iota x.\ \psi\ x))\ in\ v]$
proof (*cases* $[(\exists\ \alpha.\ (\alpha^P) = (\iota x\ .\ \varphi\ x))\ in\ v] \wedge [(\exists\ \beta.\ (\beta^P) = (\iota x\ .\ \psi\ x))\ in\ v]$)
assume $[(\exists\ \alpha.\ (\alpha^P) = (\iota x\ .\ \varphi\ x))\ in\ v] \wedge [(\exists\ \beta.\ (\beta^P) = (\iota x\ .\ \psi\ x))\ in\ v]$
then obtain α **and** β **where**
$[(\alpha^P) = (\iota x\ .\ \varphi\ x)\ in\ v] \wedge [(\beta^P) = (\iota x\ .\ \psi\ x)\ in\ v]$
apply − **unfolding** *conn-defs* **by** *PLM-solver*
moreover {
moreover have $[(\alpha) = (\beta) \equiv \Box((\alpha) = (\beta))\ in\ v]$ **by** *PLM-solver*
ultimately have $[((\iota x.\ \varphi\ x) = (\beta^P) \equiv \Box((\iota x.\ \varphi\ x) = (\beta^P)))\ in\ v]$
using *l-identity*[**where** $\varphi{=}\lambda\ \alpha\ .\ (\alpha) = (\beta^P) \equiv \Box((\alpha) = (\beta^P))$, *axiom-instance*]
modus-ponens **unfolding** *identity-ν-def* **by** *metis*
}
ultimately show *?thesis*
using *l-identity*[**where** $\varphi{=}\lambda\ \alpha\ .\ (\iota x\ .\ \varphi\ x) = (\alpha) \equiv \Box((\iota x\ .\ \varphi\ x) = (\alpha))$, *axiom-instance*]
modus-ponens **by** *metis*
next
assume $\neg([(\exists\ \alpha.\ (\alpha^P) = (\iota x\ .\ \varphi\ x))\ in\ v] \wedge [(\exists\ \beta.\ (\beta^P) = (\iota x\ .\ \psi\ x))\ in\ v])$
hence $\neg[(\!|A!,(\iota x\ .\ \varphi\ x)|\!)\ in\ v] \wedge \neg[(\iota x\ .\ \varphi\ x) =_E (\iota x\ .\ \psi\ x)\ in\ v]$
$\vee\ \neg[(\!|A!,(\iota x\ .\ \psi\ x)|\!)\ in\ v] \wedge \neg[(\iota x\ .\ \varphi\ x) =_E (\iota x\ .\ \psi\ x)\ in\ v]$
unfolding *identity*$_E$*-infix-def*
using *cqt-5*[*axiom-instance*] *PLM.contraposition-1 SimpleExOrEnc.intros*
vdash-properties-10 **by** *meson*
hence $\neg[(\iota x\ .\ \varphi\ x) = (\iota x\ .\ \psi\ x)\ in\ v]$
apply − **unfolding** *identity-defs* **by** *PLM-solver*
thus *?thesis* **apply** − **apply** *PLM-solver*
using *qml-2*[*axiom-instance*, *deduction*] **by** *auto*
qed

A.9.8. Quantification

— TODO: think about the distinction in PM here
lemma *rule-ui*[*PLM*,*PLM-elim*,*PLM-dest*]:
$[\forall\,\alpha\ .\ \varphi\ \alpha\ in\ v] \Longrightarrow [\varphi\ \beta\ in\ v]$
by (*meson cqt-1*[*axiom-instance*, *deduction*])
lemmas $\forall\ E$ = *rule-ui*

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

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

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

lemmas $\forall\, I$ = *universal*

lemma *cqt-basic-1*[*PLM*]:
[$(\forall\, \alpha.\ (\forall\, \beta\ .\ \varphi\ \alpha\ \beta)) \equiv (\forall\, \beta.\ (\forall\, \alpha.\ \varphi\ \alpha\ \beta))$ *in* v]
by *PLM-solver*
lemma *cqt-basic-2*[*PLM*]:
[$(\forall\, \alpha.\ \varphi\ \alpha \equiv \psi\ \alpha) \equiv ((\forall\, \alpha.\ \varphi\ \alpha \rightarrow \psi\ \alpha)\ \&\ (\forall\, \alpha.\ \psi\ \alpha \rightarrow \varphi\ \alpha))$ *in* v]
by *PLM-solver*
lemma *cqt-basic-3*[*PLM*]:
[$(\forall\, \alpha.\ \varphi\ \alpha \equiv \psi\ \alpha) \rightarrow ((\forall\, \alpha.\ \varphi\ \alpha) \equiv (\forall\, \alpha.\ \psi\ \alpha))$ *in* v]
by *PLM-solver*
lemma *cqt-basic-4*[*PLM*]:
[$(\forall\, \alpha.\ \varphi\ \alpha\ \&\ \psi\ \alpha) \equiv ((\forall\, \alpha.\ \varphi\ \alpha)\ \&\ (\forall\, \alpha.\ \psi\ \alpha))$ *in* v]
by *PLM-solver*
lemma *cqt-basic-6*[*PLM*]:
[$(\forall\, \alpha.\ (\forall\, \alpha.\ \varphi\ \alpha)) \equiv (\forall\, \alpha.\ \varphi\ \alpha)$ *in* v]
by *PLM-solver*
lemma *cqt-basic-7*[*PLM*]:
[$(\varphi \rightarrow (\forall\, \alpha\ .\ \psi\ \alpha)) \equiv (\forall\, \alpha.(\varphi \rightarrow \psi\ \alpha))$ *in* v]
by *PLM-solver*
lemma *cqt-basic-8*[*PLM*]:
[$((\forall\, \alpha.\ \varphi\ \alpha) \vee (\forall\, \alpha.\ \psi\ \alpha)) \rightarrow (\forall\, \alpha.\ (\varphi\ \alpha \vee \psi\ \alpha))$ *in* v]
by *PLM-solver*
lemma *cqt-basic-9*[*PLM*]:
[$((\forall\, \alpha.\ \varphi\ \alpha \rightarrow \psi\ \alpha)\ \&\ (\forall\, \alpha.\ \psi\ \alpha \rightarrow \chi\ \alpha)) \rightarrow (\forall\, \alpha.\ \varphi\ \alpha \rightarrow \chi\ \alpha)$ *in* v]
by *PLM-solver*
lemma *cqt-basic-10*[*PLM*]:
[$((\forall\, \alpha.\ \varphi\ \alpha \equiv \psi\ \alpha)\ \&\ (\forall\, \alpha.\ \psi\ \alpha \equiv \chi\ \alpha)) \rightarrow (\forall\, \alpha.\ \varphi\ \alpha \equiv \chi\ \alpha)$ *in* v]
by *PLM-solver*
lemma *cqt-basic-11*[*PLM*]:
[$(\forall\, \alpha.\ \varphi\ \alpha \equiv \psi\ \alpha) \equiv (\forall\, \alpha.\ \psi\ \alpha \equiv \varphi\ \alpha)$ *in* v]
by *PLM-solver*
lemma *cqt-basic-12*[*PLM*]:
[$(\forall\, \alpha.\ \varphi\ \alpha) \equiv (\forall\, \beta.\ \varphi\ \beta)$ *in* v]
by *PLM-solver*

lemma *existential*[*PLM*,*PLM-intro*]:
[$\varphi\ \alpha$ *in* v] $\Longrightarrow$ [$\exists\ \alpha.\ \varphi\ \alpha$ *in* v]
unfolding *exists-def* **by** *PLM-solver*
lemmas $\exists\, I$ = *existential*
lemma *instantiation-*[*PLM*,*PLM-elim*,*PLM-dest*]:
$[\![$[$\exists\, \alpha\ .\ \varphi\ \alpha$ *in* v]; $(\bigwedge \alpha.$[$\varphi\ \alpha$ *in* v] $\Longrightarrow$ [ψ *in* v]$)]\!] \Longrightarrow$ [ψ *in* v]
unfolding *exists-def* **by** *PLM-solver*

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

lemma *cqt-further-1*[*PLM*]:
[$(\forall\, \alpha.\ \varphi\ \alpha) \rightarrow (\exists\, \alpha.\ \varphi\ \alpha)$ *in* v]
by *PLM-solver*
lemma *cqt-further-2*[*PLM*]:
[$(\neg(\forall\, \alpha.\ \varphi\ \alpha)) \equiv (\exists\, \alpha.\ \neg\varphi\ \alpha)$ *in* v]
unfolding *exists-def* **by** *PLM-solver*
lemma *cqt-further-3*[*PLM*]:
[$(\forall\, \alpha.\ \varphi\ \alpha) \equiv \neg(\exists\, \alpha.\ \neg\varphi\ \alpha)$ *in* v]

unfolding *exists-def* **by** *PLM-solver*
lemma *cqt-further-4*[*PLM*]:
[(¬(∃ α. φ α)) ≡ (∀ α. ¬φ α) *in v*]
unfolding *exists-def* **by** *PLM-solver*
lemma *cqt-further-5*[*PLM*]:
[(∃ α. φ α & ψ α) → ((∃ α. φ α) & (∃ α. ψ α)) *in v*]
unfolding *exists-def* **by** *PLM-solver*
lemma *cqt-further-6*[*PLM*]:
[(∃ α. φ α ∨ ψ α) ≡ ((∃ α. φ α) ∨ (∃ α. ψ α)) *in v*]
unfolding *exists-def* **by** *PLM-solver*
lemma *cqt-further-10*[*PLM*]:
[(φ (α::'a::*id-eq*) & (∀ β . φ β → β = α)) ≡ (∀ β . φ β ≡ β = α) *in v*]
apply *PLM-solver*
using *l-identity*[*axiom-instance*, *deduction*, *deduction*] *id-eq-2*[*deduction*]
apply *blast*
using *id-eq-1* **by** *auto*
lemma *cqt-further-11*[*PLM*]:
[((∀ α. φ α) & (∀ α. ψ α)) → (∀ α. φ α ≡ ψ α) *in v*]
by *PLM-solver*
lemma *cqt-further-12*[*PLM*]:
[((¬(∃ α. φ α)) & (¬(∃ α. ψ α))) → (∀ α. φ α ≡ ψ α) *in v*]
unfolding *exists-def* **by** *PLM-solver*
lemma *cqt-further-13*[*PLM*]:
[((∃ α. φ α) & (¬(∃ α. ψ α))) → (¬(∀ α. φ α ≡ ψ α)) *in v*]
unfolding *exists-def* **by** *PLM-solver*
lemma *cqt-further-14*[*PLM*]:
[(∃ α. ∃ β. φ α β) ≡ (∃ β. ∃ α. φ α β) *in v*]
unfolding *exists-def* **by** *PLM-solver*

lemma *nec-exist-unique*[*PLM*]:
[(∀ x. φ x → □(φ x)) → ((∃!x. φ x) → (∃!x. □(φ x))) *in v*]
proof (*rule CP*)
assume *a*: [∀ x. φ x → □φ x *in v*]
show [(∃!x. φ x) → (∃!x. □φ x) *in v*]
proof (*rule CP*)
assume [(∃!x. φ x) *in v*]
hence [∃ α. φ α & (∀ β. φ β → β = α) *in v*]
by (*simp only*: *exists-unique-def*)
then obtain α **where** *1*:
[φ α & (∀ β. φ β → β = α) *in v*]
by (*rule* ∃ *E*)
{
fix β
have [□φ β → β = α *in v*]
using *1* &*E*(*2*) *qml-2*[*axiom-instance*]
ded-thm-cor-3 ∀ *E* **by** *fastforce*
}
hence [∀ β. □φ β → β = α *in v*] **by** (*rule* ∀ *I*)
moreover have [□(φ α) *in v*]
using *1* &*E*(*1*) *a vdash-properties-10 cqt-orig-1*[*deduction*]
by *fast*
ultimately have [∃ α. □(φ α) & (∀ β. □φ β → β = α) *in v*]
using &*I* ∃ *I* **by** *fast*
thus [(∃!x. □φ x) *in v*]
unfolding *exists-unique-def* **by** *assumption*
qed
qed

A.9.9. Actuality and Descriptions

lemma *nec-imp-act*[*PLM*]: [$\Box\varphi \rightarrow \mathcal{A}\varphi$ *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*]:
[$\mathcal{A}(\mathcal{A}\varphi \rightarrow \varphi)$ *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*]:
[$\mathcal{A}(\varphi \rightarrow \mathcal{A}\varphi)$ *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*]:
[$(\mathcal{A}\varphi$ & $\mathcal{A}\psi) \rightarrow \mathcal{A}(\varphi$ & $\psi)$ *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*]:
[$\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$ *in* v]
unfolding *equiv-def*
by (*PLM-solver PLM-intro*: *act-conj-act-3*[**where** φ=$\mathcal{A}\varphi \rightarrow \varphi$
and ψ=$\varphi \rightarrow \mathcal{A}\varphi$, *deduction*])
lemma *closure-act-1a*[*PLM*]:
[$\mathcal{A}\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$ *in* v]
using *logic-actual-nec-4*[*axiom-instance*]
act-conj-act-4 $\equiv E(1)$
by *blast*
lemma *closure-act-1b*[*PLM*]:
[$\mathcal{A}\mathcal{A}\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$ *in* v]
using *logic-actual-nec-4*[*axiom-instance*]
act-conj-act-4 $\equiv E(1)$
by *blast*
lemma *closure-act-1c*[*PLM*]:
[$\mathcal{A}\mathcal{A}\mathcal{A}\mathcal{A}(\mathcal{A}\varphi \equiv \varphi)$ *in* v]
using *logic-actual-nec-4*[*axiom-instance*]
act-conj-act-4 $\equiv E(1)$
by *blast*
lemma *closure-act-2*[*PLM*]:
[$\forall\,\alpha.\ \mathcal{A}(\mathcal{A}(\varphi\ \alpha) \equiv \varphi\ \alpha)$ *in* v]
by *PLM-solver*

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

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

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

lemma *RA-2*[*PLM*,*PLM-intro*]:
([ψ *in* dw] $\Longrightarrow$ [φ *in* dw]) $\Longrightarrow$ ([$\mathcal{A}\psi$ *in* dw] $\Longrightarrow$ [$\mathcal{A}\varphi$ *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*]:
[$\mathcal{A}\varphi$ *in* dw] $\Longrightarrow$ [φ *in* dw]
using *logic-actual*[*necessitation-averse-axiom-instance*, *equiv-lr*] **.**

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

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

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

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

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

```
  private lemma ActualBoxI[PLM-intro]:
    [□φ in v] ⟹ [𝒜(□φ) in v]
    using qml-act-2[axiom-instance, equiv-lr] .
  private lemma ActualBoxE[PLM-elim, PLM-dest]:
    [𝒜(□φ) in v] ⟹ [□φ in v]
    using qml-act-2[axiom-instance, equiv-rl] .
  private lemma NotActualBoxD[PLM-dest]:
    ¬[𝒜(□φ) in v] ⟹ ¬[□φ in v]
    using ActualBoxI by blast

  private lemma ActualDisjI[PLM-intro]:
    [𝒜φ ∨ 𝒜ψ in v] ⟹ [𝒜(φ ∨ ψ) in v]
    unfolding disj-def by PLM-solver
  private lemma ActualDisjE[PLM-elim,PLM-dest]:
    [𝒜(φ ∨ ψ) in v] ⟹ [𝒜φ ∨ 𝒜ψ in v]
    unfolding disj-def by PLM-solver
  private lemma NotActualDisjD[PLM-dest]:
    ¬[𝒜(φ ∨ ψ) in v] ⟹ ¬[𝒜φ ∨ 𝒜ψ in v]
    using ActualDisjI by blast

  private lemma ActualForallI[PLM-intro]:
    [∀ x . 𝒜(φ x) in v] ⟹ [𝒜(∀ x . φ x) in v]
    using logic-actual-nec-3[axiom-instance, equiv-rl] .
  lemma ActualForallE[PLM-elim,PLM-dest]:
    [𝒜(∀ x . φ x) in v] ⟹ [∀ x . 𝒜(φ x) in v]
    using logic-actual-nec-3[axiom-instance, equiv-lr] .
  lemma NotActualForallD[PLM-dest]:
    ¬[𝒜(∀ x . φ x) in v] ⟹ ¬[∀ x . 𝒜(φ x) in v]
    using ActualForallI by blast

  lemma ActualActualI[PLM-intro]:
    [𝒜φ in v] ⟹ [𝒜𝒜φ in v]
    using logic-actual-nec-4[axiom-instance, equiv-lr] .
  lemma ActualActualE[PLM-elim,PLM-dest]:
    [𝒜𝒜φ in v] ⟹ [𝒜φ in v]
    using logic-actual-nec-4[axiom-instance, equiv-rl] .
  lemma NotActualActualD[PLM-dest]:
    ¬[𝒜𝒜φ in v] ⟹ ¬[𝒜φ in v]
    using ActualActualI by blast
end

lemma ANeg-1[PLM]:
  [¬𝒜φ ≡ ¬φ in dw]
  by PLM-solver
lemma ANeg-2[PLM]:
  [¬𝒜¬φ ≡ φ in dw]
  by PLM-solver
lemma Act-Basic-1[PLM]:
  [𝒜φ ∨ 𝒜¬φ in v]
  by PLM-solver
lemma Act-Basic-2[PLM]:
  [𝒜(φ & ψ) ≡ (𝒜φ & 𝒜ψ) in v]
  by PLM-solver
lemma Act-Basic-3[PLM]:
  [𝒜(φ ≡ ψ) ≡ ((𝒜(φ → ψ)) & (𝒜(ψ → φ))) in v]
  by PLM-solver
lemma Act-Basic-4[PLM]:
  [(𝒜(φ → ψ) & 𝒜(ψ → φ)) ≡ (𝒜φ ≡ 𝒜ψ) in v]
```

by *PLM-solver*
lemma *Act-Basic-5*[*PLM*]:
[$\mathcal{A}(\varphi \equiv \psi) \equiv (\mathcal{A}\varphi \equiv \mathcal{A}\psi)$ *in v*]
by *PLM-solver*
lemma *Act-Basic-6*[*PLM*]:
[$\Diamond\varphi \equiv \mathcal{A}(\Diamond\varphi)$ *in v*]
unfolding *diamond-def* **by** *PLM-solver*
lemma *Act-Basic-7*[*PLM*]:
[$\mathcal{A}\varphi \equiv \Box\mathcal{A}\varphi$ *in v*]
by (*simp add*: *qml-2*[*axiom-instance*] *qml-act-1*[*axiom-instance*] $\equiv I$)
lemma *Act-Basic-8*[*PLM*]:
[$\mathcal{A}(\Box\varphi) \rightarrow \Box\mathcal{A}\varphi$ *in v*]
by (*metis qml-act-2*[*axiom-instance*] *CP Act-Basic-7* $\equiv E(1)$
$\equiv E(2)$ *nec-imp-act vdash-properties-10*)
lemma *Act-Basic-9*[*PLM*]:
[$\Box\varphi \rightarrow \Box\mathcal{A}\varphi$ *in v*]
using *qml-act-1*[*axiom-instance*] *ded-thm-cor-3 nec-imp-act* **by** *blast*
lemma *Act-Basic-10*[*PLM*]:
[$\mathcal{A}(\varphi \vee \psi) \equiv \mathcal{A}\varphi \vee \mathcal{A}\psi$ *in v*]
by *PLM-solver*

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

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

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

lemma *hintikka*[*PLM*]:
[$(x^P = (\iota x.\ \varphi\ x)) \equiv (\varphi\ x\ \&\ (\forall\ z\ .\ \varphi\ z \rightarrow z = x))$ *in dw*]
proof −
have [$(\forall\ z\ .\ \varphi\ z \equiv z = x) \equiv (\varphi\ x\ \&\ (\forall\ z\ .\ \varphi\ z \rightarrow z = x))$ *in dw*]
unfolding *identity-ν-def* **apply** *PLM-solver* **using** *id-eq-obj-1* **apply** *simp*
using *l-identity*[**where** φ=$\lambda\ x\ .\ \varphi\ x$, *axiom-instance*,
deduction, *deduction*]

```
    using id-eq-obj-2[deduction] unfolding identity-ν-def by fastforce
  thus ?thesis using ≡E(5) fund-cont-desc by blast
 qed

lemma russell-axiom-a[PLM]:
 [(⦇F, ιx. φ x⦈) ≡ (∃ x . φ x & (∀ z . φ z → z = x) & ⦇F, x^P⦈) in dw]
 (is [?lhs ≡ ?rhs in dw])
 proof −
  {
   assume 1: [?lhs in dw]
   hence [∃ α. α^P = (ιx. φ x) in dw]
   using cqt-5[axiom-instance, deduction]
         SimpleExOrEnc.intros
   by blast
   then obtain α where 2:
    [α^P = (ιx. φ x) in dw]
    using ∃ E by auto
   hence 3: [φ α & (∀ z . φ z → z = α) in dw]
    using hintikka[equiv-lr] by simp
   from 2 have [(ιx. φ x) = (α^P) in dw]
    using l-identity[where α=α^P and β=ιx. φ x and φ=λ x . x = α^P,
          axiom-instance, deduction, deduction]
          id-eq-obj-1[where x=α] by auto
   hence [⦇F, α^P⦈ in dw]
   using 1 l-identity[where β=α^P and α=ιx. φ x and φ=λ x . ⦇F,x⦈,
                 axiom-instance, deduction, deduction] by auto
   with 3 have [φ α & (∀ z . φ z → z = α) & ⦇F, α^P⦈ in dw] by (rule &I)
   hence [?rhs in dw] using ∃ I[where α=α] by simp
  }
  moreover {
   assume [?rhs in dw]
   then obtain α where 4:
    [φ α & (∀ z . φ z → z = α) & ⦇F, α^P⦈ in dw]
    using ∃ E by auto
   hence [α^P = (ιx . φ x) in dw] ∧ [⦇F, α^P⦈ in dw]
    using hintikka[equiv-rl] &E by blast
   hence [?lhs in dw]
    using l-identity[axiom-instance, deduction, deduction]
    by blast
  }
  ultimately show ?thesis by PLM-solver
 qed

lemma russell-axiom-g[PLM]:
 [⦃ιx. φ x,F⦄ ≡ (∃ x . φ x & (∀ z . φ z → z = x) & ⦃x^P, F⦄) in dw]
 (is [?lhs ≡ ?rhs in dw])
 proof −
  {
   assume 1: [?lhs in dw]
   hence [∃ α. α^P = (ιx. φ x) in dw]
   using cqt-5[axiom-instance, deduction] SimpleExOrEnc.intros by blast
   then obtain α where 2: [α^P = (ιx. φ x) in dw] by (rule ∃ E)
   hence 3: [(φ α & (∀ z . φ z → z = α)) in dw]
    using hintikka[equiv-lr] by simp
   from 2 have [(ιx. φ x) = α^P in dw]
    using l-identity[where α=α^P and β=ιx. φ x and φ=λ x . x = α^P,
          axiom-instance, deduction, deduction]
          id-eq-obj-1[where x=α] by auto
```

```
    hence [⦃α^P, F⦄ in dw]
    using 1 l-identity[where β=α^P and α=ιx. φ x and φ=λ x . ⦃x,F⦄,
                       axiom-instance, deduction, deduction] by auto
    with 3 have [(φ α & (∀ z . φ z → z = α)) & ⦃α^P, F⦄ in dw]
      using &I by auto
    hence [?rhs in dw] using ∃ I[where α=α] by (simp add: identity-defs)
  }
  moreover {
    assume [?rhs in dw]
    then obtain α where 4:
      [φ α & (∀ z . φ z → z = α) & ⦃α^P, F⦄ in dw]
      using ∃ E by auto
    hence [α^P = (ιx . φ x) in dw] ∧ [⦃α^P, F⦄ in dw]
      using hintikka[equiv-rl] &E by blast
    hence [?lhs in dw]
      using l-identity[axiom-instance, deduction, deduction]
      by fast
  }
  ultimately show ?thesis by PLM-solver
qed

lemma russell-axiom[PLM]:
  assumes SimpleExOrEnc ψ
  shows [ψ (ιx. φ x) ≡ (∃ x . φ x & (∀ z . φ z → z = x) & ψ (x^P)) in dw]
  (is [?lhs ≡ ?rhs in dw])
  proof −
    {
      assume 1: [?lhs in dw]
      hence [∃ α. α^P = (ιx. φ x) in dw]
      using cqt-5[axiom-instance, deduction] assms by blast
      then obtain α where 2: [α^P = (ιx. φ x) in dw] by (rule ∃ E)
      hence 3: [(φ α & (∀ z . φ z → z = α)) in dw]
        using hintikka[equiv-lr] by simp
      from 2 have [(ιx. φ x) = (α^P) in dw]
        using l-identity[where α=α^P and β=ιx. φ x and φ=λ x . x = α^P,
              axiom-instance, deduction, deduction]
              id-eq-obj-1[where x=α] by auto
      hence [ψ (α^P) in dw]
        using 1 l-identity[where β=α^P and α=ιx. φ x and φ=λ x . ψ x,
                           axiom-instance, deduction, deduction] by auto
      with 3 have [φ α & (∀ z . φ z → z = α) & ψ (α^P) in dw]
        using &I by auto
      hence [?rhs in dw] using ∃ I[where α=α] by (simp add: identity-defs)
    }
    moreover {
      assume [?rhs in dw]
      then obtain α where 4:
        [φ α & (∀ z . φ z → z = α) & ψ (α^P) in dw]
        using ∃ E by auto
      hence [α^P = (ιx . φ x) in dw] ∧ [ψ (α^P) in dw]
        using hintikka[equiv-rl] &E by blast
      hence [?lhs in dw]
        using l-identity[axiom-instance, deduction, deduction]
        by fast
    }
    ultimately show ?thesis by PLM-solver
  qed
```

lemma *unique-exists*[*PLM*]:
[(∃ *y* . y^P = (ι*x*. φ *x*)) ≡ (∃!*x* . φ *x*) *in dw*]
proof((*rule* ≡*I*, *rule CP*, *rule-tac*[*2*] *CP*))
assume [∃ *y*. y^P = (ι*x*. φ *x*) *in dw*]
then obtain α **where**
[α^P = (ι*x*. φ *x*) *in dw*]
by (*rule* ∃ *E*)
hence [φ α **&** (∀ β. φ β → β = α) *in dw*]
using *hintikka*[*equiv-lr*] **by** *auto*
thus [∃!*x* . φ *x in dw*]
unfolding *exists-unique-def* **using** ∃ *I* **by** *fast*
next
assume [∃!*x* . φ *x in dw*]
then obtain α **where**
[φ α **&** (∀ β. φ β → β = α) *in dw*]
unfolding *exists-unique-def* **by** (*rule* ∃ *E*)
hence [α^P = (ι*x*. φ *x*) *in dw*]
using *hintikka*[*equiv-rl*] **by** *auto*
thus [∃ *y*. y^P = (ι*x*. φ *x*) *in dw*]
using ∃ *I* **by** *fast*
qed

lemma *y-in-1*[*PLM*]:
[x^P = (ι*x* . φ) → φ *in dw*]
using *hintikka*[*equiv-lr*, *conj1*] **by** (*rule CP*)

lemma *y-in-2*[*PLM*]:
[z^P = (ι*x* . φ *x*) → φ *z in dw*]
using *hintikka*[*equiv-lr*, *conj1*] **by** (*rule CP*)

lemma *y-in-3*[*PLM*]:
[(∃ *y* . y^P = (ι*x* . φ (x^P))) → φ (ι*x* . φ (x^P)) *in dw*]
proof (*rule CP*)
assume [(∃ *y* . y^P = (ι*x* . φ (x^P))) *in dw*]
then obtain *y* **where** *1*:
[y^P = (ι*x*. φ (x^P)) *in dw*]
by (*rule* ∃ *E*)
hence [φ (y^P) *in dw*]
using *y-in-2*[*deduction*] **unfolding** *identity-ν-def* **by** *blast*
thus [φ (ι*x*. φ (x^P)) *in dw*]
using *l-identity*[*axiom-instance*, *deduction*,
deduction] *1* **by** *fast*
qed

lemma *act-quant-nec*[*PLM*]:
[(∀ *z* . (𝒜φ *z* ≡ *z* = *x*)) ≡ (∀ *z*. 𝒜𝒜φ *z* ≡ *z* = *x*) *in v*]
by *PLM-solver*

lemma *equi-desc-descA-1*[*PLM*]:
[(x^P = (ι*x* . φ *x*)) ≡ (x^P = (ι*x* . 𝒜φ *x*)) *in v*]
using *descriptions*[*axiom-instance*] **apply** (*rule* ≡*E*(*5*))
using *act-quant-nec* **apply** (*rule* ≡*E*(*5*))
using *descriptions*[*axiom-instance*]
by (*meson* ≡*E*(*6*) *oth-class-taut-4-a*)

lemma *equi-desc-descA-2*[*PLM*]:
[(∃ *y* . y^P = (ι*x*. φ *x*)) → ((ι*x* . φ *x*) = (ι*x* . 𝒜φ *x*)) *in v*]
proof (*rule CP*)

```
    assume [∃ y. y^P = (ιx. φ x) in v]
    then obtain y where
      [y^P = (ιx. φ x) in v]
      by (rule ∃ E)
    moreover hence [y^P = (ιx. 𝒜φ x) in v]
      using equi-desc-descA-1[equiv-lr] by auto
    ultimately show [(ιx. φ x) = (ιx. 𝒜φ x) in v]
      using l-identity[axiom-instance, deduction, deduction]
      by fast
  qed

lemma equi-desc-descA-3[PLM]:
  assumes SimpleExOrEnc ψ
  shows [ψ (ιx. φ x) → (∃ y . y^P = (ιx. 𝒜φ x)) in v]
  proof (rule CP)
    assume [ψ (ιx. φ x) in v]
    hence [∃ α. α^P = (ιx. φ x) in v]
      using cqt-5[OF assms, axiom-instance, deduction] by auto
    then obtain α where [α^P = (ιx. φ x) in v] by (rule ∃ E)
    hence [α^P = (ιx . 𝒜φ x) in v]
      using equi-desc-descA-1[equiv-lr] by auto
    thus [∃ y. y^P = (ιx. 𝒜φ x) in v]
      using ∃ I by fast
  qed

lemma equi-desc-descA-4[PLM]:
  assumes SimpleExOrEnc ψ
  shows [ψ (ιx. φ x) → ((ιx. φ x) = (ιx. 𝒜φ x)) in v]
  proof (rule CP)
    assume [ψ (ιx. φ x) in v]
    hence [∃ α. α^P = (ιx. φ x) in v]
      using cqt-5[OF assms, axiom-instance, deduction] by auto
    then obtain α where [α^P = (ιx. φ x) in v] by (rule ∃ E)
    moreover hence [α^P  = (ιx . 𝒜φ x) in v]
      using equi-desc-descA-1[equiv-lr] by auto
    ultimately show [(ιx. φ x)  = (ιx . 𝒜φ x) in v]
      using l-identity[axiom-instance, deduction, deduction] by fast
  qed

lemma nec-hintikka-scheme[PLM]:
  [(x^P = (ιx. φ x)) ≡ (𝒜φ x & (∀ z . 𝒜φ z → z = x)) in v]
  using descriptions[axiom-instance]
  apply (rule ≡E(5))
  apply PLM-solver
   using id-eq-obj-1 apply simp
   using id-eq-obj-2[deduction]
         l-identity[where α=x, axiom-instance, deduction, deduction]
   unfolding identity-ν-def
   apply blast
  using l-identity[where α=x, axiom-instance, deduction, deduction]
  id-eq-2[where 'a=ν, deduction] unfolding identity-ν-def by meson

lemma equiv-desc-eq[PLM]:
  assumes ⋀x.[𝒜(φ x ≡ ψ x) in v]
  shows [(∀ x . ((x^P = (ιx . φ x)) ≡ (x^P = (ιx . ψ x)))) in v]
  proof(rule ∀ I)
    fix x
    {
```

```
  assume [x^P = (ιx . φ x) in v]
  hence 1: [A_φ x & (∀ z. A_φ z → z = x) in v]
    using nec-hintikka-scheme[equiv-lr] by auto
  hence 2: [A_φ x in v] ∧ [(∀ z. A_φ z → z = x) in v]
    using &E by blast
  {
     fix z
     {
       assume [A_ψ z in v]
       hence [A_φ z in v]
        using assms[where x=z] apply − by PLM-solver
       moreover have [A_φ z → z = x in v]
         using 2 cqt-1[axiom-instance,deduction] by auto
       ultimately have [z = x in v]
        using vdash-properties-10 by auto
     }
     hence [A_ψ z → z = x in v] by (rule CP)
  }
  hence [(∀ z . A_ψ z → z = x) in v] by (rule ∀I)
  moreover have [A_ψ x in v]
    using 1[conj1] assms[where x=x]
    apply − by PLM-solver
  ultimately have [A_ψ x & (∀ z. A_ψ z → z = x) in v]
    by PLM-solver
  hence [x^P = (ιx. ψ x) in v]
   using nec-hintikka-scheme[where φ=ψ, equiv-rl] by auto
}
moreover {
  assume [x^P = (ιx . ψ x) in v]
  hence 1: [A_ψ x & (∀ z. A_ψ z → z = x) in v]
    using nec-hintikka-scheme[equiv-lr] by auto
  hence 2: [A_ψ x in v] ∧ [(∀ z. A_ψ z → z = x) in v]
    using &E by blast
  {
    fix z
    {
      assume [A_φ z in v]
      hence [A_ψ z in v]
        using assms[where x=z]
        apply − by PLM-solver
      moreover have [A_ψ z → z = x in v]
        using 2 cqt-1[axiom-instance,deduction] by auto
      ultimately have [z = x in v]
        using vdash-properties-10 by auto
    }
    hence [A_φ z → z = x in v] by (rule CP)
  }
  hence [(∀ z. A_φ z → z = x) in v] by (rule ∀I)
  moreover have [A_φ x in v]
    using 1[conj1] assms[where x=x]
    apply − by PLM-solver
  ultimately have [A_φ x & (∀ z. A_φ z → z = x) in v]
    by PLM-solver
  hence [x^P = (ιx. φ x) in v]
    using nec-hintikka-scheme[where φ=φ,equiv-rl]
    by auto
}
ultimately show [x^P = (ιx. φ x) ≡ (x^P) = (ιx. ψ x) in v]
```

```
    using ≡I CP by auto
  qed

lemma UniqueAux:
  assumes [(𝒜φ (α::ν) & (∀ z . 𝒜(φ z) → z = α)) in v]
  shows [(∀ z . (𝒜(φ z) ≡ (z = α))) in v]
  proof −
    {
      fix z
      {
        assume [𝒜(φ z) in v]
        hence [z = α in v]
          using assms[conj2, THEN cqt-1[where α=z,
                      axiom-instance, deduction],
                    deduction] by auto
      }
      moreover {
        assume [z = α in v]
        hence [α = z in v]
          unfolding identity-ν-def
          using id-eq-obj-2[deduction] by fast
        hence [𝒜(φ z) in v] using assms[conj1]
          using l-identity[axiom-instance, deduction,
                           deduction] by fast
      }
      ultimately have [(𝒜(φ z) ≡ (z = α)) in v]
        using ≡I CP by auto
    }
    thus [(∀ z . (𝒜(φ z) ≡ (z = α))) in v]
    by (rule ∀I)
  qed

lemma nec-russell-axiom[PLM]:
  assumes SimpleExOrEnc ψ
  shows [(ψ (ιx. φ x)) ≡ (∃ x . (𝒜φ x & (∀ z . 𝒜(φ z) → z = x))
                           & ψ (x^P))) in v]
  (is [?lhs ≡ ?rhs in v])
  proof −
    {
      assume 1: [?lhs in v]
      hence [∃α. (α^P) = (ιx. φ x) in v]
        using cqt-5[axiom-instance, deduction] assms by blast
      then obtain α where 2: [(α^P) = (ιx. φ x) in v] by (rule ∃E)
      hence [(∀ z . (𝒜(φ z) ≡ (z = α))) in v]
        using descriptions[axiom-instance, equiv-lr] by auto
      hence 3: [(𝒜φ α) & (∀ z . (𝒜(φ z) → (z = α))) in v]
        using cqt-1[where α=α and φ=λ z . (𝒜(φ z) ≡ (z = α)),
                    axiom-instance, deduction, equiv-rl]
        using id-eq-obj-1[where x=α] unfolding identity-ν-def
        using hintikka[equiv-lr] cqt-basic-2[equiv-lr,conj1]
        &I by fast
      from 2 have [(ιx. φ x) = (α^P)  in v]
        using l-identity[where β=(ιx. φ x) and φ=λ x . x = (α^P),
              axiom-instance, deduction, deduction]
              id-eq-obj-1[where x=α] by auto
      hence [ψ (α^P) in v]
        using 1 l-identity[where α=(ιx. φ x) and φ=λ x . ψ x,
                           axiom-instance, deduction,
```

```
                    deduction] by auto
    with 3 have [(𝒜φ α & (∀ z . 𝒜(φ z) → (z = α))) & ψ (α^P) in v]
      using &I by simp
    hence [?rhs in v]
      using ∃ I[where α=α]
      by (simp add: identity-defs)
  }
  moreover {
    assume [?rhs in v]
    then obtain α where 4:
      [(𝒜φ α & (∀ z . 𝒜(φ z) → z = α)) & ψ (α^P) in v]
      using ∃ E by auto
    hence [(∀ z . (𝒜(φ z) ≡ (z = α))) in v]
      using UniqueAux &E(1) by auto
    hence [(α^P) = (ιx . φ x) in v] ∧ [ψ (α^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]:
  [(∃ y . (y^P) = (ιx. φ x)) ≡ (∃! x . 𝒜(φ x)) in v] (is [?lhs ≡ ?rhs in v])
  proof −
    {
      assume [?lhs in v]
      then obtain α where
        [((α^P) = (ιx. φ x)) in v]
        by (rule ∃ E)
      hence [(|A!,(ιx. φ x)|) in v] ∨ [(α^P) =_E (ιx. φ x) in v]
        apply − unfolding identity-defs by PLM-solver
      then obtain x where
        [((𝒜φ x & (∀ z . 𝒜(φ z) → z = x))) in v]
        using nec-russell-axiom[where ψ=λx . (|A!,x|), equiv-lr, THEN ∃ E]
        using nec-russell-axiom[where ψ=λx . (α^P) =_E x, equiv-lr, THEN ∃ E]
        using SimpleExOrEnc.intros unfolding identity_E-infix-def
        by (meson &E)
      hence [?rhs in v] unfolding exists-unique-def by (rule ∃ I)
    }
    moreover {
      assume [?rhs in v]
      then obtain x where
        [((𝒜φ x & (∀ z . 𝒜(φ z) → z = x))) in v]
        unfolding exists-unique-def by (rule ∃ E)
      hence [∀ z. 𝒜φ z ≡ z = x in v]
        using UniqueAux by auto
      hence [(x^P) = (ιx. φ x) in v]
        using descriptions[axiom-instance, equiv-rl] by auto
      hence [?lhs in v] by (rule ∃ I)
    }
    ultimately show ?thesis
      using ≡I CP by auto
  qed
```

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

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

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

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

lemma *unique-box-desc*[*PLM*]:
[$(\forall\ x\ .\ (\varphi\ x \rightarrow \Box(\varphi\ x))) \rightarrow ((\exists !x\ .\ \varphi\ x)$
$\rightarrow (\forall\ y\ .\ (y^P = (\iota x\ .\ \varphi\ x)) \rightarrow \varphi\ y))$ *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*]:
(⋀*v*.[*φ* → *ψ in v*]) ⟹ [□*φ* → □*ψ in v*]
using *RN qml-1*[*axiom-instance*] *vdash-properties-10* **by** *blast*

lemma *RM-1-b*[*PLM*]:
(⋀*v*.[*χ in v*] ⟹ [*φ* → *ψ in v*]) ⟹ ([□*χ in v*] ⟹ [□*φ* → □*ψ in v*])
using *RN-2 qml-1*[*axiom-instance*] *vdash-properties-10* **by** *blast*

lemma *RM-2*[*PLM*]:
(⋀*v*.[*φ* → *ψ in v*]) ⟹ [◇*φ* → ◇*ψ in v*]
unfolding *diamond-def*
using *RM-1 contraposition-1* **by** *auto*

lemma *RM-2-b*[*PLM*]:
(⋀*v*.[*χ in v*] ⟹ [*φ* → *ψ in v*]) ⟹ ([□*χ in v*] ⟹ [◇*φ* → ◇*ψ in v*])
unfolding *diamond-def*
using *RM-1-b contraposition-1* **by** *blast*

lemma *KBasic-1*[*PLM*]:
[□*φ* → □(*ψ* → *φ*) *in v*]
by (*simp only*: *pl-1*[*axiom-instance*] *RM-1*)
lemma *KBasic-2*[*PLM*]:
[□(¬*φ*) → □(*φ* → *ψ*) *in v*]
by (*simp only*: *RM-1 useful-tautologies-3*)
lemma *KBasic-3*[*PLM*]:
[□(*φ* & *ψ*) ≡ □*φ* & □*ψ in v*]
apply (*rule* ≡*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*]:
[□(*φ* ≡ *ψ*) ≡ (□(*φ* → *ψ*) & □(*ψ* → *φ*)) *in v*]
apply (*rule* ≡*I*)
unfolding *equiv-def* **using** *KBasic-3 PLM.CP* ≡*E*(*1*)
apply *blast*
using *KBasic-3 PLM.CP* ≡*E*(*2*)
by *blast*
lemma *KBasic-5*[*PLM*]:
[(□(*φ* → *ψ*) & □(*ψ* → *φ*)) → (□*φ* ≡ □*ψ*) *in v*]
by (*metis qml-1*[*axiom-instance*] *CP* &*E* ≡*I vdash-properties-10*)
lemma *KBasic-6*[*PLM*]:
[□(*φ* ≡ *ψ*) → (□*φ* ≡ □*ψ*) *in v*]
using *KBasic-4 KBasic-5* **by** (*metis equiv-def ded-thm-cor-3* &*E*(*1*))
lemma [(□*φ* ≡ □*ψ*) → □(*φ* ≡ *ψ*) *in v*]
nitpick[*expect*=*genuine*, *user-axioms*, *card* = *1*, *card i* = *2*]
oops — countermodel as desired
lemma *KBasic-7*[*PLM*]:
[(□*φ* & □*ψ*) → □(*φ* ≡ *ψ*) *in v*]
proof (*rule CP*)
assume [□*φ* & □*ψ in v*]
hence [□(*ψ* → *φ*) *in v*] ∧ [□(*φ* → *ψ*) *in v*]
using &*E KBasic-1 vdash-properties-10* **by** *blast*

thus $[\Box(\varphi \equiv \psi)\ in\ v]$
using *KBasic-4* $\equiv E(2)$ *intro-elim-1* **by** *blast*
qed

lemma *KBasic-8*[*PLM*]:
$[\Box(\varphi\ \&\ \psi) \rightarrow \Box(\varphi \equiv \psi)\ in\ v]$
using *KBasic-7 KBasic-3*
by (*metis equiv-def PLM.ded-thm-cor-3* $\&E(1)$)
lemma *KBasic-9*[*PLM*]:
$[\Box((\neg\varphi)\ \&\ (\neg\psi)) \rightarrow \Box(\varphi \equiv \psi)\ in\ v]$
proof (*rule CP*)
assume $[\Box((\neg\varphi)\ \&\ (\neg\psi))\ in\ v]$
hence $[\Box((\neg\varphi) \equiv (\neg\psi))\ in\ v]$
using *KBasic-8 vdash-properties-10* **by** *blast*
moreover have $\bigwedge v.[((\neg\varphi) \equiv (\neg\psi)) \rightarrow (\varphi \equiv \psi)\ in\ v]$
using *CP* $\equiv E(2)$ *oth-class-taut-5-d* **by** *blast*
ultimately show $[\Box(\varphi \equiv \psi)\ in\ v]$
using *RM-1 PLM.vdash-properties-10* **by** *blast*
qed

lemma *rule-sub-lem-1-a*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [(\neg\psi) \equiv (\neg\chi)\ in\ v]$
using *qml-2*[*axiom-instance*] $\equiv E(1)$ *oth-class-taut-5-d*
vdash-properties-10
by *blast*
lemma *rule-sub-lem-1-b*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [(\psi \rightarrow \Theta) \equiv (\chi \rightarrow \Theta)\ in\ v]$
by (*metis equiv-def contraposition-1 CP* $\&E(2)$ $\equiv I$
$\equiv E(1)$ *rule-sub-lem-1-a*)
lemma *rule-sub-lem-1-c*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [(\Theta \rightarrow \psi) \equiv (\Theta \rightarrow \chi)\ in\ v]$
by (*metis CP* $\equiv I$ $\equiv E(3)$ $\equiv E(4)$ $\neg\neg I$
$\neg\neg E$ *rule-sub-lem-1-a*)
lemma *rule-sub-lem-1-d*[*PLM*]:
$(\bigwedge x.[\Box(\psi\ x \equiv \chi\ x)\ in\ v]) \Longrightarrow [(\forall\,\alpha.\ \psi\ \alpha) \equiv (\forall\,\alpha.\ \chi\ \alpha)\ in\ v]$
by (*metis equiv-def* $\forall\ I$ *CP* $\&E \equiv I$ *raa-cor-1*
vdash-properties-10 rule-sub-lem-1-a $\forall\ E$)
lemma *rule-sub-lem-1-e*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [\boldsymbol{\mathcal{A}}\psi \equiv \boldsymbol{\mathcal{A}}\chi\ in\ v]$
using *Act-Basic-5* $\equiv E(1)$ *nec-imp-act*
vdash-properties-10
by *blast*
lemma *rule-sub-lem-1-f*[*PLM*]:
$[\Box(\psi \equiv \chi)\ in\ v] \Longrightarrow [\Box\psi \equiv \Box\chi\ in\ v]$
using *KBasic-6* $\equiv I$ $\equiv E(1)$ *vdash-properties-9*
by *blast*

named-theorems *Substable-intros*

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

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

lemma *Substable-intro-not*[*Substable-intros*]:

```
    assumes Substable cond ψ
    shows Substable cond (λ φ . ¬(ψ φ))
    using assms unfolding Substable-def
    using rule-sub-lem-1-a RN-2 ≡E oth-class-taut-5-d by metis
  lemma Substable-intro-impl[Substable-intros]:
    assumes Substable cond ψ
        and Substable cond χ
    shows Substable cond (λ φ . ψ φ → χ φ)
    using assms unfolding Substable-def
    by (metis ≡I CP intro-elim-6-a intro-elim-6-b)
  lemma Substable-intro-box[Substable-intros]:
    assumes Substable cond ψ
    shows Substable cond (λ φ . □(ψ φ))
    using assms unfolding Substable-def
    using rule-sub-lem-1-f RN by meson
  lemma Substable-intro-actual[Substable-intros]:
    assumes Substable cond ψ
    shows Substable cond (λ φ . 𝒜(ψ φ))
    using assms unfolding Substable-def
    using rule-sub-lem-1-e RN by meson
  lemma Substable-intro-all[Substable-intros]:
    assumes ∀ x . Substable cond (ψ x)
    shows Substable cond (λ φ . ∀ x . ψ x φ)
    using assms unfolding Substable-def
    by (simp add: RN rule-sub-lem-1-d)

  named-theorems Substable-Cond-defs
end

class Substable =
  fixes Substable-Cond :: 'a⇒'a⇒bool
  assumes rule-sub-nec:
    ⋀ φ ψ χ Θ v . ⟦PLM.Substable Substable-Cond φ; Substable-Cond ψ χ⟧
      ⟹ Θ [φ ψ in v] ⟹ Θ [φ χ in v]

instantiation o :: Substable
begin
  definition Substable-Cond-o where [PLM.Substable-Cond-defs]:
    Substable-Cond-o ≡ λ φ ψ . ∀ v . [φ ≡ ψ in v]
  instance proof
    interpret PLM .
    fix φ :: o ⇒ o and  ψ χ :: o and Θ :: bool ⇒ bool and v::i
    assume Substable Substable-Cond φ
    moreover assume Substable-Cond ψ χ
    ultimately have [φ ψ ≡ φ χ in v]
    unfolding Substable-def by blast
    hence [φ ψ in v] = [φ χ in v] using ≡E by blast
    moreover assume Θ [φ ψ in v]
    ultimately show Θ [φ χ in v] by simp
  qed
end

instantiation fun :: (type, Substable) Substable
begin
  definition Substable-Cond-fun where [PLM.Substable-Cond-defs]:
    Substable-Cond-fun ≡ λ φ ψ . ∀ x . Substable-Cond (φ x) (ψ x)
  instance proof
    interpret PLM .
```

```
    fix φ:: ('a ⇒ 'b) ⇒ o and  ψ χ :: 'a ⇒ 'b and Θ v
    assume Substable Substable-Cond φ
    moreover assume Substable-Cond ψ χ
    ultimately have [φ ψ ≡ φ χ in v]
      unfolding Substable-def by blast
    hence [φ ψ in v] = [φ χ in v] using ≡E by blast
    moreover assume Θ [φ ψ in v]
    ultimately show Θ [φ χ in v] by simp
  qed
end

context PLM
begin

  lemma Substable-intro-equiv[Substable-intros]:
    assumes Substable cond ψ
        and Substable cond χ
    shows Substable cond (λ φ . ψ φ ≡ χ φ)
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-conj[Substable-intros]:
    assumes Substable cond ψ
        and Substable cond χ
    shows Substable cond (λ φ . ψ φ & χ φ)
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-disj[Substable-intros]:
    assumes Substable cond ψ
        and Substable cond χ
    shows Substable cond (λ φ . ψ φ ∨ χ φ)
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-diamond[Substable-intros]:
    assumes Substable cond ψ
    shows Substable cond (λ φ . ◇(ψ φ))
    unfolding conn-defs by (simp add: assms Substable-intros)
  lemma Substable-intro-exist[Substable-intros]:
    assumes ∀ x . Substable cond (ψ x)
    shows Substable cond (λ φ . ∃ x . ψ x φ)
    unfolding conn-defs by (simp add: assms Substable-intros)

  lemma Substable-intro-id-o[Substable-intros]:
    Substable Substable-Cond (λ φ . φ)
    unfolding Substable-def Substable-Cond-o-def by blast
  lemma Substable-intro-id-fun[Substable-intros]:
    assumes Substable Substable-Cond ψ
    shows Substable Substable-Cond (λ φ . ψ (φ x))
    using assms unfolding Substable-def Substable-Cond-fun-def
    by blast

  method PLM-subst-method for ψ::'a::Substable and χ::'a::Substable =
    (match conclusion in Θ [φ χ in v] for Θ and φ and v ⇒
      ‹(rule rule-sub-nec[where Θ=Θ and χ=χ and ψ=ψ and φ=φ and v=v],
        ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
        unfold Substable-Cond-defs)›)

  method PLM-autosubst =
    (match premises in ⋀v . [ψ ≡ χ in v] for ψ and χ ⇒
      ‹ match conclusion in Θ [φ χ in v] for Θ φ and v ⇒
        ‹(rule rule-sub-nec[where Θ=Θ and χ=χ and ψ=ψ and φ=φ and v=v],
          ((fast intro: Substable-intros, ((assumption)+)?)+; fail),
```

unfold Substable-Cond-defs)› ›)

method *PLM-autosubst1* =
(*match* **premises in** $\bigwedge v\ x$. $[\psi\ x \equiv \chi\ x\ in\ v]$
for ψ::'*a*::*type*⇒o **and** χ::'*a*⇒o ⇒
‹ *match conclusion in* Θ $[\varphi\ \chi\ in\ v]$ *for* Θ φ *and* v ⇒
‹(*rule rule-sub-nec*[*where* Θ=Θ *and* χ=χ *and* ψ=ψ *and* φ=φ *and* v=v],
((*fast intro*: *Substable-intros*, ((*assumption*)+)*?*)+; *fail*),
unfold Substable-Cond-defs)› ›)

method *PLM-autosubst2* =
(*match* **premises in** $\bigwedge v\ x\ y$. $[\psi\ x\ y \equiv \chi\ x\ y\ in\ v]$
for ψ::'*a*::*type*⇒'*a*⇒o **and** χ::'*a*::*type*⇒'*a*⇒o ⇒
‹ *match conclusion in* Θ $[\varphi\ \chi\ in\ v]$ *for* Θ φ *and* v ⇒
‹(*rule rule-sub-nec*[*where* Θ=Θ *and* χ=χ *and* ψ=ψ *and* φ=φ *and* v=v],
((*fast intro*: *Substable-intros*, ((*assumption*)+)*?*)+; *fail*),
unfold Substable-Cond-defs)› ›)

method *PLM-subst-goal-method* **for** φ::'*a*::*Substable*⇒o **and** ψ::'*a* =
(*match* **conclusion in** Θ $[\varphi\ \chi\ in\ v]$ **for** Θ **and** χ **and** v ⇒
‹(*rule rule-sub-nec*[*where* Θ=Θ *and* χ=χ *and* ψ=ψ *and* φ=φ *and* v=v],
((*fast intro*: *Substable-intros*, ((*assumption*)+)*?*)+; *fail*),
unfold Substable-Cond-defs)›)

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

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

lemma *rule-sub-nec2*[*PLM*]:
assumes *Substable Substable-Cond* φ
shows $(\bigwedge v\ x\ y\ .[\psi\ x\ y \equiv \chi\ x\ y\ in\ v]) \Longrightarrow \Theta\ [\varphi\ \psi\ in\ v] \Longrightarrow \Theta\ [\varphi\ \chi\ in\ v]$
proof −
assume $(\bigwedge v\ x\ y\ .[\psi\ x\ y \equiv \chi\ x\ y\ in\ v])$
hence $[\varphi\ \psi\ in\ v] = [\varphi\ \chi\ in\ v]$
using *assms RN* **unfolding** *Substable-def Substable-Cond-defs*
using ≡*I CP* ≡*E*(*1*) ≡*E*(*2*) **by** *metis*
thus $\Theta\ [\varphi\ \psi\ in\ v] \Longrightarrow \Theta\ [\varphi\ \chi\ in\ v]$ **by** *auto*
qed

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

lemma *rule-sub-remark-2*:
assumes $(\bigwedge v.[(\!|R,x,y|\!) \equiv ((\!|R,x,y|\!)\ \&\ ((\!|Q,a|\!) \vee (\neg(\!|Q,a|\!))))\ in\ v])$
and $[p \rightarrow (\!|R,x,y|\!)\ in\ v]$
shows$[p \rightarrow ((\!|R,x,y|\!)\ \&\ ((\!|Q,a|\!) \vee (\neg(\!|Q,a|\!))))\ \ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

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

lemma *rule-sub-remark-4*:
assumes $\bigwedge v\ x.[(\neg(\neg(\!|P,x^P|\!))) \equiv (\!|P,x^P|\!)\ in\ v]$
and $[\boldsymbol{\mathcal{A}}(\neg(\neg(\!|P,x^P|\!)))\ in\ v]$
shows $[\boldsymbol{\mathcal{A}}(\!|P,x^P|\!)\ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst1* **by** *auto*

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

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

lemma *rule-sub-remark-7*:
assumes $\bigwedge v.[\varphi \equiv (\neg(\neg\varphi))\ in\ v]$
and $[\Box(\varphi \rightarrow \varphi)\ in\ v]$
shows $[\Box((\neg(\neg\varphi)) \rightarrow \varphi)\ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-8*:
assumes $\bigwedge v.[\boldsymbol{\mathcal{A}}\varphi \equiv \varphi\ in\ v]$
and $[\Box(\boldsymbol{\mathcal{A}}\varphi)\ in\ v]$
shows $[\Box(\varphi)\ in\ v]$
apply (*insert assms*) **apply** *PLM-autosubst* **by** *auto*

lemma *rule-sub-remark-9*:
assumes $\bigwedge v.[(\!|P,a|\!) \equiv ((\!|P,a|\!)\ \&\ ((\!|Q,b|\!) \vee (\neg(\!|Q,b|\!))))\ in\ v]$
and $[(\!|P,a|\!) = (\!|P,a|\!)\ in\ v]$
shows $[(\!|P,a|\!) = ((\!|P,a|\!)\ \&\ ((\!|Q,b|\!) \vee (\neg(\!|Q,b|\!))))\ in\ v]$
unfolding *identity-defs* **apply** (*insert assms*)
apply *PLM-autosubst* **oops** — no match as desired

— *dr-alphabetic-rules* implicitly holds
— *dr-alphabetic-thm* implicitly holds

lemma *KBasic2-1*[*PLM*]:
[$\Box\varphi \equiv \Box(\neg(\neg\varphi))$ *in* v]
apply (*PLM-subst-method* φ $(\neg(\neg\varphi))$)
by *PLM-solver*+

lemma *KBasic2-2*[*PLM*]:
[$(\neg(\Box\varphi)) \equiv \Diamond(\neg\varphi)$ *in* v]
unfolding *diamond-def*
apply (*PLM-subst-method* φ $\neg(\neg\varphi)$)
by *PLM-solver*+

lemma *KBasic2-3*[*PLM*]:
[$\Box\varphi \equiv (\neg(\Diamond(\neg\varphi)))$ *in* v]
unfolding *diamond-def*
apply (*PLM-subst-method* φ $\neg(\neg\varphi)$)
apply *PLM-solver*
by (*simp add*: *oth-class-taut-4-b*)
lemmas *Df*$\Box$ = *KBasic2-3*

lemma *KBasic2-4*[*PLM*]:
[$\Box(\neg(\varphi)) \equiv (\neg(\Diamond\varphi))$ *in* v]
unfolding *diamond-def*
by (*simp add*: *oth-class-taut-4-b*)

lemma *KBasic2-5*[*PLM*]:
[$\Box(\varphi \rightarrow \psi) \rightarrow (\Diamond\varphi \rightarrow \Diamond\psi)$ *in* v]
by (*simp only*: *CP RM-2-b*)
lemmas *K*$\Diamond$ = *KBasic2-5*

lemma *KBasic2-6*[*PLM*]:
[$\Diamond(\varphi \vee \psi) \equiv (\Diamond\varphi \vee \Diamond\psi)$ *in* v]
proof −
have [$\Box((\neg\varphi)$ & $(\neg\psi)) \equiv (\Box(\neg\varphi)$ & $\Box(\neg\psi))$ *in* v]
using *KBasic-3* **by** *blast*
hence [$(\neg(\Diamond(\neg((\neg\varphi)$ & $(\neg\psi))))) \equiv (\Box(\neg\varphi)$ & $\Box(\neg\psi))$ *in* v]
using *Df*$\Box$ **by** (*rule* $\equiv$*E*(*6*))
hence [$(\neg(\Diamond(\neg((\neg\varphi)$ & $(\neg\psi))))) \equiv ((\neg(\Diamond\varphi))$ & $(\neg(\Diamond\psi)))$ *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 [$(\neg(\Diamond(\varphi \vee \psi))) \equiv ((\neg(\Diamond\varphi))$ & $(\neg(\Diamond\psi)))$ *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 [$(\neg(\neg(\Diamond(\varphi \vee \psi)))) \equiv (\neg((\neg(\Diamond\varphi))\&(\neg(\Diamond\psi))))$ *in* v]
by (*rule oth-class-taut-5-d*[*equiv-lr*])
hence [$\Diamond(\varphi \vee \psi) \equiv (\neg((\neg(\Diamond\varphi))$ & $(\neg(\Diamond\psi))))$ *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*]:
[$(\Box\varphi \vee \Box\psi) \rightarrow \Box(\varphi \vee \psi)$ *in* v]

```
  proof −
    have ⋀ v . [φ → (φ ∨ ψ) in v]
      by (metis contraposition-1 contraposition-2 useful-tautologies-3 disj-def)
    hence [□φ → □(φ ∨ ψ) in v] using RM-1 by auto
    moreover {
       have ⋀ v . [ψ → (φ ∨ ψ) in v]
         by (simp only: pl-1[axiom-instance] disj-def)
       hence [□ψ → □(φ ∨ ψ) in v]
         using RM-1 by auto
    }
    ultimately show ?thesis
      using oth-class-taut-10-d vdash-properties-10 by blast
  qed

lemma KBasic2-8[PLM]:
  [◇(φ & ψ) → (◇φ & ◇ψ) in v]
  by (metis CP RM-2 &I oth-class-taut-9-a
          oth-class-taut-9-b vdash-properties-10)

lemma KBasic2-9[PLM]:
  [◇(φ → ψ) ≡ (□φ → ◇ψ) in v]
  apply (PLM-subst-method (¬(□φ)) ∨ (◇ψ) □φ → ◇ψ)
   using oth-class-taut-5-k[equiv-sym] apply simp
  apply (PLM-subst-method (¬φ) ∨ ψ φ → ψ)
   using oth-class-taut-5-k[equiv-sym] apply simp
  apply (PLM-subst-method ◇(¬φ) ¬(□φ))
   using KBasic2-2[equiv-sym] apply simp
  using KBasic2-6 .

lemma KBasic2-10[PLM]:
  [◇(□φ) ≡ (¬(□◇(¬φ))) in v]
  unfolding diamond-def apply (PLM-subst-method φ ¬¬φ)
  using oth-class-taut-4-b oth-class-taut-4-a by auto

lemma KBasic2-11[PLM]:
  [◇◇φ ≡ (¬(□□(¬φ))) in v]
  unfolding diamond-def
  apply (PLM-subst-method □(¬φ) ¬(¬(□(¬φ))))
  using oth-class-taut-4-b oth-class-taut-4-a by auto

lemma KBasic2-12[PLM]: [□(φ ∨ ψ) → (□φ ∨ ◇ψ) in v]
  proof −
    have [□(ψ ∨ φ) → (□(¬ψ) → □φ) in v]
      using CP RM-1-b ∨E(2) by blast
    hence [□(ψ ∨ φ) → (◇ψ ∨ □φ) in v]
      unfolding diamond-def disj-def
      by (meson CP ¬¬E vdash-properties-6)
    thus ?thesis apply −
      apply (PLM-subst-method (◇ψ ∨ □φ) (□φ ∨ ◇ψ))
       apply (simp add: PLM.oth-class-taut-3-e)
      apply (PLM-subst-method (ψ ∨ φ) (φ ∨ ψ))
       apply (simp add: PLM.oth-class-taut-3-e)
      by assumption
  qed

lemma TBasic[PLM]:
  [φ → ◇φ in v]
  unfolding diamond-def
```

apply (*subst contraposition-1*)
apply (*PLM-subst-method* $\Box\neg\varphi$ $\neg\neg\Box\neg\varphi$)
apply (*simp add*: *PLM.oth-class-taut-4-b*)
using *qml-2*[**where** φ=$\neg\varphi$, *axiom-instance*]
by *simp*
lemmas *T*$\Diamond$ = *TBasic*

lemma *S5Basic-1*[*PLM*]:
[$\Diamond\Box\varphi \rightarrow \Box\varphi$ *in v*]
proof (*rule CP*)
assume [$\Diamond\Box\varphi$ *in v*]
hence [$\neg\Box\Diamond\neg\varphi$ *in v*]
using *KBasic2-10*[*equiv-lr*] **by** *simp*
moreover have [$\Diamond(\neg\varphi) \rightarrow \Box\Diamond(\neg\varphi)$ *in v*]
by (*simp add*: *qml-3*[*axiom-instance*])
ultimately have [$\neg\Diamond\neg\varphi$ *in v*]
by (*simp add*: *PLM.modus-tollens-1*)
thus [$\Box\varphi$ *in v*]
unfolding *diamond-def* **apply** −
apply (*PLM-subst-method* $\neg\neg\varphi$ φ)
using *oth-class-taut-4-b*[*equiv-sym*] **apply** *simp*
unfolding *diamond-def* **using** *oth-class-taut-4-b*[*equiv-rl*]
by *simp*
qed
lemmas *5*$\Diamond$ = *S5Basic-1*

lemma *S5Basic-2*[*PLM*]:
[$\Box\varphi \equiv \Diamond\Box\varphi$ *in v*]
using *5*$\Diamond$ *T*$\Diamond$ ≡*I* **by** *blast*

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

lemma *S5Basic-4*[*PLM*]:
[$\varphi \rightarrow \Box\Diamond\varphi$ *in v*]
using *T*$\Diamond$[*deduction*, *THEN S5Basic-3*[*equiv-lr*]]
by (*rule CP*)

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

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

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

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

lemmas *4◇ = S5Basic-8*

lemma *S5Basic-9*[*PLM*]:
[◇◇$\varphi \equiv$ ◇φ *in v*]
using *4◇ T◇* **by** (*rule* ≡*I*)

lemma *S5Basic-10*[*PLM*]:
[□($\varphi \vee$ □ψ) ≡ (□$\varphi \vee$ □ψ) *in v*]
apply (*rule* ≡*I*)
apply (*PLM-subst-goal-method* $\lambda\ \chi$. □($\varphi \vee$ □ψ) → (□$\varphi \vee\ \chi$) ◇□ψ)
using *S5Basic-2*[*equiv-sym*] **apply** *simp*
using *KBasic2-12* **apply** *assumption*
apply (*PLM-subst-goal-method* $\lambda\ \chi$.(□$\varphi \vee\ \chi$) → □($\varphi \vee$ □ψ) □□ψ)
using *S5Basic-7*[*equiv-sym*] **apply** *simp*
using *KBasic2-7* **by** *auto*

lemma *S5Basic-11*[*PLM*]:
[□($\varphi \vee$ ◇ψ) ≡ (□$\varphi \vee$ ◇ψ) *in v*]
apply (*rule* ≡*I*)
apply (*PLM-subst-goal-method* $\lambda\ \chi$. □($\varphi \vee$ ◇ψ) → (□$\varphi \vee\ \chi$) ◇◇ψ)
using *S5Basic-9* **apply** *simp*
using *KBasic2-12* **apply** *assumption*
apply (*PLM-subst-goal-method* $\lambda\ \chi$.(□$\varphi \vee\ \chi$) → □($\varphi \vee$ ◇ψ) □◇ψ)
using *S5Basic-3*[*equiv-sym*] **apply** *simp*
using *KBasic2-7* **by** *assumption*

lemma *S5Basic-12*[*PLM*]:
[◇(φ & ◇ψ) ≡ (◇φ & ◇ψ) *in v*]
proof −
have [□((¬φ) ∨ □(¬ψ)) ≡ (□(¬φ) ∨ □(¬ψ)) *in v*]
using *S5Basic-10* **by** *auto*
hence *1*: [(¬□((¬φ) ∨ □(¬ψ))) ≡ ¬(□(¬φ) ∨ □(¬ψ)) *in v*]
using *oth-class-taut-5-d*[*equiv-lr*] **by** *auto*
have *2*: [(◇(¬((¬φ) ∨ (¬(◇ψ))))) ≡ (¬((¬(◇φ)) ∨ (¬(◇ψ))))) *in v*]
apply (*PLM-subst-method* □¬ψ ¬◇ψ)
using *KBasic2-4* **apply** *simp*
apply (*PLM-subst-method* □¬φ ¬◇φ)
using *KBasic2-4* **apply** *simp*
apply (*PLM-subst-method* (¬□((¬φ) ∨ □(¬ψ))) (◇(¬((¬φ) ∨ (□(¬ψ))))))
unfolding *diamond-def*
apply (*simp add*: *RN oth-class-taut-4-b rule-sub-lem-1-a rule-sub-lem-1-f*)
using *1* **by** *assumption*
show *?thesis*
apply (*PLM-subst-method* ¬((¬φ) ∨ (¬◇ψ)) φ & ◇ψ)
using *oth-class-taut-6-a*[*equiv-sym*] **apply** *simp*
apply (*PLM-subst-method* ¬((¬(◇φ)) ∨ (¬◇ψ)) ◇φ & ◇ψ)
using *oth-class-taut-6-a*[*equiv-sym*] **apply** *simp*
using *2* **by** *assumption*
qed

lemma *S5Basic-13*[*PLM*]:
[◇(φ & (□ψ)) ≡ (◇φ & (□ψ)) *in v*]
apply (*PLM-subst-method* ◇□ψ □ψ)
using *S5Basic-2*[*equiv-sym*] **apply** *simp*
using *S5Basic-12* **by** *simp*

lemma *S5Basic-14*[*PLM*]:
[□(φ → (□ψ)) ≡ □(◇φ → ψ) *in v*]

proof (*rule* ≡*I*; *rule CP*)
assume [□(φ → □ψ) *in v*]
moreover {
have ⋀*v*.[□(φ → □ψ) → (◇φ → ψ) *in v*]
proof (*rule CP*)
fix *v*
assume [□(φ → □ψ) *in v*]
hence [◇φ → ◇□ψ *in v*]
using *K*◇[*deduction*] **by** *auto*
thus [◇φ → ψ *in v*]
using *B*◇ *ded-thm-cor-3* **by** *blast*
qed
hence [□(□(φ → □ψ) → (◇φ → ψ)) *in v*]
by (*rule RN*)
hence [□(□(φ → □ψ)) → □((◇φ → ψ)) *in v*]
using *qml-1*[*axiom-instance*, *deduction*] **by** *auto*
}
ultimately show [□(◇φ → ψ) *in v*]
using *S5Basic-6 CP vdash-properties-10* **by** *meson*
next
assume [□(◇φ → ψ) *in v*]
moreover {
fix *v*
{
assume [□(◇φ → ψ) *in v*]
hence *1*: [□◇φ → □ψ *in v*]
using *qml-1*[*axiom-instance*, *deduction*] **by** *auto*
assume [φ *in v*]
hence [□◇φ *in v*]
using *S5Basic-4*[*deduction*] **by** *auto*
hence [□ψ *in v*]
using *1*[*deduction*] **by** *auto*
}
hence [□(◇φ → ψ) *in v*] ⟹ [φ → □ψ *in v*]
using *CP* **by** *auto*
}
ultimately show [□(φ → □ψ) *in v*]
using *S5Basic-6 RN-2 vdash-properties-10* **by** *blast*
qed

lemma *sc-eq-box-box-1*[*PLM*]:
[□(φ → □φ) → (◇φ ≡ □φ) *in v*]
proof(*rule CP*)
assume *1*: [□(φ → □φ) *in v*]
hence [□(◇φ → φ) *in v*]
using *S5Basic-14*[*equiv-lr*] **by** *auto*
hence [◇φ → φ *in v*]
using *qml-2*[*axiom-instance*, *deduction*] **by** *auto*
moreover from *1* **have** [φ → □φ *in v*]
using *qml-2*[*axiom-instance*, *deduction*] **by** *auto*
ultimately have [◇φ → □φ *in v*]
using *ded-thm-cor-3* **by** *auto*
moreover have [□φ → ◇φ *in v*]
using *qml-2*[*axiom-instance*] *T*◇
by (*rule ded-thm-cor-3*)
ultimately show [◇φ ≡ □φ *in v*]
by (*rule* ≡*I*)
qed

lemma *sc-eq-box-box-2*[*PLM*]:
[□(φ → □φ) → ((¬□φ) ≡ (□(¬φ))) *in v*]
proof (*rule CP*)
assume [□(φ → □φ) *in v*]
hence [(¬□(¬φ)) ≡ □φ *in v*]
using *sc-eq-box-box-1*[*deduction*] **unfolding** *diamond-def* **by** *auto*
thus [((¬□φ) ≡ (□(¬φ))) *in v*]
by (*meson CP* ≡*I* ≡*E*(*3*)
≡*E*(*4*) ¬¬*I* ¬¬*E*)
qed

lemma *sc-eq-box-box-3*[*PLM*]:
[(□(φ → □φ) & □(ψ → □ψ)) → ((□φ ≡ □ψ) → □(φ ≡ ψ)) *in v*]
proof (*rule CP*)
assume *1*: [(□(φ → □φ) & □(ψ → □ψ)) *in v*]
{
assume [□φ ≡ □ψ *in v*]
hence [(□φ & □ψ) ∨ ((¬(□φ)) & (¬(□ψ))) *in v*]
using *oth-class-taut-5-i*[*equiv-lr*] **by** *auto*
moreover {
assume [□φ & □ψ *in v*]
hence [□(φ ≡ ψ) *in v*]
using *KBasic-7*[*deduction*] **by** *auto*
}
moreover {
assume [(¬(□φ)) & (¬(□ψ)) *in v*]
hence [□(¬φ) & □(¬ψ) *in v*]
using *1* &*E* &*I* *sc-eq-box-box-2*[*deduction*, *equiv-lr*]
by *metis*
hence [□((¬φ) & (¬ψ)) *in v*]
using *KBasic-3*[*equiv-rl*] **by** *auto*
hence [□(φ ≡ ψ) *in v*]
using *KBasic-9*[*deduction*] **by** *auto*
}
ultimately have [□(φ ≡ ψ) *in v*]
using *CP* ∨*E*(*1*) **by** *blast*
}
thus [□φ ≡ □ψ → □(φ ≡ ψ) *in v*]
using *CP* **by** *auto*
qed

lemma *derived-S5-rules-1-a*[*PLM*]:
assumes ⋀*v*. [χ *in v*] ⟹ [◇φ → ψ *in v*]
shows [□χ *in v*] ⟹ [φ → □ψ *in v*]
proof −
have [□χ *in v*] ⟹ [□◇φ → □ψ *in v*]
using *assms RM-1-b* **by** *metis*
thus [□χ *in v*] ⟹ [φ → □ψ *in v*]
using *S5Basic-4 vdash-properties-10 CP* **by** *metis*
qed

lemma *derived-S5-rules-1-b*[*PLM*]:
assumes ⋀*v*. [◇φ → ψ *in v*]
shows [φ → □ψ *in v*]
using *derived-S5-rules-1-a all-self-eq-1 assms* **by** *blast*

lemma *derived-S5-rules-2-a*[*PLM*]:

assumes $\bigwedge v.\ [\chi\ in\ v] \Longrightarrow [\varphi \rightarrow \Box\psi\ in\ v]$
shows $[\Box\chi\ in\ v] \Longrightarrow [\Diamond\varphi \rightarrow \psi\ in\ v]$
proof −
have $[\Box\chi\ in\ v] \Longrightarrow [\Diamond\varphi \rightarrow \Diamond\Box\psi\ in\ v]$
using *RM-2-b assms* **by** *metis*
thus $[\Box\chi\ in\ v] \Longrightarrow [\Diamond\varphi \rightarrow \psi\ in\ v]$
using *B◇ vdash-properties-10 CP* **by** *metis*
qed

lemma *derived-S5-rules-2-b[PLM]*:
assumes $\bigwedge v.\ [\varphi \rightarrow \Box\psi\ in\ v]$
shows $[\Diamond\varphi \rightarrow \psi\ in\ v]$
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 α
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 *B◇* **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 α
{
fix v
have $[(\forall\,\alpha.\ \varphi\ \alpha) \rightarrow \varphi\ \alpha\ in\ v]$ **using** *cqt-orig-1* **by** *metis*
}
hence $[\Box(\forall\,\alpha\ .\ \varphi\ \alpha) \rightarrow \Box(\varphi\ \alpha)\ in\ v]$ **using** *RM-1* **by** *auto*
}
hence $[\forall\,\alpha\ .\ \Box(\forall\,\alpha\ .\ \varphi\ \alpha) \rightarrow \Box(\varphi\ \alpha)\ in\ v]$ **using** $\forall I$ **by** *metis*
thus *?thesis* **using** *cqt-orig-2[deduction]* **by** *metis*
qed
lemmas *CBF = BFs-2*

lemma *BFs-3[PLM]*:
$[\Diamond(\exists\ \alpha.\ \varphi\ \alpha) \rightarrow (\exists\ \alpha\ .\ \Diamond(\varphi\ \alpha))\ in\ v]$
proof −
have $[(\forall\,\alpha.\ \Box(\neg(\varphi\ \alpha))) \rightarrow \Box(\forall\,\alpha.\ \neg(\varphi\ \alpha))\ in\ v]$
using *BF* **by** *metis*
hence *1*: $[(\neg(\Box(\forall\,\alpha.\ \neg(\varphi\ \alpha)))) \rightarrow (\neg(\forall\,\alpha.\ \Box(\neg(\varphi\ \alpha))))\ in\ v]$
using *contraposition-1* **by** *simp*

```
    have 2: [◇(¬(∀ α. ¬(φ α))) → (¬(∀ α. □(¬(φ α)))) in v]
      apply (PLM-subst-method ¬□(∀ α . ¬(φ α)) ◇(¬(∀ α. ¬(φ α))))
      using KBasic2-2 1 by simp+
    have [◇(¬(∀ α. ¬(φ α))) → (∃ α . ¬(□(¬(φ α)))) in v]
      apply (PLM-subst-method ¬(∀ α. □(¬(φ α))) ∃ α. ¬(□(¬(φ α))))
       using cqt-further-2 apply metis
      using 2 by metis
    thus ?thesis
      unfolding exists-def diamond-def by auto
  qed
lemmas BF◇ = BFs-3

lemma BFs-4[PLM]:
  [(∃ α . ◇(φ α)) → ◇(∃ α. φ α) in v]
  proof −
    have 1: [□(∀ α . ¬(φ α)) → (∀ α. □(¬(φ α))) in v]
      using CBF by auto
    have 2: [(∃ α . (¬(□(¬(φ α))))) → (¬(□(∀ α. ¬(φ α)))) in v]
      apply (PLM-subst-method ¬(∀ α. □(¬(φ α))) (∃ α . (¬(□(¬(φ α))))))
       using cqt-further-2 apply blast
      using 1 using contraposition-1 by metis
    have [(∃ α . (¬(□(¬(φ α))))) → ◇(¬(∀ α . ¬(φ α))) in v]
      apply (PLM-subst-method ¬(□(∀ α. ¬(φ α))) ◇(¬(∀ α. ¬(φ α))))
       using KBasic2-2 apply blast
      using 2 by assumption
    thus ?thesis
      unfolding diamond-def exists-def by auto
  qed
lemmas CBF◇ = BFs-4

lemma sign-S5-thm-1[PLM]:
  [(∃ α. □(φ α)) → □(∃ α. φ α) in v]
  proof (rule CP)
    assume [∃ α . □(φ α) in v]
    then obtain τ where [□(φ τ) in v]
      by (rule ∃E)
    moreover {
      fix v
      assume [φ τ in v]
      hence [∃ α . φ α in v]
        by (rule ∃I)
    }
    ultimately show [□(∃ α . φ α) in v]
      using RN-2 by blast
  qed
lemmas Buridan = sign-S5-thm-1

lemma sign-S5-thm-2[PLM]:
  [◇(∀ α . φ α) → (∀ α . ◇(φ α)) in v]
  proof −
    {
      fix α
      {
        fix v
        have [(∀ α . φ α) → φ α in v]
          using cqt-orig-1 by metis
      }
      hence [◇(∀ α . φ α) → ◇(φ α) in v]
```

```
      using RM-2 by metis
    }
    hence [∀ α . ◇(∀ α . φ α) → ◇(φ α) in v]
      using ∀I by metis
    thus ?thesis
      using cqt-orig-2[deduction] by metis
  qed
lemmas Buridan◇ = sign-S5-thm-2

lemma sign-S5-thm-3[PLM]:
  [◇(∃ α . φ α & ψ α) → ◇((∃ α . φ α) & (∃ α . ψ α)) in v]
  by (simp only: RM-2 cqt-further-5)

lemma sign-S5-thm-4[PLM]:
  [((□(∀ α. φ α → ψ α)) & (□(∀ α . ψ α → χ α))) → □(∀α. φ α → χ α) in v]
  proof (rule CP)
    assume [□(∀α. φ α → ψ α) & □(∀α. ψ α → χ α) in v]
    hence [□((∀α. φ α → ψ α) & (∀α. ψ α → χ α)) in v]
      using KBasic-3[equiv-rl] by blast
    moreover {
      fix v
      assume [((∀α. φ α → ψ α) & (∀α. ψ α → χ α)) in v]
      hence [(∀ α . φ α → χ α) in v]
        using cqt-basic-9[deduction] by blast
    }
    ultimately show [□(∀α. φ α → χ α) in v]
      using RN-2 by blast
  qed

lemma sign-S5-thm-5[PLM]:
  [((□(∀α. φ α ≡ ψ α)) & (□(∀α. ψ α ≡ χ α))) → (□(∀α. φ α ≡ χ α)) in v]
  proof (rule CP)
    assume [□(∀α. φ α ≡ ψ α) & □(∀α. ψ α ≡ χ α) in v]
    hence [□((∀α. φ α ≡ ψ α) & (∀α. ψ α ≡ χ α)) in v]
      using KBasic-3[equiv-rl] by blast
    moreover {
      fix v
      assume [((∀α. φ α ≡ ψ α) & (∀α. ψ α ≡ χ α)) in v]
      hence [(∀ α . φ α ≡ χ α) in v]
        using cqt-basic-10[deduction] by blast
    }
    ultimately show [□(∀α. φ α ≡ χ α) in v]
      using RN-2 by blast
  qed

lemma id-nec2-1[PLM]:
  [◇((α::'a::id-eq) = β) ≡ (α = β) in v]
  apply (rule ≡I; rule CP)
   using id-nec[equiv-lr] derived-S5-rules-2-b CP modus-ponens apply blast
  using T◇[deduction] by auto

lemma id-nec2-2-Aux:
  [(◇φ) ≡ ψ in v] ⟹ [(¬ψ) ≡ □(¬φ) in v]
  proof −
    assume [(◇φ) ≡ ψ in v]
    moreover have ⋀φ ψ. [(¬φ) ≡ ψ in v] ⟹ [(¬ψ) ≡ φ in v]
      by PLM-solver
    ultimately show ?thesis
```

unfolding *diamond-def* **by** *blast*
qed

lemma *id-nec2-2*[*PLM*]:
[((α::'a::*id-eq*) $\neq$ β) $\equiv$ $\Box$(α $\neq$ β) *in* *v*]
using *id-nec2-1*[*THEN* *id-nec2-2-Aux*] **by** *auto*

lemma *id-nec2-3*[*PLM*]:
[($\Diamond$((α::'a::*id-eq*) $\neq$ β)) $\equiv$ (α $\neq$ β) *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) = (ιx. φ *x*)) $\rightarrow$ ($\exists$ *y* . $\Box$((y^P) = (ιx. φ *x*))) *in* *v*]
proof (*rule* *CP*)
assume [$\exists$ *y*. (y^P) = (ιx. φ *x*) *in* *v*]
then obtain *y* **where** [(y^P) = (ιx. φ *x*) *in* *v*]
by (*rule* $\exists$ *E*)
hence [$\Box$(y^P = (ιx. φ *x*)) *in* *v*]
using *l-identity*[*axiom-instance*, *deduction*, *deduction*]
cqt-1[*axiom-instance*] *all-self-eq-2*[**where** '*a*=ν]
modus-ponens **unfolding** *identity-*ν*-def* **by** *fast*
thus [$\exists$ *y*. $\Box$((y^P) = (ιx. φ *x*)) *in* *v*]
by (*rule* $\exists$ *I*)
qed

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

lemma *en-eq-1*[*PLM*]:
[$\Diamond$⦃*x*,*F*⦄ $\equiv$ $\Box$⦃*x*,*F*⦄ *in* *v*]
using *encoding*[*axiom-instance*] *RN*
sc-eq-box-box-1 *modus-ponens* **by** *blast*
lemma *en-eq-2*[*PLM*]:
[⦃*x*,*F*⦄ $\equiv$ $\Box$⦃*x*,*F*⦄ *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*⦄ *in* *v*]
using *encoding*[*axiom-instance*] *derived-S5-rules-2-b* $\equiv$*I* *T*$\Diamond$ **by** *auto*
lemma *en-eq-4*[*PLM*]:
[(⦃*x*,*F*⦄ $\equiv$ ⦃*y*,*G*⦄) $\equiv$ ($\Box$⦃*x*,*F*⦄ $\equiv$ $\Box$⦃*y*,*G*⦄) *in* *v*]
by (*metis* *CP* *en-eq-2* $\equiv$*I* $\equiv$*E*(*1*) $\equiv$*E*(*2*))
lemma *en-eq-5*[*PLM*]:
[$\Box$(⦃*x*,*F*⦄ $\equiv$ ⦃*y*,*G*⦄) $\equiv$ ($\Box$⦃*x*,*F*⦄ $\equiv$ $\Box$⦃*y*,*G*⦄) *in* *v*]
using $\equiv$*I* *KBasic-6* *encoding*[*axiom-necessitation*, *axiom-instance*]
sc-eq-box-box-3[*deduction*] &*I* **by** *simp*
lemma *en-eq-6*[*PLM*]:
[(⦃*x*,*F*⦄ $\equiv$ ⦃*y*,*G*⦄) $\equiv$ $\Box$(⦃*x*,*F*⦄ $\equiv$ ⦃*y*,*G*⦄) *in* *v*]
using *en-eq-4* *en-eq-5* *oth-class-taut-4-a* $\equiv$*E*(*6*) **by** *meson*
lemma *en-eq-7*[*PLM*]:
[($\neg$⦃*x*,*F*⦄) $\equiv$ $\Box$($\neg$⦃*x*,*F*⦄) *in* *v*]
using *en-eq-3*[*THEN* *id-nec2-2-Aux*] **by** *blast*
lemma *en-eq-8*[*PLM*]:
[$\Diamond$($\neg$⦃*x*,*F*⦄) $\equiv$ ($\neg$⦃*x*,*F*⦄) *in* *v*]
unfolding *diamond-def* **apply** (*PLM-subst-method* ⦃*x*,*F*⦄ $\neg\neg$⦃*x*,*F*⦄)
using *oth-class-taut-4-b* **apply** *simp*
apply (*PLM-subst-method* ⦃*x*,*F*⦄ $\Box$⦃*x*,*F*⦄)

using *en-eq-2* **apply** *simp*
using *oth-class-taut-4-a* **by** *assumption*

lemma *en-eq-9*[*PLM*]:
[$\Diamond(\neg\{\!|x,F|\!\}) \equiv \Box(\neg\{\!|x,F|\!\})$ *in* v]
using *en-eq-8 en-eq-7* $\equiv$*E*(*5*) **by** *blast*

lemma *en-eq-10*[*PLM*]:
[$\mathcal{A}\{\!|x,F|\!\} \equiv \{\!|x,F|\!\}$ *in* v]
apply (*rule* $\equiv$*I*)
using *encoding*[*axiom-actualization*, *axiom-instance*,
THEN logic-actual-nec-2[*axiom-instance*, *equiv-lr*],
deduction, *THEN qml-act-2*[*axiom-instance*, *equiv-rl*],
THEN en-eq-2[*equiv-rl*]] *CP*
apply *simp*
using *encoding*[*axiom-instance*] *nec-imp-act ded-thm-cor-3* **by** *blast*

A.9.11. The Theory of Relations

lemma *beta-equiv-eq-1-1*[*PLM*]:
assumes *IsProperInX* φ
and *IsProperInX* ψ
and $\bigwedge x.[\varphi\ (x^P) \equiv \psi\ (x^P)$ *in* $v]$
shows [$(\!|\boldsymbol{\lambda}\ y.\ \varphi\ (y^P),\ x^P|\!) \equiv (\!|\boldsymbol{\lambda}\ y.\ \psi\ (y^P),\ x^P|\!)$ *in* v]
using *lambda-predicates-2-1*[*OF assms*(*1*), *axiom-instance*]
using *lambda-predicates-2-1*[*OF assms*(*2*), *axiom-instance*]
using *assms*(*3*) **by** (*meson* $\equiv$*E*(*6*) *oth-class-taut-4-a*)

lemma *beta-equiv-eq-1-2*[*PLM*]:
assumes *IsProperInXY* φ
and *IsProperInXY* ψ
and $\bigwedge x\ y.[\varphi\ (x^P)\ (y^P) \equiv \psi\ (x^P)\ (y^P)$ *in* $v]$
shows [$(\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \varphi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \psi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)$ *in* v]
using *lambda-predicates-2-2*[*OF assms*(*1*), *axiom-instance*]
using *lambda-predicates-2-2*[*OF assms*(*2*), *axiom-instance*]
using *assms*(*3*) **by** (*meson* $\equiv$*E*(*6*) *oth-class-taut-4-a*)

lemma *beta-equiv-eq-1-3*[*PLM*]:
assumes *IsProperInXYZ* φ
and *IsProperInXYZ* ψ
and $\bigwedge x\ y\ z.[\varphi\ (x^P)\ (y^P)\ (z^P) \equiv \psi\ (x^P)\ (y^P)\ (z^P)$ *in* $v]$
shows [$(\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \psi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)$ *in* v]
using *lambda-predicates-2-3*[*OF assms*(*1*), *axiom-instance*]
using *lambda-predicates-2-3*[*OF assms*(*2*), *axiom-instance*]
using *assms*(*3*) **by** (*meson* $\equiv$*E*(*6*) *oth-class-taut-4-a*)

lemma *beta-equiv-eq-2-1*[*PLM*]:
assumes *IsProperInX* φ
and *IsProperInX* ψ
shows [$(\Box(\forall\ x\ .\ \varphi\ (x^P) \equiv \psi\ (x^P))) \rightarrow$
$(\Box(\forall\ x\ .\ (\!|\boldsymbol{\lambda}\ y.\ \varphi\ (y^P),\ x^P|\!) \equiv (\!|\boldsymbol{\lambda}\ y.\ \psi\ (y^P),\ x^P|\!)))$ *in* v]
apply (*rule qml-1*[*axiom-instance*, *deduction*])
apply (*rule RN*)
proof (*rule CP*, *rule* $\forall$ *I*)
fix $v\ x$
assume [$\forall\ x.\ \varphi\ (x^P) \equiv \psi\ (x^P)$ *in* v]
hence $\bigwedge x.[\varphi\ (x^P) \equiv \psi\ (x^P)$ *in* $v]$
by *PLM-solver*

thus $[(\!|\boldsymbol{\lambda}\ y.\ \varphi\ (y^P),\ x^P|\!) \equiv (\!|\boldsymbol{\lambda}\ y.\ \psi\ (y^P),\ x^P|\!)\ \textit{in}\ v]$
using *assms beta-equiv-eq-1-1* **by** *auto*
qed

lemma *beta-equiv-eq-2-2*[*PLM*]:
assumes *IsProperInXY* φ
and *IsProperInXY* ψ
shows $[(\Box(\forall\ x\ y\ .\ \varphi\ (x^P)\ (y^P) \equiv \psi\ (x^P)\ (y^P))) \rightarrow$
$(\Box(\forall\ x\ y\ .\ (\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \varphi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \psi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)))\ \textit{in}\ v]$
apply (*rule qml-1*[*axiom-instance*, *deduction*])
apply (*rule RN*)
proof (*rule CP*, *rule* $\forall I$, *rule* $\forall I$)
fix *v x y*
assume $[\forall x\ y.\ \varphi\ (x^P)\ (y^P) \equiv \psi\ (x^P)\ (y^P)\ \textit{in}\ v]$
hence $(\bigwedge x\ y.[\varphi\ (x^P)\ (y^P) \equiv \psi\ (x^P)\ (y^P)\ \textit{in}\ v])$
by (*meson* $\forall E$)
thus $[(\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \varphi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \psi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!)\ \textit{in}\ v]$
using *assms beta-equiv-eq-1-2* **by** *auto*
qed

lemma *beta-equiv-eq-2-3*[*PLM*]:
assumes *IsProperInXYZ* φ
and *IsProperInXYZ* ψ
shows $[(\Box(\forall\ x\ y\ z\ .\ \varphi\ (x^P)\ (y^P)\ (z^P) \equiv \psi\ (x^P)\ (y^P)\ (z^P))) \rightarrow$
$(\Box(\forall\ x\ y\ z\ .\ (\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \psi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)))\ \textit{in}\ v]$
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 $[\forall x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P) \equiv \psi\ (x^P)\ (y^P)\ (z^P)\ \textit{in}\ v]$
hence $(\bigwedge x\ y\ z.[\varphi\ (x^P)\ (y^P)\ (z^P) \equiv \psi\ (x^P)\ (y^P)\ (z^P)\ \textit{in}\ v])$
by (*meson* $\forall E$)
thus $[(\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)$
$\equiv (\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \psi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!)\ \textit{in}\ v]$
using *assms beta-equiv-eq-1-3* **by** *auto*
qed

lemma *beta-C-meta-1*[*PLM*]:
assumes *IsProperInX* φ
shows $[(\!|\boldsymbol{\lambda}\ y.\ \varphi\ (y^P),\ x^P|\!) \equiv \varphi\ (x^P)\ \textit{in}\ v]$
using *lambda-predicates-2-1*[*OF assms*, *axiom-instance*] **by** *auto*

lemma *beta-C-meta-2*[*PLM*]:
assumes *IsProperInXY* φ
shows $[(\!|\boldsymbol{\lambda}^2\ (\lambda\ x\ y.\ \varphi\ (x^P)\ (y^P)),\ x^P,\ y^P|\!) \equiv \varphi\ (x^P)\ (y^P)\ \textit{in}\ v]$
using *lambda-predicates-2-2*[*OF assms*, *axiom-instance*] **by** *auto*

lemma *beta-C-meta-3*[*PLM*]:
assumes *IsProperInXYZ* φ
shows $[(\!|\boldsymbol{\lambda}^3\ (\lambda\ x\ y\ z.\ \varphi\ (x^P)\ (y^P)\ (z^P)),\ x^P,\ y^P,\ z^P|\!) \equiv \varphi\ (x^P)\ (y^P)\ (z^P)\ \textit{in}\ v]$
using *lambda-predicates-2-3*[*OF assms*, *axiom-instance*] **by** *auto*

lemma *relations-1*[*PLM*]:
assumes *IsProperInX* φ
shows $[\exists\ F.\ \Box(\forall\ x.\ (\!|F,x^P|\!) \equiv \varphi\ (x^P))\ \textit{in}\ v]$

using *assms* **apply** − **by** *PLM-solver*

lemma *relations-2*[*PLM*]:
assumes *IsProperInXY* φ
shows [$\exists$ F. $\Box(\forall$ x y. $(\!|F,x^P,y^P|\!) \equiv \varphi\ (x^P)\ (y^P))$ *in* v]
using *assms* **apply** − **by** *PLM-solver*

lemma *relations-3*[*PLM*]:
assumes *IsProperInXYZ* φ
shows [$\exists$ F. $\Box(\forall$ x y z. $(\!|F,x^P,y^P,z^P|\!) \equiv \varphi\ (x^P)\ (y^P)\ (z^P))$ *in* v]
using *assms* **apply** − **by** *PLM-solver*

lemma *prop-equiv*[*PLM*]:
shows [$(\forall$ x . $(\{\!|x^P,F|\!\} \equiv \{\!|x^P,G|\!\})) \rightarrow F = G$ *in* v]
proof (*rule CP*)
assume *1*: [$\forall x$. $\{\!|x^P,F|\!\} \equiv \{\!|x^P,G|\!\}$ *in* v]
{
fix x
have [$\{\!|x^P,F|\!\} \equiv \{\!|x^P,G|\!\}$ *in* v]
using *1* **by** (*rule* $\forall E$)
hence [$\Box(\{\!|x^P,F|\!\} \equiv \{\!|x^P,G|\!\})$ *in* v]
using *PLM.en-eq-6* $\equiv E(1)$ **by** *blast*
}
hence [$\forall x$. $\Box(\{\!|x^P,F|\!\} \equiv \{\!|x^P,G|\!\})$ *in* v]
by (*rule* $\forall I$)
thus [$F = G$ *in* v]
unfolding *identity-defs*
by (*rule BF*[*deduction*])
qed

lemma *propositions-lemma-1*[*PLM*]:
[$\boldsymbol{\lambda}^0\ \varphi = \varphi$ *in* v]
using *lambda-predicates-3-0*[*axiom-instance*] **.**

lemma *propositions-lemma-2*[*PLM*]:
[$\boldsymbol{\lambda}^0\ \varphi \equiv \varphi$ *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 $\bigwedge x.$[$\boldsymbol{\mathcal{A}}(\varphi\ x \equiv \psi\ x)$ *in* v]
shows [$(\chi{::}\kappa{\Rightarrow}o)\ (\boldsymbol{\iota}x.\ \varphi\ x) = \chi\ (\boldsymbol{\iota}x.\ \psi\ x)$ *in* v]
proof −
have [$\boldsymbol{\lambda}^0\ (\chi\ (\boldsymbol{\iota}x.\ \varphi\ x)) = \boldsymbol{\lambda}^0\ (\chi\ (\boldsymbol{\iota}x.\ \psi\ x))$ *in* v]
using *assms lambda-predicates-4-0*[*axiom-instance*]
by *blast*
hence [$(\chi\ (\boldsymbol{\iota}x.\ \varphi\ x)) = \boldsymbol{\lambda}^0\ (\chi\ (\boldsymbol{\iota}x.\ \psi\ x))$ *in* v]
using *propositions-lemma-1*[*THEN id-eq-prop-prop-8-b*[*deduction*]]
id-eq-prop-prop-9-b[*deduction*] **&***I*
by *blast*
thus *?thesis*
using *propositions-lemma-1 id-eq-prop-prop-9-b*[*deduction*] **&***I*
by *blast*
qed

lemma *propositions*[*PLM*]:
[$\exists$ p . $\Box(p \equiv p')$ *in* v]

by *PLM-solver*

lemma *pos-not-equiv-then-not-eq*[*PLM*]:
[$\Diamond(\neg(\forall\, x.\ (\!|F,x^P|\!) \equiv (\!|G,x^P|\!))) \rightarrow F \neq G$ *in* v]
unfolding *diamond-def*
proof (*subst contraposition-1*[*symmetric*], *rule CP*)
assume [$F = G$ *in* v]
thus [$\Box(\neg(\neg(\forall\, x.\ (\!|F,x^P|\!) \equiv (\!|G,x^P|\!))))$ *in* v]
apply (*rule l-identity*[*axiom-instance*, *deduction*, *deduction*])
by *PLM-solver*
qed

lemma *thm-relation-negation-1-1*[*PLM*]:
[$(\!|F^-,\ x^P|\!) \equiv \neg(\!|F,\ x^P|\!)$ *in* v]
unfolding *propnot-defs*
apply (*rule lambda-predicates-2-1*[*axiom-instance*])
by *show-proper*

lemma *thm-relation-negation-1-2*[*PLM*]:
[$(\!|F^-,\ x^P,\ y^P|\!) \equiv \neg(\!|F,\ x^P,\ y^P|\!)$ *in* v]
unfolding *propnot-defs*
apply (*rule lambda-predicates-2-2*[*axiom-instance*])
by *show-proper*

lemma *thm-relation-negation-1-3*[*PLM*]:
[$(\!|F^-,\ x^P,\ y^P,\ z^P|\!) \equiv \neg(\!|F,\ x^P,\ y^P,\ z^P|\!)$ *in* v]
unfolding *propnot-defs*
apply (*rule lambda-predicates-2-3*[*axiom-instance*])
by *show-proper*

lemma *thm-relation-negation-2-1*[*PLM*]:
[$(\neg(\!|F^-,\ x^P|\!)) \equiv (\!|F,\ x^P|\!)$ *in* v]
using *thm-relation-negation-1-1*[*THEN oth-class-taut-5-d*[*equiv-lr*]]
apply $-$ **by** *PLM-solver*

lemma *thm-relation-negation-2-2*[*PLM*]:
[$(\neg(\!|F^-,\ x^P,\ y^P|\!)) \equiv (\!|F,\ x^P,\ y^P|\!)$ *in* v]
using *thm-relation-negation-1-2*[*THEN oth-class-taut-5-d*[*equiv-lr*]]
apply $-$ **by** *PLM-solver*

lemma *thm-relation-negation-2-3*[*PLM*]:
[$(\neg(\!|F^-,\ x^P,\ y^P,\ z^P|\!)) \equiv (\!|F,\ x^P,\ y^P,\ z^P|\!)$ *in* v]
using *thm-relation-negation-1-3*[*THEN oth-class-taut-5-d*[*equiv-lr*]]
apply $-$ **by** *PLM-solver*

lemma *thm-relation-negation-3*[*PLM*]:
[$(p)^- \equiv \neg p$ *in* v]
unfolding *propnot-defs*
using *propositions-lemma-2* **by** *simp*

lemma *thm-relation-negation-4*[*PLM*]:
[$(\neg((p{::}\mathrm{o})^-)) \equiv p$ *in* v]
using *thm-relation-negation-3*[*THEN oth-class-taut-5-d*[*equiv-lr*]]
apply $-$ **by** *PLM-solver*

lemma *thm-relation-negation-5-1*[*PLM*]:
[$(F{::}\Pi_1) \neq (F^-)$ *in* v]
using *id-eq-prop-prop-2*[*deduction*]

l-identity[**where** $\varphi{=}\lambda\ G\ .\ (\!|G,x^P|\!) \equiv (\!|F^-,x^P|\!)$, *axiom-instance*, *deduction*, *deduction*]
oth-class-taut-4-a thm-relation-negation-1-1 $\equiv E(5)$
oth-class-taut-1-b modus-tollens-1 CP
by *meson*

lemma *thm-relation-negation-5-2*[*PLM*]:
$[(F{::}\Pi_2) \neq (F^-)\ in\ v]$
using *id-eq-prop-prop-5-a*[*deduction*]
l-identity[**where** $\varphi{=}\lambda\ G\ .\ (\!|G,x^P,y^P|\!) \equiv (\!|F^-,x^P,y^P|\!)$, *axiom-instance*, *deduction*, *deduction*]
oth-class-taut-4-a thm-relation-negation-1-2 $\equiv E(5)$
oth-class-taut-1-b modus-tollens-1 CP
by *meson*

lemma *thm-relation-negation-5-3*[*PLM*]:
$[(F{::}\Pi_3) \neq (F^-)\ in\ v]$
using *id-eq-prop-prop-5-b*[*deduction*]
l-identity[**where** $\varphi{=}\lambda\ G\ .\ (\!|G,x^P,y^P,z^P|\!) \equiv (\!|F^-,x^P,y^P,z^P|\!)$, *axiom-instance*, *deduction*, *deduction*]
oth-class-taut-4-a thm-relation-negation-1-3 $\equiv E(5)$
oth-class-taut-1-b modus-tollens-1 CP
by *meson*

lemma *thm-relation-negation-6*[*PLM*]:
$[(p{::}o) \neq (p^-)\ in\ v]$
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::Π_1) $\equiv$ *NonContingent* (F^-) *in* v]
proof (*rule* $\equiv I$; *rule CP*)
assume [*NonContingent F in v*]
hence [$\Box(\forall x.(\!|F,x^P|\!)) \vee \Box(\forall x.\neg(\!|F,x^P|\!))$ *in* v]
unfolding *NonContingent-def Necessary-defs Impossible-defs* .
hence [$\Box(\forall x.\ \neg(\!|F^-,x^P|\!)) \vee \Box(\forall x.\ \neg(\!|F,x^P|\!))$ *in* v]
apply −
apply (*PLM-subst-method* $\lambda\ x\ .\ (\!|F,x^P|\!)\ \lambda\ x\ .\ \neg(\!|F^-,x^P|\!)$)
using *thm-relation-negation-2-1*[*equiv-sym*] **by** *auto*
hence [$\Box(\forall x.\ \neg(\!|F^-,x^P|\!)) \vee \Box(\forall x.\ (\!|F^-,x^P|\!))$ *in* v]
apply −
apply (*PLM-subst-goal-method*
$\lambda\ \varphi\ .\ \Box(\forall x.\ \neg(\!|F^-,x^P|\!)) \vee \Box(\forall x.\ \varphi\ x)\ \lambda\ x\ .\ \neg(\!|F,x^P|\!)$)
using *thm-relation-negation-1-1*[*equiv-sym*] **by** *auto*
hence [$\Box(\forall x.\ (\!|F^-,x^P|\!)) \vee \Box(\forall x.\ \neg(\!|F^-,x^P|\!))$ *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 [$\Box(\forall x.\ \neg(\!|F^-,x^P|\!)) \vee \Box(\forall x.\ (\!|F^-,x^P|\!))$ *in* v]
unfolding *NonContingent-def Necessary-defs Impossible-defs*
by (*rule oth-class-taut-3-e*[*equiv-lr*])
hence [$\Box(\forall x.(\!|F,x^P|\!)) \vee \Box(\forall x.(\!|F^-,x^P|\!))$ *in* v]
apply −
apply (*PLM-subst-method* $\lambda\ x\ .\ \neg(\!|F^-,x^P|\!)\ \lambda\ x\ .\ (\!|F,x^P|\!)$)
using *thm-relation-negation-2-1* **by** *auto*
hence [$\Box(\forall x.\ (\!|F,x^P|\!)) \vee \Box(\forall x.\ \neg(\!|F,x^P|\!))$ *in* v]
apply −
apply (*PLM-subst-method* $\lambda\ x\ .\ (\!|F^-,x^P|\!)\ \lambda\ x\ .\ \neg(\!|F,x^P|\!)$)
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* $F \equiv \Diamond(\exists\ x\ .\ (\!|F,x^P|\!))$ **&** $\Diamond(\exists\ x\ .\ \neg(\!|F,x^P|\!))$ *in* v]
proof (*rule* $\equiv I$; *rule CP*)
assume [*Contingent F in v*]
hence [$\neg(\Box(\forall x.(\!|F,x^P|\!)) \vee \Box(\forall x.\neg(\!|F,x^P|\!)))$ *in* v]
unfolding *Contingent-def Necessary-defs Impossible-defs* .
hence [$(\neg\Box(\forall x.(\!|F,x^P|\!)))$ **&** $(\neg\Box(\forall x.\neg(\!|F,x^P|\!)))$ *in* v]
by (*rule oth-class-taut-6-d*[*equiv-lr*])
hence [$(\Diamond\neg(\forall x.\neg(\!|F,x^P|\!)))$ **&** $(\Diamond\neg(\forall x.(\!|F,x^P|\!)))$ *in* v]
using *KBasic2-2*[*equiv-lr*] **&***I* **&***E* **by** *meson*
thus [$(\Diamond(\exists\ x.(\!|F,x^P|\!)))$ **&** $(\Diamond(\exists x.\ \neg(\!|F,x^P|\!)))$ *in* v]
unfolding *exists-def* **apply** −
apply (*PLM-subst-method* $\lambda\ x\ .\ (\!|F,x^P|\!)\ \lambda\ x\ .\ \neg\neg(\!|F,x^P|\!)$)
using *oth-class-taut-4-b* **by** *auto*
next
assume [$(\Diamond(\exists\ x.(\!|F,x^P|\!)))$ **&** $(\Diamond(\exists x.\ \neg(\!|F,x^P|\!)))$ *in* v]
hence [$(\Diamond\neg(\forall x.\neg(\!|F,x^P|\!)))$ **&** $(\Diamond\neg(\forall x.(\!|F,x^P|\!)))$ *in* v]
unfolding *exists-def* **apply** −
apply (*PLM-subst-goal-method*
$\lambda\ \varphi\ .\ (\Diamond\neg(\forall x.\neg(\!|F,x^P|\!)))$ **&** $(\Diamond\neg(\forall x.\ \varphi\ x))\ \lambda\ x\ .\ \neg\neg(\!|F,x^P|\!)$)
using *oth-class-taut-4-b*[*equiv-sym*] **by** *auto*

hence $[(\neg\Box(\forall\, x.(\!|F,x^P|\!))) \;\&\; (\neg\Box(\forall\, x.\neg(\!|F,x^P|\!)))\ in\ v]$
using *KBasic2-2*[*equiv-rl*] &*I* &*E* **by** *meson*
hence $[\neg(\Box(\forall\, x.(\!|F,x^P|\!)) \vee \Box(\forall\, x.\neg(\!|F,x^P|\!)))\ in\ v]$
by (*rule oth-class-taut-6-d*[*equiv-rl*])
thus [*Contingent F in v*]
unfolding *Contingent-def Necessary-defs Impossible-defs* .
qed

lemma *thm-cont-prop-3*[*PLM*]:
[*Contingent* $(F{::}\Pi_1) \equiv$ *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*]:
$[\Diamond(\exists\ x\ .\ (\!|F,x^P|\!)\ \&\ (\Diamond(\neg(\!|F,x^P|\!)))) \equiv \Diamond(\exists\ x\ .\ ((\neg(\!|F,x^P|\!))\ \&\ \Diamond(\!|F,x^P|\!)))in\ v]$
proof −
have $[\Diamond(\exists\ x\ .\ (\!|F,x^P|\!)\ \&\ (\Diamond(\neg(\!|F,x^P|\!))))\ in\ v]$
$= [(\exists\ x\ .\ \Diamond((\!|F,x^P|\!)\ \&\ \Diamond(\neg(\!|F,x^P|\!))))\ in\ v]$
using *BF*◇[*deduction*] *CBF*◇[*deduction*] **by** *fast*
also have ... $= [\exists\ x\ .\ (\Diamond(\!|F,x^P|\!)\ \&\ \Diamond(\neg(\!|F,x^P|\!)))\ in\ v]$
apply (*PLM-subst-method*
$\lambda\ x\ .\ \Diamond((\!|F,x^P|\!)\ \&\ \Diamond(\neg(\!|F,x^P|\!)))$
$\lambda\ x\ .\ \Diamond(\!|F,x^P|\!)\ \&\ \Diamond(\neg(\!|F,x^P|\!)))$
using *S5Basic-12* **by** *auto*
also have ... $= [\exists\ x\ .\ \Diamond(\neg(\!|F,x^P|\!))\ \&\ \Diamond(\!|F,x^P|\!)\ in\ v]$
apply (*PLM-subst-method*
$\lambda\ x\ .\ \Diamond(\!|F,x^P|\!)\ \&\ \Diamond(\neg(\!|F,x^P|\!))$
$\lambda\ x\ .\ \Diamond(\neg(\!|F,x^P|\!))\ \&\ \Diamond(\!|F,x^P|\!))$
using *oth-class-taut-3-b* **by** *auto*
also have ... $= [\exists\ x\ .\ \Diamond((\neg(\!|F,x^P|\!))\ \&\ \Diamond(\!|F,x^P|\!))\ in\ v]$
apply (*PLM-subst-method*
$\lambda\ x\ .\ \Diamond(\neg(\!|F,x^P|\!))\ \&\ \Diamond(\!|F,x^P|\!)$
$\lambda\ x\ .\ \Diamond((\neg(\!|F,x^P|\!))\ \&\ \Diamond(\!|F,x^P|\!)))$
using *S5Basic-12*[*equiv-sym*] **by** *auto*
also have ... $= [\Diamond\ (\exists\ x\ .\ ((\neg(\!|F,x^P|\!))\ \&\ \Diamond(\!|F,x^P|\!)))\ in\ v]$
using *CBF*◇[*deduction*] *BF*◇[*deduction*] **by** *fast*
finally show *?thesis* **using** ≡*I CP* **by** *blast*
qed

lemma *lem-cont-e-2*[*PLM*]:
$[\Diamond(\exists\ x\ .\ (\!|F,x^P|\!)\ \&\ \Diamond(\neg(\!|F,x^P|\!))) \equiv \Diamond(\exists\ x\ .\ (\!|F^-,x^P|\!)\ \&\ \Diamond(\neg(\!|F^-,x^P|\!)))\ in\ v]$
apply (*PLM-subst-method* $\lambda\ x\ .\ (\!|F,x^P|\!)\ \lambda\ x\ .\ \neg(\!|F^-,x^P|\!)$)
using *thm-relation-negation-2-1*[*equiv-sym*] **apply** *simp*
apply (*PLM-subst-method* $\lambda\ x\ .\ \neg(\!|F,x^P|\!)\ \lambda\ x\ .\ (\!|F^-,x^P|\!)$)
using *thm-relation-negation-1-1*[*equiv-sym*] **apply** *simp*
using *lem-cont-e* **by** *simp*

lemma *thm-cont-e-1*[*PLM*]:
$[\Diamond(\exists\ x\ .\ ((\neg(\!|E!,x^P|\!))\ \&\ (\Diamond(\!|E!,x^P|\!))))\ in\ v]$
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::Π_1) G . ($F \neq G$ & *Contingent F* & *Contingent G*) *in v*]
apply (*rule-tac* α=$E!$ **in** $\exists\, I$, *rule-tac* α=$E!^-$ **in** $\exists\, I$)
using *thm-cont-e-2 thm-cont-e-3 thm-relation-negation-5-1* &I **by** *auto*

context
begin
qualified definition L **where** $L \equiv (\boldsymbol{\lambda}\ x\ .\ (\!|E!, x^P|\!) \rightarrow (\!|E!, x^P|\!))$

lemma *thm-noncont-e-e-1*[*PLM*]:
[*Necessary L in v*]
unfolding *Necessary-defs L-def* **apply** (*rule RN*, *rule* $\forall\, I$)
apply (*rule lambda-predicates-2-1*[*axiom-instance*, *equiv-rl*])
apply *show-proper*
using *if-p-then-p* **.**

lemma *thm-noncont-e-e-2*[*PLM*]:
[*Impossible* (L^-) *in v*]
unfolding *Impossible-defs L-def* **apply** (*rule RN*, *rule* $\forall\, I$)
apply (*rule thm-relation-negation-2-1*[*equiv-rl*])
apply (*rule lambda-predicates-2-1*[*axiom-instance*, *equiv-rl*])
apply *show-proper*
using *if-p-then-p* **.**

lemma *thm-noncont-e-e-3*[*PLM*]:
[*NonContingent* (L) *in v*]
unfolding *NonContingent-def* **using** *thm-noncont-e-e-1*
by (*rule* $\vee I(1)$)

lemma *thm-noncont-e-e-4*[*PLM*]:
[*NonContingent* (L^-) *in v*]
unfolding *NonContingent-def* **using** *thm-noncont-e-e-2*
by (*rule* $\vee I(2)$)

lemma *thm-noncont-e-e-5*[*PLM*]:
[$\exists$ (F::Π_1) G . $F \neq G$ & *NonContingent F* & *NonContingent G in v*]
apply (*rule-tac* α=L **in** $\exists\, I$, *rule-tac* α=L^- **in** $\exists\, I$)
using $\exists\, I$ *thm-relation-negation-5-1 thm-noncont-e-e-3*
thm-noncont-e-e-4 &I
by *simp*

lemma *four-distinct-1*[*PLM*]:
[*NonContingent* (F::Π_1) $\rightarrow$ $\neg$($\exists$ G . (*Contingent G* & $G = F$)) *in v*]
proof (*rule CP*)
assume [*NonContingent F in v*]
hence [$\neg$(*Contingent F*) *in v*]
unfolding *NonContingent-def Contingent-def*
apply $-$ **by** *PLM-solver*
moreover **{**
assume [$\exists$ G . *Contingent G* & $G = F$ *in v*]
then obtain P **where** [*Contingent P* & $P = F$ *in v*]

```
      by (rule ∃ E)
     hence [Contingent F in v]
      using &E l-identity[axiom-instance, deduction, deduction]
      by blast
  }
  ultimately show [¬(∃ G. Contingent G & G = F) in v]
   using modus-tollens-1 CP by blast
qed

lemma four-distinct-2[PLM]:
 [Contingent (F::Π1) → ¬(∃ G . (NonContingent G & G = F)) in v]
 proof (rule CP)
  assume [Contingent F in v]
  hence [¬(NonContingent F) in v]
   unfolding NonContingent-def Contingent-def
   apply − by PLM-solver
  moreover {
    assume [∃ G . NonContingent G & G = F in v]
    then obtain P where [NonContingent P & P = F in v]
     by (rule ∃ E)
    hence [NonContingent F in v]
     using &E l-identity[axiom-instance, deduction, deduction]
     by blast
  }
  ultimately show [¬(∃ G. NonContingent G & G = F) in v]
   using modus-tollens-1 CP by blast
qed

 lemma four-distinct-3[PLM]:
  [L ≠ (L⁻) & L ≠ E! & L ≠ (E!⁻) & (L⁻) ≠ E!
   & (L⁻) ≠ (E!⁻) & E! ≠ (E!⁻) in v]
  proof (rule &I)+
   show [L ≠ (L⁻) in v]
   by (rule thm-relation-negation-5-1)
  next
   {
    assume [L = E! in v]
    hence [NonContingent L & L = E! in v]
     using thm-noncont-e-e-3 &I by auto
    hence [∃ G . NonContingent G & G = E! in v]
     using thm-noncont-e-e-3 &I ∃ I by fast
   }
   thus [L ≠ E! in v]
    using four-distinct-2[deduction, OF thm-cont-e-2]
          modus-tollens-1 CP
    by blast
  next
   {
    assume [L = (E!⁻) in v]
    hence [NonContingent L & L = (E!⁻) in v]
     using thm-noncont-e-e-3 &I by auto
    hence [∃ G . NonContingent G & G = (E!⁻) in v]
     using thm-noncont-e-e-3 &I ∃ I by fast
   }
   thus [L ≠ (E!⁻) in v]
    using four-distinct-2[deduction, OF thm-cont-e-3]
          modus-tollens-1 CP
    by blast
```

```
    next
      {
        assume [(L⁻) = E! in v]
        hence [NonContingent (L⁻) & (L⁻) = E! in v]
          using thm-noncont-e-e-4 &I by auto
        hence [∃ G . NonContingent G & G = E! in v]
          using thm-noncont-e-e-3 &I ∃ I by fast
      }
      thus [(L⁻) ≠ E! in v]
        using four-distinct-2[deduction, OF thm-cont-e-2]
              modus-tollens-1 CP
        by blast
    next
      {
        assume [(L⁻) = (E!⁻) in v]
        hence [NonContingent (L⁻) & (L⁻) = (E!⁻) in v]
          using thm-noncont-e-e-4 &I by auto
        hence [∃ G . NonContingent G & G = (E!⁻) in v]
          using thm-noncont-e-e-3 &I ∃ I by fast
      }
      thus [(L⁻) ≠ (E!⁻) in v]
        using four-distinct-2[deduction, OF thm-cont-e-3]
              modus-tollens-1 CP
        by blast
    next
      show [E! ≠ (E!⁻) in v]
        by (rule thm-relation-negation-5-1)
    qed
end

lemma thm-cont-propos-1[PLM]:
  [NonContingent (p::o) ≡ NonContingent (p⁻) in v]
  proof (rule ≡I; rule CP)
    assume [NonContingent p in v]
    hence [□p ∨ □¬p in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs .
    hence [□(¬(p⁻)) ∨ □(¬p) in v]
      apply −
      apply (PLM-subst-method p ¬(p⁻))
      using thm-relation-negation-4[equiv-sym] by auto
    hence [□(¬(p⁻)) ∨ □(p⁻) in v]
      apply −
      apply (PLM-subst-goal-method λφ . □(¬(p⁻)) ∨ □(φ) ¬p)
      using thm-relation-negation-3[equiv-sym] by auto
    hence [□(p⁻) ∨ □(¬(p⁻)) in v]
      by (rule oth-class-taut-3-e[equiv-lr])
    thus [NonContingent (p⁻) in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs .
  next
    assume [NonContingent (p⁻) in v]
    hence [□(¬(p⁻)) ∨ □(p⁻) in v]
      unfolding NonContingent-def Necessary-defs Impossible-defs
      by (rule oth-class-taut-3-e[equiv-lr])
    hence [□(p) ∨ □(p⁻) in v]
      apply −
      apply (PLM-subst-goal-method λφ . □φ ∨ □(p⁻) ¬(p⁻))
      using thm-relation-negation-4 by auto
    hence [□(p) ∨ □(¬p) in v]
```

apply −
apply (*PLM-subst-method* p^- $\neg p$)
using *thm-relation-negation-3* **by** *auto*
thus [*NonContingent p in v*]
unfolding *NonContingent-def Necessary-defs Impossible-defs* .
qed

lemma *thm-cont-propos-2*[*PLM*]:
[*Contingent p* ≡ ◇*p* & ◇(¬*p*) *in v*]
proof (*rule* ≡*I*; *rule CP*)
assume [*Contingent p in v*]
hence [¬(□*p* ∨ □(¬*p*)) *in v*]
unfolding *Contingent-def Necessary-defs Impossible-defs* .
hence [(¬□*p*) & (¬□(¬*p*)) *in v*]
by (*rule oth-class-taut-6-d*[*equiv-lr*])
hence [(◇¬(¬*p*)) & (◇¬*p*) *in v*]
using *KBasic2-2*[*equiv-lr*] &*I* &*E* **by** *meson*
thus [(◇*p*) & (◇(¬*p*)) *in v*]
apply − **apply** *PLM-solver*
apply (*PLM-subst-method* ¬¬*p p*)
using *oth-class-taut-4-b*[*equiv-sym*] **by** *auto*
next
assume [(◇*p*) & (◇¬(*p*)) *in v*]
hence [(◇¬(¬*p*)) & (◇¬(*p*)) *in v*]
apply − **apply** *PLM-solver*
apply (*PLM-subst-method p* ¬¬*p*)
using *oth-class-taut-4-b* **by** *auto*
hence [(¬□*p*) & (¬□(¬*p*)) *in v*]
using *KBasic2-2*[*equiv-rl*] &*I* &*E* **by** *meson*
hence [¬(□(*p*) ∨ □(¬*p*)) *in v*]
by (*rule oth-class-taut-6-d*[*equiv-rl*])
thus [*Contingent p in v*]
unfolding *Contingent-def Necessary-defs Impossible-defs* .
qed

lemma *thm-cont-propos-3*[*PLM*]:
[*Contingent* (*p*::o) ≡ *Contingent* (p^-) *in v*]
using *thm-cont-propos-1*
unfolding *NonContingent-def Contingent-def*
by (*rule oth-class-taut-5-d*[*equiv-lr*])

context
begin
private definition 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* ∀*I*)
using *if-p-then-p* .

lemma *thm-noncont-propos-2*[*PLM*]:
[*Impossible* ($p_0{}^-$) *in v*]
unfolding *Impossible-defs*
apply (*PLM-subst-method* ¬p_0 $p_0{}^-$)
using *thm-relation-negation-3*[*equiv-sym*] **apply** *simp*
apply (*PLM-subst-method* p_0 ¬¬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* $\vee I(1)$)

lemma *thm-noncont-propos-4*[*PLM*]:
[*NonContingent* ($p_0{}^-$) *in v*]
unfolding *NonContingent-def* **using** *thm-noncont-propos-2*
by (*rule* $\vee I(2)$)

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 . $p \neq q$ & *Contingent p* & *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::Π_0) $\rightarrow$ ($\neg(\exists\ q\ .\ Contingent\ q$ & $q = p)$) *in v*]
proof (*rule CP*)
assume [*NonContingent p in v*]
hence [$\neg$(*Contingent p*) *in v*]
unfolding *NonContingent-def Contingent-def*
apply − **by** *PLM-solver*
moreover {
assume [$\exists\ q\ .\ Contingent\ q$ & $q = p$ *in v*]
then obtain r **where** [*Contingent r* & $r = p$ *in v*]
by (*rule* $\exists E$)

```
      hence [Contingent p in v]
        using &E l-identity[axiom-instance, deduction, deduction]
        by blast
    }
    ultimately show [¬(∃ q. Contingent q & q = p) in v]
      using modus-tollens-1 CP by blast
  qed

lemma four-distinct-props-2[PLM]:
  [Contingent (p::o) → ¬(∃ q . (NonContingent q & q = p)) in v]
  proof (rule CP)
    assume [Contingent p in v]
    hence [¬(NonContingent p) in v]
      unfolding NonContingent-def Contingent-def
      apply − by PLM-solver
    moreover {
      assume [∃ q . NonContingent q & q = p in v]
      then obtain r where [NonContingent r & r = p in v]
       by (rule ∃E)
      hence [NonContingent p in v]
        using &E l-identity[axiom-instance, deduction, deduction]
        by blast
    }
    ultimately show [¬(∃ q. NonContingent q & q = p) in v]
      using modus-tollens-1 CP by blast
  qed

lemma four-distinct-props-4[PLM]:
  [p₀ ≠ (p₀⁻) & p₀ ≠ q₀ & p₀ ≠ (q₀⁻) & (p₀⁻) ≠ q₀
    & (p₀⁻) ≠ (q₀⁻) & q₀ ≠ (q₀⁻) in v]
  proof (rule &I)+
    show [p₀ ≠ (p₀⁻) in v]
      by (rule thm-relation-negation-6)
    next
      {
        assume [p₀ = q₀ in v]
        hence [∃ q . NonContingent q & q = q₀ in v]
          using &I thm-noncont-propos-3 ∃I[where α=p₀]
          by simp
      }
      thus [p₀ ≠ q₀ in v]
        using four-distinct-props-2[deduction, OF basic-prop-2]
              modus-tollens-1 CP
        by blast
    next
      {
        assume [p₀ = (q₀⁻) in v]
        hence [∃ q . NonContingent q & q = (q₀⁻) in v]
          using thm-noncont-propos-3 &I ∃I[where α=p₀] by simp
      }
      thus [p₀ ≠ (q₀⁻) in v]
        using four-distinct-props-2[deduction, OF basic-prop-3]
              modus-tollens-1 CP
      by blast
    next
      {
        assume [(p₀⁻) = q₀ in v]
        hence [∃ q . NonContingent q & q = q₀ in v]
```

```
      using thm-noncont-propos-4 &I ∃ I[where α=p₀⁻] by auto
    }
    thus [(p₀⁻) ≠ q₀ in v]
      using four-distinct-props-2[deduction, OF basic-prop-2]
          modus-tollens-1 CP
      by blast
  next
    {
      assume [(p₀⁻) = (q₀⁻) in v]
      hence [∃ q . NonContingent q & q = (q₀⁻) in v]
        using thm-noncont-propos-4 &I ∃ I[where α=p₀⁻] by auto
    }
    thus [(p₀⁻) ≠ (q₀⁻) in v]
      using four-distinct-props-2[deduction, OF basic-prop-3]
          modus-tollens-1 CP
      by blast
  next
    show [q₀ ≠ (q₀⁻) in v]
      by (rule thm-relation-negation-6)
  qed

lemma cont-true-cont-1[PLM]:
  [ContingentlyTrue p → Contingent p in v]
  apply (rule CP, rule thm-cont-propos-2[equiv-rl])
  unfolding ContingentlyTrue-def
  apply (rule &I, drule &E(1))
   using T◊[deduction] apply simp
  by (rule &E(2))

lemma cont-true-cont-2[PLM]:
  [ContingentlyFalse p → Contingent p in v]
  apply (rule CP, rule thm-cont-propos-2[equiv-rl])
  unfolding ContingentlyFalse-def
  apply (rule &I, drule &E(2))
   apply simp
  apply (drule &E(1))
  using T◊[deduction] by simp

lemma cont-true-cont-3[PLM]:
  [ContingentlyTrue p ≡ ContingentlyFalse (p⁻) in v]
  unfolding ContingentlyTrue-def ContingentlyFalse-def
  apply (PLM-subst-method ¬p p⁻)
   using thm-relation-negation-3[equiv-sym] apply simp
  apply (PLM-subst-method p ¬¬p)
  by PLM-solver+

lemma cont-true-cont-4[PLM]:
  [ContingentlyFalse p ≡ ContingentlyTrue (p⁻) in v]
  unfolding ContingentlyTrue-def ContingentlyFalse-def
  apply (PLM-subst-method ¬p p⁻)
   using thm-relation-negation-3[equiv-sym] apply simp
  apply (PLM-subst-method p ¬¬p)
  by PLM-solver+

lemma cont-tf-thm-1[PLM]:
  [ContingentlyTrue q₀ ∨ ContingentlyFalse q₀ in v]
  proof −
    have [q₀ ∨ ¬q₀ in v]
```

```
    by PLM-solver
  moreover {
    assume [q0 in v]
    hence [q0 & ◇¬q0 in v]
      unfolding q0-def
      using qml-4[axiom-instance,conj2] &I
      by auto
  }
  moreover {
    assume [¬q0 in v]
    hence [(¬q0) & ◇q0 in v]
      unfolding q0-def
      using qml-4[axiom-instance,conj1] &I
      by auto
  }
  ultimately show ?thesis
    unfolding ContingentlyTrue-def ContingentlyFalse-def
    using ∨E(4) CP by auto
qed

lemma cont-tf-thm-2[PLM]:
[ContingentlyFalse q0 ∨ ContingentlyFalse (q0⁻) in v]
using cont-tf-thm-1 cont-true-cont-3[where p=q0]
      cont-true-cont-4[where p=q0]
apply − by PLM-solver

lemma cont-tf-thm-3[PLM]:
[∃ p . ContingentlyTrue p in v]
proof (rule ∨E(1); (rule CP)?)
  show [ContingentlyTrue q0 ∨ ContingentlyFalse q0 in v]
    using cont-tf-thm-1 .
next
  assume [ContingentlyTrue q0 in v]
  thus ?thesis
    using ∃I by metis
next
  assume [ContingentlyFalse q0 in v]
  hence [ContingentlyTrue (q0⁻) in v]
    using cont-true-cont-4[equiv-lr] by simp
  thus ?thesis
    using ∃I by metis
qed

lemma cont-tf-thm-4[PLM]:
[∃ p . ContingentlyFalse p in v]
proof (rule ∨E(1); (rule CP)?)
  show [ContingentlyTrue q0 ∨ ContingentlyFalse q0 in v]
    using cont-tf-thm-1 .
next
  assume [ContingentlyTrue q0 in v]
  hence [ContingentlyFalse (q0⁻) in v]
    using cont-true-cont-3[equiv-lr] by simp
  thus ?thesis
    using ∃I by metis
next
  assume [ContingentlyFalse q0 in v]
  thus ?thesis
    using ∃I by metis
```

```
    qed

  lemma cont-tf-thm-5[PLM]:
    [ContingentlyTrue p & Necessary q → p ≠ q in v]
    proof (rule CP)
      assume [ContingentlyTrue p & Necessary q in v]
      hence 1: [◇(¬p) & □ q in v]
        unfolding ContingentlyTrue-def Necessary-defs
        using &E &I by blast
      hence [¬□p in v]
        apply – apply (drule &E(1))
        unfolding diamond-def
        apply (PLM-subst-method ¬¬p p)
        using oth-class-taut-4-b[equiv-sym] by auto
      moreover {
        assume [p = q in v]
        hence [□p in v]
          using l-identity[where α=q and β=p and φ=λ x . □ x,
                           axiom-instance, deduction, deduction]
                1[conj2] id-eq-prop-prop-8-b[deduction]
          by blast
      }
      ultimately show [p ≠ q in v]
        using modus-tollens-1 CP by blast
    qed

  lemma cont-tf-thm-6[PLM]:
    [(ContingentlyFalse p & Impossible q) → p ≠ q in v]
    proof (rule CP)
      assume [ContingentlyFalse p & Impossible q in v]
      hence 1: [◇p & □(¬q) in v]
        unfolding ContingentlyFalse-def Impossible-defs
        using &E &I by blast
      hence [¬◇q in v]
        unfolding diamond-def apply – by PLM-solver
      moreover {
        assume [p = q in v]
        hence [◇q in v]
          using l-identity[axiom-instance, deduction, deduction] 1[conj1]
                id-eq-prop-prop-8-b[deduction]
          by blast
      }
      ultimately show [p ≠ q in v]
        using modus-tollens-1 CP by blast
    qed
end

lemma oa-contingent-1[PLM]:
  [O! ≠ A! in v]
  proof –
    {
      assume [O! = A! in v]
      hence [(λx. ◇(|E!,x^P|)) = (λx. ¬◇(|E!,x^P|)) in v]
        unfolding Ordinary-def Abstract-def .
      moreover have [(|(λx. ◇(|E!,x^P|)), x^P|) ≡ ◇(|E!,x^P|) in v]
        apply (rule beta-C-meta-1)
        by show-proper
      ultimately have [(|(λx. ¬◇(|E!,x^P|)), x^P|) ≡ ◇(|E!,x^P|) in v]
```

```
    using l-identity[axiom-instance, deduction, deduction] by fast
   moreover have [(|(λx. ¬◇(|E!,x^P|)), x^P|) ≡ ¬◇(|E!,x^P|) in v]
    apply (rule beta-C-meta-1)
    by show-proper
   ultimately have [◇(|E!,x^P|) ≡ ¬◇(|E!,x^P|) in v]
    apply − by PLM-solver
  }
  thus ?thesis
   using oth-class-taut-1-b modus-tollens-1 CP
   by blast
 qed

lemma oa-contingent-2[PLM]:
 [(|O!,x^P|) ≡ ¬(|A!,x^P|) in v]
 proof −
   have [(|(λx. ¬◇(|E!,x^P|)), x^P|) ≡ ¬◇(|E!,x^P|) in v]
    apply (rule beta-C-meta-1)
    by show-proper
   hence [(¬(|(λx. ¬◇(|E!,x^P|)), x^P|)) ≡ ◇(|E!,x^P|) in v]
    using oth-class-taut-5-d[equiv-lr] oth-class-taut-4-b[equiv-sym]
        ≡E(5) by blast
   moreover have [(|(λx. ◇(|E!,x^P|)), x^P|) ≡ ◇(|E!,x^P|) in v]
    apply (rule beta-C-meta-1)
    by show-proper
   ultimately show ?thesis
    unfolding Ordinary-def Abstract-def
    apply − by PLM-solver
 qed

lemma oa-contingent-3[PLM]:
 [(|A!,x^P|) ≡ ¬(|O!,x^P|) in v]
 using oa-contingent-2
 apply − by PLM-solver

lemma oa-contingent-4[PLM]:
 [Contingent O! in v]
 apply (rule thm-cont-prop-2[equiv-rl], rule &I)
 subgoal
  unfolding Ordinary-def
  apply (PLM-subst-method λ x . ◇(|E!,x^P|) λ x . (|λx. ◇(|E!,x^P|),x^P|))
   apply (safe intro!: beta-C-meta-1[equiv-sym])
    apply show-proper
  using BF◇[deduction, OF thm-cont-prop-2[equiv-lr, OF thm-cont-e-2, conj1]]
  by (rule T◇[deduction])
 subgoal
  apply (PLM-subst-method λ x . (|A!,x^P|) λ x . ¬(|O!,x^P|))
   using oa-contingent-3 apply simp
  using cqt-further-5[deduction,conj1, OF A-objects[axiom-instance]]
  by (rule T◇[deduction])
 done

lemma oa-contingent-5[PLM]:
 [Contingent A! in v]
 apply (rule thm-cont-prop-2[equiv-rl], rule &I)
 subgoal
  using cqt-further-5[deduction,conj1, OF A-objects[axiom-instance]]
  by (rule T◇[deduction])
 subgoal
```

```
    unfolding Abstract-def
    apply (PLM-subst-method λ x . ¬◇(|E!,x^P|) λ x . (|λx. ¬◇(|E!,x^P|),x^P|))
     apply (safe intro!: beta-C-meta-1[equiv-sym])
      apply show-proper
    apply (PLM-subst-method λ x . ◇(|E!,x^P|) λ x . ¬¬◇(|E!,x^P|))
     using oth-class-taut-4-b apply simp
    using BF◇[deduction, OF thm-cont-prop-2[equiv-lr, OF thm-cont-e-2, conj1]]
    by (rule T◇[deduction])
  done

lemma oa-contingent-6[PLM]:
  [(O!^-) ≠ (A!^-) in v]
  proof −
    {
      assume [(O!^-) = (A!^-) in v]
      hence [(λx. ¬(|O!,x^P|)) = (λx. ¬(|A!,x^P|)) in v]
        unfolding propnot-defs .
      moreover have [(|(λx. ¬(|O!,x^P|)), x^P|) ≡ ¬(|O!,x^P|) in v]
        apply (rule beta-C-meta-1)
        by show-proper
      ultimately have [(|λx. ¬(|A!,x^P|),x^P|) ≡  ¬(|O!,x^P|) in v]
        using l-identity[axiom-instance, deduction, deduction]
        by fast
      hence [(¬(|A!,x^P|)) ≡ ¬(|O!,x^P|) in v]
        apply −
        apply (PLM-subst-method (|λx. ¬(|A!,x^P|),x^P|) (¬(|A!,x^P|)))
         apply (safe intro!: beta-C-meta-1)
        by show-proper
      hence [(|O!,x^P|) ≡ ¬(|O!,x^P|) in v]
        using oa-contingent-2 apply − by PLM-solver
    }
    thus ?thesis
      using oth-class-taut-1-b modus-tollens-1 CP
      by blast
  qed

lemma oa-contingent-7[PLM]:
  [(|O!^-,x^P|) ≡ ¬(|A!^-,x^P|) in v]
  proof −
    have [(¬(|λx. ¬(|A!,x^P|),x^P|)) ≡ (|A!,x^P|) in v]
      apply (PLM-subst-method (¬(|A!,x^P|)) (|λx. ¬(|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 [(|λx. ¬(|O!,x^P|),x^P|) ≡ ¬(|O!,x^P|) in v]
      apply (rule beta-C-meta-1)
      by show-proper
    ultimately show ?thesis
      unfolding propnot-defs
      using oa-contingent-3
      apply − by PLM-solver
  qed

lemma oa-contingent-8[PLM]:
  [Contingent (O!^-) in v]
  using oa-contingent-4 thm-cont-prop-3[equiv-lr] by auto

lemma oa-contingent-9[PLM]:
```

[*Contingent* ($A!^{-}$) *in v*]
using *oa-contingent-5 thm-cont-prop-3*[*equiv-lr*] **by** *auto*

lemma *oa-facts-1*[*PLM*]:
[$(\!|O!,x^P|\!) \rightarrow \Box(\!|O!,x^P|\!)$ *in v*]
proof (*rule CP*)
assume [$(\!|O!,x^P|\!)$ *in v*]
hence [$\Diamond(\!|E!,x^P|\!)$ *in v*]
unfolding *Ordinary-def* **apply** −
apply (*rule beta-C-meta-1*[*equiv-lr*])
by *show-proper*
hence [$\Box\Diamond(\!|E!,x^P|\!)$ *in v*]
using *qml-3*[*axiom-instance*, *deduction*] **by** *auto*
thus [$\Box(\!|O!,x^P|\!)$ *in v*]
unfolding *Ordinary-def*
apply −
apply (*PLM-subst-method* $\Diamond(\!|E!,x^P|\!)$ $(\!|\boldsymbol{\lambda}x.\ \Diamond(\!|E!,x^P|\!),x^P|\!)$)
apply (*safe intro*!: *beta-C-meta-1*[*equiv-sym*])
by *show-proper*
qed

lemma *oa-facts-2*[*PLM*]:
[$(\!|A!,x^P|\!) \rightarrow \Box(\!|A!,x^P|\!)$ *in v*]
proof (*rule CP*)
assume [$(\!|A!,x^P|\!)$ *in v*]
hence [$\neg\Diamond(\!|E!,x^P|\!)$ *in v*]
unfolding *Abstract-def* **apply** −
apply (*rule beta-C-meta-1*[*equiv-lr*])
by *show-proper*
hence [$\Box\Box\neg(\!|E!,x^P|\!)$ *in v*]
using *KBasic2-4*[*equiv-rl*] *4*$\Box$[*deduction*] **by** *auto*
hence [$\Box\neg\Diamond(\!|E!,x^P|\!)$ *in v*]
apply −
apply (*PLM-subst-method* $\Box\neg(\!|E!,x^P|\!)$ $\neg\Diamond(\!|E!,x^P|\!)$)
using *KBasic2-4* **by** *auto*
thus [$\Box(\!|A!,x^P|\!)$ *in v*]
unfolding *Abstract-def*
apply −
apply (*PLM-subst-method* $\neg\Diamond(\!|E!,x^P|\!)$ $(\!|\boldsymbol{\lambda}x.\ \neg\Diamond(\!|E!,x^P|\!),x^P|\!)$)
apply (*safe intro*!: *beta-C-meta-1*[*equiv-sym*])
by *show-proper*
qed

lemma *oa-facts-3*[*PLM*]:
[$\Diamond(\!|O!,x^P|\!) \rightarrow (\!|O!,x^P|\!)$ *in v*]
using *oa-facts-1* **by** (*rule derived-S5-rules-2-b*)

lemma *oa-facts-4*[*PLM*]:
[$\Diamond(\!|A!,x^P|\!) \rightarrow (\!|A!,x^P|\!)$ *in v*]
using *oa-facts-2* **by** (*rule derived-S5-rules-2-b*)

lemma *oa-facts-5*[*PLM*]:
[$\Diamond(\!|O!,x^P|\!) \equiv \Box(\!|O!,x^P|\!)$ *in v*]
using *oa-facts-1*[*deduction*, *OF oa-facts-3*[*deduction*]]
T$\Diamond$[*deduction*, *OF qml-2*[*axiom-instance*, *deduction*]]
$\equiv$*I CP* **by** *blast*

lemma *oa-facts-6*[*PLM*]:

```
  [◇(|A!,x^P|) ≡ □(|A!,x^P|) in v]
  using oa-facts-2[deduction, OF oa-facts-4[deduction]]
    T◇[deduction, OF qml-2[axiom-instance, deduction]]
    ≡I CP by blast

lemma oa-facts-7[PLM]:
  [(|O!,x^P|) ≡ 𝒜(|O!,x^P|) in v]
  apply (rule ≡I; rule CP)
   apply (rule nec-imp-act[deduction, OF oa-facts-1[deduction]]; assumption)
  proof −
    assume [𝒜(|O!,x^P|) in v]
    hence [𝒜(◇(|E!,x^P|)) in v]
      unfolding Ordinary-def apply −
      apply (PLM-subst-method (|λx. ◇(|E!,x^P|),x^P|) ◇(|E!,x^P|))
      apply (safe intro!: beta-C-meta-1)
      by show-proper
    hence [◇(|E!,x^P|) in v]
      using Act-Basic-6[equiv-rl] by auto
    thus [(|O!,x^P|) in v]
      unfolding Ordinary-def apply −
      apply (PLM-subst-method ◇(|E!,x^P|) (|λx. ◇(|E!,x^P|),x^P|))
       apply (safe intro!: beta-C-meta-1[equiv-sym])
      by show-proper
  qed

lemma oa-facts-8[PLM]:
  [(|A!,x^P|) ≡ 𝒜(|A!,x^P|) in v]
  apply (rule ≡I; rule CP)
   apply (rule nec-imp-act[deduction, OF oa-facts-2[deduction]]; assumption)
  proof −
    assume [𝒜(|A!,x^P|) in v]
    hence [𝒜(¬◇(|E!,x^P|)) in v]
      unfolding Abstract-def apply −
      apply (PLM-subst-method (|λx. ¬◇(|E!,x^P|),x^P|) ¬◇(|E!,x^P|))
      apply (safe intro!: beta-C-meta-1)
      by show-proper
    hence [𝒜(□¬(|E!,x^P|)) in v]
      apply −
      apply (PLM-subst-method (¬◇(|E!,x^P|)) (□¬(|E!,x^P|)))
      using KBasic2-4[equiv-sym] by auto
    hence [¬◇(|E!,x^P|) in v]
      using qml-act-2[axiom-instance, equiv-rl] KBasic2-4[equiv-lr] by auto
    thus [(|A!,x^P|) in v]
      unfolding Abstract-def apply −
      apply (PLM-subst-method ¬◇(|E!,x^P|) (|λx. ¬◇(|E!,x^P|),x^P|))
      apply (safe intro!: beta-C-meta-1[equiv-sym])
      by show-proper
  qed

lemma cont-nec-fact1-1[PLM]:
  [WeaklyContingent F ≡ WeaklyContingent (F^−) in v]
  proof (rule ≡I; rule CP)
    assume [WeaklyContingent F in v]
    hence wc-def: [Contingent F & (∀ x . (◇(|F,x^P|) → □(|F,x^P|))) in v]
      unfolding WeaklyContingent-def .
    have [Contingent (F^−) in v]
      using wc-def[conj1] by (rule thm-cont-prop-3[equiv-lr])
    moreover {
```

```
  {
    fix x
    assume [◇(|F⁻,x^P|) in v]
    hence [¬□(|F,x^P|) in v]
      unfolding diamond-def apply −
      apply (PLM-subst-method ¬(|F⁻,x^P|) (|F,x^P|))
       using thm-relation-negation-2-1 by auto
    moreover {
      assume [¬□(|F⁻,x^P|) in v]
      hence [¬□(|λx. ¬(|F,x^P|),x^P|) in v]
        unfolding propnot-defs .
      hence [◇(|F,x^P|) in v]
        unfolding diamond-def
        apply − apply (PLM-subst-method (|λx. ¬(|F,x^P|),x^P|) ¬(|F,x^P|))
        apply (safe intro!: beta-C-meta-1)
        by show-proper
      hence [□(|F,x^P|) in v]
        using wc-def[conj2] cqt-1[axiom-instance, deduction]
              modus-ponens by fast
    }
    ultimately have [□(|F⁻, x^P|) in v]
      using ¬¬E modus-tollens-1 CP by blast
  }
  hence [∀ x . ◇(|F⁻,x^P|) → □(|F⁻, x^P|) in v]
    using ∀I CP by fast
}
ultimately show [WeaklyContingent (F⁻) in v]
  unfolding WeaklyContingent-def by (rule &I)
next
assume [WeaklyContingent (F⁻) in v]
hence wc-def: [Contingent (F⁻) & (∀ x . (◇(|F⁻,x^P|) → □(|F⁻,x^P|))) in v]
  unfolding WeaklyContingent-def .
have [Contingent F in v]
  using wc-def[conj1] by (rule thm-cont-prop-3[equiv-rl])
moreover {
  {
    fix x
    assume [◇(|F,x^P|) in v]
    hence [¬□(|F⁻,x^P|) in v]
      unfolding diamond-def apply −
      apply (PLM-subst-method ¬(|F,x^P|) (|F⁻,x^P|))
      using thm-relation-negation-1-1[equiv-sym] by auto
    moreover {
      assume [¬□(|F,x^P|) in v]
      hence [◇(|F⁻,x^P|) in v]
        unfolding diamond-def
        apply − apply (PLM-subst-method (|F,x^P|) ¬(|F⁻,x^P|))
        using thm-relation-negation-2-1[equiv-sym] by auto
      hence [□(|F⁻,x^P|) in v]
        using wc-def[conj2] cqt-1[axiom-instance, deduction]
              modus-ponens by fast
    }
    ultimately have [□(|F, x^P|) in v]
      using ¬¬E modus-tollens-1 CP by blast
  }
  hence [∀ x . ◇(|F,x^P|) → □(|F, x^P|) in v]
    using ∀I CP by fast
}
```

```
    ultimately show [WeaklyContingent (F) in v]
      unfolding WeaklyContingent-def by (rule &I)
  qed

lemma cont-nec-fact1-2[PLM]:
  [(WeaklyContingent F & ¬(WeaklyContingent G)) → (F ≠ G) in v]
  using l-identity[axiom-instance,deduction,deduction] &E &I
        modus-tollens-1 CP by metis

lemma cont-nec-fact2-1[PLM]:
  [WeaklyContingent (O!) in v]
  unfolding WeaklyContingent-def
  apply (rule &I)
   using oa-contingent-4 apply simp
  using oa-facts-5 unfolding equiv-def
  using &E(1) ∀ I by fast

lemma cont-nec-fact2-2[PLM]:
  [WeaklyContingent (A!) in v]
  unfolding WeaklyContingent-def
  apply (rule &I)
   using oa-contingent-5 apply simp
  using oa-facts-6 unfolding equiv-def
  using &E(1) ∀ I by fast

lemma cont-nec-fact2-3[PLM]:
  [¬(WeaklyContingent (E!)) in v]
  proof (rule modus-tollens-1, rule CP)
    assume [WeaklyContingent E! in v]
    thus [∀ x . ◇(|E!,x^P|) → □(|E!,x^P|) in v]
    unfolding WeaklyContingent-def using &E(2) by fast
  next
    {
      assume 1: [∀ x . ◇(|E!,x^P|) → □(|E!,x^P|) in v]
      have [∃ x . ◇(|E!,x^P|) & ◇(¬(|E!,x^P|)) in v]
        using qml-4[axiom-instance,conj1, THEN BFs-3[deduction]] .
      then obtain x where [◇(|E!,x^P|) & ◇(¬(|E!,x^P|)) in v]
        by (rule ∃ E)
      hence [◇(|E!,x^P|) & ◇(¬(|E!,x^P|)) in v]
        using KBasic2-8[deduction] S5Basic-8[deduction]
              &I &E by blast
      hence [□(|E!,x^P|) & (¬□(|E!,x^P|)) in v]
        using 1[THEN ∀ E, deduction] &E &I
              KBasic2-2[equiv-rl] by blast
      hence [¬(∀ x . ◇(|E!,x^P|) → □(|E!,x^P|)) in v]
        using oth-class-taut-1-a modus-tollens-1 CP by blast
    }
    thus [¬(∀ x . ◇(|E!,x^P|) → □(|E!,x^P|)) in v]
      using reductio-aa-2 if-p-then-p CP by meson
  qed

lemma cont-nec-fact2-4[PLM]:
  [¬(WeaklyContingent (PLM.L)) in v]
  proof −
    {
      assume [WeaklyContingent PLM.L in v]
      hence [Contingent PLM.L in v]
        unfolding WeaklyContingent-def using &E(1) by blast
```

}
thus *?thesis*
using *thm-noncont-e-e-3*
unfolding *Contingent-def NonContingent-def*
using *modus-tollens-2 CP* **by** *blast*
qed

lemma *cont-nec-fact2-5*[*PLM*]:
[$O! \neq E!$ & $O! \neq (E!^{-})$ & $O! \neq PLM.L$ & $O! \neq (PLM.L^{-})$ *in v*]
proof ((*rule* &*I*)+)
show [$O! \neq E!$ *in v*]
using *cont-nec-fact2-1 cont-nec-fact2-3*
cont-nec-fact1-2[*deduction*] &*I* **by** *simp*
next
have [¬(*WeaklyContingent* ($E!^{-}$)) *in v*]
using *cont-nec-fact1-1*[*THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-lr*]
cont-nec-fact2-3 **by** *auto*
thus [$O! \neq (E!^{-})$ *in v*]
using *cont-nec-fact2-1 cont-nec-fact1-2*[*deduction*] &*I* **by** *simp*
next
show [$O! \neq PLM.L$ *in v*]
using *cont-nec-fact2-1 cont-nec-fact2-4*
cont-nec-fact1-2[*deduction*] &*I* **by** *simp*
next
have [¬(*WeaklyContingent* ($PLM.L^{-}$)) *in v*]
using *cont-nec-fact1-1*[*THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-lr*]
cont-nec-fact2-4 **by** *auto*
thus [$O! \neq (PLM.L^{-})$ *in v*]
using *cont-nec-fact2-1 cont-nec-fact1-2*[*deduction*] &*I* **by** *simp*
qed

lemma *cont-nec-fact2-6*[*PLM*]:
[$A! \neq E!$ & $A! \neq (E!^{-})$ & $A! \neq PLM.L$ & $A! \neq (PLM.L^{-})$ *in v*]
proof ((*rule* &*I*)+)
show [$A! \neq E!$ *in v*]
using *cont-nec-fact2-2 cont-nec-fact2-3*
cont-nec-fact1-2[*deduction*] &*I* **by** *simp*
next
have [¬(*WeaklyContingent* ($E!^{-}$)) *in v*]
using *cont-nec-fact1-1*[*THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-lr*]
cont-nec-fact2-3 **by** *auto*
thus [$A! \neq (E!^{-})$ *in v*]
using *cont-nec-fact2-2 cont-nec-fact1-2*[*deduction*] &*I* **by** *simp*
next
show [$A! \neq PLM.L$ *in v*]
using *cont-nec-fact2-2 cont-nec-fact2-4*
cont-nec-fact1-2[*deduction*] &*I* **by** *simp*
next
have [¬(*WeaklyContingent* ($PLM.L^{-}$)) *in v*]
using *cont-nec-fact1-1*[*THEN oth-class-taut-5-d*[*equiv-lr*],
equiv-lr] *cont-nec-fact2-4* **by** *auto*
thus [$A! \neq (PLM.L^{-})$ *in v*]
using *cont-nec-fact2-2 cont-nec-fact1-2*[*deduction*] &*I* **by** *simp*
qed

lemma *id-nec3-1*[*PLM*]:
[$((x^P) =_E (y^P)) \equiv (\Box((x^P) =_E (y^P)))$ *in v*]
proof (*rule* ≡*I*; *rule CP*)

assume $[(x^P) =_E (y^P)\ in\ v]$
hence $[(\!|O!,x^P|\!)\ in\ v] \wedge [(\!|O!,y^P|\!)\ in\ v] \wedge [\Box(\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ in\ v]$
using *eq-E-simple-1*[*equiv-lr*] **using** &*E* **by** *blast*
hence $[\Box(\!|O!,x^P|\!)\ in\ v] \wedge [\Box(\!|O!,y^P|\!)\ in\ v]$
$\wedge\ [\Box\Box(\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ in\ v]$
using *oa-facts-1*[*deduction*] *S5Basic-6*[*deduction*] **by** *blast*
hence $[\Box((\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)\ \&\ \Box(\forall\ F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!)))\ in\ v]$
using &*I KBasic-3*[*equiv-rl*] **by** *presburger*
thus $[\Box((x^P) =_E (y^P))\ in\ v]$
apply −
apply (*PLM-subst-method*
$(\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)\ \&\ \Box(\forall\ F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))$
$(x^P) =_E (y^P)$)
using *eq-E-simple-1*[*equiv-sym*] **by** *auto*
next
assume $[\Box((x^P) =_E (y^P))\ in\ v]$
thus $[((x^P) =_E (y^P))\ in\ v]$
using *qml-2*[*axiom-instance*,*deduction*] **by** *simp*
qed

lemma *id-nec3-2*[*PLM*]:
$[\Diamond((x^P) =_E (y^P)) \equiv ((x^P) =_E (y^P))\ in\ v]$
proof (*rule* ≡*I*; *rule CP*)
assume $[\Diamond((x^P) =_E (y^P))\ in\ v]$
thus $[(x^P) =_E (y^P)\ in\ v]$
using *derived-S5-rules-2-b*[*deduction*] *id-nec3-1*[*equiv-lr*]
CP modus-ponens **by** *blast*
next
assume $[(x^P) =_E (y^P)\ in\ v]$
thus $[\Diamond((x^P) =_E (y^P))\ in\ v]$
by (*rule TBasic*[*deduction*])
qed

lemma *thm-neg-eqE*[*PLM*]:
$[((x^P) \neq_E (y^P)) \equiv (\neg((x^P) =_E (y^P)))\ in\ v]$
proof −
have $[(x^P) \neq_E (y^P)\ in\ v] = [(\!|(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ (x^P) =_E (y^P)))^-,\ x^P,\ y^P|\!)\ in\ v]$
unfolding $not\text{-}identical_E\text{-}def$ **by** *simp*
also have ... $= [\neg(\!|(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ (x^P) =_E (y^P))),\ x^P,\ y^P|\!)\ in\ v]$
unfolding *propnot-defs*
apply (*safe intro*!: *beta-C-meta-2*[*equiv-lr*] *beta-C-meta-2*[*equiv-rl*])
by *show-proper*+
also have ... $= [\neg((x^P) =_E (y^P))\ in\ v]$
apply (*PLM-subst-method*
$(\!|(\boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ (x^P) =_E (y^P))),\ x^P,\ y^P|\!)$
$(x^P) =_E (y^P)$)
apply (*safe intro*!: *beta-C-meta-2*)
unfolding *identity-defs* **by** *show-proper*
finally show *?thesis*
using ≡*I CP* **by** *presburger*
qed

lemma *id-nec4-1*[*PLM*]:
$[((x^P) \neq_E (y^P)) \equiv \Box((x^P) \neq_E (y^P))\ in\ v]$
proof −
have $[(\neg((x^P) =_E (y^P))) \equiv \Box(\neg((x^P) =_E (y^P)))\ in\ v]$
using *id-nec3-2*[*equiv-sym*] *oth-class-taut-5-d*[*equiv-lr*]
KBasic2-4[*equiv-sym*] *intro-elim-6-e* **by** *fast*

```
    thus ?thesis
      apply -
      apply (PLM-subst-method (¬((x^P) =_E (y^P))) (x^P) ≠_E (y^P))
      using thm-neg-eqE[equiv-sym] by auto
  qed

lemma id-nec4-2[PLM]:
  [◇((x^P) ≠_E (y^P)) ≡ ((x^P) ≠_E (y^P)) in v]
  using ≡I id-nec4-1[equiv-lr] derived-S5-rules-2-b CP T◇ by simp

lemma id-act-1[PLM]:
  [((x^P) =_E (y^P)) ≡ (𝒜((x^P) =_E (y^P))) in v]
  proof (rule ≡I; rule CP)
    assume [(x^P) =_E (y^P) in v]
    hence [□((x^P) =_E (y^P)) in v]
      using id-nec3-1[equiv-lr] by auto
    thus [𝒜((x^P) =_E (y^P)) in v]
      using nec-imp-act[deduction] by fast
  next
    assume [𝒜((x^P) =_E (y^P)) in v]
    hence [𝒜(⦇O!,x^P⦈ & ⦇O!,y^P⦈ & □(∀ F . ⦇F,x^P⦈ ≡ ⦇F,y^P⦈)) in v]
      apply -
      apply (PLM-subst-method
            (x^P) =_E (y^P)
            (⦇O!,x^P⦈ & ⦇O!,y^P⦈ & □(∀ F . ⦇F,x^P⦈ ≡ ⦇F,y^P⦈))))
      using eq-E-simple-1 by auto
    hence [𝒜⦇O!,x^P⦈ & 𝒜⦇O!,y^P⦈ & 𝒜(□(∀ F . ⦇F,x^P⦈ ≡ ⦇F,y^P⦈)) in v]
      using Act-Basic-2[equiv-lr] &I &E by meson
    thus [(x^P) =_E (y^P) in v]
      apply - apply (rule eq-E-simple-1[equiv-rl])
      using oa-facts-7[equiv-rl] qml-act-2[axiom-instance, equiv-rl]
           &I &E by meson
  qed

lemma id-act-2[PLM]:
  [((x^P) ≠_E (y^P)) ≡ (𝒜((x^P) ≠_E (y^P))) in v]
  apply (PLM-subst-method (¬((x^P) =_E (y^P))) ((x^P) ≠_E (y^P)))
   using thm-neg-eqE[equiv-sym] apply simp
  using id-act-1 oth-class-taut-5-d[equiv-lr] thm-neg-eqE intro-elim-6-e
       logic-actual-nec-1[axiom-instance,equiv-sym] by meson

end

class id-act = id-eq +
  assumes id-act-prop: [𝒜(α = β) in v] ⟹ [(α = β) in v]

instantiation ν :: id-act
begin
  instance proof
    interpret PLM .
    fix x::ν and y::ν and v::i
    assume [𝒜(x = y) in v]
    hence [𝒜(((x^P) =_E (y^P)) ∨ (⦇A!,x^P⦈ & ⦇A!,y^P⦈
          & □(∀ F . ⦃x^P,F⦄ ≡ ⦃y^P,F⦄))) in v]
      unfolding identity-defs by auto
    hence [𝒜(((x^P) =_E (y^P))) ∨ 𝒜((⦇A!,x^P⦈ & ⦇A!,y^P⦈
          & □(∀ F . ⦃x^P,F⦄ ≡ ⦃y^P,F⦄))) in v]
      using Act-Basic-10[equiv-lr] by auto
```

```
    moreover {
      assume [𝒜(((x^P) =_E (y^P))) in v]
      hence [(x^P) = (y^P) in v]
       using id-act-1[equiv-rl] eq-E-simple-2[deduction] by auto
    }
    moreover {
      assume [𝒜((|A!,x^P|) & (|A!,y^P|) & □(∀ F . {|x^P,F|} ≡ {|y^P,F|})) in v]
      hence [𝒜(|A!,x^P|) & 𝒜(|A!,y^P|) & 𝒜(□(∀ F . {|x^P,F|} ≡ {|y^P,F|})) in v]
        using Act-Basic-2[equiv-lr] &I &E by meson
      hence [(|A!,x^P|) & (|A!,y^P|) & (□(∀ F . {|x^P,F|} ≡ {|y^P,F|})) in v]
        using oa-facts-8[equiv-rl] qml-act-2[axiom-instance,equiv-rl]
          &I &E by meson
      hence [(x^P) = (y^P) in v]
       unfolding identity-defs using ∨I by auto
    }
    ultimately have [(x^P) = (y^P) in v]
      using intro-elim-4-a CP by meson
    thus [x = y in v]
      unfolding identity-defs by auto
  qed
end

instantiation Π1 :: id-act
begin
  instance proof
    interpret PLM .
    fix F::Π1 and G::Π1 and v::i
    show [𝒜(F = G) in v] ⟹ [(F = G) in v]
      unfolding identity-defs
      using qml-act-2[axiom-instance,equiv-rl] by auto
  qed
end

instantiation o :: id-act
begin
  instance proof
    interpret PLM .
    fix p :: o and q :: o and v::i
    show [𝒜(p = q) in v] ⟹ [p = q in v]
      unfolding identity_o-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,x^P,y^P|)) = (λy. (|G,x^P,y^P|))
            & (λy. (|F,y^P,x^P|)) = (λy. (|G,y^P,x^P|))) in v]
        using a logic-actual-nec-3[axiom-instance, equiv-lr] cqt-basic-4[equiv-lr] ∀E
        unfolding identity_2-def by fast
      hence [((λy. (|F,x^P,y^P|)) = (λy. (|G,x^P,y^P|)))
             & ((λy. (|F,y^P,x^P|)) = (λy. (|G,y^P,x^P|))) in v]
        using &I &E id-act-prop Act-Basic-2[equiv-lr] by metis
```

```
    }
    thus [F = G in v] unfolding identity-defs by (rule ∀ 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,z^P,x^P,y^P|)) = (λz. (|G,z^P,x^P,y^P|))
                   & (λz. (|F,x^P,z^P,y^P|)) = (λz. (|G,x^P,z^P,y^P|))
                   & (λz. (|F,x^P,y^P,z^P|)) = (λz. (|G,x^P,y^P,z^P|))
    {
      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 ≡I CP id-nec[equiv-lr, THEN nec-imp-act[deduction]]
          id-act-prop by metis

  lemma id-act-4[PLM]:
    [((α::('a::id-act)) ≠ β) ≡ 𝒜(α ≠ β) 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) = (ιx . x = y) in v]
    using descriptions[axiom-instance,equiv-rl]
          id-act-3[equiv-sym] ∀I by fast
```

TODO A.2. *More discussion/thought about eta conversion and the strength of the axiom lambda-predicates-3-∗ which immediately implies the following very general lemmas.*

```
  lemma eta-conversion-lemma-1[PLM]:
    [(λ x . (|F,x^P|)) = F in v]
    using lambda-predicates-3-1[axiom-instance] .
```

lemma *eta-conversion-lemma-0*[*PLM*]:
[($\boldsymbol{\lambda}^0$ p) = p *in* v]
using *lambda-predicates-3-0*[*axiom-instance*] .

lemma *eta-conversion-lemma-2*[*PLM*]:
[($\boldsymbol{\lambda}^2$ (λ x y . (|F,x^P,y^P|))) = F *in* v]
using *lambda-predicates-3-2*[*axiom-instance*] .

lemma *eta-conversion-lemma-3*[*PLM*]:
[($\boldsymbol{\lambda}^3$ (λ x y z . (|F,x^P,y^P,z^P|))) = F *in* v]
using *lambda-predicates-3-3*[*axiom-instance*] .

lemma *lambda-p-q-p-eq-q*[*PLM*]:
[(($\boldsymbol{\lambda}^0$ p) = ($\boldsymbol{\lambda}^0$ q)) $\equiv$ (p = q) *in* v]
using *eta-conversion-lemma-0*
l-identity[*axiom-instance*, *deduction*, *deduction*]
eta-conversion-lemma-0[*eq-sym*] $\equiv$*I CP*
by *metis*

A.9.12. The Theory of Objects

lemma *partition-1*[*PLM*]:
[$\forall$ x . (|$O!$,x^P|) $\vee$ (|$A!$,x^P|) *in* v]
proof (*rule* $\forall$*I*)
fix x
have [$\Diamond$(|$E!$,x^P|) $\vee$ $\neg\Diamond$(|$E!$,x^P|) *in* v]
by *PLM-solver*
moreover have [$\Diamond$(|$E!$,x^P|) $\equiv$ (|$\boldsymbol{\lambda}$ y . $\Diamond$(|$E!$,y^P|), x^P|) *in* v]
apply (*rule beta-C-meta-1*[*equiv-sym*])
by *show-proper*
moreover have [($\neg\Diamond$(|$E!$,x^P|)) $\equiv$ (|$\boldsymbol{\lambda}$ y . $\neg\Diamond$(|$E!$,y^P|), x^P|) *in* v]
apply (*rule beta-C-meta-1*[*equiv-sym*])
by *show-proper*
ultimately show [(|$O!$, x^P|) $\vee$ (|$A!$, x^P|) *in* v]
unfolding *Ordinary-def Abstract-def* **by** *PLM-solver*
qed

lemma *partition-2*[*PLM*]:
[$\neg$($\exists$ x . (|$O!$,x^P|) & (|$A!$,x^P|)) *in* v]
proof −
{
assume [$\exists$ x . (|$O!$,x^P|) & (|$A!$,x^P|) *in* v]
then obtain b **where** [(|$O!$,b^P|) & (|$A!$,b^P|) *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*]:
[(|$O!$,x|) $\rightarrow$ (x =$_E$ x) *in* v]
proof (*rule CP*)
assume [(|$O!$,x|) *in* v]
moreover have [$\Box$($\forall$ F . (|F,x|) $\equiv$ (|F,x|)) *in* v]
by *PLM-solver*

ultimately show $[(x) =_E (x)\ in\ v]$
using &I *eq-E-simple-1*[*equiv-rl*] **by** *blast*
qed

lemma *ord-eq-Eequiv-2*[*PLM*]:
$[(x =_E y) \rightarrow (y =_E x)\ in\ v]$
proof (*rule CP*)
assume $[x =_E y\ in\ v]$
hence *1*: $[(|O!,x|) \& (|O!,y|) \& \Box(\forall\ F\ .\ (|F,x|) \equiv (|F,y|))\ in\ v]$
using *eq-E-simple-1*[*equiv-lr*] **by** *simp*
have $[\Box(\forall\ F\ .\ (|F,y|) \equiv (|F,x|))\ in\ v]$
apply (*PLM-subst-method*
$\lambda\ F\ .\ (|F,x|) \equiv (|F,y|)$
$\lambda\ F\ .\ (|F,y|) \equiv (|F,x|)$)
using *oth-class-taut-3-g 1*[*conj2*] **by** *auto*
thus $[y =_E x\ in\ v]$
using *eq-E-simple-1*[*equiv-rl*] *1*[*conj1*]
&E &I **by** *meson*
qed

lemma *ord-eq-Eequiv-3*[*PLM*]:
$[((x =_E y) \& (y =_E z)) \rightarrow (x =_E z)\ in\ v]$
proof (*rule CP*)
assume *a*: $[(x =_E y) \& (y =_E z)\ in\ v]$
have $[\Box((\forall\ F\ .\ (|F,x|) \equiv (|F,y|)) \& (\forall\ F\ .\ (|F,y|) \equiv (|F,z|)))\ in\ v]$
using *KBasic-3*[*equiv-rl*] *a*[*conj1*, *THEN eq-E-simple-1*[*equiv-lr*,*conj2*]]
a[*conj2*, *THEN eq-E-simple-1*[*equiv-lr*,*conj2*]] &I **by** *blast*
moreover {
{
fix *w*
have $[((\forall\ F\ .\ (|F,x|) \equiv (|F,y|)) \& (\forall\ F\ .\ (|F,y|) \equiv (|F,z|)))$
$\rightarrow (\forall\ F\ .\ (|F,x|) \equiv (|F,z|))\ in\ w]$
by *PLM-solver*
}
hence $[\Box(((\forall\ F\ .\ (|F,x|) \equiv (|F,y|)) \& (\forall\ F\ .\ (|F,y|) \equiv (|F,z|)))$
$\rightarrow (\forall\ F\ .\ (|F,x|) \equiv (|F,z|)))\ in\ v]$
by (*rule RN*)
}
ultimately have $[\Box(\forall\ F\ .\ (|F,x|) \equiv (|F,z|))\ in\ v]$
using *qml-1*[*axiom-instance*,*deduction*,*deduction*] **by** *blast*
thus $[x =_E z\ in\ v]$
using *a*[*conj1*, *THEN eq-E-simple-1*[*equiv-lr*,*conj1*,*conj1*]]
using *a*[*conj2*, *THEN eq-E-simple-1*[*equiv-lr*,*conj1*,*conj2*]]
eq-E-simple-1[*equiv-rl*] &I
by *presburger*
qed

lemma *ord-eq-E-eq*[*PLM*]:
$[((|O!,x^P|) \vee (|O!,y^P|)) \rightarrow ((x^P = y^P) \equiv (x^P =_E y^P))\ in\ v]$
proof (*rule CP*)
assume $[(|O!,x^P|) \vee (|O!,y^P|)\ in\ v]$
moreover {
assume $[(|O!,x^P|)\ in\ v]$
hence $[(x^P = y^P) \equiv (x^P =_E y^P)\ in\ v]$
using ≡*I CP l-identity*[*axiom-instance*, *deduction*, *deduction*]
ord-eq-Eequiv-1[*deduction*] *eq-E-simple-2*[*deduction*] **by** *metis*
}
moreover {

assume $[(\!|O!,y^P|\!)\ in\ v]$
hence $[(x^P = y^P) \equiv (x^P =_E y^P)\ in\ v]$
using $\equiv I$ *CP l-identity*[*axiom-instance*, *deduction*, *deduction*]
ord-eq-Eequiv-1[*deduction*] *eq-E-simple-2*[*deduction*] *id-eq-2*[*deduction*]
ord-eq-Eequiv-2[*deduction*] *identity-ν-def* **by** *metis*
}
ultimately show $[(x^P = y^P) \equiv (x^P =_E y^P)\ in\ v]$
using *intro-elim-4-a CP* **by** *blast*
qed

lemma *ord-eq-E*[*PLM*]:
$[((\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)) \rightarrow ((\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!)) \rightarrow x^P =_E y^P)\ in\ v]$
proof (*rule CP*; *rule CP*)
assume *ord-xy*: $[(\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)\ in\ v]$
assume $[\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!)\ in\ v]$
hence $[(\!|\boldsymbol{\lambda}\ z\ .\ z^P =_E x^P,\ x^P|\!) \equiv (\!|\boldsymbol{\lambda}\ z\ .\ z^P =_E x^P,\ y^P|\!)\ in\ v]$
by (*rule* $\forall E$)
moreover have $[(\!|\boldsymbol{\lambda}\ z\ .\ z^P =_E x^P,\ x^P|\!)\ in\ v]$
apply (*rule beta-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** *simp*
ultimately have $[(\!|\boldsymbol{\lambda}\ z\ .\ z^P =_E x^P,\ y^P|\!)\ in\ v]$
using $\equiv E$ **by** *blast*
hence $[y^P =_E x^P\ in\ v]$
unfolding *identity*$_E$*-infix-def*
apply (*safe intro*!:
beta-C-meta-1[**where** $\varphi = \lambda\ z\ .\ (\!|$*basic-identity*$_E$,$z$,$x^P|\!)$, *equiv-lr*])
by *show-proper*
thus $[x^P =_E y^P\ in\ v]$
by (*rule ord-eq-Eequiv-2*[*deduction*])
qed

lemma *ord-eq-E2*[*PLM*]:
$[((\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)) \rightarrow$
$((x^P \neq y^P) \equiv (\boldsymbol{\lambda} z\ .\ z^P =_E x^P) \neq (\boldsymbol{\lambda} z\ .\ z^P =_E y^P))\ in\ v]$
proof (*rule CP*; *rule* $\equiv I$; *rule CP*)
assume *ord-xy*: $[(\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)\ in\ v]$
assume $[x^P \neq y^P\ in\ v]$
hence $[\neg(x^P =_E y^P)\ in\ v]$
using *eq-E-simple-2 modus-tollens-1* **by** *fast*
moreover {
assume $[(\boldsymbol{\lambda} z\ .\ z^P =_E x^P) = (\boldsymbol{\lambda} z\ .\ z^P =_E y^P)\ in\ v]$
moreover have $[(\!|\boldsymbol{\lambda} z\ .\ z^P =_E x^P,\ x^P|\!)\ in\ v]$
apply (*rule beta-C-meta-1*[*equiv-rl*])
unfolding *identity*$_E$*-infix-def*
apply *show-proper*
using *ord-eq-Eequiv-1*[*deduction*] *ord-xy*[*conj1*]
unfolding *identity*$_E$*-infix-def* **by** *presburger*
ultimately have $[(\!|\boldsymbol{\lambda} z\ .\ z^P =_E y^P,\ x^P|\!)\ in\ v]$
using *l-identity*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
hence $[x^P =_E y^P\ in\ v]$
unfolding *identity*$_E$*-infix-def*
apply (*safe intro*!:
beta-C-meta-1[**where** $\varphi = \lambda\ z\ .\ (\!|$*basic-identity*$_E$,$z$,$y^P|\!)$, *equiv-lr*])
by *show-proper*
}

ultimately show $[(\boldsymbol{\lambda} z\ .\ z^P =_E x^P) \neq (\boldsymbol{\lambda} z\ .\ z^P =_E y^P)\ in\ v]$
using *modus-tollens-1 CP* **by** *blast*
next
assume *ord-xy*: $[(\!|O!,x^P|\!)\ \&\ (\!|O!,y^P|\!)\ in\ v]$
assume $[(\boldsymbol{\lambda} z\ .\ z^P =_E x^P) \neq (\boldsymbol{\lambda} z\ .\ z^P =_E y^P)\ in\ v]$
moreover {
assume $[x^P = y^P\ in\ v]$
hence $[(\boldsymbol{\lambda} z\ .\ z^P =_E x^P) = (\boldsymbol{\lambda} z\ .\ z^P =_E y^P)\ in\ v]$
using *id-eq-1 l-identity*[*axiom-instance*, *deduction*, *deduction*]
by *fast*
}
ultimately show $[x^P \neq y^P\ in\ v]$
using *modus-tollens-1 CP* **by** *blast*
qed

lemma *ab-obey-1*[*PLM*]:
$[((\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)) \rightarrow ((\forall\ F\ .\ \{\!|x^P, F|\!\} \equiv \{\!|y^P, F|\!\}) \rightarrow x^P = y^P)\ in\ v]$
proof(*rule CP*; *rule CP*)
assume *abs-xy*: $[(\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ in\ v]$
assume *enc-equiv*: $[\forall\ F\ .\ \{\!|x^P, F|\!\} \equiv \{\!|y^P, F|\!\}\ in\ v]$
{
fix P
have $[\{\!|x^P, P|\!\} \equiv \{\!|y^P, P|\!\}\ in\ v]$
using *enc-equiv* **by** (*rule* $\forall E$)
hence $[\Box(\{\!|x^P, P|\!\} \equiv \{\!|y^P, P|\!\})\ in\ v]$
using *en-eq-2 intro-elim-6-e intro-elim-6-f*
en-eq-5[*equiv-rl*] **by** *meson*
}
hence $[\Box(\forall\ F\ .\ \{\!|x^P, F|\!\} \equiv \{\!|y^P, F|\!\})\ in\ v]$
using *BF*[*deduction*] $\forall I$ **by** *fast*
thus $[x^P = y^P\ in\ v]$
unfolding *identity-defs*
using $\vee I(2)$ *abs-xy* $\&I$ **by** *presburger*
qed

lemma *ab-obey-2*[*PLM*]:
$[((\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)) \rightarrow ((\exists\ F\ .\ \{\!|x^P, F|\!\}\ \&\ \neg\{\!|y^P, F|\!\}) \rightarrow x^P \neq y^P)\ in\ v]$
proof(*rule CP*; *rule CP*)
assume *abs-xy*: $[(\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ in\ v]$
assume $[\exists\ F\ .\ \{\!|x^P, F|\!\}\ \&\ \neg\{\!|y^P, F|\!\}\ in\ v]$
then obtain P **where** *P-prop*:
$[\{\!|x^P, P|\!\}\ \&\ \neg\{\!|y^P, P|\!\}\ in\ v]$
by (*rule* $\exists E$)
{
assume $[x^P = y^P\ in\ v]$
hence $[\{\!|x^P, P|\!\} \equiv \{\!|y^P, P|\!\}\ in\ v]$
using *l-identity*[*axiom-instance*, *deduction*, *deduction*]
oth-class-taut-4-a **by** *fast*
hence $[\{\!|y^P, P|\!\}\ in\ v]$
using *P-prop*[*conj1*] **by** (*rule* $\equiv E$)
}
thus $[x^P \neq y^P\ in\ v]$
using *P-prop*[*conj2*] *modus-tollens-1 CP* **by** *blast*
qed

lemma *ordnecfail*[*PLM*]:
$[(\!|O!,x^P|\!) \rightarrow \Box(\neg(\exists\ F\ .\ \{\!|x^P, F|\!\}))\ in\ v]$
proof (*rule CP*)

```
    assume [(|O!,x^P|) in v]
    hence [□(|O!,x^P|) in v]
      using oa-facts-1[deduction] by simp
    moreover hence [□((|O!,x^P|) → (¬(∃ F . {|x^P, F|}))) in v]
      using nocoder[axiom-necessitation, axiom-instance] by simp
    ultimately show [□(¬(∃ F . {|x^P, F|})) in v]
      using qml-1[axiom-instance, deduction, deduction] by fast
  qed

lemma o-objects-exist-1[PLM]:
  [◇(∃ x . (|E!,x^P|)) in v]
  proof −
    have [◇(∃ x . (|E!,x^P|) & ◇(¬(|E!,x^P|))) in v]
      using qml-4[axiom-instance, conj1] .
    hence [◇((∃ x . (|E!,x^P|)) & (∃ x . ◇(¬(|E!,x^P|)))) in v]
      using sign-S5-thm-3[deduction] by fast
    hence [◇(∃ x . (|E!,x^P|)) & ◇(∃ x . ◇(¬(|E!,x^P|))) in v]
      using KBasic2-8[deduction] by blast
    thus ?thesis using &E by blast
  qed

lemma o-objects-exist-2[PLM]:
  [□(∃ x . (|O!,x^P|)) in v]
  apply (rule RN) unfolding Ordinary-def
  apply (PLM-subst-method λ x . ◇(|E!,x^P|) λ x . (|λy. ◇(|E!,y^P|), x^P|))
   apply (safe intro!: beta-C-meta-1[equiv-sym])
   apply show-proper
  using o-objects-exist-1 BF◇[deduction] by blast

lemma o-objects-exist-3[PLM]:
  [□(¬(∀ x . (|A!,x^P|))) in v]
  apply (PLM-subst-method (∃ x. ¬(|A!,x^P|)) ¬(∀ x. (|A!,x^P|)))
   using cqt-further-2[equiv-sym] apply fast
  apply (PLM-subst-method λ x . (|O!,x^P|) λ x . ¬(|A!,x^P|))
  using oa-contingent-2 o-objects-exist-2 by auto

lemma a-objects-exist-1[PLM]:
  [□(∃ x . (|A!,x^P|)) in v]
  proof −
    {
      fix v
      have [∃ x . (|A!,x^P|) & (∀ F . {|x^P, F|} ≡ (F = F)) in v]
        using A-objects[axiom-instance] by simp
      hence [∃ x . (|A!,x^P|) in v]
        using cqt-further-5[deduction,conj1] by fast
    }
    thus ?thesis by (rule RN)
  qed

lemma a-objects-exist-2[PLM]:
  [□(¬(∀ x . (|O!,x^P|))) in v]
  apply (PLM-subst-method (∃ x. ¬(|O!,x^P|)) ¬(∀ x. (|O!,x^P|)))
   using cqt-further-2[equiv-sym] apply fast
  apply (PLM-subst-method λ x . (|A!,x^P|) λ x . ¬(|O!,x^P|))
   using oa-contingent-3 a-objects-exist-1 by auto

lemma a-objects-exist-3[PLM]:
  [□(¬(∀ x . (|E!,x^P|))) in v]
```

```
  proof −
    {
      fix v
      have [∃ x . (|A!,x^P|) & (∀ F . {|x^P, F|} ≡ (F = F)) in v]
        using A-objects[axiom-instance] by simp
      hence [∃ x . (|A!,x^P|) in v]
        using cqt-further-5[deduction,conj1] by fast
      then obtain a where
        [(|A!,a^P|) in v]
        by (rule ∃E)
      hence [¬(◇(|E!,a^P|)) in v]
        unfolding Abstract-def
        apply (safe intro!: beta-C-meta-1[equiv-lr])
        by show-proper
      hence [(¬(|E!,a^P|)) in v]
        using KBasic2-4[equiv-rl] qml-2[axiom-instance,deduction]
        by simp
      hence [¬(∀ x . (|E!,x^P|)) in v]
        using ∃I cqt-further-2[equiv-rl]
        by fast
    }
    thus ?thesis
      by (rule RN)
  qed

lemma encoders-are-abstract[PLM]:
  [(∃ F . {|x^P, F|}) → (|A!,x^P|) in v]
  using nocoder[axiom-instance] contraposition-2
        oa-contingent-2[THEN oth-class-taut-5-d[equiv-lr], equiv-lr]
        useful-tautologies-1[deduction]
        vdash-properties-10 CP by metis

lemma A-objects-unique[PLM]:
  [∃! x . (|A!,x^P|) & (∀ F . {|x^P, F|} ≡ φ F) in v]
  proof −
    have [∃ x . (|A!,x^P|) & (∀ F . {|x^P, F|} ≡ φ F) in v]
      using A-objects[axiom-instance] by simp
    then obtain a where a-prop:
      [(|A!,a^P|) & (∀ F . {|a^P, F|} ≡ φ F) in v] by (rule ∃E)
    moreover have [∀ y . (|A!,y^P|) & (∀ F . {|y^P, F|} ≡ φ F) → (y = a) in v]
      proof (rule ∀I; rule CP)
        fix b
        assume b-prop: [(|A!,b^P|) & (∀ F . {|b^P, F|} ≡ φ F) in v]
        {
          fix P
          have [{|b^P,P|} ≡ {|a^P, P|} in v]
            using a-prop[conj2] b-prop[conj2] ≡I ≡E(1) ≡E(2)
                  CP vdash-properties-10 ∀E by metis
        }
        hence [∀ F . {|b^P,F|} ≡ {|a^P, F|} in v]
          using ∀I by fast
        thus [b = a in v]
          unfolding identity-ν-def
          using ab-obey-1[deduction, deduction]
                a-prop[conj1] b-prop[conj1] &I by blast
      qed
    ultimately show ?thesis
      unfolding exists-unique-def
```

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

lemma *obj-oth-1*[*PLM*]:
[$\exists!\ x\ .\ (\!|A!,x^P|\!)$ **&** $(\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv (\!|F,\ y^P|\!))$ *in* v]
using *A-objects-unique* **.**

lemma *obj-oth-2*[*PLM*]:
[$\exists!\ x\ .\ (\!|A!,x^P|\!)$ **&** $(\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv ((\!|F,\ y^P|\!)$ **&** $(\!|F,\ z^P|\!)))$ *in* v]
using *A-objects-unique* **.**

lemma *obj-oth-3*[*PLM*]:
[$\exists!\ x\ .\ (\!|A!,x^P|\!)$ **&** $(\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv ((\!|F,\ y^P|\!) \vee (\!|F,\ z^P|\!)))$ *in* v]
using *A-objects-unique* **.**

lemma *obj-oth-4*[*PLM*]:
[$\exists!\ x\ .\ (\!|A!,x^P|\!)$ **&** $(\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv (\Box(\!|F,\ y^P|\!)))$ *in* v]
using *A-objects-unique* **.**

lemma *obj-oth-5*[*PLM*]:
[$\exists!\ x\ .\ (\!|A!,x^P|\!)$ **&** $(\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv (F = G))$ *in* v]
using *A-objects-unique* **.**

lemma *obj-oth-6*[*PLM*]:
[$\exists!\ x\ .\ (\!|A!,x^P|\!)$ **&** $(\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv \Box(\forall\ y\ .\ (\!|G,\ y^P|\!) \rightarrow (\!|F,\ y^P|\!)))$ *in* v]
using *A-objects-unique* **.**

lemma *A-Exists-1*[*PLM*]:
[$\mathcal{A}(\exists!\ x :: ('a :: id\text{-}act)\ .\ \varphi\ x) \equiv (\exists!\ x\ .\ \mathcal{A}(\varphi\ x))$ *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 α **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 β
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 α **where** *1*:
[$\mathcal{A}\varphi\ \alpha$ **&** $(\forall\,\beta.\ \mathcal{A}\varphi\ \beta \rightarrow \beta = \alpha)$ *in* v]

```
    by (rule ∃ E)
  {
    fix β
    have [𝒜(φ β) → β = α in v]
      using 1[conj2] by (rule ∀ E)
    hence [𝒜(φ β → β = α) in v]
      using logic-actual-nec-2[axiom-instance, equiv-rl] id-act-3[equiv-lr]
            vdash-properties-10 CP by blast
  }
  hence [∀ β . 𝒜(φ β → β = α) in v]
    by (rule ∀ I)
  hence [𝒜(∀ β . φ β → β = α) in v]
    using logic-actual-nec-3[axiom-instance, equiv-rl] by fast
  hence [𝒜(φ α & (∀ β . φ β → β = α)) in v]
    using 1[conj1] Act-Basic-2[equiv-rl] &I by blast
  hence [∃ α. 𝒜(φ α & (∀ β. φ β → β = α)) in v]
    using ∃ I by fast
  thus [𝒜(∃ α. φ α & (∀ β. φ β → β = α)) in v]
    using Act-Basic-11[equiv-rl] by fast
qed

lemma A-Exists-2[PLM]:
[(∃ y . y^P = (ιx . φ x)) ≡ 𝒜(∃ !x . φ x) in v]
using actual-desc-1 A-Exists-1[equiv-sym]
      intro-elim-6-e by blast

lemma A-descriptions[PLM]:
[∃ y . y^P = (ιx . (|A!,x^P|) & (∀ F . {|x^P,F|} ≡ φ 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 = (ιx . (|A!,x^P|) & (∀ F . {|x^P,F|} ≡ φ F)))
  → ((|A!,y^P|) & (∀ F . {|y^P,F|} ≡ φ F)) in dw]
using y-in-2 by auto

lemma can-ab2[PLM]:
[(y^P = (ιx . (|A!,x^P|) & (∀ F . {|x^P,F|} ≡ φ F))) → (|A!,y^P|) in v]
proof (rule CP)
  assume [y^P = (ιx . (|A!,x^P|) & (∀ F . {|x^P,F|} ≡ φ F)) in v]
  hence [𝒜(|A!,y^P|) & 𝒜(∀ F . {|y^P,F|} ≡ φ F) in v]
    using nec-hintikka-scheme[equiv-lr, conj1]
          Act-Basic-2[equiv-lr] by blast
  thus [(|A!,y^P|) in v]
    using oa-facts-8[equiv-rl] &E by blast
qed

lemma desc-encode[PLM]:
[{|ιx . (|A!,x^P|) & (∀ F . {|x^P,F|} ≡ φ F), G|} ≡ φ G in dw]
proof −
  obtain a where
  [a^P = (ιx . (|A!,x^P|) & (∀ F . {|x^P,F|} ≡ φ F)) in dw]
  using A-descriptions by (rule ∃ E)
  moreover hence [{|a^P, G|} ≡ φ G in dw]
    using hintikka[equiv-lr, conj1] &E ∀ E by fast
  ultimately show ?thesis
    using l-identity[axiom-instance, deduction, deduction] by fast
qed
```

lemma *desc-nec-encode*[*PLM*]:
[$\{\!|\iota x\ .\ (\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F),\ G\}\!\rangle \equiv \boldsymbol{\mathcal{A}}(\varphi\ G)$ *in* v]
proof −
obtain a **where**
[$a^P = (\iota x\ .\ (\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ \{\!|x^P,F|\!\} \equiv \varphi\ F))$ *in* v]
using *A-descriptions* **by** (*rule* $\exists\, E$)
moreover {
hence [$\boldsymbol{\mathcal{A}}((\!|A!,a^P|\!)\ \&\ (\forall\ F\ .\ \{\!|a^P,F|\!\} \equiv \varphi\ F))$ *in* v]
using *nec-hintikka-scheme*[*equiv-lr*, *conj1*] **by** *fast*
hence [$\boldsymbol{\mathcal{A}}(\forall\ F\ .\ \{\!|a^P,F|\!\} \equiv \varphi\ F)$ *in* v]
using *Act-Basic-2*[*equiv-lr*,*conj2*] **by** *blast*
hence [$\forall\ F\ .\ \boldsymbol{\mathcal{A}}(\ \{\!|a^P,F|\!\} \equiv \varphi\ F)$ *in* v]
using *logic-actual-nec-3*[*axiom-instance*, *equiv-lr*] **by** *blast*
hence [$\boldsymbol{\mathcal{A}}(\{\!|a^P,\ G|\!\} \equiv \varphi\ G)$ *in* v]
using $\forall\, E$ **by** *fast*
hence [$\boldsymbol{\mathcal{A}}\{\!|a^P,\ G|\!\} \equiv \boldsymbol{\mathcal{A}}(\varphi\ G)$ *in* v]
using *Act-Basic-5*[*equiv-lr*] **by** *fast*
hence [$\{\!|a^P,\ G|\!\} \equiv \boldsymbol{\mathcal{A}}(\varphi\ G)$ *in* v]
using *en-eq-10*[*equiv-sym*] *intro-elim-6-e* **by** *blast*
}
ultimately show *?thesis*
using *l-identity*[*axiom-instance*, *deduction*, *deduction*] **by** *fast*
qed

notepad
begin
fix v
let $?x = \iota x\ .\ (\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv (\exists\ q\ .\ q\ \&\ F = (\boldsymbol{\lambda}\ y\ .\ q)))$
have [$\Box(\exists\ p\ .$ *ContingentlyTrue* $p)$ *in* v]
using *cont-tf-thm-3 RN* **by** *auto*
hence [$\boldsymbol{\mathcal{A}}(\exists\ p\ .$ *ContingentlyTrue* $p)$ *in* v]
using *nec-imp-act*[*deduction*] **by** *simp*
hence [$\exists\ p\ .\ \boldsymbol{\mathcal{A}}($*ContingentlyTrue* $p)$ *in* v]
using *Act-Basic-11*[*equiv-lr*] **by** *auto*
then obtain p_1 **where**
[$\boldsymbol{\mathcal{A}}($*ContingentlyTrue* $p_1)$ *in* v]
by (*rule* $\exists\, E$)
hence [$\boldsymbol{\mathcal{A}}p_1$ *in* v]
unfolding *ContingentlyTrue-def*
using *Act-Basic-2*[*equiv-lr*] $\&E$ **by** *fast*
hence [$\boldsymbol{\mathcal{A}}p_1\ \&\ \boldsymbol{\mathcal{A}}((\boldsymbol{\lambda}\ y\ .\ p_1) = (\boldsymbol{\lambda}\ y\ .\ p_1))$ *in* v]
using $\&I$ *id-eq-1*[*THEN RN*, *THEN nec-imp-act*[*deduction*]] **by** *fast*
hence [$\boldsymbol{\mathcal{A}}(p_1\ \&\ (\boldsymbol{\lambda}\ y\ .\ p_1) = (\boldsymbol{\lambda}\ y\ .\ p_1))$ *in* v]
using *Act-Basic-2*[*equiv-rl*] **by** *fast*
hence [$\exists\ q\ .\ \boldsymbol{\mathcal{A}}(\ q\ \&\ (\boldsymbol{\lambda}\ y\ .\ p_1) = (\boldsymbol{\lambda}\ y\ .\ q))$ *in* v]
using $\exists\, I$ **by** *fast*
hence [$\boldsymbol{\mathcal{A}}(\exists\ q\ .\ q\ \&\ (\boldsymbol{\lambda}\ y\ .\ p_1) = (\boldsymbol{\lambda}\ y\ .\ q))$ *in* v]
using *Act-Basic-11*[*equiv-rl*] **by** *fast*
moreover have [$\{\!|?x,\ \boldsymbol{\lambda}\ y\ .\ p_1|\!\} \equiv \boldsymbol{\mathcal{A}}(\exists\ q\ .\ q\ \&\ (\boldsymbol{\lambda}\ y\ .\ p_1) = (\boldsymbol{\lambda}\ y\ .\ q))$ *in* v]
using *desc-nec-encode* **by** *fast*
ultimately have [$\{\!|?x,\ \boldsymbol{\lambda}\ y\ .\ p_1|\!\}$ *in* v]
using $\equiv E$ **by** *blast*
end

lemma *Box-desc-encode-1*[*PLM*]:
[$\Box(\varphi\ G) \rightarrow \{\!|(\iota x\ .\ (\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ \{\!|x^P,\ F|\!\} \equiv \varphi\ F)),\ G|\!\}\rangle$ *in* v]
proof (*rule CP*)

```
    assume [□(φ G) in v]
    hence [𝒜(φ G) in v]
      using nec-imp-act[deduction] by auto
    thus [⦃ιx . (|A!,x^P|) & (∀ F . ⦃x^P,F⦄ ≡ φ F), G⦄ in v]
      using desc-nec-encode[equiv-rl] by simp
  qed

lemma Box-desc-encode-2[PLM]:
  [□(φ G) → □(⦃(ιx . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄ ≡ φ G) in v]
  proof (rule CP)
    assume a: [□(φ G) in v]
    hence [□(⦃(ιx . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄ → φ G) in v]
      using KBasic-1[deduction] by simp
    moreover {
      have [⦃(ιx . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄ in v]
        using a Box-desc-encode-1[deduction] by auto
      hence [□⦃(ιx . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄ in v]
        using encoding[axiom-instance,deduction] by blast
      hence [□(φ G → ⦃(ιx . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄) in v]
        using KBasic-1[deduction] by simp
    }
    ultimately show [□(⦃(ιx . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)), G⦄
                    ≡ φ G) in v]
      using &I KBasic-4[equiv-rl] by blast
  qed

lemma box-phi-a-1[PLM]:
  assumes [□(∀ F . φ F → □(φ F)) in v]
  shows [((|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)) → □((|A!,x^P|)
        & (∀ F . ⦃x^P, F⦄ ≡ φ F)) in v]
  proof (rule CP)
    assume a: [((|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)) in v]
    have [□(|A!,x^P|) in v]
      using oa-facts-2[deduction] a[conj1] by auto
    moreover have [□(∀ F . ⦃x^P, F⦄ ≡ φ F) in v]
      proof (rule BF[deduction]; rule ∀I)
        fix F
        have ϑ: [□(φ F → □(φ F)) in v]
          using assms[THEN CBF[deduction]] by (rule ∀E)
        moreover have [□(⦃x^P, F⦄ → □⦃x^P, F⦄) in v]
          using encoding[axiom-necessitation, axiom-instance] by simp
        moreover have [□⦃x^P, F⦄ ≡ □(φ F) in v]
          proof (rule ≡I; rule CP)
            assume [□⦃x^P, F⦄ in v]
            hence [⦃x^P, F⦄ in v]
              using qml-2[axiom-instance, deduction] by blast
            hence [φ F in v]
              using a[conj2] ∀E[where 'a=Π1] ≡E by blast
            thus [□(φ F) in v]
              using ϑ[THEN qml-2[axiom-instance, deduction], deduction] by simp
          next
            assume [□(φ F) in v]
            hence [φ F in v]
              using qml-2[axiom-instance, deduction] by blast
            hence [⦃x^P, F⦄ in v]
              using a[conj2] ∀E[where 'a=Π1] ≡E by blast
            thus [□⦃x^P, F⦄ in v]
              using encoding[axiom-instance, deduction] by simp
```

```
      qed
    ultimately show [□(⦃x^P,F⦄ ≡ φ F) in v]
      using sc-eq-box-box-3[deduction, deduction] &I by blast
  qed
 ultimately show [□((|A!,x^P|) & (∀ F. ⦃x^P,F⦄ ≡ φ F)) in v]
  using &I KBasic-3[equiv-rl] by blast
qed

lemma box-phi-a-2[PLM]:
 assumes [□(∀ F . φ F → □(φ F)) in v]
 shows [y^P = (ιx . (|A!,x^P|) & (∀ F. ⦃x^P, F⦄ ≡ φ F))
       → ((|A!,y^P|) & (∀ F . ⦃y^P, F⦄ ≡ φ F)) in v]
 proof −
  let ?ψ = λ x . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)
  have [∀ x . ?ψ x → □(?ψ x) in v]
   using box-phi-a-1[OF assms] ∀I by fast
  hence [(∃! x . ?ψ x) → (∀ y . y^P = (ιx . ?ψ x) → ?ψ y) in v]
   using unique-box-desc[deduction] by fast
  hence [(∀ y . y^P = (ιx . ?ψ x) → ?ψ y) in v]
   using A-objects-unique modus-ponens by blast
  thus ?thesis by (rule ∀E)
qed

lemma box-phi-a-3[PLM]:
 assumes [□(∀ F . φ F → □(φ F)) in v]
 shows [⦃ιx . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F), G⦄ ≡ φ G in v]
 proof −
  obtain a where
   [a^P = (ιx . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ φ F)) in v]
   using A-descriptions by (rule ∃E)
  moreover {
   hence [(∀ F . ⦃a^P, F⦄ ≡ φ F) in v]
    using box-phi-a-2[OF assms, deduction, conj2] by blast
   hence [⦃a^P, G⦄ ≡ φ G in v] by (rule ∀E)
  }
  ultimately show ?thesis
   using l-identity[axiom-instance, deduction, deduction] by fast
 qed

lemma null-uni-uniq-1[PLM]:
 [∃! x . Null (x^P) in v]
 proof −
  have [∃ x . (|A!,x^P|) & (∀ F . ⦃x^P, F⦄ ≡ (F ≠ F)) in v]
   using A-objects[axiom-instance] by simp
  then obtain a where a-prop:
   [(|A!,a^P|) & (∀ F . ⦃a^P, F⦄ ≡ (F ≠ F)) in v]
   by (rule ∃E)
  have 1: [(|A!,a^P|) & (¬(∃ F . ⦃a^P, F⦄)) in v]
   using a-prop[conj1] apply (rule &I)
   proof −
    {
     assume [∃ F . ⦃a^P, F⦄ in v]
     then obtain P where
      [⦃a^P, P⦄ in v] by (rule ∃E)
     hence [P ≠ P in v]
      using a-prop[conj2, THEN ∀E, equiv-lr] by simp
     hence [¬(∃ F . ⦃a^P, F⦄) in v]
      using id-eq-1 reductio-aa-1 by fast
```

```
      }
      thus [¬(∃ F . {|a^P, F|}) in v]
        using reductio-aa-1 by blast
    qed
  moreover have [∀ y . ((|A!,y^P|) & (¬(∃ F . {|y^P, F|}))) → y = a in v]
    proof (rule ∀I; rule CP)
      fix y
      assume 2: [(|A!,y^P|) & (¬(∃ F . {|y^P, F|})) in v]
      have [∀ F . {|y^P, F|} ≡ {|a^P, F|} in v]
        using cqt-further-12[deduction] 1[conj2] 2[conj2] &I by blast
      thus [y = a in v]
        using ab-obey-1[deduction, deduction]
        &I[OF 2[conj1] 1[conj1]] identity-ν-def by presburger
    qed
  ultimately show ?thesis
    using &I ∃I
    unfolding Null-def exists-unique-def by fast
qed

lemma null-uni-uniq-2[PLM]:
[∃! x . Universal (x^P) in v]
proof −
  have [∃ x . (|A!,x^P|) & (∀ F . {|x^P, F|} ≡ (F = F)) in v]
    using A-objects[axiom-instance] by simp
  then obtain a where a-prop:
    [(|A!,a^P|) & (∀ F . {|a^P, F|} ≡ (F = F)) in v]
    by (rule ∃E)
  have 1: [(|A!,a^P|) & (∀ F . {|a^P, F|}) in v]
    using a-prop[conj1] apply (rule &I)
    using ∀I a-prop[conj2, THEN ∀E, equiv-rl] id-eq-1 by fast
  moreover have [∀ y . ((|A!,y^P|) & (∀ F . {|y^P, F|})) → y = a in v]
    proof (rule ∀I; rule CP)
      fix y
      assume 2: [(|A!,y^P|) & (∀ F . {|y^P, F|}) in v]
      have [∀ F . {|y^P, F|} ≡ {|a^P, F|} in v]
        using cqt-further-11[deduction] 1[conj2] 2[conj2] &I by blast
      thus [y = a in v]
        using ab-obey-1[deduction, deduction]
          &I[OF 2[conj1] 1[conj1]] identity-ν-def
        by presburger
    qed
  ultimately show ?thesis
    using &I ∃I
    unfolding Universal-def exists-unique-def by fast
qed

lemma null-uni-uniq-3[PLM]:
[∃ y . y^P = (ιx . Null (x^P)) in v]
using null-uni-uniq-1[THEN RN, THEN nec-imp-act[deduction]]
      A-Exists-2[equiv-rl] by auto

lemma null-uni-uniq-4[PLM]:
[∃ y . y^P = (ιx . Universal (x^P)) in v]
using null-uni-uniq-2[THEN RN, THEN nec-imp-act[deduction]]
      A-Exists-2[equiv-rl] by auto

lemma null-uni-facts-1[PLM]:
[Null (x^P) → □(Null (x^P)) in v]
```

proof (*rule CP*)
assume [*Null* (x^P) *in v*]
hence *1*: [(|*A*!,x^P|) **&** (¬(∃ *F* . ⦃x^P,*F*⦄)) *in v*]
unfolding *Null-def* **.**
have [□(|*A*!,x^P|) *in v*]
using *1*[*conj1*] *oa-facts-2*[*deduction*] **by** *simp*
moreover have [□(¬(∃ *F* . ⦃x^P,*F*⦄)) *in v*]
proof −
{
assume [¬□(¬(∃ *F* . ⦃x^P,*F*⦄)) *in v*]
hence [◇(∃ *F* . ⦃x^P,*F*⦄) *in v*]
unfolding *diamond-def* **.**
hence [∃ *F* . ◇⦃x^P,*F*⦄ *in v*]
using *BF*◇[*deduction*] **by** *blast*
then obtain *P* **where** [◇⦃x^P,*P*⦄ *in v*]
by (*rule* ∃ *E*)
hence [⦃x^P, *P*⦄ *in v*]
using *en-eq-3*[*equiv-lr*] **by** *simp*
hence [∃ *F* . ⦃x^P, *F*⦄ *in v*]
using ∃ *I* **by** *fast*
}
thus *?thesis*
using *1*[*conj2*] *modus-tollens-1 CP*
useful-tautologies-1[*deduction*] **by** *metis*
qed
ultimately show [□*Null* (x^P) *in v*]
unfolding *Null-def*
using &*I KBasic-3*[*equiv-rl*] **by** *blast*
qed

lemma *null-uni-facts-2*[*PLM*]:
[*Universal* (x^P) → □(*Universal* (x^P)) *in v*]
proof (*rule CP*)
assume [*Universal* (x^P) *in v*]
hence *1*: [(|*A*!,x^P|) **&** (∀ *F* . ⦃x^P,*F*⦄) *in v*]
unfolding *Universal-def* **.**
have [□(|*A*!,x^P|) *in v*]
using *1*[*conj1*] *oa-facts-2*[*deduction*] **by** *simp*
moreover have [□(∀ *F* . ⦃x^P,*F*⦄) *in v*]
proof (*rule BF*[*deduction*]; *rule* ∀ *I*)
fix *F*
have [⦃x^P, *F*⦄ *in v*]
using *1*[*conj2*] **by** (*rule* ∀ *E*)
thus [□⦃x^P, *F*⦄ *in v*]
using *encoding*[*axiom-instance*, *deduction*] **by** *auto*
qed
ultimately show [□*Universal* (x^P) *in v*]
unfolding *Universal-def*
using &*I KBasic-3*[*equiv-rl*] **by** *blast*
qed

lemma *null-uni-facts-3*[*PLM*]:
[*Null* ($\mathbf{a}_\emptyset$) *in v*]
proof −
let *?ψ* = λ *x* . *Null x*
have [((∃! *x* . *?ψ* (x^P)) → (∀ *y* . y^P = (ι*x* . *?ψ* (x^P)) → *?ψ* (y^P))) *in v*]
using *unique-box-desc*[*deduction*] *null-uni-facts-1*[*THEN* ∀ *I*] **by** *fast*
have *1*: [(∀ *y* . y^P = (ι*x* . *?ψ* (x^P)) → *?ψ* (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* . ($|A!,x^P|$) & ($|A!,y^P|$) & ($x \neq y$)
& ($\boldsymbol{\lambda}$ *z* . ($|R,z^P,x^P|$)) = ($\boldsymbol{\lambda}$ *z* . ($|R,z^P,y^P|$)) *in v*]
proof (*rule* $\forall$ *I*)
fix *R*
obtain *a* **where** ϑ:
[($|A!,a^P|$) & ($\forall$ *F* . $\{\!|a^P, F|\!\} \equiv$ ($\exists$ *y* . ($|A!,y^P|$)
& *F* = ($\boldsymbol{\lambda}$ *z* . ($|R,z^P,y^P|$)) & $\neg\{\!|y^P, F|\!\}$)) *in v*]
using *A-objects*[*axiom-instance*] **by** (*rule* $\exists$ *E*)
{
assume [$\neg\{\!|a^P, (\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$))$|\!\}$ *in v*]
hence [$\neg$(($|A!,a^P|$) & ($\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$)) = ($\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$))
& $\neg\{\!|a^P, (\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$))$|\!\}$) *in v*]
using ϑ[*conj2*, *THEN* $\forall$ *E*, *THEN oth-class-taut-5-d*[*equiv-lr*], *equiv-lr*]
cqt-further-4[*equiv-lr*] $\forall$ *E* **by** *fast*
hence [($|A!,a^P|$) & ($\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$)) = ($\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$))
$\rightarrow$ $\{\!|a^P, (\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$))$|\!\}$ *in v*]
apply − **by** *PLM-solver*
hence [$\{\!|a^P, (\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$))$|\!\}$ *in v*]
using ϑ[*conj1*] *id-eq-1* &*I vdash-properties-10* **by** *fast*
}
hence *1*: [$\{\!|a^P, (\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$))$|\!\}$ *in v*]
using *reductio-aa-1 CP if-p-then-p* **by** *blast*
then obtain *b* **where** ξ:
[($|A!,b^P|$) & ($\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$)) = ($\boldsymbol{\lambda}\ z$. ($|R,z^P,b^P|$))
& $\neg\{\!|b^P, (\boldsymbol{\lambda}\ z$. ($|R,z^P,a^P|$))$|\!\}$ *in v*]

```
    using ϑ[conj2, THEN ∀ E, equiv-lr] ∃ E by blast
  have [a ≠ b in v]
    proof −
      {
        assume [a = b in v]
        hence [{|b^P, (λ z . (|R,z^P,a^P|))|} in v]
          using 1 l-identity[axiom-instance, deduction, deduction] by fast
        hence ?thesis
          using ξ[conj2] reductio-aa-1 by blast
      }
      thus ?thesis using reductio-aa-1 by blast
    qed
  hence [(|A!,a^P|) & (|A!,b^P|) & a ≠ b
        & (λ z . (|R,z^P,a^P|)) = (λ z . (|R,z^P,b^P|)) in v]
    using ϑ[conj1] ξ[conj1, conj1] ξ[conj1, conj2] &I by presburger
  hence [∃ y . (|A!,a^P|) & (|A!,y^P|) & a ≠ y
        & (λz. (|R,z^P,a^P|)) = (λz. (|R,z^P,y^P|)) in v]
    using ∃ I by fast
  thus [∃ x y . (|A!,x^P|) & (|A!,y^P|) & x ≠ y
        & (λz. (|R,z^P,x^P|)) = (λz. (|R,z^P,y^P|)) in v]
    using ∃ I by fast
qed

lemma aclassical-2[PLM]:
  [∀ R . ∃ x y . (|A!,x^P|) & (|A!,y^P|) & (x ≠ y)
    & (λ z . (|R,x^P,z^P|)) = (λ z . (|R,y^P,z^P|)) in v]
  proof (rule ∀ I)
    fix R
    obtain a where ϑ:
      [(|A!,a^P|) & (∀ F . {|a^P, F|} ≡ (∃ y . (|A!,y^P|)
        & F = (λ z . (|R,y^P,z^P|)) & ¬{|y^P, F|})) in v]
      using A-objects[axiom-instance] by (rule ∃ E)
    {
      assume [¬{|a^P, (λ z . (|R,a^P,z^P|))|} in v]
      hence [¬((|A!,a^P|) & (λ z . (|R,a^P,z^P|)) = (λ z . (|R,a^P,z^P|))
            & ¬{|a^P, (λ z . (|R,a^P,z^P|))|}) in v]
        using ϑ[conj2, THEN ∀ E, THEN oth-class-taut-5-d[equiv-lr], equiv-lr]
              cqt-further-4[equiv-lr] ∀ E by fast
      hence [(|A!,a^P|) & (λ z . (|R,a^P,z^P|)) = (λ z . (|R,a^P,z^P|))
            → {|a^P, (λ z . (|R,a^P,z^P|))|} in v]
        apply − by PLM-solver
      hence [{|a^P, (λ z . (|R,a^P,z^P|))|} in v]
        using ϑ[conj1] id-eq-1 &I vdash-properties-10 by fast
    }
    hence 1: [{|a^P, (λ z . (|R,a^P,z^P|))|} in v]
      using reductio-aa-1 CP if-p-then-p by blast
    then obtain b where ξ:
      [(|A!,b^P|) & (λ z . (|R,a^P,z^P|)) = (λ z . (|R,b^P,z^P|))
        & ¬{|b^P, (λ z . (|R,a^P,z^P|))|} in v]
      using ϑ[conj2, THEN ∀ E, equiv-lr] ∃ E by blast
    have [a ≠ b in v]
      proof −
        {
          assume [a = b in v]
          hence [{|b^P, (λ z . (|R,a^P,z^P|))|} in v]
            using 1 l-identity[axiom-instance, deduction, deduction] by fast
          hence ?thesis using ξ[conj2] reductio-aa-1 by blast
        }
```

```
    thus ?thesis using ξ[conj2] reductio-aa-1 by blast
   qed
  hence [(|A!,a^P|) & (|A!,b^P|) & a ≠ b
        & (λ z . (|R,a^P,z^P|)) = (λ z . (|R,b^P,z^P|)) in v]
   using ϑ[conj1] ξ[conj1, conj1] ξ[conj1, conj2] &I by presburger
  hence [∃ y . (|A!,a^P|) & (|A!,y^P|) & a ≠ y
        & (λz. (|R,a^P,z^P|)) = (λz. (|R,y^P,z^P|)) in v]
   using ∃ I by fast
  thus [∃ x y . (|A!,x^P|) & (|A!,y^P|) & x ≠ y
        & (λz. (|R,x^P,z^P|)) = (λz. (|R,y^P,z^P|)) in v]
   using ∃ I by fast
 qed

lemma aclassical-3[PLM]:
 [∀ F . ∃ x y . (|A!,x^P|) & (|A!,y^P|) & (x ≠ y)
  & ((λ^0 (|F,x^P|)) = (λ^0 (|F,y^P|))) in v]
 proof (rule ∀ I)
  fix R
  obtain a where ϑ:
   [(|A!,a^P|) & (∀ F . {|a^P, F|} ≡ (∃ y . (|A!,y^P|)
    & F = (λ z . (|R,y^P|)) & ¬{|y^P, F|})) in v]
   using A-objects[axiom-instance] by (rule ∃ E)
  {
   assume [¬{|a^P, (λ z . (|R,a^P|))|} in v]
   hence [¬((|A!,a^P|) & (λ z . (|R,a^P|)) = (λ z . (|R,a^P|))
         & ¬{|a^P, (λ z . (|R,a^P|))|}) in v]
    using ϑ[conj2, THEN ∀ E, THEN oth-class-taut-5-d[equiv-lr], equiv-lr]
          cqt-further-4[equiv-lr] ∀ E by fast
   hence [(|A!,a^P|) & (λ z . (|R,a^P|)) = (λ z . (|R,a^P|))
         → {|a^P, (λ z . (|R,a^P|))|} in v]
    apply − by PLM-solver
   hence [{|a^P, (λ z . (|R,a^P|))|} in v]
    using ϑ[conj1] id-eq-1 &I vdash-properties-10 by fast
  }
  hence 1: [{|a^P, (λ z . (|R,a^P|))|} in v]
   using reductio-aa-1 CP if-p-then-p by blast
  then obtain b where ξ:
   [(|A!,b^P|) & (λ z . (|R,a^P|)) = (λ z . (|R,b^P|))
    & ¬{|b^P, (λ z . (|R,a^P|))|} in v]
   using ϑ[conj2, THEN ∀ E, equiv-lr] ∃ E by blast
  have [a ≠ b in v]
   proof −
    {
     assume [a = b in v]
     hence [{|b^P, (λ z . (|R,a^P|))|} in v]
      using 1 l-identity[axiom-instance, deduction, deduction] by fast
     hence ?thesis
      using ξ[conj2] reductio-aa-1 by blast
    }
    thus ?thesis using reductio-aa-1 by blast
   qed
  moreover {
   have [(|R,a^P|) = (|R,b^P|) in v]
    unfolding identity_o-def
    using ξ[conj1, conj2] by auto
   hence [(λ^0 (|R,a^P|)) = (λ^0 (|R,b^P|)) in v]
    using lambda-p-q-p-eq-q[equiv-rl] by simp
  }
```

ultimately have $[(\!|A!,a^P|\!)\ \&\ (\!|A!,b^P|\!)\ \&\ a \neq b$
$\&\ ((\boldsymbol{\lambda}^0\ (\!|R,a^P|\!)) = (\boldsymbol{\lambda}^0\ (\!|R,b^P|\!)))\ \textit{in}\ v]$
using $\vartheta[\textit{conj1}]\ \xi[\textit{conj1},\ \textit{conj1}]\ \xi[\textit{conj1},\ \textit{conj2}]\ \&I$
by *presburger*
hence $[\exists\ y\ .\ (\!|A!,a^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ a \neq y$
$\&\ (\boldsymbol{\lambda}^0\ (\!|R,a^P|\!)) = (\boldsymbol{\lambda}^0\ (\!|R,y^P|\!))\ \textit{in}\ v]$
using $\exists\, I$ **by** *fast*
thus $[\exists\ x\ y\ .\ (\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ x \neq y$
$\&\ (\boldsymbol{\lambda}^0\ (\!|R,x^P|\!)) = (\boldsymbol{\lambda}^0\ (\!|R,y^P|\!))\ \textit{in}\ v]$
using $\exists\, I$ **by** *fast*
qed

lemma *aclassical2*[*PLM*]:
$[\exists\ x\ y\ .\ (\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ x \neq y\ \&\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \textit{in}\ v]$
proof −
let $?R_1 = \boldsymbol{\lambda}^2\ (\lambda\ x\ y\ .\ \forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))$
have $[\exists\ x\ y\ .\ (\!|A!,x^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ x \neq y$
$\&\ (\boldsymbol{\lambda}z.\ (\!|?R_1,z^P,x^P|\!)) = (\boldsymbol{\lambda}z.\ (\!|?R_1,z^P,y^P|\!))\ \textit{in}\ v]$
using *aclassical-1* **by** (*rule* $\forall\, E$)
then obtain a **where**
$[\exists\ y\ .\ (\!|A!,a^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ a \neq y$
$\&\ (\boldsymbol{\lambda}z.\ (\!|?R_1,z^P,a^P|\!)) = (\boldsymbol{\lambda}z.\ (\!|?R_1,z^P,y^P|\!))\ \textit{in}\ v]$
by (*rule* $\exists\, E$)
then obtain b **where** *ab-prop*:
$[(\!|A!,a^P|\!)\ \&\ (\!|A!,b^P|\!)\ \&\ a \neq b$
$\&\ (\boldsymbol{\lambda}z.\ (\!|?R_1,z^P,a^P|\!)) = (\boldsymbol{\lambda}z.\ (\!|?R_1,z^P,b^P|\!))\ \textit{in}\ v]$
by (*rule* $\exists\, E$)
have $[(\!|?R_1,\ a^P,\ a^P|\!)\ \textit{in}\ v]$
apply (*rule beta-C-meta-2*[*equiv-rl*])
apply *show-proper*
using *oth-class-taut-4-a*[*THEN* $\forall\, I$] **by** *fast*
hence $[(\!|\boldsymbol{\lambda}\ z\ .\ (\!|?R_1,\ z^P,\ a^P|\!),\ a^P|\!)\ \textit{in}\ v]$
apply − **apply** (*rule beta-C-meta-1*[*equiv-rl*])
apply *show-proper*
by *auto*
hence $[(\!|\boldsymbol{\lambda}\ z\ .\ (\!|?R_1,\ z^P,\ b^P|\!),\ a^P|\!)\ \textit{in}\ v]$
using *ab-prop*[*conj2*] *l-identity*[*axiom-instance*, *deduction*, *deduction*]
by *fast*
hence $[(\!|?R_1,\ a^P,\ b^P|\!)\ \textit{in}\ v]$
apply (*safe intro*!: *beta-C-meta-1*[**where** $\varphi =$
$\lambda z\ .\ (\!|\boldsymbol{\lambda}^2\ (\lambda x\ y.\ \forall F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!)),z,b^P|\!),\ \textit{equiv-lr}$])
by *show-proper*
moreover have $\textit{IsProperInXY}\ (\lambda x\ y.\ \forall F.\ (\!|F,x|\!) \equiv (\!|F,y|\!))$
by *show-proper*
ultimately have $[\forall F.\ (\!|F,a^P|\!) \equiv (\!|F,b^P|\!)\ \textit{in}\ v]$
using *beta-C-meta-2*[*equiv-lr*] **by** *blast*
hence $[(\!|A!,a^P|\!)\ \&\ (\!|A!,b^P|\!)\ \&\ a \neq b\ \&\ (\forall F.\ (\!|F,a^P|\!) \equiv (\!|F,b^P|\!))\ \textit{in}\ v]$
using *ab-prop*[*conj1*] $\&I$ **by** *presburger*
hence $[\exists\ y\ .\ (\!|A!,a^P|\!)\ \&\ (\!|A!,y^P|\!)\ \&\ a \neq y\ \&\ (\forall F.\ (\!|F,a^P|\!) \equiv (\!|F,y^P|\!))\ \textit{in}\ v]$
using $\exists\, I$ **by** *fast*
thus *?thesis* **using** $\exists\, I$ **by** *fast*
qed

A.9.13. Propositional Properties

lemma *prop-prop2-1*:
$[\forall\ p\ .\ \exists\ F\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)\ \textit{in}\ v]$
proof (*rule* $\forall\, I$)

```
    fix p
    have [(λ x . p) = (λ x . p) in v]
      using id-eq-prop-prop-1 by auto
    thus [∃ F . F = (λ x . p) in v]
      by PLM-solver
  qed

lemma prop-prop2-2:
  [F = (λ x . p) → □(∀ x . (|F,x^P|) ≡ p) in v]
  proof (rule CP)
    assume 1: [F = (λ x . p) in v]
    {
      fix v
      {
        fix x
        have [(|(λ x . p), x^P|) ≡ p in v]
          apply (rule beta-C-meta-1)
          by show-proper
      }
      hence [∀ x . (|(λ x . p), x^P|) ≡ p in v]
        by (rule ∀I)
    }
    hence [□(∀ x . (|(λ x . p), x^P|) ≡ p) in v]
      by (rule RN)
    thus [□(∀x. (|F,x^P|) ≡ p) in v]
      using l-identity[axiom-instance,deduction,deduction,
            OF 1[THEN id-eq-prop-prop-2[deduction]]] by fast
  qed

lemma prop-prop2-3:
  [Propositional F → □(Propositional F) in v]
  proof (rule CP)
    assume [Propositional F in v]
    hence [∃ p . F = (λ x . p) in v]
      unfolding Propositional-def .
    then obtain q where [F = (λ x . q) in v]
      by (rule ∃E)
    hence [□(F = (λ x . q)) in v]
      using id-nec[equiv-lr] by auto
    hence [∃ p . □(F = (λ x . p)) in v]
      using ∃I by fast
    thus [□(Propositional F) in v]
      unfolding Propositional-def
      using sign-S5-thm-1[deduction] by fast
  qed


lemma prop-indis:
  [Indiscriminate F → (¬(∃ x y . (|F,x^P|) & (¬(|F,y^P|)))) in v]
  proof (rule CP)
    assume [Indiscriminate F in v]
    hence 1: [□((∃x. (|F,x^P|)) → (∀x. (|F,x^P|))) in v]
      unfolding Indiscriminate-def .
    {
      assume [∃ x y . (|F,x^P|) & ¬(|F,y^P|) in v]
      then obtain x where [∃ y . (|F,x^P|) & ¬(|F,y^P|) in v]
        by (rule ∃E)
      then obtain y where 2: [(|F,x^P|) & ¬(|F,y^P|) in v]
```

```
      by (rule ∃ E)
    hence [∃ x . (|F, x^P|) in v]
      using &E(1) ∃ I by fast
    hence [∀ x . (|F,x^P|) in v]
      using 1[THEN qml-2[axiom-instance, deduction], deduction] by fast
    hence [(|F,y^P|) in v]
      using cqt-orig-1[deduction] by fast
    hence [(|F,y^P|) & (¬(|F,y^P|)) in v]
      using 2 &I &E by fast
    hence [¬(∃ x y . (|F,x^P|) & ¬(|F,y^P|)) in v]
      using pl-1[axiom-instance, deduction, THEN modus-tollens-1]
            oth-class-taut-1-a by blast
  }
  thus [¬(∃ x y . (|F,x^P|) & ¬(|F,y^P|)) in v]
    using reductio-aa-2 if-p-then-p deduction-theorem by blast
qed


lemma prop-in-thm:
  [Propositional F → Indiscriminate F in v]
  proof (rule CP)
    assume [Propositional F in v]
    hence [□(Propositional F) in v]
      using prop-prop2-3[deduction] by auto
    moreover {
      fix w
      assume [∃ p . (F = (λ y . p)) in w]
      then obtain q where q-prop: [F = (λ y . q) in w]
        by (rule ∃ E)
      {
        assume [∃ x . (|F,x^P|) in w]
        then obtain a where [(|F,a^P|) in w]
          by (rule ∃ E)
        hence [(|λ y . q, a^P|) in w]
          using q-prop l-identity[axiom-instance,deduction,deduction] by fast
        hence q: [q in w]
          apply (safe intro!: beta-C-meta-1[where φ=λy. q, equiv-lr])
           apply show-proper
          by simp
        {
          fix x
          have [(|λ y . q, x^P|) in w]
            apply (safe intro!: q beta-C-meta-1[equiv-rl])
            by show-proper
          hence [(|F,x^P|) in w]
            using q-prop[eq-sym] l-identity[axiom-instance, deduction, deduction]
            by fast
        }
        hence [∀ x . (|F,x^P|) in w]
          by (rule ∀ I)
      }
      hence [(∃ x . (|F,x^P|)) → (∀ x . (|F, x^P|)) in w]
        by (rule CP)
    }
    ultimately show [Indiscriminate F in v]
      unfolding Propositional-def Indiscriminate-def
      using RM-1[deduction] deduction-theorem by blast
  qed
```

lemma *prop-in-f-1*:
[*Necessary F* $\rightarrow$ *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* $\rightarrow$ *Indiscriminate F in v*]
proof −
{
fix *w*
have [$(\neg(\exists\ x\ .\ (\!|F,x^P|\!))) \rightarrow ((\exists\ x\ .\ (\!|F,x^P|\!)) \rightarrow (\forall\ x\ .\ (\!|F,x^P|\!)))$ *in w*]
using *useful-tautologies-3* **by** *auto*
hence [$(\forall\ x\ .\ \neg(\!|F,x^P|\!)) \rightarrow ((\exists\ x\ .\ (\!|F,x^P|\!)) \rightarrow (\forall\ x\ .\ (\!|F,x^P|\!)))$ *in w*]
apply − **apply** (*PLM-subst-method* $\neg(\exists\ x.\ (\!|F,x^P|\!))\ (\forall\ x.\ \neg(\!|F,x^P|\!))$)
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*:
[$\neg$(*Indiscriminate* (*E*!)) *in v*]
proof (*rule reductio-aa-2*)
show [$\Box\neg(\forall x.\ (\!|E!,x^P|\!))$ *in v*]
using *a-objects-exist-3* **.**
next
assume [*Indiscriminate E*! *in v*]
thus [$\neg\Box\neg(\forall\ x\ .\ (\!|E!,x^P|\!))$ *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*:
[$\neg$(*Indiscriminate* (*E*!$^-$)) *in v*]
proof (*rule reductio-aa-2*)
assume [*Indiscriminate* (*E*!$^-$) *in v*]
moreover have [$\Box(\exists\ x\ .\ (\!|E!^-,\ x^P|\!))$ *in v*]
apply (*PLM-subst-method* $\lambda\ x\ .\ \neg(\!|E!,\ x^P|\!)\ \lambda\ x\ .\ (\!|E!^-,\ x^P|\!)$)
using *thm-relation-negation-1-1*[*equiv-sym*] **apply** *simp*
unfolding *exists-def*
apply (*PLM-subst-method* $\lambda\ x\ .\ (\!|E!,\ x^P|\!)\ \lambda\ x\ .\ \neg\neg(\!|E!,\ x^P|\!)$)
using *oth-class-taut-4-b* **apply** *simp*
using *a-objects-exist-3* **by** *auto*
ultimately have [$\Box(\forall x.\ (\!|E!^-,x^P|\!))$ *in v*]
unfolding *Indiscriminate-def*
using *qml-1*[*axiom-instance*, *deduction*, *deduction*] **by** *blast*
thus [$\Box(\forall x.\ \neg(\!|E!,x^P|\!))$ *in v*]
apply −
apply (*PLM-subst-method* $\lambda\ x\ .\ (\!|E!^-,\ x^P|\!)\ \lambda\ x\ .\ \neg(\!|E!,\ x^P|\!)$)
using *thm-relation-negation-1-1* **by** *auto*
next
show [$\neg\Box(\forall\ x\ .\ \neg(\!|E!,\ x^P|\!))$ *in v*]
using *o-objects-exist-1*
unfolding *diamond-def exists-def*
apply −
apply (*PLM-subst-method* $\neg\neg(\forall x.\ \neg(\!|E!,x^P|\!))\ \forall x.\ \neg(\!|E!,x^P|\!)$)

using *oth-class-taut-4-b*[*equiv-sym*] **by** *auto*
qed

lemma *prop-in-f-3-c*:
[¬(*Indiscriminate* (*O*!)) *in v*]
proof (*rule reductio-aa-2*)
show [¬(∀ *x* . (|*O*!,x^P|)) *in v*]
using *a-objects-exist-2*[*THEN qml-2*[*axiom-instance*, *deduction*]]
by *blast*
next
assume [*Indiscriminate O*! *in v*]
thus [(∀ *x* . (|*O*!,x^P|)) *in v*]
unfolding *Indiscriminate-def*
using *o-objects-exist-2 qml-1*[*axiom-instance*, *deduction*, *deduction*]
qml-2[*axiom-instance*, *deduction*] **by** *blast*
qed

lemma *prop-in-f-3-d*:
[¬(*Indiscriminate* (*A*!)) *in v*]
proof (*rule reductio-aa-2*)
show [¬(∀ *x* . (|*A*!,x^P|)) *in v*]
using *o-objects-exist-3*[*THEN qml-2*[*axiom-instance*, *deduction*]]
by *blast*
next
assume [*Indiscriminate A*! *in v*]
thus [(∀ *x* . (|*A*!,x^P|)) *in v*]
unfolding *Indiscriminate-def*
using *a-objects-exist-1 qml-1*[*axiom-instance*, *deduction*, *deduction*]
qml-2[*axiom-instance*, *deduction*] **by** *blast*
qed

lemma *prop-in-f-4-a*:
[¬(*Propositional E*!) *in v*]
using *prop-in-thm*[*deduction*] *prop-in-f-3-a modus-tollens-1 CP*
by *meson*

lemma *prop-in-f-4-b*:
[¬(*Propositional* ($E!^-$)) *in v*]
using *prop-in-thm*[*deduction*] *prop-in-f-3-b modus-tollens-1 CP*
by *meson*

lemma *prop-in-f-4-c*:
[¬(*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* $\exists\, E$)
hence $[\Diamond\Box(\forall\, x.\ \{\!|x^P,F|\!\} \equiv \{\!|x^P,\boldsymbol{\lambda}x.\ p|\!\})\ in\ v]$
unfolding *identity-defs* .
hence $[\Box(\forall\, x.\ \{\!|x^P,F|\!\} \equiv \{\!|x^P,\boldsymbol{\lambda}x.\ p|\!\})\ in\ v]$
using *5*$\Diamond$[*deduction*] **by** *auto*
hence $[(F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
unfolding *identity-defs* .
thus $[\exists\ \ p\ .\ (F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
by *PLM-solver*
qed

lemma *prop-prop-nec-2*:
$[(\forall\ \ p\ .\ F \neq (\boldsymbol{\lambda}\ x\ .\ p)) \rightarrow \Box(\forall\ \ p\ .\ F \neq (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
apply (*PLM-subst-method*
$\neg(\exists\ \ p\ .\ (F = (\boldsymbol{\lambda}\ x\ .\ p)))$
$(\forall\ \ p\ .\ \neg(F = (\boldsymbol{\lambda}\ x\ .\ p))))$
using *cqt-further-4* **apply** *blast*
apply (*PLM-subst-method*
$\neg\Diamond(\exists\, p.\ F = (\boldsymbol{\lambda}x.\ p))$
$\Box\neg(\exists\, p.\ F = (\boldsymbol{\lambda}x.\ p)))$
using *KBasic2-4*[*equiv-sym*] *prop-prop-nec-1*
contraposition-1 **by** *auto*

lemma *prop-prop-nec-3*:
$[(\exists\ \ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)) \rightarrow \Box(\exists\ \ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
using *prop-prop-nec-1 derived-S5-rules-1-b* **by** *simp*

lemma *prop-prop-nec-4*:
$[\Diamond(\forall\ \ p\ .\ F \neq (\boldsymbol{\lambda}\ x\ .\ p)) \rightarrow (\forall\ \ p\ .\ F \neq (\boldsymbol{\lambda}\ x\ .\ p))\ in\ v]$
using *prop-prop-nec-2 derived-S5-rules-2-b* **by** *simp*

lemma *enc-prop-nec-1*:
$[\Diamond(\forall\ \ F\ .\ \{\!|x^P,\ F|\!\} \rightarrow (\exists\ \ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)))$
$\rightarrow (\forall\ \ F\ .\ \{\!|x^P,\ F|\!\} \rightarrow (\exists\ \ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)))\ in\ v]$
proof (*rule CP*)
assume $[\Diamond(\forall\, F.\ \{\!|x^P,F|\!\} \rightarrow (\exists\, p.\ F = (\boldsymbol{\lambda}x.\ p)))\ in\ v]$
hence *1*: $[(\forall\, F.\ \Diamond(\{\!|x^P,F|\!\} \rightarrow (\exists\, p.\ F = (\boldsymbol{\lambda}x.\ p))))\ in\ v]$
using *Buridan*$\Diamond$[*deduction*] **by** *auto*
{
fix *Q*
assume $[\{\!|x^P,Q|\!\}\ in\ v]$
hence $[\Box\{\!|x^P,Q|\!\}\ in\ v]$
using *encoding*[*axiom-instance*, *deduction*] **by** *auto*
moreover have $[\Diamond(\{\!|x^P,Q|\!\} \rightarrow (\exists\, p.\ Q = (\boldsymbol{\lambda}x.\ p)))\ in\ v]$
using *cqt-1*[*axiom-instance*, *deduction*] *1* **by** *fast*
ultimately have $[\Diamond(\exists\, p.\ Q = (\boldsymbol{\lambda}x.\ p))\ in\ v]$
using *KBasic2-9*[*equiv-lr*,*deduction*] **by** *auto*
hence $[(\exists\, p.\ Q = (\boldsymbol{\lambda}x.\ p))\ in\ v]$
using *prop-prop-nec-1*[*deduction*] **by** *auto*
}
thus $[(\forall\ \ F\ .\ \{\!|x^P,\ F|\!\} \rightarrow (\exists\ \ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)))\ in\ v]$
apply $-$ **by** *PLM-solver*
qed

lemma *enc-prop-nec-2*:
$[(\forall\ \ F\ .\ \{\!|x^P,\ F|\!\} \rightarrow (\exists\ \ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p))) \rightarrow \Box(\forall\ \ F\ .\ \{\!|x^P,\ F|\!\}$
$\rightarrow (\exists\ \ p\ .\ F = (\boldsymbol{\lambda}\ x\ .\ p)))\ in\ v]$
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 $x \equiv (\!|A!,x|\!)$ **&** $(\forall\ F.\ \{\!|x,F|\!\} \rightarrow$ *Propositional* $F)$

definition *EncodeProposition* (**infixl** $\mathbf{\Sigma}$ *70*) **where**
$x\mathbf{\Sigma}p \equiv (\!|A!,x|\!)$ **&** $\{\!|x, \boldsymbol{\lambda}\ x\ .\ p|\!\}$
definition *TrueInSituation* (**infixl** $\models$ *10*) **where**
$x \models p \equiv$ *Situation* x **&** $x\mathbf{\Sigma}p$
definition *PossibleWorld* **where**
PossibleWorld $x \equiv$ *Situation* x **&** $\Diamond(\forall\ p\ .\ x\mathbf{\Sigma}p \equiv p)$

A.10.2. Auxiliary Lemmata

lemma *possit-sit-1*:
$[$*Situation* $(x^P) \equiv \Box($*Situation* $(x^P))$ *in* $v]$
proof (*rule* $\equiv I$; *rule CP*)
assume $[$*Situation* (x^P) *in* $v]$
hence *1*: $[(\!|A!,x^P|\!)$ **&** $(\forall\ F.\ \{\!|x^P,F|\!\} \rightarrow$ *Propositional* $F)$ *in* $v]$
unfolding *Situation-def* **by** *auto*
have $[\Box(\!|A!,x^P|\!)$ *in* $v]$
using *1*[*conj1*, *THEN oa-facts-2*[*deduction*]] **.**
moreover have $[\Box(\forall\ F.\ \{\!|x^P,F|\!\} \rightarrow$ *Propositional* $F)$ *in* $v]$
using *1*[*conj2*] **unfolding** *Propositional-def*
by (*rule enc-prop-nec-2*[*deduction*])
ultimately show $[\Box$*Situation* (x^P) *in* $v]$
unfolding *Situation-def*
apply *cut-tac* **apply** (*rule KBasic-3*[*equiv-rl*])
by (*rule intro-elim-1*)
next
assume $[\Box$*Situation* (x^P) *in* $v]$
thus $[$*Situation* (x^P) *in* $v]$
using *qml-2*[*axiom-instance*, *deduction*] **by** *auto*
qed

lemma *possworld-nec*:
$[$*PossibleWorld* $(x^P) \equiv \Box($*PossibleWorld* $(x^P))$ *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 *TrueInWorldNecc*:
$[((x^P) \models p) \equiv \Box((x^P) \models p)$ *in* $v]$
proof (*rule* $\equiv I$; *rule CP*)

```
    assume [x^P ⊨ p in v]
    hence [Situation (x^P) & (⦇A!,x^P⦈ & ⦃x^P,λx. p⦄) in v]
      unfolding TrueInSituation-def EncodeProposition-def .
    hence [(□Situation (x^P) & □(⦇A!,x^P⦈)) & □⦃x^P, λx. p⦄ in v]
      using &I &E possit-sit-1[equiv-lr] oa-facts-2[deduction]
            encoding[axiom-instance,deduction] by metis
    thus [□((x^P) ⊨ p) in v]
      unfolding TrueInSituation-def EncodeProposition-def
      using KBasic-3[equiv-rl] &I &E by metis
  next
    assume [□(x^P ⊨ p) in v]
    thus [x^P ⊨ p in v]
      using qml-2[axiom-instance,deduction] by auto
  qed


lemma PossWorldAux:
  [((⦇A!,x^P⦈) & (∀ F . (⦃x^P,F⦄ ≡ (∃ p . p & (F = (λ x . p))))))
     → (PossibleWorld (x^P)) in v]
  proof (rule CP)
    assume DefX: [(⦇A!,x^P⦈) & (∀ F . (⦃x^P,F⦄ ≡
          (∃ p . p & (F = (λ x . p))))) in v]

    have [Situation (x^P) in v]
    proof −
      have [(⦇A!,x^P⦈) in v]
        using DefX[conj1] .
      moreover have [(∀ F. ⦃x^P,F⦄ → Propositional F) in v]
        proof (rule ∀ I; rule CP)
          fix F
          assume [⦃x^P,F⦄ in v]
          moreover have [⦃x^P,F⦄ ≡ (∃ p . p & (F = (λ x . p))) in v]
            using DefX[conj2] cqt-1[axiom-instance, deduction] by auto
          ultimately have [(∃ p . p & (F = (λ x . p))) in v]
            using ≡E(1) by blast
          then obtain p where [p & (F = (λ x . p)) in v]
            by (rule ∃ E)
          hence [(F = (λ x . p)) in v]
            by (rule &E(2))
          hence [(∃ p . (F = (λ x . p))) in v]
            by PLM-solver
          thus [Propositional F in v]
            unfolding Propositional-def .
        qed
      ultimately show [Situation (x^P) in v]
        unfolding Situation-def by (rule &I)
    qed
    moreover have [◇(∀ p. x^P Σ p ≡ p) in v]
      unfolding EncodeProposition-def
      proof (rule TBasic[deduction]; rule ∀ I)
        fix q
        have EncodeLambda:
          [⦃x^P, λx. q⦄ ≡ (∃ p . p & ((λx. q) = (λ x . p))) in v]
          using DefX[conj2] by (rule cqt-1[axiom-instance, deduction])
        moreover {
          assume [q in v]
          moreover have [(λx. q) = (λ x . q) in v]
            using id-eq-prop-prop-1 by auto
```

```
      ultimately have [q & ((λx. q) = (λ x . q)) in v]
        by (rule &I)
      hence [∃ p . p & ((λx. q) = (λ x . p)) in v]
        by PLM-solver
      moreover have [(|A!,x^P|) in v]
        using DefX[conj1] .
      ultimately have [(|A!,x^P|) & {|x^P, λx. q|} in v]
        using EncodeLambda[equiv-rl] &I by auto
    }
    moreover {
      assume [(|A!,x^P|) & {|x^P, λx. q|} in v]
      hence [{|x^P, λx. q|} in v]
        using &E(2) by auto
      hence [∃ p . p & ((λx. q) = (λ x . p)) in v]
        using EncodeLambda[equiv-lr] by auto
      then obtain p where p-and-lambda-q-is-lambda-p:
        [p & ((λx. q) = (λ x . p)) in v]
        by (rule ∃E)
      have [(|(λ x . p), x^P|) ≡ p in v]
        apply (rule beta-C-meta-1)
        by show-proper
      hence [(|(λ x . p), x^P|) in v]
        using p-and-lambda-q-is-lambda-p[conj1] ≡E(2) by auto
      hence [(|(λ x . q), x^P|) in v]
        using p-and-lambda-q-is-lambda-p[conj2, THEN id-eq-prop-prop-2[deduction]]
          l-identity[axiom-instance, deduction, deduction] by fast
      moreover have [(|(λ x . q), x^P|) ≡ q in v]
        apply (rule beta-C-meta-1) by show-proper
      ultimately have [q in v]
        using ≡E(1) by blast
    }
    ultimately show [(|A!,x^P|) & {|x^P,λx. q|} ≡ q in v]
      using &I ≡I CP by auto
  qed

  ultimately show [PossibleWorld (x^P) in v]
    unfolding PossibleWorld-def by (rule &I)
qed
```

A.10.3. For every syntactic Possible World there is a semantic Possible World

```
theorem SemanticPossibleWorldForSyntacticPossibleWorlds:
 ∀ x . [PossibleWorld (x^P) in w] ⟶
  (∃ v . ∀ p . [p in v] ⟷ [(x^P ⊨ p) in w])
 proof
  fix x
  {
   assume PossWorldX: [PossibleWorld (x^P) in w]
   hence SituationX: [Situation (x^P) in w]
     unfolding PossibleWorld-def apply cut-tac by PLM-solver
   have PossWorldExpanded:
    [(|A!,x^P|) & (∀ F. {|x^P,F|} → (∃ p. F = (λx. p)))
      & ◇(∀ p. (|A!,x^P|) & {|x^P,λx. p|} ≡ p) in w]
     using PossWorldX
     unfolding PossibleWorld-def Situation-def
           Propositional-def EncodeProposition-def .
   have AbstractX: [(|A!,x^P|) in w]
     using PossWorldExpanded[conj1,conj1] .
```

```
  have [◇(∀ p. ⦃x^P,λx. p⦄ ≡ p) in w]
    apply (PLM-subst-method
          λp. (|A!,x^P|) & ⦃x^P,λx. p⦄
          λ p . ⦃x^P,λx. p⦄)
     subgoal using PossWorldExpanded[conj1,conj1,THEN oa-facts-2[deduction]]
           using Semantics.T6 apply cut-tac by PLM-solver
    using PossWorldExpanded[conj2] .

  hence ∃ v. ∀ p. ([⦃x^P,λx. p⦄ in v])
                = [p in v]
   unfolding diamond-def equiv-def conj-def
   apply (simp add: Semantics.T4 Semantics.T6 Semantics.T5
                    Semantics.T8)
   by auto

  then obtain v where PropsTrueInSemWorld:
    ∀ p. ([⦃x^P,λx. p⦄ in v]) = [p in v]
    by auto
  {
    fix p
    {
      assume [((x^P) ⊨ p) in w]
      hence [((x^P) ⊨ p) in v]
        using TrueInWorldNecc[equiv-lr] Semantics.T6 by simp
      hence [Situation (x^P) & ((|A!,x^P|) & ⦃x^P,λx. p⦄) in v]
        unfolding TrueInSituation-def EncodeProposition-def .
      hence [⦃x^P,λx. p⦄ in v]
        using &E(2) by blast
      hence [p in v]
        using PropsTrueInSemWorld by blast
    }
    moreover {
      assume [p in v]
      hence [⦃x^P,λx. p⦄ in v]
        using PropsTrueInSemWorld by blast
      hence [(x^P) ⊨ p in v]
        apply cut-tac unfolding TrueInSituation-def EncodeProposition-def
        apply (rule &I) using SituationX[THEN possit-sit-1[equiv-lr]]
        subgoal using Semantics.T6 by auto
        apply (rule &I)
        subgoal using AbstractX[THEN oa-facts-2[deduction]]
          using Semantics.T6 by auto
        by assumption
      hence [□((x^P) ⊨ p) in v]
        using TrueInWorldNecc[equiv-lr] by simp
      hence [(x^P) ⊨ p in w]
        using Semantics.T6 by simp
    }
    ultimately have [p in v] ⟷ [(x^P) ⊨ p in w]
      by auto
  }
  hence (∃ v . ∀ p . [p in v] ⟷ [(x^P) ⊨ p in w])
    by blast
}
thus [PossibleWorld (x^P) in w] ⟶
     (∃ v. ∀ p . [p in v] ⟷ [(x^P) ⊨ p in w])
  by blast
```

qed

A.10.4. For every semantic Possible World there is a syntactic Possible World

theorem *SyntacticPossibleWorldForSemanticPossibleWorlds*:
$\forall\ v\ .\ \exists\ x\ .\ [PossibleWorld\ (x^P)\ in\ w] \wedge$
$(\forall\ p\ .\ [p\ in\ v] \longleftrightarrow [((x^P) \models p)\ in\ w])$
proof
fix v
have $[\exists\, x.\ (\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ (\{\!|x^P,F|\!\} \equiv$
$(\exists\ p\ .\ p\ \&\ (F = (\boldsymbol{\lambda}\ x\ .\ p)))))\ in\ v]$
using *A-objects*[*axiom-instance*] **by** *fast*
then obtain x **where** *DefX*:
$[(\!|A!,x^P|\!)\ \&\ (\forall\ F\ .\ (\{\!|x^P,F|\!\} \equiv (\exists\ p\ .\ p\ \&\ (F = (\boldsymbol{\lambda}\ x\ .\ p)))))\ in\ v]$
by (*rule* $\exists\, E$)
hence *PossWorldX*: $[PossibleWorld\ (x^P)\ in\ v]$
using *PossWorldAux*[*deduction*] **by** *blast*
hence $[PossibleWorld\ (x^P)\ in\ w]$
using *possworld-nec*[*equiv-lr*] *Semantics.T6* **by** *auto*
moreover have $(\forall\ p\ .\ [p\ in\ v] \longleftrightarrow [(x^P) \models p\ in\ w])$
proof
fix q
{
assume $[q\ in\ v]$
moreover have $[(\boldsymbol{\lambda}\ x\ .\ q) = (\boldsymbol{\lambda}\ x\ .\ q)\ in\ v]$
using *id-eq-prop-prop-1* **by** *auto*
ultimately have $[q\ \&\ (\boldsymbol{\lambda}\ x\ .\ q) = (\boldsymbol{\lambda}\ x\ .\ q)\ in\ v]$
using &I **by** *auto*
hence $[(\exists\ p\ .\ p\ \&\ ((\boldsymbol{\lambda}\ x\ .\ q) = (\boldsymbol{\lambda}\ x\ .\ p)))\ in\ v]$
by *PLM-solver*
hence *4*: $[\{\!|x^P,\ (\boldsymbol{\lambda}\ x\ .\ q)|\!\}\ in\ v]$
using *cqt-1*[*axiom-instance*,*deduction*, *OF DefX*[*conj2*], *equiv-rl*]
by *blast*
have $[(x^P \models q)\ in\ v]$
unfolding *TrueInSituation-def* **apply** (*rule* &I)
using *PossWorldX* **unfolding** *PossibleWorld-def*
using &$E(1)$ **apply** *blast*
unfolding *EncodeProposition-def* **apply** (*rule* &I)
using *DefX*[*conj1*] **apply** *simp*
using *4* **.**
hence $[(x^P \models q)\ in\ w]$
using *TrueInWorldNecc*[*equiv-lr*] *Semantics.T6* **by** *auto*
}
moreover {
assume $[(x^P \models q)\ in\ w]$
hence $[(x^P \models q)\ in\ v]$
using *TrueInWorldNecc*[*equiv-lr*] *Semantics.T6*
by *auto*
hence $[\{\!|x^P,\ (\boldsymbol{\lambda}\ x\ .\ q)|\!\}\ in\ v]$
unfolding *TrueInSituation-def EncodeProposition-def*
using &$E(2)$ **by** *blast*
hence $[(\exists\ p\ .\ p\ \&\ ((\boldsymbol{\lambda}\ x\ .\ q) = (\boldsymbol{\lambda}\ x\ .\ p)))\ in\ v]$
using *cqt-1*[*axiom-instance*,*deduction*, *OF DefX*[*conj2*], *equiv-lr*]
by *blast*
then obtain p **where** *4*:
$[(p\ \&\ ((\boldsymbol{\lambda}\ x\ .\ q) = (\boldsymbol{\lambda}\ x\ .\ p)))\ in\ v]$
by (*rule* $\exists\, E$)
have $[(\!|(\boldsymbol{\lambda}\ x\ .\ p),x^P|\!) \equiv p\ in\ v]$

```
      apply (rule beta-C-meta-1)
      by show-proper
    hence [(|(λ x . q),x^P|) ≡ p in v]
      using l-identity[where β=(λ x . q) and α=(λ x . p),
                       axiom-instance, deduction, deduction]
      using 4[conj2,THEN id-eq-prop-prop-2[deduction]] by meson
    hence [(|(λ x . q),x^P|) in v] using 4[conj1] ≡E(2) by blast
    moreover have [(|(λ x . q),x^P|) ≡ q in v]
      apply (rule beta-C-meta-1)
      by show-proper
    ultimately have [q in v]
      using ≡E(1) by blast
  }
  ultimately show [q in v] ⟷ [(x^P) ⊨ q in w]
    by blast
 qed
 ultimately show ∃ x . [PossibleWorld (x^P) in w]
                  ∧ (∀ p . [p in v] ⟷ [(x^P) ⊨ p in w])
  by auto
qed
end
```

A.11. Artificial Theorems

Remark A.24. *Some examples of theorems that can be derived from the meta-logic, but which are (presumably) not derivable from the deductive system PLM itself.*

TODO: add theorem about missing state dependence of $\nu\upsilon$.

```
locale ArtificialTheorems
begin

  lemma lambda-enc-1:
    [(|λx . {|x^P, F|} ≡ {|x^P, F|}, y^P|) in v]
    by (auto simp: meta-defs meta-aux conn-defs forall-Π1-def)

  lemma lambda-enc-2:
    [(|λ x . {|y^P, G|}, x^P|) ≡ {|y^P, G|} in v]
    by (auto simp: meta-defs meta-aux conn-defs forall-Π1-def)
```

Remark A.25. *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 . {|x^P, F|}, x^P|) → {|x^P, 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 A.26. *Instead the following two statements hold.*

```
  lemma lambda-enc-4:
    [(|(λ x . {|x^P, F|}), x^P|) in v] = (∃ y . νυ y = νυ x ∧ [{|y^P, F|} in v])
    by (simp add: meta-defs meta-aux)
```

lemma *lambda-ex*:
$[(\!|(\boldsymbol{\lambda}\ x\ .\ \varphi\ x),\ x^P|\!)\ in\ v] = (\exists\ y\ .\ \nu\upsilon\ y = \nu\upsilon\ x \wedge [\varphi\ y\ in\ v])$
by (*simp add*: *meta-defs meta-aux*)

Remark A.27. *These statements can also be translated to statements in the embedded logic.*

lemma *lambda-ex-emb*:
$[(\!|(\boldsymbol{\lambda}\ x\ .\ \varphi\ x),\ x^P|\!) \equiv (\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ y)\ in\ v]$
proof(*rule MetaSolver.EquivI*)
interpret *MetaSolver* **.**
{
assume $[(\!|(\boldsymbol{\lambda}\ x\ .\ \varphi\ x),\ x^P|\!)\ in\ v]$
then obtain y **where** $\nu\upsilon\ y = \nu\upsilon\ x \wedge [\varphi\ y\ in\ v]$
using *lambda-ex* **by** *blast*
moreover hence $[(\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ in\ v]$
apply − **apply** *meta-solver*
by (*simp add*: *Semantics.*d_κ*-proper Semantics.ex1-def*)
ultimately have $[\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ y\ in\ v]$
using *ExIRule ConjI* **by** *fast*
}
moreover {
assume $[\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ y\ in\ v]$
then obtain y **where** *y-def*: $[(\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ y\ in\ v]$
by (*rule ExERule*)
hence $\bigwedge F\ .\ [(\!|F,x^P|\!)\ in\ v] = [(\!|F,y^P|\!)\ in\ v]$
apply − **apply** (*drule ConjE*) **apply** (*drule conjunct1*)
apply (*drule AllE*) **apply** (*drule EquivE*) **by** *simp*
hence $[(\!|make\Pi_1\ (\lambda\ u\ s\ w\ .\ \nu\upsilon\ y = u),x^P|\!)\ in\ v]$
$= [(\!|make\Pi_1\ (\lambda\ u\ s\ w\ .\ \nu\upsilon\ y = u),y^P|\!)\ in\ v]$ **by** *auto*
hence $\nu\upsilon\ y = \nu\upsilon\ x$ **by** (*simp add*: *meta-defs meta-aux*)
moreover have $[\varphi\ y\ in\ v]$ **using** *y-def ConjE* **by** *blast*
ultimately have $[(\!|(\boldsymbol{\lambda}\ x\ .\ \varphi\ x),\ x^P|\!)\ in\ v]$
using *lambda-ex* **by** *blast*
}
ultimately show $[(\!|\boldsymbol{\lambda}x.\ \varphi\ x,x^P|\!)\ in\ v]$
$= [\exists\ y.\ (\forall\ F.\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \varphi\ y\ in\ v]$
by *auto*
qed

lemma *lambda-enc-emb*:
$[(\!|(\boldsymbol{\lambda}\ x\ .\ \{\!|x^P,\ F|\!\}),\ x^P|\!) \equiv (\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ \{\!|y^P,\ F|\!\})\ in\ v]$
using *lambda-ex-emb* **by** *simp*

lemma *lambda-desc*:
$[(\!|(\boldsymbol{\lambda}\ x\ .\ (\!|F,\boldsymbol{\iota}z\ .\ \varphi\ z\ x|\!)),\ x^P|\!) \equiv (\exists\ y\ .\ (\forall\ F\ .\ (\!|F,x^P|\!) \equiv (\!|F,y^P|\!))\ \&\ (\!|F,\boldsymbol{\iota}z\ .\ \varphi\ z\ y|\!))\ in\ v]$
using *lambda-ex-emb* **by** *simp*

end

A.12. Sanity Tests

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

A.12.1. Consistency

lemma *True*
nitpick[*expect*=*genuine*, *user-axioms*, *satisfy*]
by *auto*

A.12.2. Intensionality

lemma $[(\boldsymbol{\lambda} y.\ (q \vee \neg q)) = (\boldsymbol{\lambda} y.\ (p \vee \neg p))\ in\ v]$
unfolding *identity-*Π_1*-def conn-defs*
apply (*rule* $Eq_1 I$) **apply** (*simp add*: *meta-defs*)
nitpick[*expect* = *genuine*, *user-axioms*=*true*, *card i* = *2*,
card j = *2*, *card* ω = *1*, *card* σ = *1*,
sat-solver = *MiniSat-JNI*, *verbose*, *show-all*]
oops — Countermodel by Nitpick
lemma $[(\boldsymbol{\lambda} y.\ (p \vee q)) = (\boldsymbol{\lambda} y.\ (q \vee p))\ in\ v]$
unfolding *identity-*Π_1*-def*
apply (*rule* $Eq_1 I$) **apply** (*simp add*: *meta-defs*)
nitpick[*expect* = *genuine*, *user-axioms*=*true*,
sat-solver = *MiniSat-JNI*, *card i* = *2*,
card j = *2*, *card* σ = *1*, *card* ω = *1*,
card υ = *2*, *verbose*, *show-all*]
oops — Countermodel by Nitpick

A.12.3. Concreteness coindices with Object Domains

lemma *OrdCheck*:
$[(\!|\boldsymbol{\lambda}\ x\ .\ \neg\Box(\neg(\!|E!, x^P|\!)), x|\!)\ in\ v] \longleftrightarrow$
(*proper x*) $\wedge$ (*case* (*rep x*) *of* $\omega\nu$ *y* $\Rightarrow$ *True* | - $\Rightarrow$ *False*)
using *OrdinaryObjectsPossiblyConcreteAxiom*
apply (*simp add*: *meta-defs meta-aux split*: ν.*split* υ.*split*)
using $\nu\upsilon$-$\omega\nu$-*is*-$\omega\upsilon$ **by** *fastforce*
lemma *AbsCheck*:
$[(\!|\boldsymbol{\lambda}\ x\ .\ \Box(\neg(\!|E!, x^P|\!)), x|\!)\ in\ v] \longleftrightarrow$
(*proper x*) $\wedge$ (*case* (*rep x*) *of* $\alpha\nu$ *y* $\Rightarrow$ *True* | - $\Rightarrow$ *False*)
using *OrdinaryObjectsPossiblyConcreteAxiom*
apply (*simp add*: *meta-defs meta-aux split*: ν.*split* υ.*split*)
using *no-*$\alpha\omega$ **by** *blast*

A.12.4. Justification for Meta-Logical Axioms

Remark A.28. *OrdinaryObjectsPossiblyConcreteAxiom is equivalent to "all ordinary objects are possibly concrete".*

lemma *OrdAxiomCheck*:
OrdinaryObjectsPossiblyConcrete $\longleftrightarrow$
($\forall$ *x*. ($[(\!|\boldsymbol{\lambda}\ x\ .\ \neg\Box(\neg(\!|E!, x^P|\!)), x^P|\!)\ in\ v]$
$\longleftrightarrow$ (*case x of* $\omega\nu$ *y* $\Rightarrow$ *True* | - $\Rightarrow$ *False*)))
unfolding *Concrete-def*
apply (*simp add*: *meta-defs meta-aux split*: ν.*split* υ.*split*)
using $\nu\upsilon$-$\omega\nu$-*is*-$\omega\upsilon$ **by** *fastforce*

Remark A.29. *OrdinaryObjectsPossiblyConcreteAxiom is equivalent to "all abstract objects are necessarily not concrete".*

lemma *AbsAxiomCheck*:
OrdinaryObjectsPossiblyConcrete $\longleftrightarrow$
($\forall$ *x*. ([(|$\boldsymbol{\lambda}$ *x* . $\Box(\neg(|E!, x^P|)), x^P|)$ *in v*]
$\longleftrightarrow$ (*case x of* $\alpha\nu$ *y* $\Rightarrow$ *True* | - $\Rightarrow$ *False*)))
apply (*simp add*: *meta-defs meta-aux split*: ν.*split* υ.*split*)
using $\nu\upsilon$-$\omega\nu$-*is*-$\omega\upsilon$ *no*-$\alpha\omega$ **by** *fastforce*

Remark A.30. *PossiblyContingentObjectExistsAxiom is equivalent to the corresponding statement in the embedded logic.*

lemma *PossiblyContingentObjectExistsCheck*:
PossiblyContingentObjectExists $\longleftrightarrow$ [$\neg(\Box(\forall$ *x*. $(|E!,x^P|) \rightarrow \Box(|E!,x^P|)))$ *in v*]
apply (*simp add*: *meta-defs forall*-ν-*def meta-aux split*: ν.*split* υ.*split*)
by (*metis* ν.*simps*(*5*) $\nu\upsilon$-*def* υ.*simps*(*1*) *no*-$\sigma\omega$ υ.*exhaust*)

Remark A.31. *PossiblyNoContingentObjectExistsAxiom is equivalent to the corresponding statement in the embedded logic.*

lemma *PossiblyNoContingentObjectExistsCheck*:
PossiblyNoContingentObjectExists $\longleftrightarrow$ [$\neg(\Box(\neg(\forall$ *x*. $(|E!,x^P|) \rightarrow \Box(|E!,x^P|))))$ *in v*]
apply (*simp add*: *meta-defs forall*-ν-*def meta-aux split*: ν.*split* υ.*split*)
using $\nu\upsilon$-$\omega\nu$-*is*-$\omega\upsilon$ **by** *blast*

A.12.5. Relations in the Meta-Logic

Remark A.32. *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* . $\Box((|F,x^P|) \equiv (|G,x^P|))$ *in v*] $\longleftrightarrow$
$((\lambda\ x\ .\ (eval\Pi_1\ F)\ x\ dj) = (\lambda\ x\ .\ (eval\Pi_1\ G)\ x\ dj))$
proof
assume *1*: [$\forall$ *x*. $\Box((|F,x^P|) \equiv (|G,x^P|))$ *in v*]
{
fix *v*
fix *y*
obtain *x* **where** *y-def*: $y = \nu\upsilon\ x$
by (*meson* $\nu\upsilon$-*surj surj-def*)
have ($\exists\, r\ o_1$. *Some* $r = d_1\ F \wedge$ *Some* $o_1 = d_\kappa\ (x^P) \wedge o_1 \in$ *ex1 r v*) =
($\exists\, r\ o_1$. *Some* $r = d_1\ G \wedge$ *Some* $o_1 = d_\kappa\ (x^P) \wedge o_1 \in$ *ex1 r v*)
using *1* **apply** − **by** *meta-solver*
moreover obtain *r* **where** *r-def*: *Some* $r = d_1\ F$
unfolding d_1-*def* **by** *auto*
moreover obtain *s* **where** *s-def*: *Some* $s = d_1\ G$
unfolding d_1-*def* **by** *auto*
moreover have *Some* $x = d_\kappa\ (x^P)$
using d_κ-*proper* **by** *simp*
ultimately have $(x \in$ *ex1 r v*$) = (x \in$ *ex1 s v*$)$
by (*metis option.inject*)
hence $(eval\Pi_1\ F)\ y\ dj\ v = (eval\Pi_1\ G)\ y\ dj\ v$
using *r-def s-def y-def* **by** (*simp add*: d_1.*rep-eq ex1-def*)
}
thus $(\lambda x.\ eval\Pi_1\ F\ x\ dj) = (\lambda x.\ eval\Pi_1\ G\ x\ dj)$
by *auto*
next
assume *1*: $(\lambda x.\ eval\Pi_1\ F\ x\ dj) = (\lambda x.\ eval\Pi_1\ G\ x\ dj)$

```
  {
    fix y v
    obtain x where x-def: x = νυ 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 νυ-surj x-def)
    hence [(|F,y^P|) ≡ (|G,y^P|) in v]
      apply − apply meta-solver
      using r-def s-def by (metis Semantics.dκ-proper option.inject)
  }
  thus [∀ x. □((|F,x^P|) ≡ (|G,x^P|)) in v]
    using T6 T8 by fast
qed
```

Remark A.33. *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,x^P|) ≡ (|G,x^P|)) 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 A.34. *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,x^P|) ≡ (|G,x^P|)) 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-meta*:
$[(\!|(\boldsymbol{\lambda}\ x\ .\ \varphi\ x),x^P|\!)\ in\ v] = (\exists\ y\ .\ \nu\upsilon\ y = \nu\upsilon\ x \wedge evalo\ (\varphi\ y)\ dj\ v)$
unfolding *meta-defs* $\nu\upsilon$*-def* **apply** *transfer* **using** $\nu\upsilon$*-def* **by** *auto*

lemma *lambda-interpret-1*:
assumes $[a = b\ in\ v]$
shows $(\boldsymbol{\lambda}x.\ (\!|R,x^P,a|\!)) = (\boldsymbol{\lambda}x\ .\ (\!|R,x^P,\ b|\!))$
proof −
have $a = b$
using *MetaSolver.Eq*κ*S Semantics.*d_κ*-inject assms identity-*κ*-def* **by** *auto*
thus *?thesis* **by** *simp*
qed

lemma *lambda-interpret-2*:
assumes $[a = (\boldsymbol{\iota}y.\ (\!|G,y^P|\!))\ in\ v]$
shows $(\boldsymbol{\lambda}x.\ (\!|R,x^P,a|\!)) = (\boldsymbol{\lambda}x\ .\ (\!|R,x^P,\ \boldsymbol{\iota}y.\ (\!|G,y^P|\!)|\!))$
proof −
have $a = (\boldsymbol{\iota}y.\ (\!|G,y^P|\!))$
using *MetaSolver.Eq*κ*S Semantics.*d_κ*-inject assms identity-*κ*-def* **by** *auto*
thus *?thesis* **by** *simp*
qed

end

Bibliography

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

[2] 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.

[3] G. Rosen. Abstract objects. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2017 edition, 2017.

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

[5] E. N. Zalta. The theory of abstract objects. http://mally.stanford.edu/theory.html. Accessed: April 04, 2017.